Simple Fortran and C Implementations of MD with Euler's Algorithm

The following implementations should be compatible with common Fortran and C compilers. The program reads a set of execution parameters and the initial conditions for a simulation from a plain-text input file and prints the positions of the particles at user-specified intervals. The execution parameters include, in order, the number of atoms in the chain (`nAtoms` = *n*), the force constant (`k`), the particle mass (`m`), the integration time increment (`h`), the time interval between printings (`hp`), and the length of the simulation (`len`). Any self-consistent set of units can be employed. The maximum number of atoms in the simulation, set internally by the parameter `nAtomsX`, is currently 100.

As an example, the input file corresponding to the simulation of Figure 3 is

```
3  1.0  1.0  0.01  0.2  20.0
1.0  0.0
0.0  0.0
-1.0  0.0
```

and a fragment of the output is

```
MOLECULAR DYNAMICS USING EULER INTEGRATION

nAtoms =    3
k =  1.0000E+00
m =  1.0000E+00
h =  1.0000E-02
hp =  2.0000E-01
len =  2.0000E+01


   Time      Atom #      Position
_____________________________________

 0.0000E+00   1    1.0000000000E+00
 0.0000E+00   2    0.0000000000E+00
 0.0000E+00   3   -1.0000000000E+00
 2.0000E-01   1    9.8104841125E-01
 2.0000E-01   2    0.0000000000E+00
 2.0000E-01   3   -9.8104841125E-01
 4.0000E-01   1    9.2291006930E-01
 4.0000E-01   2    0.0000000000E+00
 4.0000E-01   3   -9.2291006930E-01
```

```
* Program md_euler in Fortran

      implicit none

      integer  i, j, nAtoms, nAtomsX, nSteps, PrntFreq
      parameter( nAtomsX  = 100 )
*****************************************************************
* i, j     : Loop indices.
* nAtoms   : Number of atoms in chain. (IN)
* nAtomsX  : Maximum number of atoms permitted in chain.
* nSteps   : Number of time steps in computation.
* PrntFreq : Number of time steps between printing.
*****************************************************************

      double precision  ZERO
      parameter( ZERO = 0.0D0 )
      double precision  k, m, h, hp, len
      double precision  F(nAtomsX), Q(nAtomsX), V(nAtomsX)
*****************************************************************
* k     : Force constant. (IN)
* m     : Particle mass. (IN)
* h     : Time step. (IN)
* hp    : Time interval between printing. (IN)
* len   : Length of simulation: len = h*nSteps. (IN)
* F()   : Force vector.
* Q()   : Position vector. (IN/OUT)
* V()   : Velocity vector. (IN)
*****************************************************************


*  Read parameters and initial conditions from stdin
      read *,  nAtoms, k, m, h, hp, len
      do i = 1, nAtoms
           read *,  Q(i), V(i)
      end do

*  Compute number of integration steps and printing frequency
      nSteps = nint( len / h )
      PrntFreq = nint( hp / h )

*  Print header, parameters, and initial conditions
      print *, 'MOLECULAR DYNAMICS USING EULER INTEGRATION'
      print *
      print 93,  nAtoms
      print 94,  k
      print 95,  m
      print 96,  h
      print 97,  hp
      print 98,  len
      print *
      print *
      print *,  '   Time     Atom #     Position '
      print *,  '________________________________________'
      print *
      do i = 1, nAtoms
           print 99,  ZERO, i, Q(i)
      end do
```

```
*  Main loop
       do j = 1, nSteps
**  Compute right neighbor forces
            do i = 1, nAtoms - 1
                 F(i) = Q(i+1) - Q(i)
            end do
            F(nAtoms) = ZERO
**  Add left neighbor forces
            do i = 2, nAtoms
                 F(i) = F(i) + Q(i-1) - Q(i)
            end do
**  Advance position and velocity
           do i = 1, nAtoms
                 Q(i) = Q(i) + h*V(i)
                 V(i) = V(i) + h*k*F(i)/m
           end do
**  Print current results
           if ( mod(j, PrntFreq) .eq. 0 ) then
                 do i = 1, nAtoms
                      print 99,  j*h, i, Q(i)
                 end do
           end if
      end do


 93    format( 'nAtoms = ', I4 )
 94    format( 'k = ', 1E11.4 )
 95    format( 'm = ', 1E11.4 )
 96    format( 'h = ', 1E11.4 )
 97    format( 'hp = ', 1E11.4 )
 98    format( 'len = ', 1E11.4 )
 99    format( 1E11.4, 1X, I3, 3X, 1E17.10 )

       end
```

```
/* Program md_euler in C */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define nAtomsX 100           /* Maximum number of particles */

int main()
{

      int i, j;               /* Loop indices */
      int nAtoms;             /* Number of atoms in chain (IN) */

      double k;               /* Force constant (IN) */
      double m;               /* Particle mass (IN) */
      double h;               /* Time step (IN) */
      double hp;              /* Time interval between printing (IN) */
      double len;             /* Length of simulation: len = h*nSteps (IN) */
      double nSteps;          /* Number of time steps in computation */
      double PrntFreq;        /* Number of time steps between printing */
      double F[nAtomsX];      /* Force vector */
      double Q[nAtomsX];      /* Position vector (IN/OUT) */
      double V[nAtomsX];      /* Velocity vector (IN) */

      double ZERO = 0.0;


/* Read parameters and initial conditions from stdin */
      scanf("%d %lf %lf %lf %lf %lf", &nAtoms, &k, &m, &h, &hp, &len );
      for ( i = 0; i < nAtoms; i++ )
      {
            scanf( "%lf %lf", &Q[i], &V[i] );
      }

/* Compute number of integration steps and printing frequency */
      nSteps = floor( (len / h) + 0.5 );
      PrntFreq = floor( (hp / h) + 0.5 );


/*  Print header, parameters, and initial conditions */
      printf( "MOLECULAR DYNAMICS USING EULER INTEGRATION\n\n");

      printf( "nAtoms = %d\n", nAtoms );
      printf( "k = %10.4E\n", k );
      printf( "m = %10.4E\n", m );
      printf( "h = %10.4E\n", h );
      printf( "hp = %10.4E\n", hp );
      printf( "len = %10.4E\n\n", len );
      printf( "   Time     Atom #    Position\n" );
      printf( "_____________________________________\n\n" );
      for ( i = 0; i < nAtoms; i++ )
      {
            printf( "%11.4E %3d   %17.10E\n", ZERO, i, Q[i] );
      }

/* Main loop */
```

```
      for ( j = 0; j < nSteps; j++ )
      {
/** Compute right neighbor forces **/
            for ( i = 0; i < nAtoms - 1; i++ )
            {
                  F[i] = Q[i+1] - Q[i];
            }
            F[nAtoms - 1] = ZERO;
/** Add left neighbor forces **/
            for ( i = 1; i < nAtoms; i++ )
            {
                  F[i] = F[i] + Q[i-1] - Q[i];
            }
/** Advance position and velocity **/
            for ( i = 0; i < nAtoms; i++ )
            {
                  Q[i] = Q[i] + h*V[i];
                  V[i] = V[i] + h*k*F[i]/m;
            }
/** Print current results **/
            if ( fmod( (j + 1), PrntFreq ) == 0 )
            {
                  for (i = 0; i < nAtoms; i++ )
                  {
                        printf( "%11.4E %3d   % 17.10E\n",
                                                (j + 1)*h, i+1, Q[i] );
                  }
            }
      }

      return (0);
}
```

Suggested Exercises for the Program `md_euler`

The following exercises are listed roughly in order of increasing difficulty.

[1] By changing the value of *h* in the input file given in the supplementary material, run the simulation of Figure 3 with $h = 10^{-2}$, 10^{-3}, 10^{-4}, and 10^{-5}. Calculate the change in $q_1(5.0)$ after each reduction of *h* and then predict the result of changing *h* to 10^{-6}.

[2] By changing the values of *k* and *m* in the input file given in the supplementary material, run the simulation of Figure 3 with *k/m* = 0.25, 0.50, 1.0, 2.0, and 4.0. Plot the motions of the particles in each case using a mathematics/graphics package such as MATLAB. Finally, plot the frequency of particle 1 as a function of *k/m* and comment on the result. (Estimate the frequency in each case by simply examining the graph of the simulation.)

[3] Using the input file given in the supplementary material as a template, run a 20 s simulation with initial conditions

$$q_1(0) = 0.5, \quad q_2(0) = -1, \quad q_3(0) = 0.5; \quad v_i(0) = 0, \quad i = 1, 2, 3.$$

Plot the dynamics using a mathematics/graphics package such as MATLAB and compare with Figure 3.

[4] Using the input file given in the supplementary material as a template, run a 20 s simulation with initial conditions

$$q_1(0) = 1.5, \quad q_2(0) = -1, \quad q_3(0) = -0.5; \quad v_i(0) = 0, \quad i = 1, 2, 3.$$

and plot the dynamics using a mathematics/graphics package such as MATLAB. Since these initial conditions are just the sum of ones used in earlier exercises, *i.e.*

$$q_1(0) = 1.0 + 0.5, \quad q_2(0) = 0.0 + (-1), \quad q_3(0) = -1.0 + 0.5,$$

and similarly for the velocities, we expect that the $q_i(t)$ are likewise the sums of the corresponding position values obtained earlier. Verify that this is indeed the case.

[5] Examine the dynamics of some systems with $n > 3$ and plot the motions of the particles using a mathematics/graphics package such as MATLAB.

[6] We assumed in the text that the energy of the chain of particles is conserved. Modify `md_euler` so that it computes the total energy (potential + kinetic) of the chain and prints it along with the positions of the particles. Now run some experiments with different values of *n* and *h* and test whether the total energy is in fact constant.

[7] Modify `md_euler` so that the two end particles have mass m_{outer} and the interior particles have mass m_{inner}. For several values of *n*, illustrate how the dynamics change as the ratio m_{inner} / m_{outer} changes.

[8] Suppose the two end particles of the chain were nearest neighbors, as if the chain were actually a ring with periodic boundary conditions. Modify `md_euler` so that it solves the equations of motion for the ring with arbitrary *n*. For several values of *n*, compare the dynamics of the ring and chain models.