Finite Difference Simulation of a One Dimensional Harmonic Oscillator

A comparison of three finite difference techniques

This experiment concern the comparison of three finite difference methods to the computational simulation of a simple harmonic oscillator. The techniques employed here were the Euler technique, the velocity form of the Verlet algorithm, and Gear 3rd order predictor-corrector method. The experiment concluded that the Euler algorithm was far too inaccurate and unstable for serious use, and also that while the Gear method is consistently the most accurate (once a suitable time step has been found), the inherent stability of the Verlet system (irrespective of time step) makes it more suitable for general use.

Andrew N Jackson 1st February 1996

Introduction

The aim of this exercise is to use a range of finite difference methods to simulate a 1-D harmonic oscillator, and to then use these simulations to distinguish between the performances of the different algorithms. Three methods are to be implemented:

- The Euler algorithm;
- The velocity form of the Verlet algorithm;
- Gear's predictor-corrector algorithm without iteration (ie of the form PEC) using a third order predictor.

The performance of these methods is best evaluated by observing the behaviour of the phase space plot (velocity against position) for the oscillator. Accurately simulated, the phase trajectory should form a circle, and an insight into the accuracy and stability of the simulation can be gained by seeing how closely the simulated trajectory compares to the accurate circular one. We can also analyse performance by seeing how well the energy of the system is conserved over time. Using the above tests, we can find which of the methods reliably produce the correct results, and what the maximum time step (Δt) that can be used is.

Theory

Before the details of the finite difference methods are explained, it is first necessary to define the problem we are trying to solve. The simulation of a 1-D harmonic oscillator reduces to the differential equation:

$$d^2x/dt^2 \quad = \quad - \omega^2 x$$

where,

$$\omega^2 \quad = \quad k/m$$

given the Hook's law constant, k, and the particle mass, m. From this it is clear that the system requires initial conditions for the velocity and position of the particle, and for our purposes these shall be set to $x_0 = 1$ and $v_0 = 1$.

As previously indicated, there are three methods to be examined in this experiment:

1 • The Euler Algorithm:

This is the most simple of the algorithms and runs as follows. Given that;

$$a_n \quad = \quad - w^2 . x_n$$

Then a single iteration goes as;

$$x_{n+1} \quad = \quad x_n + v_n \Delta t$$

$$v_{n+1} \quad = \quad v_n + a_n \Delta t$$

2 • The Velocity Form Of The Verlet Algorithm:

Based on a forward and backward Taylor expansion, the form of this algorithm per iteration is:

$$x_{n+1} \quad = \quad x_n + v_n \Delta t + \tfrac{1}{2}a_n(\Delta t)^2$$

$$v_{n+1} \quad = \quad v_n + \tfrac{1}{2}(a_{n+1} + a_n) \Delta t$$

3 • Gear's Predictor-Corrector without iteration using a third order predictor:

The non-iterative form of Gear's method consists of three steps, PEC:

- Predict (3rd order):

$$x^P_{n+1} \quad = \quad x_n + v_n(\Delta t) + {}^1/_{2!}\, a_n(\Delta t)^2 + {}^1/_{3!}\, b_n(\Delta t)^3$$

$$v^P_{n+1} \quad = \quad v_n + a_n(\Delta t) + {}^1/_{2!}\, b_n(\Delta t)^2$$

$$a^P_{n+1} \quad = \quad a_n + b_n(\Delta t)$$

$$b^P_{n+1} \quad = \quad b_n$$

Where b_n represents the third order derivative of x with respect to time for iteration n (given an initial value of zero).

- Evaluate:

Find the difference between the predicted acceleration and the acceleration evaluated from the differential equation, using the predicted positions. ie:

$$\Delta a \quad = \quad (-\omega^2 x^p_{n+1}) - (a^p_{n+1})$$

Now, define:

$$\Delta A \quad = \quad \Delta a\,(\Delta t)^2 / 2!$$

- Correct:

Use the above information to correct all predicted derivatives.

$$x_{n+1} \quad = \quad x^P_{n+1} + \alpha_0(\Delta A)$$

$$v_{n+1} \quad = \quad v^P_{n+1} + \alpha_1(\Delta A)/(\Delta t)$$

$$a_{n+1} \quad = \quad a^P_{n+1} + 2!.\alpha_2(\Delta A)/(\Delta t)^2$$

$$b_{n+1} \quad = \quad b^P_{n+1} + 3!.\alpha_3(\Delta A)/(\Delta t)^3$$

where,

$$a_0 \quad = \quad {}^1/_6$$

$$a_1 \quad = \quad {}^5/_6$$

$$a_2 \quad = \quad 1$$

$$a_3 \quad = \quad {}^1/_3$$

These values for the coefficients are looked up in tables and have been chosen to promote numerical stability. This scheme requires additional initial conditions such that:

$$a_0 = -\omega^2 x_0$$

$$b_0 = 0$$

Evaluation of the performance of the above techniques is carried out using the phase space plot and energy calculations as indicated before. Given a system where:

$$x_0 = 1$$

$$v_0 = 1$$

and,

$$m = k = 1$$

Then the oscillator should have a circular phase space trajectory of radius ●2 and conserve a total energy of 1. The range of Δt to investigate is that between 2.00 and 0.01.

At any point, the total energy of the SHO system is given by:

$$E = {}^1/_2\,(\omega^2 x^2 + v^2)$$

Method

The program I used was adapted from 'sho.f' as given on the back of the question sheet for this assignment. Once modified, the program consisted of the following basic structure:

Main Program Code

```
Define type for all program variables, double precision where needed.
CALL initialisation subroutine (see below).
CALL output routine (see below) to output initial conditions.
Main DO loop:
    Iteration DO loop (clocked):
        CALL calculation routine; Euler, Verlet, Gear (see below)
        Let $t = t + \Delta t$
        Calculate energy at new position.
        CALL output routine.
    END iteration loop, after n iterations.
    Find the number of seconds the iterations took and write the result to the screen.
    Wait for a key press.
    If key pressed was P then print  the screen.
END Main loop, unless key pressed was the space bar (ie do n more iterations).
Display on screen the total energy of the system at the last point in the simulation.
Tidy up and exit.
```

It can be seen from the above pseudo code that the program allows the user to carry the simulation on for as long as is desired, but is allowed to leave the simulation every n iterations.

Initialisation Subroutine

```
Decide which algorithm to use (ie set a variable to a particular value for whichever method).
Display program header and prompt user to enter a value for $\Delta t$ for the simulation.
Define initial conditions (x,v,a,b) and no. of iterations (n).
Initialise graphics systems and define window and axes to put phase space plot on.
Open file for output of phase space data.
```

Calculation Subroutine

This consists of a set of subroutines (called Euler, Verlet and Gear), each of which is passed the relevant variables (x, v, ω^2, Δt, as well as a and b for the Gear algorithm), and then each carries out the Euler/Verlet/Gear calculation as outlined in the theory section above.

Output Subroutine

```
Having been passed the current position and velocity variables, this routine simply:
    a) Outputs the phase space coordinates to the hard disc file, and
    b) Plots a dot on the screen at the x, v phase space position.
```

The program given in the appendix is my FORTRAN implementation of the above scheme, as adapted from the given code.

Analysing the performance of the routines:

The method of analysis breaks down into two sections, accuracy analysis and stability analysis. The simplest way to analyse how accurate the routines are is to set the period of simulation to a fixed values (10 seconds, about $1^1/_2$ periods of oscillation), and experiment with a range of values for Δt. The aim is to find the maximum value of Δt that gives a reasonable result for the total system energy after the 10 seconds of simulation. By 'a reasonable result' I mean to within some error range of the true values of 1.0, for this example 0.1% accuracy. This means we are looking for energy values within the range 0.999 to 1.001.

The analysis of the stability of the routines works in a similar way. This time we run the simulation for 100 - 1000 seconds (ie many periods of oscillation from many iterations) using the best time step from the accuracy analysis above. If a simulation is unstable then this will be illustrated graphically by the phase space plot, and so we can find out whether the best Δt values from the accuracy analysis are stable, and if not, find out at what value the solutions become acceptable stable.

Results

As indicated in the method section, the analysis of the performance of the routines breaks down into two sections, accuracy and stability.

- Accuracy Of The Algorithms:

The first routine that I examined was the Euler algorithm code. Starting with Δt set to 0.1, I ran the program and recorded the calculated value for the total system energy after 10 units of simulation time (ie the no. of iterations depends on Δt). Then, by altering the value of Δt, I managed to bring the error down to the 0.1% mark (See table 1).

As can be seen from the table, we require a very small time step to get the desired accuracy from the Euler method, to the point where 10 seconds of simulation time takes 29.3 seconds to calculate.

Figure 1 illustrates the characteristic of inaccuracy in the Euler method, ie the outward spiral phase space trajectory as shown. All Euler solutions spiralled outwards, and altering Dt just gets the spiralling down to an acceptable level. See the stability section later for more information.

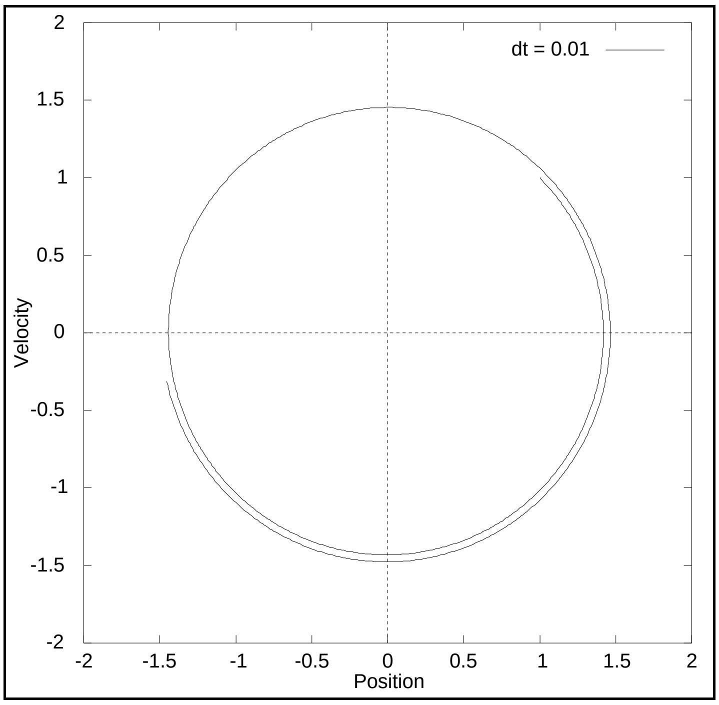


Figure 1: Euler at Δt = 0.01.

Time Step	Energy
0.1	2.7048
0.05	1.6477
0.01	1.1052
0.005	1.0513
0.001	1.0100
0.0005	1.0050
0.0001	1.0010

Table 1: Euler accuracy.

The second method I examined was the Verlet algorithm. The results for this were a lot better (see table 2). The table shows that a much larger time step can be used, in this case a time step of 0.05 gives the desired accuracy, a value some 500 times greater than that for Euler. The means that the calculation took only 0.054 seconds.

Time Step	Energy
0.5	1.03242
0.1	1.00115
0.05	1.00028

Table 2: Verlet accuracy.

Figure 2 below shows the true trajectory in phase space for the harmonic oscillator as calculated by the Verlet algorithm with Δt = 0.05s.

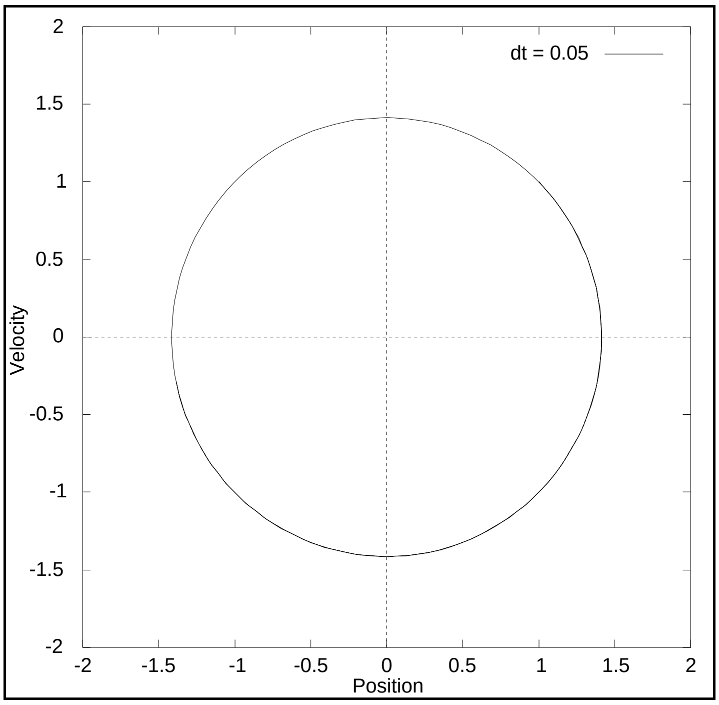
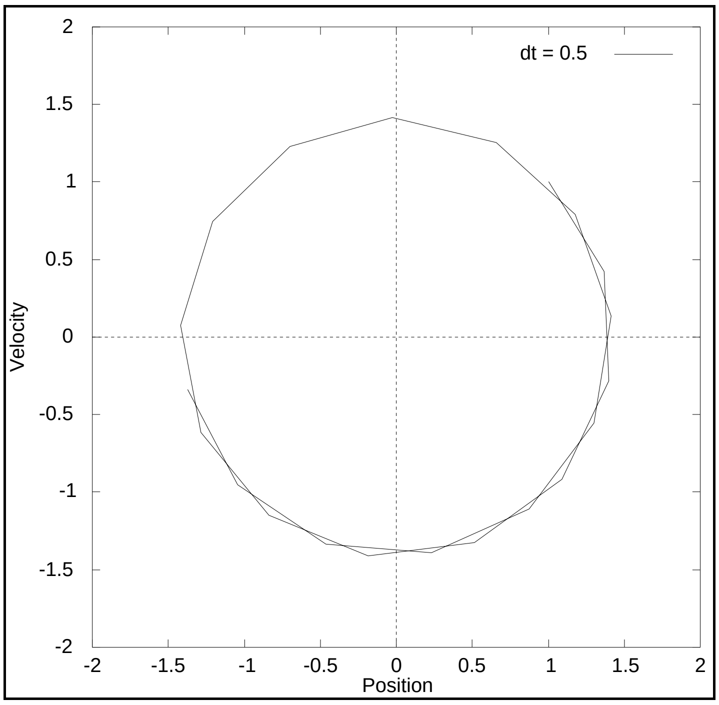


Figure 2: Verlet at Δt = 0.05.

As for the Gear algorithm, the relationship between the accuracy in the energy calculation for a particular time step went as shown in table 3. As can be seen from this data, the Gear code only needs a time step of 0.5 (10 times greater than Verlet) which in turn leads to a calculation time that will not register on the clock routines I used (ie less than one centisecond). Figure 3 show the phase space trajectory as calculated by the Gear method with Δt = 0.5.



Time Step	Energy
1.0	0.6553
0.5	0.99912

Table 3: Gear accuracy.

Figure 3: Gear at Δt = 0.5.

- Stability Of The Algorithms:

When run over long periods of time (ie tens of thousands of iterations), the characteristics of the routines become more apparent. All these experiments were conducted at the 0.1% accuracy level, ie with the values of Dt from the accuracy section above. As mentioned before, the Euler algorithm always spirals outward, and Δt just controls the degree of spiralling. Due to time restrictions, the longest simulation time I have been able to run is 100.0s as opposed to 10.0s before. This produced the phase space plot in figure 4.

Figure 4: Stability of the Euler algorithm.

From this it is clear that the Euler algorithm is to stable to a reasonable degree, but in order to get a better idea of performace it should really be compared with the other algorithms.

The Verlet algorithm was impressive in that it was stable for almost any value of Δt. Figure 5 shows the routine working at Δt = 1.0, and while the calculation is clearly in error (from the elliptical nature of the trajectory), the line is very well defined even after the 1000.0s of simulation carried out here. The 1000.0 seconds of simulation took 5.8 seconds to calculate, and the total system energy after this time was 0.999843 (ie a 0.0261% error).

Figure 5: Stability of the Verlet algorithm.

Finally we come to the stability of the Gear algorithm. While the routine performed well with $\Delta t = 0.5$ over 10.0 seconds of simulation, the story is very different for the 1000.0 second simulation (see figure 6). In order to stop the inward spiralling (ie energy loss), I reduced the value of Δt until the effect disappeared. The spiralling effect only became negligible at $\Delta t = 0.1$, however, even at this value the routine is still quicker whilst being more accurate than the Verlet algorithm; The 1000.0 seconds of simulation took 2.9 real seconds (ie two times quicker than Verlet), and gave a total system energy value of 0.9999843 (ie 0.00157% accuracy).

Figure 6: Stability of the Gear algorithm.

Conclusion

While all the routines performed this exercise reasonably well and in an acceptable time, most applications of these routines would require many more particles than in this case. This clearly makes the Euler algorithm unacceptable for serious computational problems.

Both the Verlet and Gear algorithms performed well, with Gear just being the quicker and more accurate of the two. While on the basis of this information, one might be tempted to recommend the Gear algorithm, the inherent instability in it's execution could cause problems for some simulations. In particular, this experiment gives no indication on how the routines would perform under many-body conditions, and any instability could well be amplified under these conditions.

From this I conclude that while the Verlet algorithm is slower and less accurate than Gear, it's 0.026% level of accuracy is sufficiently good to warrant its use on the basis of stability, as stability implies that the routine will perform consistently well under any circumstances.

Appendix

FORTRAN simulation code as adapted from 'sho.f':

```
      PROGRAM sho
c Finite difference simulation of a simple harmonic
oscillator
c
c Program adapted by AN Jackson, 1/1996.
c
      DOUBLE PRECISION x,v,a,b,energy,w2,dt,t
      REAL S,F
      INTEGER nsteps,method
      INTEGER*2 k
      CALL start(x,v,a,b,energy,w2,dt,nsteps,t,method)
      CALL output(x,v)
      k=32
c
      DO WHILE (k.eq.32)
       CALL CLOCK@(S)
       DO i=1,nsteps
        IF (method.eq.1) CALL Euler(x,v,w2,dt)
        IF (method.eq.2) CALL Verlet(x,v,w2,dt)
        IF (method.eq.3) CALL Gear(x,v,a,b,w2,dt)
        t=t+dt
        energy = 0.5d0*(w2*x*x + v*v)
        CALL output(x,v)
       ENDDO
       CALL CLOCK@(F)
       WRITE(*,*) ‘That took ‘,(F-S),’ secs for ‘
     +,nsteps,’ iterations.’
       CALL GET_KEY@(k)
       IF (k.EQ.ICHAR(’p’).OR.k.EQ.ICHAR(’P’)) CALL
hprintgs
      ENDDO
c
c Tidy up and exit:
c
      WRITE(21,*) ‘# Energy after ‘,NINT(t/dt),’
iterations:’
      WRITE(21,*) ‘#   ‘,energy
      CLOSE(UNIT=21)
      CALL hfinish
      WRITE(6,*) ‘Energy = ‘,energy
      STOP
      END
c
c
c
      SUBROUTINE
start(x,v,a,b,energy,w2,dt,nsteps,t,method)
      DOUBLE PRECISION x,v,a,b,energy,w2,dt,t
      REAL range
      INTEGER nsteps,method
      method=1
      WRITE(6,*)’One-Dimensional Simple Harmonic
Oscillator:’

WRITE(6,*)’~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~’
      IF (method.eq.1) WRITE(6,*)’[Euler version]’
      IF (method.eq.2) WRITE(6,*)’[Verlet version]’
      IF (method.eq.3) WRITE(6,*)’[Gear version]’
      WRITE(6,*)’ ‘
      WRITE(6,*)’ ‘
      WRITE(6,*)’Enter time step (sec):’
      READ(5,*)dt
      w2=1.0d0
      t=0.0d0
      x=1.0d0
      v=1.0d0
      a=-w2*x
      b=0.0d0
      range=2.0
      nsteps=NINT(100.0/dt)
      energy = 0.5d0*(w2*x*x + v*v)
c
c  Graphical initialisation
c
      CALL hprepit
      CALL hdefwin(1,-range,range,-range,range,
     +200,450,100,400)
      CALL hsetwin(1)
      CALL hsetcol(7)
      CALL hplotax(’position’,’velocity’)
      CALL hsetcol(7)
c
c  Output file initialisation
c
      OPEN(UNIT=21, FILE=’SHO.OUT’, STATUS=’UNKNOWN’)
      WRITE(21,*) ‘# Output form sho.for’
      IF (method.eq.1) WRITE(21,*)’#[Euler version]’
      IF (method.eq.2) WRITE(21,*)’#[Verlet version]’
      IF (method.eq.3) WRITE(21,*)’#[Gear version]’
      WRITE(21,*) ‘# ‘
      WRITE(21,*) ‘# position,velocity’
      RETURN
      END
c
c
      SUBROUTINE Euler(x,v,w2,dt)
c Euler algorithm
      DOUBLE PRECISION x,v,w2,dt,a
       a = -w2*x
       x = x + v*dt
       v = v + a*dt
      RETURN
      END
c
c
      SUBROUTINE Verlet(x,v,w2,dt)
c Verlet algorithm in its’ velocity form:
      DOUBLE PRECISION x,v,w2,dt,a1,a2
       a1 = -w2*x
       x = x + v*dt + 0.5d0*a1*dt*dt
       a2 = -w2*x
       v = v + 0.5d0*(a2+a1)*dt
      RETURN
      END
c
c
      SUBROUTINE Gear(x,v,a,b,w2,dt)
c Gear third-order PEC non-iterative algorithm:
      DOUBLE PRECISION x,v,a,b,w2,dt,Da,Idt
c Predict:
       x = x + v*dt + 0.5d0*a*dt*dt +
0.166666666d0*b*dt*dt*dt
       v = v + a*dt + 0.5d0*b*dt*dt
       a = a + b*dt
       b = b
c Evaluate:
       Da = 0.5d0*((-w2*x) - a)*dt*dt
c Correct:
       Idt=1.0d0/dt
       x = x + 0.166666666d0*Da
       v = v + 0.833333333d0*Da*Idt
       a = a + 2.0d0*Da*Idt*Idt
       b = b + 2.0d0*Da*Idt*Idt*Idt
      RETURN
      END
c
c
      SUBROUTINE output(x,v)
      DOUBLE PRECISION x,v
      WRITE(21,25) x,v
 25   FORMAT(5F12.6)
      CALL hmoveto(REAL(x),REAL(v))
      CALL hlineto(REAL(x),REAL(v))
      RETURN
      END
```