Draft version January 16, 2023
Typeset using LaTeX **twocolumn** style in AASTeX631

重力多体系シュミレーションの性能評価に関する考察 *

西濱 大将 [1]

[1] 埼玉大学理学部物理学科，*20RP021*

(Received March 1, 2021; Revised April 1, 2021; Accepted January 16, 2023)

ABSTRACT

本レポートでは，重力多体系のシミュレーションに有用な 4 次のルンゲ・クッタ法を用いて，Newton の万有引力の法則に基づくプログラムを作成し，その性能を評価することを目的としている．RK4 による数値解法の修正と近距離カットオフを導入し，さらに計算量を減らすために Barnes-Hut アルゴリズムを導入する．結果として，BH アルゴリズムを使用した場合には，RK4 法に比べ計算コストが高くなってしまい計算時間が遅くなったため，本論文では BH アルゴリズムは使用せず，RK4 のみのプログラムで計算を行った．またそれを用いて宇宙の大規模構造 (泡構造) の再現，銀河 (恒星) の生成過程，銀河同士の衝突を見る．銀河 (恒星) の生成過程では運動エネルギーが必ず 2 乗に比例して変化して行くことが分かった．これについて今後考察をして，さらに他の相互作用も含めたモデルなども勉強していきたい．

Keywords: 重力多体系, シミュレーション, 4 次のルンゲ・クッタ法, Newton の万有引力の法則, 近距離カットオフ, Barnes-Hut アルゴリズム, 泡構造, 銀河, 運動エネルギー

1. 序論

1.1. 背景・目的

重力で相互作用する多数の粒子からなる系を重力多体系といい，粒子数が N 数あるとき N 体系や N 体問題などといったりする．$N > 2$ のときは解析的な解は存在せず，数値計算を用いたシミュレーションが必要不可欠である．また，天文学では惑星リング，惑星系，星団，銀河，銀河団などが重力多体系と近似することができ，それらの生成過程や衝突過程をシミュレーションすることができるとされている．

本レポートでは 4 次のルンゲ・クッタ法 (RK4) を用いて，Newton の万有引力の法則のみに基づく重力多体系のプログラムを作成し，その性能評価を行う．重力多体系の問題は，球状星団や銀，銀河団といった多数の恒星からなる天体の時間進化や動力学を数値的にシミュレーションするのに有効であるとされている．球状星団で約 10^{4-5} 個，銀河では約 10^{10-12} 個という膨大な個数の恒星が含まれ，それら個々の星の間に働く重力を全て計算する必要がある．そのようなことから，東京大学と国立天文台 (NAOJ) が共同で重力多体問題に特化した専用計算機 (GRAPES) が開発された．これは $\mathcal{O}(N^2)$ の計算量を専用ハードウェアを用いることで計算速度を飛躍的に加速させることを目的としている．また GRAPES では

- 球状星団の重力熱力学的進化
- 銀河の動力学的進化
- 銀河の衝突・合体
- 大質量ブラックホールを持つ銀河中心核の構造
- 銀河団の動力学的進化
- 宇宙論的大規模構造形成
- 原始惑星系円盤での惑星形成
- ジャイアント・インパクトに伴う月の形成
- 惑星の環の構造・進化

のような問題をシミュレーションしてきており，多くの科学的成果を挙げている．このようなことからも，本レポートで作成したプログラム自体も同様の有用性と活用力をもっているといえ，それを用いて様々な天文学的物理現象を説明することを試み，シミュレーションを通して気づいたことをまとめる．

1.2. 注意

本レポートで用いたプログラミング言語は Julia を使用し，Matplotlib を使用しグラフを作成した．また出力した画像を動画にするには FFmpeg を使用した．

2. 数値解法 (SCHEME)

2.1. *RK4* の修正

* Released on March, 1st, 2021

物理学実験 IIa (計算機実験) で使用した RK4 は一階微分方程式に有用である．高階の微分方程式にも適応することはできるが，単純に定義に則り使用すると，大きな誤差が発生する．新しい変数を導入し，連立一階微分方程式の形に変換して解くことで誤差が最小化される．二階常微分方程式は

$$\frac{\mathrm{d}^2y}{\mathrm{d}x^2} = f\left(x, y, \frac{\mathrm{d}y}{\mathrm{d}x}\right) \tag{1}$$

と表わせる．$\mathrm{d}y/\mathrm{d}x = z$ とおくと

$$\frac{\mathrm{d}y}{\mathrm{d}x} = z, \tag{2}$$
$$\frac{\mathrm{d}z}{\mathrm{d}x} = f(x, y, z) \tag{3}$$

となる．ある $x = x_n$ での $y = y_n$，$z = z_n$ の値が既知であるとし，$x = x_{n+1} = x_n + h$ での y_n と z_n を計算する．RK4 より

$$k_1 = hz_n, \tag{4}$$
$$l_1 = hf(x_n, y_n, z_n), \tag{5}$$
$$k_1 = h\left(z_n + \frac{l_1}{2}\right), \tag{6}$$
$$l_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}, z_n + \frac{l_1}{2}), \tag{7}$$
$$k_3 = h\left(z_n + \frac{l_2}{2}\right), \tag{8}$$
$$l_3 = hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}, z_n + \frac{l_2}{2}) \tag{9}$$
$$k_4 = h(z_n + l_3), \tag{10}$$
$$l_4 = hf(x_n + h, y_n + k_3, z_n + l_3), \tag{11}$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 3k_3 + k_4), \tag{12}$$
$$z_{n+1} = z_n + \frac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4) \tag{13}$$

がいえる．

2.2. 近距離カットオフ

2 つの N 体粒子間の距離が極めて小さくなると，両者の間に働く重力は任意に大きくなり得る．本シミュレーションでは無衝突系を扱うため，N 体粒子は真の粒子ではなく，多数の粒子が占める位相空間上の領域を表す．そのため，N 体粒子間に働く重力が近距離で発散する効果は物理的ではなく，適当なカットオフにより除去される必要がある．最も簡単なカットオフとしては重力ポテンシャル Φ を

$$\Phi(r) = -\frac{GM}{\sqrt{r^2 + \varepsilon^2}} \tag{14}$$

へと変更する．ここに ε は距離の次元を持つ定数で，この距離スケールより近距離での重力の発散を抑えるられる．このポテンシャルは N 体粒子がプラマーモデルであるような質量分布を持つと仮定した場合に得られるものに等しく，ε の値は平均粒子間距離のオーダーに選ばれる．

2.3. *Barnes-Hut (BH)* アルゴリズム

Barnes-Hut アルゴリズムは，N 体問題を効率的に計算量を削減でき，1986 年に Josh Barnes と Piet Hut によって発表された．その後，多くの N 体問題で採用されてきた．すべての粒子間の相互作用を直接計算するのではなく，ツリーベースの近似方式を用いることで，問題の計算量を $\mathcal{O}(N^2)$ から $\mathcal{O}(N \log N)$ へと減らすことができ，計算時間の短縮が見込める．

これは，粒子をグループ化することで計算回数を減らす．計算する粒子間が十分に遠方である複数の粒子 (粒子群) は，粒子群の重心に位置する仮想粒子で近似できる．例えば，アンドロメダ銀河が天の川に及ぼす力は，アンドロメダ銀河の重心に位置する仮想粒子で近似することができる．2 つの銀河の距離が十分に大きければ，アンドロメダ銀河のすべての星について計算する必要はなくなり，重心に位置する仮想粒子のみを計算すればよい．この近似は，点群から粒子までの距離が大きく，点群の半径が点群から粒子までの距離に対して小さければ成立する．群距離 d と群半径 r の比を多極子許容基準 (MAC) と呼ぶ:

$$\theta = \frac{d}{r} \tag{15}$$

θ がある閾値を上回ると近似精度が悪くなり，誤差が大きくなる．Barnes と Hut は $\theta = 1$ を使っている．一般的には θ が小さいほど，シミュレーションの精度は良い．BH アルゴリズムでは，この基本原則を応用し，シミュレーション領域の空間を再帰的に細分化し，それぞれの領域間の相互作用を考える．領域を等方的に左上・左下・右上・右下の 4 つに再帰的に分割する，この方法を Quadtree(領域四本木) という．

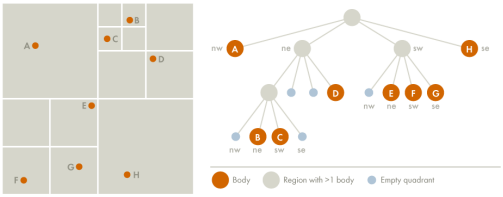

Figure 1.

図 1 のように再帰的に分割した領域を各内部ノードが 4 個までの子ノードを持つ木構造のデータ構造に対応させる．

2.3.1. *Quadtree* の構築

Quadtree を構成するには、ボディを次々に挿入していく。ノード x を根とするする木に物体 b を挿入するには，以下の再帰的手続きを行う。

(1) ノード x が物体を含んでいない場合，新しい物体 b をここに置く．

(2) ノード x が内部ノードの場合，x の質量中心と全質量を更新し，物体 b を適切な象限に再帰的に挿入する．

(3) ノード x が外部ノード，たとえば c という名前のボディを含む場合，同じリージョンに 2 つのボディ b と c があることになる．4 つの子を作ることによって，さらに領域を細分化する．次に，b と c の両方を適切な象限に再帰的に挿入する．b と c はまだ同じ象限内にある可能性があるので，一回の挿入で何回か分割されるかもしれない．最後に，x の重心と全質量を更新する。

2.3.2. 物体に作用する力の計算

ここでは物体に作用する力の計算方法について具体的に説明する．物体 b に作用する力を計算するには，Quadtree の根ノードから始めて，以下の再帰的な手順を用いる．

(1) 現在のノードが外部ノード（子を持たないノード）である場合，現在のノードが b に及ぼす力を計算し，その量を b の正味の力に加える．

(2) そうでない場合は，比 s/d を計算する．$s/d < \theta$ ならば，この内部ノードを 1 つの物体として扱い，物体 b に及ぼす力を計算し，その量を b の力に加える．

(3) そうでなければ，現在のノードの子ノードに対して再帰的にこの手順を実行する．

例として，図 1 における物体 a に作用する正味の力を計算する．また，物体 a, b, c, d, e, f の質量はそれぞれ 1, 2, 3, 4, 5, 6 kg とし，内部ノードである根は，内部に含む重心質量を表すものとする．

(1) 最初に根ノードを調べる．物体 A とノードの重心（つまり，ここではすべての物体の重心のことで，図 2 では白い点に対応する）を比較すると，比 $s/d = 100/43.1 > \theta = 0.5$ となるので，根の各子要素に対して再帰的に処理を行う．

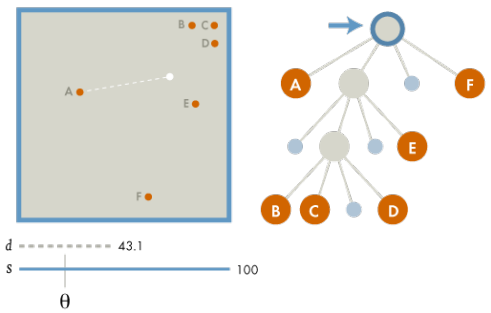

Figure 2.

(2) 最初の子は物体 a 自身であるため，何もしない．

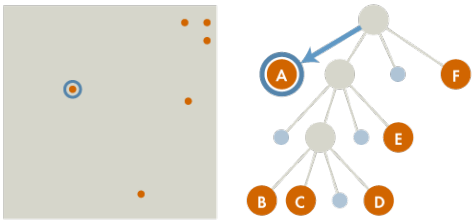

Figure 3.

(3) 次の子は空間の右上の象限を表し，物体 b, c, d, e を含みそれらの重心を表す．ここで比 $s/d = 50/62.7 > \theta$ なので，子ノードに対して力を再帰的に計算する．最初の子ノードは空なのでこれは飛ばす．

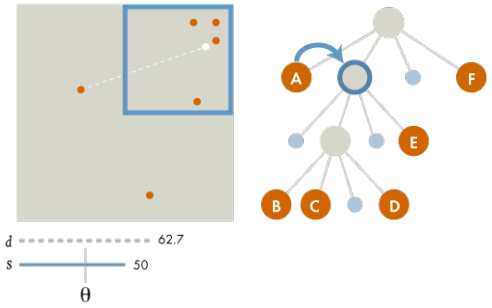

Figure 4.

(4) 次のノードは内部ノードでもあり，親ノードの右上象限を表し，物体 b, c, d を含み，それらの重心を表す．$s/d = 25/66.9 < \theta$ なので，この内部ノードを b,c,d の重心を一つの物体として扱い，物体 a にかかる力を計算し，その値を a にかかる正味の力に加える．このノードの親にはもう子供がいないので，引き続き根ノードの他の子を調べる．

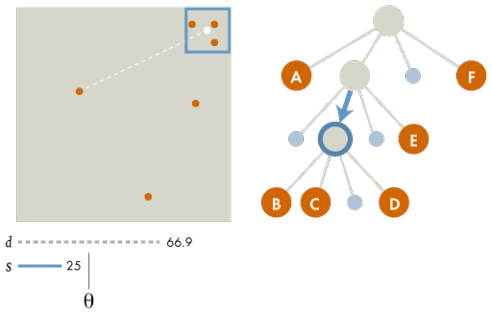

Figure 5.

(5) 次の子は外部ノードなので，物体 a と e の間の力を計算し，これを a の正味の力に加えればよい．

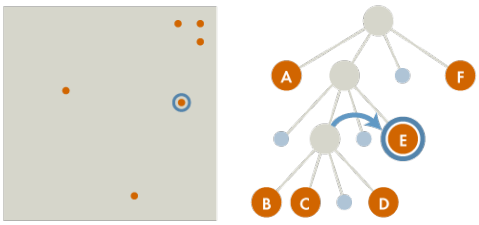

Figure 6.

(6) このレベルのノードをすべて調べた後，親ノードの次のノードに移動する．これは外部ノードなので，a と f の間の力を計算し，これを a の正味の力に加える．

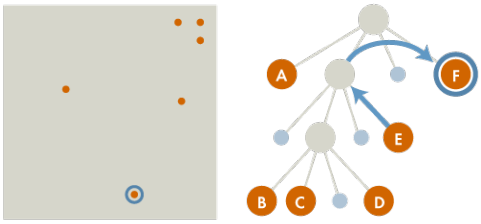

Figure 7.

3. 計算結果

3.1. *RK4* 法と *BH* アルゴリズムの計算時間の検証

本レポートでは，RK4 法と BH アルゴリズムを用いた．多くの場合，BH アルゴリズムを用いたほうがより高速に計算することができるのだが，計算時間を検証したところ BH アルゴリズムを用いたほうが非常に遅くなった．

それぞれのソースコードを Code1 と Code 2 に示す．BH アルゴリズムを使用した場合，RK4 によって 1 ステップごとに 4 回の Quadtree を構築，計算を行う必要がある．これにより単純に RK4 を使用した場合に比べ計算コストがかかってしまうと考えられる．

このようなことから，本レポートでは BH アルゴリズムは使用せず RK4 のみのプログラムで計算を行い，$N = 2000$ に留めることにする [1]．

3.2. 宇宙の大規模構造

数百から数千の銀河が集まり，銀河群，銀河団を形成しており，それが更に集まって超銀河団を形成する．超銀河団は平面状の壁のような分布をしており，それを銀河フィラメントと呼ぶ．

1 枚の銀河フィラメントと他の銀河フィラメントとの間には光を発する天体がほとんど無い領域があり，これを超空洞（ボイド）と呼ぶ．

[1] BH アルゴリズムでは RK4 法ではなく Euler 法で代替することで銀河を構成する恒星数の $N = 10^{10-12}$ ほどで計算を行うなどにより，N 体問題を高速化することができると考えられる．

宇宙の大規模構造は銀河フィラメントと超空洞が複雑に入り組んだ構造であり，これが泡のような構造であることから，「宇宙の泡構造」と呼んだりもする (図 8 [2])．

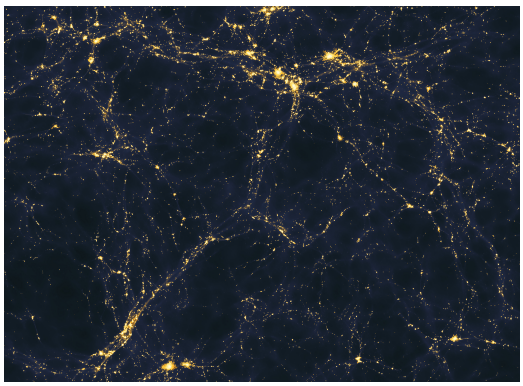

Figure 8. 宇宙の大規模構造

本レポートでは次のように初期条件を与えた．

初期位置: 一様乱数で $-2 < x < 2$, $-2 < y < 2$

初期速度: 一律 0

このシュミレーションによって図 9 のような過程を辿った．ただし，泡構造のような安定状態を作り出したわけではなく，途中過程で泡構造を生成したに過ぎない．また，1 粒子を銀河と仮定し，質量を $2 \times 10^{12} M_{\odot}$ とする．またシミュレーションでは，$N = 2000$ としたため，実際の粒子数と乖離する．そこで粒子総数が実際の場合と同じになるように設定した質量に $1.5 \times 10^6/2000$ をかけた．これによって，重力定数の関係からおよそのシミュレーションの距離オーダーと，時間オーダーが分かる．

泡構造は非常に広範囲を観測したとき，およそ 100 Mpc - 900 Mpc のオーダーであることから，これを元に初期状態からの経過時間を計算すると，およそ 1 Gyear と分かる．一般に 138 億年 (13 Gyear) 前に宇宙は誕生したとされ，それよりも十分に少ない．結果は適切であると考えられる．

また，先述したようにこの泡構造は安定状態を作り出したわけではない．そのため実際の (安定的な?) 宇宙の大規模構造と同じ意味の泡構造を再現するには，CDM モデル等を考える必要があるのかもしれない．

3.3. 銀河の生成過程

3.2 節でのシミュレーションの距離オーダーを変えて，1 粒子あたりの質量を変更すると銀河や恒星の形成過程等のシミュレーションに変えることができる．すると安

[2] Andrew Pontzen and Fabio Governato - Andrew Pontzen and Hiranya Peiris, http://www.ucl.ac.uk/mathematical-physical-sciences/news-events/maps-news-publication/maps1423

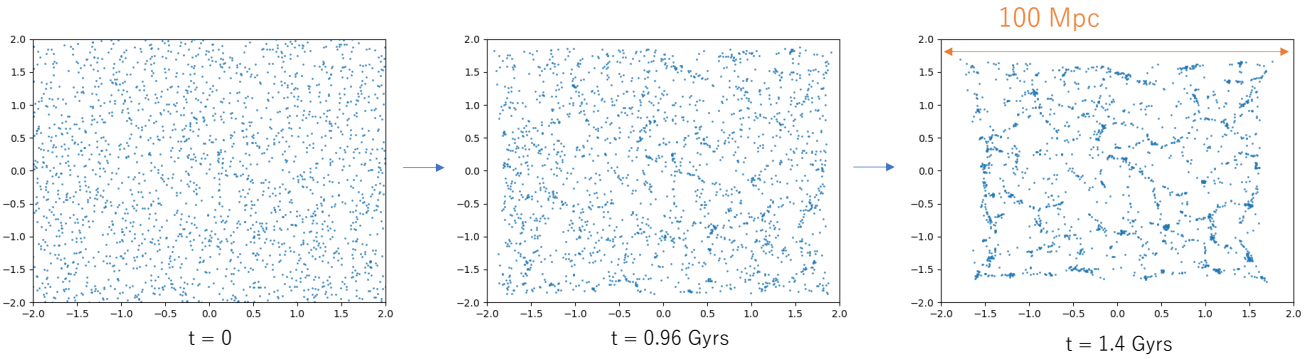

Figure 9. 泡構造の形成過程

定状態において銀河（もしくは恒星）の形成を見ることができた．

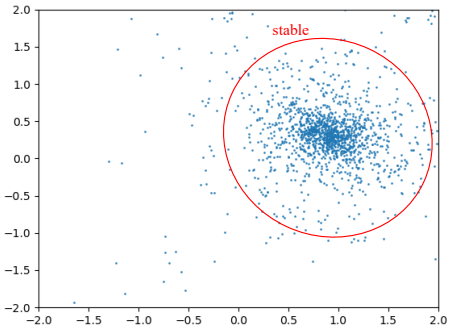

Figure 10. 銀河 (恒星) 形成過程の安定状態

また，運動エネルギーの変化を図 11 に示す．初期状態から 2 乗に比例して急激に運動エネルギーが増加し，ある瞬間を過ぎるとまた急激に安定状態へ移行することが分かる．安定状態になるとほとんど運動エネルギーに変化は見られないことが分かった．

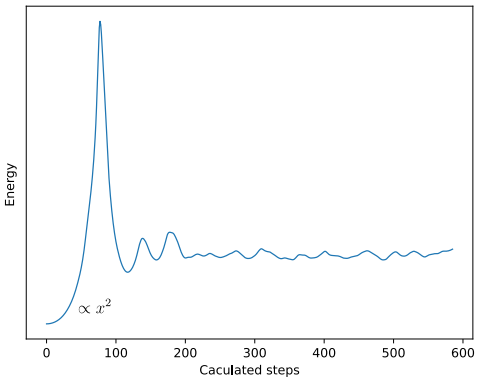

Figure 11. 銀河 (恒星) 形成過程の運動エネルギーの変化

3.4. 銀河同士の衝突

銀河同士の衝突を観察するために，次のように初期条件を与えた．そのときの様子を図 12 に示す．

初期位置: 半分ずつ粒子は $(x, y) = (\pm 1, 0)$ を中心に分散 0.3 に従った正規分布に配置する．

初期速度: $(x, y) = (\pm 1, 0)$ を中心に反時計回りに角速度 ω $(0 < \omega < 2)$ を一様乱数で与える．

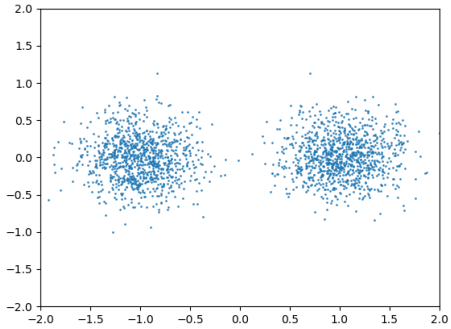

Figure 12. 2 つの銀河の初期位置

銀河同士の衝突では図 13 に示すように互いに連星運動をしながら，1 つに統合していく．また図 14 に系全体の運動エネルギーを示す．ここからお互いに銀河が接触し合う衝突瞬間に非常に大きな変化が生じている．1 つにまとまった後では運動エネルギーも安定していることが見てわかる．

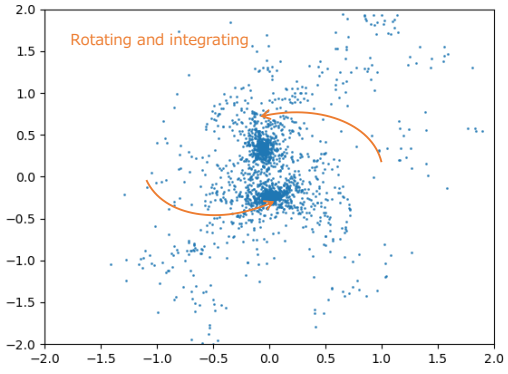

Figure 13. 銀河同士の衝突の瞬間

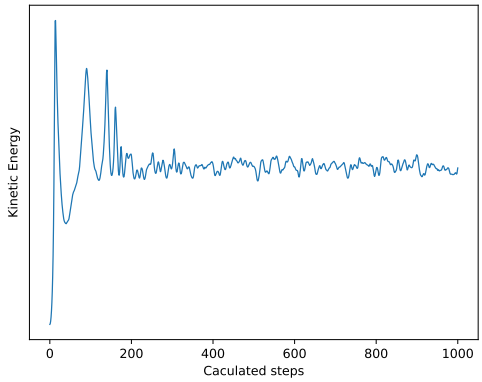

Figure 14. 銀河同士の衝突の運動エネルギーの変化

4. まとめ

本実験を通して N 体系の有用性を理解し，様々な物理現象へ応用することができることがよく分かった．また銀河 (恒星) 形成過程では運動エネルギーが必ず 2 乗に比例して変化していくことが分かった．この理由について考察の余地がある．本レポートではそれについて考察する時間ならびに知識が十分に無かったため，理由付けをすることができなった．それを含めて今後，N 体系の様々なアルゴリズム，Newton の万有引力だけでなく他の相互作用も加味したものなどについても勉強していきたい．

APPENDIX

A. 作成した動画について

本レポートで作成した動画は以下の YouTube ページに公開しておく．

銀河 (恒星) の形成過程: https://youtu.be/AAVSs8ybZGQ (本レポートで言及済み)

銀河同士の衝突 1: https://youtu.be/SVPcPVxRUKU (粒子の分散を大きくしすぎた版)

銀河同士の衝突 2: https://youtu.be/JFtwn328eaY (本レポートで言及済み)

B. ソースコード

Code 1. 状態を更新する関数

```
function update(r0, v0)
    k1 = copy(v0)
    l1::Vector{Vector{Float64}} = [zeros(2) for i in 1:N]
    for i in 1:N
        t::Vector{Float64} = zeros(2)
        for j in 1:N
            if i != j
                t .+= f(r0[i], r0[j])
            end
        end
        l1[i] = t
    end

```

```
    k2 = v0 .+ dt/2 .* l1
    l2::Vector{Vector{Float64}} = [zeros(2) for i in 1:N]

    for i in 1:N
        t::Vector{Float64} = zeros(2)
        for j in 1:N
            if i != j
                t .+= f(r0[i] + dt/2 * k1[i], r0[j] + dt/2 * k1[j])
            end
        end
        l2[i] = t
    end

    k3 = v0 .+ dt/2 .* l2
    l3::Vector{Vector{Float64}} = [zeros(2) for i in 1:N]

    for i in 1:N
        t::Vector{Float64} = zeros(2)
        for j in 1:N
            if i != j
                t .+= f(r0[i] + dt/2 * k2[i], r0[j] + dt/2 * k2[j])
            end
        end
        l3[i] = t
    end

    k4 = v0 .+ dt .* l3
    l4::Vector{Vector{Float64}} = [zeros(2) for i in 1:N]

    for i in 1:N
        t::Vector{Float64} = zeros(2)
        for j in 1:N
            if i != j
                t .+= f(r0[i] + dt * k3[i], r0[j] + dt * k3[j])
            end
        end
        l4[i] = t
    end

    k = (k1 .+ 2 .* k2 .+ 2 .* k3 .+ k4) ./ 6
    l = (l1 .+ 2 .* l2 .+ 2 .* l3 .+ l4) ./ 6

    r = r0 .+ dt .* k
    v = v0 .+ dt .* l

    return r, v
end
```

Code 2. BH アルゴリズムを使用した状態を更新する関数

```
function calc(r)
    xmax = maximum([abs(r[i][1]) for i in 1:N])
    ymax = maximum([abs(r[i][2]) for i in 1:N])
    xymax = maximum([xmax, ymax])
    xymin = - xymax
    s = xymax - xymin
    xmax = xymax
    ymax = xymax
    xmin = xymin
    ymin = xymin
    treelist = []
    d = [i for i in 1:N]
```

```
    push!(treelist,[mean([r[i][1] for i in 1:N]), mean([r[i][2] for i in 1:N]), N, 0, d, xmin,
        xmax, ymin, ymax])
    _treelist = segmentation(xmin,xmax,ymin,ymax,r,d,0,treelist)
    cat(treelist, _treelist, dims=1)
    sort!(treelist, by= x -> x[4])

    l = [zeros(2) for i in 1:N]
    for i in 1:N
        caculated::Vector{Int64} = []
        t = zeros(2)
        for j in 1:N
            if i != j
                if !(j in caculated)
                    w = r[i] - treelist[j][1:2]
                    s = treelist[j][7] - treelist[j][6]
                    theta = s/norm(w)
                    if theta < 10
                        cat(caculated, treelist[j][5], dims=1)
                        t .+= f(treelist[j][3]*M, r[i], treelist[j][1:2])
                    else
                        continue
                    end
                end
            end
        end
        l[i] = t
    end
    return l
end

function segmentation(xmin,xmax,ymin,ymax,r,d,depth,treelist)
    n = length(r)
    s = xmax - xmin
    NSEWcount::Vector{Int64} = [0,0,0,0]
    NSEWparticlelist::Vector{Vector{Int64}} = [[],[],[],[]]
    NSEWposition_x = [0.0 for i in 1:4]
    NSEWposition_y = [0.0 for i in 1:4]
    xmaxlist = [xmax, xmin + s/2, xmin + s/2, xmax]
    xminlist = [xmax - s/2, xmin, xmin, xmax - s/2]
    ymaxlist = [ymax, ymax, ymin + s/2, ymin + s/2]
    yminlist = [ymax - s/2, ymax - s/2, ymin, ymin]

    for i in d
        if (xminlist[1] <= r[i][1] <= xmaxlist[1]) && (yminlist[1] <= r[i][2] <= ymaxlist[1])
            areanum = 1
        elseif (xminlist[2] <= r[i][1] <= xmaxlist[2]) && (yminlist[2] <= r[i][2] <= ymaxlist[2])
            areanum = 2
        elseif (xminlist[3] <= r[i][1] <= xmaxlist[3]) && (yminlist[3] <= r[i][2] <= ymaxlist[3])
            areanum = 3
        elseif (xminlist[4] <= r[i][1] <= xmaxlist[4]) && (yminlist[4] <= r[i][2] <= ymaxlist[4])
            areanum = 4
        end

        push!(NSEWparticlelist[areanum], i)
        NSEWcount[areanum] += 1
        NSEWposition_x[areanum] += r[i][1]
        NSEWposition_y[areanum] += r[i][2]

    end

    for i in 1:4
        if NSEWcount[i] == 1
```

```
74              push!(treelist,[NSEWposition_x[i]/NSEWcount[i], NSEWposition_y[i]/NSEWcount[i],
                    NSEWcount[i], depth + 1, NSEWparticlelist[i], xminlist[i], xmaxlist[i], yminlist[i
                    ], ymaxlist[i]])
75          elseif NSEWcount[i] > 1
76              push!(treelist,[NSEWposition_x[i]/NSEWcount[i], NSEWposition_y[i]/NSEWcount[i],
                    NSEWcount[i], depth + 1, NSEWparticlelist[i], xminlist[i], xmaxlist[i], yminlist[i
                    ], ymaxlist[i]])
77              segmentation(xminlist[i], xmaxlist[i], yminlist[i], ymaxlist[i], r, NSEWparticlelist[i],
                    depth+1,treelist)
78          end
79      end
80      return treelist
81  end
```

REFERENCES

Barnes, J., & Hut, P. ????, 324, 446, doi: 10.1038/324446a0

Ingo Berg. ????, The Barnes-Hut Galaxy Simulator. https://beltoforion.de/en/barnes-hut-galaxy-simulator/

Jim Demmel. ????, CS267: Notes for Lecture 24, Apr 11 1996. https://people.eecs.berkeley.edu/~demmel/cs267/lecture26/lecture26.html

Tom Ventimiglia, Kevin Wayne. ????a, COS 126 Programming Assignment: Barnes-Hut Galaxy Simulator. https://www.cs.princeton.edu/courses/archive/fall04/cos126/assignments/barnes-hut.html

—. ????b, The Barnes-Hut Algorithm. http://arborjs.org/docs/barnes-hut