分子動力学シミュレーションの基礎

—200行で書くコンピュータプログラム—

片岡洋右

法政大学工学部物質化学科

〒184 小金井市梶野町 3-7-2

松本充弘

名古屋大学工学部応用物理学教室

〒464-01 名古屋市千種区不老町

近年、物性研究で広く用いられるようになってきた分子動力学シミュレーションについて、その基本原理と技法およびデータ解析の基礎を、簡単なプログラム例を用いて解説する。

1. はじめに

物性研究の技術的進歩にともなって、種々の現象を分子レベルから理解したいという要望が多く分野で高まってきている。実験的に個々の分子のダイナミクスを直接観察することは、非常に困難を伴うことが多いため、分子の運動を計算機で追跡する分子シミュレーションの手法が注目を集めている。

本稿では、簡単なコンピュータプログラムを題材にして、分子動力学法（Molecular Dynamics 法、以下では MD 法と略記する）シミュレーションの基本原理やデータ解析の初歩を解説する。題材は、わずか 200 行の簡単な FORTRAN プログラムであるが、基本的な事項は一通り網羅しているので、他の分野の方々にも概念はつかんで戴けるものと思う。

より進んだ技法や実際問題への応用については、最近、国内外で多く刊行されている分子シミュレーションの教科書 [1-4] や解説書 [5-7] を参照して戴きたい。

2. 手法

本稿では、単純液体の代表としてよく取り上げられるアルゴン流体 (Lennard-Jones 流体) の MD シミュレーションを例に、そこで用いられる技法やデータの解析方法を概説するというスタイルで話を進める。用いるプログラムは素朴なもので、計算精度の向上や効率化、データストアの方法など改良の余地は多い。読者の方は、これを土台に種々の改造を試みられるとおもしろいと思う。

まずは、Fig.1 に掲げたプログラムリストをざっと眺めていただきたい。全体はいくつかのサブルーチン (sysdef, start, forces, motion, output, および again) に分かれており、これらを main (12-28 行目) が次々と呼び出すという単純な構造をしている。特殊な機能は使っていないので、FORTRAN コンパイラを備えたたいの計算機 (パソコンも含む) 上で、ほとんど変更することなく動作すると思う。唯一、機種依存性のある部分は、系の初期配置を作るために用いている [0, 1] の一様乱数 ran() (52-57 行目) であり、各機種の数値計算ライブラリに応じて適当に書き換えて戴きたい。また、リストを見易くするために、各サブルーチンで共通な宣言部は lj.incl という名のインクルードファイルにしてある (Fig.2)。コンパイル時には、忘れないようご注意ください。

わずか 200 行とはいえ、すべてを解説する紙数はないので、MD 法の基本と関係する部分だけを以下に説明する。

2.1 運動方程式の解法

出発点は、良く知られたニュートンの運動方程式である。単原子分子のような並進運動だけの系では、 i 番目の粒子の重心座標 $\vec{r}_i$ と運動量 $\vec{p}_i$ の時間発展は、次の式で記述できる：

$$\begin{aligned} d\vec{r}_i/dt &= \vec{p}_i/m \\ d\vec{p}_i/dt &= \vec{F}_i(\{\vec{r}\}) \end{aligned} \quad (1)$$

ここで、 m は粒子の質量、また $\vec{F}_i$ は i 番目の粒子に働く力で、粒子配置のみによって決まると仮定している。

分子数 N の多体系で、この N 次元連立微分方程式を解析的に解くことは一般に不可能なため、時間に関する差分方程式になおして数値積分を行い、系の挙動を調べることになる。数学的には、式 (1) の数値積分の問題ということで話は終わるのだが、得られた $\vec{r}_i(t)$ や $\vec{p}_i(t)$ は、物理的には、各分子の軌跡と運動量を表しており、これらを解析して分子の運動を追跡することができる。この方法が分子動力学法と呼ばれているのである。

微分方程式の数値解法はどんな数値計算の教科書にもあるので詳細は省略して、例題プログラムで用いている素朴な差分法（蛙跳び法、leap-frog 法）のみを簡単に紹介する。式 (1) を次のように差分化する：

$$\begin{aligned} \vec{r}_i(t + \Delta t) &= \vec{r}_i(t) + \Delta t \cdot \vec{p}_i(t + \frac{\Delta t}{2})/m \\ \vec{p}_i(t + \frac{\Delta t}{2}) &= \vec{p}_i(t - \frac{\Delta t}{2}) + \Delta t \cdot \vec{F}_i(\{\vec{r}(t)\}) \end{aligned} \quad (2)$$

Δt は差分化の時間刻みである。通常の差分化と違い、 $\vec{r}$ と $\vec{p}$ で時間を $\Delta t/2$ だけずらしているのがミソである。こうすることで、数値積分の際の誤差の蓄積と解の不安定化が防げる。例題プログラムでは、133-153 行目のサブルーチン motion が式 (2) により時間ステップを一つ進める部分である。

もとの運動方程式 (1) には散逸力（摩擦力など）がないので、こうした時間積分を続けても、数値誤差は別として、系の全エネルギーは保存されるはずである。そのチェックのためには、ポテンシャルエネルギーと運動エネルギーを同時刻で計算すればよい。しかし、運動量と座標の時刻は $\Delta t/2$ だけずれている。138-139 行目ではこのエネルギー計算のためだけに、わ

```

0001 c#####
0002 c# lj.fort : elementary MD simulation of LJ fluids #
0003 c#####
0004 block data
0005 include 'lj.incl'
0006 c***** parameters: time(fs), length(angstrom), T(kelvin)
0007 data nint,nrept,nout / 0, 8000, 10/
0008 data dtme,rcut,vsize,temp / 5.0, 12.0, 30.06, 100.0/
0009 c***** I/O file name
0010 data cufile / 'lj'/
0011 end
0012 c#####<< call subroutines >>#####
0013 program main
0014 include 'lj.incl'
0015 open( 6,file=cufile//'.prnt')
0016 open(10,file=cufile//'.save')
0017 call sysdef
0018 call start
0019 c***** main loop
0020 do 100 irept=1,nrept,1
0021 call forces
0022 call motion
0023 if (mod(irept,nout).eq.0) call output(1)
0024 100 continue
0025 c***** ending
0026 call output(2)
0027 call again(2)
0028 end
0029 c#####<< defines system parameters >>#####
0030 subroutine sysdef
0031 include 'lj.incl'
0032 smass=39.948
0033 fceng=1.0d+07/(boltz*avgdr)
0034 sigma=3.405
0035 eps=119.8/fceng
0036 cel2= 4*eps*(sigma**12)
0037 ce06= -4*eps*(sigma**6)
0038 cf12= cel2*12
0039 cf06= ce06* 6
0040 end
0041 c#####<< setup initial configuration >>#####
0042 subroutine start
0043 include 'lj.incl'
0044 real*4 ran
0045 if (nint.eq.0) then
0046 c***** read configuration data of previous run
0047 call again(1)
0048 else
0049 c***** create conf. with random number generator ran()
0050 iseed=12321
0051 do 100 i=1,num
0052 rx(i)=vsize*ran(iseed)
0053 ry(i)=vsize*ran(iseed)
0054 rz(i)=vsize*ran(iseed)
0055 px(i)=ran(iseed)
0056 py(i)=ran(iseed)
0057 100 pz(i)=ran(iseed)
0058 endif
0059 c
0060 c***** fix center-of-mass & rescale momentum
0061 sumpx=0.0
0062 sumpy=0.0
0063 sumpz=0.0
0064 do 200 i=1,num,1
0065 sumpx=sumpx+px(i)
0066 sumpy=sumpy+py(i)
0067 200 sumpz=sumpz+pz(i)
0068 sumpx=sumpx/num
0069 sumpy=sumpy/num
0070 sumpz=sumpz/num
0071 sumkin=0.0
0072 do 210 i=1,num,1
0073 px(i)=px(i)-sumpx
0074 py(i)=py(i)-sumpy
0075 pz(i)=pz(i)-sumpz
0076 210 sumkin=sumkin+(px(i)**2+py(i)**2+pz(i)**2)
0077 if (nint.eq.0) then
0078 fc=sqrt(1.5*temp/skin)
0079 else
0080 fc=sqrt(1.5*temp/(fceng*sumkin/(2*smass*num)))
0081 endif
0082 do 220 i=1,num,1
0083 px(i)=fc*px(i)
0084 py(i)=fc*py(i)
0085 220 pz(i)=fc*pz(i)
0086 c***** reset accumulators
0087 spot=0.0
0088 skin=0.0
0089 svir=0.0
0090 return
0091 end
0092 c#####<< calculate forces >>#####
0093 subroutine forces
0094 include 'lj.incl'
0095 engpot=0.0
0096 virial=0.0
0097 do 100 i=1,num,1
0098 fx(i)=0.0d0
0099 fy(i)=0.0d0
0100 100 fz(i)=0.0d0
0101 c
0102 rcutsq=rcut*rcut
0103 vhalf=vsize/2
0104 do 200 i=2,num,1
0105 do 200 j=1,i-1,1
0106 x=rx(i)-rx(j)
0107 if (x.le.-vhalf) x=x+vsize
0108 if (x.gt. vhalf) x=x-vsize
0109 y=ry(i)-ry(j)
0110 if (y.le.-vhalf) y=y+vsize
0111 if (y.gt. vhalf) y=y-vsize
0112 z=rz(i)-rz(j)
0113 if (z.le.-vhalf) z=z+vsize
0114 if (z.gt. vhalf) z=z-vsize
0115 rsq=x*x+y*y+z*z
0116 if (rsq.gt.rcutsq) go to 200
0117 ri06=1.0/(rsq*rsq*rsq)
0118 ri12=ri06*ri06
0119 fc=(cf12*ri12+cf06*ri06)/rsq
0120 frx=x*fc
0121 fry=y*fc
0122 frz=z*fc
0123 fx(i)=fx(i)+frx
0124 fy(i)=fy(i)+fry
0125 fz(i)=fz(i)+frz
0126 fx(j)=fx(j)-frx
0127 fy(j)=fy(j)-fry
0128 fz(j)=fz(j)-frz
0129 engpot=engpot+cel2*ri12+ce06*ri06
0130 virial=virial+x*frx+y*fry+z*frz
0131 200 continue
0132 end
0133 c#####<< move particles >>#####
0134 subroutine motion
0135 include 'lj.incl'
0136 engkin=0.0
0137 do 100 i=1,num,1
0138 engkin=engkin+((px(i)+0.5*dtme*fx(i))**2
0139 + (py(i)+0.5*dtme*fy(i))**2+(pz(i)+0.5*dtme*fz(i))**2)
0140 px(i)=px(i)+fx(i)*dtme
0141 py(i)=py(i)+fy(i)*dtme
0142 pz(i)=pz(i)+fz(i)*dtme
0143 rx(i)=rx(i)+px(i)/smass*dtme
0144 if (rx(i).lt. 0) rx(i)=rx(i)+vsize
0145 if (rx(i).gt.vsize) rx(i)=rx(i)-vsize
0146 ry(i)=ry(i)+py(i)/smass*dtme
0147 if (ry(i).lt. 0) ry(i)=ry(i)+vsize
0148 if (ry(i).gt.vsize) ry(i)=ry(i)-vsize
0149 rz(i)=rz(i)+pz(i)/smass*dtme
0150 if (rz(i).lt. 0) rz(i)=rz(i)+vsize
0151 if (rz(i).gt.vsize) rz(i)=rz(i)-vsize
0152 100 continue
0153 end
0154 c#####<< data output >>#####
0155 subroutine output(iout)
0156 include 'lj.incl'
0157 if (iout.eq.1) then
0158 c***** output at every step
0159 engpot=fceng*engpot/num
0160 engkin=fceng*engkin/(2*smass*num)
0161 write(6,600) irept,engpot,engkin,engpot+engkin
0162 600 format(i7,3f10.2)
0163 write(10,700) (rx(i),ry(i),rz(i),
0164 - px(i),py(i),pz(i),i=1,num)
0165 700 format(6e14.6)
0166 spot=spot+engpot
0167 skin=skin+engkin
0168 svir=svir+fceng*virial/num
0169 else

```

Fig.1 Sample program for molecular dynamics simulation of a Lennard-Jones system

```

0170 c***** output final statistics
0171       write(6,650) 'PE = ',spot/(nrept/nout), ' K'
0172       write(6,650) 'KE = ',skin/(nrept/nout), ' K'
0173       fcpres=1.0d+24*(num/vsize**3)*boltz/3
0174       write(6,650) 'P = ',fcpres*(2*skin+svir)/(nrept/nout), ' MP'
0175 650      format(a,e13.6,a)
0176      endif
0177      end
0178 c#####<< save & read restart data >>#####
0179      subroutine again(iagain)
0180      include 'lj.incl'
0181      if (iagain.eq.1) then
0182 c***** read restart data
0183      open(11,file=cufile/'.'.read')
0184      read(11,600) vsold,skin,
0185      - (rx(i),ry(i),rz(i),px(i),py(i),pz(i),i=1,num)
0186      close(11)
0187      scale=vsized/vsold
0188      do 100 i=1,num,1
0189      rx(i)=rx(i)*scale
0190      ry(i)=ry(i)*scale
0191 100      rz(i)=rz(i)*scale
0192      else
0193 c***** save restart data
0194      open(11,file=cufile/'.'.fnal')
0195      write(11,600) vsized,skin/(nrept/nout),
0196      - (rx(i),ry(i),rz(i),px(i),py(i),pz(i),i=1,num)
0197 600      format(2f10.3,(6e14.6))
0198      close(11)
0199      endif
0200      end

```

Fig.1 (continued)

```

0001 c*****<< lj.incl : file to be included >>#####
0002      implicit real*8 (a-h,o-z)
0003      parameter (num=512,boltz=1.3806d-23,avgr=6.022d+23)
0004      common/ nctrl/ nint,nrept,irept
0005      common/ rctrl/ dtim,rct,vsized,temp
0006      common/ molcl/ smass,cel2,ce06,cf12,cf06,fceng
0007      common/ cnfig/ rx(num),ry(num),rz(num),
0008      - px(num),py(num),pz(num),fx(num),fy(num),fz(num)
0009      common/ stcs/ engpot,engkin,virial,spot,skin,svir
0010      common/ io/ cufile
0011      character*2 cufile

```

Fig.2 Include file for the sample program

ざわぎ時刻を揃えた運動量を求めている：

$$KE(t) = \frac{1}{2m} \sum_i \left(\vec{p}_i(t - \frac{\Delta t}{2}) + \frac{\Delta t}{2} \vec{F}_i(t) \right)^2 \quad (3)$$

2.2 境界条件

実験では、アボガドロ数 (6×10^{23}) 程度の分子集団で現象を研究するのが通常だが、スーパーコンピュータといえどもこのような極端な多自由度系は計算できない。簡単な相互作用の場合は 100 万個程度の粒子系の計算例もあるが、普通は数百個から数千個程度のことが多い。現実の系なら、数百個ではクラスターやマイクロクラスターの領域となり、界面効果のため物性が変わってしまう。

シミュレーションにおいて、こうした界面の効果を取り除きバルクな状態を疑似的に実現する工夫が、周期境界条件である。Fig.3 を見て戴こう。中央の白い四角形が、計算機内に実現された「空間 (unit cell)」を表している。(3 次元では分かりづらいので 2 次元の概念図である。) この cell が壁で囲まれていたとしたら、壁の近くの系の物性はバルクなものとは異なるであろう。そこで、この壁を取り除いてやる。とはいえ、粒子数は有限だから cell を広げるわけにはいかない。そこで、無限空間にこの cell が周期的に並んでいる状態を想定する。隣合う cell 間には壁はないので、粒子は自由に cell 同士を行き来できる。しかし、自分と同じ粒子が隣の cell にもいるので、

Fig.3 Schematic figure to show the concept of periodic boundary conditions

1 つの粒子が隣の cell に移ると同時に反対側から粒子が再び入ってくることで、結局一つの cell 内の粒子数は一定に保たれる。

例題プログラムでは、粒子を動かしている 143-151 行目の if 文がこの周期境界条件を実現している箇所である。粒子を動かした結果、座標 rx が負になれば一辺の長さ vsized だけ加え、逆に rx が vsized より大きくなれば vsized だけ引くことにより、rx が常に $0 \leq rx < vsized$ を満たすようになっている。

空間充填ができさえすれば、cell の形は立方体でなくとも良い。また、粒子が cell に出入りする方法を少し変えることで、ざり流動下の系のシミュレーションもできる [4]。

2.3 ポテンシャルと力

粒子間の相互作用をどうするかは、シミュレーションの核心部分である。アルゴンなら単純な「分散力」だけでほぼ実験を再現できるが、水のような水素結合を持つ物質や、熔融塩のようにクーロン力の寄与が大きい場合は、精度のよいモデルポテンシャルを見つける (あるいは自分で作り出す) ことが重要で、ポテンシャル決定のために量子化学計算が必要になることもある。

ここでは、アルゴン流体で最もよく用いられる Lennard-Jones 相互作用だけを取り上げる。このモデルでは、系のポテンシャルエネルギー PE は粒子ペアの距離だけで決まる 2 体相互作用の和で表される：

$$PE = \sum_{i>j} U(|\vec{r}_i - \vec{r}_j|) \quad (4)$$

ペアポテンシャル $U(r)$ は、斥力項と引力項の和で表される：

$$U(r) = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \quad (5)$$

ここで ϵ はポテンシャルの強さ (深さ) を表すパラメタ、 σ は粒子の直径に相当するパラメタであり、アルゴンの場合は、 $\epsilon = 119.8 \text{ K}$, $\sigma = 3.405 \text{ \AA}$ がよく用いられている (プログラム 34-35 行目)。 $U(r)$ の概形は Fig.4 に示した。

粒子に働く力 $\vec{F}_i$ は、式 (4) を座標 $\vec{r}_i$ で微分して得られる：

$$\vec{F}_i = - \sum_{\substack{j=1 \\ j \neq i}}^N \frac{\vec{r}_i - \vec{r}_j}{r_{ij}} \frac{dU(r_{ij})}{dr_{ij}} \equiv \sum_{\substack{j=1 \\ j \neq i}}^N \vec{f}_{ij} \quad (6)$$

Fig.4 Lennard-Jones interaction: (a) exact (no cut-off) potential, (b) simple cut-off, (c) cut-and-shifted, (d) cut-and-smoothed. The unit is 4ϵ (ϵ : energy parameter) for the potential and σ (particle diameter) for the length

ただし $r_{ij} = |\vec{r}_i - \vec{r}_j|$ である。作用-反作用の法則から $\vec{f}_{ij} = -\vec{f}_{ji}$ が成り立つので、力の計算は各ペアについて一回で済む (プログラム 117-128 行目)。

ところで数百個程度の系では、cell は非常に小さい。(粒子数は `lj.incl` の 3 行目で `num` として与えてある。block data 中で与えている他のパラメータと別扱いしたのは、`num` は配列の大きさを指定するのに必要だからである。) 例えば、512 個の粒子系ならセルの一辺は粒子 8 個が並ぶ ($8^3 = 512$) 程度で、粒子-粒子相互作用を粒子 3 個分くらいで打ち切らなければ、周期境界条件のために相互作用を 2 重に数えたりする不都合が起きる。この点を考慮して、プログラムでは 106-114 行目の `if` 文で、隣のセルも含めて最小距離の粒子ペアを選び出している。

ポテンシャルの打ち切り (カットオフ) にも種々の方法がある。Fig.4 (b) は単純に r_c で相互作用を打ち切った場合である。(分かりやすいように $r_c/\sigma = 1.6$ という極端に短いカットオフ距離にしている。) r_c のところでポテンシャルギャップができるので、力の計算がここで破綻する。この力の発散を無視することは、結局、Fig.4 (c) のように r_c のところでポテンシャルが連続になるように上方にシフトしたのと同じことである。系のサイズが大きくて r_c が十分に大きくとれる場合には、このシフト分は非常に小さく無視できるので、通常は深く考えない。我々のプログラムでは、 $r_c = 12 \text{ \AA}$ に設定 (8 行目) してあるので、 $r_c/\sigma \simeq 3.5$ となり、単純なカットオフでまず問題はないが、圧力の計算に長距離部分の補正を加えればなお良い。

系が小さいときには、Fig.4 (d) に示すように r_c の近くでなめらかに接続するような細工を施すこともよく行われる。また、クーロン力のような長距離相互作用の場合は特に慎重に扱う必要があり、いくつかの技法が開発されている。

2.4 圧力

系の状態方程式を求めるためには系の圧力 P を計算しなければならない。これには、統計力学の教科書に必ず出てくる次のビリアル定理を利用する：

$$P = \frac{Nk_B T}{V} + \frac{1}{3V} \left\langle \sum_i \vec{r}_i \cdot \vec{F}_i \right\rangle \quad (7)$$

ここで、 k_B はボルツマン定数、 T は系の温度、 V は体積、 $\vec{r}_i$ と $\vec{F}_i$ は粒子 i の位置ベクトルとそこに働く力である。右辺 $\langle \rangle$ 内をビリアルと呼ぶ。2 体力の場合には、さらに

$$P = \frac{Nk_B T}{V} + \frac{1}{3V} \left\langle \sum_i \sum_{j>i} \vec{r}_{ij} \cdot \vec{f}_{ij} \right\rangle \quad (8)$$

と変形できることがわかる。例題プログラムでは、この形式を用いて、力の計算の際に同時にビリアルを求め (130 行目)、その平均値から圧力を計算している (173-174 行目)。

2.5 その他

本稿の例題プログラムのテストは、15.8 Mips の性能を持つやや旧型のワークステーションで行った。512 個の粒子系で 8000 ステップに約 7 時間を要した。最新の 32 ビットパソコン (例えば 66MHz の 486 マシン) でも十分に実行できると思う。最近では、1 桁速いワークステーションが容易に入手できるので、分子シミュレーションはますます利用しやすくなっている。

例題プログラムの内、以上で触れなかったところは MD 法の本質とは関係の少ない部分であるが、簡単に述べておく。

最初の block data ではシミュレーションの各種パラメータ、`nrept` (ステップ数)、`dtime` (時間刻み Δt)、`rcut` (相互作用のカットオフ距離)、`vsize` (cell の一辺の長さ)、`temp` (目標温度) などを与えている。また、このプログラムは入出力のために `lj.prt`、`lj.save`、`lj.read`、`lj.fnal` の 4 つの外部ファイルを用いるが、ファイル名は 10 行目の変数 `cufile` で変更できる。

圧力など、物理量の時間平均として表される熱力学量は、シミュレーションの実行中に計算できる (ファイル `lj.prt` に出力) が、一般には、粒子位置・運動量などをファイル (`lj.save`) に逐次保存した上で、シミュレーション終了後、別の解析プログラムを用いて解析を進めることが多い。プログラム 161-165 行目がデータ蓄積の部分である。サブルーチン `output` は、`main` から `nout` ステップごとに呼び出されて (23 行目) ファイルへのデータの書き出しを行うと共に、ポテンシャルエネルギー等の蓄積を行っている。この `output` は最後にもう一度呼び出され (26 行目)、各種時間平均を出力する。

Fig.5 は得られた `print` ファイルの一部である。最初の列は時刻 (ステップ数)、続く 3 つの列はポテンシャルエネルギー、運

```

10      -615.80      148.41      -467.39
20      -618.50      151.12      -467.39
30      -620.09      152.69      -467.39
7990    -618.02      150.65      -467.37
8000    -616.19      148.83      -467.35
途中省略...
PE = -.618348E+03 K
KE = .150955E+03 K
P   = -.363437E+01 MP

```

Fig.5 Example of lj.prnt.

動エネルギー、全エネルギー（いずれも単位は絶対温度）である。先に述べたように、全エネルギーは一定であるべきものであり、わずかなズレ（この例では 8000 ステップの内に 0.04 K 増加している）は数値積分の誤差の蓄積による。ここをチェックしながら、時間刻み Δt を調整することができる。最後の 3 行は、ポテンシャルエネルギー PE、運動エネルギー KE、および圧力 P (単位は $\text{MPa} = 10^6 \text{ N/m}^2 \approx 10$ 気圧) の時間平均である。揺らぎを調べるため、標準偏差も計算できるように改造するとおおい。運動の自由度が 3 なので、運動エネルギー $= \frac{3}{2} k_B T$ の関係がある。Fig.5 は目標温度 100 K の場合であるから、計算結果 $\text{KE} = 150.955 \text{ K}$ は妥当である。

このプログラムを動かす際に、最初（時刻ゼロ）の粒子配置はどうするか？ 統計力学の教えでは、どんな初期配置から出発しても十分な時間が経過すれば平衡状態に落ち着くはずである。その精神に従って、コントロールパラメタ nint が 1 の時は一様乱数により初期配置と初期運動量を作り出してシミュレーションをスタートさせるようになっている（49-57 行目）。多くの場合、これは極端に非平衡な状態から出発することになるから、最初は急激に温度が上昇する。そこで、初めの内は少ないステップ（例えば nrept=100）でシミュレーションを終えて、ファイル lj.fnal に最終配置を書きだし、それを出発点にして（即ち lj.fnal を lj.read にコピーし、パラメタ nint をゼロにする）シミュレーションを再開すると、設定温度になるよう運動量を適当にスケール変換するようになってある（77-85 行目辺りを参照）。こうした温度の再設定を数回（場合により 20 回程度）おこなった後、本番のシミュレーションにとりかかることになる。もっと頭のいい方法（温度を自動的にコントロールする方法）については文献を調べて欲しい。

この例題プログラムは MD 法の本質を理解するためのプロトタイプであり、効率は良くない。例えば、力の計算のためすべての粒子ペアの距離を計算しているので、粒子数 N が非常に大きい場合は、計算の無駄が大きい。種々の高速化の方法が開発されているので、MD シミュレーションを研究の道具として用いる際には、文献を参考に大幅な改良を行うことをお勧めする。

3. データ解析

シミュレーションはあくまでも手段であって目的ではない。目的はもちろん、得られたデータの解析を通しての物性の研究にある。データ解析法は研究対象によって様々であり、熱力学量などの静的性質を調べるだけで十分な場合もあれば、分子拡散のような動的性質に興味がある場合もある。すべてを網羅するわけには行かないので、典型的なデータ解析を数例示す。熱力学・統計力学の背景については、教科書等を参照されたい。

Fig.6 P - V relations of the Lennard-Jones fluid. The lines are guides for the eyes

3.1 熱力学量

このプログラムを用いて、温度 $T=100, 150, 200 \text{ K}$ 、体積 $V=0.74\text{--}1000 \text{ cm}^3/\text{g}$ の広いパラメタ領域で、各々数千ステップの計算をおこなってみた。いずれの条件でも最初の 2 千ステップ程度で熱平衡に達した。

Fig.6 が得られた P - V 曲線である。 $T=200 \text{ K}$ では理想気体に近い単調な P - V 曲線は、温度が下がるにつれて歪んでくる。 $T=100 \text{ K}$ ではいわゆる van der Waals ループらしきものが見え、気-液相分離の領域があることが分かる。ただし cell 内が一樣かどうか分からないので、Maxwell の等面積則が成り立つ保証はない。 $T=150 \text{ K}$ は気液臨界の近傍であろう（実験値は $T_c=150.7 \text{ K}$, $P_c=4.86 \text{ MPa}$ ）。こんな簡単なシミュレーションでも、現実の熱物性はかなり良く再現できる。

これらのうち、臨界温度より上の low-density fluid ($T=200 \text{ K}$, $V=1000 \text{ cm}^3/\text{g}$) と high-density fluid (200 K , $1 \text{ cm}^3/\text{g}$)、および下での liquid (100 K , $0.8 \text{ cm}^3/\text{g}$) の 3 つについて、lj.save ファイルのデータをもとに少し詳しく解析してみよう。

3.2 構造的性

流体の構造を調べるのに用いられるのが、動径分布関数 (Radial Distribution Function) $g(r)$ である。これは、粒子の中心から半径 r のところにある他の粒子の密度に相当し、粒子間距離のヒストグラムを計算することで得られる。Fig.7 に 3 つの状態の結果を載せた。Low-density fluid は中心粒子の存在にほぼ無関係に一樣な密度分布を持つので希薄気体と考えて良い。High-density fluid と liquid の $g(r)$ は山谷を持ち、相互作用ポテンシャルを反映したある程度の構造的性が見られる。

3.3 相関関数と動的諸量

粒子の拡散挙動を調べるのには、平均 2 乗変位 (Mean Square Displacement) が用いられる：

$$\text{MSD}(t) = \langle (\vec{r}_i(t) - \vec{r}_i(0))^2 \rangle \quad (9)$$

粒子が等速直線運動をするならば、 $\text{MSD}(t)$ は t の 2 次関数となる。Fig.8 の low-density fluid の結果はそれに近い。一方、

Fig.7 Radial distribution function $g(r)$ of the Lennard-Jones fluid

Fig.8 Mean square displacement of the Lennard-Jones fluid.

液体中のように、少し進むと他の粒子との相互作用ですぐに向きや速さを変える場合には、MSDは t の1次関数になり、その傾きは自己拡散係数に相当する。Fig.8のhigh-density fluid, liquidともそうした傾向を示しており、liquidの方が拡散係数が小さいことも分かる。

こうした動的挙動は、自己速度相関関数 (Velocity Autocorrelation Function) を用いて調べることもできる。規格化した形では、

$$\text{VAF}(t) = \frac{\langle \vec{v}_i(t) \cdot \vec{v}_i(0) \rangle}{\langle \vec{v}_i(0) \cdot \vec{v}_i(0) \rangle} \quad (10)$$

と定義される。Fig.9に示されるように、low-density fluidの中では、VAFの緩和が極めて遅い (粒子は長時間速度を保っている) のに対し、high-density fluidやliquidの中では、1ピコ秒以下で緩和してしまう。特に、liquidの場合には、VAFが負になる時間領域があり、粒子が周囲と衝突してはねかえされることを示唆している。こうしたVAFの減衰振動は低温の液体の特徴である。

4. 最後に

多原子分子など実際の問題にMD法を適用するには、種々の拡張が必要であろう。実用的なアルゴリズムが既に存在する場合もあれば、試行錯誤的にソフトウェアの開発から始めなければならない場合もある。種々の応用例については文献を参照して戴きたい。

Fig.9 Velocity autocorrelation function of the Lennard-Jones fluid

コンピュータのダウンサイジング化が叫ばれ、年ごとにコストパフォーマンスの良い計算機と周辺機器が入手できる現在では、分子シミュレーションによる物性研究はますますその重要性を増しつつあると考えている。

題材に取り上げたプログラムは、自由に移植・改造して使用載いて良い。コンピュータに打ち込むのが面倒(?)な読者は、松本 (E-mail アドレスは e43212a@nucc.cc.nagoya-u.ac.jp) まで連絡戴ければ、フロッピーディスクまたは電子メールでソースを提供できる。

参考文献

- [1] 岡田勲, 大澤映二編, 「分子シミュレーション入門」 (海文堂, 1989).
- [2] 上田顯, 「コンピュータシミュレーション」 (朝倉書店, 1990).
- [3] 河村雄行, 「パソコン分子シミュレーション」 (海文堂, 1990).
- [4] M.P.Allen, D.J.Tildesley, "Computer Simulation of Liquids" (Oxford University Press, 1987).
- [5] 田中實, 山本良一編, 「計算物理学と計算化学」 (海文堂, 1988).
- [6] 日本物理学会編, 「計算物理学」 (培風館, 1991).
- [7] K.Binder (ed.), "The Monte Carlo Method in Condensed Matter Physics" (Springer, 1992).

An Introduction to Molecular Dynamics Computer Simulation

Yosuke Kataoka
Dept. of Materials Chemistry,
College of Engineering,
Hosei University
Kajino 3-7-2, Koganei 184

Mitsuhiro Matsumoto
Dept. of Applied Physics,
School of Engineering,
Nagoya University
Nagoya 464-01

Basic concepts of molecular dynamics computer simulations are given with a brief sample program. Also shown are elementary techniques of simulation and examples of data analyses.

(Received May 6, 1993.)