N 体シミュレーション立春の学校教科書

牧野淳一郎/福重俊幸/小久保英一郎
川井敦/台坂博/斎藤貴之/杉本大一郎

2024.2.13-15

目 次

第1章　特別講義　重力の特殊性

1.1　ニュートンのりんご

ニュートンはりんごが落ちるのをみて万有引力を発見したと言われる。引力は質量のある物体の間に作用するもので、後に重力とも呼ばれるようになった。ニュートンが発見した要点は、すべての物体に対して働くこと、力の強さが質量に比例すること、さらに力の強さは物体間の距離の２乗に反比例するという事であった。少し考えてみるとわかるように、引力の距離に対する関数形は、りんごが落ちるのを見ているだけでは知ることができない。地球の中心からりんごまでの距離は、どのりんごについても等しいからである。

ニュートンはりんごが落ちるのと月が地球に向かって落ちるのとを比べたのだ、と言われている。月も地上の物体と同様に、力が作用しないときには等速直線運動をするはずである。しかし月は円軌道を描いている。それは月が等速直線運動をすると同時に（りんごと同様に）地球に向かって落ちてくるからである。両者の関係がうまくつりあって円運動になる。このように考えたのであろう。彼の偉いところは、当時の思想に反して、天上の運動と地上の運動を比較したことであった。この２つの、距離が大きく異なるものを比べたから、引力の関数形がわかったのである。さらにケプラーの第３法則［惑星の軌道半（長）径の３乗と公転周期の２乗の比は惑星によらず一定］を同様に解釈する（それぞれの惑星が太陽に向かって落ちる様子を月が地球に落ちる様子と同様に考える）と、それは惑星の数だけのデータを取り込んだものであるから、引力が距離の２乗に反比例することは（それ以上のデータを調べなくても）直ちにわかることになった。

この過程で、引力が質量に比例することもわかる。ケプラーの第３法則に現れる定数は「惑星によらない」値である。この形の法則がでてくるためには、惑星に作用する力は惑星の質量に比例しなければならない。惑星の運動方程式の左辺すなわち慣性力は質量に比例するが、右辺の力にも同様に質量に比例する力が現れ、両者はキャンセルしなければならない。また力が惑星の質量に比例するならば、作用・反作用の法則によって、力は太陽の質量にも比例するはずである。そしてケプラーの第３法則の次元解析、すなわち次元を合わすことによって、力は距離の２乗に反比例しなければならないことになる。

ニュートンがこのようにして発見するに至った万有引力の法則は、重力の基本的性質のほとんど全部を含んでいる。第１に万有であること。第２に万有引力は慣性力と同じように質量に比例するということ。第３にそれは引力だけであり、引

力だけしかないことである。第 4 の性質は力の到達範囲が無限大という遠距離力になっていることである。この第４の点のために、りんごについても、それに比べて桁違いに遠い距離にある惑星についても同様に成り立ち、両者を比べることができたわけである。このようにダイナミック・レンジの広い比較は、他の種類の力ではありえない。この遠距離力という性質が、その後に持つことになる深遠な意味については、後の節で詳しく議論する。

1.2　重力を空間の構造に繰込む

先ずは重力が質量に比例することの意味を論じる。これはアインシュタインによる一般相対性理論で、積極的に取り入れられた。そこでは次のことが問われる。慣性力は質量に比例するので、その比例定数となる質量を $m_{\rm i}$ と表して慣性質量と呼ぶ。それに対し、重力が比例する質量は $m_{\rm g}$ と表して重力質量とする。そこで両者の比、$m_{\rm i}/m_{\rm g}$ はどういう物質に対しても等しいかと問うことができる。何かある特定の物質に対して $m_{\rm i} = m_{\rm g}$ になるように、それぞれの質量の単位を決めておけば、この問いは、あらゆる物質に対して $m_{\rm i} = m_{\rm g}$ であるかという問いと同等である。

そのことは、もちろん実験的に決めていくべきものである。現在までに存在している実験では、両者の比が物質によって異なるということは見出されていない。しかし完全に等しいということは実験的に証明することができないので、アインシュタインはそれが等しいということを仮定し、それを等価原理と呼んで一般相対性理論の基礎的原理とした。

一般相対性理論を考えるときには、もちろんニュートンの運動方程式というわけにはいかないが、その原理はニュートンの運動方程式からも分かる。万有引力で作用しあっている物体からなるシステムを考える。そこである物体が受ける力は、その物体の質量に比例するので、前節でも述べたように、運動方程式の両辺から消える。このことがあらゆる物体について成り立つので、物体の運動を記述する形式から質量という言葉を消すことができる。質量と同時に万有引力という言葉を消し、そして重力の効果を空間の構造の中に繰込むことができることになる。その空間の構造は、どのような質量の物体に対しても同じ影響を与える、というわけである。言い換えると、その空間の構造にしたがって物体が運動することになる。

空間の構造は、引力の計算と同様に、物体の分布（や運動）と関係づけて、アインシュタイン方程式によって決められる。こうして一般相対性理論は物体間に作用する万有引力という言葉から出発し、それを空間の構造に取り込むことによって、万有引力という概念を理論から追放してしまったことになる。この理論体系は宇宙の膨張や逆にブラックホールに収縮していく天体を取り扱うのに都合のいいものであったし、そういう問題は一般相対性理論によって初めてきちんと扱えるようになった。しかしここでは、これらの問題には直接の深入りはしない。そ

れは一般相対性理論の効果が重要になる諸現象を説明することになるだけで、重力の持っている特殊な性質を総合的に議論することにはならないからである。

ただ一つだけ注意しておきたいことに、一般相対性理論だとなぜ宇宙論に都合がよいかという問題がある。偏微分方程式で記述される物理系を解くということは、境界条件を課して、その偏微分方程式が持っている一般の解の中から、ある特定の解を選び出すことに対応する。宇宙の場合には、境界条件を課すべき宇宙の果て（系の境界）を定めることが難しい。そこで宇宙論はそれを避けて、定曲率空間だという条件にして偏微分方程式を解く。これは空間の構造がどこでも同じだとして、言い換えると、どこでもが同じ境界だとして問題を解いていることになる。こうしておいて、その結果から、「宇宙の果て」とはいったいどういうものだと考えればよいかが、議論できるようになる。

1.3 重力の遠距離性

力の基本的な性質は湯川ポテンシャルによって表現することができる。すなわち、

$$V \sim -\frac{e^{-r/\lambda}}{r}. \tag{1.1}$$

これはべき関数と指数関数からなる。一般にべき関数は特性的な長さ（物理量）を含まないが、指数関数は、exp の肩に乗る数値が無次元でなければならないので、それを無次元にするために同じ次元の特性的物理量が現れる。湯川ポテンシャルの場合、それは距離を測る単位としての λ である。式からわかるように距離 r が λ に比べて有意に大きくなると、ポテンシャルの値は急激に小さくなるので、λ は力の到達範囲と呼ばれる。この λ の大きさは素粒子論の観点からいうと、力を媒介する中間子などの粒子のコンプトン波長であり、質量に反比例するものである。

自然界にある 4 つの力のうち、強い力と弱い力はそれぞれに対応する中間子の質量によって決まる到達距離をもっており、それは fm (10^{-15} m) のオーダーである。それに対し、電磁気力は質量がゼロの光子によって媒介されるので、λ が無限大の遠距離力になっている。その結果、ポテンシャルは $1/r$ というべき関数だけによって表されることになる。このポテンシャルを微分して得られる力は距離の 2 乗に反比例するもので、重力の場合も同じである。それは重力を媒介するグラビトンの質量がゼロであるということに言い換えることができるが、グラビトンという粒子は他の素粒子ほどはっきりしているわけではない。しかし距離の 2 乗に反比例する力であるということが実験室スケールから天体スケールまで確認されているのだから、λ はその程度には大きい値であることが実験的に示されているのだといってよい。

距離の 2 乗に反比例する力が「遠」距離力であることを最もよく納得させるのは、次のような事実である。密度 ρ の物質が空間に一様に分布しているとする。その中のある点をとり、そこに及ぼされる引力を計算する。ある小さい立体角 $d\omega$ の

方向にある物質からの引力を計算する。距離 r から $r+dr$ の範囲内にある物質の質量は $\rho r^2 dr d\omega$ であり、それが距離の 2 乗に反比例する力を観測点に及ぼすから、力の大きさは $\rho dr d\omega$ に比例する。こうして、$dr d\omega$ の範囲内にある物質はその距離とは関係なく同じ引力を及ぼすことになる。そして、それを無限遠まで全部足しあげると、引力は発散することになる。こうして、遠距離にあっても同じだけの影響を及ぼすという意味で遠距離力なのである。なお、この距離 2 乗に反比例する影響は、光源が一様に分布しているときに、ある観測点に到達する光束（フラックス）を計算するのでも同じことである。これは、等しい光度の光源が無限に一様に分布しているとき、ある観測点に到達する光のフラックスが発散するということであり、それは「夜空は無限に明るいことになる」というオルバースのパラドックスとしてよく知られている。なお、重力の場合にはフォン・ノイマン= ゼーリガーのパラドックスとも呼ばれる。このようなことは、力の到達範囲 λ が有限となる力については起こらない。

同様な事情は、互いに相互作用をしている粒子を衝突させたときの散乱断面積にも現れる。粒子の散乱において、散乱される粒子が的になる粒子をどのような大きさに見るか、どの程度の大きさのものとして影響を受けるかということは散乱断面積によって記述される。近距離力で作用している粒子どうしの散乱では、散乱断面積は $\pi\lambda^2$ のオーダーの大きさである。それに対し、重力の場合には、散乱断面積は発散して無限大になる。

距離の 2 乗に反比例するクーロン力で相互作用をしている電荷どうしの散乱でも同様に散乱断面積は発散する。クーロン力も遠距離力だからである。しかし電荷にはプラスとマイナスがある。そのため、現実の場面ではプラスの電荷の周りにマイナスの電荷が集まっている。それを第 3 の粒子が離れたところから見ると、プラスの電荷による影響とマイナスの電荷による影響が打ち消し合っていて、第 3 の粒子が影響されることはない。どのくらいの範囲にまで影響が及ぶと考えてもいいのかという点については、例えば原子の場合には原子の大きさの程度（原子を離れたところから触っても感電しない)、プラズマで電子やイオンが熱運動をしている場合にはデバイ半径という特性的な量（その値は温度と密度による）の程度である。そしてその外には電荷の影響は殆ど及ばない。こうして力の実際上の到達距離、すなわち有効到達距離（effective range）$\lambda_{\rm eff}$ が決まる。それは有限の大きさだから、重力の場合と違って、影響が無限大になることはない。

こうして重力だけが「本質的な」遠距離性を示すことになる。このことが重力を他の力と区別して、特殊な性格をもたせる原因になっているわけである。

1.4 現象の相似性

遠距離力であるということには 2 つの重大な意味がある。一つは影響が遠くまで及ぶために、遠距離相関をとおして、われわれの常識を超えた現象が起こるということである。このことについては後で述べる。もう一つは、その力を特徴づ

ける特性的な長さが存在しないために、そのような力に影響されて起こる現象は特性的な長さを持たないことである。この節ではその意味を論じる。

現象に特徴的な長さが現れないということは、言い換えると、どのようなスケールでも似たような現象が起こるということである。ニュートンのりんごと惑星は同じく相似的な振舞いを示す。例えば天体の爆発現象は、中性子星のX線バーストから新星爆発、超新星爆発まで、エネルギーでも時間スケールでも何桁も異なるが、基本的には似たような現象がおこる。

そのような重力が関係して起こる現象は、べき関数で表される性質のものになる。自然現象を数学的に記述するとき、指数関数とべき関数は根本的に違う性質をもっている。指数関数では、例えば力の到達距離のように特性的な物理量が出てくる。それがないと指数関数はつくれないからである。そのような関数で記述される現象は特性的な物理量に規定される比較的狭い適用範囲に限られる。

それに対し、重力だけだと特性的な長さをもつ現象は作れない（特性的物理量をもつ他の現象と結合するときには、そちらによって決まるだけである）。だから、結果として起こる現象はべき関数で記述される。その結果、現象のスケールを変換することができる。たとえば、ax^p という関数は、x を測る単位を変えても係数 a に繰込むことができるので、変数の p 乗であるということは変化せず、結果のスケール（係数）を変更しただけになる（指数関数の場合は変数の単位を変えたことの影響は、係数の変更だけにして取り出すことはできない）。その結果、現象は自己相似的になる（カオス的になること、相関関数がべき関数になることなどとも関係がある）。膨張宇宙には銀河の分布について大規模な構造が現れるが、それらは超銀河団のスケールから銀河のスケールまで何桁にもわたって自己相似的に存在する。このようなことは到達範囲が有限の力では起こらない。例えば到達範囲が fm 程度である核力の現象では、問題になる現象はその程度の範囲に限られており、大きさがそれより何桁も大きい原子核が存在したりすることはない。

1.5 自己重力に支配される系は常にミクロな系である

力の到達範囲が無限大であることの最も重要な影響は、「天体や宇宙のように、どんなにサイズの大きい系を考えても、それは力の到達範囲よりは小さいから、（相互作用の観点からは）ミクロな系である」という、一見したところ矛盾しているような状況が成り立つことである。

ミクロなシステム（系）とマクロなシステムで根本的に違うのは次のことである。マクロなシステムでは、そのエネルギーは、一般には、そのシステムを構成している要素ないしは粒子の数 N に比例する。例えばガスの熱運動のエネルギーがそうである。そのような（内部）エネルギーは物質の量に比例するので、外延的 (extensive) な量、ないしは示量変数と呼ばれる。それに対し、システムのサイズが力の到達範囲より小さい時には、その相互作用のエネルギーはシステムの中にあ

る粒子の組み合わせの数に比例するので、N^2 に比例する。そのようなエネルギーは自己エネルギーと呼ばれるが、先ほどの意味での外延的な示量変数ではない。

あるシステムのサイズが λ より大きい、すなわちマクロなシステムであるとき、そのシステムを λ の程度の大きさの領域（作用球）に分けて考え、その中に n 個の粒子があるとする。するとその作用球における自己エネルギーは n^2 に比例する。そのような作用球は N/n 個あるから、全体の自己（相互作用）エネルギーは、両者の掛け算で、Nn に比例することになる。これは N の 1 乗に比例する外延的な量になっているので、マクロなシステムを考えるとき、熱運動のエネルギーだけでなく自己エネルギーを加えても事情は変わらない。

これに対し、重力の場合には、要素のランダムな運動のエネルギーは外延的な量であるが、自己エネルギーを含んだ全エネルギーはもはや外延的な量ではない。典型的な例は、1 個の恒星である。恒星のガスがもっている熱エネルギーは星の質量に比例するが、恒星を一つの重力で結合された系に仕立てている重力エネルギーという自己エネルギーは、質量の 2 乗に比例する。そして、両者のエネルギーがいつも同じオーダーの大きさになっている。こうして、恒星のように自己重力で結合された系の振舞いを議論するには、エネルギーが外延量ではない効果を考慮して行う必要がある。それを繰込んだ熱力学を重力熱力学と呼んでいる。

外延的でないエネルギーが現れるのは、何も重力に限ったわけではない。物体の界面や表面張力に関係したエネルギーもそうだ。そしてそのような系の振舞いのことはよく知っている。このように言われるかもしれない。しかしそのような場合は、エネルギーは質量の 2/3 乗（表面積）に比例する。それは外延的な 1 乗の場合と比べて、より弱い依存性である。それに対し、重力の場合は 2 乗で、より強い依存性の場合である。エネルギーが超外延量であるといってもよい。そして、基準になる 1 乗の場合と比べてより弱いか、より強いかによって、系の振舞いは全く異なるのである。

1.6　重力のもたらす自発的形態形成

重力の場合のもっとも簡単だが典型的な例として、半径 R の断熱壁で囲まれた領域の内部にあるガスを考える。その系は N 個の（ガス）粒子（要素）からなっているが、粒子は原子・分子レベルのものでなくても、1 個の星でも 1 個の銀河でもよい。それに応じて、全系は星であったり、星団であったり、銀河団であったりする。ここでは記述を簡単にするために、それらを単にガスと呼ぶことにする。この系は力学的に釣り合っていて、かつ熱平衡状態にあるとする。

ガスが熱平衡状態にあることは、その中に熱の流れがないということで、ガスの温度分布は一様でなければならない。つまり等温のガス球である。そこでこのガス球の熱平衡状態が安定に存在し続けるかどうかを調べることができる。例えば摂動によって中心部の熱が外に流れて中心部の温度が下がったとする。その結果、中心部の圧力が下がり、力学的釣り合いが破れ、それを回復するために中心

部が重力で収縮する。この（断熱）収縮によって中心部の温度が上がる。そこで最初に熱が流れ出たことによる温度の下がりと、それに続く収縮による温度の上がりのどちらが大きいかを問うことができる。それに対する答えは、ガス球の中でどれくらいの密度コントラストがあったか、すなわち重力の効果がどのくらいであったかによって決まる（重力の効果が無視できるときは、密度は一様であり、そのときはふつうの熱力学における問題に還元される）。

最初の平衡状態において球の中心の密度を $\rho_{\rm c}$ とし、壁のすぐ内側の密度を $\rho_{\rm b}$ として、密度コントラストを $D = \rho_{\rm c}/\rho_{\rm b}$ で表わすことにしよう。その系の摂動に対するレスポンスは線形安定性理論を使って調べることが出来る。結果は、D が 709 より小さいときには普通の熱力学（$D = 1$ の場合）と同じで、系は安定に存続する。しかし D が 709 より大きいと、圧縮による温度上昇の効果のほうが大きいことがわかる。すなわち、中心部の熱が外へ流れ出ると、中心部の温度はかえって上昇する。そして熱がますます中心部から外へ向かって流れるようになる。すなわちそういう系は熱力学的に不安定なのである。

この不安定現象は中心密度が無限大になるまで進行するので、重力熱力学的カタストロフィーと呼ばれる。その状況は系のエントロピーという言葉を使っても表現できる。最初の等温状態は熱平衡状態であったが、系のエントロピーは極小値という不安定平衡状態にあった。そこで熱伝導という不可逆過程が起こると、それに伴ってエントロピーが生成される。そして系のエントロピーはどこまでも増大していく。（一般相対性理論によるブラックホールを考えない限り）この系にはエントロピー極大の状態はない。そこでこの不安定はどこまでも続くのである。詳しく述べる紙数がないので省略するが、そのような不安定平衡状態になっていることには、系のエネルギーが超外延量になっていることが本質的に効いている。

このような不安定の結果、系の中心部に密度が集中するということが起こる。これは構造ないしは形態が形成されていくというプロセスに相当する。系が進化して構造が自発的に発生すると言ってもよい。それは密度の高いコアと密度の低いハローを成長させる差別化 (segregation) の進行である。実際の天体現象で例をあげてみよう。ガスから生まれた星は熱エネルギーを星の外に捨てるにしたがって、星の内部の温度がしだいに上がり、中心部のエントロピーはしだいに減少する。そして星は主系列星に落ちつく。中心部で水素が核反応をしてヘリウムになる段階である。星の中心部で水素が消費されてヘリウムになってしまうと、ヘリウム中心核は熱エネルギーを捨てながらさらに収縮して温度を上げる。そして星にもとからあった水素の多い外層は逆に膨張して、密度コントラストはますます大きくなる。こうして中心部に密度が集中したコアと外部に広がった外層（ハロー）からなるという、赤色巨星になる。典型的なコア・ハロー構造である。

このコア・ハロー構造は何も密度分布や温度分布に限ったことではない。このような差別化 (segregation) は、他の物理量に対しても起こる。例えば太陽系の形成過程は、ほとんどの質量が太陽に、ほとんどの角運動量が惑星系に移るという、差別化が進行する過程なのである。そのような差別化は、自己重力が本質的な役

割を果たす天体現象ではありふれた事柄であり、そのことが、天体の進化を引き起こす原動力になっているのである。宇宙については、グローバルな宇宙のことが話題になることが多いが、そこでは天体に見られるほどの顕著な形態形成はあまり見られない。天体でそれが重要な過程になっているのは、天体が非線形・非平衡（有限な速さでのエントロピー生成）・開放系（無限大の密度コントラストと、生成されたエントロピーの捨て場としての外部の空間 − エントロピーは光子としても捨てられる）になっているからである。

重力がこのように面白い現象を引き起こすのは、固体物理などで言われるロングレンジ・オーダーの典型的な場合だからである。この意味では、重力の現象だけが特異なものではない。しかし重力の現象は、相互作用が万有引力の法則という非常に単純な数学的形式をもつものであり、そのために数値的にではあるが、きちんと定量的に扱えるという特徴をもっている。そこでそれらについての理解を進めておけば、他の実効的に遠距離的な相互作用をもつシステム、ロングレンジ・オーダーの問題や、非線形・非平衡・開放系の振舞いについて、われわれが理解を進めていくための足しになりそうである。自然界に生起する現象の多くはそのようなものである。それらは線形代数という格好の良い方法では扱えないので、残された問題になっている。21 世紀の科学はそのような問題が中心にならなければならない。もっとも、これは私の個人的な感想にすぎないので、反論があれば大いに展開してほしい。

(数理科学 2000 年 5 月号から集録)

[杉本大一郎]

第2章　講義1　重力多体系の物理の基礎

2.1　はじめに

重力多体系の物理の基礎ということで、ここでは以下のような話題について簡単に触れることにしたい。

- 支配方程式 (運動方程式, ボルツマン方程式)
- ビリアル定理, 力学平衡モデル
- 2 体散乱の効果 (緩和, 力学的摩擦)

2.2　無衝突ボルツマン方程式

平衡状態は何かとかいう議論をする前に、支配方程式を決めないと話にならない。そこで、無衝突ボルツマン方程式を導入しておく。いま、粒子数が無限に多い極限を考えると、位相空間での（一体）分布関数 $f(\boldsymbol{x}, \boldsymbol{v})$ は以下の無衝突ボルツマン方程式に従う。

$$\frac{\partial f}{\partial t} + \boldsymbol{v} \cdot \nabla f - \nabla \Phi \cdot \frac{\partial f}{\partial \boldsymbol{v}} = 0, \tag{2.1}$$

ここで Φ は重力ポテンシャルであり以下のポアソン方程式の解として与えられる。

$$\nabla^2 \phi = -4\pi G \rho. \tag{2.2}$$

ここで、 G は重力定数であり、 ρ は空間での質量密度

$$\rho = m \int d\boldsymbol{v} f, \tag{2.3}$$

である。もちろん、これは、重力多体系の運動方程式

$$\frac{d^2 \boldsymbol{x}_i}{dt^2} = \sum_{j \neq i, 1 \leq j \leq N} G m_j \frac{\boldsymbol{x}_j - \boldsymbol{x}_i}{|\boldsymbol{x}_j - \boldsymbol{x}_i|^3}, \tag{2.4}$$

で、質量密度 $\rho = nm$ (ここで n は粒子の数密度) を一定に保って $m \to 0$ （従って $N \to \infty$）の極限をとったものである。なお、ここで $\boldsymbol{x}_i$ と m_i は粒子 i の位置と質量である。

式は通常のボルツマン方程式と同じであるが、特徴的なのは衝突項がないことである。後で説明するが、自己重力多体系の場合、粒子数無限大の極限では重力場が滑らかになって衝突項が消える。

まず、力学平衡という概念を導入しておこう。これは、(衝突項を無視するという近似のもとで)、分布関数が時間的に定常であるということである。衝突項がないので、一体分布関数に対してリュイビルの定理が成り立つ。従って、ラグランジュ的に見て分布関数の値（位相密度）は一定であり、ボルツマンエントロピーは、少なくとも形式的には保存するということに注意して欲しい。つまり、今考えている粒子数無限大の極限では、通常の意味での熱力学的緩和、つまり熱平衡に向かっての進化は起きない。

で、力学平衡に話を戻す。分布関数が時間的に定常とは、もちろんオイラー的に変化しないということである。このための必要十分条件を与えるのが以下のジーンズの定理である。

ジーンズの定理： 任意の無衝突ボルツマン方程式の定常解は、運動の積分を通してのみ位相空間座標に依存する。逆に、任意の運動の積分の関数は定常解を与える。

運動の積分の定義も一応与えておく。ポテンシャル Φ のもとで、ある $\boldsymbol{x}, \boldsymbol{v}$ の関数 I が運動の積分であるとは、その上で

$$\frac{d}{dt} I(\boldsymbol{x}, \boldsymbol{v}) = 0, \tag{2.5}$$

がなり立つことである。つまり、実際にすべての粒子の軌道について、その上でその量が変化しないということである。ちょっと変形すれば

$$\boldsymbol{v} \cdot \nabla I - \nabla \Phi \cdot \frac{\partial I}{\partial \boldsymbol{v}} = 0 \tag{2.6}$$

これと、上の無衝突ボルツマン方程式を比べてみると、すぐわかるように時間微分が落ちているだけである。

なお、「運動の積分」というときの流儀は 2 通りあって、一般に運動の保存量のことを「運動の積分」ということもあるが、ここでは位相空間の座標だけの関数であって同時に保存量であるものをさす。具体的には、たとえば 1 次元調和振動子で「初期の位相」というのは保存量だが運動の積分ではない。これは、時間が入ってくるからである。これに対し、エネルギー $1/v^2 + \Phi$ や、ポテンシャルが球対称（r だけの関数）の場合の角運動量ベクトル $\mathbf{L} = \boldsymbol{r} \times \boldsymbol{v}$ は運動の積分である。

以下、定理の証明を一応書いておく。ジーンズの定理をいいかえると、分布関数 f が定常であるためには、運動の積分 $I_1, I_2, ..., I_m$ があって $f = f(I_1, I_2, ..., I_m)$ の形で書けることが必要十分ということになる。

証明だが、まず「定常ならば運動の積分で書ける」というほうを考えてみる。これは、f 自体が運動の積分の定義を満たしているので、 OK。

逆のほうは、実際に f の全微分を I_k で書き下せば、それぞれが 0 になるということからいえる。

というわけで、これはなかなか強力な定理だが、一般の場合にはそれほど役に立つわけではない。というのは、ポテンシャルを与えた時に一般に運動の積分というのは 5 個あるはずだが、それらをすべて知っているということは普通はないからである。例えば、ポテンシャルに球対称とか軸対称とかいった対称性がない時には、一般にはエネルギー以外の保存量があるとは限らない。分布に球対称とかいろんな制限を付ければある程度なんとかなることになる。例えば、球対称にすれば、エネルギーの他に角運動量ベクトルの 3 成分が保存する。もう一つ保存量があるが、これは調和振動子やケプラー運動のような軌道が閉じる場合にしか意味がないので、実効的には上の 4 つが保存量である。ここで、ポテンシャルに対応して分布関数も球対称であるとすると、分布関数がエネルギー E と全角運動量 J にだけ依存するということになる。

軸対称ポテンシャルの場合には、対称軸回りの角運動量 J_z は保存量になる。それ以外には保存量がないと信じると、分布関数が E と J_z で書けるということになる。但し、一般には他に保存量がないかどうかは良くわかっていない。（これには第三積分問題という名前がある）。

ジーンズの定理は、重力多体系の特徴のうち無衝突系（に近いもの）であることから来るものを端的に表現している。すなわち、通常の流体ならば力学平衡に対応する概念は静水圧平衡である。静水圧平衡であって熱平衡ではない系を我々が扱う時の通常の仮定は局所熱平衡、つまり、十分小さなスケールで見れば熱平衡が成り立っているという仮定である。この仮定により、例えば局所的な温度というものを考えたり、熱とかその輸送を考えたりできるようになる。

ところが、重力多体系の力学平衡では与えられるのは系のどこでも共通である分布関数であり、これは熱平衡でなければもちろん熱平衡分布関数とは違っている。つまり、分布関数が本質的に局所的ではないために、熱平衡でない系では必ず局所的にみても分布関数が熱平衡からずれていて、従って局所的な温度等の熱力学的な量は自明には定義できないということになる。

もちろん、だからといって系が熱平衡に向かわないとかそういうことは（残念ながら）起こらない。が、その進化は、通常の系のように温度差を打ち消すように熱が流れるといった形では（少なくとも原理的には）表現できない。

では、そのような熱平衡に向かう進化はどのようにしておきるか？ということが問題だが、それを理解する枠組が「２体緩和」ということになる。これは後でもうちょっと詳しく話をするとして、無衝突の範囲でいくつかの話題をまとめておく。

2.3 ビリアル定理

定常状態のボルツマン方程式から、速度のモーメントをとったり空間座標のモーメントをとったりしてがちゃがちゃ計算すると、以下のビリアル定理が出てくる。

$$2K + W = 0 \tag{2.7}$$

ここで、 K は系の全運動エネルギー、つまり、空間各点での $v^2/2$ の平均に質量密度をかけて全空間で積分したものである。W は全ポテンシャルエネルギー

$$W = -\frac{1}{2}\int\int \rho(\boldsymbol{x})\rho(\boldsymbol{x}')\frac{1}{|\boldsymbol{x}-\boldsymbol{x}'|}d^3\boldsymbol{x}d^3\boldsymbol{x}' \tag{2.8}$$

である。

今、系の全エネルギーを E とすれば、 $E = K + W$ であるから、

$$E = -K = W/2 \tag{2.9}$$

ということになる。つまり、定常状態にある自己重力恒星系では、必ず全エネルギーはポテンシャルエネルギーのちょうど半分であり、絶対値が運動エネルギーに等しい。これは球対称とかそういう仮定なしに常に正しい。このために、系のなかでなにか散逸があったり、あるいは系の外にエネルギーを与えたりすると、全エネルギーの絶対値が大きくなり、そのためにポテンシャルと運動エネルギーの両方がかならずその分大きくなる。

ここで運動エネルギーといっているものは多くの場合に恒星のランダム運動、つまり熱運動のエネルギーであり、それが大きくなるということは大雑把にいって温度が上がるということである。つまり、自己重力系には、基本的にエネルギーを失うほど温度が上がるという性質、すなわち「負の比熱」があるわけである。

なお、式 (2.8) から、系の全質量を M とした時に

$$W = \frac{GM^2}{2R_{\rm vir}} \tag{2.10}$$

となるような長さの次元をもつ量 $R_{\rm vir}$ を定義出来る。これをビリアル半径と呼ぶ。$R_{\rm vir}$ は重力ポテンシャルから見た系の特徴的な大きさを与える。

2.4 力学平衡モデル

さて、N 体計算の結果を理解するにも、また単に初期条件を準備するだけのためにも、力学平衡にある恒星系モデルについてある程度知っていることは重要である。というわけで、以下球対称の場合に限って話をする。

2.4.1 球対称の場合

球対称の場合、運動の積分はエネルギーと角運動量の 3 成分で 4 つある。一般にはもう一つあるが、これは特別な場合を除いてあまり意味がないので、定常な分布関数はエネルギーと角運動量だけで書けると思っていい。

いちおう、ここで、意味がないというのはどういうことかということを説明しておく。そのためには、意味がある特別な場合というのを考えるのがよい。これは、ケプラー軌道のような、軌道が閉じる場合である。この時には、エネルギーと角運動量の他に、軌道全体の向きを表す量（近点経度）が保存する。これはちゃんと保存量になっている。

しかし、一般には軌道が閉じない。このときでも、近点経度に対応するような保存量が実は存在しているが、それにも関わらず、ある軌道がエネルギーと角運動量で決まる部分空間を覆ってしまう（数学的には、もちろんすべての点を覆えるのではなく、任意の点について、いくらでも近くにいけるというだけだが)。こうなっていると、その積分に分布関数が依存すると、連続性とか微分可能性とかに困難を生じることになる。

さて、f は E と $\mathbf{J}$ によるということにしたわけだが、いま球対称な場合ということなので $\mathbf{J}$ の方向にではなく、絶対値だけに依存するのでないといけない。したがって、実は球対称の分布関数は一般に $f(E, J)$ と書けるということになる。

我々が扱いたいのは自己重力系なので、実際にポアソン方程式を球対称の場合に書き下してみると

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\Phi}{dr}\right) = 4\pi G\int f\left(\frac{1}{2}v^2+\Phi, |\boldsymbol{r}\times\boldsymbol{v}|\right)d\boldsymbol{v}, \tag{2.11}$$

てな感じになる。

2.4.2 $f(E)$ の場合

上の場合でもまだちょっと大変なので、さらに単純化してとりあえず J にもよらない場合というのを考えてみる。これには、なかなか特別な、空間上の各点で速度分散が等方的であるという性質がある。これはどういうことかというと、一般にある方向の速度分散というのは

$$< v_e^2 >= \frac{1}{\rho}\int v_e^2 f(v^2/2+\Phi)d\boldsymbol{v} \tag{2.12}$$

となるが、f が v の絶対値にしかよらないので、v_e の方向にこの積分はよらない。まあ、速度分散がとかいうより、速度分布自体が等方的なのだから当然ではある。

以下、扱いやすくするために変数をとり直す。

$$\Psi = -\Phi+\Phi_0, \qquad \mathcal{E} = -E+\Phi_0 = \Psi - v^2/2 \tag{2.13}$$

ここで Φ_0 は定数で、普通は $\mathcal{E} > 0$ で $f > 0, \mathcal{E} \leq$ で $f = 0$ となるようにとる。

これらを使って、さらに v の角度方向に渡って積分すれば

$$\begin{aligned} \frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\Psi}{dr}\right) &= -16\pi^2 G\int_0^{\sqrt{2\Psi}} f(\Psi - \frac{1}{2}v^2)v^2 d\boldsymbol{v} \\ &= -16\pi^2 G\int_0^{\Psi} f(\mathcal{E})\sqrt{2(\Psi-\mathcal{E})}d\mathcal{E}. \end{aligned} \tag{2.14}$$

これで、一般に f を与えて Ψ を求めるとか、あるいはその逆とかが出来る。

ただし、 Ψ を与えて f を求めようってときには、求まった f が $f \geq 0$ の条件を満たすという保証はないので、そういうのは物理的には意味がない解ということになる。

2.4.3 球対称な分布関数の例

ここであげるのはあくまでも例であるが、さまざまな理由からその性質がよく調べられているものである。

ポリトロープとプラマーモデル

ある意味でもっとも簡単な分布関数の例は、 $\mathcal{E}$ の冪乗（パワー）で書けるものである。例えば

$$f(\mathcal{E}) = \begin{cases} F\mathcal{E}^{n-3/2} & (\mathcal{E} > 0) \\ 0 & \text{otherwize} \end{cases} \tag{2.15}$$

これから、まず密度を Ψ の関数として求められる（$v^2 = 2\Psi\cos\theta$ なる変数変換のあと）。で、答えは

$$\rho = c_n\Psi^n \quad (\Psi > 0) \tag{2.16}$$

となる。ただし、 c_n が有限になるためには $n > 1/2$ でないといけない。

上を使ってポアソン方程式から ρ を消去すると

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\Psi}{dr}\right) + 4\pi G c_n\Psi^n = 0 \tag{2.17}$$

変数を適当にスケーリングして

$$\frac{1}{s^2}\frac{d}{ds}\left(s^2\frac{d\psi}{ds}\right) + \psi^n = 0 \tag{2.18}$$

としたものを Lane-Emden 方程式と呼ぶ。

実際には、上の Lane-Emden 方程式を解かないとポテンシャルや密度がどうなっているかはよくわからない。で、一般の n ではこの方程式には初等的な解はないが、 $n = 5$ の場合には解があることが古くから知られている。これは

$$\phi = \frac{1}{\sqrt{1 + \frac{1}{3}s^2}} \tag{2.19}$$

の形をしている。これが Lane-Emden 方程式を満たしていることは各自確かめること。さらに、密度は $c_5\phi^5$ で与えられることになる。これをプラマーモデルという。

密度が $r = 0$ で有限で、$r \to \infty$ で $1/r^3$ より速く落ちるので、質量は有限である。

これは、天文学的になにか素晴らしいものであるというわけではないが、球状星団のうち中心密度が低いものにはまあまあ似ていなくもない。とりあえず、これの意味は、解析関数で簡単に書ける自己重力系の self-consistent なモデルであるということである。

プラマーモデルは、いろんなシミュレーションの初期条件として使われることが多い。

Hernquist Model

プラマーモデルはその存在が前世紀から知られているが、こちらは論文が発表されたのが 1990 年（というわけで、 Binney & Tremaine のときにはまだ知られていなかった）という、非常に新しいモデルである (Hernquist, L., 1990, ApJ 356, 359)。これは、ポテンシャルを

$$\Phi = -\frac{1}{r + a} \tag{2.20}$$

で与える。密度分布は

$$\rho = C\frac{a^4}{r(r + a)^3} \tag{2.21}$$

で書ける。分布関数は求まっているが、めんどくさいのでここには書かない。とりあえず、密度とポテンシャルがコンシステントになっていることは確認してみよう。なお、一般に球対称ならば

$$\frac{d\Phi}{dr} = GM_r/r^2 \tag{2.22}$$

であることに注意。これは、単に半径 r のところでの重力加速度である。

Hernquist Model には、「$r^{1/4}$ 則をかなり良く再現する」という著しい特徴がある。

$r^{1/4}$ 則とは要するに、観測される楕円銀河の表面輝度の対数（要するに等級ですね）が、半径の 1/4 乗に対して直線にのって見えるというものである。これはまあちょっと物理的には変な話で、ある意味「たまたま」みたいなところがあるが、

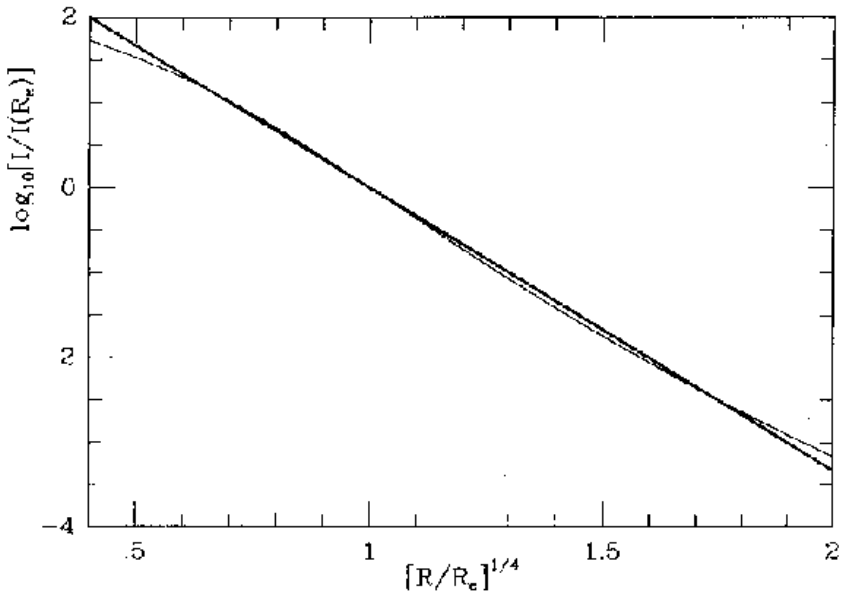


FIG. 4. Surface brightness profiles for the $R^{1/4}$ law (*thick curve*) and the present model (*thin curve*) as a function of $(R/R_e)^{1/4}$. Surface brightness is normalized to its value at R_e where R_e refers separately to the effective radii of the two models.

とにかく観測の人は結果をこれで書くことが多い。この性質と、一応解析関数で分布関数が書けるということのために、楕円銀河やダークハローのモデルとして広く使われるようになってきている。

ただし、このモデルにはいくつか妙な性質もある。数値計算という観点から重要なのは、このモデルでは中心に向かって速度分散が下がっていくことである。このために、あとで述べる 2 体緩和の結果中心部の星が加熱され、割合短い時間で構造が変化する。このため、実際に使う時には注意が必要である。

等温モデル

熱平衡状態では（古典統計なので）分布関数はマックスウェル–ボルツマン分布、すなわち

$$f(\mathcal{E}) = \frac{\rho_1}{(2\pi\sigma^2)^{3/2}} e^{\mathcal{E}/\sigma^2} = \frac{\rho_1}{(2\pi\sigma^2)^{3/2}} \exp\left(\frac{\Psi - v^2/2}{\sigma^2}\right) \tag{2.23}$$

で与えられなければならない。まず、例によってこれを速度空間で積分して密度をポテンシャルの関数として表す。この時に誤差関数についての

$$\frac{2}{\sqrt{\pi}} \int_0^\infty e^{-x^2} dx = 1 \tag{2.24}$$

を使うと、

$$\rho = \rho_1 e^{\Psi/\sigma^2} \tag{2.25}$$

ポアソン方程式にこれを入れると

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\Psi}{dr}\right) = -4\pi G\rho \tag{2.26}$$

従って、

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\log\rho}{dr}\right) = -4\pi G\sigma^2\rho \tag{2.27}$$

後はこれを数値的に解くわけだが、まず、一つ特別な解があるということを指摘しておく

$$\rho = \frac{\sigma^2}{2\pi G r^2} \tag{2.28}$$

は、上の方程式を満たし、解の一つとなっている。これを singular isothermal sphere と呼ぶ。これは self consistent なモデルではない。というのは、質量が $M_r \propto r$ となって有限ではないからである。が、例えば銀河ハローの中心部、あるいは楕円銀河についても中心部についてはこれで比較的良く近似できるものもあるということがわかっている。

特に、渦巻銀河については、「回転速度が中心からの距離に（あまり）依存しない」（いわゆる flat rotation curve）という性質が知られていて、これを説明するためには上のような $\rho \sim 1/r^2$ のダークハローが必要であるということになっている。

特別ではない解は、中心密度を有限にして中心から外側に向かって解いていけばいい。この時でも、 $r \to \infty$ の極限では singular isothermal に近付く。

流体との関係

等温モデルは、エントロピー極大であり、分布関数がボルツマン分布になっているという特別な性質がある。このため、等温ガス球と実は同じ構造をとる。以下、ガス球について方程式を導いておく。静水圧平衡の式は

$$\frac{dP}{dr} = -\rho\frac{GM_r}{r^2} \tag{2.29}$$

である。状態方程式に等温の

$$P = \frac{k_B T}{m}\rho \tag{2.30}$$

を使って P を消して、さらに M_r を微分してみれば、係数を別にして

$$C\frac{d}{dr}\left(r^2\frac{d\log\rho}{dr}\right) = -4\pi G\rho r^2 \tag{2.31}$$

要するに、 stellar system とガスで同じ方程式になっている。

なお、ポリトロープでも、ポリトロピックな状態方程式を持つガス球の密度分布と stellar system のそれとは一致する。が、等温モデルの場合とは実は本質的な

違いがある。等温モデルの場合は、分布関数そのものが一致する（ボルツマン分布であり、局所的にも大局的にもエントロピー最大）が、一般のポリトロープではそんなことはない（そもそもガス球ではジーンズの定理が成り立たないし、局所的にはボルツマン分布であるから)。

King Model

等温モデルは、すでに述べたように熱平衡（エントロピーの変分が 0 ）という重要な意味を持つ定常解ではあるが、なにしろ質量が無限大であり現実に存在しないのでちょっと困るところがある。なにか適当な仮定を置くことで、「おおむね等温モデルであり、なおかつ有限の大きさをもつ」というものを考えることはできないだろうか？

$$f(\mathcal{E}) = \frac{\rho_1}{(2\pi\sigma^2)^{3/2}} e^{\mathcal{E}/\sigma^2} = \frac{\rho_1}{(2\pi\sigma^2)^{3/2}} \exp\left(\frac{\Psi - v^2/2}{\sigma^2}\right) \tag{2.32}$$

上の分布関数で、質量が発散する理由は何かを思い出してみよう。その本質的な理由は、分布関数がエネルギー無限大 ($\mathcal{E} \to -\infty$) まで 0 にならないことにある。

有限の質量のものが自己重力でまとまっているためには、すべての粒子のエネルギーが負でないといけないので、これでは自己重力系が表現出来ないのはある意味では当然のことといえる。

それならば、ある有限のエネルギー以上のものはないことにしてしまえばいい。そのやり方にはいろいろあり得るが、とりあえず Lowered Maxwellian と呼ばれる以下のようなものを考える

$$f(\mathcal{E}) = \begin{cases} \frac{\rho_1}{(2\pi\sigma^2)^{3/2}}(e^{\mathcal{E}/\sigma^2} - 1) & (\mathcal{E} > 0) \\ 0 & (\mathcal{E} \leq 0) \end{cases} \tag{2.33}$$

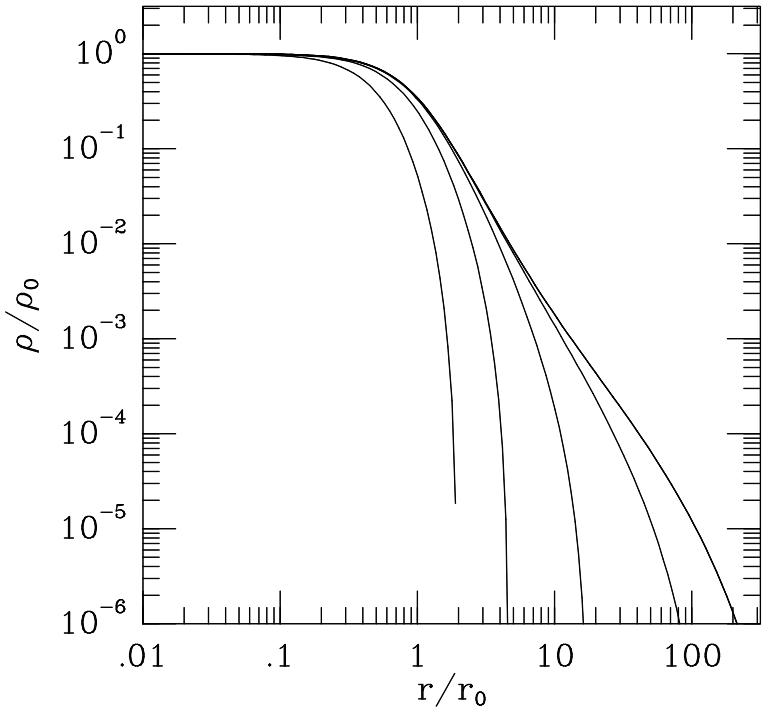
これは $\mathcal{E} = 0$ で $f = 0$ となるように、 1 を引いたというだけである。これしか方法がないというわけではないが、これは扱いやすいこともあってもっともよく使われている。

例によって、まず速度空間で積分すれば

$$\begin{aligned} \rho &= \frac{4\pi\rho_1}{(2\pi\sigma^2)^{3/2}} \int_0^{\sqrt{2\Psi}} \left[\exp\left(\frac{\Psi - v^2/2}{\sigma^2}\right) - 1\right] v^2 dv \\ &= \rho_1 \left[e^{\Psi/\sigma^2} \mathrm{erf}\left(\sqrt{\frac{\Psi}{\sigma}}\right) - \sqrt{\frac{4\Psi}{\pi\sigma^2}}\left(1 + \frac{2\Psi}{3\sigma^2}\right)\right] \end{aligned} \tag{2.34}$$

ここで erf は誤差関数で、積分が有限区間であるために出てくる。最後の項は 1 引いている分の寄与である。ちなみに

$$\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt. \tag{2.35}$$



これでポテンシャルの関数として密度が求まったので、あとはポアソン方程式に入れて数値的に解くだけである。ただし、 King model の場合境界条件についてすこしきちんと考える必要がある。

半径方向の分布は、中心から無限遠まで与えられるわけであるが、実は外側の境界をどうとるべきかはちょっと自明ではないので、とりあえず中心から初期値問題として解くことを考える。

初期条件としては、まず $d\Psi/dr = 0$ とする、すなわち、中心密度が有限の解を考える。 Ψ の原点での値 Ψ_0 は任意に選べるので、これの値によっていろいろな解がでてくる。

これは実際に数値的に解いてみたものの例である。速く落ちるものから、Ψ_0 が 1, 3, 6, 9, 12 と変えてみてある。

なお、横軸のスケールの r_0 は、

$$r_0 = \sqrt{\frac{9\sigma^2}{4\pi G\rho_0}} \tag{2.36}$$

として無次元化するのに使っている。これは、いわゆる「コア半径」というのとそこそこ一致するということになっている。通常、キングモデルのコア半径というときにはこれをさす。観測的には、中心の表面輝度の 1/2 になるところとするのが普通である。

グラフからわかるように、有限の半径 r_t で ρ は 0 になる。これは、解いていったときに Ψ が 0 になってしまうためである。この半径のことを King model の

tidal radius 潮汐半径という。このモデルのばあい、Ψ と本当のポテンシャル Φ の間に以下のような簡単な関係が成り立つことに注意。

$$\Phi = -\frac{GM}{r_t} - \Psi \tag{2.37}$$

ここで M は系の全質量である。

King Model は、球状星団のプロファイルのモデルとして非常によく使われている。なお、 $c = \log(r_t/r_0)$ のことを concentration parameter といって、観測データにキングモデルを合わせた論文では普通これがパラメータになる。理論計算では Ψ_0 が使われるので、ちょっとややこしいことが多い。

2.5 2体緩和とはなにか？

まず、2体緩和とはいったいどういうものかというところから話を始めることにする。原理的には、これがなにかというのは結構厄介な問題である。

有限粒子数の自己重力多体系を考えると、これは以下のような進化をすると考えられる。まず、最初は力学平衡になかったとすると、とりあえず力学平衡に落ちつく。粒子数が無限大であれば、無限に細かく見れば無限に時間がたっても真の力学平衡に到達するわけではないが、まあ、漸近はしていく。この時、各粒子は与えられたポテンシャルの中を運動するだけになり、それ以上進化することはなくなる。

さて、実際には有限粒子数であるので、そもそも真の力学平衡というものはない。有限の質量をもった各粒子が系の中を運動するに従って、ポテンシャルは必ず変化するからである。この変化によって各粒子の軌道も変化することになる。

それでは、粒子の軌道の変化を、粒子数が有限であることから来る成分とそれ以外に分離することは可能であろうか？系が力学平衡にあるとみなすことができればそれは可能である。つまり、力学平衡にあれば、粒子のエネルギー変化は定義によりすべて粒子数が有限であることによるからである。

が、良く考えると問題なのは、そもそも有限粒子数であるものを力学平衡とみなすとはどういうことかということである。このあたりを考えていると段々混乱してくるので、以下、理想化された状況から順番に考えていくことにしよう。なお、この節の内容は、 L. Spitzer の Dynamical Evolution of Globular Clusters にほぼ沿っている。2 体緩和に関しては、重力多体系での議論とプラズマ物理での議論は全くパラレルなものであり、上記の本の内容自体、同じ著者の Physics of Fully Ionized Gases のものとほとんど同じである。

あ、で、なぜそういうものの話をするかというわけだが、これは、N 体シミュレーションにおいて 2 体緩和は

- 系を物理的に進化させる主な要因である

か、または

- 数値誤差のおもな要因である

からである。

なお、2 体緩和が系の進化になんらかの役割を果たす系のことを普通衝突系という。そうでないのが無衝突系である。原理的には無衝突系であるためには粒子数が無限大でないといけないが、まあ、非常に大きいとか、考えている時間が短いとかで無衝突系とみなせることがある。が、そのような系を有限粒子数（大抵実際の系よりもずっと少ない）でシミュレーションすると人工的な 2 体緩和が入ってくる。これが数値誤差のおもな要因となることが多い。

2.5.1 一様等方な分布

理想化といえば一様等方な分布を仮定することである。例えばマックスウェル分布があって、その中の一つの粒子をとって考えるということをしたいわけだが、これは結構厄介なのでさらに簡単な例を考える。すなわち、速度 0 で空間内に一様（ランダム）に分布した質点を考え、その中を質量 0 のテスト粒子を飛ばして見る。

もちろん、この場合エネルギー交換はないので速度は変わらず、単に散乱されるだけだが、しかし、この例は 2 体緩和のいくつかの重要な性質を示すのですこし詳しく見ていくことにする。分布している質点の質量を m、数密度を n とする。図 2.1 のように、テスト粒子が一つの粒子から距離（インパクトパラメータ）b を速度 v で通った時に曲がる角度 θ は、実際にケプラー問題の解析解を使って

$$\begin{aligned} \tan\theta &= \frac{2b}{(b/b_0)^2-1} \\ b_0 &= \frac{Gm}{v^2} \end{aligned} \tag{2.38}$$

で与えられる。単位時間当たり、インパクトパラメータが $(b, b+db)$ の範囲にある散乱の回数は $2\pi nvbdb$ である。

さて、散乱の方向はランダムであると思われるので、平均としては（一次の項は）0 になる。しかし、 2 次の項は 0 にならない。これは

$$< \Delta\theta^2 >= 2\pi nv \int_0^{b_{max}} \delta\theta^2 bdb \tag{2.39}$$

で与えられることになる。

この式から既にいろいろな性質がわかる。が、その前に理論的な困難を解決しておく必要があるであろう。すなわち、この積分は $b \to \infty$ で発散しているのである。これについてはいくつかの考え方があった。例えば、初めて 2 体緩和の性質を理論的に調べたチャンドラセカールは、以下のように考えた。

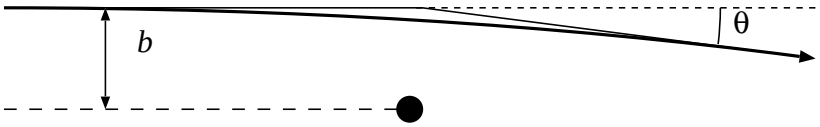


図 2.1: 2 体散乱

「平均粒子間距離よりもインパクトパラメータが大きいような散乱は、多体の干渉によって効かなくなるのでそこで積分を打ち切ってよい」

しかし、多体の干渉というようなものが実際にあるかどうかはあきらかではない。もっと素直な解釈は、実際に系にあるすべての粒子と常に同時に相互作用しているのだから、システムサイズくらいまで全部いれる（系が構造を持つ場合はちょっとややこしいが、密度の空間依存も積分のなかに入れて全空間で積分する）というものである。

数値実験の結果などから、後者の解釈すなわち全体が効くというほうが正しいということはかなり昔から大体わかっていた。歴史的には、どちらの解釈が正しいかについてはかなり最近まで論争があって、完全に決着がついたといえるのは 1994-5 年頃である。が、現在では後者の解釈が正しいということに疑いの余地はない。

式 (2.39) から、適当に近似すると

$$< \Delta\theta^2 > \sim Gnv^{-3}m^2 \log(R/r_0) \tag{2.40}$$

となる。ここで R は先に述べたシステムの大きさ、 r_0 は「大きく曲がる」ためのインパクトパラメータの値で、$b_0 = Gm/v^2$ の程度である。

さて、これからどんなことがわかるかというわけだが、これから、逆に角度変化が 1 の程度になる時間というのを求めてみると、

$$t_\theta \sim \frac{v^3}{G^2 nm^2 \log\Lambda} \tag{2.41}$$

となる。ここで Λ は上の R/r_0 を単に書き換えただけである。

今、$\log\Lambda$ の質量依存性といったものを無視すると、散乱のタイムスケールは速度の 3 乗、数密度の逆数、質量の 2 乗の逆数に比例するということがわかったことになる。特に、質量密度一定の場合というものを考えてみると、タイムスケールが各粒子の質量に比例するということがわかる。

ある大きさを持った多体系というものを考えてみよう。質量 M、ビリアル半径 R、粒子数 N とすれば、ビリアル定理から $< v^2 > /2 = GM/R$、力学的なタイムスケール、つまり典型的な速度を持つ粒子が系を横断するのにかかる時間が $t_d = R/v \sim \sqrt{R^3/GM}$ となる。これを使うと上の緩和のタイムスケールは

$$t_\theta \sim \frac{N}{\log N} t_d \tag{2.42}$$

となる。つまり、力学的なタイムスケールに比べて、2 体緩和のタイムスケールはほぼ N 倍長いということになる。このために粒子数が大きいほど無衝突系に近付くわけである。

2.5.2 どういうことを考えるかということ：流体との違い

2 体緩和によって、最終的には系が熱力学的に進化するわけであるが、これが普通の流体（ガス）とは本質的に違うものであるということをここで再確認しておこう。

ガスの場合、粒子の平均自由行程はシステムサイズよりもはるかに小さい。液体であれば平均粒子間距離は粒子のサイズ程度であるし、気体であっても通常の状況では考えている現象の空間スケールに比べて平均自由行程は小さい。ちなみに、非常に希薄な気体とか、あるいは本当に空間スケールの小さい現象では平均自由行程が問題になる。これは例えば超高層での人工衛星の回りの気体の流れとか、あるいは最近の磁気ヘッドの回りの空気の流れとかいったものである。

とにかく、通常のガスの場合、平均自由行程がシステムサイズより小さい。このために、システムサイズよりは小さく平均自由行程よりは大きいような空間スケールを考えると、そのなかでほぼ熱平衡になっていると思っていいことになる。いいかえれば、いわゆる Local thermal equilibrium (LTE) の仮定が使える。こうなると、温度とか圧力とかいった量が近似的（といっても実際上非常に高い近似精度で）に定義でき、そういったマクロな量で系の進化を扱う、特に熱の流れを拡散方程式で書くということが可能になる。

しかし、自己重力質点系では状況が全く異なる。まず、粒子数が無限大の極限では、平均自由行程も無限大であった。つまり、 LTE が成り立たないどころか、そもそも熱平衡に向かう（すなわちエントロピーを生成する）ようなメカニズムがなかったわけである。

粒子数が有限の場合も、依然として平均自由行程が長い。つまり、粒子数無限大の時の軌道から、他の粒子との相互作用によって段々ずれていくわけだが、そのずれる典型的なタイムスケールは $Nt_d/\log N$ 程度であった。つまり、流体の場合とは全く逆に、ほとんど衝突はしないで自由運動（他の粒子全体が作るポテンシャルに沿った運動）をしていて、その場が有限の粒子で表現されるための揺らぎがあるので段々軌道が変わっていくということになるわけである。

従って、ローカルな熱平衡を仮定して拡散係数／輸送係数を求めるというのとは逆に、ある一つの粒子が系の中を動き回りながらどういうふうにエネルギー等を変化させていくかという観点で見ていくことになる。

これをすこし別ないい方をすれば、通常の空間のなかでの密度や温度の変化を考える代わりに、また 6 次元位相空間のなかでの分布関数の進化を考えるということに当たる。具体的には、これまで無視してきた「衝突項」というものをちゃんと評価して、どういうものかみてやろうということである。

2.5.3 バックグラウンドの分布のもとでの有限質量のテスト粒子の振舞い

さて、以下ではバックグラウンドの粒子分布のもとでの一つのテスト粒子の振舞いを考える。2.5.1 節と違うのは、バックグラウンドも動いていることと、テスト粒子も有限の質量を持つことである。バックグラウンドの粒子は一様に分布するものとし、ある速度分布に従うとする。さらに、バックグラウンドの粒子間の相互作用とかは考えないことにする。これで本当にいいかどうかはちょっと良くわからない問題であるが、まあ、とりあえずやってみることにしよう。

前と同じく、分布している質点の質量を m、数密度を n とする。テスト粒子が一つの粒子から距離（インパクトパラメータ） p を相対速度 $\boldsymbol{V} = \boldsymbol{v}_t - \boldsymbol{v}_f$ で通った時に曲がる角度は、実際にケプラー問題の解析解を使って

$$\begin{aligned} \tan\theta &= \frac{2p}{(p/p_0)^2 - 1} \\ p_0 &= \frac{G(m_t + m_f)}{V^2} \end{aligned} \tag{2.43}$$

で与えられる。ここで、この曲がる角度は相対軌道のものであって、テスト粒子の軌道のものではないということに注意する必要がある。いきなり回りが動いていてしかもテスト粒子が質量をもつというのは面倒になるので、とりあえずテスト粒子は質量を持つが、回りは止まっている場合を考える。この時、一回の散乱での速度変化は以下の式に従う。

$$\Delta v_{\text{垂直}} = \frac{m_f}{m_t + m_f} V \sin\theta = 2V \frac{m_f}{m_t + m_f} \frac{p/p_0}{1 + (p/p_0)^2} \tag{2.44}$$

$$\Delta v_{\text{平行}} = \frac{m_f}{m_t + m_f} V(1 - \cos\theta) = -2V \frac{m_f}{m_t + m_f} \frac{1}{1 + (p/p_0)^2} \tag{2.45}$$

$$\tag{2.46}$$

前の話との違いは、速度変化に係数 $m_f/(m_f + m_t)$ がついていることだけである。これにまた単位時間当たりの衝突回数 $2\pi p n_f V dp$ を掛けて積分するが、$\Delta v_{\text{垂直}}$ に

ついては前と同様 1 次の項は落ちる。それ以外についても前と同様に計算出来て

$$< \Delta v_{\text{垂直}}^2 > = \frac{2n_f\Gamma}{V} \tag{2.47}$$

$$< \Delta v_{\text{平行}} > = -\left(1+\frac{m_t}{m_f}\right)\frac{n_f\Gamma}{V^2} \tag{2.48}$$

$$< \Delta v_{\text{平行}}^2 > = \frac{n_f\Gamma}{V\ln\Lambda} \tag{2.49}$$

ここで Γ は

$$\Gamma = 4\pi G^2 m_f^2 \ln\Lambda \tag{2.50}$$

である。ただし、 leading term でない項は適当に落ちてたりするので注意。

上の式で、$< \Delta v_{\text{垂直}}^2 >$ の項は前に扱った角度の曲がる項と同じものである。前の話と違うのは、ネットに速度が小さくなる成分がある、すなわち $< \Delta v_{\text{平行}} >$ が負で有限の値をもつということである。

これは、 dynamical friction というものである。つまり、回りが止まっているなかを粒子が走っていくと、それが回りを引っ張って動かすので、その分エネルギーを失って段々速度が落ちるわけである。これは、 m が大きい（m_f が小さい）極限では $m_f n_f$ 、つまり質量密度によっていて、バックグラウンドの粒子の質量に依存しないことに注意してほしい。これに対し、他の項は $m_f^2 n_f$ に比例していて、質量密度が同じでも粒子の質量が大きいほうが値が大きくなる。

さて、ここではとりあえず 1 次と 2 次の項を求めたわけだが、それより先の項については考えなくてもいいのだろうか？ここでは粒子の軌道変化がたくさんの散乱のランダムな重ね合わせで書けるとした。この仮定が正しければ、たくさん散乱を受けた後の速度の分布は 1 次と 2 次のモーメントで決まるガウス分布になり、従って 3 次より高いモーメントの寄与は考えなくてもいいことになる。

問題はこの仮定が正しいかどうかであるが、実は理論的にはそれほど厳密に正しいわけではない。というのは、インパクトパラメータが例えば p_0 の程度の散乱というのも現実におき、その効果はそれ以外の散乱すべての寄与に比べてせいぜい $\ln\Lambda$ 程度でしか小さくないからである。まあ、しかし、そんなことをいっていても高次の項があっては計算出来ないし、とりあえず $\ln\Lambda$ 程度で小さいということも確かなので、以下高次のモーメントは考えない。

2.5.4 バックグラウンドが速度分布を持つ場合

いよいよバックグラウンドが動いている場合ということになる。この時でも、相対速度の変化自体は前節に述べたもので正しいが、相対速度にフィールド粒子の速度成分が入ってくるところが違う。

以下、二種類の単位ベクトル系をとって、その上で考える。一つは $(\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3)$ であり、元の空間に固定されている。もう一つは $(\boldsymbol{e}'_1, \boldsymbol{e}'_2, \boldsymbol{e}'_3)$ であり、最初の成分を

相対速度 $\boldsymbol{V}$ に平行にとる。従って、後者は相手の粒子によって違うわけである。この 2 つを考えることで、相対速度の変化をもとの静止系でのテスト粒子の変化に焼き直す。

まず、1 次の項は相対速度に平行な成分だけであった。このことから、ある方向の速度変化は

$$< \Delta v_i >=< \boldsymbol{e}_i \cdot \boldsymbol{e}_1' \Delta v_{\text{平行}} >' \tag{2.51}$$

ということになる。もうすこし精密に書くと、右辺はインパクトパラメータと相手の速度の積分なので、以下のように書けることになる。

$$\begin{aligned} < \Delta v_i > &= \int dp 2\pi p V \int d\boldsymbol{v}_f f(\boldsymbol{v}_f) \Delta v_i & (2.52) \\ &= -\Gamma(1 + m/m_f) \int \frac{f(\boldsymbol{v}_f)}{V^2} (\boldsymbol{e}_i \cdot \boldsymbol{e}_1') d\boldsymbol{v}_f & (2.53) \end{aligned}$$

ここで、$\boldsymbol{v}_f = \boldsymbol{v}_t - \boldsymbol{V}$ はフィールド粒子の速度、 f は速度分布関数である。 p についての積分を先にやったことに注意して欲しい。この積分は $v_f = 0$ であった時の結果をそのまま使っている。

さて、2 次の項についてであるが、前節で見たように $\boldsymbol{V}$ に平行な成分は小さいので、垂直な成分、すなわち $\boldsymbol{e}_2'$ と $\boldsymbol{e}_3'$ の成分を考えればいい。従って、一つの方向からくるフィールド粒子との散乱を考えた時には

$$\begin{aligned} < \Delta v_i \Delta v_j > &= [(\boldsymbol{e}_i \cdot \boldsymbol{e}_2')\boldsymbol{e}_2' + (\boldsymbol{e}_i \cdot \boldsymbol{e}_3')\boldsymbol{e}_3'] \cdot [(\boldsymbol{e}_j \cdot \boldsymbol{e}_2')\boldsymbol{e}_2' + (\boldsymbol{e}_j \cdot \boldsymbol{e}_3')\boldsymbol{e}_3'] < \Delta v_{\text{垂直}}^2 > /2 \\ &= [(\boldsymbol{e}_i \cdot \boldsymbol{e}_2')(\boldsymbol{e}_j \cdot \boldsymbol{e}_2') + (\boldsymbol{e}_i \cdot \boldsymbol{e}_3')(\boldsymbol{e}_j \cdot \boldsymbol{e}_3')] < \Delta v_{\text{垂直}}^2 > /2 \\ &= Q_{ij} < \Delta v_{\text{垂直}}^2 > /2 \end{aligned} \tag{2.54}$$

となる。最後の Q_{ij} は角括弧の中をそう書いただけである。(2.54) 式にまた分布関数を掛けて積分すると、結局

$$< \Delta v_i \Delta v_j >= \Gamma \int \frac{f(\boldsymbol{v}_f)}{V} Q_{ij} d\boldsymbol{v}_f \tag{2.55}$$

ということになる。これで一応必要な 2 次までの係数はすべて書けたわけだが、あまり計算するのに使い易い形ではない。というのは、$\boldsymbol{e}_i'$ とか V とかいったものがまだややこしい形ではいったままであるからである。しかし、もうちょっと簡単な形に書き直せることが知られている。まず、1 次の項だが、

$$\frac{\partial}{\partial v_i}\left(\frac{1}{V}\right) = -\frac{1}{V^2}\frac{\partial V}{\partial V_i}\frac{\partial V_i}{\partial v_i} = \frac{V_i}{V^3} = -\frac{\boldsymbol{e}_i \cdot \boldsymbol{e}_1'}{V^2} \tag{2.56}$$

という都合のよい関係がある。$\boldsymbol{e}_1'$ をそもそも $\boldsymbol{V}$ に平行にとったから上のように出来るわけである。このため、

$$h(\boldsymbol{v}) = \int \frac{f(\boldsymbol{v}_f)}{|\boldsymbol{v} - \boldsymbol{v}_f|} d\boldsymbol{v}_f \tag{2.57}$$

なる関数 $h(\boldsymbol{v})$ を導入して、

$$< \Delta v_i >= -\Gamma(1 + m/m_f)\frac{\partial h}{\partial v_i} \tag{2.58}$$

ということになる。

2 次の項についても同様な整理が可能である。Q_{ij} は $\boldsymbol{e}_i$, $\boldsymbol{e}_j$ の $\boldsymbol{e}'_2$ と $\boldsymbol{e}'_3$ によって張られる平面への写像の内積なので、$\boldsymbol{e}'_1$ との内積の分をつけてやれば元の単位ベクトル同士の内積になる。つまり、

$$Q_{ij} = \delta_{ij} - (\boldsymbol{e}_i \cdot \boldsymbol{e}'_1)(\boldsymbol{e}_j \cdot \boldsymbol{e}'_1) = \delta_{ij} - \frac{V_i V_j}{V^2} \tag{2.59}$$

したがって、

$$\frac{\partial^2 V}{\partial v_i \partial v_j} = \frac{1}{V}\left(\delta_{ij} - \frac{V_i V_j}{V^2}\right) = Q_{ij}/V \tag{2.60}$$

というわけで、

$$g(v) = \int f(\boldsymbol{v}_f)|\boldsymbol{v} - \boldsymbol{v}_f| d\boldsymbol{v}_f \tag{2.61}$$

とおけば、

$$< \Delta v_i \Delta v_j >= \Gamma \frac{\partial^2 g}{\partial v_i \partial v_j} \tag{2.62}$$

2.5.5 バックグラウンド速度分布が熱平衡の場合

前節では、バックグラウンドの速度分布が任意のものについて、実際に計算可能な式を導いた。ここでは、速度分布が等方的な場合について式をさらに単純化してみる。

速度分布が等方的な場合、h や g の積分を、 $\boldsymbol{v}_f$ の絶対値方向と角度方向に分けることができる。角度方向の積分については、 $\boldsymbol{v}$ と $\boldsymbol{v}_f$ のなす角度を θ とし、$\mu = \cos\theta$ とすれば、球面上での積分が、まず h については

$$\int \frac{d\boldsymbol{v}_f}{|\boldsymbol{v} - \boldsymbol{v}_f|} = 2\pi \int_{-1}^{1} \frac{d\mu}{|v^2 - 2\mu v v_f + v_f^2|^{1/2}} = 4\pi \begin{cases} 1/v, & (v > v_f) \\ 1/v_f, & (v < v_f) \end{cases} \tag{2.63}$$

となる。これは、球面上に分布する電荷の作るポテンシャルと同じ式である。g についても同様に計算できて

$$\int |\boldsymbol{v} - \boldsymbol{v}_f| d\boldsymbol{v}_f = 2\pi \int_{-1}^{1} |v^2 - 2\mu v v_f + v_f^2|^{1/2} d\mu = \frac{4\pi}{3} \begin{cases} 3v + v_f^2/v, & (v > v_f) \\ 3v_f + v^2/vf, & (v < v_f) \end{cases} \tag{2.64}$$

となる。これから、

$$\begin{aligned} F_n(v) &= \int_0^v \left(\frac{v_f}{v}\right)^n f(v_f) dv_f \\ E_n(v) &= \int_v^\infty \left(\frac{v_f}{v}\right)^n f(v_f) dv_f \end{aligned} \tag{2.65}$$

というものを考えると、

$$\begin{aligned} h(v) &= 4\pi v[F_2(v) + E_1(v)] \\ g(v) &= \frac{4\pi v^3}{3}[3F_2(v) + F_4(v) + 3E_3(v) + E_1(v)] \end{aligned} \tag{2.66}$$

これらから、最終的な結果、すなわち、バックグラウンドが動いている時の、速度に平行な速度変化と垂直なそれを書き下せることになる。それらは、結局以下のようになる。

$$< \Delta v_{\text{平行}} > = -4\pi\Gamma \left(1 + \frac{m}{m_f}\right) F_2(v) \tag{2.67}$$

$$< \Delta v^2_{\text{平行}} > = \frac{8\pi\Gamma v}{3}[F_4(v) + E_1(v)] \tag{2.68}$$

$$< \Delta v^2_{\text{垂直}} > = \frac{8\pi\Gamma v}{3}[3F_2(v) - F_4(v) + 2NE_1(v)] \tag{2.69}$$

これらから、粒子のエネルギーの変化 ΔE を出すことができる。

$$\Delta E = v\Delta v_{\text{平行}} + < \Delta v^2_{\text{平行}} > /2 + < \Delta v^2_{\text{垂直}} > /2 \tag{2.70}$$

と書けるので、1 次の項は

$$< \Delta E >= 4\pi\Gamma v \left[E_1(v) - \frac{m}{m_f} F_2(v)\right] \tag{2.71}$$

となる。2 次の項については、 $(v\Delta v_{\text{平行}})^2$ 以外の項は小さいので無視すると

$$< \Delta E^2 >= \frac{8\pi\Gamma v^3}{3} [F_4(v) + E_1(v)] \tag{2.72}$$

となる。

さて、速度分布を熱平衡、すなわち

$$f_0(\boldsymbol{v}) = \frac{n_f}{(2\pi\sigma^2)^{3/2}} \exp\left(\frac{-v^2/2}{\sigma^2}\right) \tag{2.73}$$

とすると、上の係数等を具体的に計算できることになって、その形は

$$< \Delta v_{\text{平行}} > = -4\frac{n_f\Gamma}{\sigma^2} \left(1 + \frac{m}{m_f}\right) G(x) \tag{2.74}$$

$$< \Delta v^2_{\text{平行}} > = 2\sqrt{2}\frac{n_f\Gamma}{\sigma} G(x)/x \tag{2.75}$$

$$< \Delta v^2_{\text{垂直}} > = 2\sqrt{2}\frac{n_f\Gamma}{\sigma} \frac{\text{erf}(x) - G(x)}{x} \tag{2.76}$$

$$< \Delta E > = \sqrt{2}\frac{n_f\Gamma}{\sigma} \left[-\frac{m}{m_f}\text{erf}(x) + \left(1 + \frac{m}{m_f}\right) x\text{erf}'(x)\right] \tag{2.77}$$

ここで erf は誤差関数であり、

$$G(x) = \frac{\mathrm{erf}(x) - x\mathrm{erf}'(x)}{2x^2} \tag{2.78}$$

また $x = v_t/(\sqrt{2}\sigma)$ である。

山ほど式はでてきたものの、だからなんなんだという気もする。以下、上の式の意味についてちょっと考えてみる。

まず、速度の 1 次の項を見てみる。これは、速度分布には F_2 だけを通して依存しているということに注目して欲しい。例えば、マックスウェル分布のようなものを考えた時、 v が大きい極限では $F_2 \sim 1/(2\pi v^2)$ となるので、回りが止まっているときと同じく速度変化は速度の 2 乗に反比例する。これに対して、v が小さい極限では、f を一定と見なすことが出来るので $F_2 \propto v$ となる。

これは、タイムスケールを考えてみると、速度が大きい極限では減速のタイムスケールが v^3 であるのに対し、逆の極限では 一定になるということである。すなわち、非常に速度が大きい粒子が出来てしまうとこれはなかなか減速しない。もちろん、自己重力系の場合には、そのようなものは系のなかに留まるのは困難である。このような粒子（超熱的粒子）が問題になるのはプラズマの場合である。

速度が小さいほうではタイムスケールがある一定値、つまりは $v \sim \sigma$ で決まる値あたりになる。

この 1 次の項は、前に述べたように dynamical friction を表している。これが問題になる場面は、例えば恒星系が質量の違う 2 つの成分から出来ているような場合である。力学平衡状態で、分布関数に質量依存がないようなものを考えると、これは熱平衡から遠く離れている。従って、上の式で決まるタイムスケールで重いものがエネルギーを失い、軽いものがエネルギーを得る。これは、エネルギー等分配に向かう普通の熱力学的な進化である。

が、自己重力系ではこのエネルギー交換の結果熱平衡に近付くとは限らない。つまり、重いものがエネルギーを失い、軽いものがエネルギーを得るということは、それぞれの分布関数が変わり、空間分布も変わるということである。具体的には、重いものは中心に集中した分布になり、軽いものは外側に押し出される。その結果それぞれの成分の速度分散がどうなるかの細かいところは初期条件に依存するが、普通は物が中心に集まれば重力も強くなり、力学平衡では結局速度分散も大きくなる。このために、熱平衡からはかえって遠くなってしまう。

さて、次に、2 次の項を見てみる。速度に平行な成分も垂直な成分も、 v が大きい極限では 0 にいく。特に、 垂直な成分は v に反比例する。これに対し、速度が 0 の極限では、どちらも一定値に収束する。これはテスト粒子が停止している極限でも、回りの粒子によって揺さぶられるということを表しているわけである。

2.5.6　２体緩和のタイムスケール

前節では、実際にバックグラウンドの粒子も動いている場合について、２体緩和によって粒子の速度、エネルギーがどう変化するかの期待値（のモーメント）を求めた。ここでは、前節の結果を使って、まずいくつかの重要な場合について２体緩和がどのように働くかを議論する。

「タイムスケール」というのは、普通ある量 x の時間変化が

$$\frac{dx}{dt} = (-)\frac{x}{T} \tag{2.79}$$

なる形で書ける時の T のことである。もちろん、一般には T は x を含むいろんなものの関数である。

そういった意味で考えやすいのは、温度（平均運動エネルギ）が違う２つの空間一様な分布が重なりあっている時に、どのようにして２つが近付いていくかというのものである。これは、もちろん時間が立てば熱平衡に近付くわけである。以下、実際に計算してみる。

今、フィールドに質量 m_f の粒子が一様に分布しており、テスト粒子として質量 m_t のものがこれもまた一様に分布しているとする。さらに、どちらも速度分布はマックスウェルで与えられるとする。ここでは等分配を考えるので、それぞれの粒子１個当たりのエネルギーを E_f, E_t と書く。今、テスト粒子のエネルギー変化の平均を考えると、

$$\frac{d < E_t >}{dt} = 4\pi \int v_t^2 f(v_t) < \Delta E_t > dv_t \tag{2.80}$$

と書けることになる。これに前回求めた $< \Delta E >$ を入れて実際に積分を実行することができて、結果は

$$\frac{d < E_t >}{dt} = 2\sqrt{\frac{6}{\pi}} \frac{m_t n_f \Gamma}{m_f} \frac{< E_f > - < E_t >}{(v_t^2 + v_f^2)^{3/2}} \tag{2.81}$$

となる。

ここで、いくつかの極限的な場合を考えておくことは有益であろう。まず、$m_t >> m_f$ で $v_t \sim v_f$ という状況を考えてみる。これはつまり非常に重いものと軽いものが、同じような空間分布、速度分布で広がっている場合である。この時は上の式で $E_t >> E_f$ なので、

$$\frac{d \log < E_t >}{dt} = -\sqrt{\frac{3}{\pi}} \frac{m_t n_f \Gamma}{m_f v^3} \tag{2.82}$$

となる。なお、この時、変化率はテスト粒子の速度に分母の v_t^2 を通してしか依存しないので、 $v_t \to 0$ の極限でエネルギー変化（減速）のタイムスケールは一定値にいき、それは $v_t \sim v_f$ の時の値とそれほど違わない。

次に、$m_t \sim m_f$ で $v_t << v_f$ という状況を考えてみる。この時は上の式を

$$\frac{d(< E_t > /m_t)}{dt} = 2\sqrt{6/\pi}\frac{n_f \Gamma}{m_f}\frac{< E_f >}{2v_f^3} = 8\sqrt{6\pi}G^2 \ln \Lambda n_f m_f < E_f > v_f^{-3} \quad (2.83)$$

となる。ここで

$$\Gamma = 4\pi G^2 m_f^2 \ln \Lambda \quad (2.84)$$

を使った。さらに、 $E_t = m_t v_t^2/2$ などを使って書き直せば

$$\frac{d(v_t^2)}{dt} = 4\sqrt{6\pi}G^2 \ln \Lambda n_f m_f^2 v_f^{-1} \quad (2.85)$$

を得る。つまり、速度が小さい極限では、一定の率でエネルギーをもらうわけである。言いかえれば、温度が 2 倍になるタイムスケールというものは、温度に比例して小さくなるともいえる。

さて、通常「2 体緩和のタイムスケール」という時には何を指しているかというと、この等分配のタイムスケールのことではないのが普通である。が、時と場合によっていろんなものが出てくるが、まあ同じようなものである。普通に使われるのは、

$$t_r = \frac{1}{3}\frac{v_m^2}{< \Delta v_{\text{平行}}^2 >_{v=v_m}} = \frac{v_m^3}{1.22 n \Gamma} = \frac{0.065 v_m^3}{n m^2 G^2 \log \Lambda} \quad (2.86)$$

とするものである。ここで v_m は r.m.s. 速度である。1/3 になにか意味があるわけではなく、こう定義したというだけである。

これは、ローカルな量で定義されていて、例えば系全体の緩和時間といったものを考えるのにはちょっと不便なこともある。というわけで、いわゆる half-mass relaxation time t_{rh} というものを導入しておく。これは、半径 r_h の中に質量の 1/2 があるとして、その中の密度は一様であるとし、さらにビリアル定理から出てくる $T \sim 0.2GM^2/r_h$ といった関係を使って出すことが出来る。ここでビリアル半径ではなく half-mass radius を使ったのは慣習に従っただけで深い意味はない。この 2 つは通常 30% 程度の範囲で一致する。これは

$$t_{rh} = 0.138\frac{N r_h^{3/2}}{M^{1/2} G^{1/2} \log \Lambda} \quad (2.87)$$

となる。

ここで注意しないといけないことは、 t_{rh} はあくまでも球対称に近い系の half mass radius のあたりでの緩和時間であるに過ぎないということである。従って、球状星団全体の緩和時間とか、あるいは楕円銀河、銀河団といったものには有効な概念であるが、球対称から大きくずれた銀河とか、あるいは half mass radius のずっと外側やずっと内側では全く違ったものになっていることに注意する必要がある。さらに、速度分布が非等方であるとか、回転がメインであるとかでもまた話が全く変わってくる。このような場合、ローカルな緩和時間、あるいはエネル

ギー変化自体の式に戻って考えないと、タイムスケールについて全く間違った推定をしてしまうことになる。

[牧野淳一郎]

第3章　講義2　N体シミュレーションの基礎

3.1　はじめに

N体シミュレーションの基礎ということで、ここでは以下のような話題について簡単に触れることにしたい。

- 常微分方程式の数値積分法の基礎 (オイラー法を例として)
- リープフロッグ法
- コンピュータでの数値表現
- 誤差 (丸め、打ちきり), 桁落ち, 情報落ち

まあ、「そんなことはよくよく知ってる」という人は寝てて下さいな。

3.2　数値解法の基本： Euler method

まず、数値解法のもっとも単純なものである Euler 法を題材に、常微分方程式の数値解とはどういうものかについて考えていこう。

3.2.1　Euler 法—一変数の場合

まず、一変数の場合を考えよう。微分方程式の初期値問題

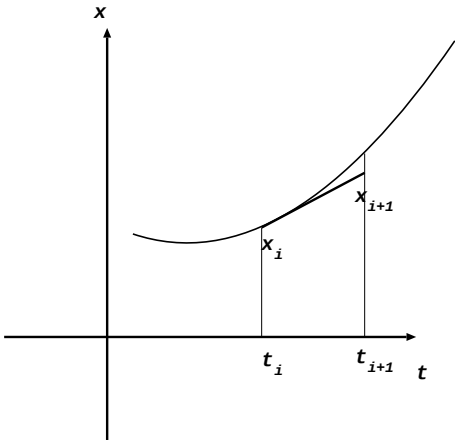
$$\dot{x} = f(x,t), \quad x(t_0) = x_0 \tag{3.1}$$

に対して、(前進) オイラー法とは以下のような方法である。

時刻 t_i での数値解が x_i であったとすると、時刻 $t_{i+1} = t_i + \Delta t$ での数値解を

$$x_{i+1} = x_i + \Delta t f(x_i, t_i) \tag{3.2}$$

という形に書く。



上の図に様子を示す。ここでは、真の解（がわかっているものとして）を曲線で示し、そこから出発したとした。

やっていることはなんということはなくて、真の解に対する時刻 t_i での接線を数値解ということにしようということである。

ある時刻 t_s での初期条件が与えられて、 t_e での解を求めようという時に、例えばその区間を n 等分して

$$t_0 = t_s, \quad t_k = t_0 + \Delta t k, \quad \Delta t = \frac{t_e - t_s}{n} \tag{3.3}$$

というふうに t_i を決めてやる。あとは x_1 から順に $x_2, x_3, ...x_n$ と計算していけば、 $t_e = t_n$ での数値解 x_n が求まるわけである。

直観的には、 n を大きくしていけば、正しい答えに近付いていきそうな気がする。しかし、本当にそうか？というのがここでの話である。

3.2.2 例：線形方程式の場合

とりあえず、答がわかっている場合にオイラー法がどう振舞うかということをみてみる。初期値問題

$$\dot{x} = kx, \quad x(0) = 1 \tag{3.4}$$

の $t = 1$ での数値解を考える。いうまでもないが、厳密解が $x = e^{kt}$ で与えられるわけである。

$$x_{i+1} = x_i + \Delta t k x_i = x_i(1 + \Delta t k) \tag{3.5}$$

なので、

$$x_e = (1 + k/n)^n \tag{3.6}$$

である。指数関数は定義により

$$e^t = \lim_{n\to\infty} (1 + t/n)^n \tag{3.7}$$

と書けるので、問題は、この「数値解」がどれくらい正しいかということであるが、

$$\lim_{n\to\infty} x_e = e^k \tag{3.8}$$

となり、$n \to \infty$ の極限で数値解が厳密解に一致するということが示せたことになる。上の証明は任意の係数、初期条件と積分区間に一般化できる。

なお、多変数の場合にもまったく同様に厳密解に収束することが証明できる。

3.3 計算機の精度

さて、前節では、とりあえずもっとも簡単な一変数線形の方程式で極限で厳密解が得られるということを証明できたわけだが、じつは実際に計算機で計算してみるとこんなにうまくはいかない。

うまくいかないというのはどういう意味かというと、実際に n をどんどん大きくしていっても、答が厳密解に近付かなくなるどころかかえって遠ざかっていくということが起きる。これは実際にプログラムをつくって確かめてみればすぐにわかる。

なぜそういうことが起きるか、また、そうならないようにして意味がある数値解を得るためにはどうすればいいかというのが、さまざまな数値解法が研究される大きな目標の一つである。ちなみに、もうひとつの目標は、なるべく短い計算時間で正確な答を得るということである。

計算機で、例えば C でプログラムを書くと、`int` と宣言する、普通整数とよぶデータ型と、`float` または `double` と宣言する、普通「実数」とよぶデータ型を必要に応じて使い分ける。微分方程式の解を求めようという場合なら、普通従属変数には `double` を使うであろう。

計算機を使う時に「実数」と呼ぶものは、数学的な意味での実数とはもちろん違うものである。なにが違うかというと、基本的には「有限の桁数」しかないということである。

つまり、数学的な実数というのは、もちろん、連続であるわけで、ということは、たとえば 1.000.....0001 と 0 が何個続いても、そういう実数というものはあるわけである。

ところが、計算機では、ある数を有限の情報量（ビット数）で表現する必要がある。今のふつうの計算機では、例えば C の float なら 32 ビット、double なら 64 ビットを使う。例えば float の場合、実際の表現は（符号）$\times(2^{\text{指数}})\times$（仮数）という形になっていて、符号に 1 ビット（0 なら正、1 なら負）、指数に 8 ビット、仮数に残りの 23 ビットを使う。

仮数は 1 と 2 の間を表現すればいいので、 0 を 1 ということにして、オール 1（16 進数で `7fffff`）を $2-2^{-23}$ に割り当てる。指数については、正負両方にとるために、 7f のときに $2^0=1$ ということにする。

なお、あと、0 というものを表現する必要があるが、これは「全ビット 0 」が 0 ということにする。

ちょっと話が細かくなったが、問題は要するに有限の桁数（ビット数）で表現されているということである。このために、四則演算、あるいは関数計算をしたときに得られる答というものが、厳密に正しいわけではなくなっている。つまり、無限小数になるものをどこかで打ち切るので、その分誤差がでるわけである。

なお、打ち切り方には、切捨て、切り上げ、四捨五入などいろいろあり得るが、最近の計算機は大抵四捨五入になっている。といっても 2 進数なので 0 捨 1 入 (最近接丸め) ということになる。さらに細かいことをいうと、捨てるところがちょうど 1 の時にどうするべきかという問題があるが、これはさらにその上が 0 になるほうにもっていく（最近接偶数丸め）ということをする計算機もある。

double であれば仮数が 52 ビット、指数が 11 ビットになる。

3.3.1 四則演算における丸め誤差

四則演算を行なった時に丸め誤差がどう発生するかをちょっとまとめておく。

簡単なのは乗算・除算である。これらについては、最近のまともな計算機ならば、計算結果は厳密な結果を丸めたものになる。浮動小数点数ならば相対誤差がでるということになる。

加算・減算についても、計算結果は厳密な結果を丸めたものになるということ自体には変わりはない。が、計算結果の大きさががもとの数と大きく違うことがあるので、相対誤差が大きく変わることがある。

例えば、今十進で有効数字 4 桁の浮動小数点表示をしていたとしよう。0.1234×10^2 から 0.1233×10^2 を引いたら、答は 0.1×10^{-1} である。この答自体は正確なもので、ここで誤差は発生していないが、しかし求まった答の有効数字は 1 桁しかない。こういうのを桁落ちという。

N 体計算では、桁落ちが問題になるのは主に相互作用の計算である。近くにある粒子の座標の引算をした時に有効桁数が落ちる。これはもうどうやっても落ちるわけで、桁落ちを改善するにはより高い精度で計算するしかない。

もう一つの例として、0.1234×10^2 に 0.1233×10^{-1} を足すことを考えてみる。答は 0.1234×10^2 に 0.1235×10^2 になって、0.233×10^{-2} はどこかに消えてしまう。ここでは実際に誤差が発生している。これはもちろん足す数のうち大きい方に対する相対誤差程度ではあるが、小さい方からみると誤差が大きくなっているのが問題である。こういうのを情報落ちという。

数値積分では、どうしても大きな数に小さな数を加えるのを繰り返すので、高い精度を実現したい時にはこれは重要な問題である。

3.3.2 丸め誤差のある場合の収束性

実は丸め誤差がある場合の計算精度の議論というのはかなり難しい。しかし、このあたりがちゃんとわかってないと、計算して出てきた答が正しいかどうかということが理解できないので、少し詳しい解説を試みる。

前にやった、線形方程式の例をそのまま使うことにしよう。式ではわからないので、プログラムに表してみる。

```
x = 1;
dt = 1.0/n;
for(i=0;i<n;i++){
   x = x + x*k*dt
}
```

例えば C で書けばこんな感じだろう。丸め誤差の結果、答がどう変わるかということを考えてみる。

これにはいくつかの方針がありえる。一つは、「悲観的」解析である。（区間解析という専門用語がある）つまり、演算毎に、誤差があったら答が最大これだけずれるというのを見積もって、答の両側に幅をつけていく。もう一つは、「楽観的」解析（確率的解析ともいう）である。丸めが最近接丸めであるときには、「計算結果は真の値の回りのある幅で一様分布する」と仮定してよい場合が多い。この時には、計算結果は確率的に真の値の回りをランダム・ウォークすることになる。

これらの 2 つの方法は原理的には美しいが、実際にはどちらも役に立たないことが多い。というのは、現実にはどちらも恐ろしく手間がかかるうえに、悲観的解析はあまりに悲観的であり、楽観的解析は逆に過度に楽観的であるからである。上の例で、どうなるか考えてみる。

このためには、まず、丸め誤差のモデル化が必要である。いま、簡単のために、任意の演算に対して丸め誤差が以下のように表現できるとする : 演算の「真の」結果が p であるとき、計算機で表現された結果は区間 $[(1-\epsilon)p, (1+\epsilon)p]$ の間に一様分布する。

まず、区間解析を試みよう。 面倒なので $k = 1$ とし、 k との乗算では誤差は生じないとする。この時、 $dt = [(1-\epsilon)/n, (1+\epsilon)/n]$ となり、`for` の最初の反復の後では

$$\begin{aligned} x^{(1)} &= [(1 + (1-\epsilon)/n)(1-\epsilon), (1 + (1+\epsilon)/n)(1+\epsilon)] \\ &\simeq [1 + 1/n - \epsilon(1 + 2/n), 1 + 1/n + \epsilon(1 + 2/n)] \end{aligned} \tag{3.9}$$

で、次の反復ではどうなるかとかいって順に計算していけばいいが、これはあまりに大変であるので以下の近似を行なう：

- 誤差は `x + x*dt` の、加算のところだけで出る。

これは、通常の場合にはそれほど悪い近似ではない。というのは、ここで出る誤差が圧倒的に大きいからである。このときは、結局、数値解が

$$\left(1+\frac{1}{n}-\epsilon\right)^n < x < \left(1+\frac{1}{n}+\epsilon\right)^n \tag{3.10}$$

の範囲にくるということになる。

今、 $n >> 1, n\epsilon << 1$ として展開すれば、

$$e(1-n\epsilon) < x < e(1+n\epsilon) \tag{3.11}$$

ということになり、 n に比例する誤差が入るということになる。つまり、刻みを細かくしていくと、真の解からどんどん遠ざかっていく（かもしれない）ということがわかる。

さて、確率的解析の場合にも同様に上の単純化をすると、結果は誤差の平均値は 0 で分散が $n\epsilon^2$ の程度ということになる。つまり、$\sqrt{n}\epsilon$ くらいの誤差である。

実際に計算してみると、区間評価では $n > \sqrt{1/\epsilon}$ で誤差を大きめに、逆に確率評価では $n > \epsilon^{-2/3}$ で小さめに見積もっているということがわかる。

この、非常に単純な場合には、どちらについてもそのようにずれる理由がわかり、精密な解析を行なえば誤差のもう少し正しい上限を与えることは不可能ではない。が、解くべき方程式や計算法がすこし複雑になると、解析が非常に困難になることは注意するべきであろう。

なお、一々丸め誤差のことを考えていると話が進まないので、以下、基本的な話は丸めのことを無視して、必要に応じて丸めの影響を考えることにする。なお、ふつう「打ち切り誤差」という時には、この意味の丸めを無視したもののことである。

3.3.3 オイラー法の収束性と打ち切り誤差

線形方程式の場合に、ステップサイズ 0 の極限で真の解に収束するということを前に示したわけだが、計算機が実際に計算するときにはもちろん有限のステップサイズである。また、上に述べたように丸めの影響があって、どうせ収束はしない。そうすると、どの程度のステップサイズの時に計算精度がどうなるかということを知る必要がある。

もちろん、プログラムを書いて、真の解との誤差を調べればいいが、もともと数値解を計算するのは真の解がわからないからなので、そうはいかないわけである。

以下、一般の微分方程式に対してオイラー法がどのように収束するかということを考えてみる。常微分方程式の初期値問題

$$dx/dt = f(t,x), \quad x(t_s) = x_0 \tag{3.12}$$

の $t_0 \leq t \leq t_1$ での解を考える。今、 関数 f が領域 $t_0 \geq t \geq t_1, |x - x_0| \leq b$ で連続であり、最大値、最小値をもつとしよう。絶対値が M でおさえられ、$t_e - t_s < b/M$ という関係がなりたつとする。さらに、上の範囲の任意の t, x と x' についてリプシッツ条件

$$|f(t,x) - f(t,x')| \leq L|x - x'| \tag{3.13}$$

を満たすものとする。さらに、n, h を、 $nh = t_e - t_s$ を満たすようにとる（n は整数)。

このとき、オイラー法が一様に一次収束する、すなわち、ある定数 C が存在し、

$$|x_i - x(t_i)| \leq Ch \tag{3.14}$$

となることを示すことができる。

3.3.4 公式の次数について

と、その前に、局所離散化誤差という概念を導入する。これは以下のように定義される。 (t_i, x_i) から出発した数値解が x_{i+1} であったとし、また厳密解が $x(t)$ であるとする。これは、あくまでも x_i からの解—局所的な解—であって、初期条件からの解ではないことに注意して欲しい。この時に、数値解の局所離散化誤差が m 次であるとは、

$$x(t_{i+1}) = x_{i+1} + O(h^{m+1}) \tag{3.15}$$

という関係を満たすこと、言い換えれば、ある定数 A が存在して、

$$|x(t_{i+1}) - x_{i+1}| \leq Ah^{m+1} \tag{3.16}$$

なる関係を満たすことである。A は方程式に依存してよいが、h には依存しない必要がある。

m ではなくて $m+1$ にするのは、 h が小さくなった分を割り引くためである。

オイラー法の局所離散化誤差は、

$$x_{i+1} = x_i + hf(x_i, t_i) \tag{3.17}$$

と、t_i での x の真の解のテイラー展開

$$x(t_i + h) = x_i + h\frac{dx}{dt} + \frac{h^2}{2}\frac{d^2x}{dt^2} + O(h^3) \tag{3.18}$$

を比べてみればわかるように、$O(h^2)$ に、つまり 1 次になっている。まあ、これはテイラー展開があればの話だが、以下展開はある、つまり f は C^∞ であるということで話を進める。

3.3.5　オイラー法の収束性の証明

証明の方針としては、具体的に x_{i+1} を構成しておく：

$$x_{i+1} = x_i + hf(x_i, t_i) \tag{3.19}$$

今、真の解との差が欲しいので、こちらも書いておくと

$$x(t_{i+1}) = x(t_i) + \int_{t_i}^{t_{i+1}} f(x(t), t)dt = x(t_i) + hf(x(t_i), t_i) + O(h^2) \tag{3.20}$$

差をとれば

$$x_{i+1} - x(t_{i+1}) = x_i - x(t_i) + h[f(x_i, t_i) - f(x(t_i), t_i)] - O(h^2) \tag{3.21}$$

絶対値をとれば

$$|x_{i+1} - x(t_{i+1})| \leq |x_i - x(t_i)| + |h[f(x_i, t_i) - f(x(t_i), t_i)]| + Ah^2 \tag{3.22}$$

ここで A は定数である。f が無限回微分可能としたので A は必ず存在する[1]。今、

$$e_i = x_i - x(t_i) \tag{3.23}$$

とおいて、さらにリプシッツ条件から変形して

$$|e_{i+1}| \leq |e_i|(1 + hM) + Ah^2 \tag{3.24}$$

ここで、明らかに $e_0 = 0$ なので、容易にわかるように

$$|e_i| \leq Ah^2 \sum_{k=0}^{i-1} (1 + hM)^k = \frac{Ah}{M}[(1 + hM)^i - 1] \tag{3.25}$$

ここで、 $nh = t_e - t_s$ であったことを思い出すと

$$(1 + hM)^i - 1 \leq e^{M(t_e - t_s)} - 1 \tag{3.26}$$

結局、

$$|x_i - x(t_i)| \leq \frac{Ah}{M}[e^{(t_e - t_s)M} - 1] \tag{3.27}$$

となる。

というわけで、一応、誤差の上限が存在して、それが h に比例するということが示された。なお、上限の形をみるとなかなか嫌な格好をしていることがわかる。つまり、時刻依存性が e^{tM} の形をしているので、積分を先に進めていくと、 かならず誤差の上限が指数関数的に増大していくことになる。これは、言い換えれば、長時間積分した場合の精度については理論的にはなかなか難しい問題があるということである。

なお、このような、ある時間範囲で積分したあとの誤差のことを「全離散化誤差」という。これに対し、 1 ステップだけ積分したあとの誤差を局所誤差、あるいは局所離散化誤差という。

[1]無限回でなくてもなんかできるはずだけど面倒なので

3.3.6　もう少し賢い方法

オイラー法は、すでに述べたように 1 次収束しかしない。つまり、計算精度をあげていこうとすると、それに比例して計算量が増えてしまう。また、ステップ数が大きくなるので丸め誤差の影響が出てくる。それでは、もう少し賢い方法はないのであろうか？

もちろん、賢い方法はいろいろあるわけである。その中で実用になっているのは主に以下の 3 種である。

1. ルンゲ・クッタ法
2. 線形多段階法
3. 補外法

とりあえず、ルンゲ・クッタ法の話をする。

3.4　ルンゲ・クッタ法

ルンゲ・クッタ法というのは、ある一般的な形に書ける公式のクラスであるが、ちょっとわかりにくいのでまず例をあげておこう。

3.4.1　二次のルンゲ・クッタ法

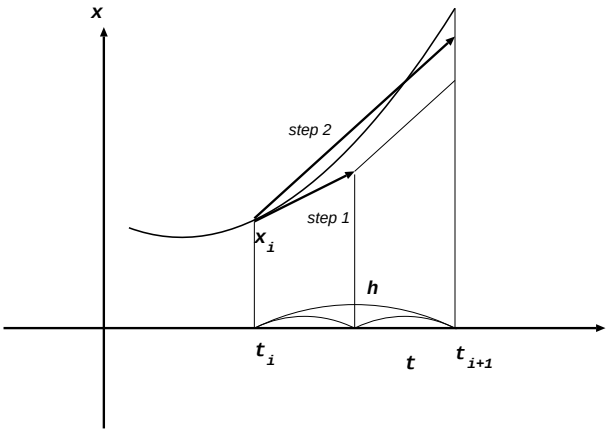
以下のような計算法を考える

$$\begin{aligned} k_1 &= x_i + \frac{h}{2} f(x_i, t_i) \\ x_{i+1} &= x_i + h f(k_1, t_i + h/2) \end{aligned} \tag{3.28}$$

これは二次の公式になっている。

局所誤差で m 次の公式は全離散化誤差でも m 次になるということは 1 次のオイラー法の場合と全く同様にして証明できる。一般には次数が高ければ刻みを小さくしていった時に速く誤差が小さくなる。

が、これはいつでも次数が高い方がいいということを意味するわけではないということには注意する必要がある。次数が高い公式は一般に局所誤差の係数も大きく、そのために刻みが大きい時には次数が低い公式よりも精度が悪くなる。また、 RK 法の場合には次数をあげると計算量も増えるので、計算量を同じにして比べるともっと悪くなる。したがって、必要な精度に応じて最適の次数があるということになる。



3.4.2 RK 法の一般形

2 次の Runge-Kutta 法については、図的な説明というものが可能である。元の点からまずオイラー法と同様に接線を引く。が、これを次の時刻まで延ばすのではなく、ステップの半分のところで止める。で、ここでもう一回微分方程式の右辺を評価する。ここでの導関数の値を使って、もとのところ (t_i, x_i) から直線を引くわけである。

実は、2 次精度を実現する RK 型の公式というのはこれ一つではない。以下のようなものも可能である。

$$\begin{aligned} k_1 &= hf(x_i, t_i) \\ k_2 &= hf(x_i + k_1, t_{i+1}) \\ x_{i+1} &= x_i + \frac{k_1 + k_2}{2} \end{aligned} \tag{3.29}$$

さらに、これらの公式を含む一般的な公式を与えることもできる。

もっと一般には、以下の形に書くことができる。

$$\begin{aligned} x_{n+1} &= x_n + h\sum_{i=1}^{s} b_i k_i \\ k_i &= f(x_n + h\sum_{j=1}^{s} a_{ij} k_j, t_n + c_i h) \end{aligned} \tag{3.30}$$

自然数 s を段数（number of stages）という。 a_{ij}, b_i, c_i はパラメータであるが、a と c は普通

$$c_i = \sum_{j=1}^{s} a_{ij} \tag{3.31}$$

となるようにとる。これは、一般にそうでないような公式は不可能ではないがあまりいいことがない（精度がよくならない）からである。

と、こう、式に書いてしまうとすぐにはわからないが、例えば $s = 2$ の場合に書き下してみると

$$\begin{aligned} x_{n+1} &= x_n + h(k_1 b_1 + k_2 b_2) \\ k_1 &= f(x_n + h(a_{11}k_1 + a_{12}k_2), t_n + c_1 h) \\ k_2 &= f(x_n + h(a_{21}k_1 + a_{22}k_2), t_n + c_2 h) \end{aligned} \tag{3.32}$$

と、まあ、こんな感じになる。こちらを良く見ればすぐにわかるように、

1. $a_{ij} \quad (j \geq i)$ がすべて 0 ならば、k_1 から順に計算していくことができる。つまり、「陽的」公式になっている。

2. $a_{ij} \quad (j > i)$ が 0 のときは、各 k_i についての式に k_i だけが入ってくる。これを半陰的 (semi-implicit) 公式という。この場合には、まず k_1 についての方程式をといて、次に k_2 についてのを解いて、、、と順番に計算出来る。

3. 上のような制約が全くない時は、「陰的」公式ということになる。このときは、すべての k_i に対する（一般には非線形な）方程式を一度に解く必要がある。

なぜ陰的公式といった面倒なものをつくるかというのはよくわからないかもしれない。これは 2 つ理由があって、一つは安定性である。

数値解の安定性というのは、つまり本当の解がなんかある時に、そこから指数関数的に離れていったりしないということだが、もちろんそのためには元の解が安定でないといけない。元の解が安定である、つまり線形化した時に固有値の実部が負である時に、数値解もそういう性質をもっていて欲しいというのが数値解の安定性の議論になる。

で、陽解法では、一般にステップサイズを大きくするとそのうちに不安定になる。これに対し、陰解法ではステップサイズに無関係に安定とか、そういった公式を作ることができる。

もう一つは、特に Runge-Kutta の場合、同じ精度を実現するのに必要な段数が少なくて済むということである。まあ、段数が少なくても非線形方程式を解かないといけないのではしょうがないところもあるが、これが例えば反復法で解けて数回くらいでまあまあの解にいくなら悪くない。

3.4.3 古典的 Runge-Kutta 法

陽的ルンゲ・クッタ法のなかでもっとも広く使われているのが「古典的」といわれる公式である。書き下すと、

$$x_{n+1} = x_n + h(k_1/6 + k_2/3 + k_3/3 + k_4/6)$$

$$
\begin{aligned}
k_1 &= f(x_n, t_n) \\
k_2 &= f(x_n + hk_1/2, t_n + h/2) \\
k_3 &= f(x_n + hk_2/2, t_n + h/2) \\
k_4 &= f(x_n + hk_3, t_n + h)
\end{aligned} \tag{3.33}
$$

というものである。これは、いろいろ良い性質をもつ。例えば

1. a_{ij} が $i - j = 1$ 以外すべて 0 なので、右辺の計算が楽である。
2. 次数が 4 次であり、4 段陽的公式で到達可能な最高次数を達成している
3. 係数が簡単な有理数なので、プログラムしやすい。また丸め誤差を小さくできる。

この公式が 4 次であることを示すのは、それほど簡単ではない。腕力に自信があるひとは挑戦してみて欲しい。

で、まだ RK 法や他の方法について知っておくべきことは無数にあるが、いちいち説明していてはそれだけでこの学校の時間全部を使っても足りないので省略する。

3.5 ハミルトン系とそのための解法

N 体シミュレーションということなので、一般的な常微分方程式の解法というよりも、もうちょっと N 体系の性質をうまく表現するような方法はないかという話である。といっても、N 体系の性質というより単にハミルトン力学系の性質になる。

3.5.1 簡単な例題

古典力学系のもっとも簡単な例といえば、一次元調和振動子、つまり、運動方程式では

$$
\frac{d^2x}{dt^2} = -kx \tag{3.34}
$$

である。これの解はもちろん単振動するものであり、それは固有値が純虚数であるからである。

では数値解はどうなるかということを考えてみる。もとの方程式が線形微分方程式なので、数値解も（よほど変な公式を使わない限り）線形差分方程式の解になる。

差分方程式の解が振動的であるためには、その固有値の絶対値がすべて 1 でないといけない。が、普通の公式ではこれは 1 にならない。もちろん、局所離散化

誤差の程度で 1 からずれるだけなので、タイムステップを十分小さくすればいいというのが「正しい」考え方かもしれない。

が、まあ、もしも数値解が厳密に振動的であれば、それは少なくとも悪い性質ではない。実は多変数の場合や非線形の場合でも本質的な事情は変わらない。ただし、こちらはまだいろいろ理論的にはっきりわかっていないことがある。が、大雑把にいうと同じような事情が成り立っている。つまり、もとが周期解であるときに、大抵の積分法では周期解から一方的にずれていく、つまり、保存量であるはずのエネルギー等が保存しなくなる。

もちろん、エネルギーが厳密には保存しないということが問題であるかどうかというのも実は難しい問題である。というのは、保存しないといってももちろん数値解の精度では保存しているわけだから、それでかまわないのではないかとも考えられるからである。

3.5.2 リープフロッグ公式

もっとも普通に使われる公式は以下の leapfrog といわれるものである。

$$v_{i+1/2} = v_{i-1/2} + \Delta t a(x_i) \tag{3.35}$$
$$x_{i+1} = x_i + \Delta t v_{i+1/2} \tag{3.36}$$

これでは速度と位置がずれた時間でしか定義されないが、出発用公式として

$$v_{1/2} = v + \Delta t a(x_0)/2 \tag{3.37}$$

を使い、さらに終了用公式として

$$v_i = v_{i-1/2} + \Delta t a(x_i)/2 \tag{3.38}$$

を使うことで最初と最後を合わせることが出来る。この形は、実は

$$x_{i+1} = x_i + \Delta t v_i + \Delta t^2 a(x_i)/2 \tag{3.39}$$
$$v_{i+1} = v_i + \Delta t [a(x_i) + a(x_{i+1})]/2 \tag{3.40}$$

と数学的に等価である（証明してみること）また、以下のようにも書ける

$$v_{i+1/2} = v_i + \Delta t a(x_i)/2 \tag{3.41}$$
$$x_{i+1} = x_i + \Delta t v_{i+1/2} \tag{3.42}$$
$$v_{i+1} = v_{i+1/2} + \Delta t a(x_{i+1})/2 \tag{3.43}$$

さらにまた、速度を消去して x_{i-1}, x_i, x_{i+1} の関係式の形でかいてあることもあるかもしれない。なお、すべて違う名前がついていたりするが、結果は丸め誤差を

別にすれば完全に同じである。時々、これらが違うものであるかのように書いてある教科書があるので注意すること。

この公式は2次である。これは展開すればわりあい簡単にわかる。

さて、局所誤差という観点からはこれは決して良い公式というわけではないが、現実にはこの公式は非常に広く使われている。

これは、別にもっとよい方法を知らないからとかではなく、実はこの方法がいくつかの意味で非常に良い方法であるからである。いくつかの意味とは、例えばエネルギーや角運動量のような保存量が非常によく保存するということである。これらの保存量については多くの場合に誤差がある程度以上増えない。

この、ある程度以上誤差が増えないというのは目覚しい性質である。通常の方法では、エネルギーの誤差は時間に比例して増えていく。従って、長時間計算をしようとすればそれだけ正確な計算をする必要がある。ところが、エネルギーの誤差は溜っていかないのならば、かならずしも精度を上げる必要はないとも考えられる。

もちろん、エネルギーが保存していればそれだけで計算が正しいということにはならない。が、理論的には、いくつかの重要な結果が得られている。まず、

1. leapfrog は symplectic method のもっとも簡単なものの一つである。
2. leapfrog は symmetric method のもっとも簡単なものの一つである。

Symplectic method については、以下のようなことが知られている

1. symplectic method は、すくなくともある種のハミルトニアンに対して使った場合に、それに近い別のハミルトン系に対する厳密解を与えることがある。
2. 周期解を持つハミルトン系に対して使った場合に、どんな量でも誤差が最悪で時間に比例してしか増えない。
3. 時間刻を変えると上のようなことは成り立たなくなる

これに対し、 symmetric method については以下のようなことが知られている

1. 周期解を持つ時間対称な系に対して使った場合に、どんな量でも誤差が最悪で時間に比例してしか増えない。
2. 時間刻みを変えてもうまくいくようにすることも出来る。

以下、まず数値例でこれらの性質を確認する。

3.5.3 数値例

調和振動子

能書きを聞いていても良くわからないので、実例を見てみよう。まず、簡単な例として調和振動子

$$\frac{d^2x}{dt^2} = -x \tag{3.44}$$

を leap frog と 2 階のルンゲクッタで解いた例を示す。初期条件は $(x, v) = (1, 0)$ で、時間刻みは $1/4$ である。

軌道とエネルギーを図に示す。非常に特徴的なのは、 leap frog ではエネルギーが周期的にしか変化しないのに対し、ルンゲクッタでは単調に増えていることである。

ルンゲクッタでは単調に変化するというのは前に説明した通りである。さて、これに対し、 leapfrog ではエネルギーが変化していないが、これはどういうことなのであろう？

実は、この調和振動子の場合には、 leapfrog 公式は以下の量

$$H' = \frac{1}{2}(x^2 + v^2) - \frac{h^2}{8}x^2 \tag{3.45}$$

を保存するということを確かめることが出来る。つまり、 (x, v) で与えられる位相平面の上で考えると、 leapfrog 公式の解は上の式で与えられる楕円の上にのっているのである。このために、エネルギーの誤差がある値よりも大きくなり得ないことになる。

非線形振動

では、非線形振動ではどうだろう？簡単な例として

$$\frac{d^2x}{dt^2} = -x^3 \tag{3.46}$$

を leap frog と 2 階のルンゲクッタで解いた例を示す。初期条件は $(x, v) = (1, 0)$ で、時間刻みは $1/8$ である。

軌道とエネルギーを図に示す。調和振動の場合と同様に、 leap frog ではエネルギーが周期的にしか変化しないのに対し、ルンゲクッタでは単調に増えている。

この場合も保存量があるので、頑張れば求まるかもしれない。

3.5.4 シンプレクティック公式

さて、 leapfrog 公式は、上で見たようにハミルトン系に対してエネルギー誤差が有界に留まるという大きな特長がある。が、2 次精度でしかない。もっと高次の方法はないのだろうか、またあるとすればどういう原理で作れるのだろうか。

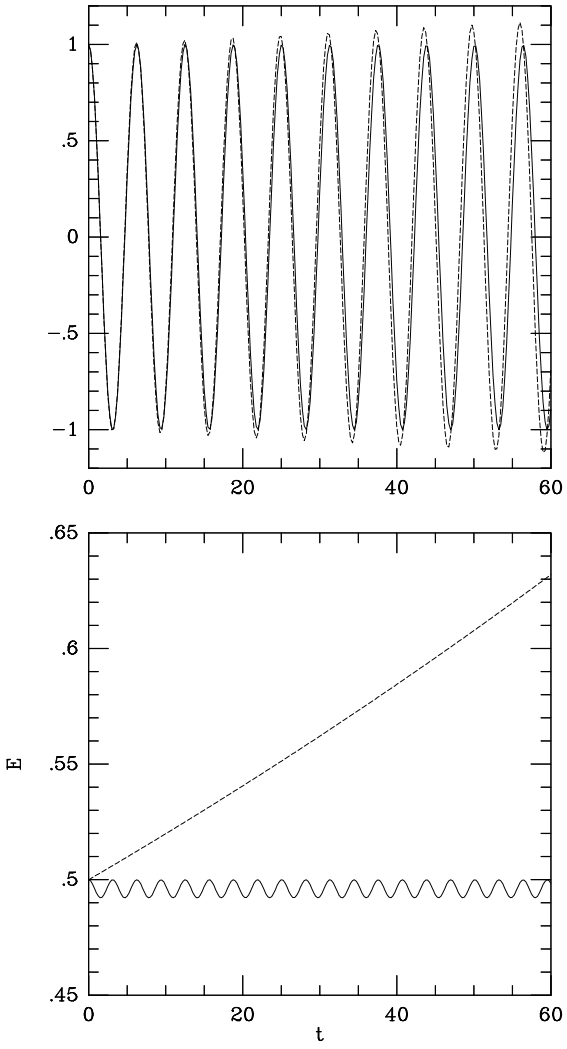


図 3.1: 調和振動子の数値積分。左：軌道。右：エネルギー。破線は 2 次のルンゲクッタ、実線は leapfrog の結果

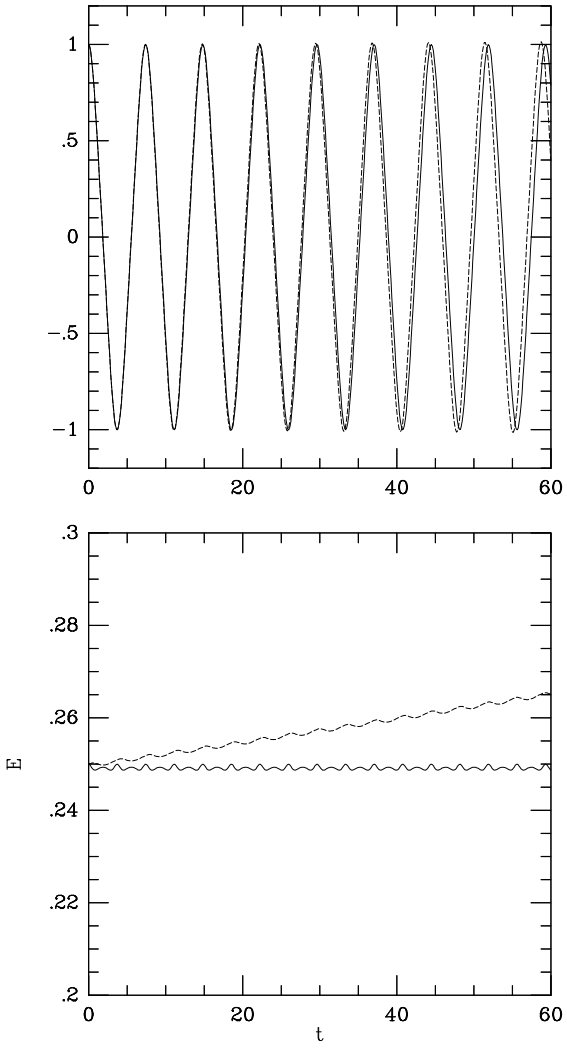


図 3.2: 非線形振動の数値積分。左：軌道。右：エネルギー。破線は 2 次のルンゲクッタ、実線は leapfrog の結果

高次の方法を構成する一つのアプローチが、シンプレクティック公式といわれるものである。これはなにかというと、積分公式がシンプレクティック写像になるように作るということである。

と、これでは意味がわかりませんね。シンプレクティック写像とは、ようするに正準変換のことである。正準変換とはなにかというのは解析力学の入門書をみて欲しいが、直観的には、力学系を不変に保つ、つまり変換まえの座標系で求めた軌道を変換したものと、変換後の座標系での力学系の軌道が厳密に同じになるようなものである。

ハミルトン力学系の解そのもの（ある時刻 t での座標から、$t+h$ での座標に移す変換）もシンプレクティックである。まあ、だから、シンプレクティックになっているような積分公式は、そうでないものに比べてなんとなく力学系の性質にあっているような気はする。

で、いいたかったことは何かっていうと、上の leapfrog 公式はこのシンプレクティック性を満たしているということだった。詳しくは、「数理科学」1995 年 6 月号にのった吉田による解説記事でもみてもらうことにして、ここでは高次の公式にはどんなものがあるかという話をしておく。

陽解法

陽解法の組み立て方はいろいろある。一つは、 RKN 系の公式で、係数をシンプレクティック性を満たすように決めるということである。これはここ 10 年で無数に論文がでた。4 次、あるいは 6 次の公式としては、吉田や鈴木による作用素分解に基づく公式が良く知られている。

これらの方法の原理は、要するに上の leap frog をタイムステップを変えていくつか組合わせるというものである。うまくタイムステップを組み合わせると誤差の高次の項を消すことができるわけである。3 段 4 次の公式、7 段 6 次の公式等が吉田によって導かれている。

なお、実は陽解法はハミルトニアンが $T(p)+V(q)$ の形の場合にしか使えないが、大抵の問題はこう書けることはいうまでもないであろう。

また、 RKN 系の公式を力任せに構成する試みもあり、4 次から 8 次までの公式が作られている。計算精度という観点からはこちらのほうが leapfrog を組合わせるものよりも圧倒的にいいということがわかっている。が、leapfrog を組合わせるものには、以下のような重要な利点があることも確かである。

- プログラムが簡単である
- 余分なメモリを必要としない

例えばこれくらいでプログラムが済む：

```
void yoshida4(nbody * nbp,
              double h,
              int & first_call)
{
    static double d1, d2;
    if (first_call != 0){
        d1 = 1.0 / (2-pow(2,1.0/3.0));
        d2 = 1 - 2*d1;
    }
    leapfrog(nbp,h*d1, first_call);
    leapfrog(nbp,h*d2, first_call);
    leapfrog(nbp,h*d1, first_call);
}
```

リープフロッグを刻みを変えて 3 回呼ぶだけである。

3.5.5 シンプレクティック公式の問題点と対応

さて、式 (3.45) をみるとわかるように、シンプレクティック公式に付随するハミルトニアン H' には時間刻み h が入っている。従って、h をふらふら変えると H' も変わって、結果的に求まった数値解は一つの力学系の軌道ではないなんか変なものになってしまう。ということは、可変時間刻みにするとシンプレクティック公式はうまく働かないのではないかということが想像される。

じつはその通りで、例えばシンプレクティックで埋め込み型のルンゲクッタというものを作って、実際に時間刻みを変えてみた人がいる。その結果、普通のルンゲクッタよりも良くならないということが発見された (1992 年頃)。問題によってはこれは致命的な欠陥となるので、さまざまな対応法が精力的に研究されているのが現状である。が、あまり一般的にうまくいく方法というのは見つかっていないようである。

なお、シンプレクティック公式において普通に時間刻みを変えると、それはシンプレクティックではなくなるということが Gear と Skeel により一般に証明されている。彼らの結果は、時間刻みも含めて考えると、写像が時間方向に歪んだもの（もとの位置によって時間刻みが変わるため）になり、そのために元の公式がシンプレクティックであれば時間刻みを変えると必ずシンプレクティックでなくなるというものである。

それでは、時間刻みを変えるのは絶対に不可能かというと、以下のような考えかたでの研究が進められている。ハミルトニアン H を

$$H = H_1 + H_2 + \cdots \tag{3.47}$$

と複数の項の和にわけ、それぞれについて違う時間刻みを与えることで実効的に時間刻みを変えるというものである。

もう一つのシンプレクティック公式の問題点は、「写像」であるということから一段法、具体的にはルンゲクッタ型の公式であるので、局所誤差に対する計算量という観点からは線形多段階法に比べて必ず悪いということである。

一般に多段階法についてシンプレクティック性のようなものがありえるかどうかというのは、現在研究が進みつつあるが良くわかっていない。

[牧野淳一郎]

第4章　講義3　GPUとGRAPEライブラリ

4.1　はじめに

本講義では計算の高速化のために CfCA の GPU クラスタを使用する方法について解説する。重力多体系のシミュレーションでは、すべての粒子からすべての粒子への重力相互作用を計算する必要がある。そして多くの場合、この部分の計算コストが全計算コストのうちの大部分を占める。より大規模な計算を行うためには、この部分の計算を高速に行うことが重要になる。

計算を高速化するには、クロックサイクルの速い CPU を使う、並列化する、アルゴリズムを工夫して計算量を減らす、など様々な手法がある。しかしクロックに関しては、1 割高速になった代わりに価格が倍になり、同時に消費電力も 1.5 倍になるという具合で、ほとんどの場合選択肢に入らない。したがって、並列化することはほぼ必須であると言ってよく、並列化の程度と計算の内容に合わせて適切な機材を選択することになる。

しかし機材を変更するためには計算コードを変更する必要があり、それにはある程度の時間がかかる。研究を進めるために計算結果を速く得たくて機材を変更しても、計算コードの変更にかかった時間のほうが長くなってしまっては意味がないし、新たなバグを埋めてしまって計算結果が信頼できなくなるというリスクもある。そこで、ライブラリや公開コードを使うことでこのリスクを回避するのが良い。[1]

実習では、GRAPE 互換ライブラリというライブラリを通して GPU を使用する。GRAPE は専用の回路で粒子間相互作用を高速かつ並列に計算するハードウェアで、国立天文台でも 2021 年度まで運用されていた。GRAPE を使うための関数を GPU で使用できるようにしたものが GRAPE 互換ライブラリである。同様に CPU でも GRAPE の関数を使用できるようにしたものとして Phantom-GRAPE という公開コードが存在する。GRAPE ライブラリを使った計算コードを作っておけば、CPU でも GPU でも動作するコードが手に入るということになる。

講義では、重力多体問題の高速化手法の紹介と、GPU および Phantom-GRAPE の実装、そしてより一般的な粒子系の計算を行うためのフレームワークである FDPS

[1] もちろんコードを書くのが好きで楽しいなら、最新の機材に合わせたコードを作ってそれを武器にするというのは選択肢の一つである。

を紹介する。テキストには現在の GPU クラスタの概要と、参考としてハードウェアとしての GRAPE に関する説明を残している。

4.2 国立天文台 GPU クラスタと過去の GRAPE

本節では GPU の使用方法と過去の GRAPE シリーズについて説明する。CfCA では 2021 年度末に GRAPE システムの運用を終了し、現在は計算加速器の積まれた計算機資源は GPU クラスタのみとなっている。

4.2.1 ネットワーク構成

国立天文台 GPU クラスタの構成を図 4.1 に示す。GPU クラスタのすべての計算ノードは、CfCA HPC ネットワークの中に置かれている。どの計算ノードも AMD EPYC プロセッサと NVIDIA A100 GPU が搭載されている。これらは Infiniband HDR ネットワークで相互接続されている。この他に、ログインノード (g00) と、高速ストレージ（/cfca-work）も同じネットワークに接続されている。

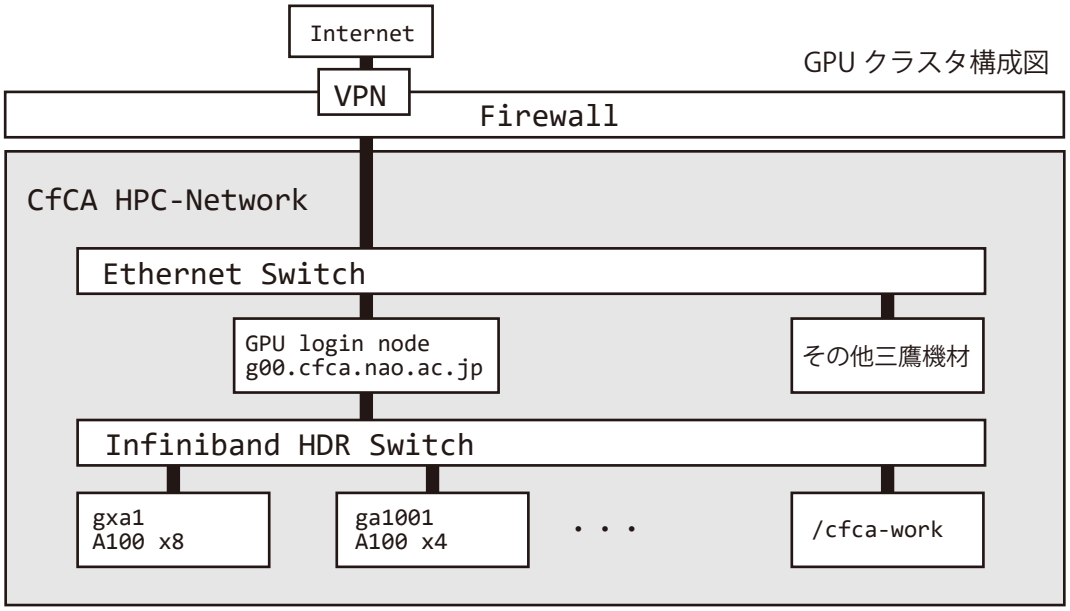


図 4.1: 国立天文台 GPU クラスタ ネットワーク構成図

通常の使い方は、g00 (GPU ログインノード) にログインして、ジョブの投入を行う方法であるが、実習においては g00 経由で GPU クラスタの計算ノードにログインして作業を行う。

表 4.1: GPU クラスタ 各ノードの仕様

gxa1, gxa2	主記憶 2TB/node, NVIDIA A100 40GB 8 基/node
ga1001,ga1002	主記憶 512GBB/node, NVIDIA A100 40GB PCIe 4 基/node
ga1003,ga1004	主記憶 512GBB/node, NVIDIA A100 80GB PCIe 4 基/node

4.2.2 GPU クラスタ概要

GPU クラスタは計算加速器として NVIDIA 社の A100 GPU を搭載した計算機群である。GPU は本来画像処理のための計算装置だが、計算加速器として高い性能を持っており様々な分野で用いられるようになった。GPU は多数のコアで並列計算を行うことで高い性能を実現しており、 A100 GPU には単精度コアが約 7000 個、倍精度コアが約 3500 個搭載されている。つまり数千並列の計算を行わないと性能を生かすことができない。この並列数はたとえば GRAPE-9 と比べて一桁大きく、GRAPE と GPU の性質上大きく異なっている点である。GPU クラスタの各ノードには 4 基または 8 基の A100 GPU が搭載されており、利用者はジョブスケジューラを通じて必要な数の GPU を占有して計算を行う。

4.3 GRAPE の動作原理

本節ではまず GRAPE がどのような仕組みで動作しているのかについて説明する。その後で、GRAPE が汎用機よりも高い性能を得られる理由について説明する。GRAPE は図 4.2 に示すように、用途に応じて何種類ものハードウェアの系統が存在する。

基本的な動作原理はどれも同じであるが、伝統的には、奇数の番号がついている GRAPE は低精度計算用、偶数のものは高精度計算用ということになっている。得られる力の精度は、おおまかには前者で 2 ～ 3 桁、後者で 7 桁程度である。当然ながら、計算に必要な精度はシミュレーションの対象や目的によって異なるので、計算前にどのハードウェアを使うべきか考える必要がある。

GRAPE は、基本的には、粒子間重力相互作用の計算のみを高速に行うハードウェアである ([12], [11])。軌道積分など重力以外の計算は、GRAPE に接続した汎用計算機上で行う (図 4.3)。GRAPE は通常の汎用機がもつ汎用性、つまりプログラムさえ書けばいろいろな計算を行えるという性質、を犠牲にして高いコストパフォーマンスを得ている。粒子間重力しか計算できないかわりに、同世代の同価格の汎用機にくらべ、10^2 倍程度高速である。

なお、GRAPE は粒子間重力しか計算できないが、だからといって直接計算以外のアルゴリズムには役に立たないわけではないことに注意して欲しい。例えば

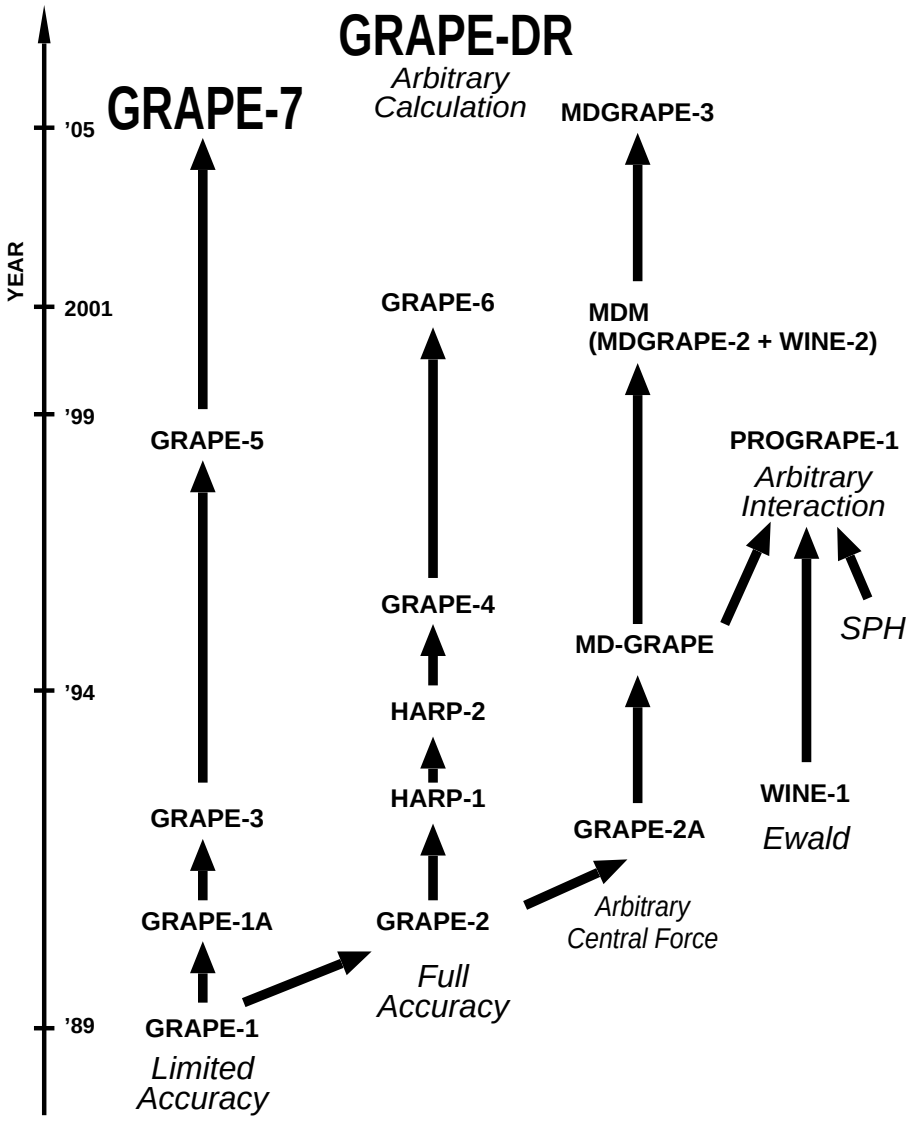


図 4.2: GRAPE とその仲間たち

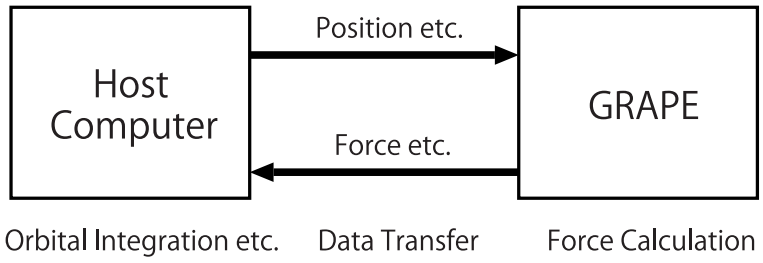


図 4.3: GRAPE の基本構造

ツリー法や P^{3}M 法などのアルゴリズムを使った場合でも粒子間重力の計算量は依然として大きいので GRAPE による高速化が可能である。

GRAPE の中核をなすのは重力計算用のパイプラインである。これは座標 $\boldsymbol{x}_j = (x_j, y_j, z_j)$ に置かれた質量 m_j の粒子が座標 $\boldsymbol{x}_i = (x_i, y_i, z_i)$ に置かれた粒子に及ぼす力 $\boldsymbol{f}_{ij}$ を計算するハードウェアである。複数の j に対する計算結果は内部のレジスタに積算されてゆく作りになっている。要するにこのパイプラインは

$$\boldsymbol{F}_i \equiv \sum_{j \neq i}^{N} \boldsymbol{f}_{ij} \equiv \sum_{j \neq i}^{N} \frac{m_j(\boldsymbol{x}_j - \boldsymbol{x}_i)}{(|\boldsymbol{x}_j - \boldsymbol{x}_i|^2 + \epsilon_i^2)^{3/2}} \tag{4.1}$$

を計算する。ここで ϵ_i はソフトニング値である。図 4.4 にパイプラインの概略図を示す。力を及ぼす側の粒子の座標 $\boldsymbol{x}_j$ と質量 m_j を外部から毎クロック 1 組ずつ

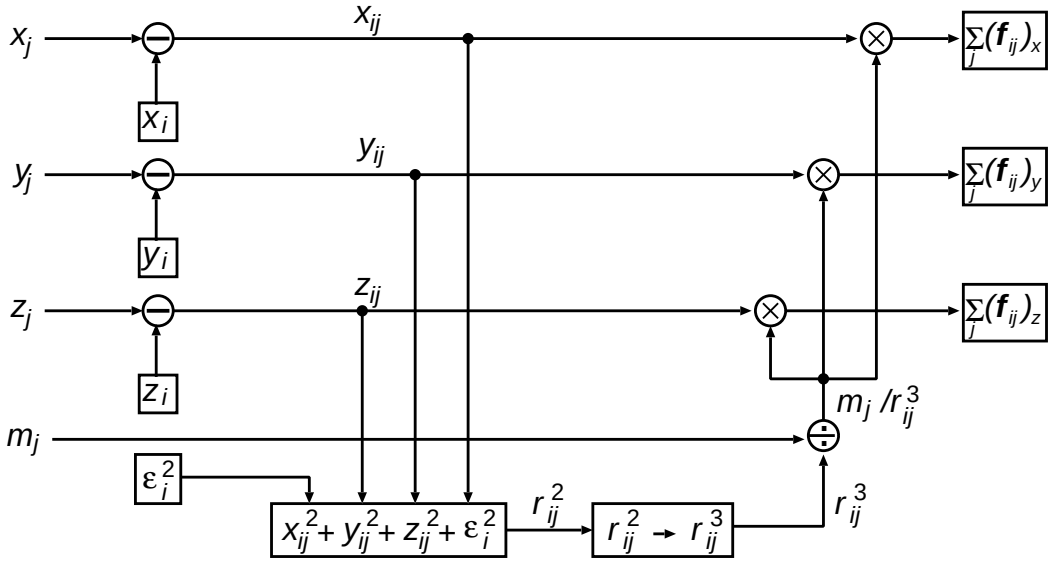


図 4.4: 重力計算パイプライン

供給すると、パイプライン中でデータが左から右へ流れつつ加工されてゆき、求まった計算結果がパイプラインの最終段で毎クロック積算されてゆく。力を受ける側の粒子の座標 $\boldsymbol{x}_i$ とソフトニング値 ϵ_i は、計算開始前にあらかじめ内部レジスタに設定しておく。

GRAPE とは基本的には、上記の重力計算用のパイプラインと、パイプラインに粒子データ $\boldsymbol{x}_j$ と m_j を供給するためのメモリをボード上にのせ、それらの制御やホスト計算機との通信を行う回路をつけたものである。ただし実際にはパイプラインは 1 本しかないわけではなく、複数本が専用 LSI チップの中に集積されており、そのようなチップ複数個とメモリが接続されてボード上に実装されている。GRAPE-9 ではカスタム LSI の代わりに再構成可能デバイス (FPGA: Field Programmable Gate Array) が使われており、その中に重力計算パイプライン回路やメモリ回路をプログラムすることで、GRAPE としての機能が実現されている。

現在、GRAPE システムで使われている GRAPE-9 の仕様では、チップ当たり 56 本のパイプラインを集積し、このチップがボード当たり 8 個搭載されている。

上記のハードウェアを使って粒子 n 個の間の重力を計算する手順は次のようになる。

1. 全 n 粒子の座標と質量を GRAPE 上のメモリに書き込む。
2. パイプライン本数 (npipe) 分の粒子の座標と質量、ソフトニング値を GRAPE 上のパイプラインレジスタに書き込む。
3. 計算開始信号を GRAPE へ送る。
4. 計算の終了を待つ。
5. 計算結果 (npipe 個の粒子へ他の粒子が及ぼす力) を GRAPE 上のパイプラインレジスタから読み出す。
6. 全ての粒子への力を計算し終った場合には計算を終了する。そうでなければ残りの粒子のうち npipe 個に対してステップ 2 から 5 を繰り返す。

この手順で力を求め、求めた力を使って軌道積分を行う、という作業を時間ステップごとに繰り返せば、直接計算法による N 体シミュレーションコードの出来上がりである。ただし、実際のコードではこの他に、座標および質量のスケーリングや、他のユーザと時分割して GRAPE を使うための排他制御、などの雑多な処理がいくつか必要になる。

以上で GRAPE の動作原理に関する説明は終りである。次はこのようなハードウェアがなぜ汎用機よりも高い性能を得られるのか、であるが、その理由は以下の 3 点にまとめられる。

- データの語長、フォーマットが最適化されている
- 重力相互作用の計算がパイプライン化されている
- 複数のパイプラインが並列化されている

まず第一の理由、データの語長、フォーマットの最適化について。汎用機では多くの場合、すべての変数を 64-bit 浮動小数点で表現し、これに対して演算を行う。しかし式 (4.1) 中のすべての演算にこれほどの語長やダイナミックレンジが必要なわけではないし、そもそも N 体シミュレーションでは力の精度がそれほど高くなくて良い場合が多いことも分かっている ([9], [7])。以上の理由から GRAPE では、必要なトランジスタ数を出来るだけ減らすように各演算器の語長や数値フォーマットを最適化している。このようなことが可能なのは GRAPE が専用機であり、計算すべき式があらかじめ決まっているからである。

次にパイプライン化について。GRAPE ではパイプライン化により、~ 10 個の演算を同時に実行している。多少いい加減になるが例を上げて説明しよう。あるクロックサイクルで j 番目の粒子がパイプラインの最終段にあり、この粒子からの力を積算中であったとする。ハードウェアがパイプライン化されていれば、このとき同時に $j+1$ 番目の粒子について m_{j+1}/r_{ij+1}^3 という割算を実行し、$j+2$ 番目の粒子については r_{ij+2}^2 を 1.5 乗して r_{ij+2}^3 を計算し、さらに $j+3$ 番目については $x_{ij+3}^2 + y_{ij+3}^2 + z_{ij+3}^2 + \epsilon_i^2$ を計算する、、、ということが行える。これは汎用機の場合に 1 クロック当りに行える演算がせいぜい 2 個程度であるのに比べると、大きな利点である。このようなことが可能なのもやはり GRAPE が計算すべき式があらかじめ決まっているからであり、またその式 (式 4.1) が異なる j に対して独立に計算できる形をしている (j について並列性をもっている) からである。

最後にパイプラインの並列化について。これはパイプラインをたくさん並べるというだけではうまくゆかない。パイプラインの本数が増えると、メモリからパイプラインへ粒子データを供給する速度が、パイプラインの動作に追い付かなくなってしまう。GRAPE では複数のパイプラインがひとつのメモリから同一の粒子データを受けとることでこの問題を回避している。これらのパイプラインは同じ j 番目の粒子から、互いに異なるインデクス $i, i+1, i+2, ...$ を持つ粒子への力を並列に計算する。こうすることで、メモリから粒子を供給する速度を上げることなくパイプライン数を増やせる。ここでは式 4.1 が異なる i に対して独立に計算できる形をしている (i について並列性をもっている) ことを利用している。さらには、仮想マルチパイプラインという方法を用いて必要なメモリバンド幅をさらに下げる工夫をしている。これは 1 本のパイプラインを時分割して使い、ひとつの粒子から異なる粒子への力を求める方法のことである。この方法は GRAPE-5/6 で使われている。この" 仮想パイプライン" という用語は GRAPE-9 でも使われているが、GRAPE-5/6 の場合と意味が異なるので注意が必要である。GRAPE-9 の場合は、一度に計算する i 粒子に対する粒子メモリ量を表すのに仮想パイプラインという言葉が使われている。

4.3.1 GRAPE-9 概要

GRAPE-9 は科学技術計算向けパイプライン回路を搭載した加速システムである。回路を書き換えることにより GRAPE-5 相当の重力計算回路と GRAPE-6 相当の重力計算回路を切り替えることができる。GRAPE-9 model800 には 8 枚のチップが搭載されているが、利用者からは 1 つのデバイスとして扱うことができる。

4.3.2 GRAPE-DR 概要

簡単に、最新の GRAPE-DR についてもふれておく。これまでの GRAPE では重力パイプライン回路が実装されていたが、その代わりに多数の汎用演算器を実

表 4.2: GRAPE-9 の主な仕様

プロセッサチップ	Altera CycloneIV GX (EP4CGX150) 8 個
ホストインターフェイス	Gen2 16 レーン PCI Express
重力相互作用計算 (GRAPE-6 相当)	80 本 149MHz 動作 (680 Gflops peak)
重力相互作用計算 (GRAPE-5 相当)	448 本 186MHz 動作 (3.17 Tflops peak)
オンボードメモリ	計 16 GB (各プロセッサチップに DDR2 メモリ 2GB を接続)

装したものが GRAPE-DR てある。そのため、重力計算のみならず、行列計算なども行うことが可能である。

4.3.3 GRAPE ライブラリ関数

ユーザは GRAPE ライブラリ関数を通して GPU や GRAPE にアクセスする。ライブラリ関数には高レベルのプログラミングインターフェースで GRAPE-5 互換である標準関数と、低レベルのプログラミングインターフェースである基本関数の 2 種類がある。標準関数を使った場合、ハードウェア周りの制御はライブラリが面倒をみるので、GRAPE-9 や GPU の枚数、デバイスの ID 番号などをユーザが意識する必要はない。GPU で GRAPE ライブラリ関数を使用するには、コード内で

`/usr/local/grape/grapepkg1.7/include/g5util.h`

をインクルードし、実行ファイル作成時に

`libcuda5, libcudart, libstdc++`

をリンクすればよい。以下では GPU と GRAPE ライブラリを使った重力計算の方法を、まず最も簡単な直接計算法のコードを例に説明する。その後で、近接粒子探索を行う方法について説明する。

標準関数を使って直接計算を行うコードの主要部分は次のようになる。

```
/* variable declaration */
static double mj[NMAX], xj[NMAX][3];
static double eps2, acc[NMAX][3], pot[NMAX];
double xmax = 10.0, xmin = -10.0, mmin = 0.001;
int i, n, nn, step, nstep;
int npipes = g5_get_number_of_pipelines();
.................
```

```
g5_open();
g5_set_range(xmin, xmax, mmin);
for (step = 0; step < nstep; step++) {
    /* force calculation */
    g5_set_jp(0, n, mj, xj);
    g5_set_eps2_to_all(eps2);
    g5_set_n(n);
    for (i = 0; i < n; i += npipes) {
        nn = npipes;
        if (i + nn > n) {
            nn = n -i;
        }
        g5_calculate_force_on_x(xj+i, acc+i, pot+i, nn);
    }
    /* orbital integration */
    ................
}
g5_close();
```

すべての関数は g9_ではなく g5_で始まっている。これは GRAPE-5 との互換性から、そのように名前が付けられているので、注意のこと。

g5_get_number_of_pipelines() は仮想パイプライン数 (通常は 256) を返す。g5_open() で GRAPE-9 を使う権限を得る。この関数は権限を得るまで返ってこない。つまり自分以外のプロセスが GRAPE-9 を使用中の場合には、使い終えるまで待ち続ける。g5_set_range() はシミュレーションで使用する座標と質量の範囲指定を行う。詳しくは後述する。g5_set_jp() は全粒子の座標と質量を GRAPE-9 上のメモリへ書き込む。g5_set_eps2_to_all() は全てのパイプラインに同一のソフトニング値の 2 乗 eps2 を設定する (別個のソフトニング値を設定することも可能)。g5_set_n() は粒子数 n を GRAPE-7 へ送る。g5_calculate_force_on_x() は位置 x[i]...x[i+nn-1] にある nn 個の粒子へ自分以外の n-1 個の粒子が及ぼす加速度の合計を計算し、acc[i]...acc[i+nn-1] へ返す。同時にポテンシャル値も pot[i]...pot[i+nn-1] へ返す。g5_close() は使い終えた GRAPE-7 の使用権を解放する。どのくらいの時間解放せずに使用するかは使用者の良心に任せられる。おおまかな目安として、1 分のオーダで解放するよう心がけたい。

説明を先送りした g5_set_range() であるが、この関数は粒子座標の最大値、最小値と、質量の最小値を設定する。座標の最大、最小値には、シミュレーション中に粒子がとりうる座標値の最大値、最小値よりも 2 倍以上大きな値を設定する。質量の最小値には、すべての粒子の質量が、最小値の整数倍で充分精度よく近似できる程度に小さな値を設定する。例えば質量 1.0 の粒子だけからなる系のシミュレーションであれば 1.0 を設定しておけばよい。質量 1.0 の粒子と 1.3 の粒子か

らなる 2 成分系の場合には 0.1 を設定する必要がある。仮に 0.2 に設定したとすると、質量 1.3 は GRAPE-9 の内部では「1.3 に最も近い 0.2 の整数倍の値」、つまり 1.2 として扱われ、1 割程度の誤差を生じる。

このような面倒な設定が必要な理由は、回路規模を減らすために GRAPE-9 内部で変則的な数値フォーマットを用いていることによる。例えば座標値は 32-bit 固定小数点形式で表現されており、ホスト計算機上でユーザが使用している 64-bit 浮動小数点形式ほどのダイナミックレンジが無い。そのためライブラリでは、ユーザに渡された座標値を 32-bit 固定小数で扱える範囲にスケールしてから GRAPE-9 へ送信している。このスケーリングに g5_set_range() で指定した最大値、最小値を用いるのである。g5_set_range() の使用方法を誤ると計算結果に大きな誤差をもたらし得る。より詳細な議論が [8] にあるので一読をお勧めする。

次に近接粒子探索を行う方法について説明する。近接粒子探索のためには、近接粒子を格納するためのバッファを用意し、上記のコードの i に関するループを次のように変更する。

```
/* variable declaration */
static int nblist[NPIPESMAX][NBMEMSIZE], nnb[NPIPESMAX];
int ii, nbof;
double h = 0.01;
.................

g5_set_h_to_all(h);
.................
    for (i = 0; i < n; i += npipes) {
        nn = npipes;
        if (i + nn > n) {
            nn = n -i;
        }
        g5_calculate_force_on_x(xj+i, acc+i, pot+i, nn);
        nbof = g5_read_neighbor_list();
        if (nbof == 1) {
             fprintf(stderr, "NB list overflow\n");
        }
        for (ii = 0; ii < nn; ii++) {
             nnb[ii] = g5_get_neighbor_list(ii, nblist[ii]);
        }
    }

.................
```

変数 nblist[NPIPES][NBMEMSIZE] と nnb[NPIPESMAX] は順に、各粒子に対して求まった近接粒子のインデクスと、その個数を格納するバッファである。NPIPES と NBMEMSIZE はそれぞれ、ボード当りの仮想パイプライン数と、求め得る近接粒子の最大数で、それぞれ、g5_get_number_of_pipelines と g5_get_nbmemsize で求められる。

g5_set_h_to_all() では、「近接している」ということを定義するための距離を設定する。つまり、距離 h 以内にある粒子が近接粒子と判定される。ここでは全てのパイプラインに同じ h の値を設定しているが、別個に指定することも可能である。

以上の準備をしたうえで g5_calculate_force_on_x() を呼ぶと、重力計算のついでに近接粒子探索が行われ、その結果が GRAPE-9 内の近接粒子メモリに保存される。これを g5_read_neighbor_list() によってホスト計算機に回収する。計算中に近接粒子メモリが溢れた場合には、関数は返り値として 1 を返す。正常終了時は 0 を返す。回収した結果はパイプライン毎に g5_get_neighbor_list() で取り出す。求まった近接粒子の個数は関数の返り値として得られる。

近接粒子メモリが溢れた場合には h の値を小さくして計算し直すなどの処理を行う必要がある。では、何粒子までなら保存できるのかであるが、その答えは「プロセッサー内の 1 パイプライン当たり 20 個、model 800 では 8 プロセッサーあるので $20 \times 8 = 160$ 個」となる。少々分かりににくいので補足をする。GRAPE-9 では、1 つの i 粒子に作用する重力を計算するのに、プロセッサー FPGA 間では j 並列の計算を行っている。すなわち、n 粒子からの力を計算するときには、model 800 では、それぞれのプロセッサー FPGA には $n/3$ 個ずつ粒子を分けて保存し、それらの j 粒子から受ける力を独立に計算して、最後に足し合わせる、ということを行っている。近傍粒子探査についても、プロセッサー間のパイプラインが独立に行う。そのため、近接粒子メモリが溢れは各プロセッサーのパイプライン毎に発生し、最悪の場合、20 個で溢れることもある。従って、より多くの近傍粒子を求めるためには、空間的に近い粒子は一つのプロセッサーに固まらないように分散して保存するなどの工夫が必要になる。

以上で GRAPE ライブラリ関数の使用方法に関する説明は終りである。直接計算法の実際に動作するコードが g00.cfca.nao.ac.jp の
/opt/muv/grape/grapepkg1.7.1/sample/direct/direct.c
に、そのコードに近接粒子探索のコードを付加したものが同じディレクトリの directnb.c にある。また、全ライブラリ関数のリファレンスが
/opt/muv/grape/grapepkg1.7.1/doc/g5user-j.pdf
および
/opt/muv/grape/grapepkg1.7.1/doc/g5nbuser-j.pdf
にある。必要に応じて参照のこと。

4.4 応用: GRAPE 上のツリー法

ここまでで GRAPE の動作原理と最も簡単な使い方についての説明を終えた。本節では GRAPE の直接計算法以外への使用例として、ツリー法への応用について説明する。講義時間の都合上、ツリー法について全く初めて触れるという方には説明が不親切になってしまっているかも知れないが、せっかくなのでこの際に大まかな概念だけでも理解して頂ければと思う。

節 4.4.1 では GRAPE を使用しない通常のツリー法の原理を説明する。次に節 4.4.2 で GRAPE への実装法について説明する。

4.4.1 ツリー法

ツリー法 ([1], [2]) は遠くの粒子をまとめて扱うことで計算量を減らすアルゴリズムである。遠くにある粒子ほど大きなかたまりにまとめる。まとめるために階層的な木構造を用いるのでツリー法と呼ばれる。まとめた粒子からの重力は、それらの粒子の重心に置いた仮想的な粒子からの重力で近似する。より精度の高い近似を行いたい場合には、多重極展開を用いる。

以下にもう少し具体的な計算手順を示す。なお説明は簡単のため 2 次元の場合について行う。

まずは下準備としてツリー構造を構築する。全ての粒子を含むような、充分に大きな正方形 (以降ルートセルと呼ぶ) を考える。これを 4 つの小さな正方形 (子セル) に分割する。そして、これら 4 つの子セルそれぞれを、さらに小さな 4 つの孫セルに分割する。この手順をセル内部の粒子が 1 個以下になるまで繰り返す。こうして構築したセルの階層構造 (ツリー構造) により、粒子の密度に応じた空間分割が行える (図 4.5)。

次に全てのセルについて、セル内のすべての粒子の質量の総和および重心を求める。あるセルの質量と重心はその子セルの質量と重心から計算できる。この事実を利用してツリー構造の最下層にあるセルからルートセルに向かって順に計算してゆけばよい。

ここまでで下準備は終りで、いよいよ力の計算にはいる。まず、ある粒子へのあるセルからの力を、次のように再帰的に定義する。

- セルが粒子から充分遠くにある場合、セルの重心からの力
- 充分遠くにはない場合、セルの子セルからの力の総計

このように粒子へのセルからの力を定義しておけば、各粒子への全系からの力は「ルートセルからの力」として求められる。なお、セルが充分遠くにあるかどうかの判定には次の条件を用いる。

$$\frac{l}{d} < \theta \tag{4.2}$$

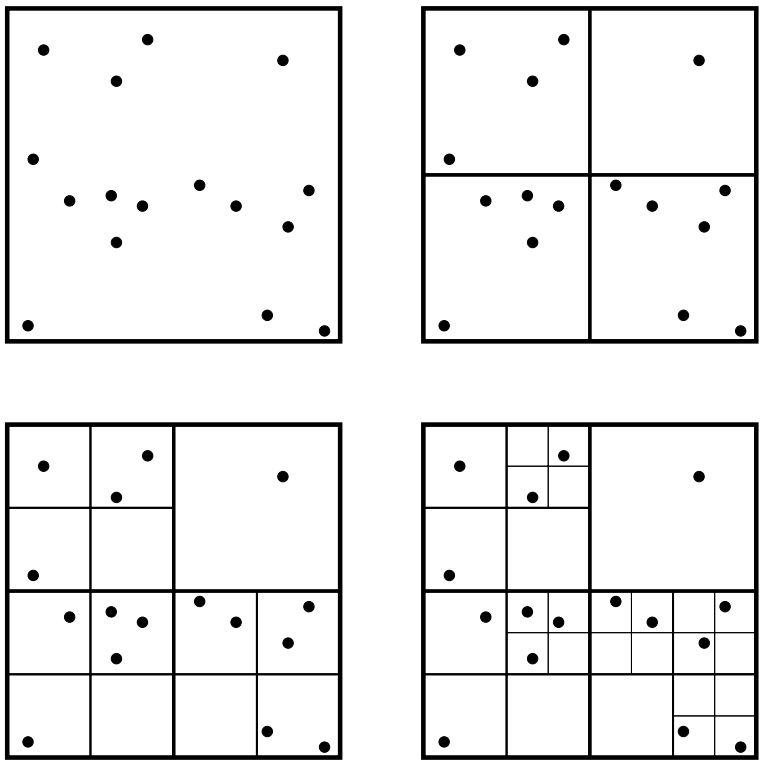


図 4.5: ツリー構造の構築

ここで l はセルの一辺の長さ、d はセルの重心から粒子までの距離である。また θ は精度を制御するパラメータで、見込み角と呼ばれる。これはユーザが与える定数である。θ の値を小さくすれば、より細かなセルからの力を計算することになり、精度は高くなるが計算量が増す。典型的には $0.3 \sim 1.0$ 程度の値を用いることが多い。

以上で GRAPE を使わないツリー法の説明は終りである。多重極展開を用いた場合の説明は省いたが、セルが持つデータとして重心の座標と質量の代わりに多重極展開系数をつかうだけで、ほぼ同様の手順で計算できる。また 3 次元の場合の説明も省略したが、これもセルを 4 つではなく 8 つに分割する点以外は同様の手順で計算できる。

4.4.2 GRAPE への実装

ツリー法を GRAPE に実装するには、セルからの力を GRAPE で計算できればよい。セルからの力は重心に置いた仮想的な粒子からの力として表現されるので、これはそのまま GRAPE を使って計算できる (多重極展開を使った場合にも GRAPE を使って計算することは可能だが、説明が若干マニアックになるのでここでは省略する)。

しかし節 4.4.1 のアルゴリズムをそのまま用いたのでは GRAPE をあまり効率良く動作させることができない。節 4.3 で述べたように、GRAPE は計算すべき式 4.1 の i および j に関する並列性を利用して汎用機よりも高い性能を得ている。しかし節 4.4.1 のアルゴリズムのままでは i, j どちらの並列性も利用できない。ある粒子へどのセルが力を及ぼすのかは前もって分かっているわけではない。式 4.2 による判定を繰り返しながらツリー構造をたどっていって、ひとつずつセルが求まり、その都度ひとつのセルからの力を計算する。このためアルゴリズムには j 並列性が無い。また、粒子に力を及ぼすセルは粒子毎に異なるため、同じセルから異なる粒子への計算を並列に行うことは出来ない。つまり i 並列性も無い。

上記の問題を解決して GRAPE を効率良く動作させるためには、アルゴリズムに若干の変更が必要となる。以下でその変更について説明する。

変更は二つのステップからなる。第一のステップではまず、粒子にどのセルが力を及ぼすのかが前もって分からない、という問題を解決する。このためにはツリー構造をたどりながら力を計算するのではなく、たどる時には力を及ぼすかどうかの判定だけを行い、及ぼすと判定したセルをリストアップすればよい。ツリー構造をたどり終えた後で、作成したリスト内のセルからの力を計算するために GRAPE を呼び出せば、j 並列性を生かした計算を行える。

最初のステップの変更で重力計算はある程度速くなる。しかしそれと同程度の計算量を持つツリー構造をたどる部分の計算は全く速くならない。第二のステップではこの部分の計算量を減らし、それと同時に、力を及ぼすセルが粒子毎に異なる、という前述の問題を解決する。このためには最初のステップのところで説明したセルのリストを、複数の粒子に対して使いまわすことができれば良い。そうすればリスト作成回数が減り、また i 並列性も生かせるようになる。互いに近くにある粒子同士ならば、それらへ力を及ぼすセルもだいたい似たようなものになるはずなので、式 4.2 の条件を少し厳しくすればリストを使いまわせそうである。

実際には「互いに近くにある粒子」を決めるためにはツリー構造を使う。ツリーをルートセルから降りてゆき、セル内の粒子数があらかじめ決めておいた定数 ($n_{\rm crit}$ とする) 以下になったら、そのセル内の粒子を近傍粒子として扱うことにする。そして力を及ぼすセルのリストは、粒子ごとにではなく、この近傍粒子ひとかたまりにつき一つだけ作成する。その際、式 4.2 における距離 d としては、近傍粒子の属するセル内の任意の点から、力を及ぼす側のセルの重心までの距離のうち最小のものを用いる。近傍粒子同士が互いに及ぼす重力は直接計算する。

近傍粒子の個数を増せばリスト作成の回数を減らせる。また i 並列性を生かせるようになり、GRAPE は効率良く動作する。一方、近傍粒子同士の直接計算のぶんだけ重力計算の量は増す。従ってリスト作成にかかる時間と重力計算にかかる時間の和が最小になるような、近傍粒子数の最適値が存在するはずである。

以上でツリー法の GRAPE への実装に関する説明は終りである。第一、第二のステップで説明した変更方法は、それぞれ [6], [3] による。GRAPE 上のツリー法に関するより詳しい説明は [10] を参照のこと。

4.5 まとめ

GPU クラスタの概要と、GRAPE ライブラリの元となる GRAPE の動作原理と基本的な使用方法について説明した。あとは各自の使用目的に応じた文献を当たり、より詳しい知識を得て頂きたい。

関連図書

[1] Appel A. W. 1985, SIAM J. Sci. Stat. Comput 6, 85

[2] Barnes, J. E., Hut, P. 1986, Nature, 324, 446

[3] Barnes J. E. 1990, J. Comp. Phys. 87, 161

[4] Brieu P. P., Summers F. J., Ostriker J. P. 1995, ApJ, 453, 566

[5] Fukushige T., Taiji M., Makino J., Ebisuzaki T., Sugimoto D. 1996, ApJ 468, 51

[6] Hernquist L. 1987, ApJS 64, 715

[7] Hernquist L., Hut P., Makino J. 1992, ApJL 402, 85

[8] Kawai, A., Fukushige, F., Makino, J., Taiji, M. 2000, PASJ, 52, 659

[9] Makino J., Ito T., Ebisuzaki T. 1990, PASJ 42, 717

[10] Makino, J. 1990, PASJ, 43, 621

[11] Makino J., Taiji M. 1998, Scientific Simulations with Special-Purpose Computers — The GRAPE Systems (John Wiley and Sons, Chichester)

[12] Sugimoto D., Chikada Y., Makino J., Ito T., Ebisuzaki T., Umemura M. 1990, Nature 345, 33

[川井敦/台坂博/押野翔一]

第5章　実習1　N体シミュレーションプログラムの実装: cold collapse を例として

5.1　目的

この実習では cold collapse 問題を例として、N 体シミュレーションの一通りの作業を実際に体験する。具体的には、C 言語を使って無衝突系の N 体シミュレーションで広く使われている時間積分法である leap-frog 法を実装する。相互重力の計算には直接計算法を使う。

5.2　cold collapse 問題

今回取り上げる cold collapse 問題は簡単に言えば、初期に粒子の速度分散が小さい (温度が低い) 粒子系が自己重力で collapse した後にどのような粒子分布になるか、という問題である。cold collapse は銀河形成、特に楕円銀河の形成過程の素過程として重要であると考えられている。例えば、de Vaucouleurs の法則として知られる楕円銀河の表面輝度 (密度) 分布、$I(r) \propto r^{1/4}$ は cold collapse によって実現されるという話がある。興味がある人は例えば

van Albada 1982, MNRAS, 201, 939

Merritt, Aguilar 1985, MNRAS, 217, 787

Aguilar, Meriitt 1990, ApJ, 354, 33

を見て欲しい。

シミュレーションでは初期条件として力学平衡にない速度分散の小さな粒子分布 (ビリアル比が 1/2 以下) を準備し、その後の粒子分布の時間発展を調べることになる。

collapse のダイナミカルな時間スケールは系の自由落下時間である。一様密度球の粒子系の場合の自由落下時間は

$$t_{\rm ff} = \left(\frac{3\pi}{32\rho_0 G} \right)^{1/2} \tag{5.1}$$

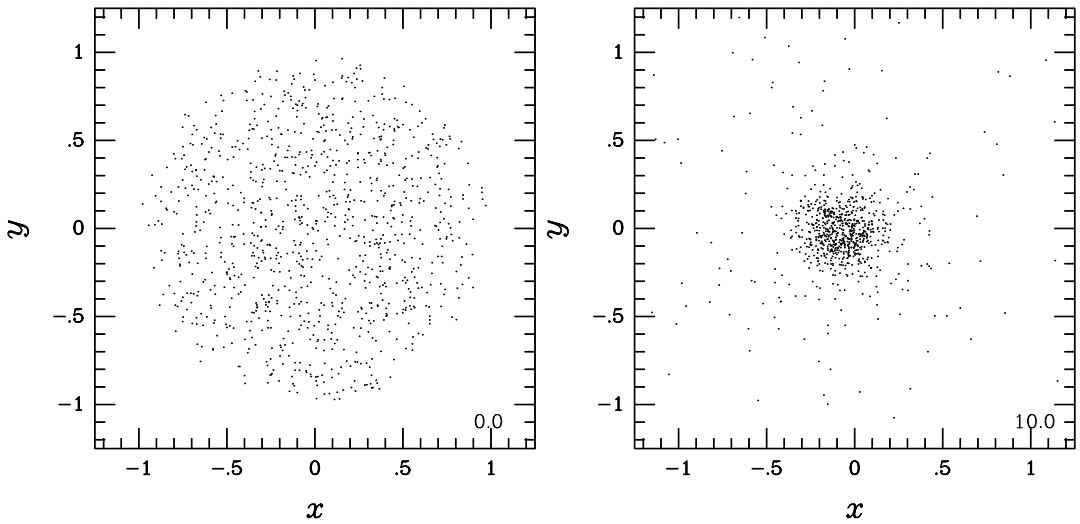


図 5.1: cold collapse の N 体シミュレーションの例

となる (導いてみよ)。ここで ρ_0 は球の初期密度である。

cold collapse の N 体シミュレーションの例を図 5.1 に示す。これは粒子数 $N = 1024$、初期ビリアル比 $r_{\rm V} = 0.1$ の一様球の初期状態 $t = 0$(左) とほぼ定常状態に達した時刻 $t = 10$(右) での x-y 面上でのスナップショットである。スナップショットとはある時刻の粒子分布のことである。

5.3 プログラミングの準備

5.3.1 プログラミング方針

プログラムは C 言語を使って書くことにしよう。どうしても Fortran が使いたい場合は Fortran を使ってもいいが、この機会に C 言語に触れてみよう。プログラムの例は C 言語で与えることにする。

プログラミングの流儀はいろいろあるが、今回は最低でも以下の点について気をつけてプログラミングを行う。

- グローバル変数は使わない。
- 変数名はわかりやすく。
- なるべく関数 (サブルーチン) を用いてプログラムをブロック化する。
- 関数はなるべく短く。
- なるべくコメントを書く。

これらを守らないと不健康なプログラムになってしまうことが多いので気をつけること。

この実習のプログラムはあまり長くはならないはずなので、1 つのファイルにまとめて書くことにしよう。さらにプログラムのコンパイルには make は使わないことにする。もちろん、使える人は make を使ってかまわない。

5.3.2 テンプレートファイル

必要な設定が書いてあるプログラムのテンプレートが/home/nbody00/practice1/template.cにある。このファイルを各自の作業ディレクトリに適当な名前でコピーし、そのファイルにプログラムを書いていくことにする。作業ディレクトリは/cfca-work/nbodyNN/以下に作成すること。/home/nbodyNN/以下も利用可能だが低速である。

```
$ cp /home/nbody00/practice1/template.c collapse.c
```

とすれば自分のディレクトリに collapse.c という名前のファイルがコピーされる。template.c の中身は以下のようになっている。

```
/************************************************************************
 template.c: template file for practice 1
************************************************************************/

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

#define NMAX 16384
#define SQR(x) ((x)*(x))

#define VMAX 1.25 /* limit of variables in window */
```

ここでは必要なヘッダファイルを読み込み、マクロを定義している。この部分の数値については指示があるまで変更しないこと。詳しい内容については実習で説明する。

5.3.3 コンパイルと実行

プログラムの名前が collapse.c だとすると、コンパイルは

```
$ cc collapse.c -o collapse -O2 -lm
```

とすればよい。
　可視化のためのグラフィックライブラリを使う場合のコンパイルは上記とは違うが、それは可視化の章で触れることにする。
　プログラムの実行は

```
$ ./collapse
```

とする。

5.4　N 体シミュレーションの流れ

　N 体シミュレーションの典型的なフローチャートを図 5.2 に示す。N 体シミュレーションは大きく分けると

初期設定 シミュレーションパラメータの設定と初期粒子分布の設定。

時間発展 重力相互作用を計算して、それをもとに粒子の軌道を時間積分し、必要な解析やデータの出力を行う。

の 2 つの部分に分けられる。
　以下、フローチャートのそれぞれの部分について、cold collapse シミュレーションの場合についてどのようなことをするか具体的に示していく。プログラミングは次の 5 段階に分けて行う。

1. パラメータ設定
2. 初期分布 (等質量等温一様球分布) 設定
3. 相互重力計算 (直接計算法)/時間積分 (固定タイムステップのリープフロッグ法)
4. グラフィックライブラリを用いた可視化
5. リアルタイム解析

スナップショット出力に関しては実習 2 で行う。

5.5　パラメータの設定

(1) 標準入力からソフトニングパラメータ、タイムステップ、シミュレーションの終了時刻、データ解析/出力の時間間隔を入力するようにし、入力された値を表示せよ。

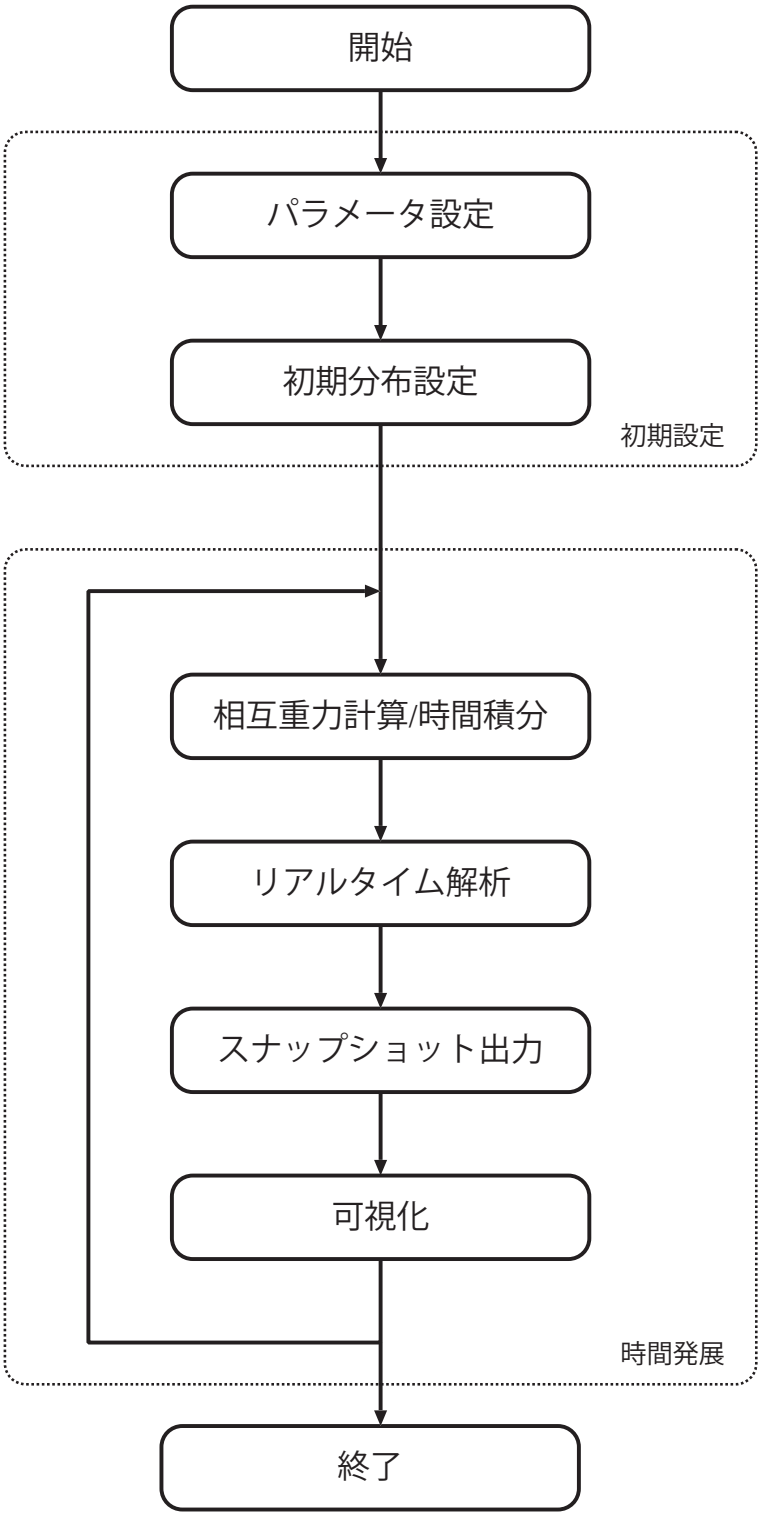


図 5.2: N 体シミュレーションのフローチャート

当り前だが設定するパラメータはどのような方法で N 体シミュレーションをするかによって決まる。今回のような固定タイムステップを使うリープフロッグ法を使う場合、シミュレーションに必要なパラメータは少なくとも 4 個ある。

ϵ	ポテンシャルソフトニングパラメータ
Δt	時間積分のタイムステップ
$t_{\rm end}$	シミュレーションの終了時刻
$t_{\rm out}$	データ解析/出力の時間間隔

これらのパラメータは普通プログラムの中で直接値を設定せずに、ファイルや標準入力から読み込むようにする。C 言語では標準入力から値を読み込む関数に `scanf` がある。倍精度変数 `v` に標準入力から値を入力するには例えば

```
double v;

fprintf(stderr, "v = ");
scanf("%lf", &v);
fprintf(stderr, "v = %g\n", v);
```

とすればよい。入力したら、確認のためにどのような値が入力されているか表示させるようにするとよい。

以下ソフトニングパラメータとタイムステップについてどのように設定すればいいか述べる。

5.5.1 ソフトニングパラメータ

ソフトニングパラメータは粒子間の重力ポテンシャルが粒子の相対距離 r が $r \to 0$ のときに発散しないように導入する。よく使われるポテンシャルソフトニングはプラマーソフトニングと呼ばれるもので $1/r_{ij}$ の重力ポテンシャル ϕ_{ij} を

$$\phi_{ij} = -\frac{Gm_i m_j}{(r_{ij}^2 + \epsilon^2)^{1/2}} \tag{5.2}$$

のようになます。

ソフトニングパラメータをどのくらいの大きさにとるべきかは難しい問題である。cold collapse の場合は、系がもっとも collapse したときに結果に大きく影響しない程度に小さく、というのが理屈である。しかし、どこまで collapse するかはビリアル比や密度分布にもよるので、実際やってみないとよくわからない。とりあえず、系の大きさの 1/100 くらいにしておこう。

5.5.2 タイムステップ

時間積分のタイムステップはシミュレーションの計算精度を決める重要なパラメータである。今回は固定タイムステップを使う。タイムステップの大きさは典型的な粒子の速さを v として、

$$\Delta t < \frac{\epsilon}{v} \tag{5.3}$$

となるようにする。さらにタイムステップは $1/2^n$ になるようにしておく。こうしておくと時刻へのタイムステップの加算での丸め誤差はなくなるし、いろいろと便利である。

実際のシミュレーションではタイムステップを変えて結果が変わらないか確認しながら、最適なタイムステップを選ぶことになる。

5.6 初期分布の設定

(2) 粒子数 N、ビリアル比 $r_{\rm V}$ の等質量の粒子からなる等温一様球分布を作りなさい。N と $r_{\rm V}$ は標準入力から入力するようにしなさい。作ったら、正しい分布が得られているか確認しなさい。

粒子の初期分布を作るのは時間発展の部分と同じくらい、ときにはそれ以上に重要である。世の中には NEMO や starlab のような N 体シミュレーション用のツールキットがあり、それらには様々な初期分布を作るプログラムが含まれている。しかし、多くの場合、初期条件は自分で作ることになる。

5.6.1 プログラムの構造

プログラムは次のように設計しよう。

関数プロトタイプ宣言

```
void make_spherical_df(int n, double m[], double x[][3], double v[][3],
    double r_v, double eps2);
.
.


void make_spherical_df(
    int n, double m[], double x[][3], double v[][3],
    double r_v, double eps2)
{
```

```
    粒子分布を作る

}
```

粒子分布を作るのに必要な関数

.

.

```
int main(void)
{

    /* number of particles */
    int n;
    /* particle data */
    static double m[NMAX], x[NMAX][3], v[NMAX][3], a[NMAX][3];
    /* system energy, Virial ratio */
    double e, e_ini, r_v;
    /* current time, timestep, end time, data interval */
    double t, dt, t_end, t_out;
    /* squared softening parameter */
    double eps2;

    必要な変数の設定

    make_spherical_df(n, m, x, v, r_v, eps2);

    確認のための粒子分布の出力

    return(0);

}
```

ここでは粒子分布を作る関数を make_spherical_df とした。引数は例である。粒子のデータなどの大きな配列は static 宣言をして恒久的な変数としたほうが無難である。

関数プロトタイプ宣言では関数の型とその引数の型や個数を定義する。これらはコンパイラが関数の型とその引数の個数と型のチェックを行うために必要である。

5.6.2 単位系

シミュレーションの単位系は都合のいいようにとればいいが、N 体シミュレーションでは、「標準」N 体単位系というものがよく使われている。これは重力定数 $G = 1$、系の質量 $M = 1$、系のエネルギー $E = -1/4$ というものである。今回は「標準」N 体単位系は使わない。今回は初期分布を一様球とするので、その球の半径 R を長さの単位としよう。つまり、単位系は

$$G = 1,\ M = 1,\ R = 1$$

とする。この場合、1 粒子の質量は $m = M/N$ となる。

この単位系の場合、式 (5.1) の自由落下時間は $t_{\rm ff} \simeq 1.1$ となる。確かめてみよ。

5.6.3 粒子分布の作り方

今回は粒子は等質量とする。使用する単位系では $m = 1/N$ となる。これをプログラムで、`1/n` と書くと整数演算だと解釈され、`n` が 2 以上のときに 0 となってしまう。浮動小数点演算であるとことを示すために `1.0/n` と書く必要があることに注意。

今回は等温一様球を初期条件とする。粒子数を N とすると粒子の分布関数は極座標を用いて

$$f(r,\theta,\phi)\mathrm{d}r\mathrm{d}\theta\mathrm{d}\phi = \frac{3N}{4\pi}r^2\sin\theta\mathrm{d}r\mathrm{d}\theta\mathrm{d}\phi \tag{5.4}$$

となる。

一様球を作る方法はいろいろあるが、ここではもっとも簡単なアルゴリズムを紹介する。

1. x, y, z を (-1,1) の一様乱数で与える。
2. $r = (x^2 + y^2 + z^2)^{1/2} < 1$ を満たせば採用する。満たさないときは 1 に戻る。

これを粒子数だけ繰り返せば一様球が作れる。一様乱数を作るには、C 言語では例えば [0,1) の範囲の一様乱数を返す関数 `drand48` を利用すればよい。

速度分布はマクスウェル分布

$$f(v_x,v_y,v_z)\mathrm{d}v_x\mathrm{d}v_y\mathrm{d}v_z = N\left(\frac{1}{2\pi\langle\sigma^2\rangle}\right)^{3/2}\exp\left[-\frac{1}{2}(v_x^2+v_y^2+v_z^2)/\sigma^2\right]\mathrm{d}v_x\mathrm{d}v_y\mathrm{d}v_z \tag{5.5}$$

となる。ここで σ は 1 次元の速度分散である。速度分布については以下の平均 0、分散 1 のガウス分布を与える次の関数を使って欲しい。

```
double gaussian(void)
/* Gaussian with mean = 0.0 and dispersion = 1.0 by Box-Muller method */
{

    double x, y, r2;
    double z;

    do{
        x = 2.0*drand48() - 1.0;
        y = 2.0*drand48() - 1.0;
        r2 = x*x + y*y;
    }while(r2 >= 1.0 || r2 == 0.0);

    z = sqrt(-2.0*log(r2)/r2)*x; /* discard another Gaussian (....*y) */

    return (z);

}
```

これは Box-Müller 法というものである。これは/home/nbody00/practice1/gaussian.c にあるのでコピーして使ってほしい。この関数の使い方は例えば

```
    for(i=0;i<n;i++){
        for(k=0;k<3;k++){
            v[i][k] = sigma*gaussian();
        }
    }
```

というふうにする。こうすると、v[i][k] は平均 0、分散 sigma のガウス分布になる。

5.6.4 初期速度分散

速度分布を作るところで 1 次元の速度分散 σ が必要になる。これは与えられたビリアル比をもとに計算することになる。ビリアル比 $r_{\rm V}$ は

$$r_{\rm V} = \frac{K}{|W|} \tag{5.6}$$

と定義される。ここで K、W は粒子系の運動エネルギーとポテンシャルエネルギーでそれぞれ

$$K = \sum_{i=1}^{N} \frac{1}{2} m_i v_i^2 \tag{5.7}$$

$$W = -\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\frac{m_i m_j}{(r_{ij}^2+\epsilon^2)^{1/2}} \tag{5.8}$$

である。W の二重ループの添字に注意。これらから粒子が等質量の場合

$$\sigma = \left(\frac{2r_{\rm V}|W|}{3M}\right)^{1/2} \tag{5.9}$$

となる。初期速度分散を計算する手順は

1. $r_{\rm V}$ を入力する。
2. 式 (5.8) を使って m, x, y, z から W を計算する。例えば

```
W = 0.0;
for(i=0;i<n-1;i++){
    for(j=i+1;j<n;j++){
      r2 = SQR(x[j][0]-x[i][0]) + SQR(x[j][1]-x[i][1])
   + SQR(x[j][2]-x[i][2]);
      W = W - m[i]*m[j]/sqrt(r2 + eps2);
    }
}
```

   となる。

3. 式 (5.9) を使って σ を計算する。例えば

```
sigma = sqrt(2.0*r_v*fabs(W)/3.0);
```

となる。粒子数 $N = 1024$、ビリアル比 $r_{\rm V} = 0.1$、ソフトニングパラメータ $\epsilon = 0.03125$ を設定した場合の速度分散は $\sigma = 0.198717$ となることを確認すること。

K や W を計算するときに、C 言語では粒子が N 個あるとすると、ループを回すときの粒子番号は 0 から $N-1$ までになることに注意すること。

5.6.5 粒子分布の確認

プログラムが書けたら、スナップショットを出力して、粒子分布が思い通りにできているか確認する必要がある。スナップショットを出力するには例えば、

```
for(i=0;i<n;i++){
    printf("%f %f %f\n", x[i][0], x[i][1], x[i][2]);
}
```

とすればよい。そしてプログラムを走らせて出力を次のようにファイルにリダイレクトする。

```
$ ./collapse > n1024.snp
```

まず、何か図を描くプログラムを使って粒子分布がどうなっているか見てみる。例えば gnuplot を使うことにする。

```
$ module load gnuplot
$ gnuplot
gnuplot> set size square
gnuplot> set yr[-1:1]
gnuplot> plot "n1024.snp" using 1:2
```

とすると図 5.1(左) と同じような x-y 面上のスナップショットを見ることができる。x-z 平面上でもどうなっているか見てみよう。gnuplot を終了するには以下のようにする。

```
gnuplot> quit
```

次に一様球になっているか確かめよう。一様球の場合、半径 r の球までに入っている粒子数 n_r は r^3 に比例するはずである。これを調べるには、スナップショットのファイルから r を計算し、それから n_r を求めればよい。これは C 言語でプログラムを書いてもいいが、awk や sort を使っても簡単に求められる。例えば

```
$ awk '{print sqrt($1^2+$2^2+$3^2)}' n1024.snp | sort -n |
  awk '{print $1, NR}' > r-nr.dat
  (紙面の都合で 2 行に書いているが実際は 1 行目に続ける)
```

とすればよい。こうすると `r-nr.dat` の 1 列目には r、2 列目には n_r が出力される (詳しくは awk や sort のマニュアルを参照して欲しい)。分布が正しく計算されていれば図 5.3 のような図が得られるはずである。

```
gnuplot> set size square
gnuplot> set logscale xy
gnuplot> set yr[1:1000]
gnuplot> plot "r-nr.dat" with lines
```

5.7 相互重力計算/時間積分

(3) 相互重力の直接計算法とリープフロッグ法を実装しなさい。実装したら積分誤差のタイムステップへの依存を確認せよ。

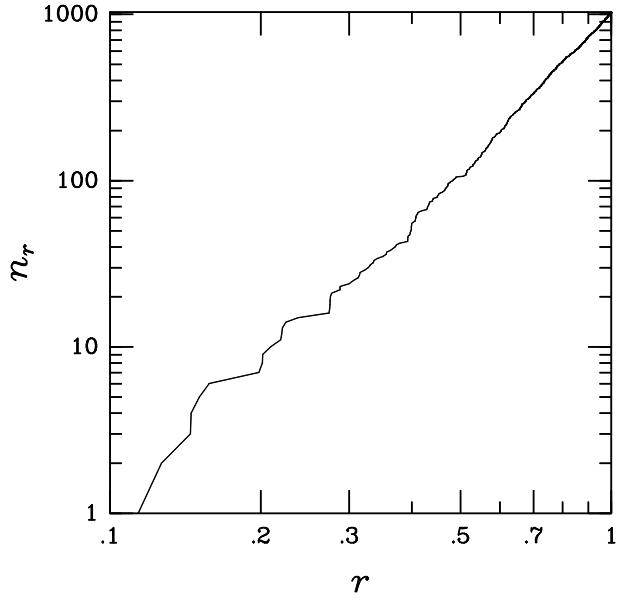


図 5.3: 粒子数 1024 の場合の一様球の $r - n_r$ 図

5.7.1 プログラムの構造

相互重力計算と時間積分の部分のプログラムの基本構造は次のようにしよう。

```
void calc_force(int n, double m[], double x[][3], double a[][3], double eps2)
{

    相互重力 (加速度) の計算 (詳細については 5.7.2)

}


void leap_frog(
    int n, double m[], double x[][3], double v[][3], double a[][3],
    double dt, double eps2)
{

    時間積分 (詳細については 5.7.3)

}


double calc_energy(int n, double m[], double x[][3], double v[][3], double eps2)
```

```
{

  運動エネルギー (K) の計算
  ポテンシャルエネルギー (W) の計算
  (詳細については 5.7.4)

  return (K + W);
}


int main(void)
{

    /* number of particles */
    int n;
    /* particle data */
    static double m[NMAX], x[NMAX][3], v[NMAX][3], a[NMAX][3];
    /* system energy, Virial ratio */
    double e, e_ini, r_v;
    /* time, timestep, end time, data interval */
    double t, dt, t_end, t_out;
    /* squared softening parameter */
    double eps2;

    /* setting simulation parameters */
    パラメータの設定

    /* initialization of particle data */
    make_spherical_df(n, m, x, v, r_v, eps2);

    /* calculate initial energy */
    e_ini = calc_energy(n, m, x, v, eps2);

    /* calculate initial acceralation */
    calc_force(n, m, x, a, eps2)

    while(t<t_end){

        /* realtime analysis */
```

```
        リアルタイム解析 (詳細については 5.9)

        /* visualization of particle data */
        可視化 (詳細については 5.8)

        /* time integration */
        leap_frog(n, m, x, v, a, dt, eps2);
        t += dt;

    }


    /* integration error check */
    e = calc_energy(n, m, x, v, eps2);
    printf("%e %e\n", dt, fabs((e - e_ini)/e_ini));

    return (0);

}
```

ここではリープフロッグ法による時間積分の関数を leap_frog、重力相互作用 (加速度) を計算する関数を calc_force とした。引数は例である。時間積分のループはいろいろな流儀があるがここでは while ループを使っている。

5.7.2 相互重力計算

粒子 i の運動方程式は

$$\frac{\mathrm{d}^2\boldsymbol{x}_i}{\mathrm{d}t^2} = \sum_{j\neq i}^{N} Gm_j \frac{\boldsymbol{x}_j - \boldsymbol{x}_i}{(|\boldsymbol{x}_j - \boldsymbol{x}_i|^2 + \epsilon^2)^{3/2}} \tag{5.10}$$

となる。相互重力計算は N 体シミュレーションの中でもっとも計算量の多い部分である。この計算は将来は GPU で行うがここではまずホストコンピュータの CPU で計算する。

今回はもっとも素直で基礎となる相互重力計算法である、直接計算法を使う。直接計算法は正直に式 (5.10) を計算するだけである。式 (5.10) を N 個の粒子について計算すると、相互作用の計算回数は N^2 になる。そこで少しでも計算量を減らすために作用反作用の法則を利用して全粒子の重力相互作用を同時に計算するようにする。すなわち、粒子 i と j についての 2 重ループを同じ粒子の組が 1 回しか実現されないように回し、そのとき計算される 2 粒子間の相対位置 $\boldsymbol{r} = \boldsymbol{x}_j - \boldsymbol{x}_i$ を使って、

粒子 i と j の 2 体相互作用 $\boldsymbol{a}_{ij}$ と $\boldsymbol{a}_{ji}$ をそれぞれ $\boldsymbol{a}_{ij} = m_j\boldsymbol{r}/|\boldsymbol{r}|^3$、$\boldsymbol{a}_{ji} = -m_i\boldsymbol{r}/|\boldsymbol{r}|^3$ として計算すればよい（分母の $|\boldsymbol{r}|^3$ にはソフトニングパラメータを含める必要があることに注意)。具体的には

```
for(i=0;i<n;i++){
    for(k=0;k<3;k++){
        a[i][k] = 0.0; /* 加速度の初期化 */
    }
}

for(i=0;i<n-1;i++){
    for(j=i+1;j<n;j++){
        r[3]、r3inv(r の 3 乗の逆数) の計算
        for(k=0;k<3;k++){
            a[i][k] +=  m[j]*r[k]*r3inv;
            a[j][k] += -m[i]*r[k]*r3inv;
        }
    }
}
```

とすればよい (わかりやすくするために計算量が少し多い書き方をしている)。これだと相互作用の計算回数は $N(N-1)/2$ ですむ。

5.7.3 リープフロッグ法

リープフロッグ法は無衝突系の N 体シミュレーションで広く使われている 2 次の積分法である。この方法は実は時間対称でかつシンプレクティックだったりして奥が深い積分法である。しかし、公式はとても簡単である。リープフロッグ法は以下のようになる。

$$\boldsymbol{v}_{1/2} = \boldsymbol{v}_0 + \boldsymbol{a}_0\frac{\Delta t}{2} \tag{5.11}$$

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \boldsymbol{v}_{1/2}\Delta t \tag{5.12}$$

$$\boldsymbol{v}_1 = \boldsymbol{v}_{1/2} + \boldsymbol{a}_1\frac{\Delta t}{2} \tag{5.13}$$

ここで 0 と 1 はそれぞれ時間ステップの最初と最後の値、1/2 はその中間の値であることを示す。

プログラムの中では

1. 式 (5.11) を使って $\boldsymbol{v}_0$、$\boldsymbol{a}_0$ から $\boldsymbol{v}_{1/2}$ を計算する。
2. 式 (5.12) を使って $\boldsymbol{x}_0$、$\boldsymbol{v}_{1/2}$ から $\boldsymbol{x}_1$ を計算する。

3. 式 (5.10) を使って $\boldsymbol{x}_1$ から $\boldsymbol{a}_1$ を計算する。

4. 式 (5.13) を使って $\boldsymbol{v}_{1/2}$、$\boldsymbol{a}_1$ から $\boldsymbol{v}_1$ を計算する。

5. 1 に戻る。

となる。具体的には

```
for(i=0;i<n;i++){
    for(k=0;k<3;k++){
        v_half[i][k] = v[i][k] + 0.5*a[i][k]*dt;
        x[i][k] += v_half[i][k]*dt;
    }
}

calc_force(n, m, x, a, eps2);

for(i=0;i<n;i++){
    for(k=0;k<3;k++){
        v[i][k] = v_half[i][k] + 0.5*a[i][k]*dt;
    }
}
```

となる (わかりやすくするために計算量とメモリ使用量が少し多い書き方をしている)。計算を始めるときは、時間積分のループに入る前に $\boldsymbol{x}_0$ から $\boldsymbol{a}_0$ を計算する必要があることに注意。

5.7.4 積分誤差の確認

リープフロッグ法の精度は 2 次である。すなわち、累積 (全離散化) 誤差は Δt の 2 乗に比例する。実際にタイムステップを変化させてある一定の時間まで積分し、累積誤差が Δt の 2 乗に比例していることを確かめよう。もし、そうなっていなかったら公式の実装が間違っている可能性がある。この誤差の確認はプログラムが正しいか確認するために重要である。積分公式を実装したら、いつも誤差評価をしなくてはいけない。

今回考えている問題には散逸は入っていないので系のエネルギーは本来保存するはずである。しかし、積分法の誤差、主に打切 (局所離散化) 誤差により、エネルギーにも誤差が発生している。リープフロッグ法ではある時刻でのエネルギー誤差はタイムステップの 2 乗に比例しているはずである。これを確認しよう。系のエネルギー E は

$$E = K + W \tag{5.14}$$

で与えられる。運動エネルギーとポテンシャルエネルギーの計算は具体的には以下のように書ける。

```
/* kinetic energy */
K = 0.0;
for(i=0;i<n;i++){
    K += m[i]*(SQR(v[i][0]) + SQR(v[i][1]) + SQR(v[i][2]));
}
K *= 0.5;

/* potential energy */
```

ポテンシャルエネルギーについては5.6.4を参照

最初にタイムステップ $\Delta t = 2^{-5}$、シミュレーションの終了時刻 $t_{\rm end} = 1.0$、粒子数 $N = 1024$、ビリアル比 $r_{\rm V} = 0.1$、ソフトニングパラメータ $\epsilon = 0.03125$ を設定した場合のエネルギーの相対誤差 $|\Delta E/E_0|$ が 1.625326e-03 となることを確認しよう。ここで E_0 は初期エネルギー、$\Delta E = E - E_0$ である。

タイムステップが $\Delta t = 2^{-5}$ の時のエネルギー誤差が確認できたら、タイムステップを $\Delta t = 2^{-n}(n = 5\text{-}9)$ と変化させて $t = 1.0$ まで積分して、エネルギーの相対誤差を計算し、それをタイムステップに対してプロットしよう。プログラムが正しければ図 5.4 のような図が得られるはずである。これは粒子数 $N = 1024$、ビリアル比 $r_{\rm V} = 0.1$ の一様等温球をソフトニングパラメータ $\epsilon = 0.03125$ で時間積分した場合である。対数での誤差の傾きがほぼ 2 になっている、すなわちエネルギー誤差はタイムステップの 2 乗に比例していることがわかる (参考: GRAPE-9 を使用した場合は、計算精度は悪くなるが、GPU は CPU と同じ精度で計算されている)。もしこうなっていなかったらプログラムのどこかが間違っているはずである。デバッグをしよう。

5.8 可視化法

(4) (3) で作ったプログラムにアニメーション表示を実装しなさい。

シミュレーションがどのように進んでいるかを見るのには例えば毎ステップスナップショットを表示させればよい。これは見ていてなかなか楽しいものである。図を表示させるにはシミュレーションのプログラムからグラフィックライブラリを呼び出すことになる。

Unix 環境ではグラフィックライブラリとして X ウィンドウシステムのライブラリ Xlib を使うのがいいかもしれない。しかし、Xlib の関数はいろいろできる分だけプリミティブで使いにくい。ここでは代わりに gnuplot を使うことにする。gnuplot に関して詳しくは http://www.gnuplot.info/ を見ていただきたい。

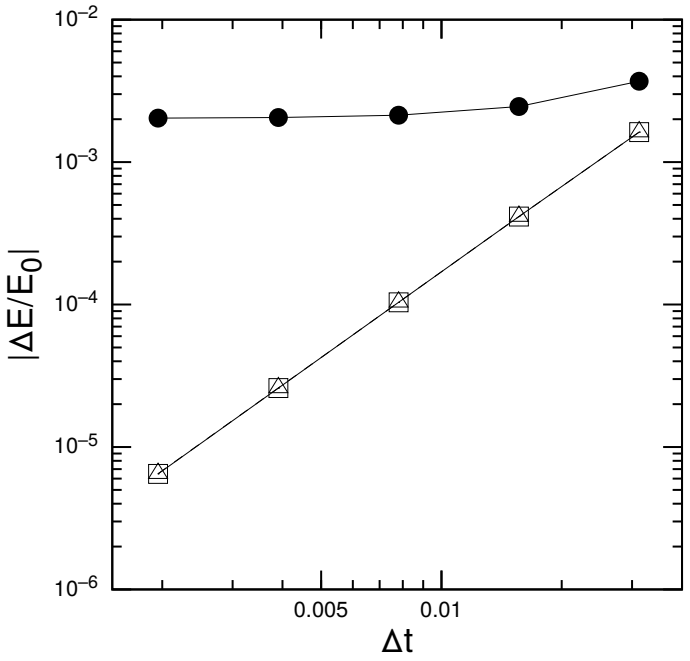


図 5.4: エネルギーの相対誤差のタイムステップへの依存 (△はホストコンピュータの CPU のみ、□は GRAPE ライブラリから GPU を使用、●は GRAPE-9 を使用)

5.8.1 グラフィック関数

この実習では gnuplot の使い方を説明するのが目的ではないので、gnuplot を使って作ったグラフィック用の関数

`open_window`	ウィンドウを開く
`close_window`	ウィンドウを閉じる
`animated_snapshot`	スナップショットをアニメーションのように表示する

を用意してある。`/home/nbody00/practice1/gnuplot.c` にあるのでコピーして使って欲しい。

プログラムに組み込むときは次のようにすればよい。

```
int main(void)
{

    略

    FILE *gp;
    open_window(&gp);

    while(t<t_end){
```

```
        時間積分

        /* visualization of particle data */
        animated_snapshot(n, t, x, &gp);

    }

    close_window(&gp);

    略

}
```

これらの関数の表示出力を調整するために、プログラムの最初のほうにマクロ変数が定義してある。いろいろ変更して試してみよう。

5.9 リアルタイム解析

(5) (4) で作ったプログラムを一定の間隔で系のビリアル比を計算するようにし、標準出力に出力せよ。そしてビリアル比の進化を調べよ。

リアルタイム解析とはシミュレーションをしながらでなくては計算できないような、つまり頻繁に解析しなくてはいけないような解析のことである。たまに解析すればいいようなものはスナップショットをファイルに出力しておいてシミュレーションの後にやればよい。

ここではリアルタイム解析の例として系のビリアル比の進化を求めてみる。力学 (ビリアル) 平衡にある粒子系のビリアル比 $r_{\rm V}$ は

$$r_{\rm V} = \frac{1}{2} \tag{5.15}$$

となる。cold collapse の場合、初期状態では力学平衡ではないので時間進化とともに力学平衡へ向かうことになる。この様子を見てみよう (厳密にはビリアル比は系に束縛されている粒子だけで計算するべきだが、系に束縛されている粒子を選び出すのは手間がかかるのでここでは全粒子についてビリアル比を計算することにする。系から escaper が多数出なければビリアルはほぼ 1/2 になる)。

ある決まった時間間隔で系に束縛されている粒子を使ってビリアル比を計算して系がビリアル平衡に近づくことを確認しよう。リアルタイム解析のプログラムは main 関数の時間積分ループの中にいれる。そして、適当な時間間隔 $t_{\rm out}$ でデー

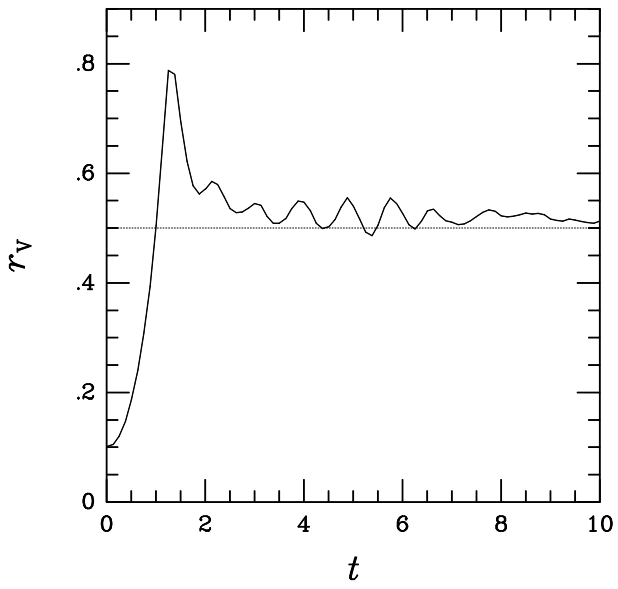


図 5.5: ビリアル比の時間進化

タを出力するようにすればよい。これを簡単に実現するには例えば割算のあまりを返す関数 fmod を使って

```
if(fmod(t, t_out)==0.0){
    リアルタイム解析
}
```

とすればよい。ただし、ここで dt は 2^n で当り前だが t_out は dt で割り切れる必要がある。

シミュレーションがうまくいっていれば図 5.5 のような図が得られるはずである。これは粒子数 $N = 1024$、ビリアル比 $r_{\rm V} = 0.1$ の一様等温球をソフトニングパラメータ $\epsilon = 0.03125$、タイムステップ $\Delta t = 0.03125$ で $t = 10$ まで時間積分した場合である。

(6) シミュレーションのパラメータ (ϵ、Δt) や初期条件 (N、$r_{\rm V}$) を変えていろいろシミュレーションをしてみよ。

5.10 スナップショット出力

ある時刻の粒子分布 $(m, \boldsymbol{x}, \boldsymbol{v})$ を出力したデータをスナップショットと言う。N 体シミュレーションではシミュレーション後の解析のために適当な間隔でスナップショットを出力することが普通である。この実習ではスナップショット出力はしないが簡単に説明しておく。

5.10.1　フォーマット

スナップショットのフォーマットは自由である。自分が使いやすいフォーマットにしておけばいいだろう。例えば

$$
\begin{aligned}
&t\\
&n\\
&m_1\ x_1\ y_1\ z_1\ v_{x1}\ v_{y1}\ v_{z1}\\
&.\\
&.\\
&m_N\ x_N\ y_N\ z_N\ v_{xN}\ v_{yN}\ v_{zN}
\end{aligned}
$$

などでいいだろう。ここでtとnは時刻と粒子数である。最低でもこれらの情報も付けておかないと解析に使いにくくなるので注意。

5.10.2　ファイル形式

ファイルの大きさを押えるためと、正確なメモリイメージを書き込むために、スナップショットはバイナリ形式で作られることが多い。バイナリの入出力用にはC言語では`fread`、`fwrite`という関数がある。

最近は大容量のディスクも安く、ファイルの大きさが問題になることは少なくなってきたので、アスキー形式のスナップショットも多く使われている。これは見てすぐにわかる、もしくはすぐに描画プログラムの入力データとして使えるという利点がある。

バイナリ形式のスナップショットの大きな利点のひとつは、これをシミュレーションの初期条件として使うことによって、シミュレーションの連続性を損なうことなくシミュレーションをつなげることにある。例えば、ある時点で一度シミュレーションを終了し、そのときにバイナリ形式のスナップショットを出力しておく。それからさらにそのスナップショットを初期条件としてシミュレーションをすると、これははじめから連続してシミュレーションをした場合と完全に同じ結果になる。アスキー形式のスナップショットを使う場合はこうはいかない。アスキー形式のデータはメモリイメージの近似表現なので、一般にアスキー形式のデータをメモリに読み込むとき、そのアスキーデータの元となったメモリイメージとは完全に同じにはならないのである。

アスキー形式のスナップショットを使うのはいいが、少なくともバイナリ形式のスナップショットもある間隔で出力するといいだろう。

5.11 プログラムのプロファイリング

(7) プログラムのプロファイルをとり、プログラムのどの部分が重い (時間がかかる) か調べよ。また、粒子数を変えると重みはどう変わるか調べてみよ。

ここまでくればめでたく cold collapse の N 体シミュレーションプログラムが動くようになっているだろう。N を大きくしてシミュレーションをしてみると、とても時間がかかると思う。これは相互重力の計算 $O(N^2)$ が重くなっているからである。どのくらい相互重力の計算に時間がかかるか見てみよう。それには gprof を使ってプログラムのプロファイルを見ればよい。そのためにはまず、プログラムをコンパイルするときに

```
$ cc -o collapse collapse.c -pg -lm
```

とプロファイリングのためのオプション-pg をつける。そして、いつも通りにプログラムを実行させると gmon.out というプロファイルデータファイルが作られる。

```
$ gprof collapse
```

とすればプロファイルを見ることができる。プロファイルは例えば

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 93.00     30.42    30.42 168655872     0.00     0.00  calc_two_body_force
  6.14     32.43     2.01      322     6.24   100.71  calc_force
  0.57     32.62     0.18  2095104     0.00     0.00  two_body_potential
  0.15     32.66     0.05      321     0.16   100.87  leap_frog
  0.03     32.67     0.01      321     0.03     0.03  plot_snapshot
  0.03     32.69     0.01        3     3.33    49.58  potential_energy
  0.03     32.70     0.01        1    10.00    56.25  make_spherical_df
  0.03     32.70     0.01                             XDrawLine
  0.02     32.71     0.01     3072     0.00     0.00  kinetic_energy
  0.00     32.71     0.00     3072     0.00     0.00  gaussian
  0.00     32.71     0.00      321     0.00     0.03  animated_snapshot2
  0.00     32.71     0.00        3     0.00    51.25  calc_system_energy
  0.00     32.71     0.00        1     0.00     0.00  close_window
  0.00     32.71     0.00        1     0.00     0.00  make_power_law_sphere
  0.00     32.71     0.00        1     0.00     0.00  open_window
```

となる。これは $N = 1024$ のシミュレーションである。$O(N^2)$ の計算、すなわち 2 粒子間の重力を計算する関数 `calc_two_body_force`(相互重力を計算する `calc_force` で呼ばれる) で計算時間の 90%以上が使われているのがわかる。初期条件設定や時間積分の部分は相互重力の部分に比べると無視できるくらいである。この傾向は粒子数が増えるほど強くなる。

普通、プログラムが動くようになったら、プロファイルを見てどの部分で時間がかるかを調べて、そこが速くならないか工夫をする。この工夫とは普通は速いアルゴリズムを使うとかソフトウェア的なものである。しかし、N 体シミュレーションの場合はソフトウェア的にはどうしようもないことが多い。そこでハードウェア的に解決しようという考えが生まれた。それが重力多体問題専用計算機 GRAPE や GPGPU である。GRAPE や GPU のような計算加速器で N 体シミュレーションで $O(N^2)$ となる計算を行うことで実行時間を短縮する. ことができる。

(発展課題) 教科書に書いてある相互重力計算とリープフロッグ法のプログラムをプロファイルを元に計算量が少ない書き方にしなさい。

5.12 おわりに

ここまでの実習では cold collapse 問題を例として簡単な N 体シミュレーションのプログラムを書いてみた。プログラムの基本構造とシミュレーションの流れが理解できてもらえたら幸いである。実習 2 ではいよいよ GPU を使うことになる。乞う御期待。

[小久保英一郎/押野翔一]

第6章　実習2　GPUを用いたN体シミュレーション: 銀河衝突を例として

6.1　目的

この実習では 銀河衝突を例として、GPU と GRAPE ライブラリを使った N 体シミュレーションを行う。具体的な実習内容は、

- GRAPE ライブラリを用いた実装
- 銀河衝突実験

である。

6.2　GRAPEライブラリの利用

GRAPE ライブラリは GRAPE を利用するための関数を GPU でも実行することができるようにしたライブラリ群である。GRAPE は、N 体シミュレーションの中で大部分を占めている重力計算部を加速する専用ハードウェアであった (詳しい説明は講義 3 参照のこと)。GRAPE ライブラリを使うことで、GRAPE と同じインターフェースで GPU の計算能力を利用することができる。

(1) 実習 1 で作成したプログラムの重力計算部を GRAPE ライブラリ を使って計算するように変更しなさい。

ここでは、GRAPE-9 を使う場合と同名の関数を使う。このライブラリを使うために、プログラムにライブラリ用ヘッダーファイルを読み込んでおく必要がある。

```
#include "g5util.h"
```

N 体シミュレーションでは、座標や質量などを GPU に送る関数、GPU から力・ポテンシャルなどの計算結果を受けとる関数、それと、初期設定などをする関数を呼ぶ必要がある。以下がサンプルプログラムである。

```
void calc_force_on_gpu(int n, double m[], double x[][3],
                           double a[][3], double pot[], double eps2)
{

    g5_set_range(-256.0,256.0,m[0]);

    g5_set_jp(0, n, m, x);
    g5_set_eps2_to_all(eps2);
    g5_set_n(n);

    g5_calculate_force_on_x(x, a, pot, n);

}
```

ここで、pot は本来ポテンシャルエネルギーを入れる配列であるが、GRAPE-9 にはポテンシャルエネルギーを計算する機能がなかったためダミーとして用意している。配列サイズは NMAX で良い。ポテンシャルエネルギーは実習 1 と同様にホストで計算する。

それぞれの関数の詳しい使い方・説明は G5PIPE User's Guide for GRAPE-7 software package version 2.1 にあるが、簡単に説明しておく。なお、関数名が全て g5_ から始まっているが、これは GRAPE-5 との互換性を保つためである。

g5_open() は GPU の使用権を得るため関数で、全ての他の関数に先だって呼ぶ必要がある。この関数 g5_open() は他に必要な設定を行なう。 GPU を使い終ったら、最後に g5_close() を呼んで使用権を解放する必要がある。したがって以下のように main ルーチンの重力計算を行う部分を以下のように g5_open と g5_close で囲む必要がある。

```
    g5_open();

       main ルーチンで calc_force_on_gpu を呼び出す区間

    g5_close();
```

g5_set_range() は送る座標･質量の範囲を設定する関数である。これは GRAPE-9 が内部表現に固定小数点表示方式を使っているので必要となる。それぞれの引数に、座標の最小値、座標の最大値、質量の最小値を与える。今回の課題のよう

な、等質量で比較的大きなソフトニングを使う計算では、座標・質量の範囲が小さいので特に問題にならない。ただし、Dynamic range が 10^6 程度を越える場合、粒子間の質量比が大きい場合には overflow への注意が必要となってくる。

`g5_set_eps2_to_all()` は、全粒子に共通の softening length を GPU に送る関数である。 `g5_set_n()` は j 粒子の数を送る関数である。`g5_set_jp()` は、j 粒子の質量・座標を GPU に送る関数である。

`g5_calculate_force_on_x()` は GPU から結果を回収する関数である。引数に i 粒子の座標・粒子数を与えると、力を計算して返す。

これらの関数を組み込めたらコンパイルをする。GPU を使うために、g00 系では最後のリンクの際に、

```
-pthread -lcuda5 -lcudart -lstdc++
```

を付け加える。

(2) (1) で作成したプログラムが正しく動作していることを確認しなさい。

コンパイルまでうまくいったら、GPU が搭載されている計算ノードに入ってプログラムを実行する (注: ログインノード `g00.cfca.nao.ac.jp` には GPU は搭載されてない)。計算ノードへのログイン方法については別資料を配布する。ただし計算ノードへのログインは講習会時のみである。講習会後の試験利用期間や利用申請を行い採択された場合の通常の利用時は、ジョブスケジューラを経由してジョブを投入して終了を待つという形での利用となる。

GPU で GRAPE ライブラリが正しく動作しているかどうかは、もっとも丁寧に確認するとすれば、

(a) 計算機を使わなくても力を計算できる位置に粒子をおいて、GPU が正しい答を返してくるか比べる。

(b) ホストコンピュータ上で計算した値と比べる。

(c) エネルギー保存が成り立つかを確認する (注: GRAPE ライブラリはポテンシャルエネルギーを返さないので、適宜粒子分布からホスト計算機で計算すること)。

(d) 計算性能が得られているか確認する。

の手順で確認する。今回は時間の都合があるので、エネルギー保存を見て確認する。

(3) 粒子数、ビリアル比などを変えて、シミュレーションを実行しなさい。

6.3 銀河衝突実験

ここでは、GPU で重力計算を行なうプログラムを用いて、N 体シミュレーションをしてみる。その例として、銀河衝突のシミュレーションを行なう。まず始めに、課題 (1)-(5) で初期条件の準備・実行・解析といったシミュレーションの一通りの流れを追ってみる。課題 (6)-(11) で、いろいろな計算パラメータを変えて、シミュレーションを実行してみる。

(1) 2 個の Plummer モデル銀河からなる合体前の初期条件を用意しなさい。初期 separation を 標準 N 体単位系で 6、初期相対速度を 0 とする。

N 体シミュレーションにとって初期条件をいかに作るかというのは、実際のシミュレーションそのものと同程度に大変な作業である。そのために、世の中に出回っている様々な N 体シミュレーション用ソフトウェアパッケージを使うことが多い。ここでは、NEMO という主に銀河力学用のパッケージを使ってみる。ただし、NEMO はログインノード (g00.cfca.nao.ac.jp) でしか使用できないので NEMO を使用するときは一旦、ログインノードに戻る。g00 でこのあと出てくる NEMO のコマンドを使うときは、事前に

```
$ module load gcc/12.2
```

しておく必要がある。NEMO を使って、2 個の Plummer モデル銀河からなる初期条件を作ってみる。Plummer モデルとは、

$$\rho = \rho_0 \left(1 + \frac{r^2}{r_0^2}\right)^{-5/2} \tag{6.1}$$

の密度分布で与えられるモデルで、$n = 5$ のポリトロープ解でもある。観測される銀河の分布にはあまり合わないが、割と簡単な式で与えられるので良く使われている。

1 個の Plummer モデルが、

```
$ mkplummer nbody=2048 out=pl2k
```

で、binary ファイル pl2k に生成される。 nbody=2048 で粒子数を指定する。プログラムの詳しい説明は、

```
$ man mkplummer
```

で得ることができる。他の NEMO のコマンドでも同様である。 mkplummer はなにも指定しないと、Heggie and Mathieu (1985) の standard unit で出力される。この単位系では、$G = M = -4E = 1$ となる。ここで、G は重力定数、M は全質量、 E は全エネルギーである。

この Plummer モデルを距離 6 離して 2 個おく。まず、1 つ目の銀河を (3,0,0) におく。モデルを移動するコマンドは snapshift で、

```
$ snapshift in=pl2k out=pl2k_0 rshift=3,0,0
```

で、移動したモデルが pl2k_0 に出力される。同様に 2 つ目の銀河も反対方向に移動する。

```
$ snapshift in=pl2k out=pl2k_1 rshift=-3,0,0
```

この移動した 2 つの銀河を snapadd で合わせる、

```
$ snapadd in=pl2k_0,pl2k_1 out=pl2kx2
```

合わさったモデルが、pl2kx2 に出力される。正しく生成されているかを、snapplot を使って確認する。

```
$ snapplot in=pl2kx2
```

範囲を xrange=, yrange= を使って、

```
$ snapplot in=pl2kx2 xrange=-5:5 yrange=-5:5
```

のように調節する。最後に、binary 形式のファイルを ascii 形式のファイルに変換しておく。

```
$ stoa in=pl2kx2 out=pl2kx2.ascii
```

(2) (1) で作った初期条件をプログラムから読み込めるように変更しなさい。

以下が pl2kx2.ascii というファイルから NEMO の ascii 形式を読み込むプログラムのサンプルである。

```
FILE *fp;

....

fp = fopen("pl2kx2.ascii","r");
fscanf(fp,"%d%d%lf",&n,&dim,&t);
for(i=0;i<n;i++) fscanf(fp,"%lf",&m[i]);
for(i=0;i<n;i++) fscanf(fp,"%lf%lf%lf",&x[i][0],&x[i][1],&x[i][2]);
for(i=0;i<n;i++) fscanf(fp,"%lf%lf%lf",&v[i][0],&v[i][1],&v[i][2]);
fclose(fp);
```

(3) プログラムを実行し、銀河を衝突させなさい。$T = 30$ まで計算せよ。

アニメーションを表示させている場合、描画サイズを広く取る必要があるためマクロの数値を変更する。数値は小数点以下まで書かないとエラーが表示される。

```
#define VMAX 5.0
```

正しく実行できていれば、$T = 12$ くらい で衝突し、$T = 20$ くらいで力学的に落ちつくはずである。

(4) 最終状態を出力するようにプログラム変更しなさい。

以下が nemoout.ascii というファイルへ NEMO の ascii 形式を出力するプログラムのサンプルである。

```
FILE *fp;

....

fp = fopen("nemoout.ascii","w");
fprintf(fp,"%d\n%d\n%e\n",n,dim,t);
for(i=0;i<n;i++) fprintf(fp,"%e\n",m[i]);
for(i=0;i<n;i++) fprintf(fp,"%e %e %e\n",x[i][0],x[i][1],x[i][2]);
for(i=0;i<n;i++) fprintf(fp,"%e %e %e\n",v[i][0],v[i][1],v[i][2]);
fclose(fp);
```

(5) 最終状態の出力結果から密度プロファイルを作りなさい。

結果の解析は、シミュレーションによって見るものが異なるので自分でプログラムを書いて行なうのが普通である。ここでは、NEMO のコマンドを使って密度プロファイルを作ってみる。

出力されたファイルは ascii 形式だが NEMO で解析するには binary 形式である必要がある。ascii 形式のファイルを binary 形式にするには

```
$ atos in=nemoout.ascii out=nemoout
```

とすればよい。ここで `nemoout` は binary 形式のデータである。密度プロファイルを作るコマンドは、`radprof` であり、

```
$ radprof in=nemoout tab=t kmax=100 > rho.dat
```

で、テーブルとなったデータが rho.dat に出力される。 `rho.dat` のデータの、1 列目が半径、2 列目が密度である。各自適当なグラフツール (gnuplot, sm など) を使ってプロファイルを描いてみよ。

(6) 初期の銀河の大きさを変えてシミュレーションを実行しなさい。

1 つ目の銀河の大きさを 1/2 にコンパクトにする。モデルのスケーリングをするコマンドは `snapscale` で、

```
$ snapscale in=pl2k out=pl2k0 rscale=0.5 vscale=1.414
```

でモデルが 1/2 にスケーリングされる。 `rscale` は座標の、`vscale` は速度のスケーリング factor を与える。ここで速度もスケーリングしているのは、ビリアル平衡を保つためである。

(7) 初期の銀河モデルを変えてシミュレーションを実行しなさい。

Plummer モデル以外の銀河モデルを用いてシミュレーションを実行してみる。NEMO を使えば、より realistic な銀河モデルである Hernquist モデルと、 Exponential Disk モデルを生成することができる。それぞれ、

```
$ mkhernquist out=hn2k nbody=2048 zerocm=f
```

```
$ mkexpdisk out=disk2k nbody=2048
```

とするとモデルができる。 Exponential Disk モデルはいろいろ他にパラメータが設定できる。詳しくは man を参照のこと。さらに、disk を回転させたい時は、snaprotate が使える。

```
$ snaprotate in=disk2k out=disk2k90 theta=90 spinvector="60,70,80"
```

(8) 初期の軌道を放物軌道にかえてシミュレーションを実行しなさい。

放物軌道では 2 つの銀河の軌道エネルギー ($E = -M^2/R + Mv^2$) が 0 になる。初期の separation を 6 のまま放物軌道にするために、それぞれの銀河に $1/\sqrt{6}$ の初速度を与える。初速度を与えるのは snapshift で、

```
$ snapshift in=pl2k out=pl2k_1 rshift=-3,0,0 vshift=0.4082,0,0
```

と、vshift=オプションで指定する。この初期条件では軌道運動だけを考えれば合体しないはずだが、軌道運動が銀河の内部運動に転化されるので合体する。

(9) 初期の軌道を双曲軌道にかえてシミュレーションを実行しなさい。

双曲軌道では 2 つの銀河の軌道エネルギーが正になる。それぞれの銀河に初速度 1 を与える。この初期条件では銀河がすりぬけてしまうはずである。

(10) 初期の軌道を系統的に変えてみて、合体の criterion を求めなさい。

(1)、(8)、(9) では、角運動量 0 でエネルギーを変えて合体するかどうかをみたが、銀河の軌道は 2 体問題なので、エネルギーと角運動量の 2 パラメータで記述される。そこで、エネルギーと角運動量を系統的に変えてみて合体できる条件を求める。 (8) で述べたように軌道運動が銀河の内部運動に転化されるため、双曲軌道でも合体できることがある。その criterion を数値実験で求めてみる。

ここではパラメータを、エネルギーと角運動量の代わりに、初期の相対速度 V とインパクトパラメータ p の二つで記述する。 V と p を系統的に変えてみて合体の criterion を求める。

6.4 その他の実験

余裕があれば、作成したGPUで重力計算を行なうプログラムを用いて、以下のN体シミュレーションを実行しなさい。

- Diskの力学的進化 (bar不安定、bending不安定)
- Cosmological hierarchical clustering
- Star clusterの長時間進化

[福重俊幸/斎藤貴之/押野翔一]

第7章　講義4　高度なN体シミュレーション法

今回の実習では無衝突系のN体シミュレーションを行なった。時間積分には2次の公式リープフロッグ法を固定タイムステップで使った。相互作用計算はおもに直接計算法を使った。

現在、N体シミュレーションをするときはこれらの方法だけでは必要な粒子数や計算精度を実現することは難しい。特に、衝突系を扱うには時間スケールの幅が大きな系を精度よく計算できなくてはならないし、宇宙論的なシミュレーションでは周期境界条件も必要になってくる。

ここでは、今回の実習のN体シミュレーションの次のステップになる、さらに高度なN体シミュレーションの方法について簡単に紹介する。

7.1　高精度時間積分法

衝突系のN体シミュレーションの場合、2次精度のリープフロッグ法では精度が足りないことが多い。高次の時間積分法にはいろいろあるが、よく使われるのは4次のエルミート法である。

7.1.1　エルミート法

エルミート法は予測子修正子法の1種ある。具体的には次のような公式になる。予測子はテイラー級数で与えられる。

$$\boldsymbol{x}_{\rm p} = \boldsymbol{x}_0 + \boldsymbol{v}_0\Delta t + \frac{\boldsymbol{a}_0}{2}\Delta t^2 + \frac{\dot{\boldsymbol{a}}_0}{6}\Delta t^3 \tag{7.1}$$

$$\boldsymbol{v}_{\rm p} = \boldsymbol{v}_0 + \boldsymbol{a}_0\Delta t + \frac{\dot{\boldsymbol{a}}_0}{2}\Delta t^2 \tag{7.2}$$

修正子はエルミート補間によって構成された加速度の補間多項式を積分して構成される。エルミート補間では2つ時刻の加速度$\boldsymbol{a}$とその時間微分$\dot{\boldsymbol{a}}$から加速度の3次の補間多項式を構成する。修正子は具体的には

$$\boldsymbol{x}_{\rm c} = \boldsymbol{x}_{\rm p} + \frac{\boldsymbol{a}_0^{(2)}}{24}\Delta t^4 + \frac{\boldsymbol{a}_0^{(3)}}{120}\Delta t^5 \tag{7.3}$$

$$\boldsymbol{v}_{\rm c} \;=\; \boldsymbol{v}_{\rm p} + \frac{\boldsymbol{a}_0^{(2)}}{6}\Delta t^3 + \frac{\boldsymbol{a}_0^{(3)}}{24}\Delta t^4 \tag{7.4}$$

となる。ここで $\boldsymbol{a}_0^{(2)}$、$\boldsymbol{a}_0^{(3)}$ は補間多項式の係数で

$$\boldsymbol{a}_0^{(2)} \;=\; \frac{-6(\boldsymbol{a}_0 - \boldsymbol{a}_1) - \Delta t(4\dot{\boldsymbol{a}}_0 + 2\dot{\boldsymbol{a}}_1)}{\Delta t^2}, \tag{7.5}$$

$$\boldsymbol{a}_0^{(3)} \;=\; \frac{12(\boldsymbol{a}_0 - \boldsymbol{a}_1) + 6\Delta t(\dot{\boldsymbol{a}}_0 + \dot{\boldsymbol{a}}_1)}{\Delta t^3} \tag{7.6}$$

である。

エルミート法では加速度の他に加速度 $\boldsymbol{a}$ の時間微分 $\dot{\boldsymbol{a}}$ も必要になる。実は GRAPE-4、GRAPE-6 は $\dot{\boldsymbol{a}}$ も計算できるようになっている。さらに予測子も計算できるようになっている。GRAPE を使えば高速にエルミート法を使うことができるのである。

エルミート法は一段法であり実装が比較的簡単である。さらに時間対称な公式でもある。エルミート法は現在衝突系の標準積分法となっている。エルミート法について詳しくは Makino & Aarseth (1992) を見ていただきたい。

7.2 タイムステップの工夫

7.2.1 独立タイムステップ法

今回の実習で、時間積分のタイムステップは固定タイムステップだった。すなわち、すべての粒子についていつも同じタイムステップを使っていた。しかし、時間スケールの幅が大きな系の場合これでは効率的に時間積分できない。例えば、数日で公転している連星系と数億年で銀河を公転している星を同じタイムステップで時間積分するのは無駄である。そこで、粒子ごとにその運動の変化の応じた大きさのタイムステップをもたせればよい。この方法を独立タイムステップ法という。独立タイムステップ法はエルミート法などの予測子修正子型の公式とともに用いる。独立タイムステップ法では粒子はそれぞれの時間にいることになる。しかし、ある粒子を時間積分するときはそのときの粒子の加速度が必要になり、そのためにはそのとき他の粒子の位置がわからなくてはならない。粒子の位置は予測子を使えば計算できる。

7.2.2 階層化タイムステップ法

この予測子を使って粒子の時間をそろえる計算を粒子ごとに行なっていたのではその計算量は $O(N^2)$ になってしまう。これはうれしくない。そこでこの計算量を減らすために、独立タイムステップ法を改良した階層化タイムステップ法とい

うのが考案された。階層化タイムステップ法では、粒子のとれるタイムステップを 2^n とする。おおざっぱに言うとこうしておくとある時刻に同時に積分される粒子が複数でてきて、その分予測子の計算の回数を少なくすることができる。階層化タイムステップ法については詳しくは、Makino (1991) を見ていただきたい。

7.3 相互作用計算の工夫

天体の N 体シミュレーションでは、粒子 1 つ 1 つの軌道を求めることが重要な場合（例えば、連星の軌道、ブラックホール回りの恒星の軌道、太陽のまわりのケプラー軌道、とか）以外は、さほど相互作用の精度を必要としない。ということで、相互作用計算を加速するために、直接計算法ではなく近似的な解法を使うのが一般的である。もっとも代表的な方法は**ツリー法**で、遠くにある粒子はまとめて計算することで、計算量を落す。また、宇宙論的 N 体シミュレーションのような周期境界条件下では、遠くの粒子からの相互作用をフーリエ変換を用いて計算する方法も用いることができる。

ツリー法は階層的なツリー構造を用いる方法で、計算量を $O(N \log N)$ に落すことができる。ツリー構造の作り方で最も代表的なのが、Barnes and Hut (1986) の方法で、空間を再帰的に 8 分割することで構造を作り上げていく。ある粒子からの相互作用を計算する際には、このツリー構造を使って、十分離れている粒子からの相互作用はノードからのそれで代表して計算し、離れていない時はさらにツリーをたどっていく。

宇宙論的 N 体シミュレーションのように周期境界条件下で行なう場合は、フーリエ変換を用いて、計算量を落すというやり方が使える。PM(Particle-Mesh) 法は、重力をメッシュを用いた高速フーリエ変換 (FFT) で求める方法である。が、この方法は空間分解能がメッシュサイズで制限されるため、あまり一般的ではない。その PM 法の改良である、P3M(Particle-Particle Particle-Mesh) 法が一般的に用いられる。この方法では、近く粒子からの寄与は直接計算して、メッシュでの FFT を使って求めた遠くからの寄与とあわせて、重力が求まる。周期境界条件下でツリー法を用いることも可能で、遠くからの寄与をフーリエ変換を用いた方法であるエワルド法もしくは PM 法と組み合わせて、重力を求める。

Barnes J. E., Hut P., 1986, Nature, 324, 446

Makino J., 1991, PASJ, 43, 859

Makino J., Aarseth S. J., 1992, PASJ, 44, 141

[福重俊幸/小久保英一郎]

第8章　講義4　N体シミュレーションの未来

8.1　はじめに

なんだか大層なお題だが、まあ、適当な話をする。

未来なんていう話をする前に、シミュレーションの現状はどうかというような話をするべきであろう。

ここでは、N体計算というよりは大規模数値シミュレーション一般について研究に使う上で現在なにが制限になっていて、それに対して今後どういうことが考えられるかということを、思いつくままに述べる。

8.2　ムーアの法則

N体シミュレーションを研究に使う上でもっとも問題なのは、やはり、本当にやりたいことをやろうとすると計算時間がかかり過ぎるということである。もちろん、実際にはプログラム開発とかデバッグとか結果の解析とか論文書きに計算自体よりもはるかに時間がかかるというのはそれもそれで嘘ではない。

しかし、計算機が十分速ければプログラム開発も速くなるだろうし、シミュレーションが例えば今1週間かかっているところが5分で終るなら、ほかのいろんなところも速くできる。結果の解析の手間も、計算結果をとっておいてなんとかするとか考えなくても、何かを見ようと思ったら再計算すればすむわけでいろいろな手間が減る。

もちろん、今1週間でやっている計算が5分で出来るようになるなら、今10年かかってもできない計算が1週間でできるということに理屈ではなるわけで、そういうことをしたければ今と同じように大変ということにはなる。でも、まあ、その分今では出来ないことができるようになるわけで、それはそれで悪くはないということにしたい。

が、ここで一つ問題なのは、今後計算機は速くなるのか？ということである。

汎用計算機の能力は、1940年代終りの電子計算機の発明以来50年間にわたってほぼ10年で100倍の割合で進歩してきた。その中で、ここ30年間の進歩は主に半導体製造技術の進歩に支えられたもの、つまり、いわゆるムーアの法則によるものである。

ムーアの法則と呼ばれるものはいろいろあるが、それらのベースになるのは大雑把にはLSIに作り込むトランジスタの1次元的な寸法が、加工技術の進歩によってほぼ3年ごとに半分になるということである。このことと、 CMOS トランジスタの動作速度がほぼ大きさに比例するということから、 3 年ごとに

- メモリチップの容量は 4 倍になる
- 論理 LSI の動作速度は 2 倍になる

というのが出てくる。

これらが計算機の速度にどう効いてくるかというわけだが、計算上はトランジスタの数が 4 倍、速度が2倍なんだから単位時間あたりに出来ることは 8 倍になるはずである。が、そうだとすると 3 年で 8 倍だから 10 年で 500 倍になって、実際には 10 年で 100 倍くらいにしかなってないというのとだいぶずれる。

これはどうしてかというと、結局トランジスタの数が 4 倍になったからといって 4 倍沢山計算する（4倍演算器を入れる）というふうになっていないからである。例えば、インテルの IA-32 の系列のプロセッサでは、ここ 10 年近く浮動小数点演算器の数は 1 ないし 2 のままである。

これにたいして、80年代には同じインテルのプロセッサの性能向上は非常に大きかった。 8087 では浮動小数点演算一つをするのに 100 サイクル程度かかっていたのが、 Pentium ではサイクルごとに1演算が原理的には可能になり、ほぼ 100 倍とトランジスタ数の増加に見あった性能向上が実現されたわけである。

これはつまり、 サイクルあたり 1 演算までだと非常に順調にトランジスタ数増加を性能向上につなげられるが、それ以上になると急に話が難しくなるということを意味している。これがなぜかということにはいろいろ理由があるが、結局のところはこの性質が 1980 年代にスーパーコンピューターの発展が遅くなった理由であり、全く同じ理由によってマイクロプロセッサの発展も実効的に遅くなってきているのである。

インテルと AMD のクロック速度の競争とかを見ていると、そうはいっても随分順調に速くなっていると思うかも知れないが、演算数が変わっていないのでほとんどクロックの分しか速くなっていないということを考慮すると 10 年で 100 倍というトレンドよりも進歩はさらに遅くなっている。

もちろん、演算器の数が増えないということは、結局そもそもマイクロプロセッサがあまり大きくならないということであり、これは結局技術が進歩した時にその結果が速度が速くなるというよりはむしろ価格が下がるという形で現れるということである。

これはたとえば、 10 年前にはもっとも速いマイクロプロセッサを積んだ計算機は IBM とか HP とか DEC とかの 1000 万円近くする機械だったのに、いまは単一プロセッサでは 10 万円そこそこで買える Intel P4 の機械よりも速い計算機というのは実は存在しないということである。

GRAPE でするように自分で計算機をつくるということをしない限り、あるいはGRAPE でもそのホストになる汎用計算機を必要とする限り、この傾向から逃れることは難しい。つまり、単一プロセッサの機械はそこそこしか速くなっていかなくて、その代わり段々安くなっていくであろう。

8.3 並列計算の必要性

今述べたことからわかるように、今後しばらくの計算機環境というものを考えると、単一プロセッサの機械はそこそこ性能は上がるがそれと同程度、あるいはそれ以上に値段が下がる。まあ、これはつまりここ数年ほどそうであったのがしばらくは続くであろうというだけである。

この状況下で、より速く計算したい、あるいはより大きな計算をしたいと思うと、どうしても並列計算ということを考えないといけない。学振についてくる科研費くらいでその気になれば 16 プロセッサの PC クラスタを組めるわけで（どこにおくかはともかく)、それにもかかわらず並列計算をしないというのは現実的な選択とはいいがたい。

とはいえ、並列計算はいろいろ面倒であるのも確かではある。特に、 PC クラスタで開発する場合にはどうしても MPI とか PVM とかいったいわゆるメッセージ・パッシングのプログラミングモデルを使うことになるので、単一プロセッサ用に書いたプログラムを少しづつ手直ししていくというわけにはいかないのが面倒さを増している。

もちろん、例えば三鷹のセンターにはいっている富士通 VPP とかあるいはNEC SX-5 といった高い並列計算機を使うと、この問題はある程度までは回避できる。つまり、普通の計算機用に書いたプログラムで、計算量が多くてしかも並列に実行できるところがあればコンパイラがそういうところを自動的に発見するなり、あるいはこちらが指定するなりすればなんとかやってくれるわけである。

VPP のように共有メモリではない場合にはもうちょっと面倒なことがないわけではないが、まあ、話としてはそういうことになる。 VPP とか SX とかまでいわなくても、 CPU 16 台程度までなら共有メモリ型の計算機というのはいろいろあって、それらの上ではコンパイラに並列化してもらうというのができなくはない。

が、この方法の問題は、「高い」ということである。 VPP とかだと 1 プロセッサ当たりの価格が数千万円するが理論ピーク性能は Intel P4 の数倍しかない。CPU 16 台の並列計算機というのもやはり数千万円といった値段にはなる。

極めて現実的な話をすると、三鷹の VPP で 500 Gflops の最大構成を一年のうち 1 ヶ月使うのと、研究室に 32 CPU くらいの PC クラスタをおいて 1 年計算するのではできることは（メモリサイズの違いが問題でなければ）ほとんど変わらない。で、何ヶ月もかけて計算するんなら、 MPI とかでプログラム開発が大変になっても引き合うという面もなくはない。

GRAPE を使うと普通の PC に比べて 1 台あたりで 10-1000 倍の性能が実現できるが、並列化できればさらに速くできるということ自体には変わりない。まあ、今は値段も 1 台あたり 100 万からするから、手軽に 16 台というわけには現在はいかないが、傾向としては同じことになる。

8.4 今後の方向？

本来、研究されているべき方向というのは、 PC クラスタ上で使いやすい、つまり単一プロセッサのプログラムからの移行・書き換えが容易なプログラム開発環境というものであるとは思うが、並列処理の研究の過去の歴史をみるとあまりそういう話にはなっていない。つまり、並列処理言語の研究は、 1970 年代から 80 年代にはいろいろあったけど今は下火になっていて、これは結局は実際のプログラムが MPI とかで書かれるようになったからである。まあ、これは並列処理言語が使えないので MPI が使われる、その結果並列処理言語の研究が進まないという循環になっているのも確かである。

というわけで、汎用計算機も数値計算向きのは自分で作ろうというようなちょっと危ない方向にいかないかぎり、計算機の発展の方向は、あまり速くないけれど安いものを大量に並べるというほうになる。しかも、計算機同士はこれまたあまり速くないネットワークでつながり、 MPI のような面倒臭いやりかたでプログラムするということになる。

ネットワークが遅いので、並列化の時にいろいろ余計なことを考えないと性能がでない。 MPI のような低レベル記述では人手でそういう余計なことができるのでそれでも性能がだせるが、並列処理言語とかだとそこまでいかないので性能を出すのが難しいことになる。このために、並列処理言語は簡単には普及しないと思われる。

まあ、面倒というかわりあい大変な分、やろうという人もそんなに沢山いなくて、そのためにやってちゃんと成果がでればお買い得なところもある。まあ、そのあたりは皆さんがこれから挑戦していく課題ということになる。

[牧野淳一郎]