Molecular dynamics in different ensembles

Javier Junquera

Equations of motion for atomic systems

Classical equation of motion for a system of N molecules interacting via a potential $\mathcal{V}$

The most fundamental form: the Lagrangian equation of motion

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_k}\right) - \left(\frac{\partial \mathcal{L}}{\partial q_k}\right) = 0$$

Where the Lagrangian function $\mathcal{L}(\vec{q}, \dot{\vec{q}})$ is defined in terms of the kinetic and potential energies, and is considered to be a function of the generalized coordinates q_k and their time derivatives $\dot{q}_k$

$$\mathcal{L} = \mathcal{K} - \mathcal{V}$$

Equations of motion for atomic systems in cartesian coordinates

Classical equation of motion for a system of N molecules interacting via a potential $\mathcal{V}$

If we consider a system of atoms with Cartesian coordinates $\vec{r}_i$, and the usual definitions of the kinetic and potential energies

$$\mathcal{K} = \sum_{i=1}^{N} \sum_{\alpha} \frac{p_{i\alpha}^2}{2m_i}$$

$$\mathcal{V} = \sum_i v_1(\vec{r}_i) + \sum_i \sum_{j>i} v_2(\vec{r}_i, \vec{r}_j) + \sum_i \sum_{j>i} \sum_{k>j>i} v_3(\vec{r}_i, \vec{r}_j, \vec{r}_k) + \ldots$$

Then, the equation of motion transforms into

$$m_i \frac{d^2 \vec{r}_i}{dt^2} = \vec{f}_i$$

where m_i is the mass of atom i and the force on that atom is given by

$$\vec{f}_i = \nabla_{\vec{r}_i} \mathcal{L} = -\nabla_{\vec{r}_i} \mathcal{V}$$

Equations of motion for atomic systems in cartesian coordinates

Classical equation of motion for a system of N molecules interacting via a potential $\mathcal{V}$

$$m_i \frac{d^2 \vec{r}_i}{dt^2} = \vec{f}_i$$

This equation also aplies to the center of mass motion of a molecule, with the force begin the total external force acting on it

Hamilton's equation of motion in cartesian coordinates

$$\dot{\vec{r}}_i = \frac{\vec{p}_i}{m_i}$$

For a derivation of these expressions, read Goldstein, Chapter 8

$$\dot{\vec{p}}_i = -\nabla_{\vec{r}_i} \mathcal{V} = \vec{f}_i$$

Hamilton or Lagrange equations of motion

Lagrange's equation

$$m_i \ddot{\vec{r}}_i = \vec{f}_i$$

Hamilton's equations

$$\dot{\vec{r}}_i = \frac{\vec{p}_i}{m_i}$$

$$\dot{\vec{p}}_i = -\nabla_{\vec{r}_i} \mathcal{V} = \vec{f}_i$$

To compute the center of mass trajectories involves solving…

A system of $3N$ second order differential equations

Or an equivalent set of $6N$ first-order differential equations

Hamilton or Lagrange equations of motion

Lagrange's equation

$$m_i \ddot{\vec{r}}_i = \vec{f}_i$$

Hamilton's equations

$$\dot{\vec{r}}_i = \frac{\vec{p}_i}{m_i}$$

$$\dot{\vec{p}}_i = -\nabla_{\vec{r}_i} \mathcal{V} = \vec{f}_i$$

To compute the center of mass trajectories involves solving…

A system of $3N$ second order differential equations

Or an equivalent set of $6N$ first-order differential equations

Conservation laws

Assuming that $\mathcal{K}$ and $\mathcal{V}$ do not depend explicitly on time, so that

$$\frac{\partial \mathcal{H}}{\partial t} = 0$$

Essential condition:
No explicitly time dependent or velocity dependent force acting

Then, the total derivative of the Hamiltonian with respect to time

$$\dot{\mathcal{H}} = \frac{d\mathcal{H}}{dt} = \frac{\partial \mathcal{H}}{\partial t} + \frac{\partial \mathcal{H}}{\partial \vec{p}}\frac{\partial \vec{p}}{\partial t} + \frac{\partial \mathcal{H}}{\partial \vec{q}}\frac{\partial \vec{q}}{\partial t}$$

From the Hamilton's equation of motion

$$\dot{q}_k = \frac{\partial \mathcal{H}}{\partial p_k} \qquad \dot{p}_k = -\frac{\partial \mathcal{H}}{\partial q_k}$$

$$\dot{\mathcal{H}} = \frac{d\mathcal{H}}{dt} = \frac{\partial \mathcal{H}}{\partial t} - \frac{\partial \mathcal{H}}{\partial \vec{p}}\frac{\partial \mathcal{H}}{\partial \vec{q}} + \frac{\partial \mathcal{H}}{\partial \vec{q}}\frac{\partial \mathcal{H}}{\partial \vec{p}} = 0$$

The Hamiltonian is a constant of motion

Independent of the existence of an external potential

The equations are time reversible

By changing the signs of all the velocities and momenta,
we will cause the molecules to retrace their trajectories.

If the equations of motion are solved correctly, the
computer-generated trajectories will also have this property

Standard method to solve ordinary differential equations: the finite difference approach

Given molecular positions, velocities, and other dynamic information at a time t

We attempt to obtain the position, velocities, etc. at a later time $t + \delta t$, to a sufficient degree of accuracy

Notes:

The equations are solved on a step by step basis

The choice of the time interval δt will depend on the method of solution, but δt will be significantly smaller than the typical time taken for a molecule to travel its own length

Many different algorithms within the finite different methodology

- Predictor-corrector algorithm

If the classical trajectory is continuous, then an estimate of the positions, velocities, accelerations at $t + \delta t$ can be obtained by a Taylor expansion about time t

$$\vec{r}^{\,p}(t+\delta t) = \vec{r}(t) + \vec{v}(t)\delta t + \frac{1}{2}\vec{a}(t)\delta t^2 + \frac{1}{6}\vec{b}(t)\delta t^3 + \cdots$$

$$\vec{v}^{\,p}(t+\delta t) = \vec{v}(t) + \vec{a}(t)\delta t + \frac{1}{2}\vec{b}(t)\delta t^2 + \cdots$$

$$\vec{a}^{\,p}(t+\delta t) = \vec{a}(t) + \vec{b}(t)\delta t + \cdots$$

$$\vec{b}^{\,p}(t+\delta t) = \vec{b}(t) + \cdots$$

p **stands for "predicted"**

$\vec{r}$ **stands for the complete set of positions**

$\vec{v}$ **stands for the complete set of velocities**

$\vec{a}$ **stands for the complete set of accelerations**

$\vec{b}$ **stands for the complete set of third derivatives of the position**

We store four "vectors" per atom: $\vec{r}, \vec{v}, \vec{a},$ and $\vec{b}$

If the classical trajectory is continuous, then an estimate of the positions, velocities, accelerations at $t + \delta t$ can be obtained by a Taylor expansion about time t

$$\vec{r}^{\,p}(t+\delta t) = \vec{r}(t) + \vec{v}(t)\delta t + \frac{1}{2}\vec{a}(t)\delta t^2 + \frac{1}{6}\vec{b}(t)\delta t^3 + \cdots$$

$$\vec{v}^{\,p}(t+\delta t) = \vec{v}(t) + \vec{a}(t)\delta t + \frac{1}{2}\vec{b}(t)\delta t^2 + \cdots$$

$$\vec{a}^{\,p}(t+\delta t) = \vec{a}(t) + \vec{b}(t)\delta t + \cdots$$

$$\vec{b}^{\,p}(t+\delta t) = \vec{b}(t) + \cdots$$

p **stands for "predicted"**

If we truncate the expansion retaining the terms given above

We have achieved our goal of approximately advancing the values of the dynamical quantites from one time step to the next

We will not generate correct trajectories as time advances because we have not introduced the equations of motion

The equations of motion are introduced through the correction step

From the new predicted positions $\vec{r}^{\,p}(t+\delta t)$ we can compute the forces at time $t+\delta t$, and hence correct accelerations $\vec{a}^{\,c}(t+\delta t)$

The correct accelerations can be compared with the predicted accelerations to estimate the size of the error in the prediction step

$$\Delta\vec{a}(t+\delta t) = \vec{a}^{\,c}(t+\delta t) - \vec{a}^{\,p}(t+\delta t)$$

This errors, and the results of the predictor step, are fed into the corrector step

$$\begin{aligned}
\vec{r}^{\,c}(t+\delta t) &= \vec{r}^{\,p}(t+\delta t) + c_0\Delta\vec{a}(t+\delta t)\\
\vec{v}^{\,c}(t+\delta t) &= \vec{v}^{\,p}(t+\delta t) + c_1\Delta\vec{a}(t+\delta t)\\
\vec{a}^{\,c}(t+\delta t) &= \vec{a}^{\,p}(t+\delta t) + c_2\Delta\vec{a}(t+\delta t)\\
\vec{b}^{\,c}(t+\delta t) &= \vec{b}^{\,p}(t+\delta t) + c_3\Delta\vec{a}(t+\delta t)
\end{aligned}$$

These positions, velocities, etc. are better approximations to the true positions, velocities, etc.

"Best choices" (leading to optimum stability and accuracy of the trajectories) for the coefficients c_0, c_1, c_2 and c_3 discussed by Gear

Different values of the coefficients are required if we include more (or fewer) position derivatives in our scheme

The coefficients also depend upon the order of the differential equation being solved (second order in our case)

Table E.1 *Gear corrector coefficients for a first-order equation*

Values	c_0	c_1	c_2	c_3	c_4	c_5
3	5/12	1	1/2			
4	3/8	1	3/4	1/6		
5	251/720	1	11/12	1/3	1/24	
6	95/288	1	25/24	35/72	5/48	1/120

Table E.2 *Gear corrector coefficients for a second-order equation*

Values	c_0	c_1	c_2	c_3	c_4	c_5
3	0	1	1			
4	1/6	5/6	1	1/3		
5	19/120	3/4	1	1/2	1/12	
6	3/20	251/360	1	11/18	1/6	1/60

If the positions, velocities, accelerations, etc. are unscaled, factors involving the time scaled are required (see algorithm below)

Implementation of Gear's predictor-corrector algorithm

The predictor part

```
      SUBROUTINE PREDIC ( DT )

C     ********************************************************************
C     ** PREDICTOR ROUTINE                                              **
C     **                                                                **
C     ** IN TIMESTEP-SCALED VARIABLES THE PREDICTOR IS THE PASCAL       **
C     ** TRIANGLE MATRIX.  IN UNSCALED VARIABLES IT IS A TAYLOR SERIES  **
C     **                                                                **
C     ** USAGE:                                                         **
C     **                                                                **
C     ** PREDIC IS CALLED TO ADVANCE THE COORDINATES, VELOCITIES ETC.   **
C     ** BY ONE TIMESTEP DT, PRIOR TO FORCE EVALUATION.                 **
C     ********************************************************************
      INTEGER     N
      PARAMETER ( N = 108 )

      REAL        DT
      REAL        RX(N), RY(N), RZ(N)
      REAL        VX(N), VY(N), VZ(N)
      REAL        AX(N), AY(N), AZ(N)
      REAL        BX(N), BY(N), BZ(N)
      REAL        CX(N), CY(N), CZ(N)
      REAL        FX(N), FY(N), FZ(N)

      INTEGER     I
      REAL        C1, C2, C3, C4
C     ********************************************************************

!     Compute the coefficients for the Taylor expansion
      C1 = DT
      C2 = C1 * DT / 2.0
      C3 = C2 * DT / 3.0
      C4 = C3 * DT / 4.0
!     Loop over all the atoms in the system
      DO 100 I = 1, N
!        Predict the new positions
!        A Taylor expansion up to fourth order is done
!        That means that we have to store upto five arrays per atom
         RX(I) = RX(I) + C1*VX(I) + C2*AX(I) + C3*BX(I) + C4*CX(I)
         RY(I) = RY(I) + C1*VY(I) + C2*AY(I) + C3*BY(I) + C4*CY(I)
         RZ(I) = RZ(I) + C1*VZ(I) + C2*AZ(I) + C3*BZ(I) + C4*CZ(I)
!        Predict the new velocities
         VX(I) = VX(I) + C1*AX(I) + C2*BX(I) + C3*CX(I)
         VY(I) = VY(I) + C1*AY(I) + C2*BY(I) + C3*CY(I)
         VZ(I) = VZ(I) + C1*AZ(I) + C2*BZ(I) + C3*CZ(I)
!        Predict the new accelerations
         AX(I) = AX(I) + C1*BX(I) + C2*CX(I)
         AY(I) = AY(I) + C1*BY(I) + C2*CY(I)
         AZ(I) = AZ(I) + C1*BZ(I) + C2*CZ(I)
!        Predict the new third derivatives of the positions
         BX(I) = BX(I) + C1*CX(I)
         BY(I) = BY(I) + C1*CY(I)
         BZ(I) = BZ(I) + C1*CZ(I)
100   ENDDO
      RETURN
      END
```

Implementation of Gear's predictor-corrector algorithm

```
      SUBROUTINE CORREC ( DT, M, K )
C     ********************************************************************
C     ** CORRECTOR ROUTINE                                              **
C     **                                                                **
C     ** CORRECTS POSITIONS, VELOCITIES ETC. AFTER FORCE EVALUATION.    **
C     ** IN TIMESTEP-SCALED VARIABLES THE NUMERICAL COEFFICIENTS ARE    **
C     ** GIVEN BY GEAR (REF ABOVE): 19/120, 3/4, 1, 1/2, 1/12.          **
C     ** IN UNSCALED FORM THESE MUST BE MULTIPLIED BY FACTORS           **
C     ** INVOLVING THE TIMESTEP AS SHOWN HERE.                          **
C     **                                                                **
C     ** PRINCIPAL VARIABLES:                                           **
C     **                                                                **
C     ** REAL    M                       ATOMIC MASS                    **
C     ** REAL    K                       KINETIC ENERGY PER ATOM        **
C     ** REAL    GEAR0,GEAR1,GEAR2,GEAR3  GEAR COEFFICIENTS            **
C     **                                                                **
C     ** USAGE:                                                         **
C     **                                                                **
C     ** IT IS ASSUMED THAT INTERMOLECULAR FORCES HAVE BEEN CALCULATED  **
C     ** AND STORED IN FX,FY,FZ. CORREC SIMPLY APPLIES THE CORRECTOR    **
C     ** EQUATIONS BASED ON THE DIFFERENCES BETWEEN PREDICTED AND       **
C     ** EVALUATED ACCELERATIONS.  IT ALSO CALCULATES KINETIC ENERGY.   **
C     ********************************************************************
        INTEGER     N
        PARAMETER ( N = 108)
        REAL        DT
        REAL        RX(N), RY(N), RZ(N)
        REAL        VX(N), VY(N), VZ(N)
        REAL        AX(N), AY(N), AZ(N)
        REAL        BX(N), BY(N), BZ(N)
        REAL        CX(N), CY(N), CZ(N)
        REAL        FX(N), FY(N), FZ(N)
        REAL        M, K

        INTEGER     I
        REAL        AXI, AYI, AZI
        REAL        CORRX, CORRY, CORRZ
        REAL        C1, C2, C3, C4
        REAL        CR, CV, CB, CC
        REAL        GEAR0, GEAR1, GEAR3, GEAR4
        PARAMETER ( GEAR0 = 19.0 / 120.0, GEAR1 = 3.0 / 4.0,
     :              GEAR3 = 1.0 / 2.0,    GEAR4 = 1.0 / 12.0 )

C     ********************************************************************
        C1 = DT
        C2 = C1 * DT / 2.0
        C3 = C2 * DT / 3.0
        C4 = C3 * DT / 4.0

        CR = GEAR0 * C2
        CV = GEAR1 * C2 / C1
        CB = GEAR3 * C2 / C3
        CC = GEAR4 * C2 / C4
```

```
        K = 0.0
!       Loop over all the atoms in the system
        DO 100 I = 1, N
!          Compute the correct accelerations
           AXI = FX(I) / M
           AYI = FY(I) / M
           AZI = FZ(I) / M
!          Compare with the predicted accelerations to estimate
!          the size of the errors
           CORRX = AXI - AX(I)
           CORRY = AYI - AY(I)
           CORRZ = AZI - AZ(I)
!          Correct the positions.
!
           RX(I) = RX(I) + CR * CORRX
           RY(I) = RY(I) + CR * CORRY
           RZ(I) = RZ(I) + CR * CORRZ
!          Correct the velocities.
           VX(I) = VX(I) + CV * CORRX
           VY(I) = VY(I) + CV * CORRY
           VZ(I) = VZ(I) + CV * CORRZ
!          Correct the accelerations.
           AX(I) = AXI
           AY(I) = AYI
           AZ(I) = AZI
!          Correct the third derivative of the positions.
           BX(I) = BX(I) + CB * CORRX
           BY(I) = BY(I) + CB * CORRY
           BZ(I) = BZ(I) + CB * CORRZ
!          Correct the fourth derivative of the positions.
           CX(I) = CX(I) + CC * CORRX
           CY(I) = CY(I) + CC * CORRY
           CZ(I) = CZ(I) + CC * CORRZ
!          Compute the corrected kinetic energy
           K = K + VX(I) ** 2 + VY(I) ** 2 + VZ(I) ** 2
100     ENDDO
        K = 0.5 * M * K
        RETURN
        END
```

The corrector part

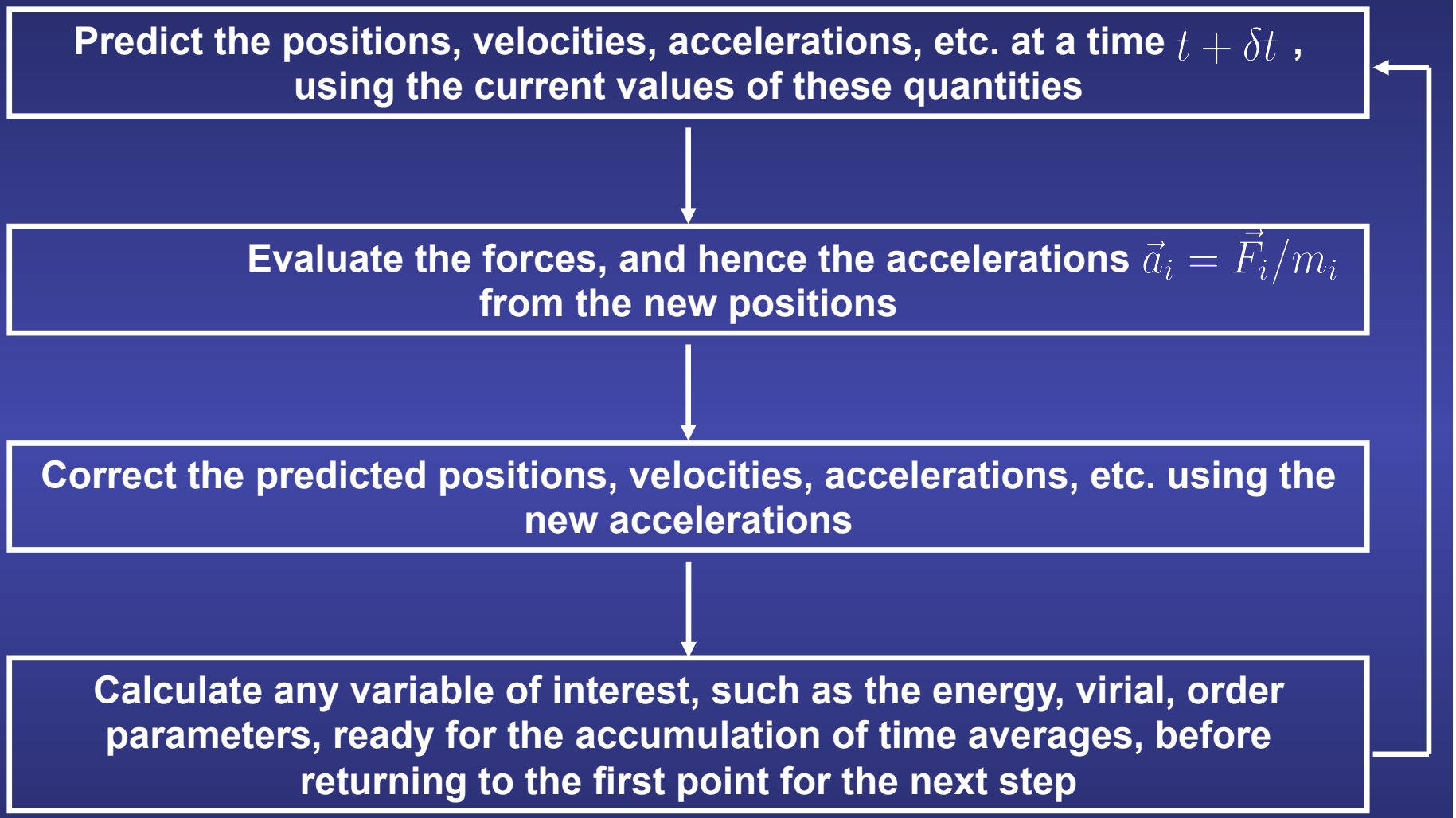
The corrector step might be iterated till convergence

New correct accelerations are computed from the positions $\vec{r}^{\,c}$ and compared with the current values of $\vec{a}^{\,c}$, so as to further refine the positions, velocities, etc.

In many applications this iteration is key to obtaining an accurate solution

However, each iteration requires a computation of the forces from particle positions (the most time-consuming part of the simulation)
$\Rightarrow$ A large number of corrector iterations would be very expensive.
Normally one (occasionally two) corrector steps are carried out

General step of a stepwise Molecular Dynamics simulation



Desirable qualities for a successful simulation algorithm

It should be fast and require little memory

Since the most time consuming part is the evaluation of the force, the raw speed of the integration algorithm is not so important

It should permit the use of long time step δt

Far more important to employ a long time step. In this way, a given period of simulation time can be covered in a modest number of steps

It should duplicate the classical trajectory as closely as possible

It should satisfy the known conservation laws for energy and momentum, and be time reversible

It should be simple in form and easy to program

Involve the storage of only a few coordinates, velocitites,…

Desirable qualities for a successful simulation algorithm

It should be fast and require little memory

Since the most time consuming part is the evaluation of the force, the raw speed of the integration algorithm is not so important

It should permit the use of long time step δt

Far more important to employ a long time step. In this way, a given period of simulation time can be covered in a modest number of steps

It should duplicate the classical trajectory as closely as possible

It should satisfy the known conservation laws for energy and momentum, and be time reversible

It should be simple in form and easy to program

Involve the storage of only a few coordinates, velocitites,…

The accuracy and stability of a simulation algorithm may be tested comparing the result with analytical simple models (Harmonic oscillator, for instance)

Any approximate method of solution will dutifully follow the classical trajectory indefinetely

Small differences in the initial conditions

Any two trajectories which are initially very close will eventually diverge from one another exponentially with time

Small numerical errors

Any small perturbation, even the tiny error associated with finite precision arithmetic, will tend to cause a computer-generated trajectory to diverge from the true classical trajectory with which it is initially coincident

The accuracy and stability of a simulation algorithm may be tested comparing the result with analytical simple models (Harmonic oscillator, for instance)

Any approximate method of solution will dutifully follow the classical trajectory indefinetely

Divergence of trajectories in molecular dynamics

As the time proceeds, other mechanical quantites become statistically uncorrelated

Small differences in the initial conditions

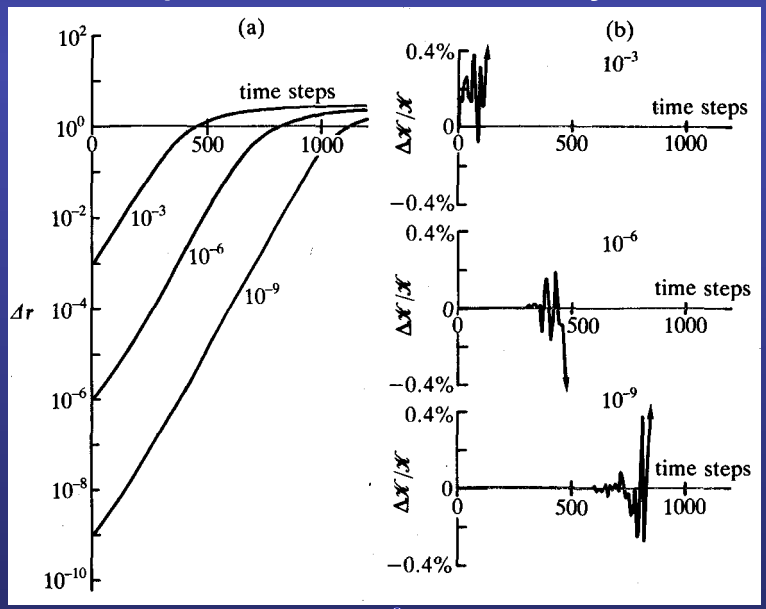
Any two trajectories which are initially very close will eventually diverge from one another exponentially with time

Molecules perturbed from the initial positions at t=0 by

$10^{-3}\sigma$

$10^{-6}\sigma$

$10^{-9}\sigma$



$$|\Delta\vec{r}|^2 = \frac{1}{N}\sum_{i=1}^{N}|\vec{r}_i(t) - \vec{r}_i^{\,0}(t)|^2$$

$\vec{r}_i^{\,0}(t)$ **Reference simulation**

$\vec{r}_i(t)$ **Perturbed simulation**

The accuracy and stability of a simulation algorithm may be tested comparing the result with analytical simple models (Harmonic oscillator, for instance)

Any approximate method of solution will dutifully follow the classical trajectory indefinetely

No integration algorithm will provide an essentially exact solution for a very long time

But this is not really required… What we need are:

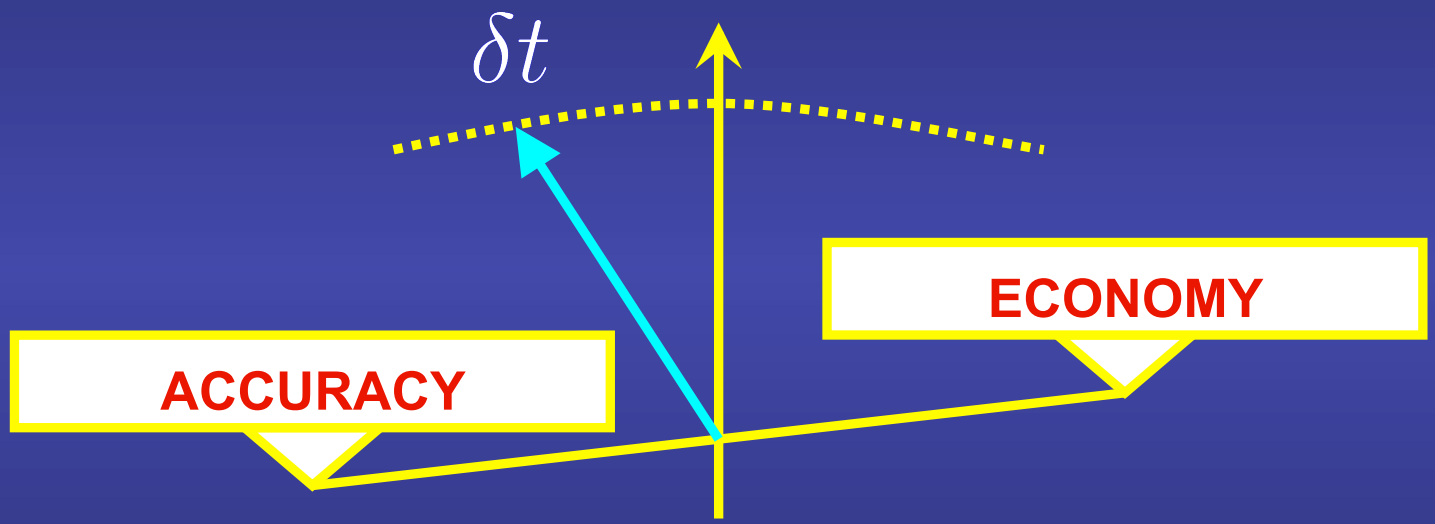
Exact solutions of the equations of motion for times comparable with the correlation times of interest, so we can calculate time correlation functions

The method has to generate states sampled from the microcanonical ensemble. The emphasis has to be given to the energy conservation.

The particles trajectories must stay on the appropriate constant-energy hypersurfaces in phase space to generate correct ensemble averages

Energy conservation is degraded as time step is increased

All simulations involve a trade-off between



A good algorithm permits a large time step to be used
while preserving acceptable energy conservation

Parameters that determine the size of δt

- Shape of the potential energy curves
- Typical particle velocities

Shorter time steps are used at high-temperatures, for light molecules, and for rapidly varying potential functions

The Verlet algorithm method of integrating the equations of motion: description of the algorithm

Direct solution of the second-order equations

Method based on:

- **the positions** $\vec{r}(t)$
- **the accelerations** $\vec{a}(t)$
- **the positions from the previous step** $\vec{r}(t-\delta t)$

A Taylor expansion of the positions around t

$$\vec{r}(t+\delta t) = \vec{r}(t) + \vec{v}(t)\delta t + \frac{1}{2}\vec{a}(t)\delta t^2 + \cdots$$

$$\vec{r}(t-\delta t) = \vec{r}(t) - \vec{v}(t)\delta t + \frac{1}{2}\vec{a}(t)\delta t^2 - \cdots$$

Adding the two equations

$$\vec{r}(t+\delta t) + \vec{r}(t-\delta t) = 2\vec{r}(t) + \vec{a}(t)\delta t^2 + \mathcal{O}(\delta t^4)$$

$$\vec{r}(t+\delta t) = 2\vec{r}(t) - \vec{r}(t-\delta t) + \vec{a}(t)\delta t^2 + \mathcal{O}(\delta t^4)$$

The Verlet algorithm method of integrating the equations of motion: some remarks

$$\vec{r}(t+\delta t) = 2\vec{r}(t) - \vec{r}(t-\delta t) + \vec{a}(t)\delta t^2 + \mathcal{O}(\delta t^4)$$

Remark 1

The velocities are not needed to compute the trajectories, but they are useful for estimating the kinetic energy (and the total energy).

They can be computed a posteriori using
[$\vec{v}(t)$ can only be computed once $\vec{r}(t+\delta t)$ is known]

$$\vec{v}(t) = \frac{\vec{r}(t+\delta t) - \vec{r}(t-\delta t)}{2\delta t} + \mathcal{O}(\delta t^2)$$

Remark 2

Whereas the errors to compute the positions are of the order of δt^4

The velocities are subject to errors of the order of δt^2

The Verlet algorithm method of integrating the equations of motion: some remarks

$$\vec{r}(t+\delta t) = 2\vec{r}(t) - \vec{r}(t-\delta t) + \vec{a}(t)\delta t^2 + \mathcal{O}(\delta t^4)$$

Remark 3

The Verlet algorithm is properly centered: $\vec{r}(t-\delta t)$ and $\vec{r}(t+\delta t)$ play symmetrical roles.

The Verlet algorithm is time reversible

Remark 4

The advancement of positions takes place all in one go, rather than in two stages as in the predictor-corrector algorithm.

The Verlet algorithm method of integrating the equations of motion: some remarks

- Let us assume that we have available current (RX) and old (RXOLD).
- The current accelerations (AX) are evaluated in the force loop as usual.
- Then, the coordinates are advanced in the following way

$$\vec{r}(t+\delta t) = 2\vec{r}(t) - \vec{r}(t-\delta t) + \vec{a}(t)\delta t^2 + \mathcal{O}(\delta t^4)$$

$$\vec{v}(t) = \frac{\vec{r}(t+\delta t) - \vec{r}(t-\delta t)}{2\delta t} + \mathcal{O}(\delta t^2)$$

```
      SUMVSQ = 0.0
      SUMVX  = 0.0
      SUMVY  = 0.0
      SUMVZ  = 0.0

      DO 100 I = 1, N

         RXNEWI   = 2.0 * RX(I) - RXOLD(I) + DTSQ * AX(I)
         RYNEWI   = 2.0 * RY(I) - RYOLD(I) + DTSQ * AY(I)
         RZNEWI   = 2.0 * RZ(I) - RZOLD(I) + DTSQ * AZ(I)
         VXI      = ( RXNEWI - RXOLD(I) ) / DT2
         VYI      = ( RYNEWI - RYOLD(I) ) / DT2
         VZI      = ( RZNEWI - RZOLD(I) ) / DT2
         SUMVSQ   = SUMVSQ + VXI ** 2 + VYI ** 2 + VZI ** 2
         SUMVX    = SUMVX + VXI
         SUMVY    = SUMVY + VYI
         SUMVZ    = SUMVZ + VZI
         RXOLD(I) = RX(I)
         RYOLD(I) = RY(I)
         RZOLD(I) = RZ(I)
         RX(I)    = RXNEWI
         RY(I)    = RYNEWI
         RZ(I)    = RZNEWI

100   CONTINUE
```

The Verlet algorithm method of integrating the equations of motion: some remarks

- Let us assume that we have available current (RX) and old (RXOLD).
- The current accelerations (AX) are evaluated in the force loop as usual.
- Then, the coordinates are advanced in the following way

Use of temporary variables (RXNEWI) to store the new positions.

This is necessary because the current values must be transferred to the "old" positions before being overwritten with the new variables.

The calculation of kinetic energy (SUMVSQ) and the total linear momentum (SUMVX) is included in the loop, since this is the only moment at which $\vec{r}(t-\delta t)$ and $\vec{r}(t+\delta t)$ are known.

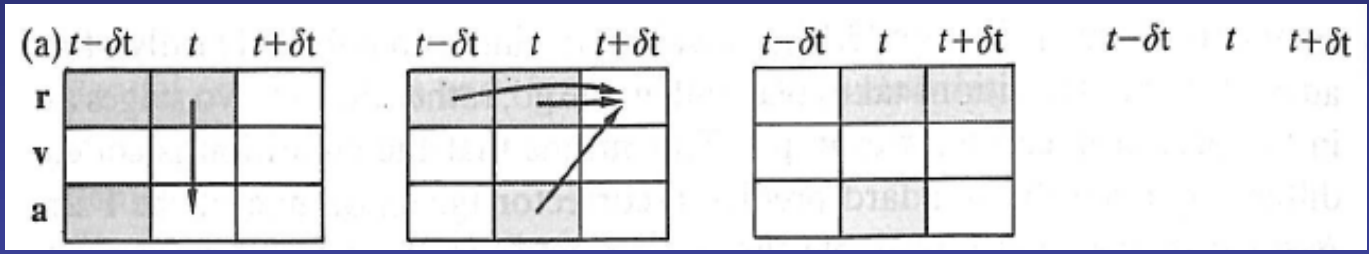
```
      SUMVSQ = 0.0
      SUMVX  = 0.0
      SUMVY  = 0.0
      SUMVZ  = 0.0

      DO 100 I = 1, N

         RXNEWI   = 2.0 * RX(I) - RXOLD(I) + DTSQ * AX(I)
         RYNEWI   = 2.0 * RY(I) - RYOLD(I) + DTSQ * AY(I)
         RZNEWI   = 2.0 * RZ(I) - RZOLD(I) + DTSQ * AZ(I)
         VXI      = ( RXNEWI - RXOLD(I) ) / DT2
         VYI      = ( RYNEWI - RYOLD(I) ) / DT2
         VZI      = ( RZNEWI - RZOLD(I) ) / DT2
         SUMVSQ   = SUMVSQ + VXI ** 2 + VYI ** 2 + VZI ** 2
         SUMVX    = SUMVX + VXI
         SUMVY    = SUMVY + VYI
         SUMVZ    = SUMVZ + VZI
         RXOLD(I) = RX(I)
         RYOLD(I) = RY(I)
         RZOLD(I) = RZ(I)
         RX(I)    = RXNEWI
         RY(I)    = RYNEWI
         RZ(I)    = RZNEWI

100   CONTINUE
```

The Verlet algorithm method of integrating the equations of motion: overall scheme



Known the positions at t, we compute the forces (and therefore the accelerations at t)

Then, we apply the Verlet algorithm equations to compute the new positions

…and we repeat the process computing the forces (and therefore the accelerations at $t + \delta t$)

The Verlet algorithm method of integrating the equations of motion: advantages

The Verlet algorithm requires $9N$ words of storage (RX, RXOLD, AX, and the corresponding for the y and z coordinates)

The algorithm is exactly reversible in time, and, given conservative forces, is guaranteed to conserve linear momentum and energy

The method has been shown to have excellent energy conserving properties even in long time steps

The Verlet algorithm method of integrating the equations of motion: drawbacks

The handling of velocities is rather awkward

It may introduce some numerical imprecision

$$\vec{r}(t+\delta t) = \underbrace{2\vec{r}(t) - \vec{r}(t-\delta t)} + \vec{a}(t)\delta t^2 + \mathcal{O}(\delta t^4)$$

A small term of order $\mathcal{O}(\delta t^2)$ …

…is added to a difference of large terms, of order $\mathcal{O}(\delta t^0)$

The half-step "leap-frog" scheme tackles the deficiencies of the Verlet algorithm

Method based on:

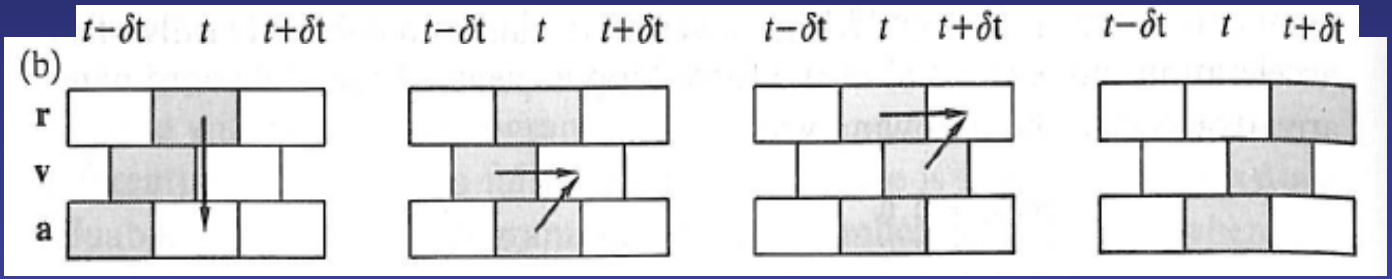
- **the positions** $\vec{r}(t)$
- **the accelerations** $\vec{a}(t)$
- **the mid-step velocities** $\vec{v}(t - \delta t/2)$

Algorithm

$$\vec{r}(t + \delta t) = \vec{r}(t) + \vec{v}(t + \delta t/2)\delta t$$

$$\vec{v}(t + \delta t/2) = \vec{v}(t - \delta t/2) + \vec{a}(t)\delta t$$

Scheme of the half-step "leap-frog" algorithm



The velocity equation is implemented first, and the velocities leap over the coordinates

$$\vec{v}(t+\delta t/2) = \vec{v}(t-\delta t/2) + \vec{a}(t)\delta t$$

Then, we apply the leap-frog algorithm to compute the new positions

$$\vec{r}(t+\delta t) = \vec{r}(t) + \vec{v}(t+\delta t/2)\delta t$$

Known the new positions, we compute the forces and therefore the accelerations…

…and we repeat the process computing the new velocities at $t+\delta t$

At a given t**, we can compute the velocities, required to calculate the total energy**

$$\vec{v}(t) = \frac{1}{2}\left[\vec{v}(t+\delta t/2) + \vec{v}(t-\delta t/2)\right]$$

The half-step "leap-frog" scheme: advantages and disadvantages

Advantages

Elimination of velocities from the leap-frog equations, the method is algebraically equivalent to Verlet's algorithm

At no stage do we take the difference of two large quantities to obtain a small value, minimizing loss of precision on a computer

Disadvantages

Still do not handle the velocities in a complete satisfactory way

The velocity Verlet algorithm: algorithm and advantages

Algorithm

$$\vec{r}(t+\delta t) = \vec{r}(t) + \vec{v}(t)\delta t + \frac{1}{2}\vec{a}(t)\delta t^2$$

$$\vec{v}(t+\delta t) = \vec{v}(t) + \frac{1}{2}\left[\vec{a}(t) + \vec{a}(t+\delta t)\right]\delta t$$

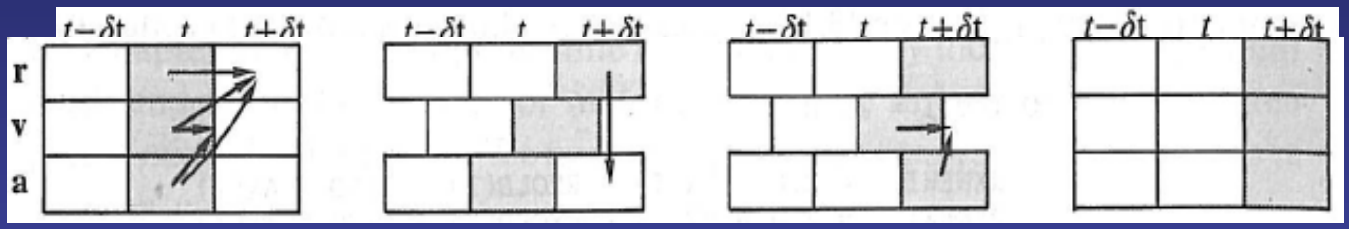
Advantages

Stores the positions, velocities, and accelerations at the same time t

Minimizes rounds-off errors

The Verlet algorithm can be recovered by eliminating the velocities

Scheme of the velocity Verlet algorithm



Known the positions, velocities and accelerations at t, we compute the new positions at $t + \delta t$

At this point the kinetic energy is available

The velocities at mid step are computed using

$$\vec{v}(t + \delta t/2) = \vec{v}(t) + \frac{1}{2}\vec{a}(t)\delta t$$

The potential energy has been evaluated in the force loop

Known the new positions, we compute the forces and therefore the accelerations at time $t + \delta t$

The velocity move is completed

$$\vec{v}(t + \delta t) = \vec{v}(t + \delta t/2) + \frac{1}{2}\vec{a}(t + \delta t)\delta t$$

The velocity Verlet algorithm: advantages

The method uses $9N$ words of storage

Numerically stable

Very simple

The velocity Verlet algorithm:

```
        SUBROUTINE MOVEA ( DT, M )

        COMMON / BLOCK1 / RX, RY, RZ, VX, VY, VZ, FX, FY, FZ

C     ***********************************************************************
C     ** FIRST PART OF VELOCITY VERLET ALGORITHM                            **
C     **                                                                    **
C     ** USAGE:                                                            **
C     **                                                                    **
C     ** THE FIRST PART OF THE ALGORITHM IS A TAYLOR SERIES WHICH           **
C     ** ADVANCES POSITIONS FROM T TO T + DT AND VELOCITIES FROM            **
C     ** T TO T + DT/2.  AFTER THIS, THE FORCE ROUTINE IS CALLED.           **
C     ***********************************************************************

        INTEGER     N
        PARAMETER ( N = 108 )

        REAL        DT, M
        REAL        RX(N), RY(N), RZ(N)
        REAL        VX(N), VY(N), VZ(N)
        REAL        FX(N), FY(N), FZ(N)

        INTEGER     I
        REAL        DT2, DTSQ2

C     ***********************************************************************

        DT2   = DT / 2.0
        DTSQ2 = DT * DT2

        DO 100 I = 1, N
!          The new positions at time (t + delta t) are calculated
           RX(I) = RX(I) + DT * VX(I) + DTSQ2 * FX(I) / M
           RY(I) = RY(I) + DT * VY(I) + DTSQ2 * FY(I) / M
           RZ(I) = RZ(I) + DT * VZ(I) + DTSQ2 * FZ(I) / M
!          The velocities at mid-step are computed
           VX(I) = VX(I) + DT2 * FX(I) / M
           VY(I) = VY(I) + DT2 * FY(I) / M
           VZ(I) = VZ(I) + DT2 * FZ(I) / M

100     ENDDO

        RETURN
        END
```

The velocity Verlet algorithm:

```
        SUBROUTINE MOVEB ( DT, M, K )

        COMMON / BLOCK1 / RX, RY, RZ, VX, VY, VZ, FX, FY, FZ

C     ***************************************************************
C     ** SECOND PART OF VELOCITY VERLET ALGORITHM                  **
C     **                                                           **
C     ** USAGE:                                                    **
C     **                                                           **
C     ** THE SECOND PART OF THE ALGORITHM ADVANCES VELOCITIES FROM  **
C     ** T + DT/2 TO T + DT. THIS ASSUMES THAT FORCES HAVE BEEN     **
C     ** COMPUTED IN THE FORCE ROUTINE AND STORED IN FX, FY, FZ.    **
C     ***************************************************************

        INTEGER     N
        PARAMETER ( N = 108 )

        REAL        DT, M, K
        REAL        RX(N), RY(N), RZ(N)
        REAL        VX(N), VY(N), VZ(N)
        REAL        FX(N), FY(N), FZ(N)

        INTEGER     I
        REAL        DT2

C     ***************************************************************

        DT2 = DT / 2.0

        K = 0.0

        DO 200 I = 1, N
!          Advances velocities from (t + \delta t/2) to (t + \delta t)
           VX(I) = VX(I) + DT2 * FX(I) / M
           VY(I) = VY(I) + DT2 * FY(I) / M
           VZ(I) = VZ(I) + DT2 * FZ(I) / M
!          Compute the kinetic energy
           K = K + VX(I) ** 2 + VY(I) ** 2 + VZ(I) ** 2

200     ENDDO

        K = 0.5 * M * K

        RETURN
        END
```

Is the code working?:
First check: the conservation laws are properly obeyed

The energy should be “constant”

In fact, small changes in energy will occur.
For simple Lennard-Jones system, fluctuations of 1 part in 10^4 are generally considered to be acceptable

Energy fluctuations might be reduced by decreasing the time step

Suggestion to check accuracy:

Several short runs should be undertaken, each starting from the same initial configuration and covering the same total time t_{run}

Each run should employ a different time step δt
(different number of steps $\tau_{\mathrm{run}} = t_{\mathrm{run}}/\delta t$)

The root mean square fluctuations for each run should be calculated

Is the code working?:
First check: the conservation laws are properly obeyed

Suggestion to check accuracy:

Several short runs should be undertaken, each starting from the same initial configuration and covering the same total time t_{run}

Each run should employ a different time step δt
(different number of steps $\tau_{\mathrm{run}} = t_{\mathrm{run}}/\delta t$ **)**

The root mean square fluctuations for each run should be calculated

If the program is functioning correctly, the Verlet algorithm should give RMS fluctuations that are accurately proportional to δt^2