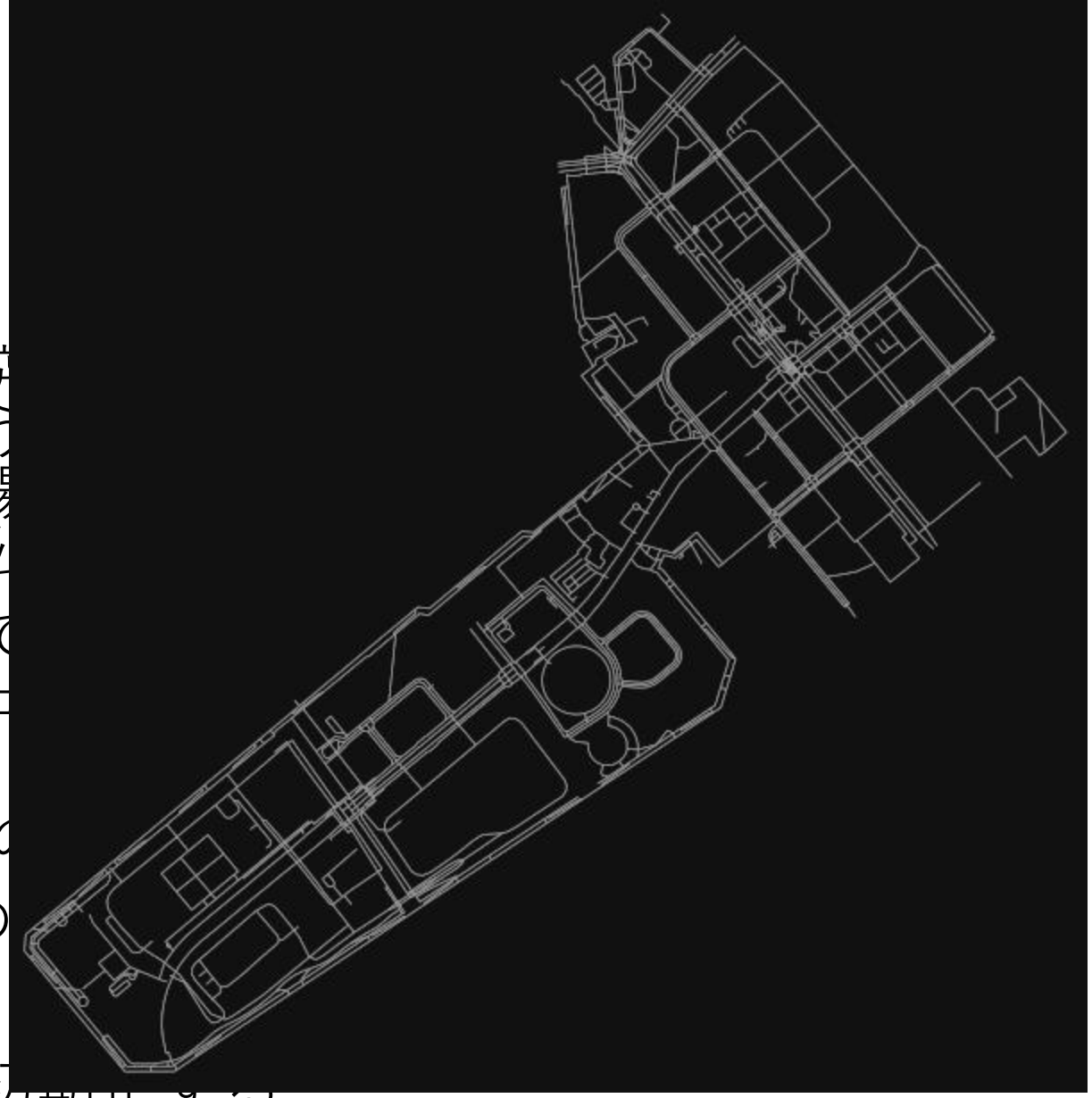
概要

- テーマ: Osmnxライブラリを使用して豊洲地区のwalkタイプグラフ（ネットワーク）データを利用して、豊洲でのある病気の感染をシミュレーションする。
- 背景: 特定の地区での人流と感染の動向を理解することは伝染の予防に役立つが、オープンデータの入手が難しいため、シミュレーションが頻繁に行われる。ただし、シミュレーションシステムの公開が少ない。
- 目的: 地理空間で人と人の間の感染をモデル化し、シミュレーションを行うことで感染の流れを理解する（システム構築）。
- 方法: 感染は一つの所から広がると仮定し、数式でモデル化した後、グラフデータを使用してシミュレーションを行った。
- 結果: シミュレーションの結果から得られる情報によれば、感染は始まりの場所から最も遠い場所に広がる傾向が見られる。
- 結論: シミュレーションの結果から、人流と感染が同時に発生する場合、これは一種の拡散であり、拡散が進むにつれて感染が増加していく様子が観察される。

- ネットで人流感染の関連研究
  わかった　一つ目はこのよう
  はシミュレーションをする場
  は会社が作ったものなので公
- 自分が人流と感染の研究して
  て問題設定して自作のシミュ
  る。
- Pythonのosmnxライブラリの
- このグラフはグラフは990の

- どのノードにも１０いると初期化する



シミュレーション問題の設定

- 初期状態は全てのノードに10を設置する。ランダムで一つのノードを5人の感染者がいると設置する。
- 離散化した状態を考える。S(t)として考えるtは0からnまでの整数
- 一つの状態はグラフ全体としてのもの　S(t)はtの時各ノードの人数と感染者と非感染それぞれ何人いるかの状態と設定
- tからt+1することが１タイムステップと考える
- １タイムステップは二つの行動をとる:1人流の移動　２：感染
- 全てのノードに対してその隣接ノードに移動する時間が全部同じと設定する
- 一つのノードの感染はそのノードにいる感染と未感染者と関連するが他のノードと関連しないと設定
- この感染の経路は間接経路はないと設定
- 一回の移動でこのノードいる全員が必ず他のノードに行くと設定

シミュレーションの方法の構築ー数式部分

- まずは人流の移動の数式を考える
- 発想はマルコフ連鎖から始まる
- S(t)の状態の時各ノードの総人数はX(S(t))とする。(990,1)のベクトル
- 遷移行列とPA(S(t)) PAと略する　行の和が１とする(990,990)の行列
- PAX＝Xなので この式は順列不変（ permutation invariance)となる
- 人流の変化は常微分方程式で考えると：$\frac{dX}{dt} = f(x)$ $f(x)$は人流変化の関数 $= fout(x) - fin(x)$
- 今回は離散化して考えるから簡単な数値法のオイラ方で計算してみる
- 二つの行動があるので 行動しない場合はa=0,移動はa=1,感染はa=2と表す
- $\mathrm{X} = X(S(t)|a = 1) + (t + 1 - t)f(X(S(t)|a = 0) = X(S(t)|a = 0) + f(X(S(t)|a = 0)$
- fの決め方を考える　f= $fout(x) - fin(x)$と考える　上記の遷移行列を使う
- $\mathrm{X}(S(t)|1) = X(S(t)|0) + (PA^T X(S(t)|0) - PAX(S(t)|0)$ 簡単に表す:$(I + PA^T - PA)X$
- PAX＝Xを使って結果は$X(S(t)|0) - PAX(S(t)|0) = 0$
- $\mathrm{X}(S(t+1)|a = 1) = PA^T X(S(t)|a = 0)$

シミュレーションの方法の構築ー数式部分

- 感染の部分を考える
- 感染の部分の計算はX(S(t))を未感染者X1(S(t))と感染者X2(S(t))を分けて計算する
- 感染の計算siモデルを使う
- $\frac{dX1}{dt} = -\frac{\beta X1.\times X2}{(X1.+X2)}.\frac{dX2}{dt} = \frac{\beta X1.\times X2}{(X1.+X2)}$
- オイラ方を使うx＋f（x）のf（x）を$\frac{\beta X1.\times X2}{(X1.+X2)}$にする
- $\mathrm{X1}(S(t)|a=2) = \mathrm{X1}(S(t)|a=1) - \frac{\beta X1(S(t)|a=1)\times X2(S(t)|a=1)}{(X1(S(t)|a=1)\times.+X2(S(t)|a=1))}$
- $\mathrm{X2}(S(t)|a=2) = \mathrm{X2}(S(t)|a=1) + \frac{\beta X1(S(t)|a=1)\times X2(S(t)|a=1)}{(X1(S(t)|a=1)\times.+X2(S(t)|a=1))}$

シミュレーションの方法の構築

- 数値計算は整数にはならない　物理空間上表すことができないからグラフでシミュレーションのシステムを構築してみる
- 実際のプログラミングの部分では移動は隣接のノーのリストをrandom_choiceで選ぶことができるので感染とデータの格納が難しいところになる
- データの格納と追跡を考えてX1,X2それぞれ(t,990,10)の配列にするX1(S(t))とX2(S(t))は(990,10)の配列になる列がノードを表す行が人数を表す　(1,1)は 2 番目のノードにいる 2 番目の人と表す
- 990はnodeリストを作る時のインデックス数にする nを何人目を表す
- （index(node),n)に格納するデータは現在の位置（どのノードにいるか)にする
- 例: $X1\left(S(t)\right) = \begin{pmatrix} node\_2 & \cdots & node\_n \\ \vdots & \ddots & \vdots \\ node\_2 & \cdots & node\_5 \end{pmatrix}$
- Cという配列を計算する　C配列は(990,990)の配列で列は出発点のノードを表す　行は現在にいるノードを表す
- 初期の状態の例：$C = \begin{pmatrix} 10 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 10 \end{pmatrix}$ 行ごと和を求める$\mathrm{CC} = \begin{pmatrix} 10 \\ \vdots \\ 10 \end{pmatrix}$ これは現在の各ノードの人数となる
- C(X1)= $\begin{pmatrix} a & \cdots & b \\ \vdots & \ddots & \vdots \\ 0 & \cdots & d \end{pmatrix}$ C(X2)= $\begin{pmatrix} e & \cdots & f \\ \vdots & \ddots & \vdots \\ 0 & \cdots & g \end{pmatrix}$ CC(x1)= $\begin{matrix} \sum row(i) \\ \vdots \\ \sum row(i) \end{matrix}$ CC(x2)= $\begin{matrix} \sum row(i) \\ \vdots \\ \sum row(i) \end{matrix}$
- 変化関数$delta$は $\frac{\beta CC1(X1) \times \beta CC2(X2)}{CC1(X1)+CC2(X2)}$ になる
- 新しく感染者がいるノードの位置（pと記入）＝ Where(X1(t=tn) .==nodes[index.(delta)]) (.=point-wise)
- Random choice(p, .round(delta))
- X1[p].= 0 x2[p].=node番号

システム構築 アルゴリズムの流れ

- Initialize X1(S(0)),X2(S(0)),trajectory(X1), trajectory(X2) node=random.choice(nodes_list)
- For i =0 to MAX_TIME_STEP:
  for all nodes in X1 X2:
  X1,X2← list(random.choice(neighbors_list))
- trajectory(X1)←X1 trajectory(X2)←X2
- initialize C(x1)←zeros(990,990),C(x2) ←zeros(990,990)
- for ii =0 to 989 :
- for l =0 to 9:
- v1= trajectory(X1)[-1][ii][l]
- v2= trajectory(X2)[-1][ii][l]
- if v!=0 :
- count1[node.index(v1)[ii] +=1
- elif c!=0:
- count2[node.index(v2)[ii] +=1

cc(x1)←sum(C(x1))on axis=1 cc(x2)←sum(C(x2))on axis=1

do: Compute delta $\leftarrow \frac{\beta CC1(X1)\times \beta CC2(X2)}{CC1(X1)+CC2(X2)}$ β~Normal(0,1)

do: index(delta>0)

for all node which delta>0

if cc(x1)> cc(x2):
cc(x2)=cc(x2)

elif. : cc(x1)< cc(x2):
cc(x2)=cc(x1)

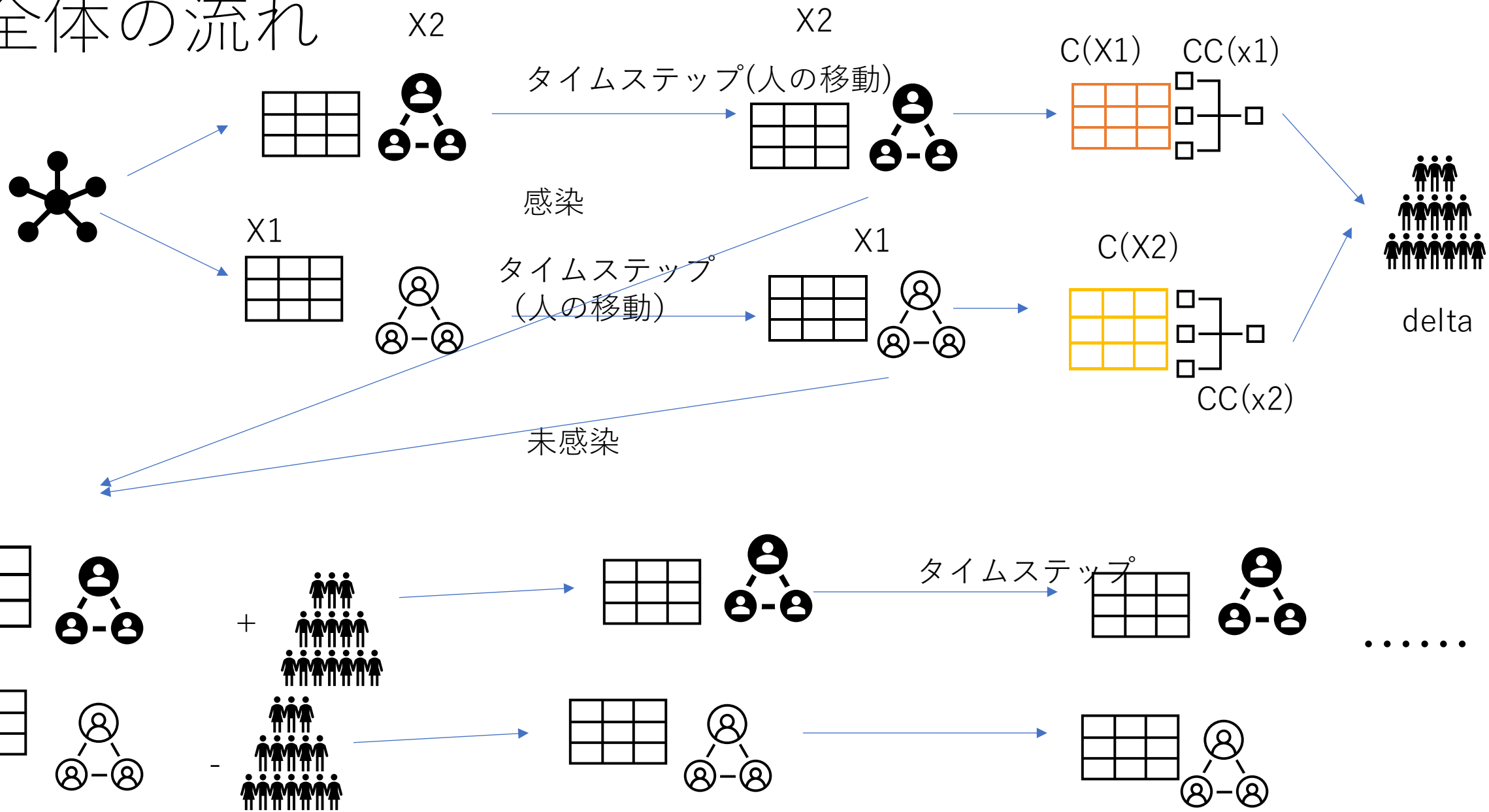
do: position ← where node [index(delta>0)]
random.choice (node,cc(x2)) in all position
trajectory(X1)[position]←0. trajectory(X2)[position]←node

コード

```
x1=20*np.ones(len(nodes))
ini=random.choice(range(len(x1)))
x1[ini]=10
x2=np.zeros(len(nodes))
x2[ini]=10
a1=a.toarray()
t=np.random.uniform(0, 2, (990, 990))
b=t*a1
for i in range(len(b)):
     #print(np.sum(b[i]))
     b[i,:]=b[i,:]/np.sum(b[i,:])
for i in range(60):
   x1=b.T@x1
   x2=b.T@x2
   beta=np.round(np.random.uniform(0,1,990),1)
   delta=np.round(beta*((x1*x2)/(x1+x2)))
   delta=np.nan_to_num(delta,nan = 0)

   for i in range(len(delta)):
    c1=delta[i]
    c2=x1[i]
    if c1<c2 or c1 ==c2 :
      x1[i]=x1[i]-c1
      x2[i]=x2[i]+c1
    elif  c1>c2 and c1 >=1 and (c1-c2)<c2 :
     x1[i]=x1[i] -(c1-c2)
     x2[i]=x2[i] +(c1-c2)
    elif c1>c2 and c1 >=1 and (c1-c2)>c2 :
     x1[i]=x1[i]-1
     x2[i]=x2[i]+1
```

```
x1=20*np.ones(len(nodes))
ini=random.choice(range(len(x1)))
x1[ini]=10
x2=np.zeros(len(nodes))
x2[ini]=10
a1=a.toarray()
for i in range(60):
   t=np.random.uniform(0, 2, (990, 990))
   b=t*a1
   for i in range(len(b)):
     #print(np.sum(b[i]))
     b[i,:]=b[i,:]/np.sum(b[i,:])
   x1=b.T@x1
   x2=b.T@x2
   beta=np.round(np.random.uniform(0,1,990),1)
   delta=np.round(beta*((x1*x2)/(x1+x2)))
   delta=np.nan_to_num(delta,nan = 0)
   for i in range(len(delta)):
    c1=delta[i]
    c2=x1[i]
    if c1<c2 or c1 ==c2 :
      x1[i]=x1[i]-c1
      x2[i]=x2[i]+c1
    elif  c1>c2 and c1 >=1 and (c1-c2)<c2 :
     x1[i]=x1[i] -(c1-c2)
     x2[i]=x2[i] +(c1-c2)
    elif c1>c2 and c1 >=1 and (c1-c2)>c2 :
     x1[i]=x1[i]-1
     x2[i]=x2[i]+1
```

全体の流れ


X2
X2
タイムステップ(人の移動)
C(X1)
CC(x1)
感染
X1
X1
タイムステップ
(人の移動)
C(X2)
CC(x2)
delta
未感染
+
-
タイムステップ
.....

数式だけで計算して結果を出す

- PA=ランダムでサンプリングしたランダム変数行列(990,990)と隣接行列と点ごと掛け算して行ベクトルを行ベクトルの和で割る
- 変数のランプリングは一様分布あるいはガウス分布を使う
- PAを固定する場合と固定しない場合ぞれぞれ計算する
- 変数行列をuniform(0,2)から作る　betaはuniform(0,1)からサンプリング
- ループを６０回する

Paを固定する場合

```
1 np.sum(x1),np.sum(x2),np.sum(x1)+np.sum(x2
```

```
(1983.4428296582828, 7916.557170341717, 9900.0)
```

Paをループごと場合

```
1 np.sum(x1),np.sum(x2),np.sum(x1)+np.sum(x2
```

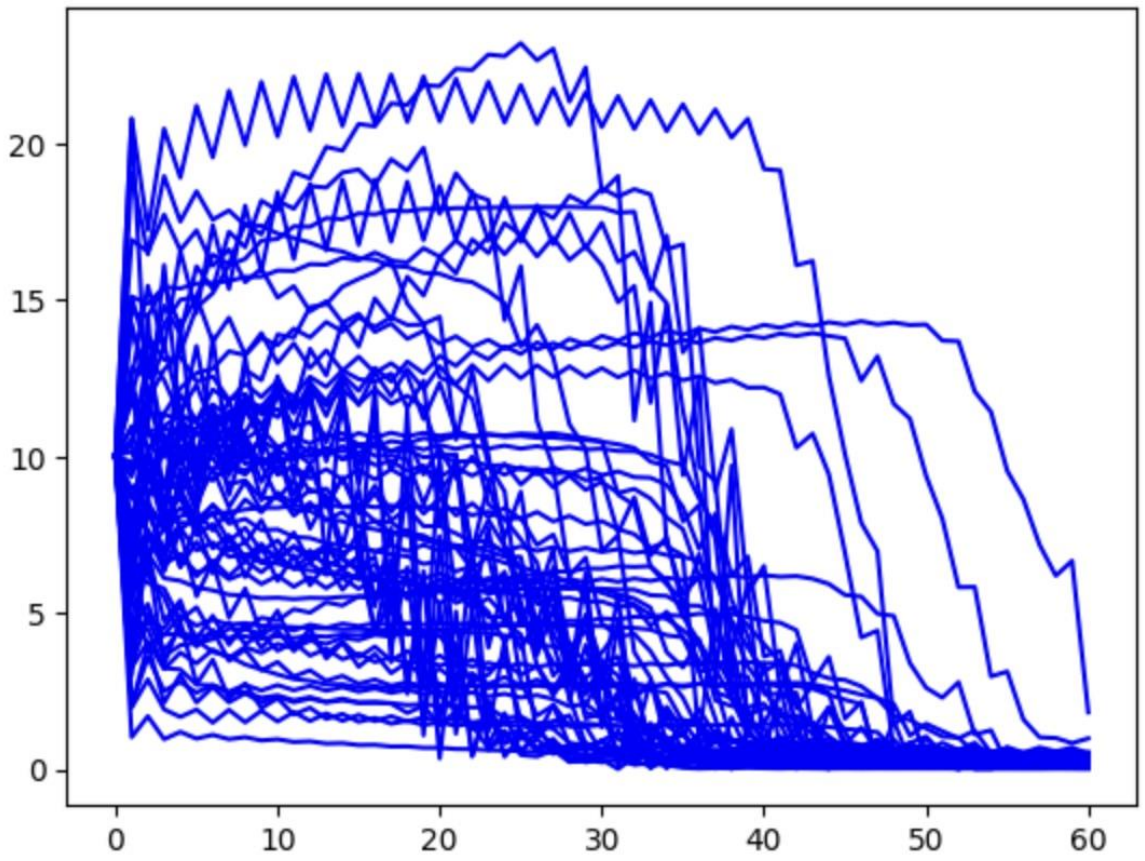
```
(5722.5594444487215, 4177.4405555512785, 9900.0)
```

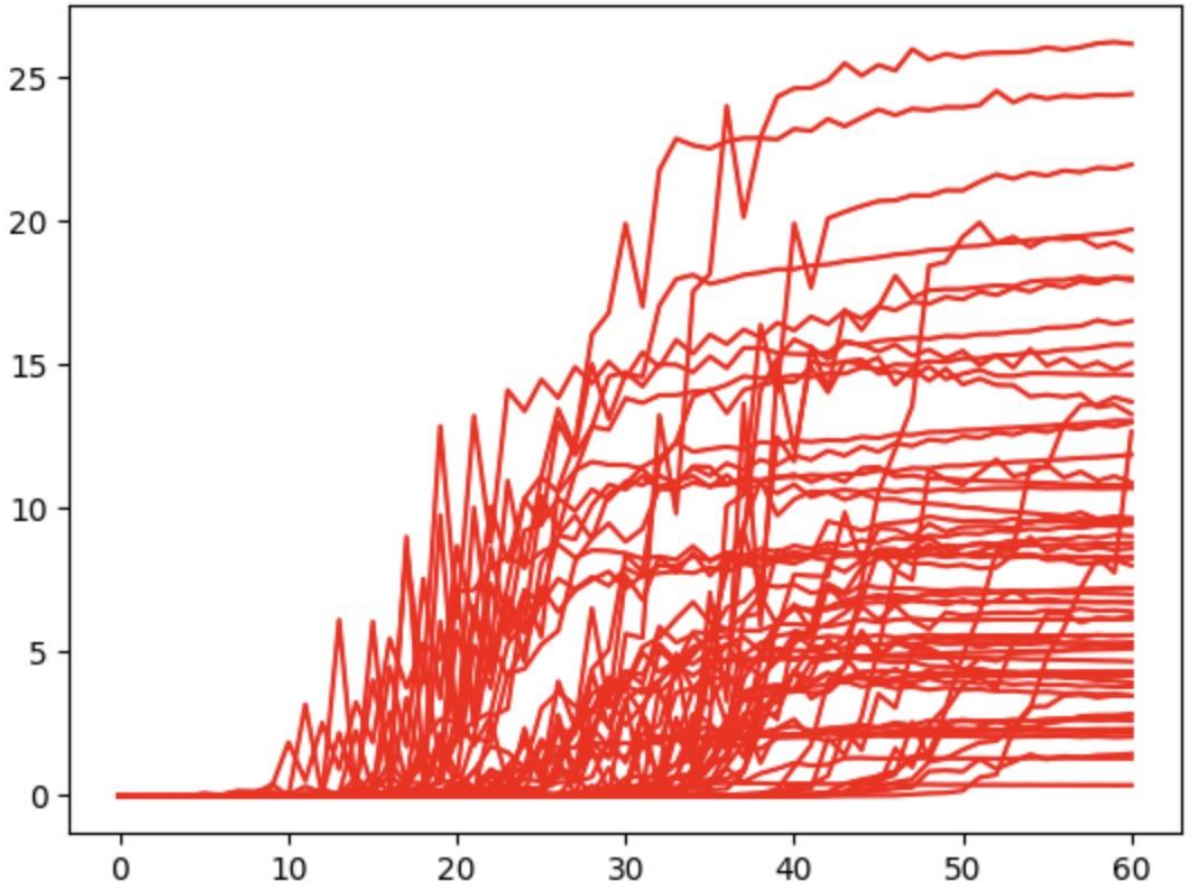
コード

```
cn=random.choice(nodes)
nod=[[i]*10 for i in nodes]
nod[nodes.index(cn)][0:5]=[0]*5
nod1=[[0]*10 for i in nodes]
nod1[nodes.index(cn)][0:5]=[cn]*5
tra2=[nod]
tra3=[nod1]
for i in range(60):
    now_p=[[ random.choice(list(G.neighbors(l))) if l !=0 else 0  for l in tra2[-1][i]] for i in range(len(nod))]
    tra2.append(now_p)
    now_p1=[[ random.choice(list(G.neighbors(l))) if l !=0 else 0  for l in tra3[-1][i]] for i in range(len(nod1))]
    tra3.append(now_p1)
    count1=np.zeros((990,990))
    count2=np.zeros((990,990))
    for i in range(990):
        for l in range(10):
         v=tra2[-1][i][l]
         c=tra3[-1][i][l]
         if v != 0 :
              count1[nodes.index(v)][i]+=1
         elif c!=0:
             count2[nodes.index(c)][i]+=1
    s=np.add(np.sum(count2,axis=1), np.sum(count1,axis=1))
    pd=np.multiply(np.sum(count2,axis=1), np.sum(count1,axis=1))
    d=np.divide(pd,s)
    d=np.nan_to_num(d,nan = 0)
    ds=np.round(np.abs(np.random.normal(loc   = 0, scale=1,size  = 990,)*d))
    nl=np.where(ds>0)[0]
    tra2=np.array(tra2)
    tra3=np.array(tra3)
```

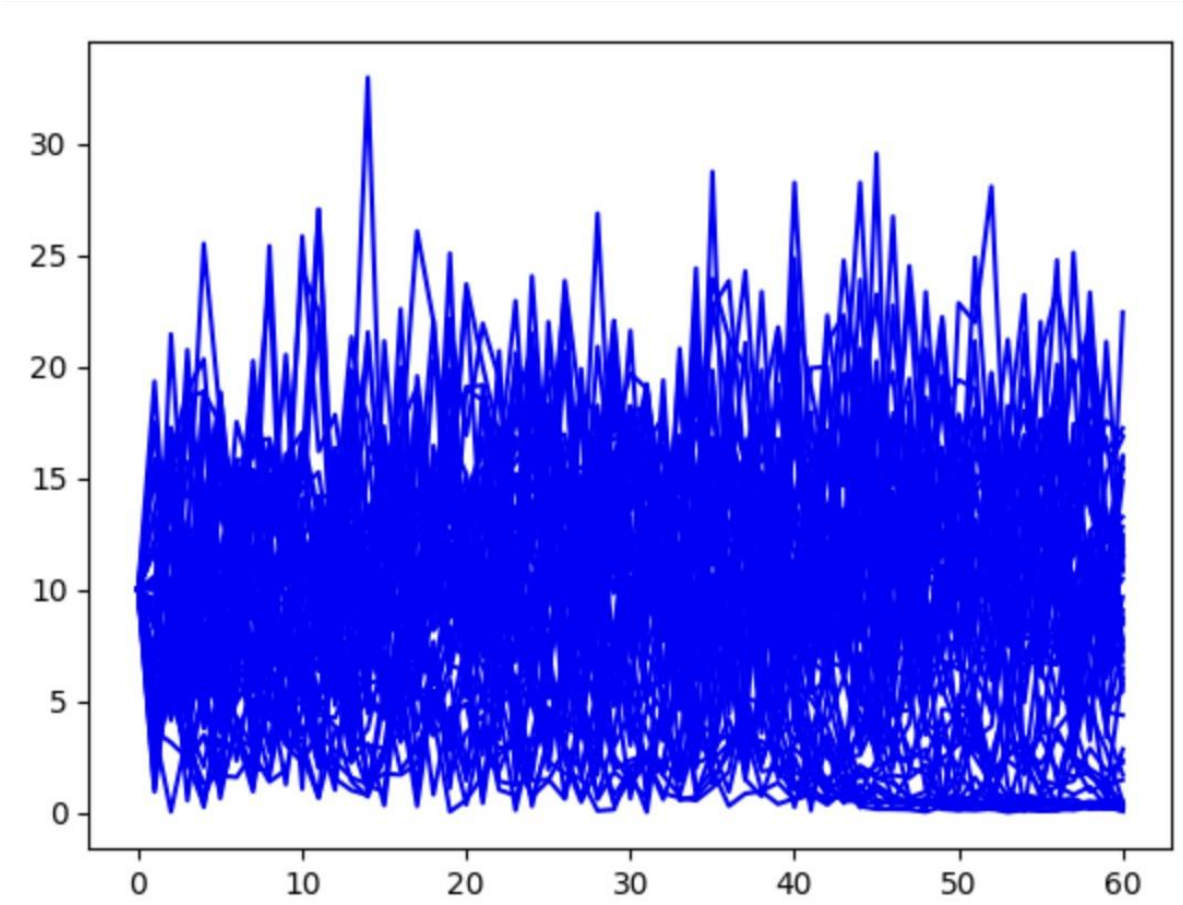
```
for i in range(len(nl)):
  c1=np.sum(count1[nl[i]])
  c2=ds[nl[i]]
  if  c1 > c2 or c1 == c2:
     on=np.where(tra2[-1]== nodes[nl[i]])
     nn=len(on[0])
     re=[np.random.choice(range(nn),int(ds
     xl=on[0][re]
     yl=on[1][re]
     tra2[-1][xl,yl] =0
     tra3[-1][xl,yl]= nodes[nl[i]]

  elif  c1 < c2 and  c1 >=1 :
     on=np.where(tra2[-1]== nodes[nl[i]])
     nn=len(on[0])
     re=[np.random.choice(range(nn),int(c1
     xl=on[0][re]
     yl=on[1][re]
     tra2[-1][xl,yl] =0
     tra3[-1][xl,yl]= nodes[nl[i]]

tra2=tra2.tolist()
tra3=tra3.tolist()
```

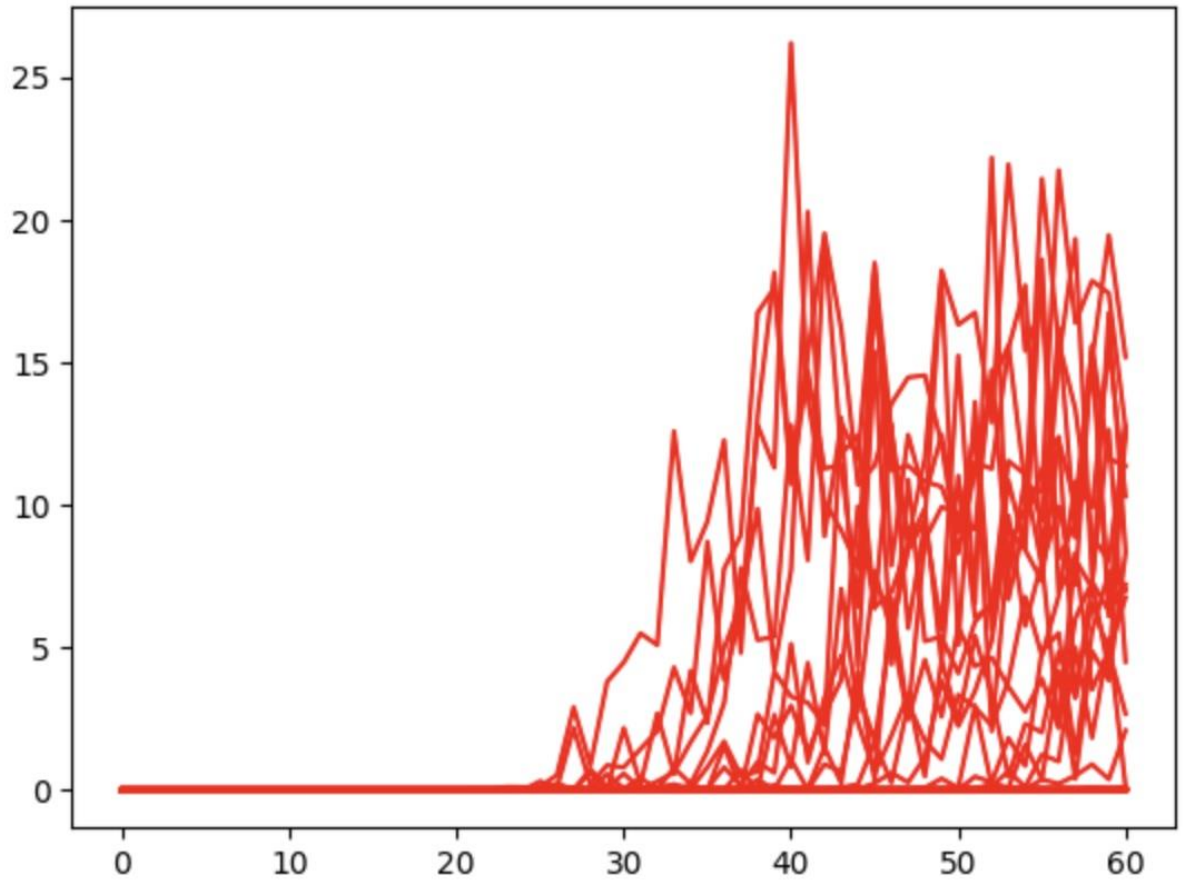
Paを固定する場合各ノード60回ループする中で人数の変化



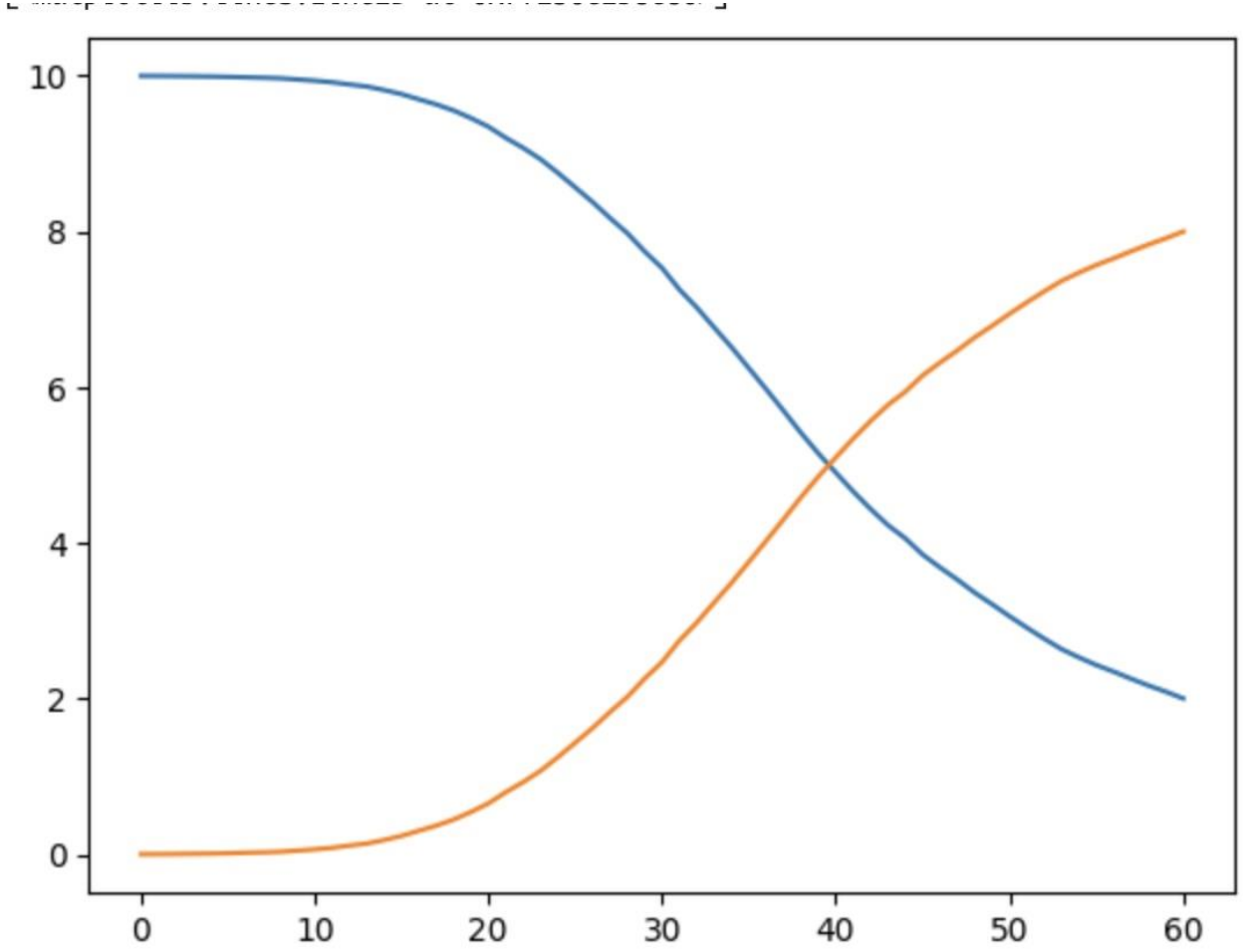


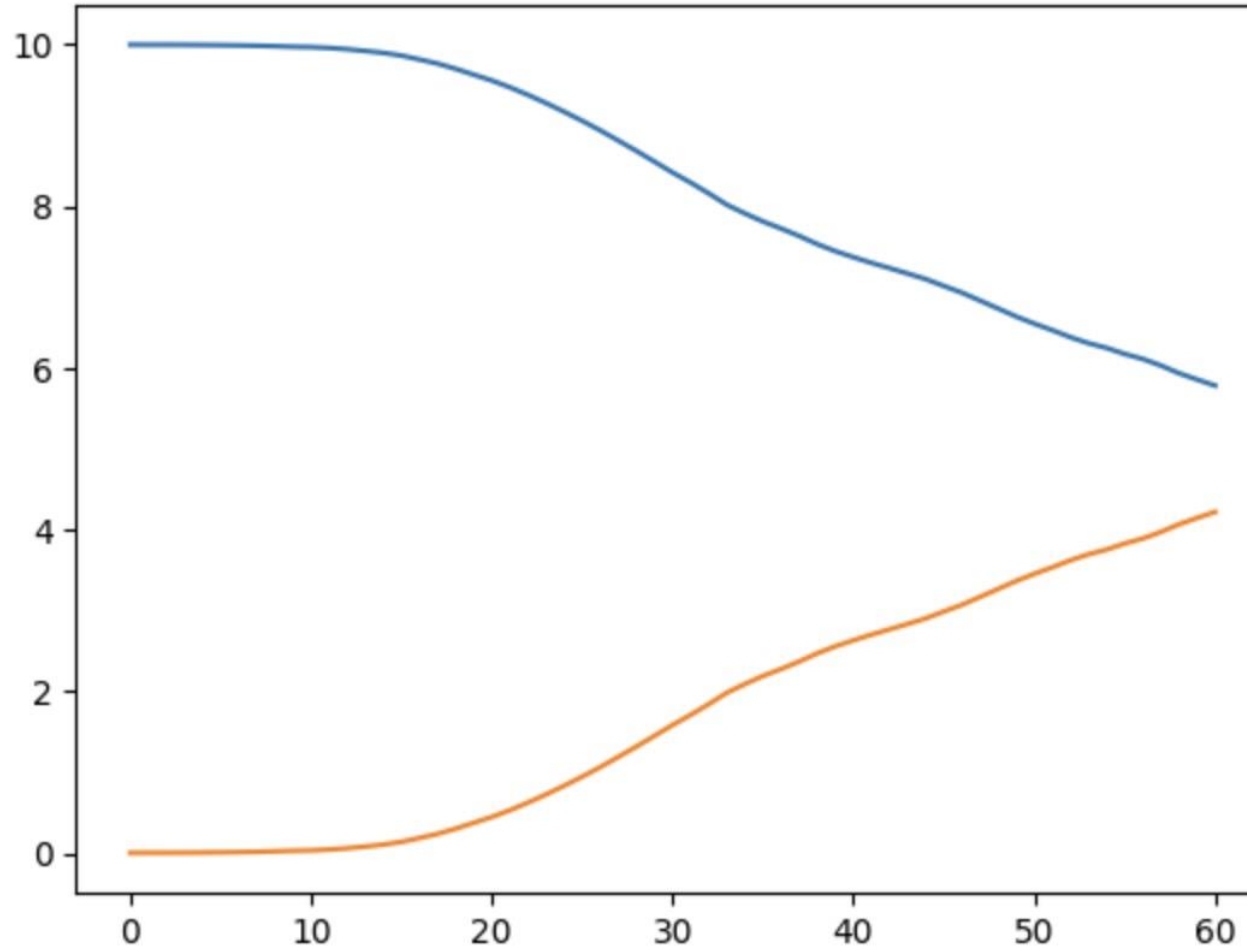
Paを固定しない場合各ノード60回ループする中で人数の変化



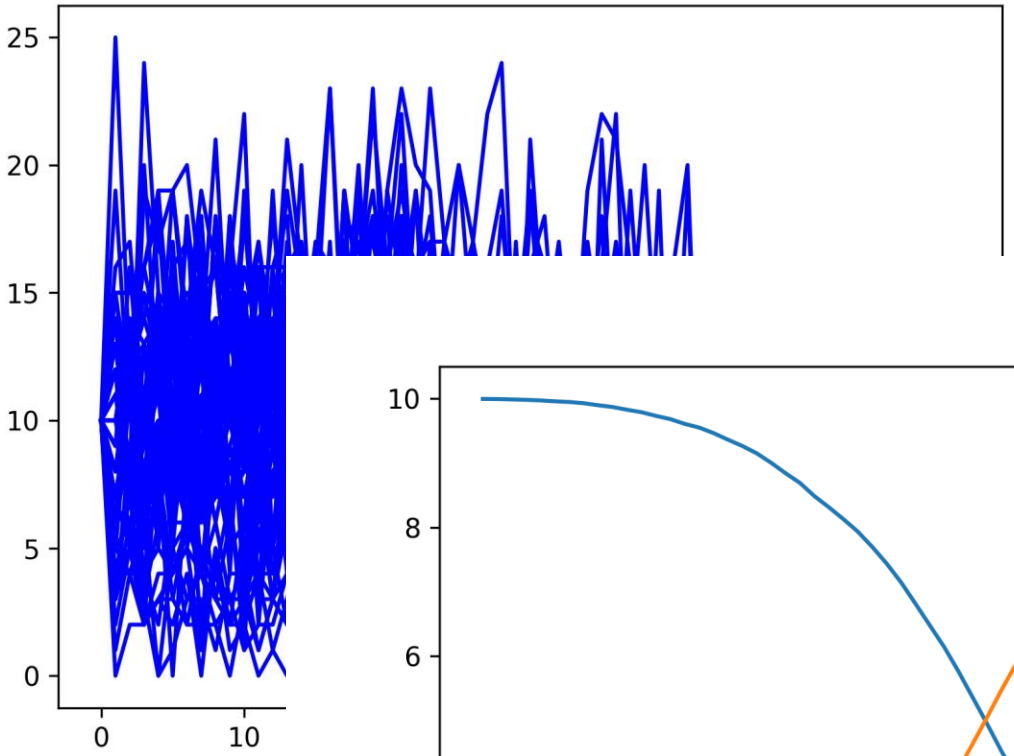


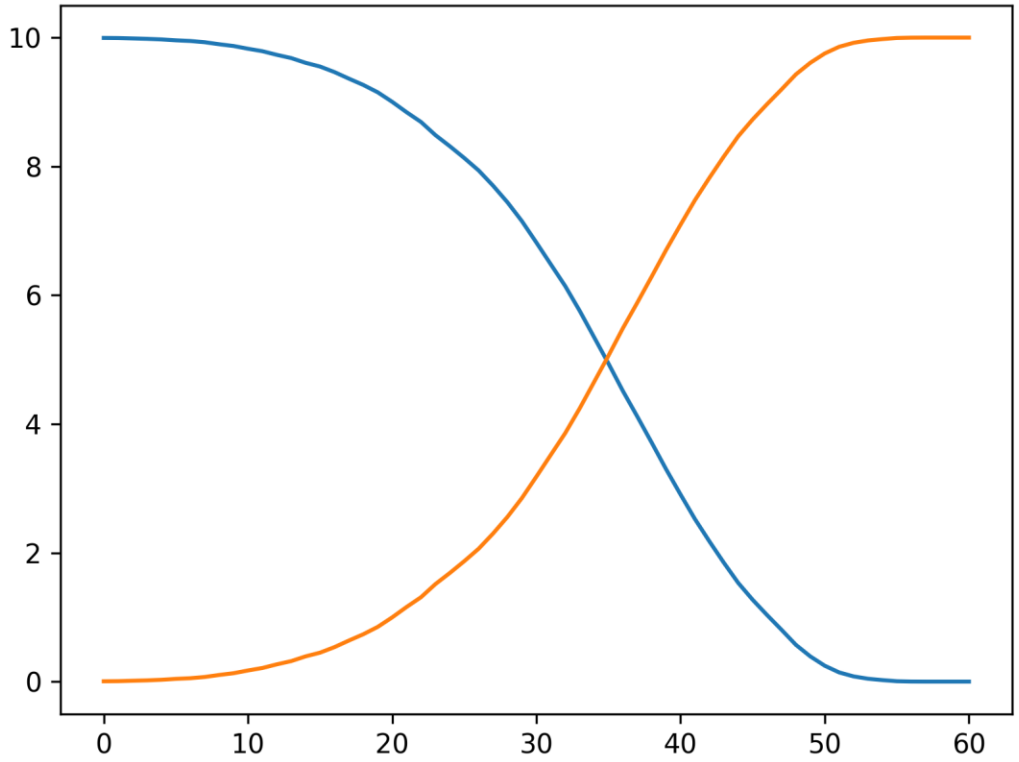
二つの場合の全ノードの平均
左が固定の場合右は固定しない場合

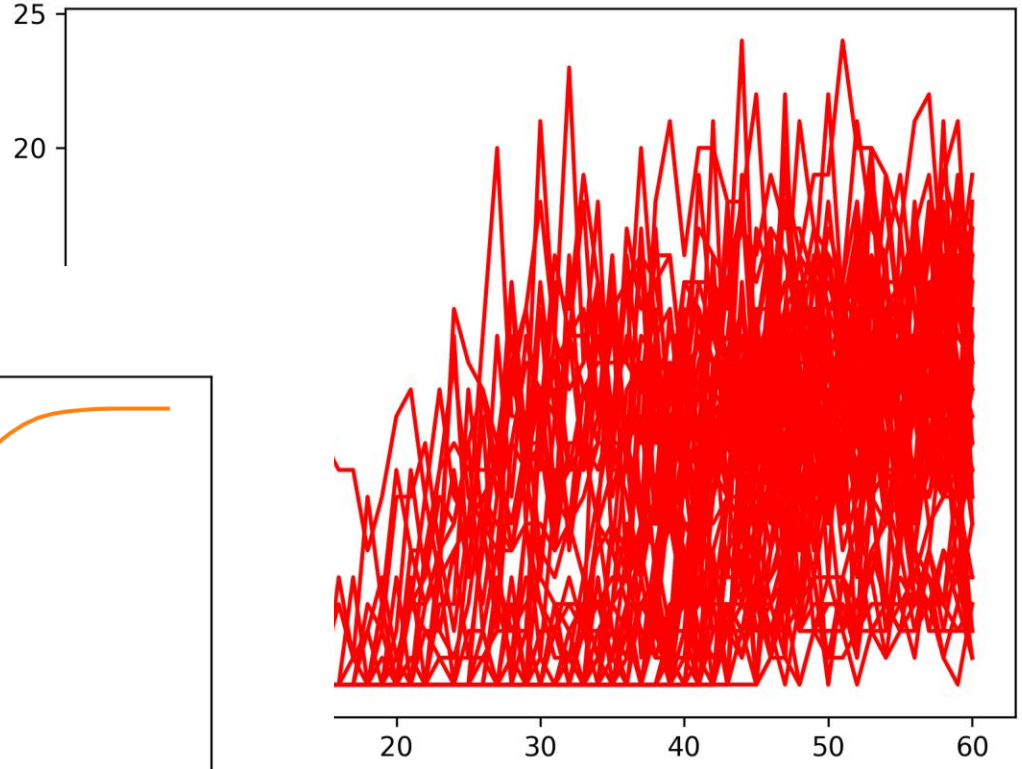




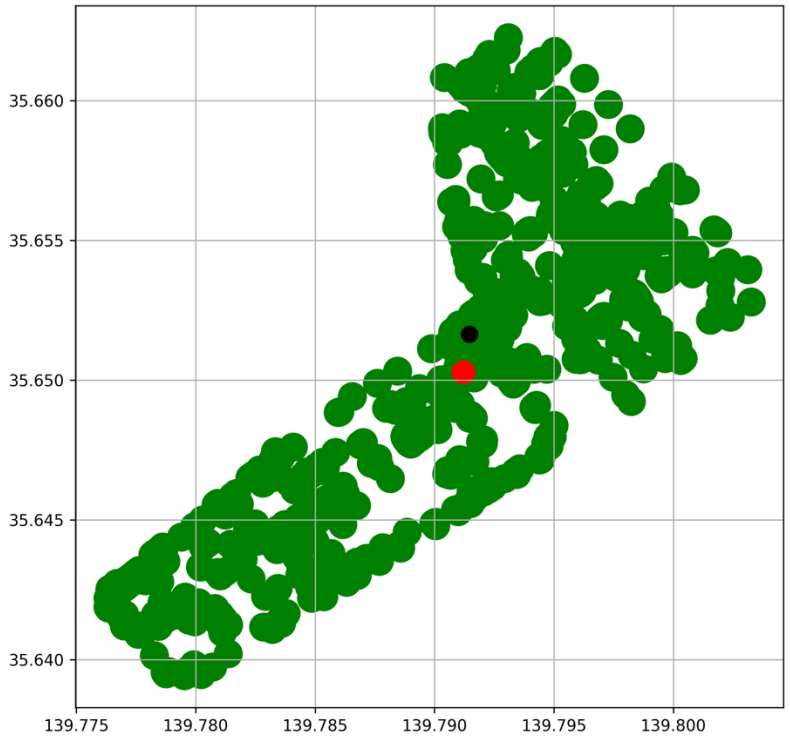
グラフでシミュレーションする





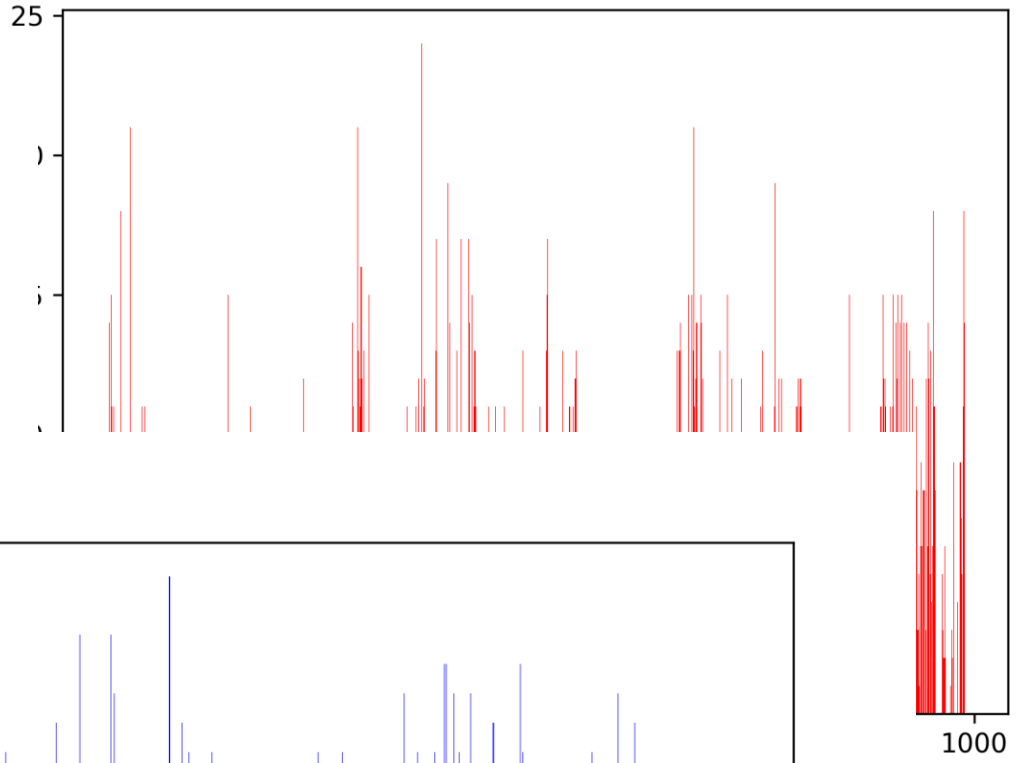


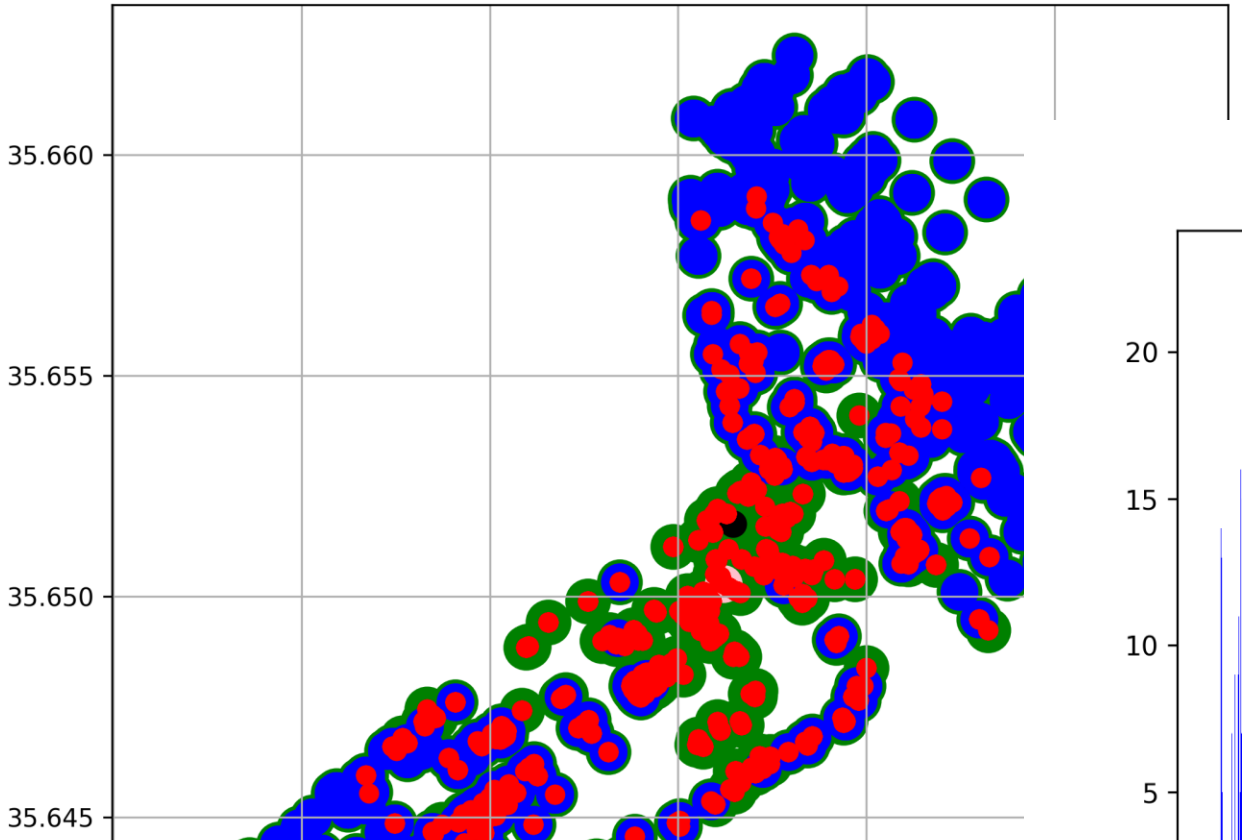
可視化グラフ

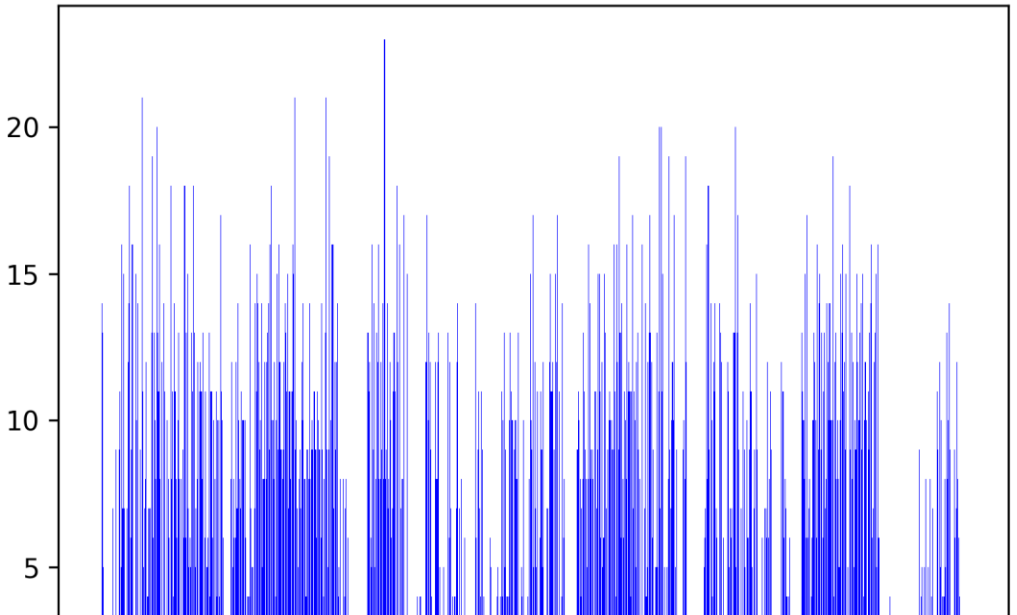

35.660
35.655
35.650
35.645
35.640
139.775
139.780
139.785
139.790
139.795
139.800

可視化30回タイムステップ

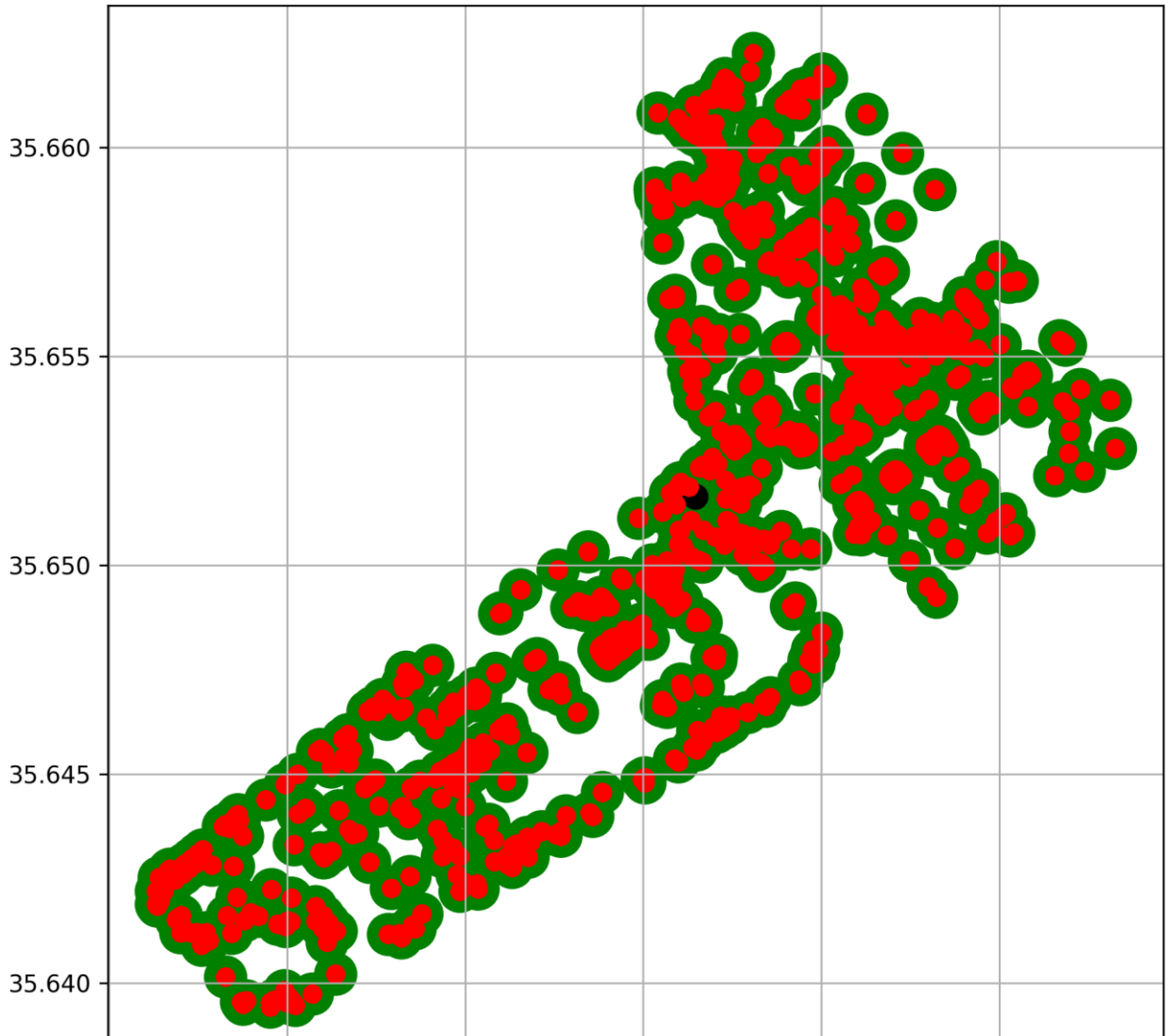
感染者



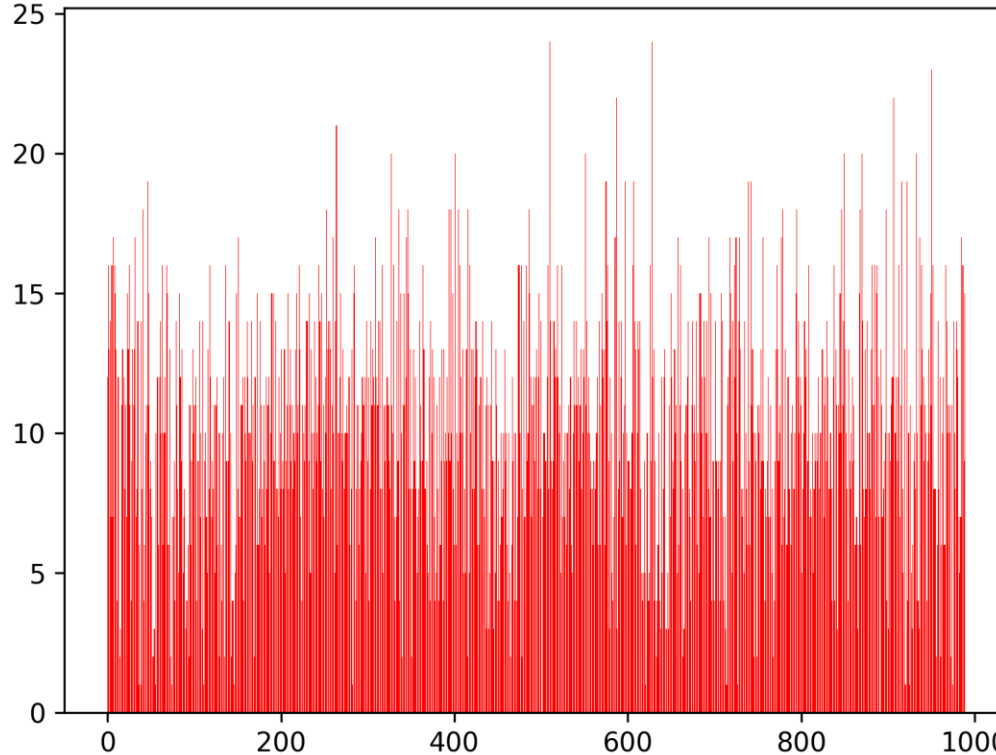




未感染者



感染者



考察

- 数式の結果のグラフを見ると遷移確率を固定しない場合感染速度が低い
- グラフから見ると感染は出発点から一番遠い遠いところに拡散するように見える
- 次はこれを一種の拡散と見て新しくモデルを作る方がいいかもしれない

問題点

- 数式だけ計算する時感染の速度がグラフのシミュレーションの感染速度が遅いに見える（同じ60回計算した結果)これは数式のモデルのパラメータ(遷移確率とdeltaのbeta)が同じものではないかつdeltaを計算する時の方法が違う両方の原因が考えられる
- 今回は移動ルート関連のものを計算しなかった
- tは0からnまで定義したが明白のものではないのでこのtは現実世界でどれだけの時間なのかはわからない
- 感染率が高く感じる
- この可視化方法は人数がわからない

まとめ

- 数値とグラフ上の両方のシミュレーションから見ると今回のモデルの感染速度は少しだけ速かった
- シミュレーションから見るとこれは一種の拡散に見える

今後の課題

- シミュレーション用のモデルを改良
- 移動ルートも考える
- 実データを使う