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$5355/612269312@44e1dbbf
// )
```

Simulation

We can feed a model into a simulation algorithm and get back a function (closure) 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 system states.

```
val stepSIRds = Step.gillespie(dMod)
// stepSIRds: (IntState, Time, Time) => IntState = smfsb.St

val tsSIRds = Sim.ts(DenseVector(100,5,0), 0.0, 10.0,
    0.05, stepSIRds)

import Sim.plotTs
plotTs(tsSIRds, "Gillespie simulation of the SIR model")
```

Plot (exact discrete stochastic dynamics)

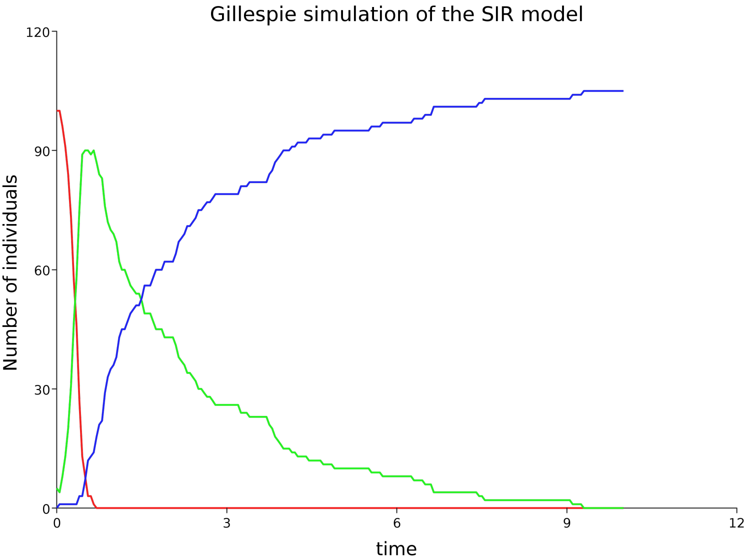

Figure 1:

Approximate simulation algorithms

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. If necessary, we can speed up simulation by discretising time, and using the *Poisson distribution* to advance the dynamics.

```
val stepSIRdsa = Step.pts(dMod)
val tsSIRdsa = Sim.ts(DenseVector(100,5,0), 0.0, 10.0,
    0.05, stepSIRdsa)
plotTs(tsSIRdsa, "Poisson simulation of the SIR model")
```

Plot (approximate discrete stochastic dynamics)

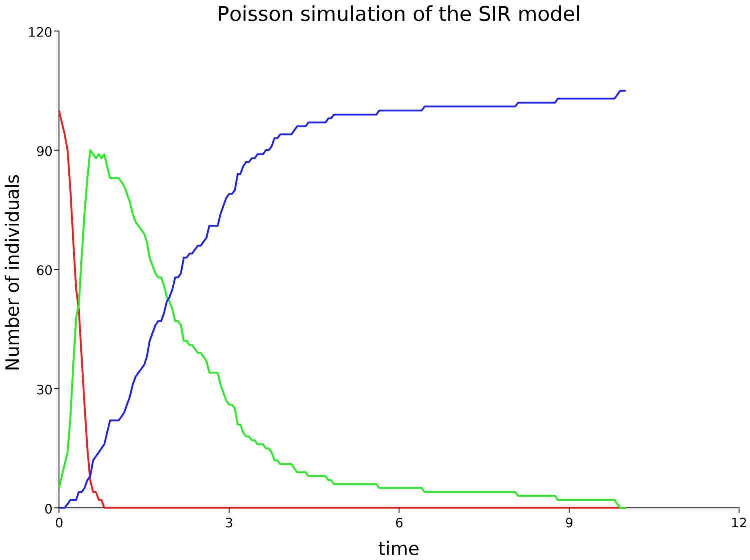

Figure 2:

Continuous state approximations

When dealing with very large populations and numbers of transition events, even the Poisson discretisation can become problematic. In this case, a continuous state approximation can be used which represents the process as a *stochastic differential equation* to be numerically integrated. For this, a continuous state instantiation of the SIR model must be used.

```
val cMod = SpnModels.sir[DoubleState]()
val stepSIRcs = Step.cle(cMod)
val tsSIRcs = Sim.ts(DenseVector(100.0,5.0,0.0),
    0.0, 10.0, 0.05, stepSIRcs)
plotTs(tsSIRcs, "Langevin simulation of the SIR model")
```

Plot (continuous stochastic dynamics)

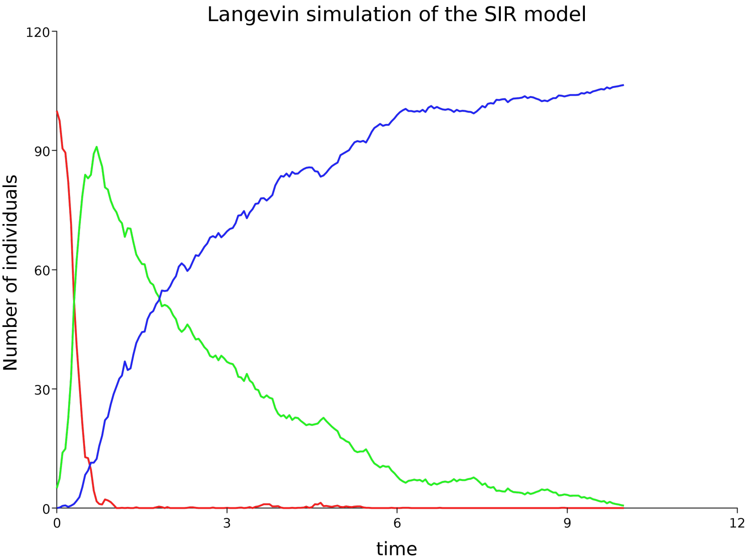

Figure 3:

Mass-action kinetics

If we aren't interested in stochastic effects, we can ignore the noise to get a representation of the model as a set of *ordinary differential equations* to be numerically integrated.

```
val stepSIRcd = Step.euler(cMod)
val tsSIRcd = Sim.ts(DenseVector(100.0,5.0,0.0),
    0.0, 10.0, 0.05, stepSIRcd)
plotTs(tsSIRcd,
    "Deterministic simulation of the SIR model")
```

Plot (continuous stochastic dynamics)

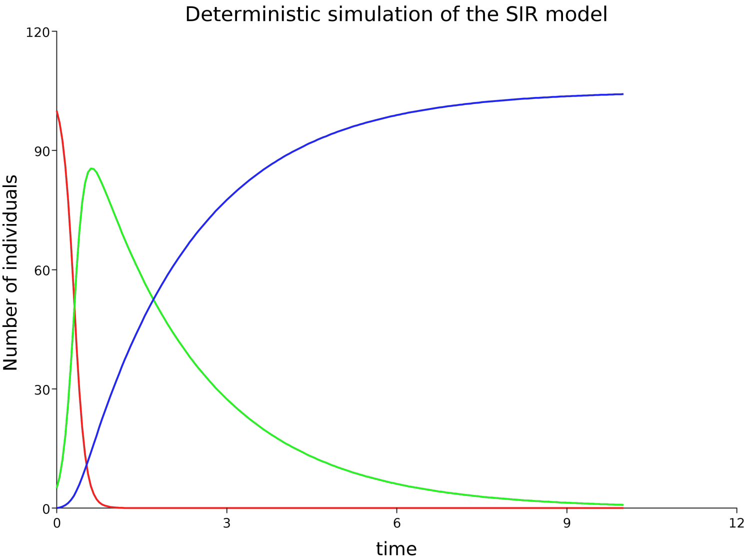

Figure 4:

Population modelling

Let's now see how to mimic the example we looked at in part 2 using discrete time deterministic kinetics. For this we need a model with appropriate parameters.

```
val p0 = DenseVector(1.0e7, 2.0, 0.0)
val cPop = SpnModels.sir[DoubleState](DenseVector(5.0e-8, 0
val stepPopcd = Step.euler(cPop)
val tsPopcd = Sim.ts(p0, 0.0, 100.0, 0.5, stepPopcd)
plotTs(tsPopcd,
    "Deterministic simulation of the SIR model")
```

Note that there isn't an exact match with the discrete time model we considered earlier, but that they are qualitatively very similar.

Plot (deterministic population dynamics)

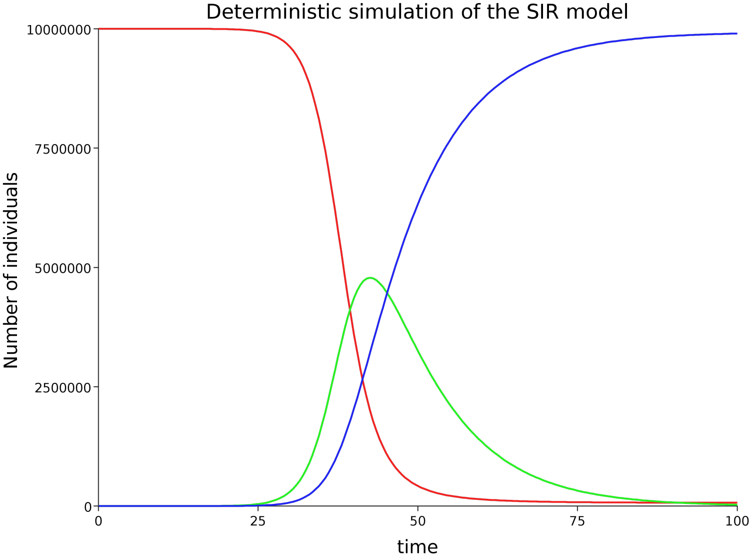

Figure 5:

Stochastic model

We can compare the ODE model with the SDE equivalent.

```
val stepPopcs = Step.cle(cPop)
val tsPopcs = Sim.ts(p0, 0.0, 100.0, 0.5, stepPopcs)
plotTs(tsPopcs,
    "Stochastic simulation of the SIR model")
```

Note that due to the very large number of individuals involved, laws of large numbers render stochastic effects imperceptible here.

Plot (stochastic population dynamics)

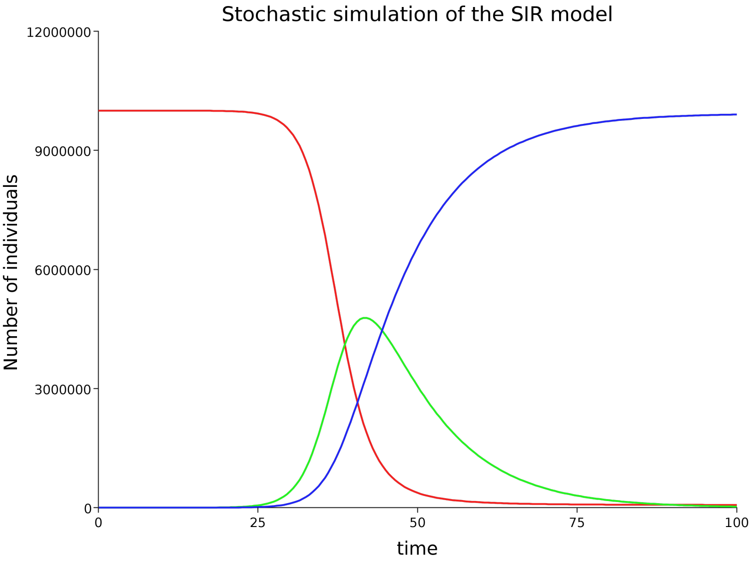

Figure 6:

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, stepSEIR)
// 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)),
```

Spatial effects

SEIR as a reaction diffusion process