天体物理学実習 II　6 月 28 日のレポート課題

遠山翔太

2022/7/4

課題１

原始惑星系円盤の進化についてのコードは以下のとおりである。

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

void initialize(double *u, double xmin, double xmax,
                double dx, int n, int ngh) {
  for (int i = ngh; i < n+ngh; ++i) {
    double x = xmin+(i-ngh+0.5)*dx;
    //  *** 初期条件を設定
    double sx = sqrt(x);
    u[i] = 2000.0/(sx*sx*sx)*exp(-x/50.0); // 距離は AU を単位としている。
  }
  return;
}

double cs2(double r) {
  return 0.044/sqrt(r);
}

void boundary_left(double *u, int n, int ngh, double t) {
  // *** 内側の境界条件を設定
  u[0] = 2.0*u[1]-u[2];
  if (u[0] < 0.0) u[0] = 0.0;
  return;
}
```

```
void boundary_right(double *u, int n, int ngh, double t) {
  // *** 外側の境界条件を設定
  u[n+ngh] = 2.0*u[n]-u[n-1];
  if (u[n+ngh] < 0.0) u[n+ngh] = 0.0;
  return;
}

void output(double *u, double t,
            double xmin, double xmax, double dx,
            int n, int ngh, int step) {
  char fn[32];
  sprintf(fn, "output%04d.dat", step);
  FILE *fp;
  fp = fopen(fn, "w");
  if (fp == NULL) {
    printf("Error: cannot open file %s\n", fn);
    exit(1);
  }

  fprintf(fp, "# t=%f, xmin=%f, xmax=%f, dx=%f, n=%d, ngh=%d, step=%d\n",
          t, xmin, xmax, dx, n, ngh, step);
  for (int i = 1; i < n+ngh; ++i) {
    double x = xmin+(i-ngh+0.5)*dx;
    fprintf(fp, "%d %g %g\n", i, x, u[i]);
  }
  fclose(fp);
  return;
}

int main(void) {
  const int NX = 1000, NGH = 1;
  const int NT = 10000000, NOUT = NT/10; // *** 適切な NT を自分で決めよ
  const double xmin = 1.0, xmax = 1000.0, tend = 1e7;
  const double alpha = 0.01;
  const double C = 0.05;
  double dx = (xmax-xmin)/NX, dt = tend/NT, t = 0.0;
  double u[NX+2*NGH], u1[NX+2*NGH];

  initialize(u, xmin, xmax, dx, NX, NGH);
```

```
  boundary_left(u, NX, NGH, t);
  boundary_right(u, NX, NGH, t);

  for (int n = 0; n < NT; ++n) {
    if (n%NOUT == 0)
      output(u, t, xmin, xmax, dx, NX, NGH, n/NOUT);

    for (int i = NGH; i < NX+NGH; ++i) {
      double x = xmin+(i-NGH+0.5)*dx;
      // *** 時間発展の式を自分で書け
      double B1,B2,B3;
      B1 = u[i+1]*(x+dx)*(x+dx)*0.044/sqrt(x+dx); //今は dx = 1.0 である。
      B2 = u[i]*x*x*0.044/sqrt(x);
      B3 = u[i-1]*(x-dx)*(x-dx)*0.044/sqrt(x-dx);
      u1[i] = u[i] + dt*C*alpha*(sqrt(x+0.5*dx)*(B1-B2)/dx-sqrt(x-0.5*dx)*(B2-B3)/dx)/x/dx;
    }
    for (int i = NGH; i < NX+NGH; ++i)
      u[i] = u1[i]; // 計算した結果を u1 から u にコピー

    t += dt;
    boundary_left(u, NX, NGH, t);
    boundary_right(u, NX, NGH, t);
  }

  output(u, t, xmin, xmax, dx, NX, NGH, NT/NOUT);

  return 0;
}
```

拡散のタイムスケールは、

$$\Delta t \sim \frac{\Delta R^2}{C\alpha c_s^2 R^{\frac{3}{2}}} \sim \frac{\Delta R^2}{0.05 \times 0.01 \times 0.044 R}$$

より R が大きいほど小さくなる。今 $1000AU$ までを考えているので、$R = 1000$ を代入すると、

$$\frac{\Delta R^2}{0.05 \times 0.01 \times 0.044 \times 1000} = 45.45...$$

となるので main 関数の中の NT は dt がみつもられた $\Delta t \sim 45$ よりも何倍か小さい値になるように設定すべきである。ここで、$dt = 10$ となるように $NT = 1000000$ として動作させた結果を図 1 に示す。また、dt を

設定する目安としてより細かく計算させるために $dt = 1$ となるように $NT = 10000000$ として動作させた結果を図 2 に、より荒く $dt = 100$ となるように $NT = 100000$ として動作させた結果を図 3 に示す。

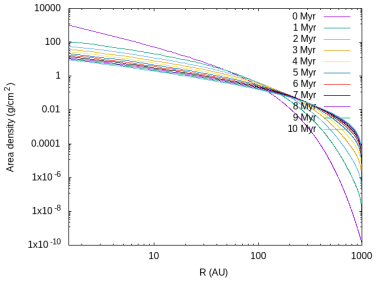

図 1　$dt = 10$

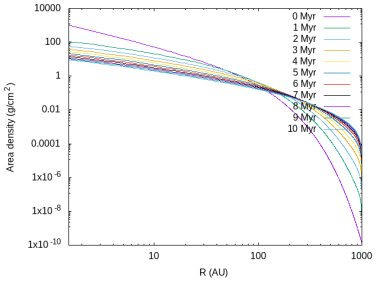

図 2　$dt = 1$

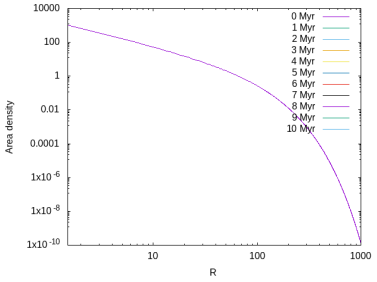

図 3　$dt = 100$

図 3 は 1 Myr 以降すなわち output0001.dat から output0010.dat の面密度が nan になってしまったので、このように線が一本しか惹かれていない。図 1、図 2 は遜色なく、そしてすべての dat ファイルに数値が書き込まれているので、$dt = 10$ でも進化を再現できると考え、以下では $NT = 1000000$ を採用する。

$T = 0, 10^5, 10^6, 10^7$ 年における分布を重ねたグラフを描くために、まず $T = 10^5$ 年でのデータを出力させるべく新たに main 関数内のループに以下のような if 分を新たに加えた。

```
  for (int n = 0; n < NT; ++n) {
    if (n%NOUT == 0) // NOUT = NT/10 より tend = 10e7 を 10 等分した間隔すなわち 10e6 ごとにデータを出力。
      output(u, t, xmin, xmax, dx, NX, NGH, n/NOUT);
    if (n == 10000) //  今 NT = 1000000 で dt = 10 だから dt*n = 10e5 のときに出力する。
      outputt(u, t, xmin, xmax, dx, NX, NGH, n);
```

関数 outputt は関数 output に手を加えて以下のようにした。

```
void outputt(double *u, double t,
             double xmin, double xmax, double dx,
             int n, int ngh, int step) {
  char fn[32];
  sprintf(fn, "output10e5.dat");
  FILE *fp;
  fp = fopen(fn, "w");
  if (fp == NULL) {
    printf("Error: cannot open file %s\n", fn);
    exit(1);
  }

  fprintf(fp, "# t=%f, xmin=%f, xmax=%f, dx=%f, n=%d, ngh=%d, step=%d\n",
          t, xmin, xmax, dx, n, ngh, step);
```

```
  for (int i = 1; i < n+ngh; ++i) {
    double x = xmin+(i-ngh+0.5)*dx;
    fprintf(fp, "%d %g %g\n", i, x, u[i]);
  }
  fclose(fp);
  return;
}
```

$T = 0, 10^5, 10^6, 10^7$ 年における分布を重ねたグラフが図 4 である。図 1、図 2 と合わせて考えると、$100AU$ 付近の面密度はあまり変わらないが、ここから中心に向かうほど時間の経過とともに面密度が下がっていき、更に外側では時間とともに面密度が上がっていく様子が見られた。変化率は内側でも外側でも時間とともに小さくなっている。例えば太陽系とこの α 粘性モデルを比べると、太陽系ではおおよそ、中心に近いほど惑星が密集しており遠方に向かって疎になっているところが、α 粘性モデルと共通しているが、木星より内側は惑星が密にあるとはいえそれぞれの質量が小さく α 粘性モデルにそぐわない。これは物質の種類を考慮に入れたモデルではないことと、物質が集積して塊になることを考えていないことが関係していると推測する。

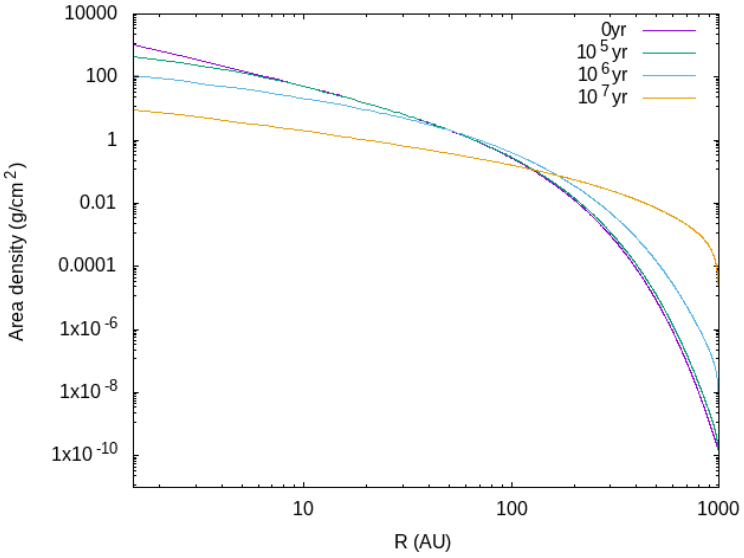

図 4　$T = 0, 10^5, 10^6, 10^7$ 年を重ねた

図 4 を見ると $5AU$ では初期状態では面密度が 3 桁の値をとっているが、2 桁の値すなわち 100 以下になるのは 10^5yr を超え、10^6yr に至る前である。太陽系の歴史は 4.6×10^9yr であるので、木星は非常に早期に形成されねばならないことになる。

課題 2

まず、SOR 法のコードであるが、Gauss-Seidel 法の関数 gauss_seidel を以下のような関数 sor に書き換えることで実現した。

```
void sor(void) {
  double omega = 2.0/(1.0+ M_PI/NX);
  for (int k = NGH; k < NZ+NGH; ++k) {
    for (int j = NGH; j < NY+NGH; ++j) {
      for (int i = NGH; i < NX+NGH; ++i) {
        u[k][j][i] += omega*((u[k][j][i-1] + u[k][j][i+1] + u[k][j-1][i] + u[k][j+1][i]
                    + u[k-1][j][i] + u[k+1][j][i] - 6.0*u[k][j][i])/6.0 - rho[k][j][i]*DX*DX/6.0);
      }
    }
  }
  return;
}
```

Jacobi 法、Gauss-Seidel 法、Red-Black-Gauss-Seidel 法、Successive Over-Relaxation 法をそれぞれ実行し、イテレーション回数を横軸、誤差の大きさを縦軸にし、横軸を対数でプロットしたものが図 5 である。ただし、GS 法と RBGS 法はイテレーション回数ごとの誤差の値がほぼ等しく、重なってしまっている。

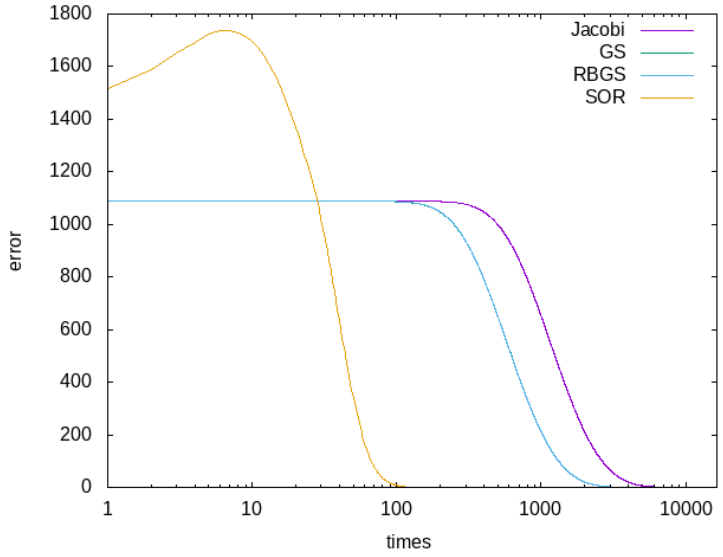

図 5　Jacobi、GS、RBGS、SOR 法の性能比較

図 5 を見ると Jacobi 法 > GS 法 ∼ RBGS 法 > SOR 法の順で収束までのイテレーション回数が小さくなっていることが確かめられた。SOR 法では、初期値は他の方法と同じであるが、その次では過緩和パラメータの影響で誤差がよりおおきくなってしまっている。

次に、SOR 法を用いたコードで、境界条件を解析解にするために、グローバル変数 $GM = 0.001/3.0$ を定義し、境界条件を与える関数を以下のように改良した。ここでの、GM の算出は以下のとおりである。まず、$4\pi G = 1$ の単位系を採用しているので、$G = \frac{1}{4\pi}$ である。さらに、半径 0.1 以下は密度 $\rho = 1$ を与えているので、$M = \frac{4}{3}\pi r^3 \rho = \frac{4}{3}\pi \times 0.1^3 \times 1$ となる。よって、$GM = \frac{1}{4\pi} \times \frac{4}{3}\pi \times 0.1^3 \times 1 = \frac{0.001}{3}$ である。

```
void boundary_inner_x(void) {
  // fixed boudary
  for (int k = NGH; k < NZ+NGH; ++k) {
    double z = ZMIN+(k-NGH+0.5)*DX;
    for (int j = NGH; j < NY+NGH; ++j) {
      double y = YMIN+(j-NGH+0.5)*DX;
      for (int i = 0; i < NGH; ++i) {
        double x = XMIN+(i-NGH+0.5)*DX;
double r = sqrt(x*x+y*y+z*z);
        u[k][j][i] = -GM/r;
      }
    }
  }
  return;
}


void boundary_outer_x(void) {
  // fixed boudary
  for (int k = NGH; k < NZ+NGH; ++k) {
    double z = ZMIN+(k-NGH+0.5)*DX;
    for (int j = NGH; j < NY+NGH; ++j) {
      double y = YMIN+(j-NGH+0.5)*DX;
      for (int i = 0; i < NGH; ++i) {
        double x = XMIN+(i-NGH+0.5)*DX;
double r = sqrt(x*x+y*y+z*z);
        u[k][j][NX+NGH+i] = -GM/r;
      }
    }
  }
  return;
}


void boundary_inner_y(void) {
  // fixed boudary
  for (int k = NGH; k < NZ+NGH; ++k) {
    double z = ZMIN+(k-NGH+0.5)*DX;
    for (int j = 0; j < NGH; ++j) {
      double y = YMIN+(j-NGH+0.5)*DX;
```

```
      for (int i = NGH; i < NX+NGH; ++i) {
        double x = XMIN+(i-NGH+0.5)*DX;
double r = sqrt(x*x+y*y+z*z);
        u[k][j][i] = -GM/r;
      }
    }
  }
  return;
}


void boundary_outer_y(void) {
  // fixed boudary
  for (int k = NGH; k < NZ+NGH; ++k) {
    double z = ZMIN+(k-NGH+0.5)*DX;
    for (int j = 0; j < NGH; ++j) {
      double y = YMIN+(j-NGH+0.5)*DX;
      for (int i = NGH; i < NX+NGH; ++i) {
        double x = XMIN+(i-NGH+0.5)*DX;
double r = sqrt(x*x+y*y+z*z);
        u[k][NY+NGH+j][i] = -GM/r;
      }
    }
  }
  return;
}


void boundary_inner_z(void) {
  // fixed boudary
  for (int k = 0; k < NGH; ++k) {
    double z = ZMIN+(k-NGH+0.5)*DX;
    for (int j = NGH; j < NY+NGH; ++j) {
      double y = YMIN+(j-NGH+0.5)*DX;
      for (int i = NGH; i < NX+NGH; ++i) {
        double x = XMIN+(i-NGH+0.5)*DX;
double r = sqrt(x*x+y*y+z*z);
        u[k][j][i] = -GM/r;
      }
    }
```

```
  }
  return;
}


void boundary_outer_z(void) {
  // fixed boudary
  for (int k = 0; k < NGH; ++k) {
    double z = ZMIN+(k-NGH+0.5)*DX;
    for (int j = NGH; j < NY+NGH; ++j) {
      double y = YMIN+(j-NGH+0.5)*DX;
      for (int i = NGH; i < NX+NGH; ++i) {
        double x = XMIN+(i-NGH+0.5)*DX;
double r = sqrt(x*x+y*y+z*z);
        u[NZ+NGH+k][j][i] = -GM/r;
      }
    }
  }
  return;
}
```

図５にこのコードの結果も加えると、図６のようになる。

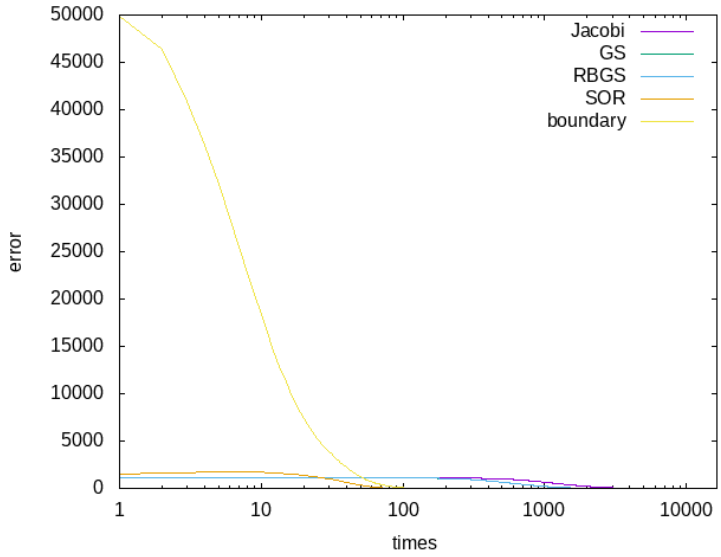

図６

初期値が非常に大きくなっているのは境界条件を０ではなくしたためであると思われる。図７は図６を図５

と同じ範囲で切り取ったものであるが、これをみると収束までのイテレーション回数は SOR 法より少し増えてしまっている。

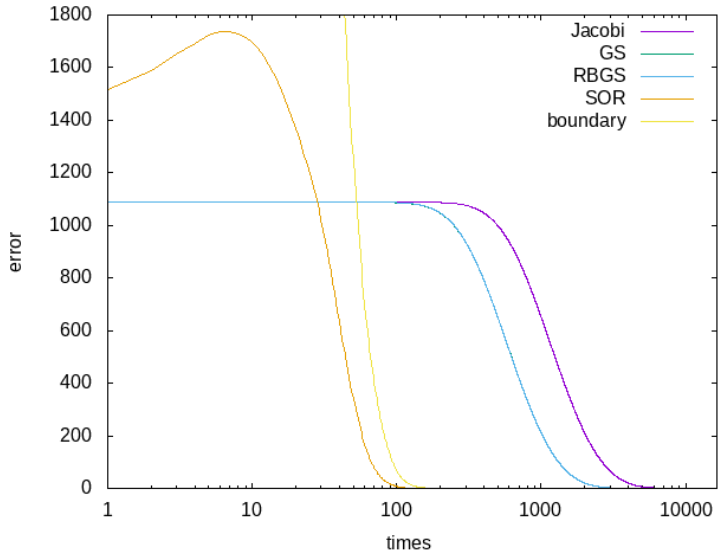

図 7