SuperCon2021 本選問題: 感染症流行のネットワーク解析

1　目的

感染症の流行予測や解析に数理モデルが広く使われている。簡単な数理モデルでは社会をひとつの大きな集団とみなすが、実際の社会は人々が密に接触する集団 (たとえば会社や学校、あるいは年齢で分かれたコミュニティなど) がいくつも集まって構成されている。いったんひとつの集団の中で誰かが感染すると、それは集団の中で早く広まるだろう。それに対して、感染症が集団の間で伝わるのはそれより遅いに違いない。しかも、直接つながりがある集団同士では比較的早く伝わるが、間接的にしかつながっていない集団の間では伝わるのに時間がかかるはずである。ところが、集団同士がどのようにつながっているかは簡単にはわからない。そこで、逆に感染症の伝わりかたから集団同士のつながりかたを推測してみよう。

2　差分 SIR モデル

ひとつの集団しかないときに感染症の流行を記述する最も簡単なモデルは SIR モデルである。このモデルでは人口を三つに分類する。S(susceptible) はまだ感染していない健康者、I(infectious) は他の人に染す能力を持つ感染者、R(removed) は病気から回復して免疫を得た人を表す。$S(t), I(t), R(t)$ という記号を第 t 日のそれぞれの人数を表す変数としよう。SIR モデルの考え方は次の二つにまとめられる

1. 第 t 日の健康者が第 $t+1$ 日に新たに感染者になる数は第 t 日の感染者数と健康者数の両方に比例する。
2. 第 t 日の感染者が第 $t+1$ 日に回復している数は第 t 日の感染者数に比例する。

これを式にまとめると以下のようになる。

$$
\begin{aligned}
S(t+1) &= S(t) - \beta S(t) I(t) \\
I(t+1) &= I(t) + \beta S(t) I(t) - \gamma I(t) \\
R(t+1) &= R(t) + \gamma I(t)
\end{aligned}
$$

足してみれば分かるように集団の人口 $N = S(t) + I(t) + R(t)$ は一定である。

これを**差分 SIR モデル**と呼ぼう (もともとの SIR モデルは微分方程式で表現されている)。β と γ は感染率と回復率に相当する定数である。なお、このモデルでは人数が整数とは限らず、実数値を取るが、そういうモデルだと思っておこう。人数が多ければ、それほど問題ではないはずである。

ここで $R_0 = \frac{\beta N}{\gamma}$ を基本再生産数と呼ぶ。感染者がほとんどいない状況を考えると、$R_0 > 1$ なら感染は拡大し、$R_0 < 1$ では感染は広がらない。図 1 は $\beta = 0.00002, \gamma = 0.1$ で $N = 10000$ の集団に第 0 日にひとりの感染者が現れた場合のその後の S, I, R の変化を示している。これは $R_0 = 2$ に対応しており、感染はいっ

たん拡大後、最終的に 8 割ほどの人が感染を経験して収束する。

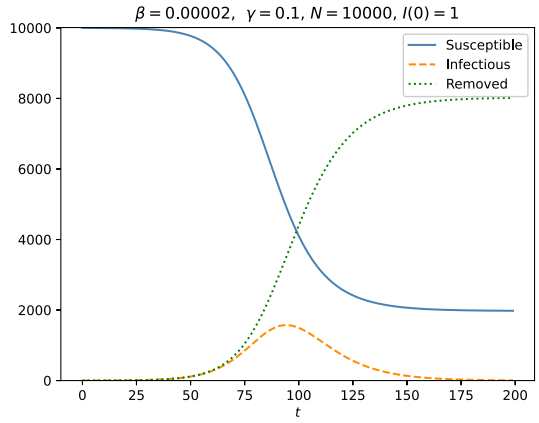

図 1: 差分 SIR モデルの振る舞い

3 ネットワーク SIR モデル

次に、図 2 のようにいくつもの集団がネットワークを作っている場合を考えよう。集団の総数を N_{group} としよう。i 番目の集団の S, I, R, N にそれぞれ添え字 i をつけて表すことにする $(i = 0 \sim N_{\mathrm{group}} - 1)$。ネットワークのつながりを表す**隣接行列** C_{ij} を以下のように定義する。(1) 集団 i と j が直接つながっていれば $C_{ij} = 1$ (2) 集団 i と j が直接つながっていなければ $C_{ij} = 0$。また、$C_{ij} = C_{ji}$ である。(3) 自分自身とはつながらない。つまり、すべての i について $C_{ii} = 0$。これを用いて**ネットワーク SIR モデル**を次のように定義しよう。

$$
\begin{aligned}
S_i(t+1) &= S_i(t) - \beta S_i(t) I_i(t) - \beta' S_i(t) \sum_{j=0}^{N_{\mathrm{group}}-1} C_{ij} I_j(t) \\
I_i(t+1) &= I_i(t) + \beta S_i(t) I_i(t) + \beta' S_i(t) \sum_{j=0}^{N_{\mathrm{group}}-1} C_{ij} I_j(t) - \gamma I_i(t) \\
R_i(t+1) &= R_i(t) + \gamma I_i(t)
\end{aligned}
$$

ここで、β' は直接つながった集団間での感染率で、集団内の感染率より小さいものとする。このモデルでも各集団の人数 N_i は一定に保たれる。図 3 は 100 集団からなるネットワークに対して、各集団の人数を 500 人から 1000 人の間でランダムに分布させ、$\beta = 0.0002, \gamma = 0.1, \beta' = 0.000001$ としたときに各集団の感染者数がどのように変化するかを見たものである。0 日目にはひとつの集団にだけひとりの感染者がいるとした。感染

者が増減するタイミングが集団ごとに異なるのが分かる。

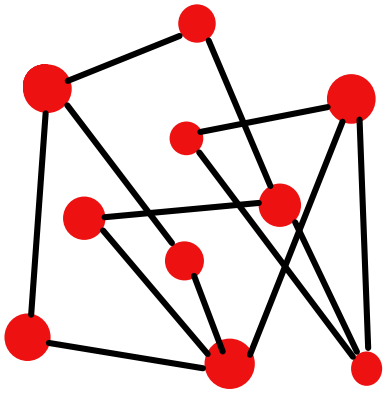

図 2: ネットワークの例 ($N_{\text{group}} = 10$, リンク数 13)

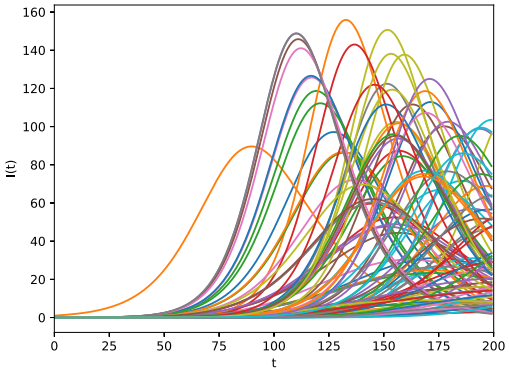

図 3: 100 集団からなるネットワーク SIR モデルの感染者数

4 問題

各集団の人数が与えられるが、集団間のつながりは未知である。第 0 日に 0 番目の集団で一人が感染者になった。その後、ネットワーク SIR モデルに従ってネットワーク内で感染が広がってゆく。第 T 日の各集団での感染者数の組 $\{I_i^{\text{prob}}(T)\}(i = 0 \sim N_{\text{group}} - 1)$ が問題として与えられるので、それを再現する隣接行列 C_{ij} を推定するプログラムを作成してほしい。

推定精度は次の**誤差二乗和**で定義する。推定された隣接行列を使って求められた第 T 日の各集団での感染者数の組を $\{I_i^{\text{inf}}(T)\}(i = 0 \sim N_{\text{group}} - 1)$ とすると、誤差二乗和は

$$E = \sum_{i=0}^{N_{\text{group}}-1} (I_i^{\text{inf}}(T) - I_i^{\text{prob}}(T))^2$$

で求められる。これが小さいほど推定はうまくいっているものとしよう。この誤差二乗和をできるだけ小さくするような隣接行列を推定する。

5 詳細

1. 入力として与えられるのは各集団の人数 N_i の組と各集団での T 日目の感染者数 I_i^{prob} の組である。これらは提供されるヘッダーに含まれる SC_input 関数を呼ぶことで読み込まれる。
2. 集団の数は 100 に固定される。日数 T は 200 日に固定される。感染率と回復率は $\beta = 0.0002$、$\beta' = 0.000001$、$\gamma = 0.1$ に固定される。これらの値はヘッダーファイルで定義されている。
3. 集団間のつながりの総数は問題ごとに異なるが、問題で与えられる。
4. 作成するプログラムの出力は隣接行列 C_{ij} の組である。これは提供されるヘッダーに含まれる SC_output 関数を呼ぶことで書き出される。
5. プログラムはタイムアウトで強制終了させるので、終了処理は書かなくてよい。それまでに結果を何度

出力しても構わないが、制限時間内に出力された最後の結果が審査に使われる。

6 勝利条件

1. 提出されたプログラムで問題を 3 問解く。実行時間の上限は各問 5 分とする (実際には 6 分間走らせ、5 分以内に出力された最後の結果を採用する)。
2. 各問について、誤差二乗和が小さいものから順に順位をつけ、1 位 20 点、2 位 19 点、以下同様に点数化する。誤差二乗和が全く同じときには出力までの時間が短いチームを上位とする。
3. 3 問の点数を合計して総合順位をつける。点数が等しい時には、3 問の計算時間の合計が短いチームを上位とする。

7 ヘッダーファイル

配布するコンテスト専用ヘッダーファイル sc21.h には以下のように定数・配列・関数が定義されている。

```
# define N_GROUP 100

const double BETA=0.0002;
const double BETA2=0.000001;
const double GAMMA=0.1;

const int T=200;

void SC_input();
void SC_output();

int N_LINK;

int N[N_GROUP];
double I_PROB[N_GROUP];

int C[N_GROUP][N_GROUP];

double TIME0;

void SC_input(){
    int rank = 0;
#ifdef MPI_VERSION
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
#endif

    if(0 == rank){
        scanf("%d",&N_LINK);
        int i;
        for (i=0; i< N_GROUP; i++){
            scanf("%d\n",&N[i]);
        }
        for (i=0; i< N_GROUP; i++){
            scanf("%lf",&I_PROB[i]);
        }
    }
#ifdef MPI_VERSION
    MPI_Bcast(&N_LINK, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(N, N_GROUP, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(I_PROB, N_GROUP, MPI_DOUBLE, 0, MPI_COMM_WORLD);
#endif

    TIME0=omp_get_wtime();
}

void SC_output(){

  printf("# elapsed time= %f \n",omp_get_wtime()-TIME0);

  int i,j;
  for (i=0; i < N_GROUP; i++){
    for (j=i+1;  j < N_GROUP; j++){
      printf("%d ",C[i][j]);};
  };
  printf("\n");
  fflush(NULL);
};
```

これを

```
# include <stdio.h>
# include <math.h>
# include "cssl.h"
# include <stdlib.h>
# include <omp.h>
```

```
# include "mpi.h"
# include "sc21.h"
```

のようにインクルードして使う。

1. 関数 SC_input() はプログラム中で最初の実行文とすること (MPI を使用するときには、最初の実行文が MPI_init にする必要があり、SC_input() はその直後の実行文とする)。これ以前に実行文を書いてはいけない。SC_input を実行することにより、整数変数 N_LINK にその問題での集団間を結ぶリンクの総数が、整数配列 N[N_GROUP] に各集団の人数が、また double 型配列 I_PROB[N_GROUP] に各集団の T 日目の感染者数が格納される (人数だが、モデルの性質上整数ではないことに注意)。なお、データは標準入力から読み込まれる。データの読み込みには必ずこの SC_input 関数を使うこと。
2. 解答の C_{ij} を整数配列 C[N_GROUP][N_GROUP] に格納し、関数 SC_output() を呼ぶことにより、結果が出力される。SC_output 関数以外の方法で出力してはならない。SC_output は何度呼んでもよい。SC_output の中身を読めば分かる通り、C_{ij} の半分だけが出力される ($C_{ij} = C_{ji}$ だから)。なお、配列 C の要素は 0 または 1 でなくてはならない。それ以外の数値が出たときは不正解とする。また、集団間を結ぶリンクの総数は N_LINK でなくてはならない (つまり、配列 C に含まれる 1 の数は 2N_LINK 個である)。
3. 定義されている定数とグローバル変数は以下の通りである。
   N_GROUP　集団の総数 (define で 100 に設定されている)
   const double BETA=0.0002　β
   const double BETA2=0.000001　β'
   const double GAMMA=0.1　γ
   const int T=200　T
   int N_LINK　集団間のつながりの総数 (ネットワークを描いた時の線の数)。つまり、隣接行列中の 1 の総数の半分。
   double TIME0　時間測定のための変数

8 乱数

0 から 1 までの乱数を発生させるには c_dm_vranu5 関数を使う。使い方は

1. ```
   int ix,icon;
   double dwork[8];
   double r[n];
   ```
   の四つの変数と配列を宣言。n には発生させたい乱数の数以上の整数を指定する。
2. ix に乱数の種となる整数を入れる。乱数の種が違えば違う乱数列が発生されるので、MPI で並列に計算する時には各スレッドごとに違う種を使えばよい。
3. `c_dm_vranu5(&ix, r, n, (long)0, dwork, &icon);`
   を呼ぶと、n 個の乱数が r に格納され、ix は 0 にリセットされる。
   ```

9　コンパイルオプション

コンパイル時のオプションは以下のように統一する。プログラム名を prog.cpp とすると C++ 言語の場合

```
mpiFCC -Ofast -fopenmp -Nclang -SSL2BLAMP prob.cpp
```

また C 言語の場合、プログラム名を prog.c とすると

```
mpifcc -Ofast -fopenmp -Nclang -SSL2BLAMP prob.c
```

でコンパイルする。

10　ジョブスクリプト

今回のコンテストでは OpenMP のスレッド数や MPI の並列数を実行時に指定する必要があるので、ジョブスクリプトも提出してもらう。MPI を使う場合のジョブスクリプトは以下のようになる。elapse は実行時間制限 6 分の指定である。C 言語か C++ 言語かによって、適切なコンパイルコマンドを残す。指定しなくてはならないのは 5 行めの MPI 並列数 (この例では 4) と 8 行目の OpenMP スレッド数 (この例では 12) である。実行時には標準入力から問題を読み、標準出力に結果を出す。

```
#!/bin/bash
#PJM --rsc-list "node=1"
#PJM --rsc-list "rscgrp=supercon2021"
#PJM --rsc-list "elapse=0:06:00"
#PJM --mpi "proc=4"
#PJM -j

export OMP_NUM_THREADS=12
echo $OMP_NUM_THREADS

# do not generate empty output files
export PLE_MPI_STD_EMPTYFILE=off

date

# compile: choose mpiFCC (for C++) or mpifcc (for C) and edit the source file name
#  C++
mpiFCC -Ofast -fopenmp -Nclang -SSL2BLAMP prog.cpp
#  C
#mpifcc -Ofast -fopenmp -Nclang -SSL2BLAMP prog.c
```

```
# execute the program
mpiexec --stdin input1 --stdout output1.${PJM_JOBID} ./a.out
```

また、MPI を使わない場合にはコンパイルと実行の仕方が異なり、以下のようになる。OpenMP のスレッド数は 1 ノードの最大である 48 としてある。

```
#!/bin/bash
#PJM --rsc-list "node=1"
#PJM --rsc-list "rscgrp=supercon2021"
#PJM --rsc-list "elapse=0:06:00"
#PJM -j

export OMP_NUM_THREADS=48
echo $OMP_NUM_THREADS

# do not generate empty output files
export PLE_MPI_STD_EMPTYFILE=off

date

# compile: choose FCC (for C++) or fcc (for C) and edit the source file name
#  C++
FCC -Ofast -fopenmp -Nclang -SSL2BLAMP prog.cpp
#  C
#fcc -Ofast -fopenmp -Nclang -SSL2BLAMP prog.c

# execute the program
./a.out < input1 > output1.${PJM_JOBID}
```

のようにする。

11 提出物

プログラムのソースコードとジョブスクリプトを提出する。なお、ソースコードは必ずひとつのファイルにまとめてあること。分割したファイルや makefile の使用は認めない。提出方法の詳細は期間中に説明する。