計算天文学 II 第11回 データ構造とアルゴリズム(2)

牧野淳一郎

2002 年 1 月 21 日

1　ノードの物理データ

力を実際に計算するまえの最後の準備として、各ノードについてそのなかの粒子の重心と総質量を出したい。これは以下のようになる。

```
void  bhnode::set_cm_quantities()
{
    if (nparticle > 1){
        int i;
        pos = 0.0;
        mass = 0.0;
        for(i=0;i<8;i++){
            if (child[i] != NULL){
                child[i]->set_cm_quantities();
                real mchild = child[i]->mass;
                pos += mchild*child[i]->pos;
                mass += mchild;
            }
        }
        pos /= mass;
    }
}
```

粒子の数が 1 であれば、既に粒子データがコピーされているので別になにもしなくていい。1 より大きければ、実際にある子ノード全部を見て、モーメント、質量を出し、モーメントを質量で割れば重心がでる。

ここでも再帰を使っている。つまり、実際に `child[i]->mass` 等を使う前に、`child[i]` について質量、重心を計算させているのである。

再帰を使うことで、ちょっと考えると大変そうな処理が非常に簡単に実現されている。

2　力の計算

さて、このように木構造ができてしまうと、実際の力の計算は非常に簡単になる。以下が力を計算する関数である。

```
void bhnode::accumulate_force_from_tree(vector & ipos, real eps2, real theta2,
                                        vector & acc,
                                        real & phi)
{
    vector dx = pos - ipos;
    real r2 = dx*dx;
    if ((r2*theta2 > l*l) || (nparticle == 1)){
        accumulate_force_from_point(dx, r2, eps2, acc, phi, mass);
    }else{
        for(int i=0;i<8;i++){
            if (child[i] != NULL){
                child[i]->accumulate_force_from_tree(ipos,eps2,theta2, acc, phi);
            }
        }
    }
}
```

これはメンバ関数になっていて、あるノードから引数で渡ってくる場所への力（加速度）とポテンシャルを計算する（渡ってきた変数に足し込む）。距離の 2 乗を計算し、それが $r^2\theta^2 > l^2$ を満たしていれば実際に加速度を計算する。2 乗のままで比べるのは、単に平方根計算をしないですませるためである。また、粒子数が 1 であればやはり同様に加速度を計算する。そうでなければ、子ノードを順番にみて重力を計算する。ここでも再帰をつかう。

質点からの力、ポテンシャルの計算は以下のようになる。

```
void accumulate_force_from_point(vector dx, real r2, real eps2,
                                 vector & acc,
                                 real & phi,
                                 real jmass)
{
    double r2inv = 1/(r2+eps2);
    double rinv  = sqrt(r2inv);
    double r3inv = r2inv*rinv;
    phi -= jmass*rinv;
    acc += jmass*r3inv*dx;
}
```

3　各関数のテスト

というわけで、一応木構造をつくって重力計算をするためのデータ構造、アルゴリズムとその実現をきわめて簡単に紹介した。これを実際に多体問題の数値解を求めるプログラムにするためには、時間積分、初期条件の生成 / 読み込み、結果の表示 / 解析等いろんなことをするプログラムを作る必要がある。

それらに入る前に、とりあえず今まで出来たところを動かしてみることにする。

割合いろんな関数を作っていろんなことをやらせるので、全体を動かして時にちゃんと動いているかどうかをどうやって判断し、さらに動いていない時にどこがおかしいかを見つけ出すのはそんなに簡単ではなくなる。

ある程度大きな、沢山の関数からできたプログラムをテスト・デバッグするための基本的な考え方は以下の 2 つである。

1. それぞれの関数を単体でテストする。
2. 組み合わせたプログラムをテストするが、動作が正しいかどうかを関数ごとにチェックする。

もちろん、プログラムをデバッグする時の大きな問題は、

- プログラムは書いた人の意図した通りにではなく、書いた通りに動く
- プログラムを書いた人の意図が正しいとは限らない

ということである。特に後者がデバッグを難しいものにする。

というわけで、まず、ツリー構造を作り、それが正しいかどうか調べる。それから実際に重力を計算し、それもツリー構造を使わないで総当りで計算したものとくらべてみよう。メインプログラムは以下のような感じになる。

```
#ifdef TEST
int main()
{
    particle * pp;
    int n;
    cerr << "Enter n:";
    cin >> n ;
    pp = new particle[n];
    double rsize = 1.0;
    create_uniform_sphere(pp, n, 0 , rsize);
    for(int i =0;i <n;i++){
        PRC(i); PRL(pp[i].pos);
    }
    bhnode * bn = NULL;

    int nnodes = n*2;
    bn = new bhnode[nnodes];
    bn->assign_root(vector(0.0), rsize*2, pp, n);
    int heap_remainder = nnodes-1;
    bhnode * btmp = bn + 1;
    bn->create_tree_recursive(btmp,  heap_remainder);
    PRL(bn->sanity_check());
    bn->set_cm_quantities();
    bn->dump();
    double  eps2 = 0.01;
    calculate_uncorrected_gravity_direct(pp, n, eps2);
    cerr << "Direct force \n";
    particle * p = pp;
    for(int i = 0; i<n; i++){
        PR(i); PR(p->pos);PR(p->phi); PRL(p->acc);
        p++;
    }
    cerr << "Tree   force \n";
    clear_acc_and_phi(pp, n);
    p = pp;
    for(int i = 0; i<n; i++){
        calculate_gravity_using_tree(p, bn, eps2, 0.5);
        PR(i); PR(p->pos);PR(p->phi); PRL(p->acc);
        p++;
    }
    return 0;
}
#endif
```

プログラム全体は

http://grape.astron.s.u-tokyo.ac.jp/%7Emakino/kougi/keisan_tenmongakuII/programs/simpletre

にあるので細かい関数については省略するが、関数 `sanity_check` はちょっと中を見ておこう。

```
int bhnode::sanity_check()
{
    int i;
    int iret = 0;
    if (nparticle == 1 ){
        // this is the lowest level node. Things to check:
        // particle is in the cell
        particle * p = pfirst;
        if(inbox(pos,p->pos,l)){
            cerr << "Error, particle out of box ... \n";
            dump();
            return 1;
        }
    }else{

        // This is the non-leaf node. Check the position and side
        // length of the child cells and then check recursively..
        for(i=0;i<8;i++){
            if (child[i] != NULL){
                int err = 0;
                err = child[i]->sanity_check();
                if (l*0.5 != child[i]->l) err += 2;
                vector relpos = cpos-child[i]->cpos;
                for (int k = 0 ; k<ndim;k++){
                    if (fabs(relpos[k]) !=l*0.25)err += 4;
                }
                if (err){
                    cerr << "Child " << i << " Error type = " << err << endl;
                    dump();
                }
                iret += err;
            }
        }
    }
    return iret;
}
```

ここでは、以下のようなチェックを行なっている。

1. ノードに粒子が 1 つだけなら、それがちゃんと中にあるかどうか。
2. そうでなければ、自分の子供は正しい位置、大きさを持つかどうか。

で、さらに再帰的に子供についても同じチェックをする。

`inbox` はこんなものである。

```
int inbox(vector  & cpos, // center of the box
          vector  & pos,  // position of the particle
          real l)         // length of one side of the box

{
    for(int  i = 0; i< ndim; i++){
        if (fabs(pos[i]-cpos[i]) > l*0.5) return 1;
    }
    return 0;
}
```

ここではベクトルの各要素についてそれぞれ調べる必要がある。

さて、この関数が「エラー」を返して来たとしても、少なくとも以下の4通りの可能性がある。

- ツリーを作る関数が間違っている。
- `sanity_check` が間違っている。
- ツリーを作る以前に既になにか間違っている。
- 上の3つの任意の組み合わせ。

というわけで、最初は粒子1つとか2つで何ができるかみるといったところから始めないと後が大変ではある。逆に数が少なければいろいろ書き出して人間がチェックするのも不可能ではない。以下の関数はツリー構造そのものを再帰的にプリントアウトする。`spc` は単に引数の数だけ空白文字を印刷する関数である。

```
void bhnode::dump(int indent)
{
    int i;
    spc(indent); cerr << "node center pos " << cpos ;
    cerr << endl;
    spc(indent); cerr << "node cm  " << pos << " m " << mass ;
    if (nparticle == 1){
        cerr << " IS LEAF" ;PRL(nparticle);
        particle * p = pfirst;
        for(i = 0; i < nparticle; i++){
            for(int j=0;j<indent+2;j++)cerr << " ";
            PRL(p->pos);
            p= p->next;
        }
    }else{
        cerr << " IS _not_ LEAF ";PRL(nparticle);
        for(i=0;i<8;i++){
            if (child[i] != NULL){
                child[i]->dump(indent + 2);
            }
        }
    }
}
```

以下は $N = 2$ での実行結果である

```
BHtree
Enter n:2
nbody = 2,  power_index = 0
i = 0,  pp[i].pos = -0.20707  0.680971  -0.293328
i = 1,  pp[i].pos = -0.106833  -0.362614  0.772857
bn->sanity_check() = 0
node center pos 0  0  0
node cm  -0.156952  0.159178  0.239765 m 1 IS _not_ LEAF nparticle = 2
  node center pos -0.5  -0.5  0.5
  node cm  -0.106833  -0.362614  0.772857 m 0.5 IS LEAFnparticle = 1
    p->pos = -0.106833  -0.362614  0.772857
  node center pos -0.5  0.5  -0.5
  node cm  -0.20707  0.680971  -0.293328 m 0.5 IS LEAFnparticle = 1
    p->pos = -0.20707  0.680971  -0.293328
Direct force
i = 0 p->pos = -0.20707  0.680971  -0.293328 p->phi = -0.33364 p->acc = 0.014891  -0.155032  0.158389
i = 1 p->pos = -0.106833  -0.362614  0.772857 p->phi = -0.33364 p->acc = -0.014891  0.155032  -0.158389
Tree   force
i = 0 p->pos = -0.20707  0.680971  -0.293328 p->phi = -0.33364 p->acc = 0.014891  -0.155032  0.158389
i = 1 p->pos = -0.106833  -0.362614  0.772857 p->phi = -0.33364 p->acc = -0.014891  0.155032  -0.158389
```

それらしいツリー構造ができていること、力の計算結果が総当りとツリーを使ったもので一致していることがわかる。もちろん、粒子が2個の時にはツリーを使っても相手は1個なので、結果は完全に一致しなければおかしい。まあ、確かに一致しているというわけである。

2粒子でそれらしく動いたからといって正しいということにはならない。もっと大きな粒子数でいろいろ調べ、さらに opening angle も変えて調べる必要があるがこれは課題ということにする。

4　時間積分

時間積分には前にやったリープフロッグ公式を使うことにする。どんなふうに書いてもいいが、今回は、分かりやすいと思われる

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

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

の形を使うことにしよう。以下の関数を particle クラスに入れる。

```
        void predict(real dt){
            real dt2 = dt*dt*0.5;
            pos += dt*vel + dt2*acc;
            vel +=  (dt*0.5)*acc;
        }
        void correct(real dt){
            vel +=  (dt*0.5)*acc;
        }
```

時間積分は以下のように、まず予測子を計算、それから加速度を計算し、最後に速度の修正子を計算する。

```
void integrate(bhnode * bn,
               int nnodes,
               particle * pp,
               int n,
               real eps2,
```

```
               real theta,
               real dt)
{
    for(int i = 0;i<n;i++) pp[i].predict(dt);
    calculate_gravity(bn,nnodes,pp,n,eps2,theta);
    for(int i = 0;i<n;i++) pp[i].correct(dt);
}
```

重力を計算する関数はこんな感じになる。

```
void calculate_gravity(bhnode* bn,
                       int nnodes,
                       particle * pp,
                       int n,
                       real eps2,
                       real theta)
{
    real rsize = calculate_size(pp,n);
    bn->assign_root(vector(0.0), rsize*2, pp, n);
    int heap_remainder = nnodes-1;
    bhnode * btmp = bn + 1;
    bn->create_tree_recursive(btmp,  heap_remainder);
    //    bn->dump();
    //    PRL(bn->sanity_check());
    bn->set_cm_quantities();
    clear_acc_and_phi(pp, n);

    particle * p = pp;
    for(int i = 0; i<n; i++){
        calculate_gravity_using_tree(p, bn, eps2, theta);
        //        PR(i); PR(p->pos);PR(p->phi); PRL(p->acc);
        p++;
    }
}
```

これは基本的には前のテスト用のメインプログラムから重力計算のところを切り出してまとめただけである。

5 プログラム全体の構造

さて、これまで断片的にいろいろな関数を説明してきたが、プログラム全体としてはどんな構造にすればいいだろうか ?やらないといけないことは以下のようなものである。

1. ソフトニング、時間刻みなどのパラメータを読み込む。
2. 初期条件を作るなり、ファイルから読み込むなりする。
3. 重力を計算する。
4. 以下のループをシミュレーション時刻の終わりまで繰り返す。
5. 予測子を計算。
6. 重力を計算。
7. 修正子を計算。
8. 出力が必要であればなんか書く。

```
int main()
{
    particle * pp;
```

```
    int n;
    cerr << "Enter n:";
    cin >> n ;
    pp = new particle[n];
    double rsize = 1.0;
    create_uniform_sphere(pp, n, 0 , rsize);
    bhnode * bn = NULL;

    int nnodes = n*2+100;
    bn = new bhnode[nnodes];
    real eps2;
    real theta;
    real dt;
    real tend;
    int iout;
    cerr << "Enter eps2, theta, dt, tend, iout:";
    cin >> eps2 >>theta >>dt >>tend >> iout;
    cerr << "eps2=" << eps2 << " theta=" <<theta <<" dt=" << dt
         << " tend=" << tend << " iout=" << iout <<endl;
    calculate_gravity(bn,nnodes,pp,n,eps2, theta);
    cerr << "Initial data"<<endl;
    print_energies(pp,n,0);
    int istep = 0;
    for(real t=0;t<tend; t+=dt){
        integrate(bn, nnodes, pp, n, eps2, theta, dt);
        istep++;
        if (istep % iout == 0){
            cerr << "Time=" << t+dt <<endl;
            print_cm(pp, n);
            print_energies(pp,n,1);
        }
    }
    return 0;
}
```

説明してない関数が沢山あるが、説明は来週に回してちょっと動かしてみよう。

6 練習

1. 参考プログラム

   http://grape.astron.s.u-tokyo.ac.jp/~makino/
   kougi/keisan_tenmongakuII/programs/index.html

   にある BHtree.C を動かしてみて、総当たりで計算するのと木構造で計算するのとで計算した加速度、ポテンシャルの値を比べ、θ を小さくしたときに差がどのように小さくなるか、また小さくなりかたは粒子数によってどのように変わるか調べてみよ。
2. 上の誤差が理論的にはどうなるべきか考察せよ。
3. 計算時間はどうなるか、粒子数 100 程度から 10,000 程度まででで調べてみよ。