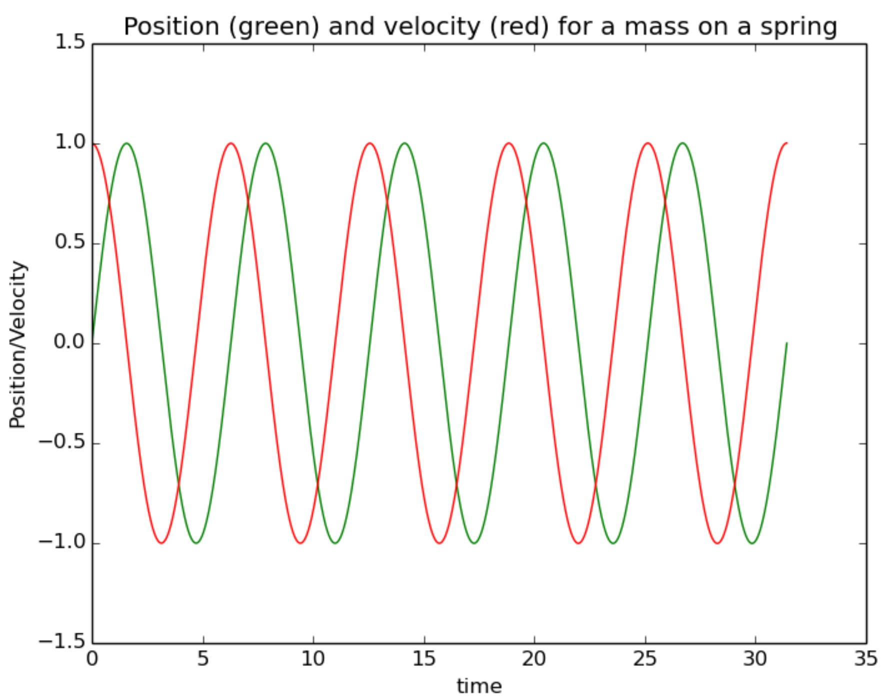
Ph 20 Set 3

Jacob Abrahams
Section 2

4/23/15

Problem 1.1

All my code will be at the end. Here's my plot using $x_0 = 0$ and $v_0 = 1$, with $h = .0001$



Problem 1.2

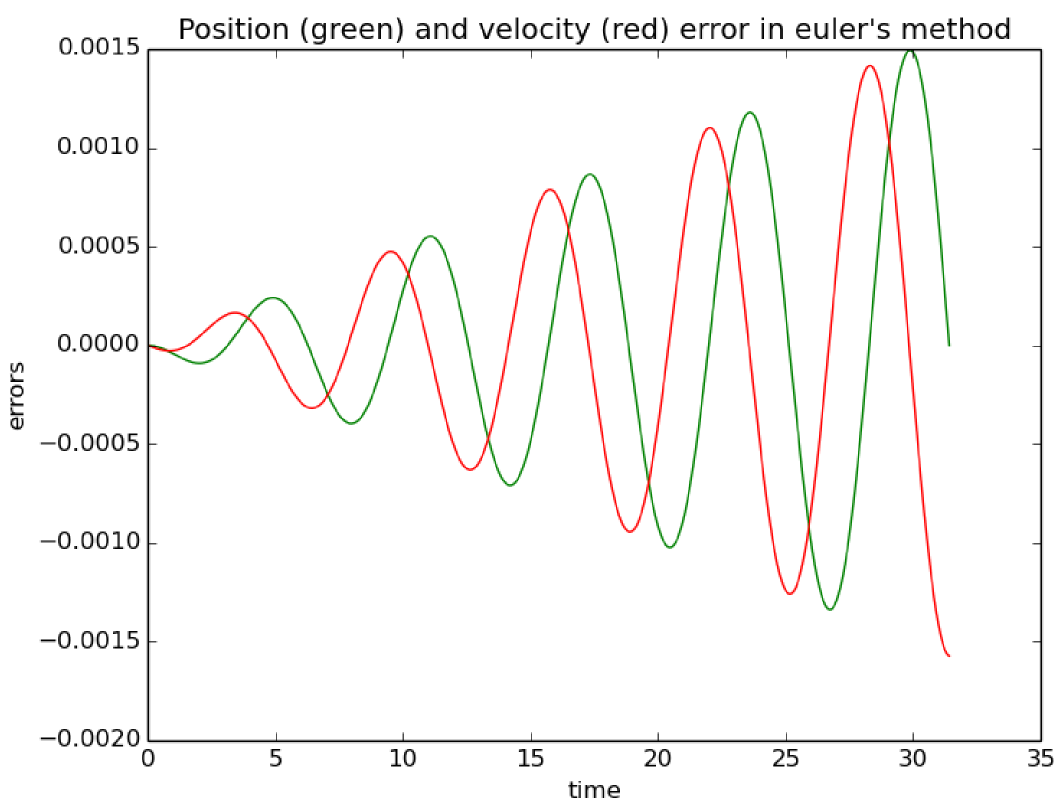
With those initial conditions, the equations for x and v are

$$x = A\sin(\omega t). \quad v = B\cos(\omega t)$$

Our initial conditions obviously give $B = 1$ and this derivatiee means $A = 1$. I'm honestly not entirely sure why, but implementing this without any constants out front lead (empirically) to $\omega = 1$ so it's actually just

$$x = \sin t, \quad v = \cos t$$

And we get the gorgeous figure



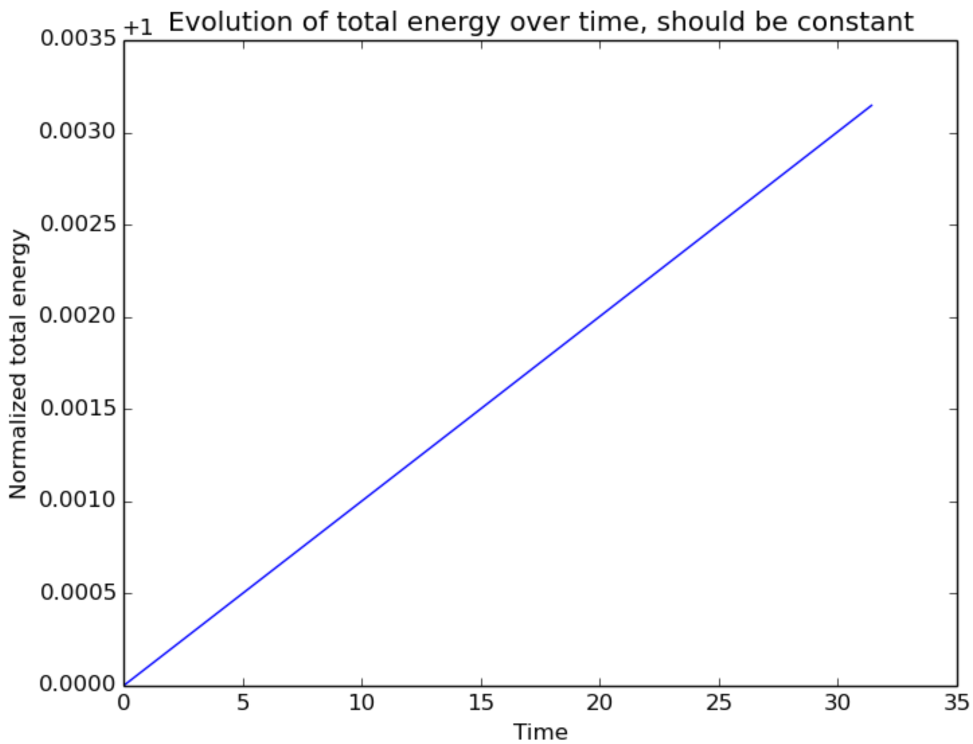
Problem 1.3

I did what was suggested, plotting $h_0, h_0/2, h_0/4, h_0/8, h_0/16$. My computer, however, couldn't finish when using $h_0 = .0001$ (making the largest step size my old one) so I switched it to $h_0 = .0016$ so that the finest step size would be the one I used above. This produced

which does in fact have really nice linear maximum errors.

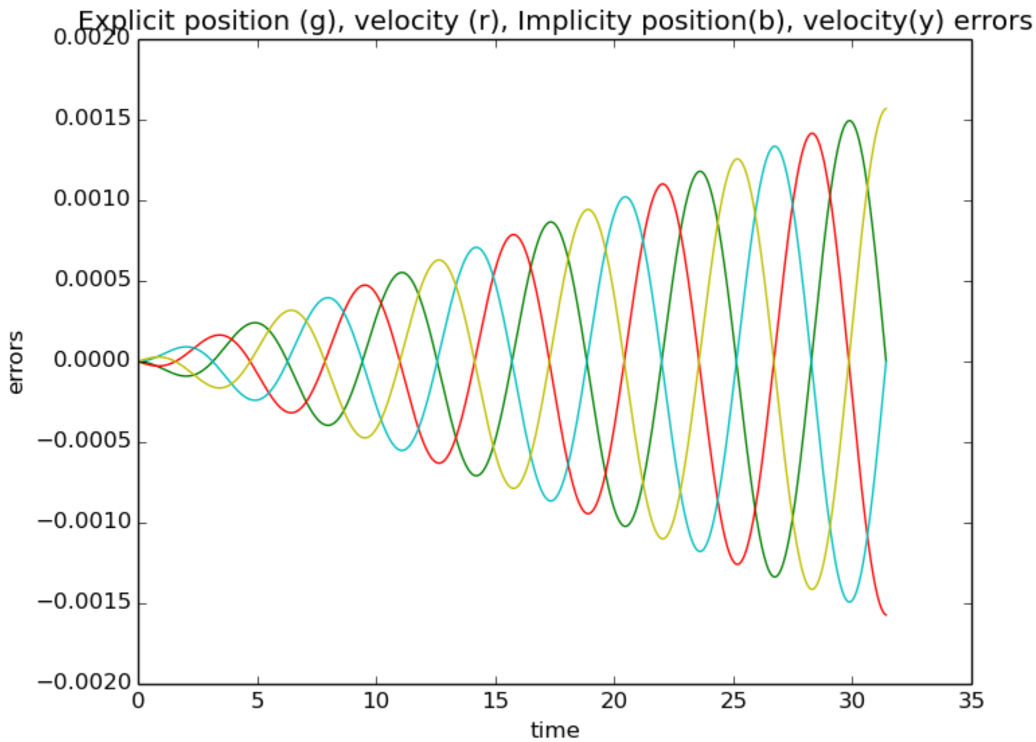
Problem 1.4

I made the plot accoridng to $E = v^2 + x^2$ and it does give what looks roughly like the envelope for the v and x errors we saw growing in problem 2, which is what we'd expect. I went back to a step size of $h = .0001$ for this. Note python labelled the y axis really annoyingly and I'm not sure (can't be bothered) to sort out how to fix it, but that +1 above is indicating that the yaxis values are actually $1.0005 \cdots 1.0035$, not the values indicated.



Problem 1.5

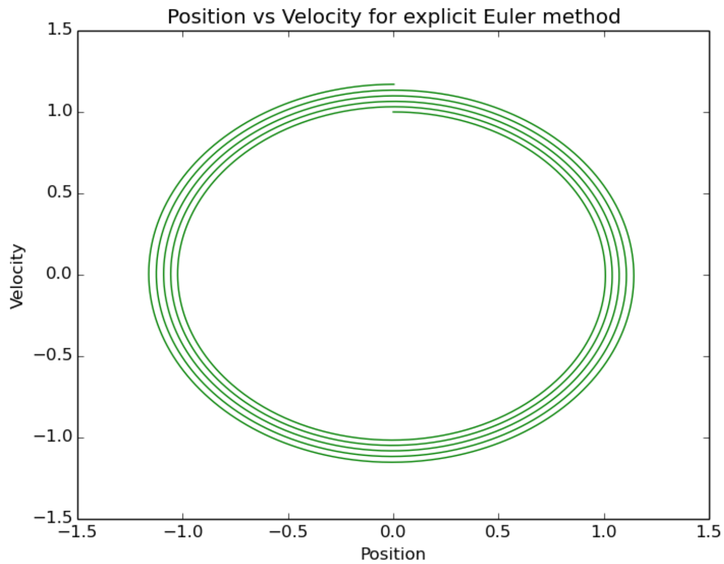
$$x_{i+1} = x_i + hv_{i+1}, \quad v_{i+1} = v_i - hx_{i+1}$$
$$\implies v_{i+1} = v_i - h\left(x_i + hv_{i+1}\right)$$
$$v_{i+1} = \frac{v_i - hx_i}{1 + h^2}$$
$$\implies x_{i+1} = \frac{hv_i + x_i}{1 + h^2}$$



They appear to have errors which grow identically. I won't plot total energy because the two plots will just be on top of eachother, and this gets the point across in a much prettier way anyway.

Problem 2.1

With $v_0 = 1$, $x_0 = 0$ and $h = .01$, these phase plots result, explicit first.

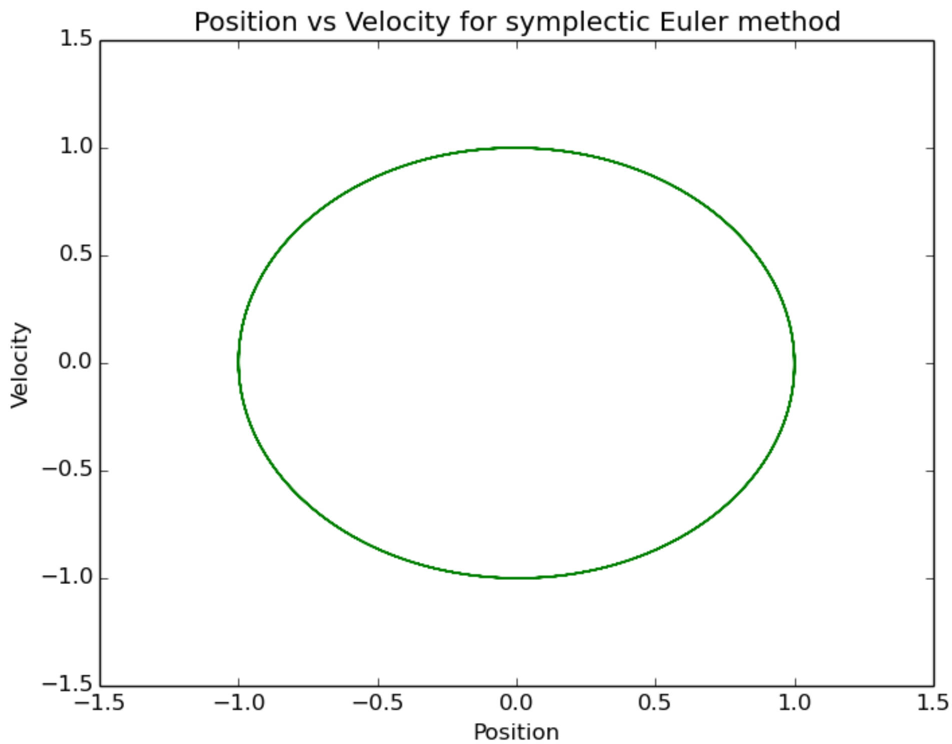


Interestingly, the explicit method spirals outward, and the implicit method spirals inward.

Problem 2.2

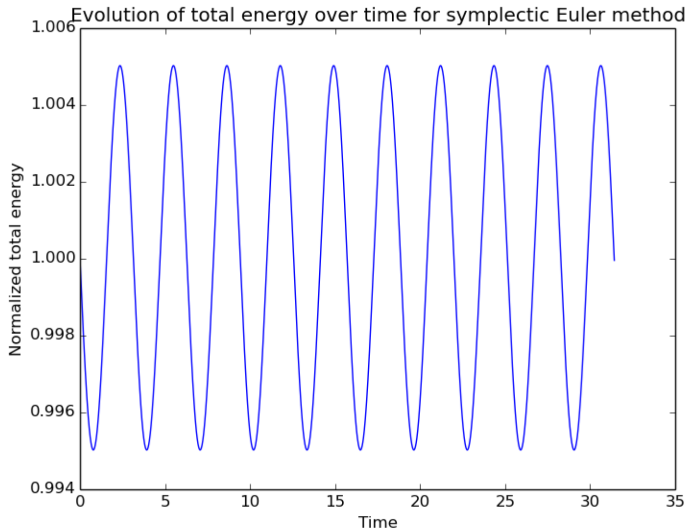
$$v_{i+1} = v_i - hx_{x+i}, \quad x_{i+1} = x_i + hv_i$$

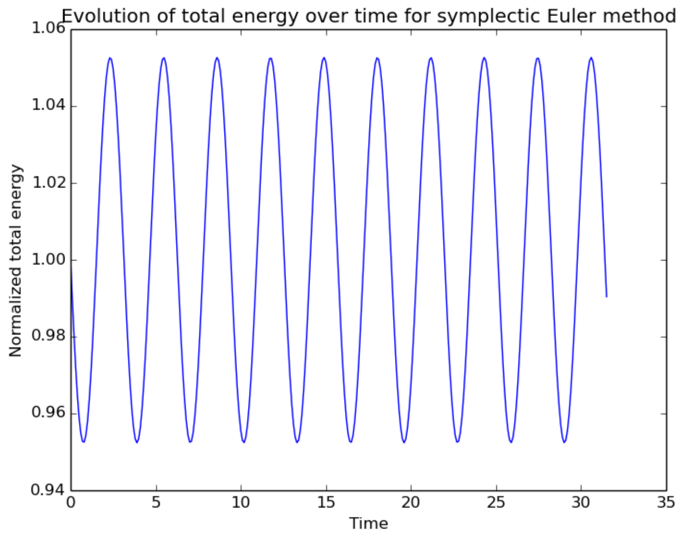
$$\implies v_{i+1} = v_i - h\left(x_i + hv_i\right) = (1 - h^2)v_i - hx_i$$



This uses the same number of cycles as the previous, so it is clear that there is no spiraling going on or, if there is, it is so small that it isn't resolved on this plot. This method, at least from an energy conservation viewpoint, is clearly better than the previous two methods.

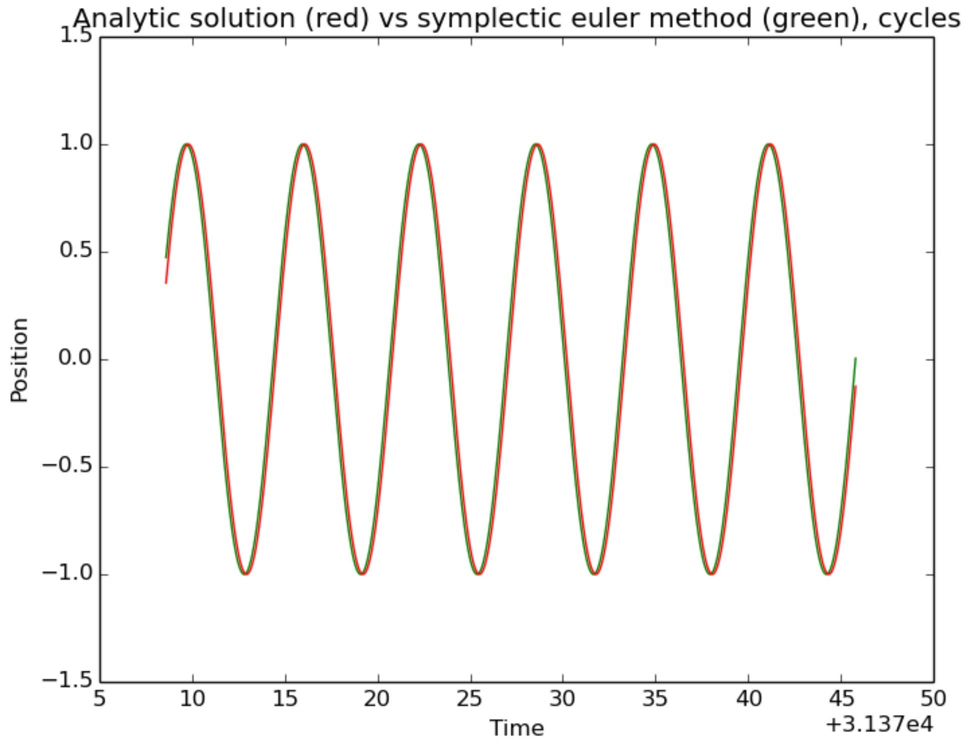
Problem 2.3





These both use $x_0 = 0,\ v_0 = 1$, but the first uses $h = .01$ and the second uses $h = .1$. It is clear that the energy is not constant, it oscillates with time, but it oscillates around what at least appears to be a constant value, so energy is... mostly... conserved.

Problem 2.4



My computer had a really bad time making this, but eventually I got it to work. This is with $h = .01$. All the previous plots I told it to compute 5 cycles, this time I told it to compute 5000 and then only display the last 5ish (rounding error, I didn't give the cutoff super carefully) cycles. Even after 1000 cycles, the offset is pretty small, but it does exist.

Code

Here's my makefile

```
Ph20Set4Latex.tex: Ph20Set3OscillatorFigure1.png
   Ph20Set3ErrorPlot.png Ph20Set3hcomparison.png
   Ph20Set3Energies.png Ph20Set3ImplicitErrors.png
   Ph20Set3ExplicitPhase.png Ph20Set3ImplicitPhase.png
   Ph20Set3SymplecticPhase.png
   Ph20Set3SymplecticEnergy1.png
   Ph20Set3SymplecticEnergy2.png
   Ph20Set3PhaseComparison.png
        pdflatex Ph20Set4Latex.tex
Ph20Set3OscillatorFigure1.png: Makefile
   Ph20Set3Oscillator.py
        python Ph20Set3Oscillator.py
           Ph20Set3OscillatorFigure1.png 0 1 .0001 5
           plot
Ph20Set3ErrorPlot.png: Makefile Ph20Set3Oscillator.py
        python Ph20Set3Oscillator.py Ph20Set3ErrorPlot.
           png 0 1 .0001 5 errors
Ph20Set3hcomparison.png: Makefile Ph20Set3Oscillator.py
        python Ph20Set3Oscillator.py
           Ph20Set3hcomparison.png 0 1 .0016 5
           hcomparison
Ph20Set3Energies.png: Makefile Ph20Set3Oscillator.py
        python Ph20Set3Oscillator.py Ph20Set3Energies.
           png 0 1 .0001 5 energy
Ph20Set3ImplicitErrors.png: Makefile Ph20Set3Oscillator
   .py
        python Ph20Set3Oscillator.py
           Ph20Set3ImplicitErrors.png 0 1 .0001 5
           impliciterror
Ph20Set3ExplicitPhase.png: Makefile Ph20Set3Oscillator.
   py
        python Ph20Set3Oscillator.py
           Ph20Set3ImplicitPhase.png 0 1 .01 5 expphase
```

```
Ph20Set3ImplicitPhase.png: Makefile Ph20Set3Oscillator.
   py
        python Ph20Set3Oscillator.py
            Ph20Set3ImplicitPhase.png 0 1 .01 5 impphase
Ph20Set3SymplecticPhase.png: Makefile
   Ph20Set3Oscillator.py
        python Ph20Set3Oscillator.py
            Ph20Set3SymplecticPhase.png 0 1 .01 5
            symphase
Ph20Set3SymplecticEnergy1.png: Makefile
   Ph20Set3Oscillator.py
        python Ph20Set3Oscillator.py
            Ph20Set3SymplecticEnergy1.png 0 1 .01 5
            symenergy
Ph20Set3SymplecticEnergy2.png: Makefile
   Ph20Set3Oscillator.py
        python Ph20Set3Oscillator.py
            Ph20Set3SymplecticEnergy2.png 0 1 .1 5
            symenergy
Ph20Set3PhaseComparison.png: Makefile
   Ph20Set3Oscillator.py
        python Ph20Set3Oscillator.py
            Ph20Set3PhaseComparison.png 0 1 .01 5
            phasecheck
```

And python code

```python
import math, numpy, sys
import matplotlib.pyplot as pplot

# Exciting change
# Read input, optional last argument plots errors
   instead of values
filename = sys.argv[1]
initialx = float(sys.argv[2])
initialv = float(sys.argv[3])
step = float(sys.argv[4])
cycles = int(sys.argv[5])
plottype = sys.argv[6]
```

```
def nextx(oldx, oldv, h):
        return oldx + h*oldv

def nextv(oldx, oldv, h):
        return oldv - h*oldx

def oscillate(startx, startv, h, laps):
        counter = 0
        position = 0
        x = [startx]
        v = [startv]
        time = [0]
        while(counter < laps):
                x.append(nextx(x[position],v[position],
                   h))
                v.append(nextv(x[position],v[position],
                   h))
                position += 1
                time.append(position*h)
                if(x[position] >= startx and x[position
                   -1] < startx):
                        counter += 1
        return [x,v,time]

def computeimplicites(startx, startv, h, laps):
        counter = 0
        p = 0
        x = [startx]
        v = [startv]
        time = [0]
        while(counter < laps):
                v.append((v[p]-h*x[p])/(1+h*h))
                x.append((h*v[p] + x[p])/(1+h*h))
                p += 1
                time.append(p*h)
                if(x[p] >= startx and x[p-1] < startx):
                        counter += 1
```

```
        return [x,v,time]

def symplecticeuler(startx, startv, h, laps):
        counter = 0
        p = 0
        x = [startx]
        v = [startv]
        time = [0]
        while(counter < laps):
                v.append((1-h*h)*v[p] - h*x[p])
                x.append(x[p] + h*v[p])
                p += 1
                time.append(p*h)
                if(x[p] >= startx and x[p-1] < startx):
                        counter += 1
        return [x,v,time]

def computereals(xamp,vamp,omega,h,steps):
        x = []
        v = []
        t = []
        for i in range(steps):
                x.append(xamp*math.sin(omega*h*i))
                v.append(vamp*math.cos(omega*h*i))
                t.append(h*i)
        return [x,v,t]

if(plottype == "plot"):
        numericals = oscillate(initialx, initialv, step
            , cycles)
        pplot.plot(numericals[2],numericals[0],c='g')
        pplot.plot(numericals[2],numericals[1],c='r')
        pplot.title("Position (green) and velocity (red
            ) for a mass on a spring")
        pplot.xlabel("time")
        pplot.ylabel("Position/Velocity")
        pplot.savefig(filename)
elif(plottype == "errors"):
```

```
        numericals = oscillate(initialx, initialv, step, cycles)
        reals = computereals(1,1,1,step,len(numericals[0]))
        errors = [[],[],[]]
        errors[0] = [reals[0][i] - numericals[0][i] for i in range(len(reals[0]))]
        errors[1] = [reals[1][i] - numericals[1][i] for i in range(len(reals[1]))]
        errors[2] = reals[2]
        pplot.plot(errors[2],errors[0],c='g')
        pplot.plot(errors[2],errors[1],c='r')
        pplot.title("Position (green) and velocity (red) error in euler's method")
        pplot.xlabel("time")
        pplot.ylabel("errors")
        pplot.savefig(filename)
elif(plottype == "hcomparison"):
        thingstoplot = [[],[]]
        hs = [step, step/2, step/4, step/8, step/16]
        for i in range(len(hs)):
                numerical = oscillate(initialx, initialv, hs[i], cycles)
                real = computereals(1,1,1,hs[i],len(numerical[0]))
                error = [[],[],[]]
                error[0] = [real[0][i] - numerical[0][i] for i in range(len(real[0]))]
                error[1] = [real[1][i] - numerical[1][i] for i in range(len(real[1]))]
                thingstoplot[0].append(max(error[0]))
                thingstoplot[1].append(max(error[1]))
        pplot.plot(hs,thingstoplot[0],c='g')
        pplot.plot(hs,thingstoplot[1],c='r')
        pplot.title("Maximum error vs step size, xerror in green and verror in red")
        pplot.xlabel("Step size")
        pplot.ylabel("Maximum error")
```

```
        pplot.savefig(filename)
    elif(plottype == "energy"):
        numericals = oscillate(initialx, initialv, step
            , cycles)
        energies = [numericals[0][i]**2 + numericals
            [1][i]**2 for i in range(len(numericals[0]))
            ]
        pplot.plot(numericals[2], energies)
        pplot.title("Evolution of total energy over 
            time, should be constant")
        pplot.xlabel("Time")
        pplot.ylabel("Normalized total energy")
        pplot.savefig(filename)
    elif(plottype == "impliciterror"):
        numericals = oscillate(initialx, initialv, step
            , cycles)
        implicites = computeimplicites(initialx,
            initialv,step,cycles)
        reals = computereals(1,1,1,step,len(numericals
            [0]))
        impreals = computereals(1,1,1,step,len(
            implicites[0]))
        errors = [[],[],[],[]]
        errors[0] = [reals[0][i] - numericals[0][i] for
             i in range(len(reals[0]))]
        errors[1] = [reals[1][i] - numericals[1][i] for
             i in range(len(reals[1]))]
        errors[2] = [impreals[0][i] - implicites[0][i]
            for i in range(len(impreals[0]))]
        errors[3] = [impreals[1][i] - implicites[1][i]
            for i in range(len(impreals[1]))]
        pplot.plot(reals[2],errors[0],c='g')
        pplot.plot(reals[2],errors[1],c='r')
        pplot.plot(impreals[2],errors[2],c='c')
        pplot.plot(impreals[2],errors[3],c='y')
        pplot.title("Explicit position (g), velocity (r
            ), Implicity position(b), velocity(y) errors
            ")
```

```
        pplot.xlabel("time")
        pplot.ylabel("errors")
#       pplot.axis([0,35,-.002,.002])
        pplot.savefig(filename)
    elif(plottype == "expphase"):
        numericals = oscillate(initialx, initialv, step
            , cycles)
        pplot.plot(numericals[0],numericals[1],c='g')
        pplot.title("Position vs Velocity for explicit 
            Euler method")
        pplot.xlabel("Position")
        pplot.ylabel("Velocity")
        pplot.savefig(filename)
    elif(plottype == "impphase"):
        implicites = computeimplicites(initialx,
            initialv, step, cycles)
        pplot.plot(implicites[0],implicites[1],c='g')
        pplot.title("Position vs Velocity for implicit 
            Euler method")
        pplot.xlabel("Position")
        pplot.ylabel("Velocity")
        pplot.savefig(filename)
    elif(plottype == "symphase"):
        positions = symplecticeuler(initialx, initialv,
             step, cycles)
        pplot.plot(positions[0],positions[1],c='g')
        pplot.title("Position vs Velocity for 
            symplectic Euler method")
        pplot.xlabel("Position")
        pplot.ylabel("Velocity")
        pplot.savefig(filename)
    elif(plottype == "symenergy"):
        positions = symplecticeuler(initialx, initialv,
             step, cycles)
        energies = [positions[0][i]**2 + positions[1][i
            ]**2 for i in range(len(positions[0]))]
        pplot.plot(positions[2], energies)
```

```
        pplot.title("Evolution of total energy over 
           time for symplectic Euler method")
        pplot.xlabel("Time")
        pplot.ylabel("Normalized total energy")
        pplot.savefig(filename)
elif(plottype == "phasecheck"):
        symplects = symplecticeuler(initialx, initialv,
           step,1000*cycles)
        reals = computereals(1,1,1,step,len(symplects
           [0]))
        length = len(symplects[0])
        start = int(length/1000)*999
        pplot.plot(symplects[2][start:],symplects[0][
           start:],c='g')
        pplot.plot(reals[2][start:],reals[0][start:],c=
           'r')
        pplot.title("Analytic solution (red) vs 
           symplectic euler method (green), cycles")
        pplot.xlabel("Time")
        pplot.ylabel("Position")
        pplot.savefig(filename)
else:
        print "6th command must be the type of plot, '
           plot', 'errors', 'energy', 'implicites', '
           expphase', 'symenergy', 'symphase', '
           phasecheck', or 'hcomparison'"
```