

MMCSS tutorial(jp)

Takanori Saiki

最終更新日：2025 年 2 月 2 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する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

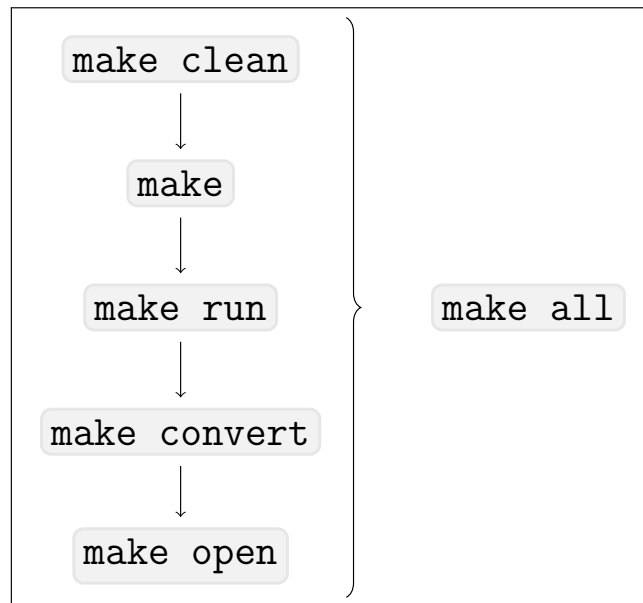


図1 make コマンドによる一連の流れ

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` でまとめて実行できます。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できる。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な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(細胞)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せます。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 `yml` 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の削除はできません。

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・関数を定義できます。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勞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ミュレ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、周期境界条件) 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、お互いに参照をも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ファイルです。

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。唯一無二の値が割り当てられる
- 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、確認を一通りする。
- `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和説明を表示する。

6 付録 B : サンプルコード

6-0.1 クラスタ形成モデル

Listing 1 `calcCellCellForce()`

```

1 Vec3 UserSimulation::calcCellCellForce(std::shared_ptr<UserCell> c) const
    noexcept
2 {
3     auto aroundCells = cellList.aroundCellList(c);
4     Vec3 force = Vec3::zero();
5
6     for (auto i : aroundCells) {
7         if (cells[i]->getCellType() == CellType::WORKER) {
8             force += Simulation::calcRemoteForce(c, cells[i]);
9         }
10    }
11    force = force.normalize();
12
13    for (auto i : aroundCells) {
14        if (cells[i]->getCellType() != CellType::NONE) {
15            force += Simulation::calcVolumeExclusion(c, cells[i]);
16        }
17    }
18
19    return force.timesScalar(SimulationSettings::DELTA_TIME);
20 }
```

6-1 ネットワーク形成モデル

Listing 2 `UserSimulation.hpp`

```

1 // add member variables
2 std::vector<std::vector<bool>> bondMatrix;
3 const double dMax = 100.0;
4 const double dMin = 40.0;
```

```

5  const double dEx = 20.0;
6  const double lambda = 20.0;

```

Listing 3 UserSimulation()

```

1  UserSimulation::UserSimulation(/* args */)
2  {
3      bondMatrix.resize(SimulationSettings::CELL_NUM, std::vector<bool>(
4          SimulationSettings::CELL_NUM, false));
5  }

```

Listing 4 stepEndProcess()

```

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

```

Listing 5 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         force += -diff.normalize().timesScalar(std::exp(-dist / lambda)).
            timesScalar(1.0);
13
14         if (bondMatrix[c->id][cell->id]) {
15             force += -diff.normalize().timesScalar(std::max((dist - dMin) /
                (dMax - dMin), 0.0)).timesScalar(30.0);
16         }
17
18         force += diff.normalize().timesScalar(std::max((dEx - dist) / dEx,
                0.0)).timesScalar(30.0);
19     }
20
21     force = force.timesScalar(SimulationSettings::DELTA_TIME);
22
23     return force;
24 }

```

6-1.1 細胞塊成長モデル

Listing 6 UserSimulation.hpp

```

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 7 UserSimulation()

```

1  UserSimulation::UserSimulation(/* args */)
2  {

```



```

3     bondMatrix.resize(SimulationSettings::CELL_NUM, std::vector<bool>(
        SimulationSettings::CELL_NUM, false));
4 }

```

Listing 8 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 9 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 10 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).
15                     timesScalar(2.0);
16             } else {
17                 force += -diff.normalize().timesScalar(std::max(0.0, (dist
18                     - dMin) / (dMax - dMin))).timesScalar(2.0);
19             }
20         }
21
22         force += -diff.normalize().timesScalar(std::exp(-dist / lambda)).
23             timesScalar(0.05);
24
25         if (dist < dCont) {
26             force += diff.normalize().timesScalar((dCont - dist) / dCont).
27                 timesScalar(10.0);
28         }
29     }
30 }

```

```

24         }
25     }
26
27     force = force.timesScalar(SimulationSettings::DELTA_TIME);
28
29     return force;
30 }

```

6-1.2 細胞塊回転モデル

Listing 11 calcCellCellForce()

```

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

```

```

30         if (dist < BONDING_LEN) {
31             force_cont += v;
32         }
33     }
34
35     force += force_cont.normalize().timesScalar(0.2);
36     force = force.timesScalar(SimulationSettings::DELTA_TIME);
37
38     return force;
39 }

```

6-2 信号分子拡散モデル

Listing 12 stepEndProcess()

```

1 void UserSimulation::stepEndProcess() noexcept
2 {
3     for (int32_t i = 0; i < (int32_t)cells.size(); i++) {
4         if (cells[i]->getCellType() == CellType::DEAD || cells[i]->
5             getCellType() == CellType::NONE) {
6             continue;
7         }
8
9         Vec3 pos = cells[i]->getPosition();
10        cells[i]->updateState(moleculeSpaces[0]->getMoleculeNum(pos));
11    }
12 }

```

Listing 13 calcCellCellForce()

```

1 Vec3 UserSimulation::calcCellCellForce(std::shared_ptr<UserCell> c) const
2     noexcept
3 {
4     return Vec3::zero();
5 }

```

class UserCell のメンバ変数・メンバ関数に以下を追加する。

Listing 14 class UserCell(UserCell.hpp)

```

1     private:
2     double cAMP = 0;
3     double activeReceptor = 0.0;
4     double diffCamp = 0;
5     double diffActiveReceptor = 0;

```

```

6     static constexpr double C = 10;
7     static constexpr double K1 = 0.036;
8     static constexpr double K2 = 0.666;
9     static constexpr double L1 = 10;
10    static constexpr double L2 = 0.005;
11    static constexpr double Q = 4000;
12    static constexpr double SIGMA = 6.0;
13    static constexpr double Ki = 1.7;
14    static constexpr double Ke = 5.4;
15    static constexpr double Kt = 0.9;
16    static constexpr double ALPHA = 1;
17    static constexpr double H = 5;
18    static constexpr double THETA = 0.01;
19    static constexpr double LAMBDA = 0.01;
20    static constexpr double EPSILON = 1;
21
22    public:
23    double emitMolecule(int32_t moleculeId) noexcept override;
24    double calcSynthesis(double extracellularCAMP) const noexcept;
25    double f1(double cAMP) const noexcept;
26    double f2(double cAMP) const noexcept;
27    double calcInactiveReceptor(double extracellularCAMP) const noexcept;
28    double calcActiveReceptor(double extracellularCAMP) const noexcept;
29
30    void setDiffState(double extracellularCAMP) noexcept;
31    void updateState(double extracellularCAMP) noexcept;

```

class UserCell の追加したメンバ関数の定義を以下に示す。

Listing 15 class UserCell(UserCell.cpp)

```

1 double UserCell::emitMolecule(int32_t moleculeId) noexcept
2 {
3     return cAMP * Kt / H;
4 }
5
6 double UserCell::calcSynthesis(double extracellularCAMP) const noexcept
7 {
8     const double y = (activeReceptor * extracellularCAMP) / (1.0 +
9         extracellularCAMP);
9     const double num = ALPHA * (LAMBDA * THETA + EPSILON * y * y);
10    const double den = 1 + ALPHA * THETA + EPSILON * y * y * (1 + ALPHA);
11    const double phi = num / den;
12

```

```

13     return Q * SIGMA * phi / (Ki + Kt);
14 }
15
16 double UserCell::f1(double cAMP) const noexcept
17 {
18     return (K1 + K2 * cAMP) / (1 + cAMP);
19 }
20
21 double UserCell::f2(double cAMP) const noexcept
22 {
23     return (K1 * L1 + K2 * L2 * C * cAMP) / (1 + C * cAMP);
24 }
25
26 void UserCell::setDiffState(double extracellularCAMP) noexcept
27 {
28     diffCamp = calcSynthesis(extracellularCAMP);
29     diffActiveReceptor = (-activeReceptor * f1(extracellularCAMP) + (1.0 -
        activeReceptor) * f2(extracellularCAMP));
30 }
31
32 void UserCell::updateState(double extracellularCAMP) noexcept
33 {
34     cAMP = diffCamp;
35     activeReceptor += diffActiveReceptor * SimulationSettings::DELTA_TIME;
36     activeReceptor = std::max(0.0, activeReceptor);
37 }

```

Listing 16 class UserMoleculeSpace(UserMoleculeSpace.hpp)

```

1 private:
2     const double _D = 0.024 * 1000000.0;

```

Listing 17 calcConcentrationDiff()

```

1 void UserMoleculeSpace::calcConcentrationDiff() noexcept
2 {
3     constexpr double hydrolysisCoefficient = 5.4;
4
5     for (u_int32_t x = 1; x <= width; x++) {
6         for (u_int32_t y = 1; y <= height; y++) {
7             for (u_int32_t z = 1; z <= depth; z++) {
8                 deltaMoleculeSpace[x][y][z] = diffuse(x, y, z);
9                 deltaMoleculeSpace[x][y][z] += -hydrolysisCoefficient *
                    moleculeSpace[x][y][z];

```

```

10         }
11     }
12 }
13
14 const int32_t width = SimulationSettings::FIELD_X_LEN;
15 const int32_t height = SimulationSettings::FIELD_Y_LEN;
16 const int32_t depth = SimulationSettings::FIELD_Z_LEN;
17
18 for (u_int32_t i = 0; i < cells.size(); i++) {
19     std::shared_ptr<UserCell>& cell = cells[i];
20
21     int32_t x = (int32_t)((cell->getPosition().x + width / 2)) + 1;
22     int32_t y = (int32_t)((cell->getPosition().y + height / 2)) + 1;
23     int32_t z = (int32_t)((cell->getPosition().z + depth / 2)) + 1;
24
25     deltaMoleculeSpace[x][y][z] += cell->emitMolecule(ID);
26 }
27
28 for (u_int32_t i = 0; i < cells.size(); i++) {
29     int32_t x = (int32_t)((cells[i]->getPosition().x + width / 2)) + 1;
30     int32_t y = (int32_t)((cells[i]->getPosition().y + height / 2)) + 1;
31     int32_t z = (int32_t)((cells[i]->getPosition().z + depth / 2)) + 1;
32
33     cells[i]->setDiffState(moleculeSpace[x][y][z]);
34 }
35 }

```

Listing 18 config.yaml

```

1 cell:
2     cell_seed: 10 # の位置などを設定する際に用いるシード値 cell
3     cell_num: 500000 # の初期個数。cellTODO: 複数種類の細胞に対応する。細胞数を
      増やすとやたら重くなる。理由を調べる
4     position_update_method: EULER # AB4, AB3, AB2, EULER から選択
5
6 simulation:
7     sim_step: 100000 # シミュレーションの総ステップ数
8     output_interval: 100 # 出力間隔
9     field_x_len: 1024 # シミュレーションフィールドの座標の長さ x
10    field_y_len: 1024 # シミュレーションフィールドの座標の長さ y
11    field_z_len: 0 # シミュレーションフィールドの座標の長さ z
12    delta_time: 0.001 # ステップあたりのシミュレーション内時間 1
13

```

```

14 molecule:
15     default_molecule_nums: # 各分子の初期個数リストで表現する ( )
16         - 3000
17     delta_time: 0.001
18     field_x_len: 1024 # 分子管理用フィールドの座標の長さ x >= 1
19     field_y_len: 1024 # 分子管理用フィールドの座標の長さ y >= 1
20     field_z_len: 1 # 分子管理用フィールドの座標の長さ z >= 1
21     boundary_condition: # 分子管理用フィールドの境界条件
22         xy1: NEUMANN
23         xy2: NEUMANN
24         yz1: NEUMANN
25         yz2: NEUMANN
26         zx1: NEUMANN
27         zx2: NEUMANN
28
29 # 今回は使用しないので関係なし
30 cell_list:
31     use_cell_list: false # cell を用いるかどうか list
32     grid_size_mag: 32 # cell におけるグリッドの分割倍率。最小は、値は list12^で
        ある必要がある。n
33     search_radius: 256 # この半径内 (の差 position)にあるを力の計算対象とする。
        cell

```
