Particle Filtering for Agent Based Models

Introduction

Some references for ABMs and inference:

- Ross et al. (2017), Lima et al. (2021), Christ et al. (2021), Rocha et al. (2021)

Some references for SMC:

- Dai et al. (n.d.), Endo, Van Leeuwen, and Baguelin (2019), Dahlin and Schön (n.d.), Svensson and Schön (n.d.)

Example

The Susceptible / Infected / Recovered (SIR) model has three parameters: one describing how infectious the pathogen is, one describing how much contact a host has with other hosts and one describing how quickly a host recovers.

$$
\begin{aligned}
\frac{dS}{dt} &= -\beta S \frac{I}{N} \\
\frac{dI}{dt} &= \beta S \frac{I}{N} - \gamma I \\
\frac{dR}{dt} &= \gamma I
\end{aligned}
$$

In order to estimate these parameters, we can assume that they come from prior distribution suggested by the literature and ideally then use a standard Markov Chain Monte Carlo (MCMC) technique to sample from the posterior. But with an Agent-Based Model (ABM), the likelihood is almost always intractable. We thus approximate the likelihood using particle filtering. The samples for the parameters that arise in this way are then drawn from the posterior and not just an approximation to the posterior.

In a nutshell, we draw the parameters from a proposal distribution and then run the particle filter to calculate the likelihood and compare likelihoods as in standard MCMC in order to run the chain.

Preliminary results have given a very good fit against observed data of an influenza outbreak in a boarding school in the UK.

A Deterministic Haskell Model

First let us set the necessary Haskell extensions and import the required modules.

```
!-- This is all an HTML comment.
```

```haskell
{-# LANGUAGE ScopedTypeVariables #-}
{-# LANGUAGE FlexibleContexts    #-}
{-# LANGUAGE OverloadedLists     #-}
{-# LANGUAGE NumDecimals         #-}
{-# LANGUAGE ViewPatterns        #-}
{-# LANGUAGE BangPatterns        #-}
{-# LANGUAGE QuasiQuotes         #-}
{-# LANGUAGE GeneralizedNewtypeDeriving #-}
```

```haskell
{-# OPTIONS_GHC -Wall              #-}
{-# OPTIONS_GHC -Wno-type-defaults #-}
```

```haskell
module Main (
main
) where
```

```haskell
import           Numeric.Sundials
import           Numeric.LinearAlgebra
import           Prelude hiding (putStr, writeFile)
import           Katip.Monadic
import qualified Data.Vector.Storable as VS
import           Data.List (transpose)
import           System.Random
import           System.Random.Stateful (IOGenM, newIOGenM)
```

```haskell
import           Data.Random.Distribution.Normal
import qualified Data.Random as R
```

```haskell
import           Control.Monad.IO.Class (MonadIO)
```

```haskell
import           Data.PMMH
import           Data.OdeSettings
import           Data.Chart
```

–>

Define the state and parameters for the model (FIXME: the infectivity rate and the contact rate are always used as $c\beta$ and are thus non-identifiable - I should probably just write the model with a single parameter $\alpha = c\beta$).

Extensions and imports for this Literate Haskell file

```haskell
data SirState = SirState {
    sirStateS :: Double
  , sirStateI :: Double
  , sirStateR :: Double
  } deriving (Eq, Show)

data SirParams = SirParams {
    sirParamsBeta  :: Double
  , sirParamsC     :: Double
  , sirParamsGamma :: Double
  } deriving (Eq, Show)

data Sir = Sir {
    sirS     :: SirState
  , sirP     :: SirParams
  } deriving (Eq, Show)
```

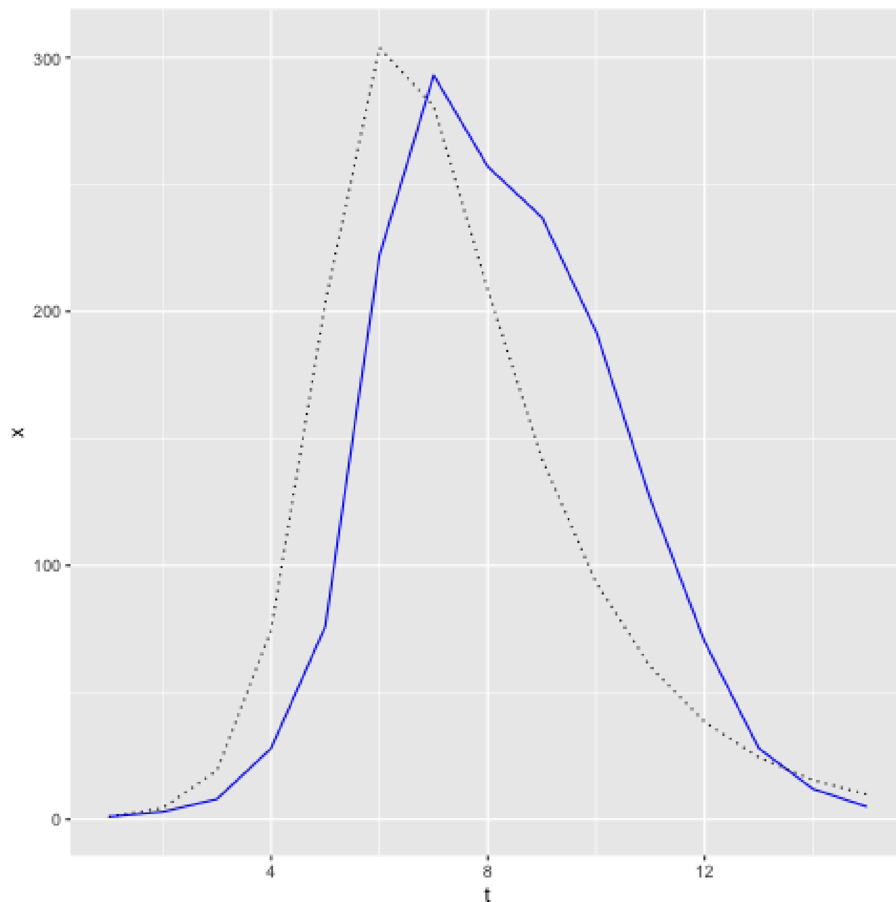
Define the actual ODE problem itself (FIXME: we can hide a lot more of the unnecessary details)

```haskell
sir :: Vector Double -> Sir -> OdeProblem
sir ts ps = emptyOdeProblem
  { odeRhs = odeRhsPure f
  , odeJacobian = Nothing
  , odeInitCond = [initS, initI, initR]
  , odeEventHandler = nilEventHandler
  , odeMaxEvents = 0
  , odeSolTimes = ts
  , odeTolerances = defaultTolerances
  }
  where
    f _ (VS.toList -> [s, i, r]) =
      let n = s + i + r in
        [ -beta * c * i / n * s
        , beta * c * i / n * s - gamma * i
        , gamma * i
        ]
    f _ _ = error $ "Incorrect number of parameters"

    beta  = realToFrac (sirParamsBeta  $ sirP ps)
    c     = realToFrac (sirParamsC     $ sirP ps)
    gamma = realToFrac (sirParamsGamma $ sirP ps)
    initS = realToFrac (sirStateS $ sirS ps)
    initI = realToFrac (sirStateI $ sirS ps)
    initR = realToFrac (sirStateR $ sirS ps)

sol :: MonadIO m =>
       (a -> b -> OdeProblem) -> b -> a -> m (Matrix Double)
```

```
sol s ps ts = do
  x <- runNoLoggingT $ solve (defaultOpts $ ARKMethod SDIRK_5_3_4) (s ts ps)
  case x of
    Left e  -> error $ show e
    Right y -> return (solutionMatrix y)
```

We can now run the model and compare its output to the actuals.

```
testSol :: IO ([Double])
testSol = do
  m <- sol sir (Sir (SirState 762 1 0) (SirParams 0.2 10.0 0.5)) (vector us)
  let n = tr m
  return $ toList (n!1)
```



FIXME: Sadly this does not work and I would rather write the draft first and then fight with BlogLiterately.

```
[ghci]
```

```
import Data.List
:t transpose
import Numeric.LinearAlgebra
:t vector
import Data.PMMH
:t pf
```

Generalising the Model

We see that e.g. on day our model predicts 93 students in the sick bay while in fact there are 192 students there. What we would like to do is to use the last observation to inform our prediction. First we have to generalise our model so that it can be influenced by the data by allowing the state to be a general distribution rather than a particular value. Once we have done this, we can, using particle filtering, approximate the conditional probability measure of the state given the observations prior to the state we wish to estimate. How this is done more precisely is given in the section "Whatever."

So here is the generalised model where we add noise to the state. N.B. the invariant that the sum of the susceptible, infected and recovered remains constant no longer holds.

```
topF :: R.StatefulGen a IO => SirParams -> a -> SirState -> IO SirState
topF ps gen qs = do
  m <- sol sir (Sir qs ps) [0.0, 1.0]
  newS <- R.sampleFrom gen (normal (log (m!1!0)) 0.1)
  newI <- R.sampleFrom gen (normal (log (m!1!1)) 0.1)
  newR <- R.sampleFrom gen (normal (log (m!1!2)) 0.1)
  return (SirState (exp newS) (exp newI) (exp newR))
```

Apparently the person recording the outbreak only kept records of how many students were sick on any given day. We create a type for the daily observation and a function to create this from the state. In this case the observation function is particularly simple.

```
newtype Observed = Observed { observed :: Double } deriving (Eq, Show, Num, Fractional)
```

```
topG :: SirState -> Observed
topG = Observed . sirStateI
```

Particle Filtering in Practice

Since the number of infected students under the ODE model is not a whole number, we can without too much embarassment make the assumption that the probability density function for the observations is normally distributed,

```haskell
topD :: Observed -> Observed -> Double
topD x y = R.logPdf (Normal (observed x) 0.1) (observed y)
```

Now we can define a function that takes the current set of particles, their weights and the loglikelihood (FIXME: of what?) runs the particle filter for one time step and returns the new set of particles, new weights, the updated loglikelihood and a predicted value for the number of inspections.

FIXME: Include code here

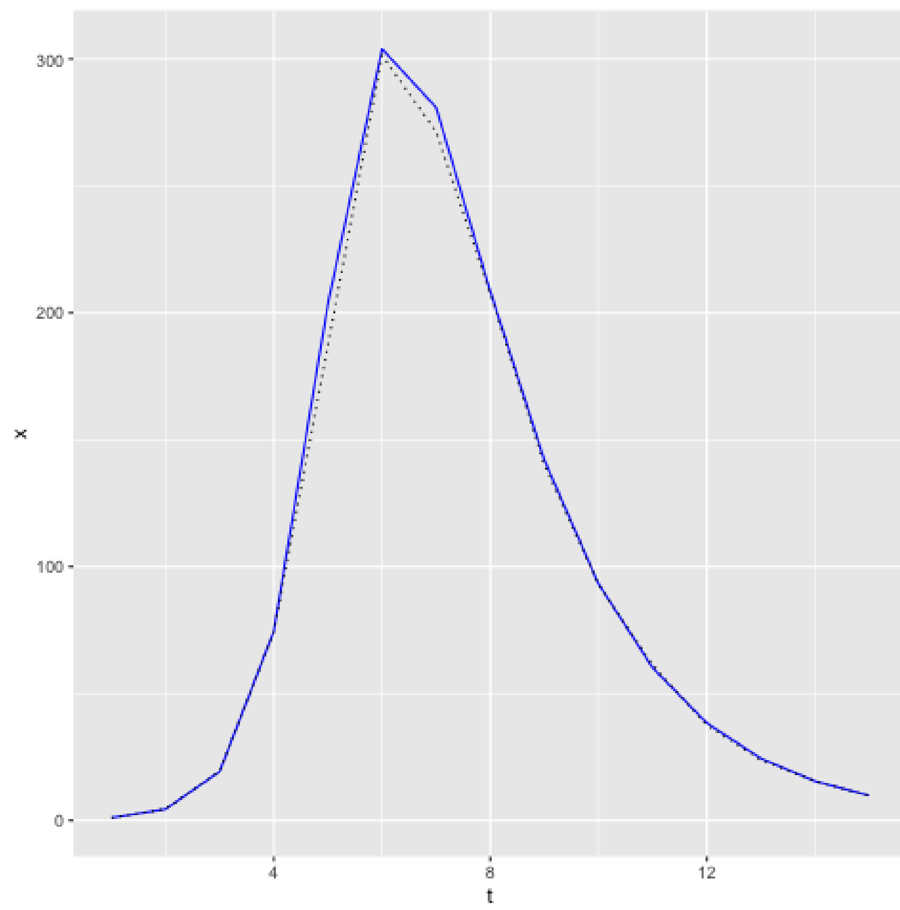
Further we can create some initial values and seed the random number generator (FIXME: I don't think this is really seeded).

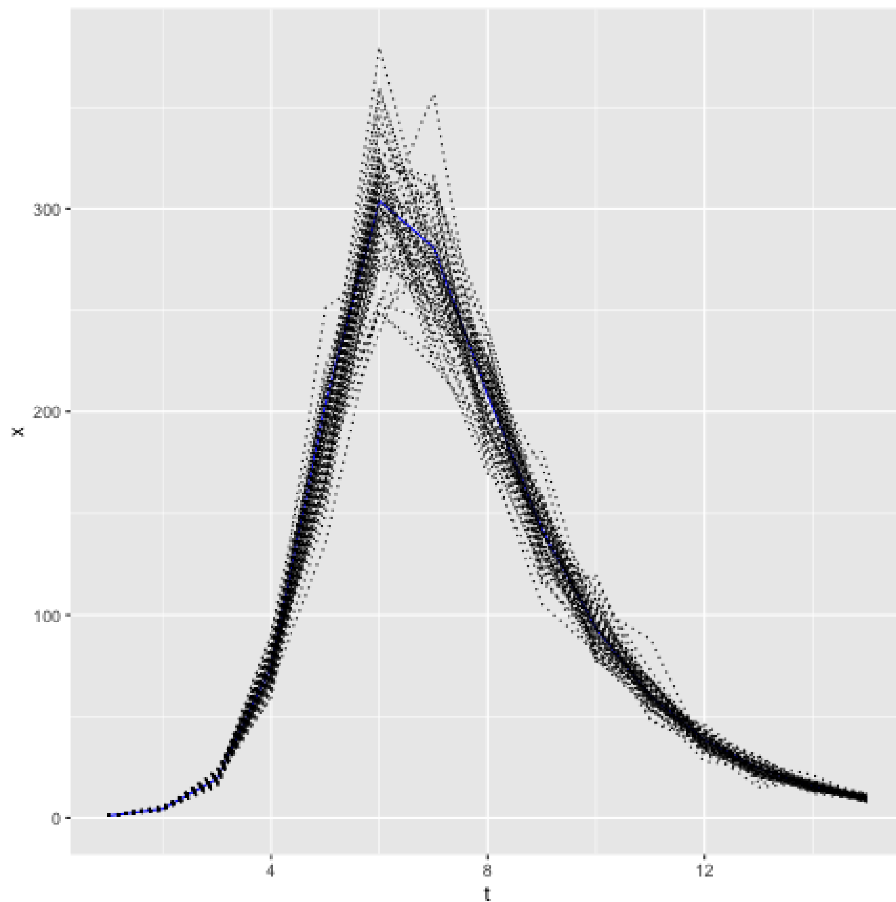
```haskell
nParticles :: Int
nParticles = 64

initParticles :: Particles SirState
initParticles = [SirState 762 1 0 | _ <- [1 .. nParticles]]

initWeights :: Particles Double
initWeights = [ recip (fromIntegral nParticles) | _ <- [1 .. nParticles]]

newStdGenM :: IO (IOGenM StdGen)
newStdGenM = newIOGenM =<< newStdGen

us :: [Double]
us = map fromIntegral [1 .. length actuals]

actuals :: [Double]
actuals = [1, 3, 8, 28, 76, 222, 293, 257, 237, 192, 126, 70, 28, 12, 5]
```

We can finally run the model against the data and plot the results.

FIXME: Include code here

```haskell
main :: IO ()
main = do
  q <- testSol
  chart (zip us actuals) [q] "diagrams/modelActuals.png"
  setStdGen (mkStdGen 42)
  stdGen <- newStdGenM
  ps <- predicteds (g' stdGen (topF (SirParams 0.2 10.0 0.5)) topG topD) initParticles initW
  let qs :: [[Double]]
      qs = transpose $ map (map sirStateI) $ snd ps
  chart (zip us q) [map observed $ fst ps] "diagrams/predicteds.png"
  chart (zip us q) qs "diagrams/generateds.png"
```


300
200
100
0
x
4
8
12
t



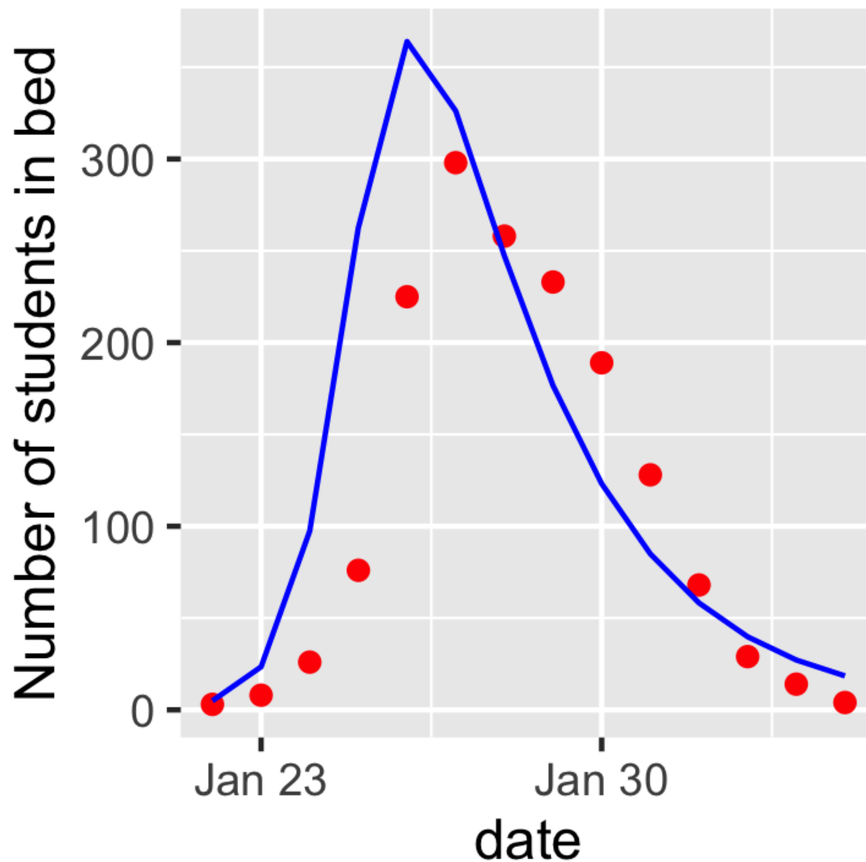
Estimating the Paramaters via MCMC

This is all find and dandy but we have assumed that we know the infection rate and recovery rate parameters. In reality we don't know these. We could use Markov Chain Monte Carlo (or Hamiltonian Monte Carlo) using the deterministic SIR model. FIXME: we could use the Stan example to draw some pictures but for Agent Based Models, this is rarely available.

```
functions {
  array[] real sir(real t, array[] real y, array[] real theta,
                   array[] real x_r, array[] int x_i) {
    real S = y[1];
    real I = y[2];
    real R = y[3];
    real N = x_i[1];
```

```
    real beta = theta[1];
    real gamma = theta[2];

    real dS_dt = -beta * I * S / N;
    real dI_dt = beta * I * S / N - gamma * I;
    real dR_dt = gamma * I;

    return {dS_dt, dI_dt, dR_dt};
  }
}
data {
  int<lower=1> n_days;
  array[3] real y0;
  real t0;
  array[n_days] real ts;
  int N;
  array[n_days] int cases;
}
transformed data {
  array[0] real x_r;
  array[1] int x_i = {N};
}
parameters {
  real<lower=0> gamma;
  real<lower=0> beta;
  real<lower=0> phi_inv;
}
transformed parameters {
  array[n_days, 3] real y;
  real phi = 1. / phi_inv;
  {
    array[2] real theta;
    theta[1] = beta;
    theta[2] = gamma;

    y = integrate_ode_rk45(sir, y0, t0, ts, theta, x_r, x_i);
  }
}
model {
  //priors
  beta ~ normal(2, 1);
  gamma ~ normal(0.4, 0.5);
  phi_inv ~ exponential(5);

  //sampling distribution
  //col(matrix x, int n) - The n-th column of matrix x. Here the number of infected people
```

```
  cases ~ neg_binomial_2(col(to_matrix(y), 2), phi);
}
generated quantities {
  real R0 = beta / gamma;
  real recovery_time = 1 / gamma;
  array[n_days] real pred_cases;
  pred_cases = neg_binomial_2_rng(col(to_matrix(y), 2), phi);
}
```



Markov Process and Chains

A **probability kernel** is a mapping $K : \mathbb{X} \times \mathcal{Y} \to \overline{\mathbb{R}}_+$ where $(\mathbb{X}, \mathcal{X})$ and $(\mathbb{Y}, \mathcal{Y})$ are two measurable spaces such that $K(s, \cdot)$ is a probability measure on $\mathcal{Y}$ for all $s \in \mathbb{X}$ and such that $K(\cdot, A)$ is a measurable function on $\mathbb{X}$ for all $A \in \mathcal{Y}$.

A sequence of random variables $X_{0:T}$ from $(\mathbb{X}, \mathcal{X})$ to $(\mathbb{X}, \mathcal{X})$ with joint distribution

given by

$$\mathbb{P}_T(X_{0:T} \in \mathrm{d}x_{0:T}) = \mathbb{P}_0(\mathrm{d}x_0) \prod_{s=1}^{T} K_s(x_{s-1}, \mathrm{d}x_s)$$

where K_t are a sequence of probability kernels is called a (discrete-time) **Markov process**. The measure so given is a path measure.

Note that, e.g.,

$$\mathbb{P}_1((X_0, X_1) \in A_0 \times A_1) = \int_{A_0 \times A_1} \mathbb{P}_0(\mathrm{d}x_0) K_1(x_0, \mathrm{d}x_1)$$

It can be shown that

$$\mathbb{P}_T(X_t \in \mathrm{d}x_t \,|\, X_{0:t-1} = x_{0:t-1}) = \mathbb{P}_T(X_t \in \mathrm{d}x_t, X_{t-1} = x_{t-1}) = K_t(x_{t-1}, \mathrm{d}x_t)$$

and this is often used as the defintion of a (discrete-time) Markov Process.

Let $(\mathbb{X}, \mathcal{X})$ and $(\mathbb{Y}, \mathcal{Y})$ be two measure (actually Polish) spaces. We define a hidden Markov model as a $(\mathbb{X} \times \mathbb{Y}, X \otimes \mathcal{Y})$-measurable Markov process $(X_n, Y_n)_{n \geq 0}$ whose joint distribution is given by

$$\mathbb{P}_T(X_{0:T} \in \mathrm{d}x_{0:T}, Y_{0:T} \in \mathrm{d}y_{0:T}) = \mathbb{P}_0(\mathrm{d}x_0) F_s(x_0, \mathrm{d}y_0) \prod_{s=1}^{T} K_s(x_{s-1}, \mathrm{d}x_s) F_s(x_s, \mathrm{d}y_s)$$

Writing $\mathbb{Q}_0(\mathrm{d}x_0, \mathrm{d}y_0) = \mathbb{P}_0(\mathrm{d}x_0) F_0(x_0, \mathrm{d}y_0)$ and $L_t((x_{t-1}, y_{t-1}), (\mathrm{d}x_t, \mathrm{d}y_t)) = K_t(x_{t-1}, \mathrm{d}x_t) F_t(x_t, \mathrm{d}y_t)$ we see that this is really is a Markov process:

$$\mathbb{P}_T(X_{0:T} \in \mathrm{d}x_{0:T}, Y_{0:T} \in \mathrm{d}y_{0:T}) = \mathbb{P}_0(\mathrm{d}x_0) F_0(x_0, \mathrm{d}y_0) \prod_{s=1}^{T} K_s(x_{s-1}, \mathrm{d}x_s) F_s(x_s, \mathrm{d}y_s) = \mathbb{Q}_0(\mathrm{d}x_0, \mathrm{d}y_0) \prod_{s=1}^{T} L_s((x_{s-1}$$

We make the usual assumption that

$$F_t(x_t, \mathrm{d}y_t) = f_t(x_t, y_t) \nu(\mathrm{d}y)$$

We can marginalise out $X_{0:T}$:

$$\mathbb{P}_T(Y_{0:t} \in \mathrm{d}y_{0:t}) = \mathbb{E}_{\mathbb{P}_t}\left[\prod_{s=0}^{t} f_s(X_s, y_s)\right] \prod_{s=0}^{t} \nu(\mathrm{d}y_s)$$

And writing

$$p_T(y_{0:t}) = p_t(y_{0:t}) = \mathbb{E}_{\mathbb{P}_t}\left[\prod_{s=0}^{t} f_s(X_s, y_s)\right]$$

We can write

$$\mathbb{P}_t(X_{0:t} \in \mathrm{d}x_{0:t} \,|\, Y_{0:t} = y_{0:t}) = \frac{1}{p_t(y_{0:t})}\left[\prod_{s=0}^{t} f(x_s, y_s)\right]\mathbb{P}_t(\mathrm{d}_{0:t})$$

We can generalise this. Let us start by with a Markov process

$$\mathbb{M}_T(X_{0:T} \in \mathrm{d}x_{0:T}) = \mathbb{M}_0(\mathrm{d}x_0)\prod_{s=1}^{T} M_s(x_{s-1}, \mathrm{d}x_s)$$

and then assume that we are given a sequence of potential functions (the nomenclature appears to come from statistical physics) $G_0 : \mathcal{X} \to \mathbb{R}^+$ and $G_t : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$ for $1 \leq t \leq T$. Then a sequence of Feynman-Kac models is given by a change of measure (FIXME: not even mentioned so far) from $\mathbb{M}_t$:

$$\mathbb{Q}_t(\mathrm{d}x_{0:t}) \triangleq \frac{1}{L_t} G_0(x_0)\left[\prod_{s=1}^{t} G_s(x_{s-1}, x_s)\right]\mathbb{M}_t(\mathrm{d}x_{0:t})$$

(N.B. we don't yet know this is a Markov measure - have we even defined a Markov measure?)

where

$$L_t = \int_{\mathcal{X}^{t+1}} G_0(x_0)\left[\prod_{s=1}^{t} G_s(x_{s-1}, x_s)\right]\mathbb{M}_t(\mathrm{d}x_{0:t})$$

With some manipulation we can write these recursively (FIXME: we had better check this).

Extension (of the path) which we will use to derive the predictive step:

$$\mathbb{Q}_{t-1}\left(\,\mathrm{d}x_{t-1:t}\right) = \mathbb{Q}_{t-1}\left(\,\mathrm{d}x_{t-1}\right) M_t\left(x_{t-1},\,\mathrm{d}x_t\right)$$

and the change of measure step which we will use to derive the correction step:

$$\mathbb{Q}_t\left(\,\mathrm{d}x_{t-1:t}\right) = \frac{1}{\ell_t} G_t\left(x_{t-1}, x_t\right)\mathbb{Q}_{t-1}\left(\,\mathrm{d}x_{t-1:t}\right)$$

where

$$\ell_0 = L_0 = \int_{\mathcal{X}} G_0\left(x_0\right) M_0\left(\mathrm{~d} x_0\right)$$

and

$$\ell_t = \frac{L_t}{L_{t-1}} = \int_{\mathcal{X}^2} G_t\left(x_{t-1}, x_t\right) \mathbb{Q}_{t-1}\left(\mathrm{~d} x_{t-1: t}\right)$$

for $t \geq 1$.

For a concrete example (bootstrap Feynman-Kac), we can take

$$\begin{aligned}
&\mathbb{M}_0\left(\mathrm{~d} x_0\right)=\mathbb{P}_0\left(\mathrm{~d} x_0\right), \quad G_0\left(x_0\right)=f_0\left(x_0, y_0\right) \\
&M_t\left(x_{t-1}, \mathrm{~d} x_t\right)=K_t\left(x_{t-1}, \mathrm{~d} x_t\right), \quad G_t\left(x_{t-1}, x_t\right)=f_t\left(x_t, y_t\right)
\end{aligned}$$

Then using extension and marginalising we have

$$\mathbb{P}_{t-1}\left(X_t \in \mathrm{d} x_t \mid Y_{0: t-1}=y_{0: t-1}\right)=\int_{x_{t-1} \in \mathcal{X}} K_t\left(x_{t-1}, \mathrm{~d} x_t\right) \mathbb{P}_t\left(X_{t-1} \in \mathrm{d} x_{t-1} \mid Y_{0: t-1}=y_{0: t-1}\right)$$

And using change of measure and marginalising we have

$$\mathbb{P}_t\left(X_t \in \mathrm{d} x_t \mid Y_{0: t-1}=y_{0: t-1}\right)=\frac{1}{\ell_t} f_t\left(x_t, y_t\right) \mathbb{P}_{t-1}\left(X_t \in \mathrm{d} x_t \mid Y_{0: t-1}=y_{0: t-1}\right)$$

If we define an operator P on measures as:

$$\mathrm{P} \rho \triangleq \int \rho(\mathrm{d} x) K\left(x, \mathrm{d} x^{\prime}\right)$$

and an operator C_t as:

$$\mathrm{C}_t \rho \triangleq \frac{\rho(d x) f\left(x, y_t\right)}{\int \rho(d x) f\left(x, y_t\right)}$$

$$\pi_n \triangleq \mathbf{P}\left(X_n \in \cdot \mid Y_1, \ldots, Y_n\right)$$

$$\pi_{n-1} \stackrel{\text{prediction}}{\longrightarrow} \mathrm{P} \pi_{n-1} \stackrel{\text{correction}}{\longrightarrow} \pi_n=\mathrm{C}_n \mathrm{P} \pi_{n-1}$$

$$\hat{\pi}_{n-1} \stackrel{\text{prediction}}{\longrightarrow} \mathrm{P} \hat{\pi}_{n-1} \stackrel{\text{sampling}}{\longrightarrow} \mathrm{S}^N \mathrm{P} \hat{\pi}_{n-1} \stackrel{\text{correction}}{\longrightarrow} \hat{\pi}_n:=\mathrm{C}_n \mathrm{~S}^N \mathrm{P} \hat{\pi}_{n-1}$$

$$S^N \rho := \frac{1}{N}\sum_{i=1}^{N} \delta_{X(i)}, \quad X(1), \ldots, X(N) \text{ are i.i.d. samples with distribution } \rho$$

$$\sup_{|f|\leq 1} \mathbf{E}\, |\pi_n f - \hat{\pi}_n f| \leq \frac{C}{\sqrt{N}}$$

References

Christ, Andreas, Sølvsten Jørgensen, Atiyo Ghosh, and Marc Sturrock. 2021. "Efficient inference for agent-based models of real-world phenomena Author summary," 1–32.

Dahlin, Johan, and Thomas B Schön. n.d. "Getting Started with Particle Metropolis-Hastings for Inference in Nonlinear Dynamical Models." https://cran.r-project.org/package=pmhtutorial.

Dai, Chenguang, Jeremy Heng, Pierre E Jacob, and Nick Whiteley. n.d. "An invitation to sequential Monte Carlo samplers." https://arxiv.org/abs/2007.11936v1.

Endo, Akira, Edwin Van Leeuwen, and Marc Baguelin. 2019. "Introduction to particle Markov-chain Monte Carlo for disease dynamics modellers." https://doi.org/10.1016/j.epidem.2019.100363.

Lima, Ernesto A. B. F., Danial Faghihi, Russell Philley, Jianchen Yang, John Virostko, Caleb M. Phillips, and Thomas E. Yankeelov. 2021. *Bayesian calibration of a stochastic, multiscale agent-based model for predicting in vitro tumor growth.* Vol. 17. 11. https://doi.org/10.1371/journal.pcbi.1008845.

Rocha, Heber L., Inês Godet, Furkan Kurtoglu, John Metzcar, Kali Konstantinopoulos, Soumitra Bhoyar, Daniele M. Gilkes, and Paul Macklin. 2021. "A persistent invasive phenotype in post-hypoxic tumor cells is revealed by fate mapping and computational modeling." *iScience* 24 (9). https://doi.org/10.1016/j.isci.2021.102935.

Ross, Robert J. H., R. E. Baker, Andrew Parker, M. J. Ford, R. L. Mort, and C. A. Yates. 2017. "Using approximate Bayesian computation to quantify cell–cell adhesion parameters in a cell migratory process." *Npj Systems Biology and Applications* 3 (1): 1–9. https://doi.org/10.1038/s41540-017-0010-7.

Svensson, Andreas, and Thomas B Schön. n.d. "Comparing Two Recent Particle Filter Implementations of Bayesian System Identification." http://www.it.uu.se/katalog/andsv164.