Particle Filters

Dexter Barrows
November 10, 2015

1 Intro

Particle filters are similar to MCMC-based methods in that they attempt to draw samples from an approximation of the posterior distribution of model parameters θ given observed data D. Instead of constructing a Markov chain and approximating its stationary distribution, a cohort of "particles" are used to move through the data in an on-line (sequential) fashion with the cohort being culled of poorly-performing particles at each iteration. Further, the culled particles are replenished from surviving particles, in a sense setting up a process not dissimilar from Darwinian selection.

2 Formulation

Particle filters, also called Sequential Monte-Carlo (SMC) or bootstrap filters, feature similar core functionality as the venerable Kalman Filter. As the algorithm moves through the data (sequence of observations), a prediction-update cycle is used to simulate the evolution of the model M with different particular parameter selections, track how closely these predictions approximate the new observed value, and update the current cohort appropriately.

Two separate functions are used to simulate the evolution and observation processes. The "true" state evolution is specified by

$$X_{t+1} \sim f_1(X_t, \theta), \tag{1}$$

And the observation process by

$$Y_t \sim f_2(X_t, \theta). \tag{2}$$

Note that components of θ can contribute to both functions, but a typical formulation is to have some components contribute to $f_1(\cdot, \theta)$ and others to $f_2(\cdot, \theta)$.

The prediction part of the cycle utilises $f_1(\cdot, \theta)$ to update each particle's current state estimate to the next time step, while $f_2(\cdot, \theta)$ is used to evaluate a weighting w for each particle which will be used to determine how closely that particle is estimating the true underlying state of the system. Note that $f_2(\cdot, \theta)$ could be thought of as a probability of observing a piece of data y_t given the particle's current state estimate and parameter set, $P(y_t|X_t, \theta)$. Then, the new cohort of particles is drawn from the old cohort proportional to the weights. This process is repeated until the set of observations D is exhausted.

3 Algorithm

Now we can formalize the particle filter.

We will denote each particle $p^{(j)}$ as the j^{th} particle consisting of a state estimate at time t, $X_t^{(j)}$, a parameter set $\theta^{(j)}$, and a weight $w^{(j)}$. Note that the state estimates will evolve with the system as the cohort traverses the data.

Algorithm 1: SIR particle filter

```
  /* Select a starting point                                          */
  Input  : Observations $D = y_1, y_2, ..., y_T$, initial particle distribution $P_0$ of size $J$
  /* Setup                                                            */
1 Initialize particle cohort by sampling $(p^{(1)}, p^{(2)}, ..., p^{(T)})$ from $P_0$
2 for $t = 1 : T$ do
      /* Evolve                                                       */
3     for $j$ = 1:$J$ do
4         $X_t^{(j)} \leftarrow f_1(X_{t-1}^{(j)}, \theta^{(j)})$
      /* Weight                                                       */
5     for $j$ = 1:$J$ do
6         $w^{(j)} \leftarrow P(y_t|X_t^{(j)}, \theta^{(j)}) = f_2(X_t^{(j)}, \theta^{(j)})$
      /* Normalize                                                    */
7     for $j$ = 1:$J$ do
8         $w^{(j)} \leftarrow w^{(j)} / \sum_1^J w^{(j)}$
      /* Resample                                                     */
9     $p^{(1:J)} \leftarrow \text{sample}(p^{(1:J)}, \text{prob} = w, \text{replace} = true)$
  /* Samples from approximated posterior distribution                 */
  Output: Cohort of posterior samples $(\theta^{(1)}, \theta^{(2)}, ..., \theta^{(J)})$
```

4 Iterated Filtering

5 Data Cloning

6 IF2

Algorithm 2: IF2

```
/* Select a starting point                                              */
Input   : Initialize $\theta^{(1)}$
1  for $i = 2 : N$ do
       /* Resample moments                                              */
2      for $i = 1 : n$ do
3          r(i) $\leftarrow \mathcal{N}(0, 1)$
       /* Leapfrog initialization                                       */
4      $\theta_0 \leftarrow \theta^{(i-1)}$
5      $r_0 \leftarrow r - \nabla U(\theta_0) \cdot \varepsilon/2$
       /* Leapfrog intermediate steps                                   */
6      for $j = 1 : L - 1$ do
7          $\theta_j \leftarrow \theta_{j-1} + M^{-1} r_{j-1} \cdot \varepsilon$
8          $r_j \leftarrow r_{j-1} - \nabla U(\theta_j) \cdot \varepsilon$
       /* Leapfrog last steps                                           */
9      $\theta^* \leftarrow \theta_{L-1} + M^{-1} r_{L-1} \cdot \varepsilon$
10     $r^* \leftarrow \nabla U(\theta_L) \cdot \varepsilon/2 - r_{L-1}$
       /* Evaluate acceptance ratio                                     */
11     $r = \exp\left[H(\theta^{(i-1)}, r) - H(\theta^*, r^*)\right]$
       /* Sample                                                        */
12     $u \sim \mathcal{U}(0, 1)$
       /* Step acceptance criterion                                     */
13     if $u < \min\{1, r\}$ then
14         $\theta^{(i)} = \theta^*$
15     else
16         $\theta^{(i)} = \theta^{(i-1)}$
/* Samples from approximated posterior distribution                     */
Output: Chain of samples $(\theta^{(1)}, \theta^{(2)}, ..., \theta^{(N)})$
```

7 Fitting an SIR Model to Synthetic Epidemic Data with IF2

Here we will examine a test case in which Hamiltonian MCMC will be used to fit a Susceptible-Infected-Removed (SIR) epidemic model to mock infectious count data.

The synthetic data was produced by taking the solution to a basic SIR ODE model, sampling it at regular intervals, and perturbing those values by adding in observation noise. The SIR model used was

$$\begin{aligned} \tfrac{dS}{dt} &= -\beta I S \\ \tfrac{dI}{dt} &= \beta I S - r I \\ \tfrac{dR}{dt} &= r I \end{aligned} \tag{3}$$

where S is the number of individuals susceptible to infection, I is the number of infectious individuals, R is the number of recovered individuals, $\beta = R_0 r/N$ is the force of infection, R_0 is the number of secondary cases per infected individual, r is the recovery rate, and N is the population size.

The solution to this system was obtained using the `ode()` function from the `deSolve` package. The required derivative array function in the format required by `ode()` was specified as

```
SIR ← function(Time, State, Pars) {

    with(as.list(c(State, Pars)), {

        B   ← R0*r/N     # calculate Beta
        BSI ← B*S*I      # save product
        rI  ← r*I        # save product

        dS = -BSI        # change in Susceptible people
        dI = BSI - rI    # change in Infected people
        dR = rI          # change in Removed (recovered people)

        return(list(c(dS, dI, dR)))

    })

}
```

The true parameter values were set to $R_0 = 3.0, r = 0.1, N = 500$ by

```
pars  ← c(R0  ← 3.0,  # new infected people per infected person
          r   ← 0.1,  # recovery rate
          N   ← 500)  # population size
```

The system was integrated over $[0, 100]$ with infected counts drawn at each integer time step. These timings were set using

```
T ← 100                              # total integration time
times ← seq(0, T, by = 1)            # times to draw solution values
```

The initial conditions were set to 5 infectious individuals, 495 people susceptible to infection, and no one had yet recovered from infection and been removed. These were set using

```
y_ini ← c(S = 495, I = 5, R = 0)    # initial conditions
```

The `ode()` function is called as

```
odeout ← ode(y_ini, times, SIR, pars)
```

where `odeout` is a $(T + 1) \times 4$ matrix where the rows correspond to solutions at the given times (the first row is the initial condition), and the columns correspond to the solution times and S-I-R counts at those times.

The observation error was taken to be $\varepsilon_{obs} \sim \mathcal{N}(0, \sigma)$, where individual values were drawn for each synthetic data point.

These “true” values were perturbed to mimic observation error by

```
set.seed(1001)  # set RNG seed for reproducibility
sigma ← 5       # observation error standard deviation
infec_counts_raw ← odeout[,3] + rnorm(101, 0, sigma)
infec_counts     ← ifelse(infec_counts_raw < 0, 0, infec_counts)
```

where the last two lines simply set negative observations (impossible) to 0.

Plotting the data using the `ggplot2` package by

```
plotdata ← data.frame(times=1:(T+1),true=odeout[,3],data=infec_counts)

g ← ggplot(plotdata, aes(times)) +
        geom_line(aes(y = true, colour = "True")) +
        geom_point(aes(y = data, colour = "Data")) +
        labs(x = "Time", y = "Infection count", color = "") +
        scale_color_brewer(palette="Paired") +
        theme(panel.background = element_rect(fill = "#F0F0F0"))

print(g)
```

we obtain

Appendices

A Full R code

This code will run all the indicated analysis and produce all plots.

B Full C++ code

Stan model code to be used with the preceding R code.