MMCSS tutorial(jp)

Takanori Saiki

最終更新日：2024 年 12 月 14 日

1　はじめに

MMCSS(Multicellular Molecular Communication System Simulator) は多細胞分子通信系のシミュレーションを実施する汎用シミュレータです。シミュレーションにおいて基盤となるプログラムは用意されているので、ユーザはパラメータを変更することで、あるいは独自に拡張をおこなうことで自身の考えたシミュレーションモデルを簡単に実装することができます。

2　導入

ここでは、MMCSS を導入するための手順を説明します。使用環境は Mac OS か Linux を想定していますが、いくつかのツールを揃えることで Windows でも実行可能になります。

2-1　ダウンロード

GitHub の MMCSS のページ (https://github.com/saikiRA1011/CellNetworkShapeSimulation) からソフトウェアをダウンロードします。ダウンロード先はどこでも良いですが、シミュレーションの実行後に画像ファイルや動画ファイルが作成されるため容量の余裕があるドライブに配置しましょう。

2-2　セッティング

MMCSS ではいくつかの外部ツールをシミュレーションの作成、実行に用いています。以下はそのツールとバージョンの一例です。

- gcc 11.3.0 (C++11 が使用できるバージョン)
- Python 3.9.12
- pip 22.1.2
- make 3.81
- yaml-cpp 0.8

Makefile では gcc-14 を使用していますが、インストールされている gcc のバージョンに合わせて変更してください。また OpenMP を使用しているため、他のコンパイラは使用にあたって注意が必要です。

Python で用いるライブラリは以下のコマンドから一括でインストールできます。

```
1  pip install -r requirements.txt
```

3　実行

ここではシミュレーションの実行方法を説明します。MMCSS では make を用いてビルド・シミュレーション実行・動画ファイルの生成を実行できるようにしています。

シミュレータをビルドする際には単に、`make` を実行してください。実行ファイルとして SimMain が生成されます。また、シミュレーションを実行する際には `make run` を実行してください。シミュレーション結果は result ディレクトリに、ステップごとにテキストファイルとして

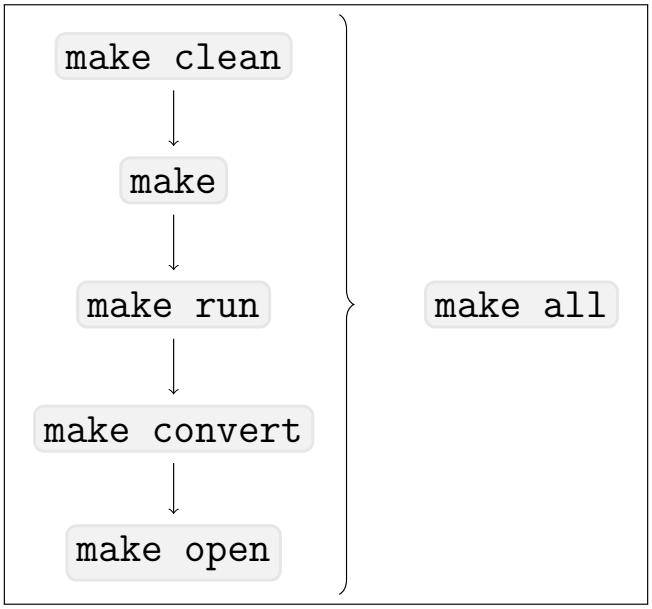

図 1　make コマンドによる一連の流れ

出力されます。次に、シミュレーション結果から動画ファイルを生成する際には`make convert`を実行してください。最終生成物の out.mp4 は video ディレクトリに、中間生成物の各ステップごとの画像は image ディレクトリに出力されます。最後に、生成した動画を確認したい時は`make open`を実行してください。

また、これらの一連の処理は`make all`でまとめて実行可能です。図 1に一連の流れを示します。

4　シミュレーションの自作

ユーザは自身でプログラムを書くことで (あるいはパラメータを変更することで)、シミュレーションを自作することができます。また、ファイルの内容をデフォルトに戻したい場合は、5-2で解説している`make reset`を実行することで初期化できます。ここでは、ユーザが編集するファイルごとにシミュレーションの自作に必要な知識を説明します。

4-1　config.yml, SimulationSettings.hpp(.cpp)

config.yml と SimulationSettings.hpp(.cpp) にはシミュレーションに必要となるパラメータを定義します。例えば、シミュレーションのステップ数や、Cell(バイオナノマシン) の個数、シミュレーションの step あたりの単位時間です。それぞれのパラメータの説明は当該ファイルにコメントとして記述されているので、それらを参考にしてください。

また、必要であればパラメータを増やすことも可能です。自身で必要だと思ったシミュレーションのパラメータを SimulationSettings.hpp に追加し、SimulationSettings.cpp に yaml ファイルからの読み取り用コードを try ブロック内に追記してください。またこれらパラメータは static 変数として定義されるので、必ず SimulationSettings.cpp 末尾で初期化してください。

パラメータを設定したい場合は、config.yaml に key と値を記述してください。また、自身で定義したパラメータがある場合は、このファイルにも追記することを忘れないでください。

デフォルトで用意されているパラメータを削除することはできません。

4-2 UserSimulation.hpp(.cpp)

UserSimulation.hpp(.cpp) にはシミュレーションのモデルや初期条件 (Cell の分布など) を定義することができます。現在定義可能な関数は、Cell の初期化 (initCells)、ステップ実行前処理 (stepPreprocess)、細胞間に働く力学モデル (calcCellCellForce)、ステップ実行後処理 (stepEndProcess) に加え、細胞間相互作用の計算 (calcCellCellForce) のみです。

シミュレーションで用いる細胞の種類や分子の種類などもここで定義します。

このファイルも SimulationSettings.hpp と同様、ユーザが独自に変数・関数を定義することができるます。実装したいモデルに合わせて関数を追加・変更してください。

4-3 UserCell.hpp(.cpp)

UserCell.hpp(.cpp) には細胞の挙動を定義することができます。

例えば、細胞の死滅タイミングや、分裂タイミング、代謝によるエネルギーや保有分子濃度の変化等を実装することができます。

細胞の増減を実装するにあたりユーザが最も苦労することは現存する細胞のリストをどのように管理するかです。MMCSS ではこの面倒な管理を Cell.hpp(.cpp) 内で実装しているため、ユーザは細胞がいつ・どのような状況で増減するかのみを実装することでシチュエーションを再現できます。

4-4 UserMoleculeSpace.hpp(.cpp)

UserMoleculeSpace.hpp(.cpp) にはシミュレーションで扱いたい分子の系を定義することができます。

親クラスである MoleculeSpace にいくつかの境界条件 (ノイマン条件や、ディリクレ条件、PBC) が実装されているため、これらの境界条件を用いる際はユーザが処理を追加する必要はありません。

また MoleculeSpace は Cell と相互的に作用させることができるように、お互いに参照を持つように実装してあります。この機能は UserMoleculeSpace と UserCell にも引き継がれているため、細胞と分子の複雑な相互作用をシミュレーションする際でも、ユーザは面倒なクラス間の参照関係を実装する必要はありません。

5　付録 A

5-1　シミュレーションの結果のファイル

シミュレーションを実行すると result ディレクトリに cells_step 番号 のファイル名で複数のテキストファイルが出力されます。これがシミュレーションの結果を出力したものです。

テキストファイルの中身はタブ文字区切りで、以下の形式でシミュレーション結果が記されています。

```
1 ヘッダ行
2
3   ID typeID X Y Z Vx Vy Vz R N_contact Contact_ID_1 Contact_ID_2 ...
```

それぞれのデータの説明は以下の通りです。

- ID：Cell の識別子。唯一無二のものが割り当てられる
- typeID：Cell の種類の識別子。種類ごとに異なる動作を定義できる
- X：Cell の X 座標の位置
- Y：Cell の Y 座標の位置
- Z：Cell の Z 座標の位置
- Vx：Cell の X 方向の速度
- Vy：Cell の Y 方向の速度
- Vz：Cell の Z 方向の速度
- R：Cell の半径
- N_contact：接触している Cell の数
- Contact_ID：接触している Cell の ID

5-2　make コマンド

MMCSS ではシミュレーションの実行から結果の確認に関わる make コマンド以外にも、いくつかのコマンドを用意しています。以下はそれらの解説です。

- `make`：プログラムをビルドする。
- `make run`：シミュレーションを実行する。
- `make convert`：シミュレーション結果を動画ファイルに変換する。
- `make open`：シミュレーション結果の動画ファイルを確認する。
- `make clean`：オブジェクトファイルをすべて削除する。
- `make all`：オブジェクトファイルをすべて削除してから、シミュレーションのビルド、実行、動画の生成、確認を一通りおこなう。
- `make reset`：シミュレーションのユーザ設定ファイルをデフォルトに戻す。

- `make data-cleanup` : シミュレーションの生成物 (テキスト、画像、動画) をすべて削除する。
- `make data-archive` : 現在保存されてあるシミュレーション結果を zip アーカイブする。
- `make archive-restore date=YYYYMMDD_HHmm` : アーカイブファイルの日時を指定することで、アーカイブの内容を復元する。
- `make archive-cleanup` : すべてのアーカイブファイルを削除する。
- `make help` : Makefile に登録されているコマンドの使用方法と説明を表示する。

6 付録 B : サンプルコード

6-0.1 回転モデル

Listing 1 calcCellCellForce()

```
1  auto aroundCells = cellList.aroundCellList(c);
2  Vec3 force = Vec3::zero();
3  const Vec3 center = Vec3(0, 0, 0);
4  const Vec3 diff_from_center = c->getPosition() - center;
5  Vec3 force_cont = Vec3::zero();
6
7  constexpr double COEFFICIENT = 1.0;
8  constexpr double REPUlSION_C = 0.20;
9  constexpr double REPULSION_LEN = 15;
10 constexpr double BONDING_LEN = 5;
11
12 force += -diff_from_center.normalize().timesScalar(COEFFICIENT);
13
14 for (auto i : aroundCells) {
15     auto cell = cells[i];
16
17     if (c->id == cell->id)
18         continue;
19
20     const Vec3 diff = c->getPosition() - cell->getPosition();
21     const double dist = diff.length();
22
23     if (dist < REPULSION_LEN) {
24         force += diff.timesScalar(1.0 / dist).timesScalar(REPUlSION_C).
               timesScalar((REPULSION_LEN - dist) / REPULSION_LEN);
25     }
26
27     const Vec3 v = cell->getVelocity();
28     if (dist < BONDING_LEN) {
```

```
29          force_cont += v;
30      }
31  }
32
33  force += force_cont.normalize().timesScalar(0.2);
34  force = force.timesScalar(SimulationSettings::DELTA_TIME);
35
36  return force;
```

6-0.2 成長モデル

Listing 2　class UserCell

```
1  // add member variables
2  private:
3  std::vector<std::vector<bool>> bondMatrix;
4    const double dMax = 150.0;
5    const double dMin = 80.0;
6    const double dCont = 20.0;
7    const double lambda = 20.0;
```

Listing 3　UserSimulation()

```
1  UserSimulation::UserSimulation(/* args */)
2  {
3    bondMatrix.resize(SimulationSettings::CELL_NUM, std::vector<bool>(
         SimulationSettings::CELL_NUM, false));
4  }
```

Listing 4　initCells()

```
1  void UserSimulation::initCells() noexcept
2  {
3    std::uniform_real_distribution<double> rand_r(0, 150);
4    std::uniform_real_distribution<double> rand_theta(0, 2 * M_PI);
5    for (int i = 0; i < SimulationSettings::CELL_NUM; i++) {
6        double r = rand_r(rand_gen);
7        double theta = rand_theta(rand_gen);
8
9        double x = r * std::cos(theta);
10       double y = r * std::sin(theta);
11
12       UserCell c(CellType::WORKER, x, y, 10);
13       cells.push_back(std::make_shared<UserCell>(c));
```

```
14    }
15  }
```

Listing 5 stepPreprocess()

```
1   void UserSimulation::stepPreprocess() noexcept
2   {
3       int32_t preCellCount = cells.size();
4
5       for (int i = 0; i < preCellCount; i++) {
6           cells[i]->initForce();
7       }
8
9       for (int i = 0; i < preCellCount; i++) {
10          cells[i]->clearAdhereCells();
11          for (int j = 0; j < preCellCount; j++) {
12              if (i == j) {
13                  continue;
14              }
15
16              const Vec3 diff = cells[i]->getPosition() - cells[j]->
                    getPosition();
17              const double dist = diff.length();
18              if (dist <= dCont) {
19                  bondMatrix[i][j] = true;
20              }
21
22              if (bondMatrix[i][j] && dist >= dMax) {
23                  bondMatrix[i][j] = false;
24              }
25
26              if (bondMatrix[i][j]) {
27                  cells[i]->adhere(*cells[j]);
28              }
29          }
30      }
31  }
```

Listing 6 calcCellCellForce()

```
1   Vec3 UserSimulation::calcCellCellForce(std::shared_ptr<UserCell> c) const
        noexcept
2   {
3       Vec3 force = Vec3::zero();
```

```
4
5       for (auto cell : cells) {
6           if (c->id == cell->id)
7               continue;
8
9           const Vec3 diff = c->getPosition() - cell->getPosition();
10          const double dist = diff.length();
11
12          if (bondMatrix[c->id][cell->id]) {
13              if (c->adhereCells.size() <= 3) {
14                  force += diff.normalize().timesScalar((dMax - dist) / dMax).
                        timesScalar(2.0);
15              } else {
16                  force += -diff.normalize().timesScalar(std::max(0.0, (dist
                        - dMin) / (dMax - dMin))).timesScalar(2.0);
17              }
18          }
19
20          force += -diff.normalize().timesScalar(std::exp(-dist / lambda)).
                timesScalar(0.05);
21
22          if (dist < dCont) {
23              force += diff.normalize().timesScalar((dCont - dist) / dCont).
                    timesScalar(10.0);
24          }
25      }
26
27      force = force.timesScalar(SimulationSettings::DELTA_TIME);
28
29      return force;
30  }
```