Matthew S. Norton
PHY 112 Lab Report CL-4
Damped Oscillator: Feynman-Newton Method for
Solving First-Order Differential Equations
March 2, 2006

Abstract
In this lab, I used the Feynman-Newton method for solving first-order differential equations to simulate the motion of a damped oscillator. I did this lab to increase my knowledge of FORTRAN programming, to gain a better understanding of the motion of a damped oscillator, and to study another method for solving first-order differential equations, the Feynman-Newton method. I studied the results from the Feynman-Newton methods for a $\Delta t = 0.25$s and $\Delta t = 1.0$s and then integrate from $t = 0$ to $t = 25$ with an initial displacement of 1m and an initial velocity of 0 m/s.

Introduction
This lab was designed to relate to the class discussion of harmonic motion, especially in regard to the motion of a damped oscillator. The oscillator modeled in this experiment was a spring with a mass attached to one end. This is demonstrated in figure 1.

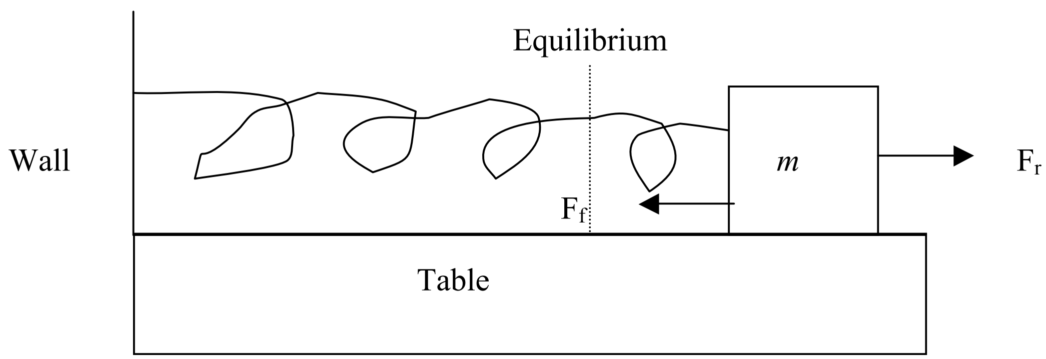

Fig. 1. An example setup for a spring as a damped oscillator

In this lab, we started with the equation for the drag force, F_{damp}, which is directed in the opposite direction of the direction of motion.

$$F_{damp} = c \cdot v, \tag{1}$$

where c is the damping coefficient and v is the velocity of the oscillating mass. Since F_{damp} is a force, it can be written in the form of Newton's second law with a mass, m, acted upon by a linear restoring force with a spring constant, k, which is.

$$F(t) = m \cdot a(t) = m \cdot \frac{d^2x}{dt^2} = -k \cdot x(t) - c \cdot v(t), \tag{2}$$

and is a second-order differential equation. Equation 2 can be rewritten as two first-order differential equation,

$$\frac{dv}{dt} = \frac{F(t)}{m}, \tag{3}$$

and

$$\frac{dx}{dt} = v(t). \tag{4}$$

One could try and use the Euler method to solve these first-order differential equations, but the results are erroneous at best. Therefore, a better way of solving first-order differential equations was needed, the Feynman-Newton method, or half step method. Using the Feynman-Newton method, the solution to a first-order differential equation,

$$\frac{df(t)}{dt} = q(t), \tag{5}$$

is obtained by using the value of the function *q(t)* halfway between *t* and *t* + *Δt*. This is shown in the following equation,

$$f(t+\Delta t) \approx f(t) + \Delta t \cdot q\left(t + \frac{\Delta t}{2}\right), \tag{6}$$

which is the algorithm for the Feynman-Newton method.

```
IMPLICIT NONE

! Declare variables
     INTEGER :: I, Icnt
     Double Precision X, V, A, dt, Xold, Vold, Xint, Vint, C1, C2, Time, TimeX, TimeV, K, C, M

     OPEN (30, FILE = "F-N.txt") ! Open the output file for Feynman-Newton method
     OPEN (40, FILE = "Euler.txt") ! Open the output file for Euler method

     Dt = .25
     Time = 0
     WRITE (30, '(A5, T15, A1, T30, A5, T45, A1, T60, A1, T75)') "TimeX", "X", "TimeV", "V", "A"
     WRITE (40, '(A5, T15, A1, T30, A5, T45, A1, T60, A1, T75)') "TimeX", "X", "TimeV", "V", "A"
     PRINT *, "Please input the mass, m"
     READ *, M
     PRINT *, "Please input the spring constant, k"
     READ *, K
     PRINT *, "Please input the damping coefficient, c"
     READ *, C
     PRINT *, "Please input the initial position"
     READ *, Xint
     PRINT *, "Please input the initial velocity"
     READ *, Vint
```

```
C1 = K / M
C2 = C / M
A = -(C1 * Xint + C2 * Vint)
Xold = Xint
Vold = Vint + Dt * A / 2.

DO I = 1, 100
X = Xold + dt * Vold
A = -(C1 * X + C2 * Vold)
V = Vold + Dt * A
TimeX = TimeX + Dt
Timev = TimeX + Dt / 2.
WRITE (30, 1000) TimeX, X, TimeV, V, A
Xold = X
Vold = V
END DO
! Reset variables
A = -(C1 * Xint + C2 * Vint)
Xold = Xint
Vold = Vint
Icnt = 0
X = 0
V = 0
A = 0
TimeX = 0
TimeV = 0
DO I = 1, 100
X = Xold + dt * Vold
V = Vold + Dt * A
A = -(C1 * X + C2 * V)
TimeX = TimeX + Dt
Timev = TimeX + Dt / 2.
WRITE (40, 1000) TimeX, X, TimeV, V, A
Xold = X
Vold = V
END DO
1000     FORMAT (5(E12.5, 3X))
CLOSE (30) ! Close the output file
CLOSE (40) ! Close the output file
PRINT *, "Output files, F-N.txt and Euler.txt have been closed, program is complete!"
PRINT *, "Have a great day!"
END PROGRAM
```

Fig. 2. Source code for a FORTRAN program for the motion of a damped oscillator

Figure 2 shows the source code for the program that I used to solve the first-order differential equations. In lines 1 and 2 of the program, I declare my variables. Then in lines 3 and 4 I open the output files. In lines 5 and 6, I initialize the change in time and the time. In lines 8 and 9, I write to the first line of the output files the column labels to be read into KaleidaGraph. Then in lines 9 through 18, I prompt the user for the mass on the end of the spring, the spring constant, the damping coefficient, the initial position, and the initial velocity. In lines 19 through 23, I initialize the initial conditions for the spring. In lines 24 through 33, I perform a DO loop that calculates the first-order differential equations using the Feynman-Newton method. In lines 34 through 42, I reinitialize the conditions to their initial state. Then in lines 43 through 52, I perform a DO loop that calculates the first-order differential equations using the Euler method. In line 53, I have the format statement for writing the data calculated the DO loops to their respective files. In lines 54 and 55, I close the output files. Then in lines 56 through 58, I tank the user for using the program and I end the program.

Results and Discussion

Figure 3 shows the position versus time for the Euler method for a Δt of 0.25s and 1.0s. For this figure, both values for Δt appear to produce similar results until time reaches about 13 seconds. Then the values for Δt = 1.0s just appears to "blow up" and seemingly gain energy, which defies the second law of thermodynamics. While the graph of Δt = 0.25s also seems to gain energy, it not as drastic at the graph of Δt = 1.0s. The graph of Δt = 0.25s appears to be more accurate that the graph of Δt = 1.0s. The graph of Δt = 0.25s is more accurate because it provides more data points and the time interval between the data points is lessened.

Figure 4 shows the position versus time for Δt = 0.25s and Δt = 1.0s, this is the same graph as in figure 3, but it is zoomed in to show the movement of the Δt = 0.25s curve. This graph shows how the Euler method, even for a relatively small time step, produces results that cannot be trusted. While, it is better than the results for Δt = 1.0s, Δt = 0.25s does not produce good results.

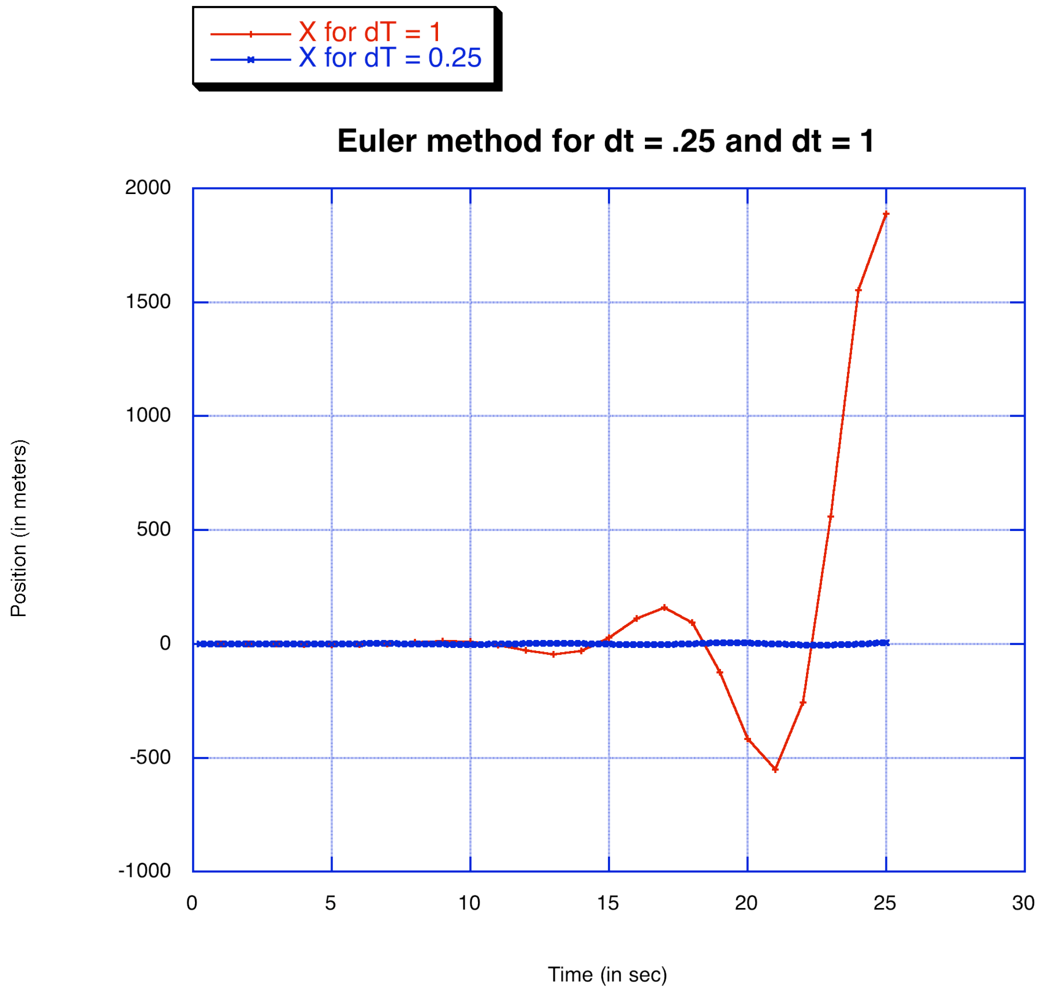

Fig. 3. Position vs. Time for a Δt of 0.25s and a Δt 1.0s for the Euler method

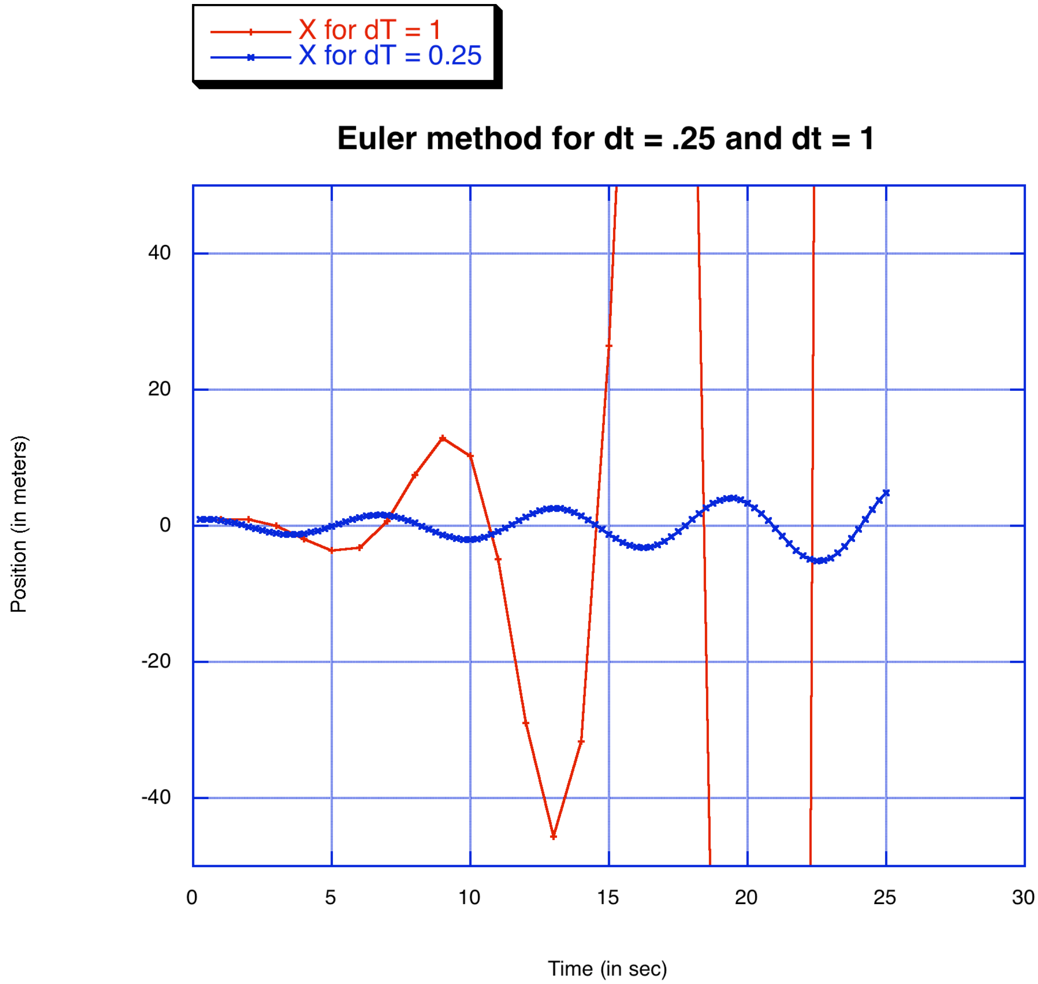

Fig. 4. Position vs. time for Δt = 0.25s and 1.0s for Euler Method, zoomed in to show Δt = 0.25s

At this point in the experiment, I switched from the Euler method to the Feynman-Newton method for solving first-order differential equations.

Figure 5 depicts the position versus time for Δt = 0.25s and 1.0s for the Feynman-Newton method. The different values for Δt produce similar results until time reached about 13 seconds. At that point, the graph of Δt = 1.0s gets out phase with the graph of Δt = 0.25s. The graph of Δt = 0.25s appears to be more accurate than the graph of Δt = 1.0s. This is because the graph of Δt = 0.25s contains more data points than the graph of Δt = 1.0s and the graph of Δt = 0.25s appears to be smoother than the graph of Δt = 1.0s. Because there was only 25 data points for the graph of Δt = 1.0s, it appears to be more jagged and rough. Therefore, the graph of Δt = 0.25s should be more accurate than the graph of Δt = 1.0s.

Figure 6 shows the plot of velocity versus acceleration.

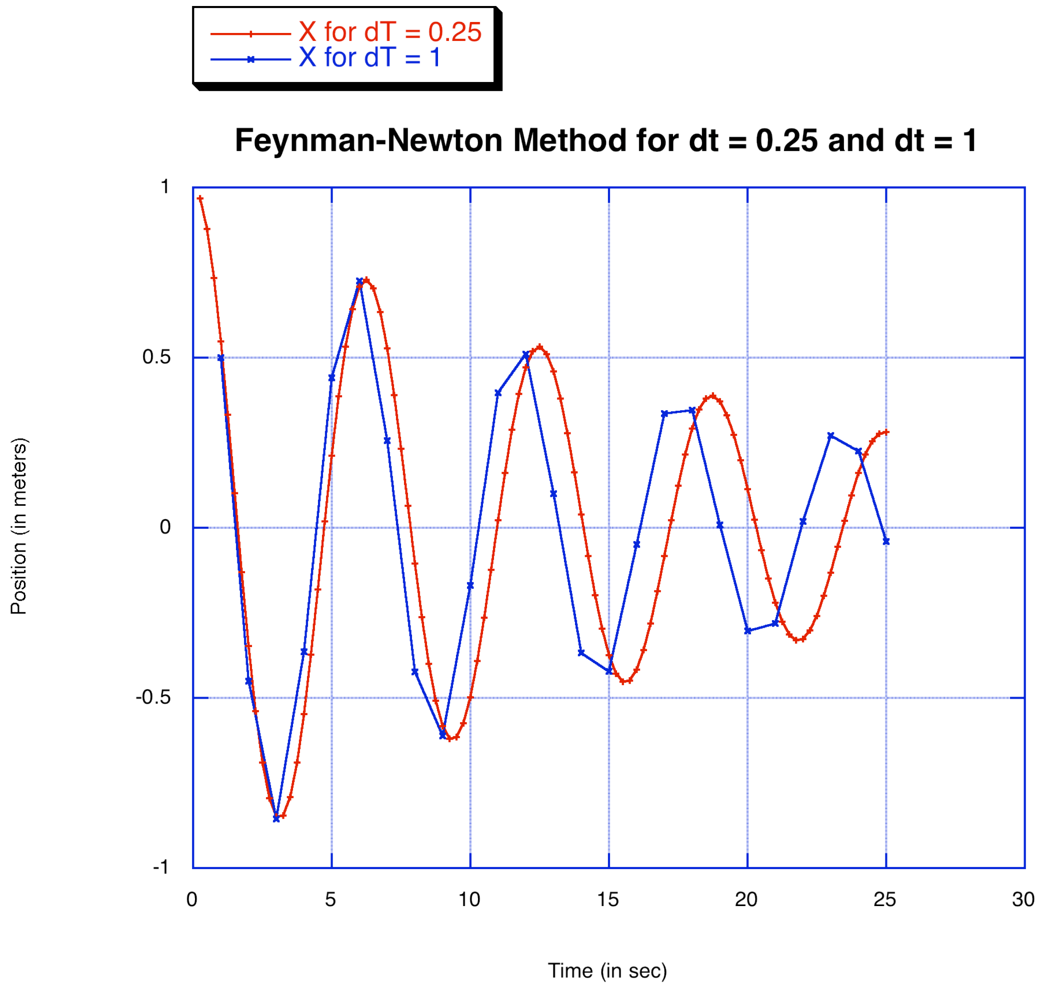

Fig. 5. Position vs. time for Δt = 0.25s and 1.0s for Feynman-Newton method