2026 年 8 月 17 日

SuperCon 2026 本選問題
さまざまな大きさの円盤を効率よく詰め込もう

1　目的

本課題では，ある大きさの箱の中に，さまざまな大きさの球をできるだけ効率よく詰め込む方法を考える．このような充填問題は，単なるパズルにとどまらず，粒状物質からなる材料の性質を理解する上で，また限られた空間を有効に利用するための最適化問題としても重要である．特に，球の大きさが異なる場合には，小さな球が大きな球のすき間に入り込むため，球の大きさの組合せや配置によって詰まり方が大きく変化する．

ここでは問題を単純化し，正方形領域内に，形や大きさの変化しない剛体円盤を，互いに重なり合うことなく敷き詰める 2 次元の問題を考える．この問題は，物理学において

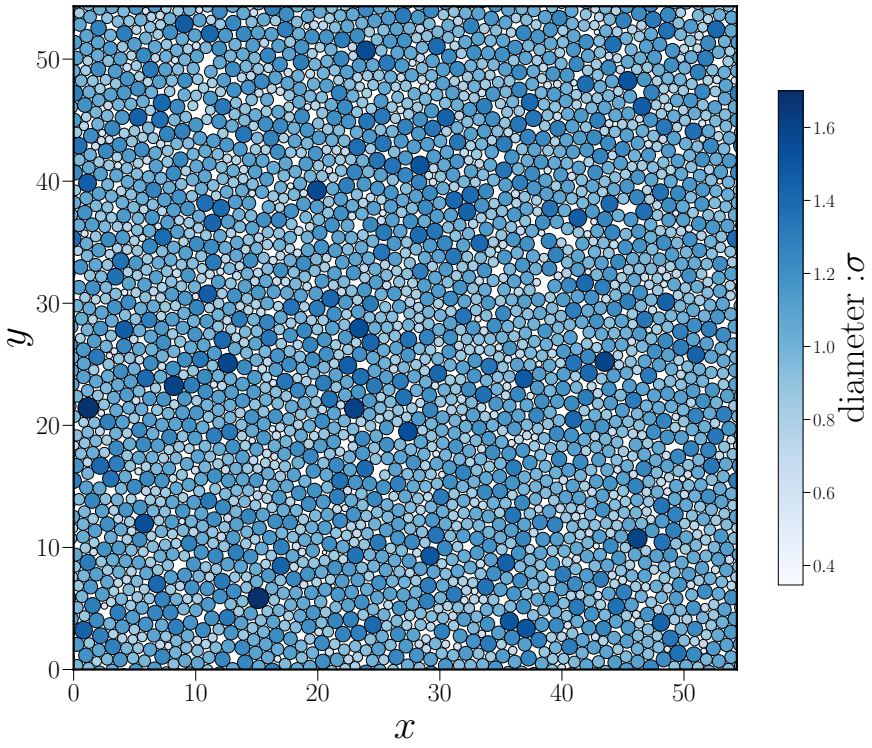


図 1　さまざまな大きさをもつ剛体円盤の敷き詰め．円盤同士は互いに重なり合わないものとする．

「ジャミング」と呼ばれる現象として広く知られている．大きさに顕著な分布をもつ 2 次元の剛体円盤系においては，円盤の面積分率

$$\varphi = \sum_{j=1}^{N} \pi \left(\frac{\sigma_j^2}{4L^2} \right)$$

が 84–86% あたりに達すると，円盤を重複なく敷き詰めることが極めて困難になることが知られている（この閾値を「ジャミング転移点」とよぶ）．ここで，N は剛体円盤の数，L は正方形領域の 1 辺の長さ，σ_j は円盤 j の直径である．また，このジャミング転移点は，円盤の配置によっても変化しうることが知られている．したがって本課題では，円盤の最適な配置を探索し，いかに高密度なジャミング配置を実現できるかを競う．

(注)：もし球の大きさが均一であれば，六方最密充填と呼ばれる構造が最密（面積分率 $\pi/2\sqrt{3} \sim 0.91$) になる．

2　剛体円盤の敷き詰め

本節では，剛体円盤の敷き詰め問題を定式化するために必要な諸条件を定義する．

2.1　剛体円盤

剛体円盤とは，形や大きさが変化しない円盤である．また，複数の円盤が互いに重なり合うことは許されないものとする（図 1 は，ジャミング転移点近傍にある剛体円盤のとある配置の例を示したものである）．

2.2　正方形領域の定義：周期境界条件

一つの角を原点とし第 1 象限内に位置する一辺の長さ L の正方形領域内に，N 個の剛体円盤を配置する．ただし，この領域には**周期境界条件**を課し，正方形の上下の境界，および左右の境界がそれぞれつながっているものとする．周期境界条件下では，境界から領域外に出た円盤は，反対側の境界から現れることに注意する（図 2）．

周期境界条件は，有限のサイズの領域を扱う際に生じる境界（表面）の影響を排除し，無限に広がる系を近似的に再現する手法として，物理学や材料科学における数値シミュレーションで広く用いられている．

ここで，円盤 i の中心座標を (x_i, y_i)，円盤 j の中心座標を (x_j, y_j) とする．周期境界

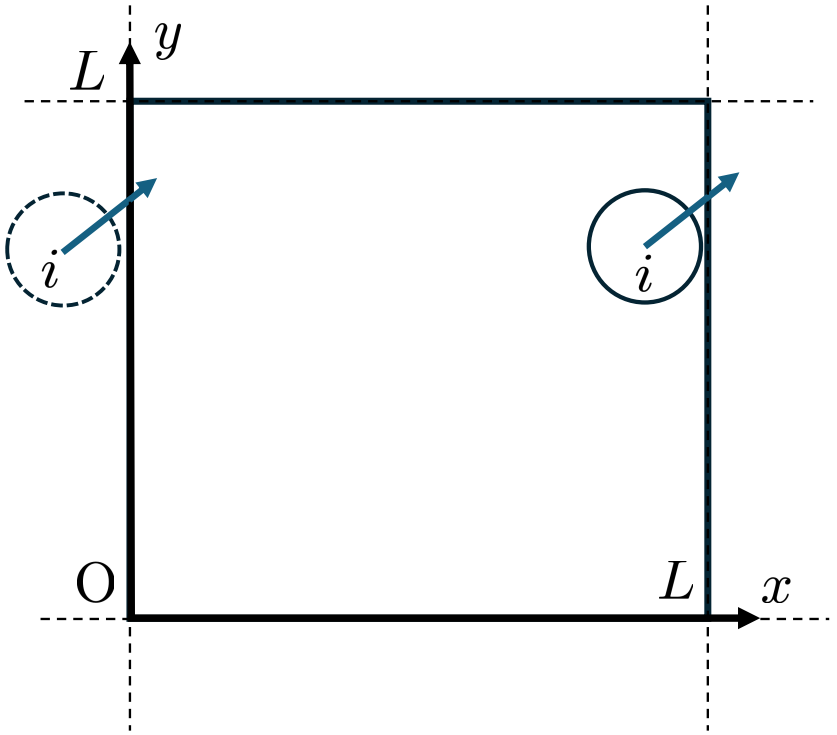


図 2　周期的境界条件．上下の境界，および左右の境界がそれぞれつながっている．

条件のもとでは，2 つの円盤の中心間最短距離は，単純なユークリッド距離

$$d_{ij} = \sqrt{\Delta x_{ij}^2 + \Delta y_{ij}^2}$$

とは限らない．ここで，$\Delta x_{ij} = x_i - x_j$，$\Delta y_{ij} = y_i - y_j$ である．これは，正方形領域の境界をまたいだ経路の方が，領域内を直接結ぶ経路よりも短くなる場合があるためである．

一辺の長さ L の正方形領域が周期的につながっていることを考えると，Δx_{ij} の大きさが $L/2$ を超える場合には，反対側の境界を経由した方が距離が短くなる．そこで，周期境界条件を課した変位 ΔX_{ij} を，次のように場合分けして定義する (図 3 は 1 番目のケースである．)．

$$\Delta X_{ij} = \begin{cases} \Delta x_{ij} - L, & \Delta x_{ij} \geqq \dfrac{L}{2} \text{ のとき（}i\text{ を固定し }j\text{ を右側のセルにシフト）} \\ \Delta x_{ij} + L, & \Delta x_{ij} < -\dfrac{L}{2} \text{ のとき（}i\text{ を固定し }j\text{ を左側のセルにシフト）} \\ \Delta x_{ij}, & -\dfrac{L}{2} \leqq \Delta x_{ij} < \dfrac{L}{2} \text{ のとき（そのまま）} \end{cases}$$

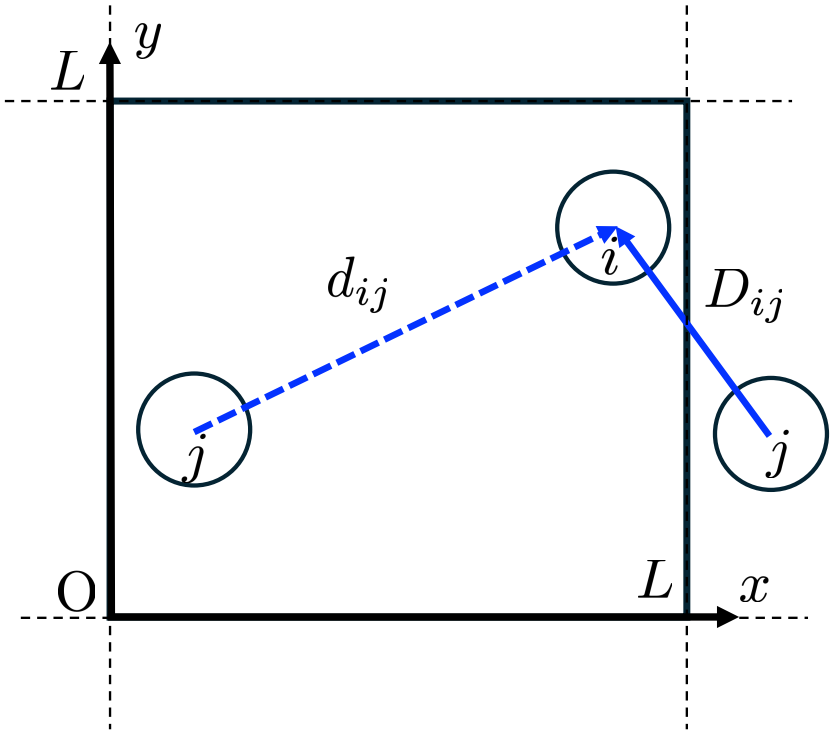


図 3　周期的境界条件下における 2 点間の最短距離の考え方．円盤 i と j の最短距離は D_{ij} となる．

y 座標についても同様に，

$$\Delta Y_{ij} = \begin{cases} \Delta y_{ij} - L, & \Delta y_{ij} \geqq \dfrac{L}{2} \\ \Delta y_{ij} + L, & \Delta y_{ij} < -\dfrac{L}{2} \\ \Delta y_{ij}, & -\dfrac{L}{2} \leqq \Delta y_{ij} < \dfrac{L}{2} \end{cases}$$

と定義される．

この場合分けは，「L の整数倍を足し引きして，差を $-L/2$ から $L/2$ の範囲に収める」という一つの操作にまとめることができる．この操作は，床関数（floor function）$\lfloor \cdot \rfloor$（実数を超えない最大の整数を与える関数）を用いると，

$$\Delta X_{ij} = \Delta x_{ij} - L \left\lfloor \frac{\Delta x_{ij}}{L} + \frac{1}{2} \right\rfloor, \qquad \Delta Y_{ij} = \Delta y_{ij} - L \left\lfloor \frac{\Delta y_{ij}}{L} + \frac{1}{2} \right\rfloor$$

と簡潔に表せる．実際，$-L/2 \leq \Delta x_{ij} < L/2$ のときは $\lfloor \Delta x_{ij}/L + 1/2 \rfloor = 0$ となり $\Delta X_{ij} = \Delta x_{ij}$（変化なし）となる．$\Delta x_{ij} \geq L/2$ のときは $\lfloor \Delta x_{ij}/L + 1/2 \rfloor = 1$ とな

り $\Delta X_{ij} = \Delta x_{ij} - L$ となる．$\Delta x_{ij} < -L/2$ のときは $\lfloor \Delta x_{ij}/L + 1/2 \rfloor = -1$ となり $\Delta X_{ij} = \Delta x_{ij} + L$ である．したがって，上記の場合分けと一致することが確認できる．

このとき，周期境界条件下における円盤 i と円盤 j の中心間最短距離 D_{ij} は，

$$D_{ij} = \sqrt{(\Delta X_{ij})^2 + (\Delta Y_{ij})^2}$$

で与えられることになる（図 3）．

以下の問題では，円盤の最適配置を探索する際，円盤間の重なりの有無を調べるために，上記の周期境界条件を考慮した上で，多数の円盤間の最短距離を効率よく計算することが重要な課題となる．

3 問題

第 2.2 節で導入した周期境界条件をもつ正方形領域内に，第 2.1 節で導入した $N = 3000$ 個の剛体円盤を配置することを目指す．各円盤の直径は，平均値がおよそ 1.0 となるように与えられ，同一のデータセット内でも円盤ごとに異なる値をとる．さらに，各円盤の初期座標 (x_i^0, y_i^0) が与えられる．この初期座標は，一辺の長さ $L_0 = 57.0$ の正方形領域に収まるように一様乱数を用いて生成したものであり，剛体円盤としての条件は満たしていないことに注意せよ．いま，この初期配置から出発し，正方形領域の大きさを変化させながら各円盤を適宜動かすことで，**最終的に**以下の条件を満たす配置を得ることを目指す．

- **非重複条件**: N 個すべての円盤が互いに重ならないこと．
- **初期配置からの変位の大きさの制限**: 一辺の長さ L の正方形領域に対して定義される周期境界条件下において測られる，各円盤の中心座標 (x_i, y_i) と初期座標 (x_i^0, y_i^0) との間の最短距離の平均値が 10 以内に収まっていること．

上記の条件のもと，正方形領域の一辺の長さ L をできるだけ小さくし，そのときの各円盤の中心座標 (x_i, y_i) も併せて求めよ．採点方法に関しては，以下のルールおよび勝利条件をよく読むこと．

4 ルール

- 3000 個の円盤の初期座標および直径を記録したデータセットを 8 組，すなわち 8 問分の入力データとして与える．入力ファイル名は，`input_1.txt` から `input_8.txt` までとする．

  各入力ファイルは 3 列 3000 行からなり，各行には円盤の初期座標 x_i^0，y_i^0，および直径 σ_i が，この順に記録されている．円盤の直径は，同一のデータセット内でも円盤ごとにばらつきをもち，さらに，そのばらつきの度合いもデータセットごとに異なることに注意せよ．

  初期座標は，一辺の長さ $L_0 = 57.0$ の正方形領域内に，一様乱数を用いて生成したものである．したがって，初期配置は剛体円盤としての非重複条件を満たしていないことに注意せよ．

  事前に，練習用データセット 8 組を生成するためのプログラム `gene_initial_conditions.cpp` を配布する．データの生成方法については，同プログラムを参照せよ．

  本番の採点には，同一の生成プログラムを用いて生成した，練習用とは異なるデータセットを使用する．
- 各問の入力データは, `sc26.h` に定義された関数 `sc26_input_initial_conditions()` を用いて読み込むこと．

  その後，各自が作成したプログラムを用いて，正方形領域の一辺の長さ L および各円盤の位置を変化させ，以下の条件を満たす円盤配置と L を求めること．
- **8 問を処理するジョブ全体の実行時間の上限を 600 秒とする．**

  採点時には，提出されたジョブスクリプト `run-sc26teamXX.sh` （`XX` はチーム番号（2 桁）であり，詳細は後述する）が富岳に投入される．その実行中に 8 問が処理されるようにしてほしい．8 問を逐次的に処理しても良いし、並列に処理しても良い．(いくつかの例はサンプルのジョブスクリプトによって示す.) 実行時間は，各出力ファイルに記録された絶対時刻に基づいて判定する．具体的には，8 問のうち最も早い開始時刻を t_{start}，最も遅い終了時刻を t_{end} とし，

  $$t_{\mathrm{end}} - t_{\mathrm{start}} < 600 \text{ s}$$

  を満たさなければならない．

  ここで，t_{start} は，8 問のうち最初に `sc26_input_initial_conditions()` が呼び

出された時刻であり，t_{end} は，8 問のうち最後に sc26_output() が呼び出された時刻である．

したがって，600 秒の制限は，各問について個別に 600 秒が与えられることを意味するものではなく，**8 問すべてを通した全体の実時間に対して適用される．**この時間内に，8 問すべての出力ファイルが生成されるようにすること．

- sc26.h を変更してはならない．採点には，主催者が公開した sc26.h を使用する．
- **出力データの形式およびファイル名は，sc26.h に定義された関数 sc26_output() の仕様に従うものとする．**

問題番号 ens に対して，sc26_output() を実行すると，出力ファイル coord_ens.txt が生成される．

例えば，ens = 1 の場合には，coord_1.txt が生成される．

同じ問題番号に対して sc26_output() を複数回呼び出した場合，呼び出すたびに新しい結果が出力ファイルの末尾に追記される（上書きはされない）．**したがって，過去の実行で生成された古い coord_ens.txt が残っていると，新しい結果がその末尾に追記されてしまう．開発段階においては，実行のたびに，古い coord_*.txt をあらかじめ削除してから計算を開始すること．**

出力データは，sc26_output() が呼び出されるたびに，以下の形式のブロックが 1 つずつ末尾に追記される．

```
L
start_time
end_time
x[0] y[0]
x[1] y[1]
...
x[SC26_Np-1] y[SC26_Np-1]
END
```

各ブロックの第 1 行には，正方形領域の一辺の長さ L が出力される．

第 2 行には，sc26_input_initial_conditions() が呼び出された時点の開始絶対時刻が出力される．

第 3 行には，sc26_output() が呼び出された時点の終了絶対時刻が出力される．

開始時刻および終了時刻は，1970 年 1 月 1 日 00 時 00 分 00 秒（UTC）を基準とする UNIX 時刻として，秒単位の倍精度浮動小数点数で記録される．

第 4 行から第 3003 行までには，各円盤の中心座標 x_i，y_i が，この順に空白で区切って出力される．**円盤の順番は，入力データにおける順番と一致させること．**
各ブロックの最終行には，そのブロックの書き込みが最後まで正常に完了したことを示す終端マーカー `END` が出力される．採点においては，全体の制限時間内（t_{start} から 600 秒以内）に完全に書き込まれた（すなわち `END` まで出力された）最後のブロックが，各問題の有効な回答として採用される．制限時間を超えて出力されたブロックや，書き込みが途中で終了した不完全なブロックは無効とみなされる．

- すべての異なる円盤対 i, j について，周期境界条件下で定義される中心間最短距離 D_{ij} が，

$$D_{ij} \geqq \frac{\sigma_i + \sigma_j}{2}$$

を満たすものとする．ここで，σ_i および σ_j は，それぞれ円盤 i および円盤 j の直径である．

- 本課題では, 倍精度浮動小数点数演算に伴う丸め誤差を考慮し, 絶対値が 1.0×10^{-10} 以下の差異は 0 とみなす.
したがって，重なりに関する条件については,

$$\frac{\sigma_i + \sigma_j}{2} - D_{ij} < 1.0 \times 10^{-10}$$

であれば, 数値誤差の範囲内として, 近似的に非重複条件を満たすものと判定する.

- 各円盤 i について，提出された中心座標 (x_i, y_i) と初期座標 (x_i^0, y_i^0) の間の変位を，提出された一辺の長さ L に対する周期境界条件のもとで評価する．
各方向の最短変位を,

$$\Delta x_i = (x_i - x_i^0) - L \left\lfloor \frac{x_i - x_i^0}{L} + \frac{1}{2} \right\rfloor,$$

$$\Delta y_i = (y_i - y_i^0) - L \left\lfloor \frac{y_i - y_i^0}{L} + \frac{1}{2} \right\rfloor$$

と定義する．このとき，円盤の移動距離の平均値が,

$$\frac{1}{N} \sum_{i=1}^{N} \sqrt{(\Delta x_i)^2 + (\Delta y_i)^2} \leqq 10$$

を満たすものとする．

- 各円盤の直径には，対応する入力データに記録された値を用いること．円盤の順番を入れ替えたり，直径を変更したりしてはならない．すべての数値データは倍精度浮動小数点数として扱い，解答の出力には `sc26.h` に定義された `sc26_output()` を使用すること．
- 提出プログラムにおいて許される入出力はすべて `sc26.h` に定義された入出力用の関数を用いたもののみである。これらは指定された名前のファイルを扱う。**プログラム開発段階においてはこれ以外に、例えば標準入出力を用いた入出力を自由に行なって良いが、提出するプログラムにおいては該当部分を適切に削除ないしコメントアウトすること。**なお `MPI` の場合、標準入出力の扱いは慎重に行う必要がある。例えば `mpiexec -std-proc xxx.txt` とすると `xxx.txt` という名前のファイルに標準出力および標準エラー出力がまとめて書き出される。

5 勝利条件

- 各問題について，提出された正方形領域の一辺の長さ L および各円盤の中心座標をもとに，すべての円盤が互いに重なっていないこと，初期配置からの変位の大きさ，計算時間を審査委員会が用意した検証用プログラム（`check_results.cpp`）によって確認する．
- 上記の条件を満たした解答について，問題ごとに L の小さい順に順位を決定し，以下のように得点を与える．
  - 1 位：5 点
  - 2 位：4 点
  - 3 位：3 点
  - 4 位：2 点
  - 5 位：1 点
  - 6 位以下：0 点
- 8 問の得点を合計し，その合計得点に基づいて総合順位を決定する．総合得点が同点の場合には，以下の順に順位を決定する．
  1. 正解数（条件を満たす問題数）が多いチームを上位とする．
  2. 正解数が同じ場合には，L の値の合計が小さいチームを上位とする．
  3. L の値の合計も同じ場合には，全計算時間が短いチームを上位とする．

6　ヘッダファイル

コンテスト専用ヘッダファイル sc26.h には，本課題で使用する定数，タイマー関数，ならびにデータの入出力関数が定義されている．sc26.h は，リポジトリ「SC2026Final」内の **header** ディレクトリからダウンロードできる．

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cfloat>
#include <chrono>

#define SC26_Np 3000 // 円盤の数.

double sc26_elapsed_seconds(void)
{
    static std::chrono::high_resolution_clock::time_point
        SC26_start_time =
         std::chrono::high_resolution_clock::now();

    return std::chrono::duration<double>(
         std::chrono::high_resolution_clock::now()
         - SC26_start_time
    ).count();
}


double sc26_absolute_seconds(void)
{
    return std::chrono::duration<double>(
         std::chrono::system_clock::now().time_since_epoch()
    ).count();
}

double sc26_problem_start_time(int ens, double new_time = -1.0)
```

```
{
    static double start_time[9] = {};
    static bool recorded[9] = {};

    if(ens < 1 || ens > 8)
        return -1.0;

    if(new_time >= 0.0 && !recorded[ens]){
        start_time[ens] = new_time;
        recorded[ens] = true;
    }

    return start_time[ens];
}

void sc26_input_initial_conditions(
    double *x,
    double *y,
    double *diameter,
    int *ens,
    char *argv[]
)
{
    *ens = atoi(argv[1]);

    sc26_elapsed_seconds();

    sc26_problem_start_time(
        *ens,
        sc26_absolute_seconds()
    );

    int rank = 0;

#ifdef MPI_VERSION
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
#endif

    if(rank == 0){
```

```
        char filename[128];
        std::ifstream file;

        sprintf(filename, "input_%d.txt", *ens);
        file.open(filename);

        for(int i = 0; i < SC26_Np; i++)
            file >> x[i] >> y[i] >> diameter[i];

        file.close();
    }

#ifdef MPI_VERSION
    MPI_Bcast(x, SC26_Np, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(y, SC26_Np, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(diameter, SC26_Np, MPI_DOUBLE, 0, MPI_COMM_WORLD);
#endif
}

void sc26_output(
    double *x,
    double *y,
    double L,
    int ens
)
{
    double start_time = sc26_problem_start_time(ens);

    double end_time = sc26_absolute_seconds();

    char filename[128];
    std::ofstream file;

    sprintf(filename, "coord_%d.txt", ens);

    file.open(filename, std::ios::app);

    file << std::setprecision(16);

    file << L << std::endl;
```

```
    file << start_time << std::endl;
    file << end_time << std::endl;

    for(int i = 0; i < SC26_Np; i++)
        file << x[i] << " " << y[i] << std::endl;
    file << "END" << std::endl;

    file.close();
}
```

上記のヘッダファイル sc26.h は，例えば以下のようにインクルードして使用する．

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <cfloat>
#include <omp.h>
#include "mpi.h"（mpi を用いる場合）
#include "sc26.h"
```

sc26.h は，他のヘッダファイルよりも後にインクルードすること．なお，**配布されたヘッダファイルを改変することは一切認めない．**

以下では，ヘッダファイルに定義されている各関数について説明する．

- `void sc26_input_initial_conditions(double *x, double *y, double *diameter, int *ens, char *argv[])`
  問題番号 ens に対応する入力ファイルから，円盤の初期座標および直径を読み込む関数である．
  問題番号 ens は，コマンドライン引数 argv[1] から取得される．したがって，実行時には問題番号を第 1 引数として指定すること．例えば，問題 1（ens = 1）を実行する場合には，./a.out 1 のように実行する．
  入力ファイル名は input_ens.txt であり，例えば ens = 1 の場合には，input_1.txt が読み込まれる．

読み込まれた各円盤の x 座標，y 座標，直径は，それぞれ配列 x，y，diameter に格納される．また，本関数が呼び出された時点が，その問題の開始時刻として記録される．

sc26_input_initial_conditions() **は，プログラムの冒頭で，各問題につき一度だけ呼び出すこと．**関数 sc26_input_initial_conditions() は，変数の宣言およびメモリの確保を除き，プログラム中で最初に実行する関数とすること．ただし，MPI を使用する場合は，MPI_Init を先に実行し，その直後に sc26_input_initial_conditions() を実行すること．これら以外の処理を sc26_input_initial_conditions() より前に行ってはならない．

- void sc26_output(double *x, double *y, double L, int ens)
  解答データを出力する関数であり，解答の提出には必ず本関数を使用すること．引数には，それぞれ円盤の x 座標，y 座標，正方形領域の一辺の長さ L，および問題番号 ens を与える．
  本関数を実行すると，問題番号 ens に対応する出力ファイル coord_ens.txt が作成される．例えば ens = 1 の場合には，coord_1.txt が出力される．
  開始時刻および終了時刻は，UNIX 時刻を秒単位で表した double 型の値として出力される．
  sc26_output() は，プログラム中で複数回呼び出してもよい．SC26_output() 関数以外の方法で出力してはならない．
- double double sc26_elapsed_seconds(void)
  プログラムの実行中に，基準時刻からの経過時間を秒単位で取得する関数である．初回の呼び出し時に，その時点の時刻が基準時刻として記録される．2 回目以降の呼び出しでは，基準時刻から呼び出し時点までの経過時間が，double 型の値として返される．
  本関数は sc26_input_initial_conditions() の内部でも呼び出されるため，通常は入力関数を実行した時点からの経過時間を取得できる．**参加者は，計算時間の確認や終了条件の判定などを目的として，プログラム内の任意の箇所で本関数を呼び出してよい．**
- double sc26_absolute_seconds(void)
  呼び出し時点の絶対時刻を，UNIX 時刻として秒単位で返す関数である．UNIX 時刻とは，1970 年 1 月 1 日 00 時 00 分 00 秒（UTC）からの経過秒数である．
  本関数は，各問題の開始時刻および終了時刻を記録するために，
  sc26_input_initial_conditions() および sc26_output() の内部で使用され

る．**通常，参加者が本関数を直接呼び出す必要はない．**

- `double sc26_problem_start_time(int ens, double new_time = -1.0)`
  問題番号 `ens` に対応する開始時刻を保存または取得する関数である．
  第 2 引数 `new_time` に 0 以上の値が与えられた場合，その値が問題 `ens` の開始時刻として保存される．ただし，開始時刻が保存されるのは，各問題について最初の 1 回だけであり，その後は変更されない．
  第 2 引数 `new_time` を省略した場合には，すでに保存されている問題 `ens` の開始時刻が返される．
  本関数は，`sc26_input_initial_conditions()` および `sc26_output()` の内部で使用される．**通常，参加者が本関数を直接呼び出す必要はない．**

7 サンプルプログラム

サンプルプログラムおよびそのコンパイル用・実行用スクリプトは，リポジトリ「SC2026Final」からダウンロードできる．

- **sample-single**:
  - `sc26-sample-single.cpp`：シングルコアサンプルプログラム．コマンドライン引数を用いて問題番号を入力する．
    円盤間の重なりの分だけ，仮想的な位置エネルギーが増加する仕様になっており，これを解消する方向に円盤は力を受け運動する．これは，最急降下法と呼ばれる手法の一種であり，重なりが解消すると力を受けなくなり円盤は静止する．その後圧縮し同様の過程を繰り返す．これらは極めて原始的な実装であり，工夫の余地がある．
  - `compile-sc26-sample-single.sh`：コンパイル用スクリプト．
  - `run-sc26-sample-single.sh`：実行用スクリプトの例．各自，プログラムにあわせ適宜変更せよ．
- **sample-omp**:
  - `sc26-sample-omp.cpp`：`sc26-sample-single.cpp` に対して OpenMP による並列化を施したサンプルプログラム．コマンドライン引数を用いて問題番号を入力する．
  - `compile-sc26-sample-omp.sh`：コンパイル用スクリプト．
  - `run-sc26-sample-omp.sh`：実行用スクリプトの例．各自，プログラムにあ

わせ適宜変更せよ．

- **sample-mpi**:
  - `sc26-sample-mpi.cpp`：`sc26-sample-omp.cpp` に対してさらに MPI による並列化を施したサンプルプログラム．異なる探索を MPI 並列によって行なっている．簡単な実装であり、工夫の余地がある．引数を用いて問題番号を入力する．
  - `compile-sc26-sample-mpi.sh`：コンパイル用スクリプト．
  - `run-sc26-sample-mpi.sh`：実行用スクリプトの例．各自，プログラムにあわせ適宜変更せよ．
- **others**:
  - `gene_initial_conditions.cpp`：初期座標と直径分布を生成するプログラム．実行するたびに（実行時刻に応じて）異なる条件が生成されるため，練習用として用いよ．
  - `check_results.cpp`：採点の際に用いる判定プログラム．各自，用いることで，自身のプログラムが本課題の条件を満たしているか確かめよ．
  - `snapshot.py`：`input_ens.txt` と `coord_ens.txt` を入力データとして，円盤配置を可視化するプログラム．`coord_ens.txt` は `sc26_output()` が呼び出されるたびに追記される形式であるため，そのうち最後に完全に書き込まれたブロック（末尾に終端マーカー `END` を伴うもの）を読み取り，静止画として出力する．Python の `matplotlib` を用いている．必要に応じて適宜用いよ．
  - `movie.py`： `input_ens.txt` と `coord_ens.txt` を入力データとして，円盤配置を可視化し，それらを時系列に繋げ動画を生成するプログラム．`coord_ens.txt` 中に追記された，完全に書き込まれた（終端マーカー `END` を伴う）ブロックをすべて時系列順に読み取り，各ブロックを 1 フレームとしてアニメーション化する．gif および MP4 形式で保存するものであり，MP4 形式の動画生成には ffmpeg が必要であるため，各自の環境に応じて適宜インストールされたい．

8 スーパーコンピュータ

本選では，理化学研究所のスーパーコンピュータ富岳を用いる．具体的な使い方は，Web 上で公開されているマニュアルを参考にすること．

9　コンパイル方法

コンパイルオプションは以下のとおりとする．OpenMP/MPI をどのように使うかに依存しない．このコンパイルによって出来る実行ファイル `a.out` をジョブスクリプトの `run-sc26teamXX.sh` の中で用いるものとする．

```
mpiFCCpx -Nclang -Ofast -fopenmp -Rpass=.* -std=c++17 -SSL2BLAMP sc26teamXX.cpp
```

10　提出物

各チームは，以下の 3 点を提出すること．ここで，`XX` はチーム番号（2 桁: 01 -20）を表す．

- プログラムのソースコード 1 点：`sc26teamXX.cpp`
- ジョブスクリプト 1 点：`run-sc26teamXX.sh`
- レポート (フォーマット自由): どのようなプログラミングをしたのか、特に工夫した点などを解説してほしい．また運営側からアンケートとして質問することがあるのでその回答も含めてほしい．その詳細については Discord でのアナウンスに注意してほしい．

ソースコードは，必ず 1 つのファイルにまとめること．複数のソースファイルへの分割は認めない．すなわち，ヘッダファイル `sc26.h` を `#include` する単一のソースファイルとすること．使用できる言語は C または C++ とし，ファイルの拡張子は `.cpp` とせよ．単一ソースファイルを複数提出することも認めない．各チームが提出できるプログラムは 1 つのみである．ジョブスクリプトは，`bash` コマンドで実行できる形式とすること．`Makefile` の使用は認めない．ジョブスクリプトの記述方法については，第 7 節に示すサンプルを参考にすること．**上記 3 点のファイルを，締切までに Discord のチャンネル「個別相談-XX-チーム名」（他のチームからは見えない）から提出すること。**

提出前には，提出物の内容，ファイル名，および提出先を十分に確認すること．特に，以下のいずれかに該当する場合には，審査不能と判断され，当該チームが失格となることがある．

- 提出物が上記の要件を満たしていない場合

- 複数のソースファイルが提出されている場合
- 定められたコンパイル方法でソースコードを正常にコンパイルできない場合
- 提出された実行用スクリプトを用いて，プログラムを正常に実行できない場合
- 必要なファイルが不足している場合，またはファイル名が指定された形式と異なる場合

したがって，**提出前に，すべての提出物が指定された環境で正常にコンパイルおよび実行できることを必ず確認すること．**