理論天体物理学特論I

牧野淳一郎

1998 年 1 月 27 日

最終回

というわけで、この講義も最終回である。予定では

- 自己重力多体系におけるカオス
- 外場の影響

というようなことをすることになっていたが、どちらもわりと面倒な話で、ちょっと簡単には終らない。また、「本当かどうか (結果が信用できるか)」についてもまだ議論があるような話が多い。参考文献だけ紹介しておく。

1　自己重力多体系におけるカオス

Goodman, J., Heggie, D. C. and Hut, P., *On the Exponential Instability of N-Body Systems* , APJ 1994, 415, 715.

Fukushige, T., Makino, J., Nishimura, O. and Ebisuzaki, T., *Smoothing of the Anisotropy of the Cosmic Background Radiation by Multiple Gravitational Scattering* , PASJ, 1995, 47, 493-508.

Funato, Y., Makino, J., and Ebisuzaki, T., *Attenuation of luminosity of distant galaxies by multiple gravitational lensing* , APJL, 1994, 424, L17–20.

2　外場の影響

Chernoff, D. F. and Weinberg, M. D., *Evolution of globular clusters in the Galaxy* , APJ, 1990, 351, 121–156.

Fukushige, T., Heggie, D. C., *Pre-collapse evolution of galactic globular clusters* , MNRAS, 1995, 276, 206–218.

3　数値計算法

これだけでおわりにしては何なので、少し役に立つ (かもしれない) 話をしておく。

ここまで、自己重力多体系での物理過程の話をしてきたわけだが、では、いったいそういった系を

衝突系の場合には無衝突ボルツマン方程式を直接数値積分するというような方法もあり得る。また、衝突系の場合には orbit-averaged Fokker-Planck 方程式を数値計算するということも考えられる。

しかし、次元を落した系でなければ、上のような方法は現実的ではない。計算に必要なメモリ量や計算時間があまりに莫大なものになるからである。このため、どちらの場合でも、実際に多粒子系の数値計算を直接行なうという方法がとられてきた。

4　解くべき問題

解くべき問題は重力多体問題、つまり重力で相互作用する質点系がどのようにふるまうかという問題である。

系の進化を表す方程式は

$$\frac{d^2\mathbf{r}_i}{dt^2} = -\sum_{j\neq i} Gm_j \frac{\mathbf{r}_i - \mathbf{r}_j}{r_{ij}^3} \tag{1}$$

つまり、単にある粒子はほかのすべての粒子からの重力を受けるというそれだけである。

5　解く方法

解くべき常微分方程式系は式 1 で与えられるので、初期条件を作ればあとは各粒子の位置からそれぞれへの重力を計算するサブルーチンを書き、それを適当な数値計算ライブラリに渡して数値積分すればそれでおしまいと行けば話は簡単だが、それでは今日わざわざこの講義をする必要はない。そううまく行かないのには 2 つの理由がある。

第一の理由は、ナイーブな方法では 1 ステップ当たりの計算量が粒子数の 2 乗に比例して増えてしまうということである。もちろんもっとうまい方法がなければしょうがないが、残念なことに計算量を $O(N^2)$ から $O(N\log N)$ ないしは $O(N)$ に落とす方法が知られている。これらの原理は、基本的には遠くの粒子からの力は適当にまとめるということである。

もう一つの理由は、すべての粒子を同じ時間刻みで積分すると、ステップ数が非常に大きくなってしまうということである。例えばたくさんの微惑星が太陽のまわりをまわっているという計算をしたとする。ほとんどの粒子に対しては、太陽からの重力が圧倒的であり、また軌道は円軌道に近い。従って、例えば一周期を 100 ステップとかそれくらいで刻んでやれば十分正確に軌道を追いかけることができる。しかし、いくつかの微惑星は他の微惑星と近接遭遇をしているかもしれない。するとこの近接遭遇をちゃんと計算するためには時間刻みを短くする必要がある。このときに、すべての粒子の軌道を同じ時間刻みで積分していると、別に近接遭遇中でもなんでもない粒子もすべて短い時間刻みで計算されることになり、ステップ数が非常に大きくなる。この問題は、粒子毎にバラバラな時間刻みを与え、必要に応じてそれらを独立に変化させせることができれば解決できる。

なお、第二の問題、すなわち近接遭遇の問題には、別の対応もある。それは、相互作用のポテンシャルを変えて、近接遭遇が起きても時間刻みを短くしなくてもいいようにすることである。具体的には、ふつうの重力ポテンシャルは$-Gm/r$ で与えられるが、これを例えば$-Gm/\sqrt{r^2+\epsilon^2}$といった形にしてしまう。このようにすれば、 ϵより小さいところでは重力が大きくならないので、粒子の典型的な速度を v とすれば時間刻みをϵ/v にくらべて十分小さく取っておけばいい。これをソフトニングという。

ソフトニングをしていいかどうかは問題による。例えば微惑星の計算では、近接遭遇で軌道が変化する、あるいは物理的に衝突して合体するといった過程を計算したいので、ソフトニングをしたの

ではなんだかわからなくなってしまう。これにたいして銀河のシミュレーションとかいった場合には、近接遭遇で軌道が変化するといった効果は非常に小さく、ソフトニングが有効となる。

6　第一の問題への対策

銀河、銀河団のシミュレーションでは、ソフトニングが使えるためにタイムスケールの問題は無い。このために微惑星とか球状星団にくらべれば話は簡単であり、第一の問題、すなわち計算量が粒子数の 2 乗に比例して増えるという問題だけ考えればいい。

とはいえ、これは大きな問題である。例えば普通の渦巻銀河を、2 体緩和によって壊れてしまわないように表現するためには少なくとも 10^5個程度の粒子が必要である。このような銀河 1000 個から銀河団を作れば粒子数は 1 億個となり、1 ステップ計算するのに 10^{17}演算程度が必要になる。これは例えば 100 Gflops の超並列計算機で 10 日程度であり、1 万ステップ計算するのに 300 年かかってしまう。

もう少し速く計算するためにひろく使われているのが Barnes-Hut アルゴリズムあるいはツリーコードと呼ばれる方法である。原理は簡単で、遠くのほうの粒子はグループにまとめ、グループからの力は多重極展開（といっても普通に使うのはせいぜい 1 次または 2 次、1 次の展開は要するに重心で置き換えること）で計算する。

グループ分けは粒子毎に作っていてはしょうがないので、空間を階層的な立方体からなるツリー構造に組織してあらかじめすべての階層で多重極展開を計算しておく。ある粒子へのある立方体からの力は、それらが「十分に離れて」いれば多重極展開を計算して得られる。そうでなければ下の階層に再帰的に降りていって、十分離れているとみなせるところまでいって計算する。系全体からの力も、単に系全体に対応する立方体からの力を計算することで求められる。

この方法では計算量が $O(N^2)$ から $(ON\log N)$ になる。もう少し（原理的には）賢い方法として、$O(N)$ になるアルゴリズムが知られている。この方法では、力を及ぼすほうだけでなく受ける方もまとめて計算する。つまり、力を及ぼすほうの回りで有効なポテンシャルの多重極展開を、受ける方の中心でのテイラー展開に変換し、そのテイラー展開を各粒子の位置で計算することで力が求められる。

6.1　計算量の減少と計算効率の減少

$O(N\log N)$ の計算法は天文シミュレーションでは広く使われている。またパイプライン型ベクタプロセッサや並列計算機での実行もかなりよく研究されている。これに対し $O(N)$ のアルゴリズム (Fast Multipole Method, FMM) はまだ天文の問題に使えるような実用的な実装（実際に走るプログラム）を作った人がいないので、以下ツリーコードについてその実装上の問題点を簡単にまとめる。

ツリーコードでどのような問題が発生するかを考えるためには、比較の対象として $O(N^2)$ の直接計算アルゴリズムを考えることが有用である。これは、ベクタプロセッサでは非常に高い効率で走る。またほとんどいかなる並列計算機でも、マシンの理論ピークに近い性能を出すことができる。

もっともピーク性能を出すのが困難であると考えられる分散メモリ MIMD 型の時にどのような計算法を使えるかを簡単に説明すると以下のようになる。

まず各プロセッサに（大体）同じ数の粒子を割り当てる。自分ので持っている粒子同士の相互作用はそのまま計算すればいい。それ以外の他のところとの総誤差用は、プロセッサを 1 次元的なリングとみなして、粒子をたらい回しにして順次計算することができる。また、あるプロセッサのデー

タを全プロセッサに効率良く放送できる場合にはその機能を使ったほうがプログラムが容易である。

計算速度に対し必要な通信速度は、プロセッサ数をpとしてオーダとしてはp/Nになるので、$p << N$（普通成り立つ）では通信速度は問題にならない。共有メモリならば話はもっと簡単になる。

さて、ツリーコードの場合、演算量は減るが計算速度（1秒に何回浮動小数点演算をするかという意味での）は、普通のスカラープロセッサでも低下する。これは、ある粒子への力を計算する時の配列のアクセスの仕方が不規則であり、キャッシュミスの確率が高いことが大きな理由である。また、そもそも分岐自体が多いことも性能低化の理由となる。

ベクタプロセッサではより困難な問題、すなわち再帰呼び出しがあるようなループはベクトル処理できないという問題が発生する。これは87年頃にいくつかのアプローチによって解決された。（詳細は省く）

これらのアプローチは、アルゴリズムの変更による計算量の増大またはメモリアクセスの不連続性による計算速度の低下というペナルティを持つものの、もっとましな方法は知られていないということもあり現在も広く使われている。

共有メモリMIMD型の並列計算機の場合、再帰呼び出しも並列化可能で特に問題なく実行可能となる。ただし、分散共有メモリ（仮想共有メモリ）の場合、物理的には分散メモリなので後に述べる分散メモリのマシンと同じ配慮が必要になる。

分散メモリでツリーコードを実行する方法については、Caltechのグループは87年頃からいくつかの方法を試み、まあまあの結果を得ている。基本的なアイディアは、粒子を空間的に近いグループに分けるということである。分けるのには、それ自体にツリーによる空間分割を利用する方法や、適当に空間を半分に切っていく方法などが知られている。

7 第二の問題への対策

7.1 独立時間刻み

第二の問題、すなわちタイムスケールの問題には、粒子毎に別々の時間刻みを与えることで対応することになる。この場合、時間積分は「次の時刻」すなわち現在の時刻と時間刻みの和がもっとも短い粒子を選び、それの軌道を積分してその時刻と時間刻みを改めて計算し、最初に戻って次の粒子を選ぶという具合にすすむことになる。

この方法の場合、ベクタプロセッサでの実行は容易である。というのは、ベクトルの総和というのはすべての現存のベクタプロセッサで効率良く実行できる処理であるからである。

並列計算機の場合、一旦各プロセッサで自分の分を計算し、その総和をとってから軌道積分に進むということになる。これは$p << N$であれば不可能ではないが、時間刻み共通の場合に比べて難しい。これは、一回の通信でのデータ転送量が小さいため、ピーク転送速度ではなくメッセージレイテンシ（最初のデータを送り出してからつくまでの時間）が問題になるからである。実際に、必ずしも高い性能が得られないことがわかっている。

7.2 2重時間刻み

「独立時間刻み」法の改良として、「独立2重時間刻み」あるいはneighbor schemeといわれる方法がある。これは、ある粒子への力をその回りの粒子からの分と遠くからの分に分割し、遠くからの分は長い時間刻みで積分するという方法である。理論的には、単純な方法に比べた時の計算量の減

少が $N^{1/4}$ に比例するということになっている。

この方法の並列化は困難である。近くの粒子からの力の計算ではメモリアクセスが不規則でしかもループ長が短くなるので、ベクタプロセッサでも効率が大きく低下することが知られている。

7.3 ブロック時間刻み

並列計算機上で効率をあげるには、一度に 1 つの粒子しか動かさないのではなく、なるべくたくさんの粒子を並列に積分できることが望ましい。これは時間刻みを 2 のベキ乗に離散化することで可能になる。このようにすれば、ある粒子が積分される時にそれと同じかあるいは短い時間刻みを持つ粒子は並列に積分して良く、並列度は劇的に向上できる（理論的には $N^{2/3}$ と見積もられている）。現在までの分散メモリ型並列計算機への独立時間刻み法の実装は、すべてこの方法（ブロック時間刻み）によっている。

8 数値積分法

ここまでは、「相互作用をいつ、どうやって計算するか」という話をしてきた。しかし、実際に数値解を求めるには求まった重力加速度を使って各粒子の軌道を積分する必要がある。以下、その方法について簡単に述べる。

8.1 The Euler method

これは知らない人はいないと思いたいが、ある微分方程式の初期値問題

$$dx/dt = f(x,t) \tag{2}$$

$$x(0) = x_0 \tag{3}$$

があるときに、時刻 $t+\Delta t$ での近似解 x_{i+1} を、時刻 t での近似解 x_iを使って

$$x_{i+1} = x_i + f(x_i, t) \tag{4}$$

とする方法である。局所打ち切り誤差は $O(\Delta t^2)$ であり、したがって大域誤差は $O(\Delta t)$ ということになる。プログラミングは簡単であるが、誤差があまりに大きくほとんど実用にはならない。

8.2 The leapfrog method

これはセパラブルなハミルトン系、すなわちハミルトニアンが運動量の関数と位置の関数に分かれるものに対する特別な方法である。通常の、運動量に対するハミルトニアンが 2 次形式である場合には、

$$v_{i+1/2} = v_{i-1/2} + \Delta t a(x_i) \tag{5}$$

$$x_{i+1} = x_i + \Delta t v_{i+1/2} \tag{6}$$

と書くのが普通である。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

を使い、さらに終了用公式として

$$v_i = v_{i-1/2} + \Delta t a(x_i)/2 \tag{8}$$

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

$$\begin{aligned} x_{i+1} &= x_i + \Delta t v_i + \Delta t^2 a(x_i)/2 \qquad (9) \\ v_{i+1} &= v_i + \Delta t[a(x_i) + a(x_{i+1})]/2 \qquad (10) \end{aligned}$$

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

$$\begin{aligned} v_{i+1/2} &= v_i + \Delta t a(x_i)/2 \qquad (11) \\ x_{i+1} &= x_i + \Delta t v_{i+1/2} + \Delta t^2 a(x_i)/2 \qquad (12) \\ v_{i+1} &= v_{i+1/2} + \Delta t a(x_{i+1})/2 \qquad (13) \end{aligned}$$

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。

この公式は局所誤差が $O(\Delta t^3)$、大域誤差が $O(\Delta t^2)$ である。

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、現実には特に無衝突系の計算ではこの公式は非常に広く使われている。衝突系の計算で、独立時間刻みを使うような場合にはこの方法は使われないが、それ以外の非常に多くの問題はこの方法で解かれているといってよい。

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、例えば力学平衡の系でエネルギーや角運動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

2. 時間刻を変えてもうまくいくようにすることも出来る。

というわけで、leap frog が、それよりも局所的には誤差が小さいルンゲ・クッタとかに比べて良い結果を与えるのは当然のことである。

8.3 The Aarseth method

さて、独立時間刻みを使った場合には、話はそう簡単にはいかない。というのは、symplectic method の良い点も symmetric method の良い点も失われるからである。このため、通常はより高次の方法が用いられる。まあ、cosmological な計算なんかでは独立時間刻みで 2 次の公式を使っている例もないわけではないが、結果を信用できるかどうかには注意したほうがよいかもしれない。

高次の方法としては、伝統的には Aarseth が 60 年代に開発した方法が用いられてきた。これは、いわゆる線形多段法 linear multistep method に基づくものである。具体的な実現についてはここでは省くが、基本的に、過去数ステップの加速度から補外多項式を作り、それを積分して位置と速度を更新する。独立時間刻みを使うに当たっては、他の粒子の位置は補外多項式で計算する。

8.4 The Hermite method

Aarseth method は広く使われてきたが、結構面倒であるし高次にしていくと計算量が増え、また安定性が急激に悪くなる。もう少し簡単な方法はないかということで考えられたのが Hermite method である。これでは、加速度の他にその時間導関数まで直接計算し、それから補外、補間多項式を作る。これは最近かなり広く使われるようになってきた。

9 レポート課題

以下のどれでも OK。

1) これまでに出た問題のなかから、適当に何か解いてレポートにせよ。
2) これまでに紹介された論文（どれでも OK）のなかから、どれかを読んで内容を要約し、それが正しいかどうかについて議論せよ。
3) 自分の研究対象について、その振舞いに「自己重力性」が重要な振舞いを果たすかどうかについて議論せよ。