# Phys 20 Lab 3 - Numerical Solutions to ODEs: Spring Problem

Chris Dudiak

May 7, 2013

## 1 Explicit Euler

```
def springProp(z, h):
    # loop over the lists simultaneously and compute new values
    for i, (x, v) in enumerate(z):
        if (i + 1 >= N):
            break
        z[i + 1] = (x + h * v, v - h * x)
    return z
```

This function sets the next values based on the old values. Using this function, the position and velocity are graphed vs time. For this set:

$$x_0 = 5.0$$
$$v_0 = 0.0$$
$$h = .01$$

Unless otherwise stated. Plotting the propagation of the spring for 2000 steps, we get:

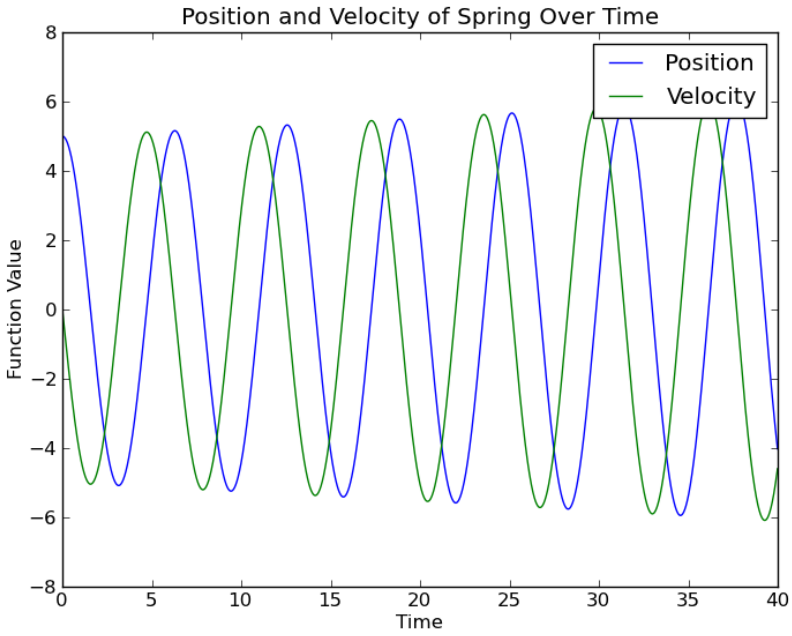


Figure 1: Explicit Evaluation of Euler's Method on Spring

## 2 Analytical Solution

For the analytic solution of $a = \frac{d^2x}{dt^2} = -x$ with the initial conditions above and $\frac{k}{m} = 1$:

$$x(t) = 5.0\cos(t)$$
$$v(t) = -5.0\sin(t)$$

Plotting the errors we find:

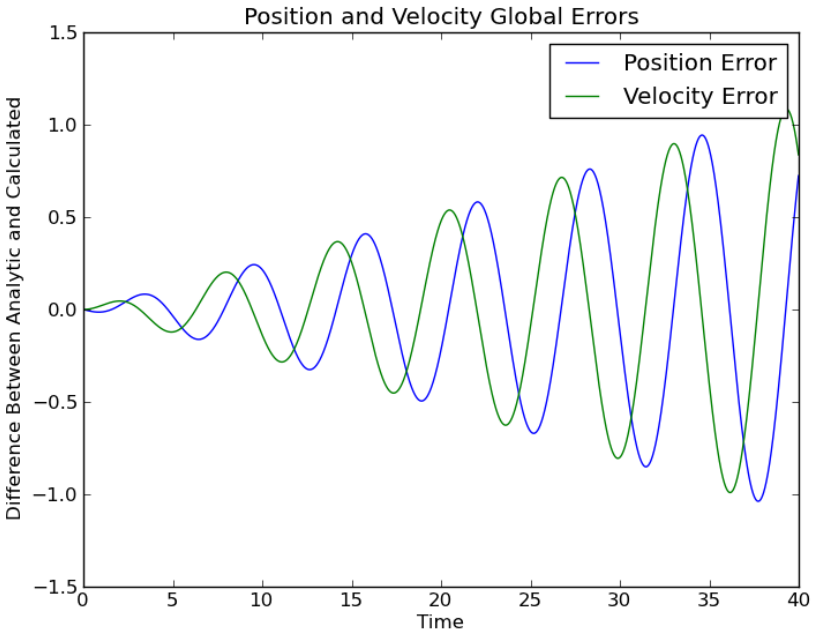


Figure 2: Explicit Evaluation of Euler's Method Errors from Analytic Solution

As time increase, the maximum magnitude of the error in both position and velocity gets larger; this increase in maximum peaks is linear with time.

# 3 Truncation Errors

Using h = .0027, we see the following truncation errors:

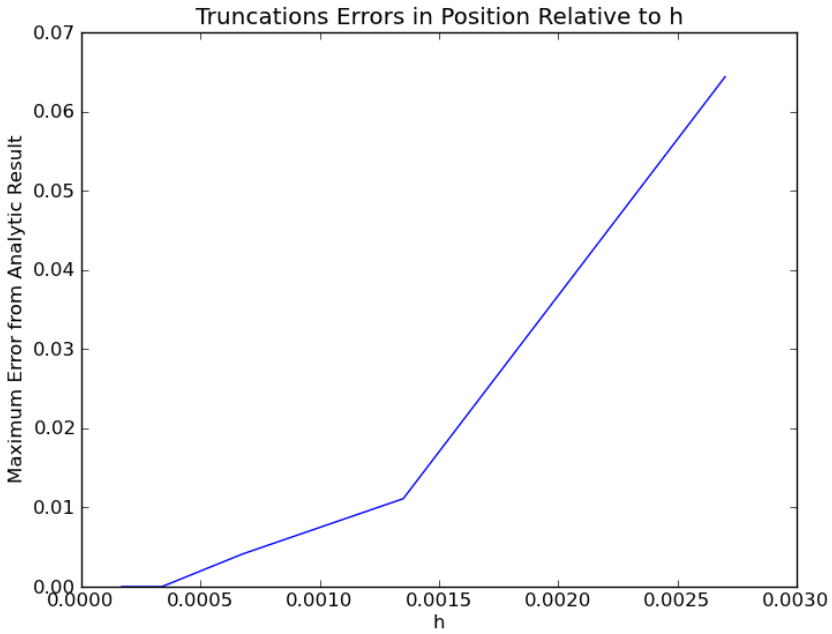


Figure 3: Explicit Evaluation of Euler's Method Truncation Errors on h

As the value of h increases, the maximum error from the analytic solution increases as well. At extremely small h's, the error is almost 0 as far as the floating point values can detect. The error grows quadratically as h increases though.

# 4 Energy

Looking at Total Energy $E = v^2 + x^2$ as a function of time:

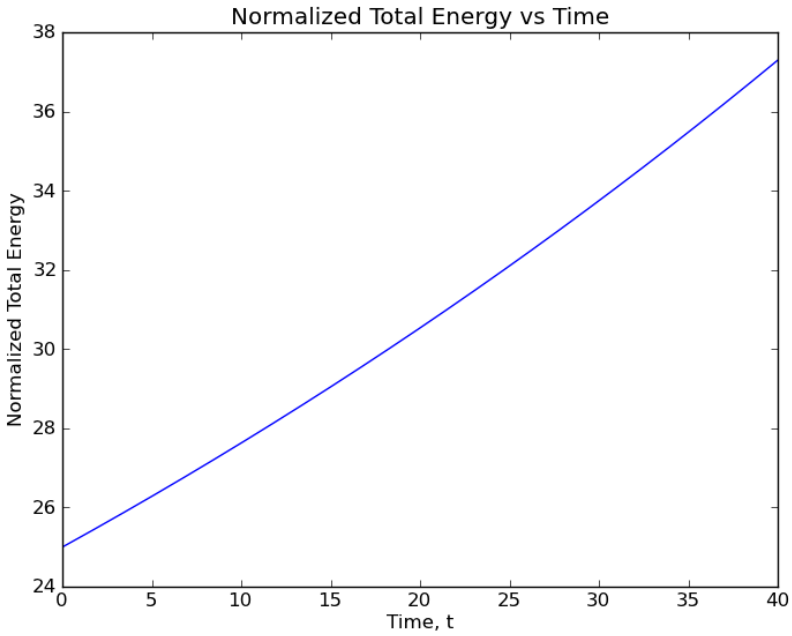


Figure 4: Explicit Evaluation of Euler's Method Energy

We see that the energy is increasing quadratically over time due to the squares in the position and velocities, which grow linearly. Eventually, this energy would go to infinity rather than remain constant due to the errors in the floating point precision. At small values of h, the energy could remain near 25 as expected, but the error would probably still propagate forward over time and build on itself.

# 5 Implicit Euler

Solving the implicit solution, we can find:

$$\begin{pmatrix} x_{i+1} \\ v_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -h \\ h & 1 \end{pmatrix}^{-1} \begin{pmatrix} x_i \\ v_i \end{pmatrix}$$

Using this, we can calculate the global errors and energy as before:

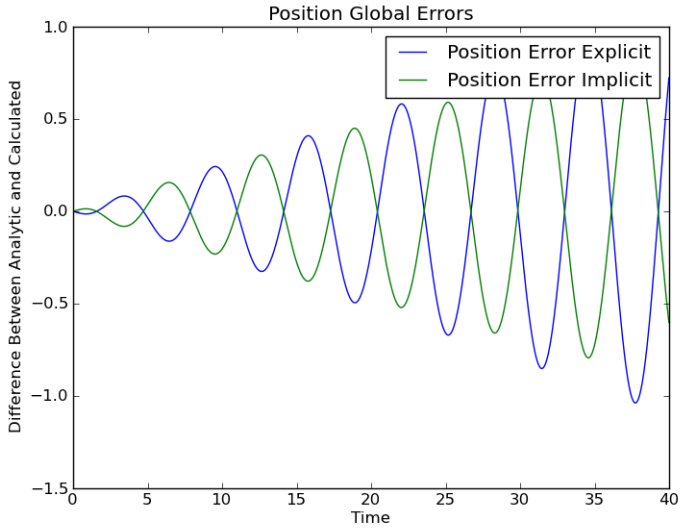


Figure 5: Implicit and Explicit Euler's Method Position Errors

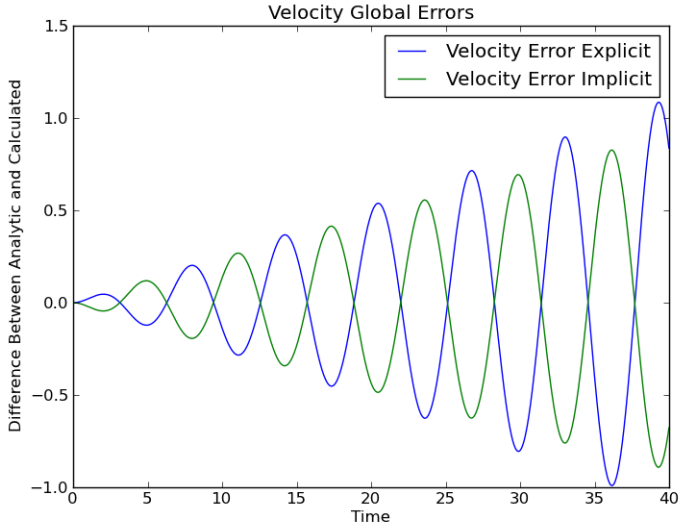


Figure 6: Implicit and Explicit Euler's Method Velocity Errors

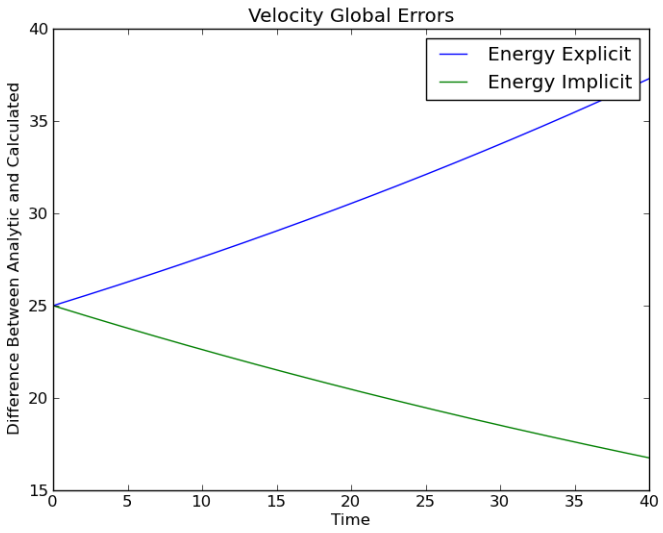


Figure 7: Implicit and Explicit Euler's Method Energy

As we can see, the implicit errors are of the same form but mirror the explicit errors given that they are calculated from new values rather than old values. They evolve similarly although the peaks for the explicit errors are increasing faster as they are unbounded. The energies however go in opposite directions; the implicit energy tends to zero rather than positive infinity for the same reason. The energy is expressed as $v^2 + x^2$ so the implicit energy decays to zero but the explicit blows up indefinitely to infinity.

# 6 Conservation of Energy

## 6.1 Implicit and Explicit Phase-Space

Since the energy is not conserved in the implicit and explicit cases, they will not form complete circles in phase space. The explicit form, which tends to infinite energy, spirals outward, and the implicit solution, which tends to zero spirals inward.

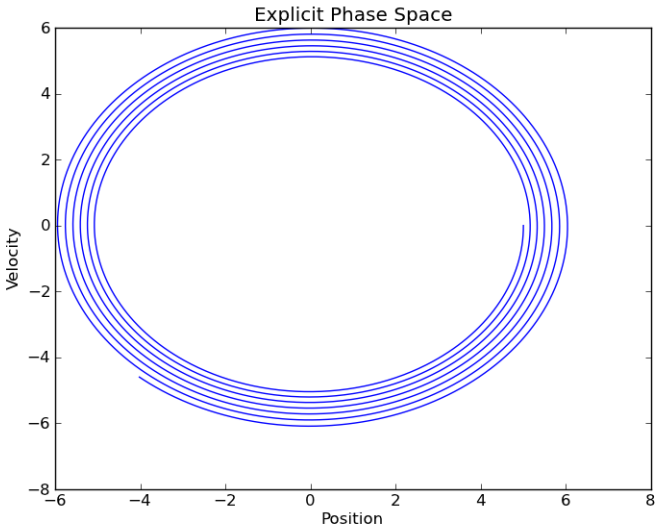


Figure 8: Explicit Phase $v_0 = 0.0, d_0 = 5.0$: Spirals Out

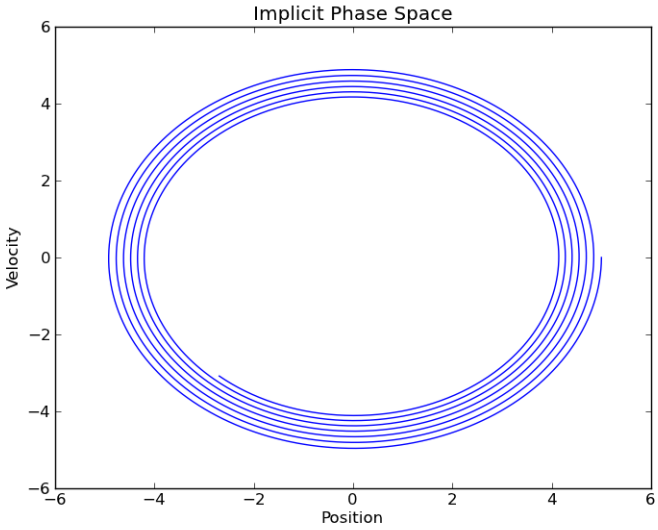


Figure 9: Implicit Phase $v_0 = 0.0, d_0 = 5.0$: Spirals In

## 6.2 Symplectic Euler

The Symplectic Euler method uses:

$$x_{i+1} = x_i + hv_i$$
$$v_{i+1} = v_i - hx_{i+1}$$

This is generated by the following code:

```
def springSymp(z, h):
    # loop over the lists simultaneously and compute new values
    for i, (x, v) in enumerate(z):
        if (i + 1 >= N):
            break
        x_next = x + h * v
        z[i + 1] = (x_next, v - h * x_next)
    return z
```

Since energy is now conserved much better, this creates circles in phase space rather than spirals:

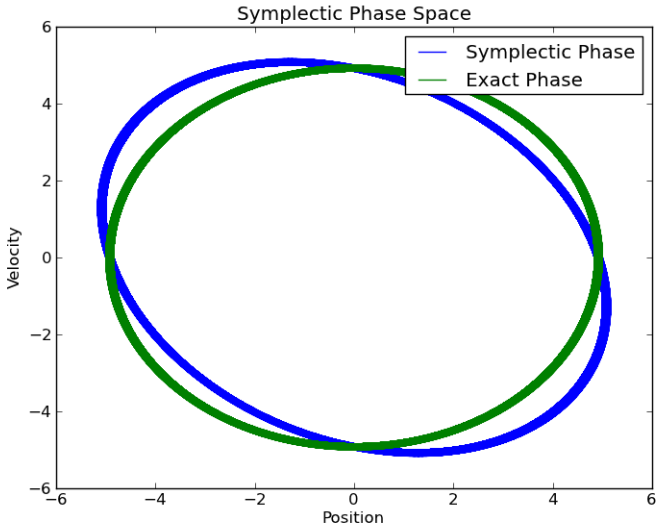


Figure 10: Symplectic Phase $v_0 = 0.0, d_0 = 5.0$, h = .5: Energy Conserved

We see that the two curves agree at the initial conditions but that the symplectic phase is slightly askew from the exact result. At points it has lower energy and at points higher energy. If we varied the initial conditions, we would expect both to still agree at those points but the oscillations could vary in magnitude based on the initial conditions and expected total energy.

## 6.3 Symplectic Energy

Looking at the energy evolution of the Symplectic Euler, we find:

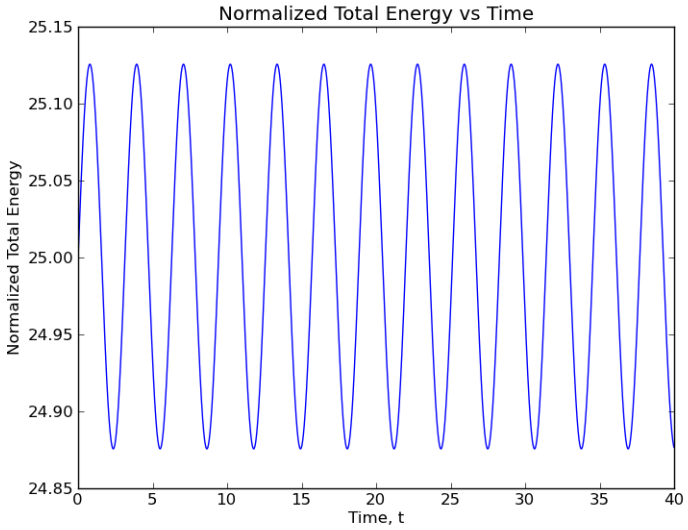


Figure 11: Symplectic Energy $v_0 = 0.0, d_0 = 5.0$: Energy Oscillates

Rather than remaining perfectly constant, the energy oscillates sinusoidally. Unlike the implicit and explicit methods, it does not go to infinity over time. The maximum energy deviation is about .05% of the initial total energy, which is a small variation.

# 7 Code and Info

## 7.1 Output

The program reports the following:

```
Using default initial conditions: x0 = 5, v0 = 0
Successfully Ran Script
```

## 7.2 Code

Code for this week's set:

```
#!/usr/bin/env python

# Chris Dudiak
# This program investigates numerically the motion of a spring.

import math
import numpy as np
```

```
import operator as op
import sys
import matplotlib.pyplot as plt

# global constants
N = 4000

# explicit Euler method
def spring(show, x0, v0):
    h = .01
    # create a list of the two lists zipped together
    z = zip([x0 for i in range(N)], [v0 for i in range(N)])
    z0 = z[:]

    springProp(z, h)

    # unzip the lists back to x, v
    (xs, vs) = zip(*z)
    # plot the values
    fig = plt.figure()
    ts = np.arange(0, N) * h
    plt.plot(ts, xs, label="Position")
    plt.plot(ts, vs, label="Velocity")
    title = "Position and Velocity of Spring Over Time"
    plt.title(title)
    plt.xlabel("Time")
    plt.ylabel("Function Value")
    plt.legend()
    if show:
        plt.show()
    fig.savefig("explicit.png")

    # analytic
    xE = 5.0 * np.cos(ts)
    vE = -5.0 * np.sin(ts)
    # plot errors
    fig = plt.figure()
    plt.plot(ts, xE - xs, label="Position Error")
    plt.plot(ts, vE - vs, label="Velocity Error")
    title = "Position and Velocity Global Errors"
    plt.title(title)
    plt.xlabel("Time")
    plt.ylabel("Difference Between Analytic and Calculated")
    plt.legend()
    if show:
        plt.show()
```

```
fig.savefig("errors.png")

h2 = .0027

# truncation error
hs = [h2 * (0.5 ** i) for i in range(5)]
errors = []
for h in hs:
    xsForH = map(lambda x: x[0], springProp(z0[:], h))
    tvals = np.arange(0, N) * h
    xEtest = 5.0 * np.cos(tvals)
    errors.append(max(xEtest - xsForH))
# plot errors
fig = plt.figure()
plt.plot(hs, errors)
title = "Truncations Errors in Position Relative to h"
plt.title(title)
plt.xlabel("h")
plt.ylabel("Maximum Error from Analytic Result")
if show:
    plt.show()
fig.savefig("trunc.png")

# energy
energy = map(lambda x, v: x**2 + v**2, xs, vs)
# plot errors
fig = plt.figure()
plt.plot(ts, energy)
title = "Normalized Total Energy vs Time"
plt.title(title)
plt.xlabel("Time, t")
plt.ylabel("Normalized Total Energy")
if show:
    plt.show()
fig.savefig("energy.png")

# implicit
val = np.matrix([x0, v0]).transpose()
h = .01
mat = np.linalg.inv(np.matrix([[1, -h],[h, 1]]))
impVals = [0 for i in range(N)]
for i in range(N):
    impVals[i] = val
    val = mat * val
xsImp = map(lambda x: x[0, 0], impVals)
vsImp = map(lambda v: v[1, 0], impVals)
```

```
# plot explicit phase
fig = plt.figure()
plt.plot(xs, vs)
title = "Explicit Phase Space"
plt.title(title)
plt.xlabel("Position")
plt.ylabel("Velocity")
if show:
    plt.show()
fig.savefig("ephase.png")

# plot implicit phase
fig = plt.figure()
plt.plot(xsImp, vsImp)
title = "Implicit Phase Space"
plt.title(title)
plt.xlabel("Position")
plt.ylabel("Velocity")
if show:
    plt.show()
fig.savefig("iphase.png")

# show difference with explicit
# plot errors
fig = plt.figure()
plt.plot(ts, xE - xs, label="Position Error Explicit")
plt.plot(ts, xE - xsImp, label="Position Error Implicit")
title = "Position Global Errors"
plt.title(title)
plt.xlabel("Time")
plt.ylabel("Difference Between Analytic and Calculated")
plt.legend()
if show:
    plt.show()
fig.savefig("positionComp.png")

fig = plt.figure()
plt.plot(ts, vE - vs, label="Velocity Error Explicit")
plt.plot(ts, vE - vsImp, label="Velocity Error Implicit")
title = "Velocity Global Errors"
plt.title(title)
plt.xlabel("Time")
plt.ylabel("Difference Between Analytic and Calculated")
plt.legend()
if show:
```

```
    plt.show()
fig.savefig("velocityComp.png")

energyImp = map(lambda x, v: x**2 + v**2, xsImp, vsImp)
fig = plt.figure()
plt.plot(ts, energy, label="Energy Explicit")
plt.plot(ts, energyImp, label="Energy Implicit")
title = "Velocity Global Errors"
plt.title(title)
plt.xlabel("Time")
plt.ylabel("Difference Between Analytic and Calculated")
plt.legend()
if show:
    plt.show()
fig.savefig("energyComp.png")

# symplectic Euler
h = .5
ts = np.arange(0, N) * h
z = z0[:]
springSymp(z, h)
# unzip the lists back to x, v
(xsSym, vsSym) = zip(*z)
# analytic
xE = 5.0 * np.cos(ts)
vE = -5.0 * np.sin(ts)
# plot the values alongside analytic
fig = plt.figure()
plt.plot(xsSym, vsSym, label="Symplectic Phase")
plt.plot(xE, vE, label="Exact Phase")
title = "Symplectic Phase Space"
plt.title(title)
plt.xlabel("Position")
plt.ylabel("Velocity")
plt.legend()
if show:
    plt.show()
fig.savefig("sphase.png")

# symplectic Euler energy
h = .01
ts = np.arange(0, N) * h
z = z0[:]
springSymp(z, h)
# unzip the lists back to x, v
(xsSym, vsSym) = zip(*z)
```

```
    energySymp = map(lambda x, v: x**2 + v**2, xsSym, vsSym)
    # plot energy
    fig = plt.figure()
    plt.plot(ts, energySymp)
    title = "Normalized Total Energy vs Time"
    plt.title(title)
    plt.xlabel("Time, t")
    plt.ylabel("Normalized Total Energy")
    if show:
        plt.show()
    fig.savefig("symenergy.png")

    print "Successfully Ran Script"


# Takes a zipped list z and a step h and calculates all the new values for the s
def springProp(z, h):
    # loop over the lists simultaneously and compute new values
    for i, (x, v) in enumerate(z):
        if (i + 1 >= N):
            break
        z[i + 1] = (x + h * v, v - h * x)
    return z

# Takes a zipped list z and a step h and calculates all the new values for the s
# using the symplectic Euler method.
def springSymp(z, h):
    # loop over the lists simultaneously and compute new values
    for i, (x, v) in enumerate(z):
        if (i + 1 >= N):
            break
        x_next = x + h * v
        z[i + 1] = (x_next, v - h * x_next)
    return z


if __name__ == "__main__":
    if len(sys.argv) != 4 and len(sys.argv) != 2:
        print "usage: %s bool_to_show_plot(0 or 1) [x0 v0]" % sys.argv[0]
        sys.exit(1)
    elif len(sys.argv) == 4:
        print "Using initial conditions: x0 = %s, v0 = %s" % (sys.argv[2], sys.a
        spring(int(sys.argv[1]), float(sys.argv[2]), float(sys.argv[3]))
    else:
        print "Using default initial conditions: x0 = 5, v0 = 0"
        spring(int(sys.argv[1]), 5.0, 0.0)
```