Part 3: Modelling epidemics in continuous time and using stochastic processes

Darren Wilkinson

darrenjw.github.io

Abstracting model structure from simulation approach

The structure of an SIR model

We will use my library, `scala-smfsb`, associated with *my book*. SIR and SEIR models are included in the library. The library uses a *Petri net* approach to separate the representation of the structure of the model from the method we use to simulate its dynamics.

```
import smfsb._
import breeze.linalg._
import breeze.numerics._

val dMod = SpnModels.sir[IntState]()
// dMod: Spn[IntState] = UnmarkedSpn(
//   List("S", "I", "R"),
//   1  1  0
// 0  1  0  ,
//   0  2  0
// 0  0  1  ,
//   smfsb.SpnModels$$$Lambda$6775/2093194212@44a0b56e
// )
```

Simulation

We can feed a model into a simulation algorithm and get back a function for simulating from the dynamics of the process. We can then feed this function for simulating from the transition kernel of the process into a function that unfolds the dynamics into a time series of values.

```
val stepSIRds = Step.gillespie(dMod)
// stepSIRds: (IntState, Time, Time) => IntState = smfsb.S
val tsSIRds = Sim.ts(DenseVector(100,5,0), 0.0, 10.0,
    0.05, stepSIRds)
// tsSIRds: Ts[IntState] = List(
//   (0.0, DenseVector(100, 5, 0)),
//   (0.05, DenseVector(97, 8, 0)),
//   (0.1, DenseVector(92, 13, 0)),
//   (0.15000000000000002, DenseVector(80, 24, 1)),
//   (0.2, DenseVector(66, 37, 2)),
//   (0.25, DenseVector(49, 53, 3)),
//   (0.3, DenseVector(41, 59, 5)),
//   (0.35, DenseVector(30, 67, 8)),
```

Plot

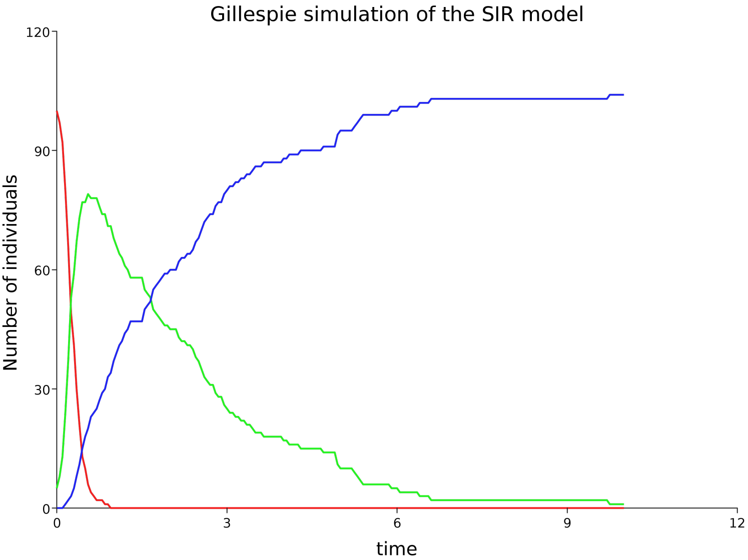

Figure 1:

Approximating the discrete stochastic dynamics

The *Gillespie algorithm* simulates every transition event explicitly. This leads to exact simulation of the underlying stochastic process, but can come at a high computational price.

```
val cMod = SpnModels.sir[DoubleState]()
// cMod: Spn[DoubleState] = UnmarkedSpn(
//   List("S", "I", "R"),
//   1  1  0
// 0  1  0  ,
//   0  2  0
// 0  0  1  ,
//   smfsb.SpnModels$$$Lambda$6775/2093194212@2c8f1e77
// )
val stepSIRcd = Step.euler(cMod)
// stepSIRcd: (DoubleState, Time, Time) => DoubleState = sm
```

SEIR

SEIR

```
val stepSEIR = Step.gillespie(SpnModels.seir[IntState]())
// stepSEIR: (IntState, Time, Time) => IntState = smfsb.St
val tsSEIR = Sim.ts(DenseVector(100,5,0,0), 0.0, 20.0, 0.05
// tsSEIR: Ts[IntState] = List(
//   (0.0, DenseVector(100, 5, 0, 0)),
//   (0.05, DenseVector(100, 5, 0, 0)),
//   (0.1, DenseVector(100, 5, 0, 0)),
//   (0.15000000000000002, DenseVector(100, 5, 0, 0)),
//   (0.2, DenseVector(100, 5, 0, 0)),
//   (0.25, DenseVector(100, 5, 0, 0)),
//   (0.3, DenseVector(100, 5, 0, 0)),
//   (0.35, DenseVector(100, 5, 0, 0)),
//   (0.39999999999999997, DenseVector(100, 5, 0, 0)),
//   (0.44999999999999996, DenseVector(100, 5, 0, 0)),
//   (0.49999999999999994, DenseVector(100, 5, 0, 0)),
//   (0.5499999999999999, DenseVector(100, 5, 0, 0)),
//   (0.6, DenseVector(100, 5, 0, 0)),
//   (0.65, DenseVector(100, 5, 0, 0)),
```

Spatial effects

SEIR as a reaction diffusion process