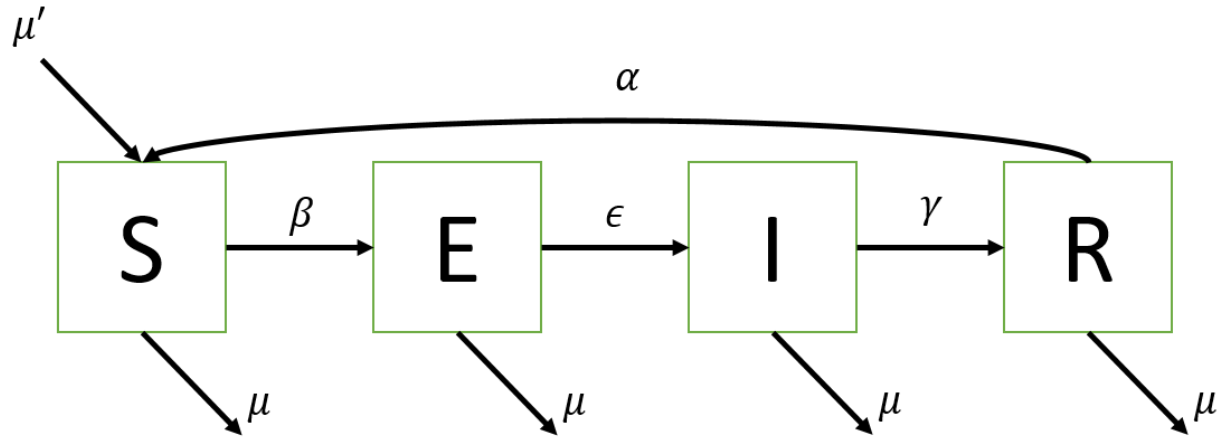
传染病模型实验报告

传染病模型是描述了不同类型的传染病在人群中传播的数学模型，在本次实验中，我实现了一个存在人口出生和死亡的 SEIRS 模型，并以这个最为复杂的模型导出其各个特例，包括 SI、SIS、SIR、SIRS 模型。

基本模型

存在再生产的 SEIRS 模型如下图所示：



Explanation of states:

- S: $susceptible$
- E: $exposed$
- I: $infected$
- R: $recovered$

Explanation of edges:

- μ: $death\ rate$
- μ': $birth\ rate$
- β: $transmisson\ rate$
- ϵ: $incubation\ rate$
- γ: $recover\ rate$
- α: $immunity\ loss\ rate$

如上伪状态机描述了种群中任意一个实体的状态转移，在某一时刻，一个实体必然拥有一个状态，并且有一定的概率转移至下一个状态，满足一个类马尔科夫状态机（但感病状态转移概率随群体状态改变而变化）。

四种实体状态分别表示当前实体处于易感病期、潜伏期、传染期、康复期。除了传染率β以外的所有转移参数均为独立的概率，而感病期实体被传染的概率还与传染期个体在种群中的比重有着线性关系。新出生的个体必然处于易感病期。

不过由于 Monte Carlo 方法的离散性以及死亡转移的特殊性（实体被移除），在我的仿真实现中，每一次的迭代时状态转移满足如下优先级：

$$Death = Birth > Others$$

即在一次迭代中，先做染病状态转移，再除去死亡个体，再加入出生个体。

另外，本模型还有如下超参数：

- $Population: initial\ number\ of\ entities$
- $Infected\ Share: initial\ share\ of\ infected\ entities$
- $Iteration: number\ of\ interation\ to\ simulate$

以及对α、ϵ两个参数的特殊处理：当这两个参数为1时，其连接的两个状态会被合并。

特例模型

由上文所述，SI、SIS、SIR、SIRS 模型均为 SEIRS 模型的特例，而且为了简单起见，我们往往忽略人口的再生产（即$\mu = \mu' = 0$）。

- SIRS 模型：$\epsilon = 1$情况下 SEIRS 模型的特例
  在这个模型里，所有感病实体在被传染后立即进入传染期。
- SIR 模型：$\epsilon = 1, \alpha = 0$情况下 SEIRS 模型的特例
  在这个模型里，所有康复后的实体将不再感病
- SIS 模型：$\epsilon = 1, \alpha = 1$情况下 SEIRS 模型的特例
  在这个模型里，所有康复后的实体仍为易感病的状态
- SI 模型：$\epsilon = 1, \gamma = 0$情况下 SEIRS 模型的特例（此时α的取值已不再有意义）
  在这个模型里，所有感病实体不会康复，即只能死亡

为了节约篇幅，这里不再给出这些特例模型的状态转移图，这些图都是如上 SEIRS 转移图的某一部分。

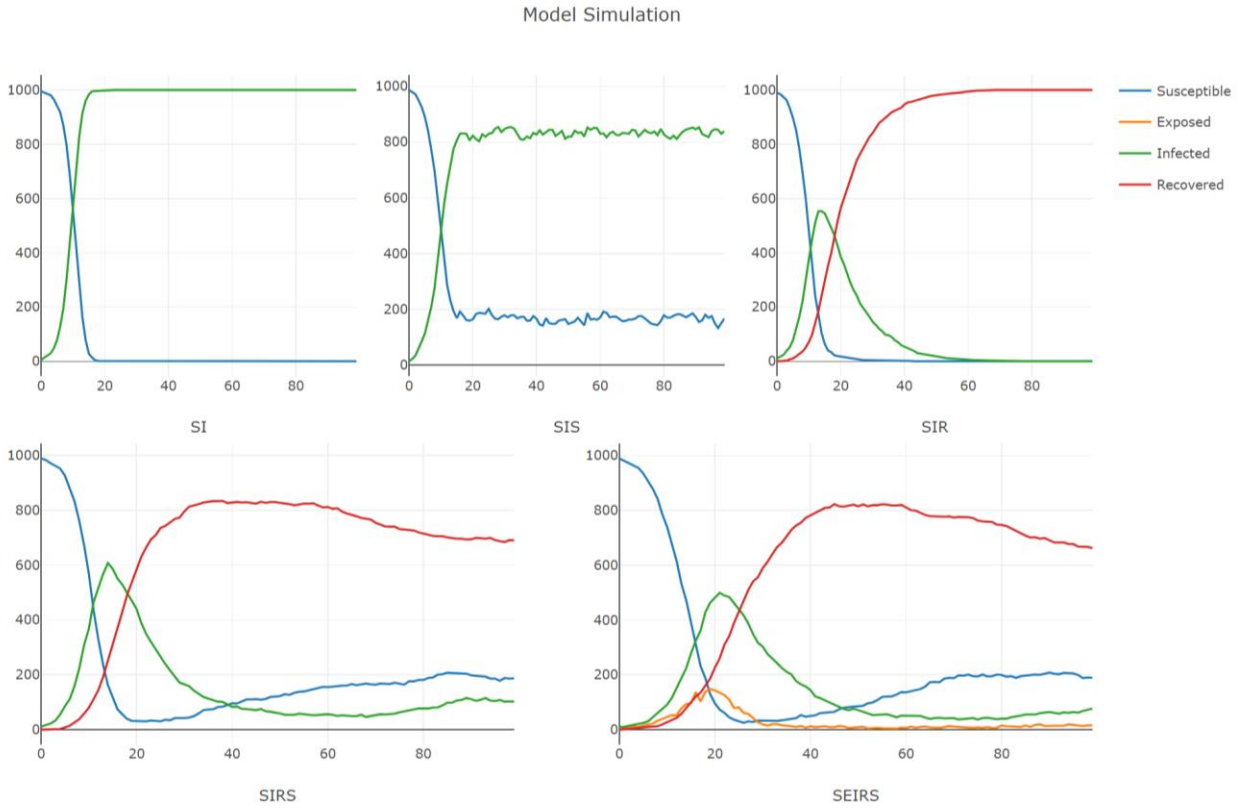
实验仿真

实验仿真中所有图的 x 轴均为迭代次数，y 轴均为人口。

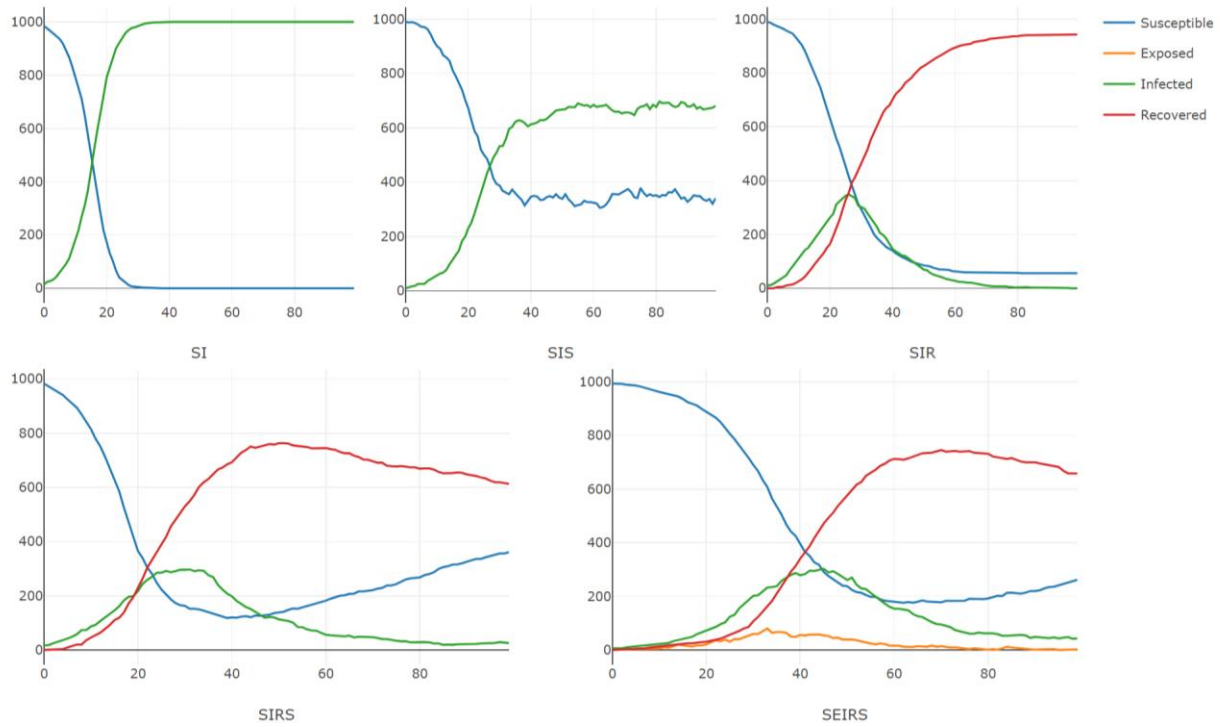
静态模型

静态模型即人口总量恒定，不存在人口的死亡和再生产。

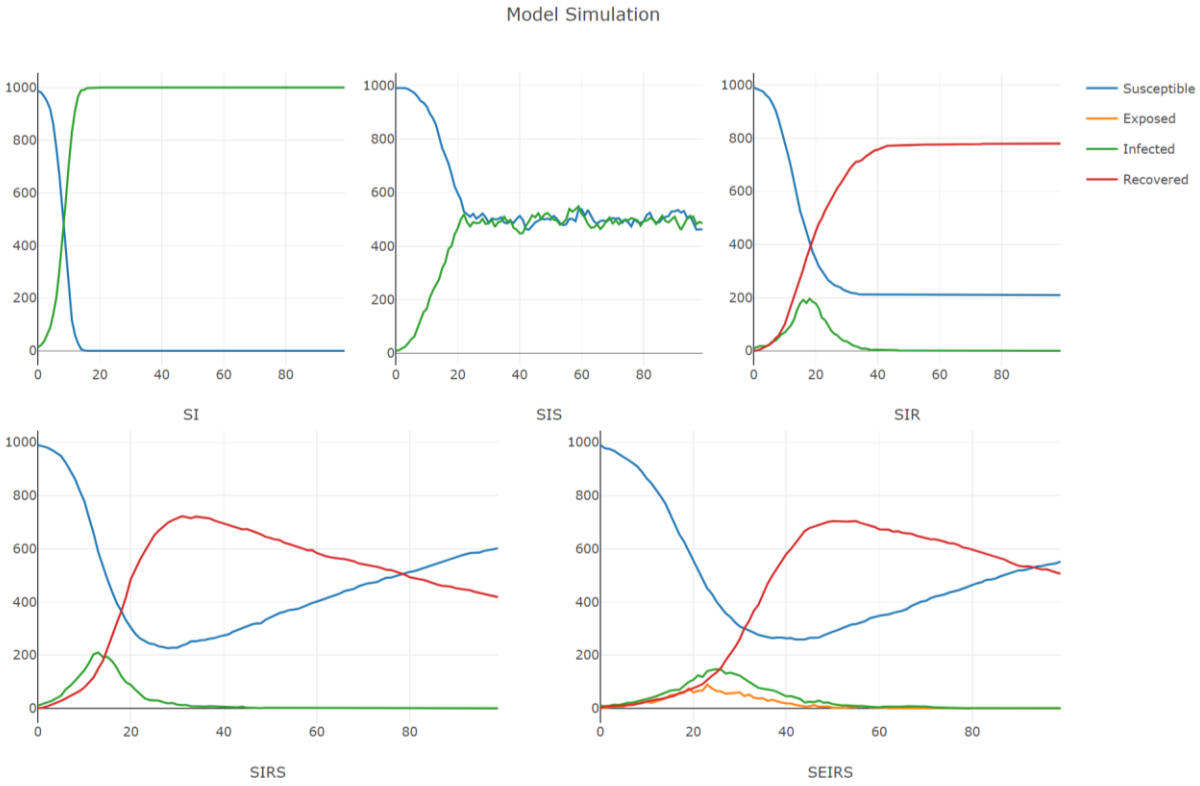
$$\beta = 0.6, \gamma = 0.1, \alpha = 0.01, \epsilon = 0.5$$



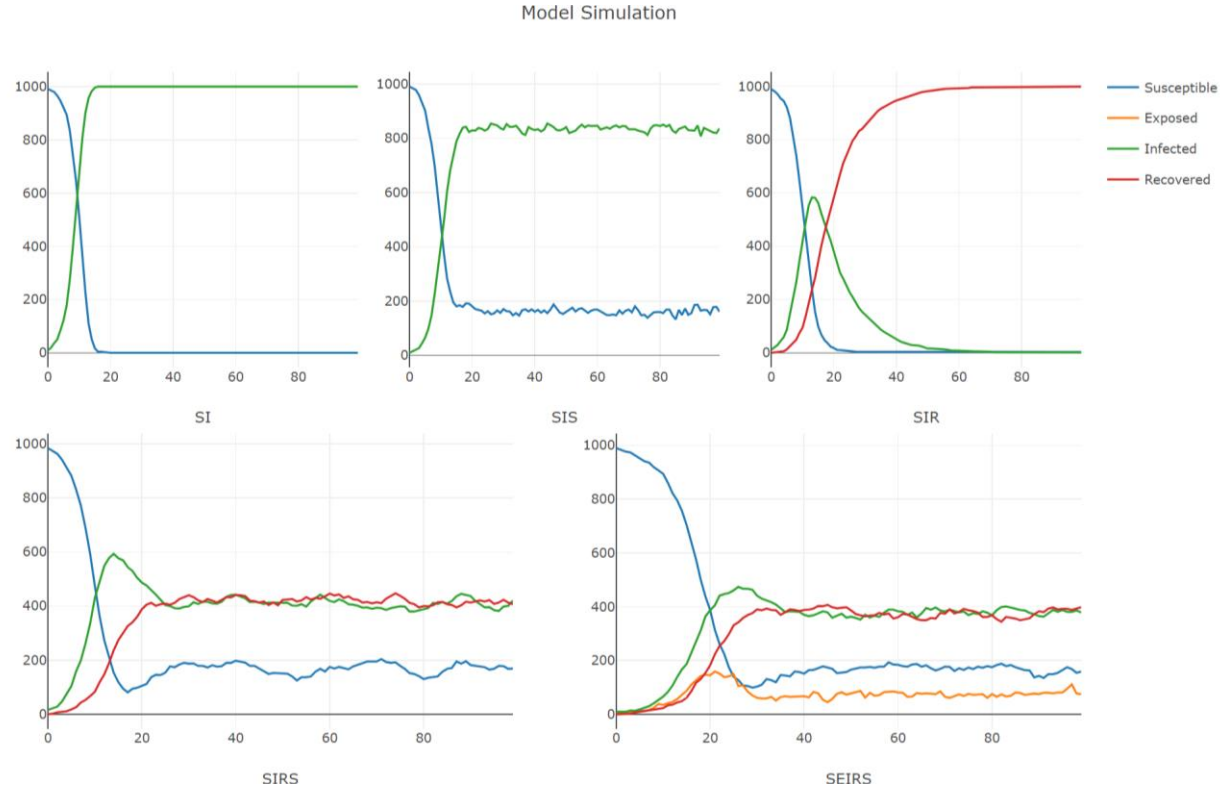
$$\beta = 0.3, \gamma = 0.1, \alpha = 0.01, \epsilon = 0.5$$



$$\beta = 0.6, \gamma = 0.3, \alpha = 0.01, \epsilon = 0.5$$

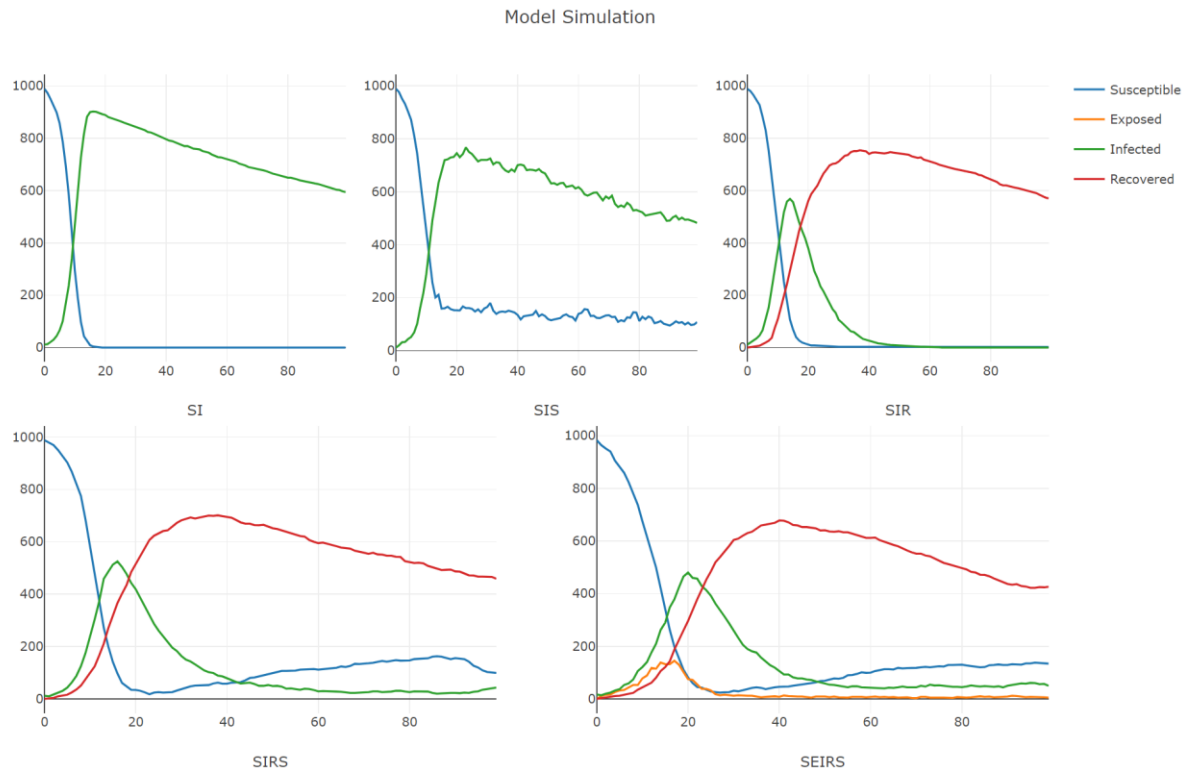


$$\beta = 0.6, \gamma = 0.3, \alpha = 0.1, \epsilon = 0.5$$

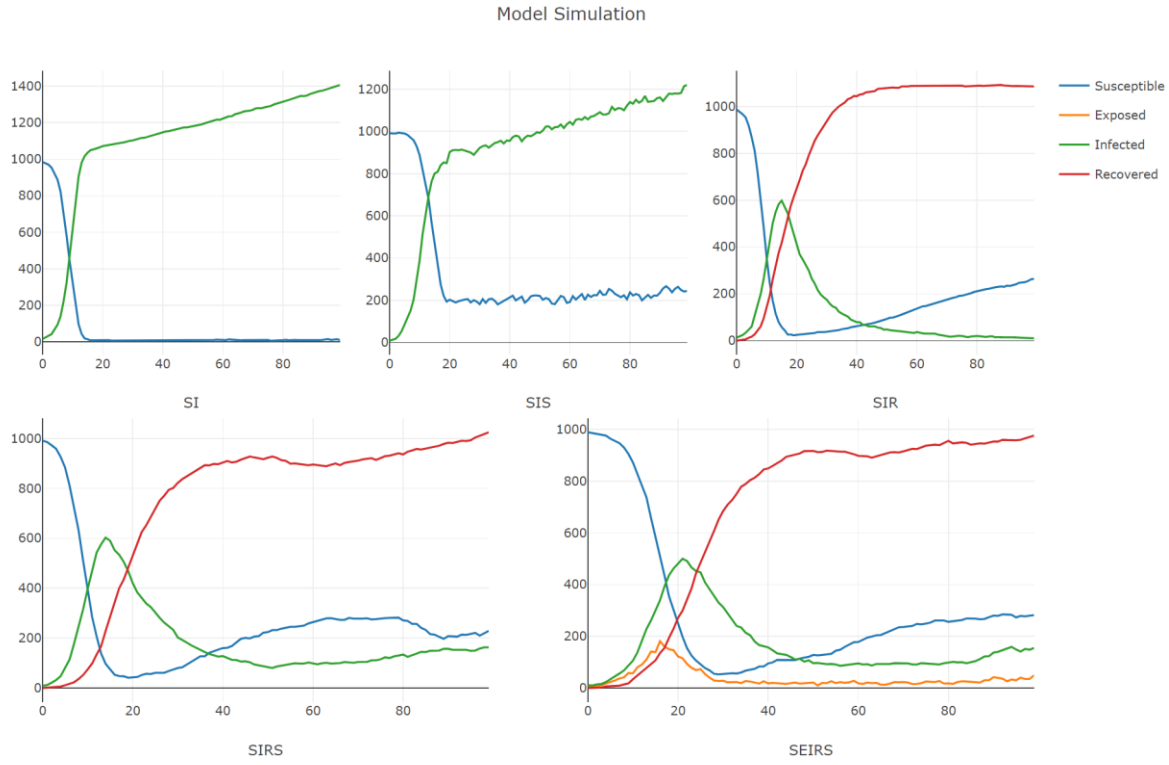


动态模型

$$\mu = 0.005, \mu' = 0.001$$



$$\mu = 0.001, \mu' = 0.005$$



仿真总结

由上节 Monte Carlo 仿真得到的曲线可知：

在静态模型中，所有模型在一定数量迭代后均会趋于稳定。

- SI 模型在一定数量的迭代后，易感病个体消灭，所有个体均处于传染状态，其消灭速度仅与β有关，呈现指数增长。
- SIS 模型中，由于感病个体有一定概率康复变回易感病个体，最终整个种群中两种个体的比例会达到一个动态平衡，比例分配由β和γ共同决定。
- SIR 模型中，由于状态转移的单向性，最终传染期的个体会全部消失。另外，当参数合适时，由于传染期个体消灭时种群还没有被完全传染，种群中会残留一部分易感病个体，其在总人口中所占的比例由β和γ共同决定。
- SIRS 模型是 SIS 与 SIR 模型的结合，由于从传染期到易感病期之间多了一个康复免疫期作为缓冲，传染期个体很容易被消灭，而又由于康复期个体会逐渐回到易感病状态，最终所有个体将均处于易感病的状态。
- SEIRS 模型的表现基本与 SIRS 模型一致，潜伏期作为感病期的缓冲状态，其个体数量的表现与传染期基本一致，但总数较少。

在动态模型中，由于人口的死亡与再生产速率和疾病传染速率相比往往可以忽略不计，其在系统稳定前对整体趋势的影响是微乎其微的，但当系统稳定后，出生率和死亡率将会决定系统的后续走向。

实现细节

本次实验的仿真我使用了 F#实现，实验代码依赖于 XPlot.Plotly（上文仿真图为生成数据后手动拼接成 Javascript 代码）

用户可以通过 Model.createStatic 或者 Model.createDynamic 创建模型实例，并通过 Simulation.reproduce 或 Simulation.sample 来得到一次蒙特卡洛仿真的样本。模型参数由 ModelCategory 类型指定。

```
open System
open XPlot.Plotly

// mu: death rate
// mu': birth rate
// beta: contact rate
// gamma: recover rate
// alpha: immunity loss rate
// epsilon: incubation rate
type ModelParameters =
    | ModelParameters of mu:float * mu':float * beta:float * gamma:float * alpha:float *
epsilon:float

type ModelCategory =
    | SI    of beta:float
    | SIS   of beta:float*gamma:float
    | SIR   of beta:float*gamma:float
```

```
    | SIRS  of beta:float*gamma:float*alpha:float
    | SEIRS of beta:float*gamma:float*alpha:float*epsilon:float

module Model =
    let createDynamic mu mu' category =
        let beta, gamma, alpha, epsilon =
            match category with
            | SI(b)             -> b, 0., 1., 1.
            | SIS(b, g)         -> b, g , 1., 1.
            | SIR(b, g)         -> b, g , 0., 1.
            | SIRS(b, g, a)     -> b, g , a , 1.
            | SEIRS(b, g, a, e) -> b, g , a , e

        ModelParameters(mu, mu', beta, gamma, alpha, epsilon)

    let createStatic =
        createDynamic 0. 0.

module Simulation =
    type EntityState =
        | Susceptible
        | Exposed
        | Infected
        | Recovered

    let reproduce infectShare population n model (rand: Random) =
        let mu, mu', beta, gamma, alpha, epsilon =
            match model with | ModelParameters(m, m', b, g, a, e) -> m, m', b, g, a, e

        // sample and test a probability
        let test prob =
            rand.NextDouble() < prob

        // iterate on a sample
        let transformSample (s, e, i, r) state =
            // total number of population
            let n = float (s+e+i+r)
            // next state of the sample
            match state with
            | Susceptible ->
                if test <| (beta * float i) / n
                then (if epsilon<1. then Exposed else Infected)
                else Susceptible
            | Exposed ->
                if test <| epsilon
                then Infected
                else Exposed
            | Infected ->
                if test <| gamma
                then (if alpha<1. then Recovered else Susceptible)
                else Infected
            | Recovered ->
                if test <| alpha
                then Susceptible
                else Recovered

        // returns (s, e, i, r)
        let computePopulation samples =
```

```
            samples
            |> List.countBy id
            |> List.map (fun (state, cnt) ->
                            match state with
                            | Susceptible -> (cnt, 0, 0, 0)
                            | Exposed     -> (0, cnt, 0, 0)
                            | Infected    -> (0, 0, cnt, 0)
                            | Recovered   -> (0, 0, 0, cnt)
                        )
            |> List.reduce (fun (s, e, i, r) (s', e', i', r') -> (s+s', e+e', i+i',
r+r'))

        // init model with initial probability of being infected
        let initModel prob n =
            List.init n (fun _ -> if test prob then Infected else Susceptible)

        // iterate model via Monte Carlo simulation
        let iterateModel population samples =
            let n = let (s, e, i, r) = population in float (s+e+i+r)

            samples
            // deal with disease transmission
            |> Seq.map (transformSample population)
            // deal with death
            |> Seq.filter (fun _ -> test mu |> not)
            // deal with birth
            |> Seq.append (initModel 0. (mu' * n |> int))
            // realize
            |> Seq.toList

        let result =
            initModel infectShare population
            |> Seq.unfold (fun samples ->
                                let population = computePopulation samples
                                Some (population, iterateModel population samples)
                            )
            |> Seq.take n

        let s = result |> Seq.map (fun (x, _, _, _) -> x)
        let e = result |> Seq.map (fun (_, x, _, _) -> x)
        let i = result |> Seq.map (fun (_, _, x, _) -> x)
        let r = result |> Seq.map (fun (_, _, _, x) -> x)

        s, e, i, r

    let sample infectShare population n model =
        reproduce infectShare population n model <| Random()


// Example Parameters: [mu=0.001; mu'=0.005; beta=0.6; gamma=0.1; alpha=0.01]
[<EntryPoint>]
let main argv =
    // hyper-parameters:
    let infectShare = 0.01
    let population = 1000
    let iteration = 100

    // parameters:
```

```
// respectively, transmission rate, recover rate, immunity loss rate, incubation rate
let beta, gamma, alpha, epsilon = 0.6, 0.1, 0.05, 0.5

let simulateModel (title, category) =
     // simulate
     let s, e, i, r =
         Model.createStatic category
         |> Simulation.sample infectShare population iteration

     // display
     [s; e; i; r]
     |> List.map (Seq.zip (Seq.initInfinite id))
     |> Chart.Line
     |> Chart.WithTitle title
     |> Chart.WithLabels ["Susceptible"; "Exposed"; "Infected"; "Recovered"]
     |> Chart.WithXTitle "Iteration"
     |> Chart.WithYTitle "Population"
     |> Chart.Show

[ "SI",    SI(beta);
  "SIS",   SIS(beta, gamma);
  "SIR",   SIR(beta, gamma);
  "SIRS",  SIRS(beta, gamma, alpha);
  "SEIRS", SEIRS(beta, gamma, alpha, epsilon)]
|> List.iter simulateModel

0
```

参考资料

- https://wiki.eclipse.org/Introduction_to_Compartment_Models
- https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology