Inference for SDE models via Approximate Bayesian Computation distilled

Matthew Wong

2024-02-28

Introduction and main research problem:

Stochastic differential equations (SDEs) are a useful choice of models for various dynamic processes in many fields, such as neuronal modelling, financial mathematics, population dynamics, pharmacokinetics/pharmacodynamics and modelling of physiological and chemical dynamics. As such, literature on the estimation and inference on parameters of SDEs has been a popular topic over the last twenty years. From a Bayesian perspective, however, there have been computational challenges using the usual MCMC techniques for drawing from parameter posteriors from SDEs. The author of this paper presents a method using approximate Bayesian computation to circumvent some of these computational challenges.

To set up the problem in presented in the paper, let $X_{t,\psi}$ be the solution to a SDE, with t representing a discrete time space, and ψ the vector of unknown parameters which are to be estimated. However, this true process is typically unobservable, and we instead observe $y_i = f(X_i, \epsilon_i)$, where y_i can be thought of as the process of interest perturbed by some measurement error ϵ. This measurement error ϵ_i is assumed to be drawn i.i.d from some distribution with mean 0 and covariance matrix $\sigma^2 I$, where σ is to be estimated from the data. Then the full parameter vector to be estimated is $\theta =< \psi, \sigma >$.

Problems with the standard methods:

The difficulty in using the standard method of MCMC comes from the erratic nature of the posterior surface in many cases for SDEs. Quite often, the posterior surface is multimodal and sparce. In cases such as these, it is difficult for a standard MCMC algorithm to adequately explore the parameter space, as a large step size will result in infrequent movement along the posterior space, and a small step size will make it difficult to traverse the entire space. In both cases, the chain's convergence will be poor, leading to computational issues. In addition to this, when the explicit solution to the SDE is not available, such as in the case presented in this paper, where the true process is not observed, further complication is added to the inference of parameters.

Rather than a standard MCMC algorithm, the author of this paper first uses an augmented likelihood to deal with the measurement error. They then propose using an approximate Bayesian likelihood method to remove a true likelihood from the algorithm, and instead targeting an approximation of the posterior space.

Simplified version of the main result:

To begin, we define the kernel $q((\theta_{old}, x_{old}), (\theta_{new}, x_{new})) = u(\theta_{new}|\theta_{old})v(x_{new}|\theta_{new})$ in order to move from (θ_{old}, x_{old}) to (θ_{new}, x_{new}). Then, the standard Metropolis-Hastings acceptance probability will be

$$\alpha((\theta_{old}, x_{old}), (\theta_{new}, x_{new})) = min(1, \frac{\pi(\theta_{new})\pi(y|x_{new}, \theta_{new})u(\theta_{old}|\theta_{new})}{\pi(\theta_{old})\pi(y|x_{old}, \theta_{old})u(\theta_{new}|\theta_{old})})$$

Now, approximate Bayesian computation is used to replace $\pi(y|x, \theta)$ with some simulated values which are sufficiently "close" to the real data y. In this case, a summary statistic S is chosen, and the distance between the simulated data y_{sim} and y is defined to be

$$\rho_\delta = \frac{1}{\delta} K(\frac{|S(y_{sim}) - S(y)|}{\delta})$$

where K is a kernel smoother and δ is its corresponding bandwidth. Then, in the acceptance probability above, $\pi(y|x,\theta)$ is replaced with ρ_δ. In this case, δ is also a quantity which can examined a-posteri by its own chain as well.

Simplified algorithm:

To set up the algorithm, we first define the following. Let $y_i = X_i(t) + \epsilon_i$, where each ϵ_i is distributed as i.i.d $N(0, \sigma_\epsilon^2)$, and σ_ϵ^2 is a parameter to be estimated. Then, to define our SDE, let $dX(t) = \beta X(t)dt + \sigma X(t)dB(t)$, where β, σ are parameters to be estimated, and $B(t)$ is a Brownian motion process. Suppose then that we wish to find a solution to this SDE. We let $X(t) = f(t, B(t))$, and this will give a solution to the SDE as

$$dX(t) = (\frac{df}{dt} + \frac{1}{2}\frac{d^2f}{dx^2})dt + \frac{df}{dx}dB(t)$$

From here, we can match coefficients, and we know that

$$\beta f = \frac{df}{dt} + \frac{1}{2}\frac{d^2f}{dx^2}$$
$$\sigma f = \frac{df}{dx}$$

The second equation gives $f = e^{\sigma x + g(t)}$, and plugging this into the first equation gives $\beta f = g'(t)f + \frac{\sigma^2}{2}f$. Then we have that $g'(t) = \beta - \frac{\sigma^2}{2}$. Finally, putting all of this together gives

$$X(t) = x_0 e^{(\beta - \frac{\sigma^2}{2})t + \sigma B(t)}$$

as our solution.

To simplify this algorithm even further, we let $\sigma = 1$ to have one less parameter to estimate. For our choice of Brownian motion process, we use a univariate normal white noise process with mean 0 and variance 1. We also let $x_0 = 1$. For our candidate distributions, we use a symmetric uniform random walk for each of the parameters. For our choice of summary statistic, we use the sample mean. For our choice of kernel, we use a Gaussian kernel. Then from here, our algorithm is

Algorithm 1: Simplified algorithm

---

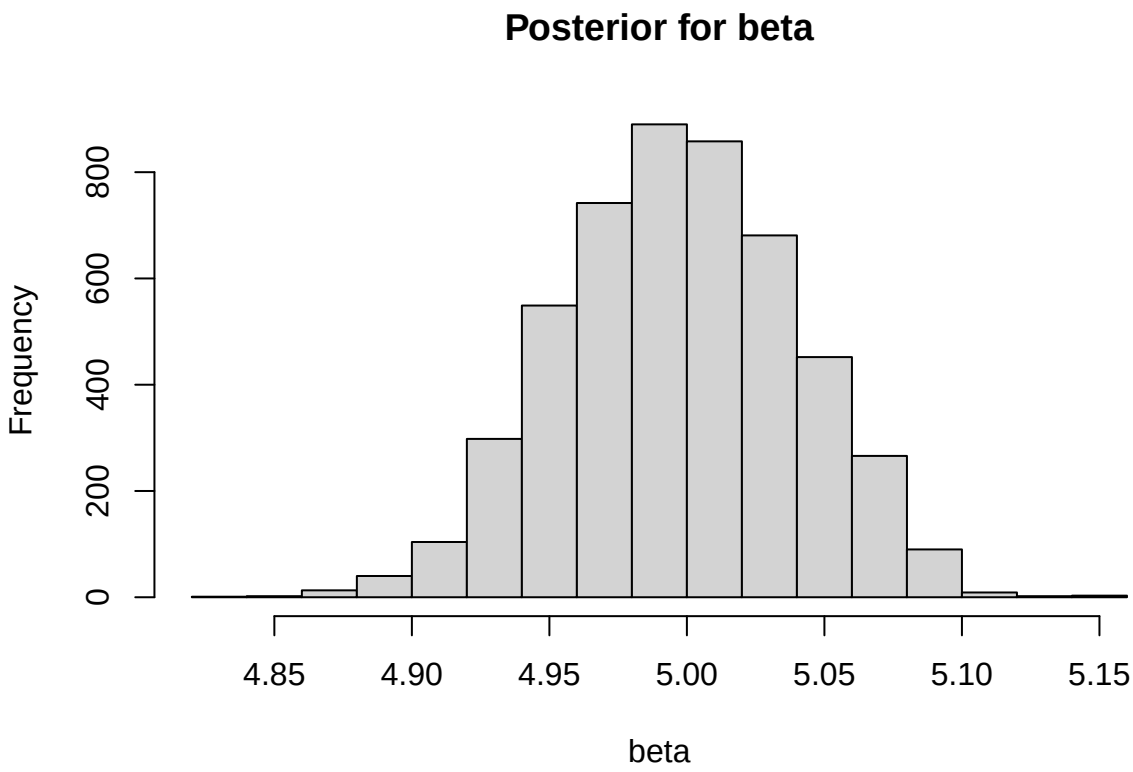
1. Initialize $\theta_{start} = (\beta_0, \sigma_0^2, \delta_0)$, and simulate $x_0 \sim e^{(\beta_0 - \frac{1}{2})t + W(t)}$ and $y_0 \sim x_0 + N(0, \sigma_0^2)$.

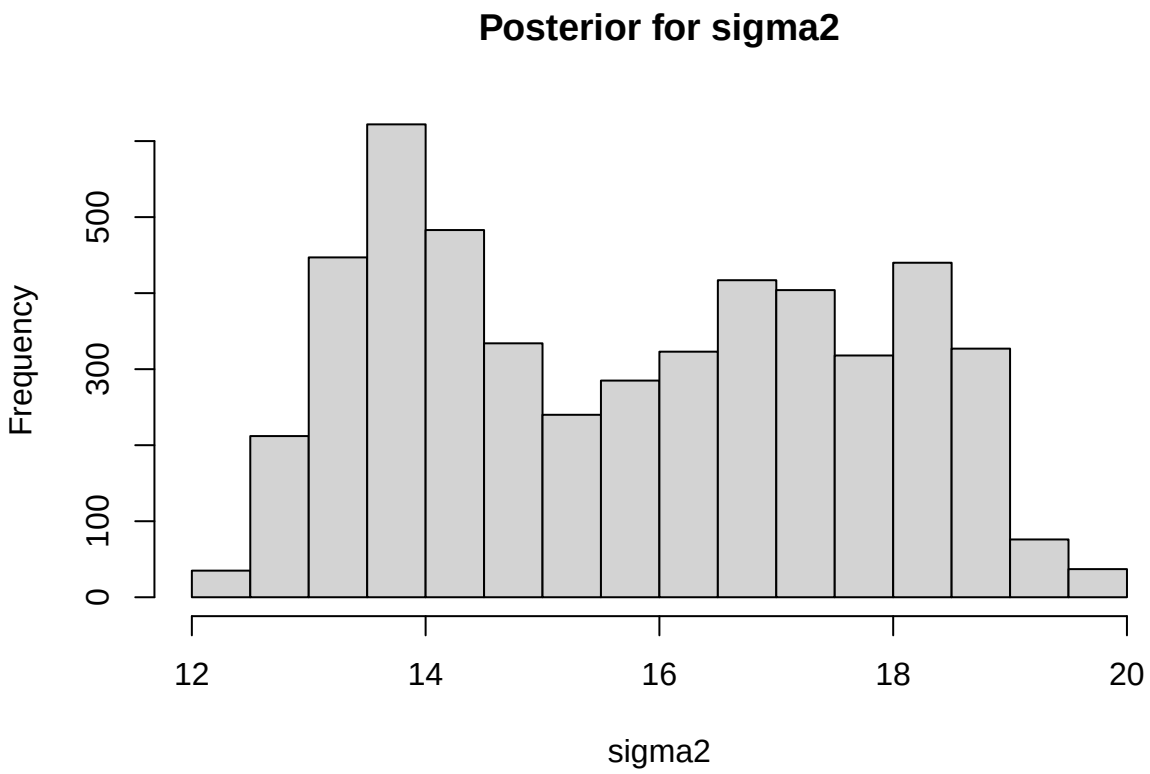
   At the $(r+1)$th MCMC iteration
2. Generate $\beta_{new}, \sigma_{new}^2, \delta_{new}$ from a uniform random walk on $\beta_{old}, \sigma_{old}^2, \delta_{old}$
3. Generate $x_{new} \sim e^{(\beta_{new} - \frac{1}{2})t + W(t)}$, $y_{sim,new} \sim x_{new} + N(0, \sigma_{new}^2)$, and calculate $\bar{y}_{sim,new}$
4. With probability $\alpha = min(1, \frac{\frac{1}{\delta_{new}} K(\frac{\bar{y}_{sim,new} - \bar{y}}{\delta_{new}})}{\frac{1}{\delta_{old}} K(\frac{\bar{y}_{sim,old} - \bar{y}}{\delta_{old}})})$, set $(\beta_{old}, \sigma_{old}^2, \delta_{old}) = (\beta_{new}, \sigma_{new}^2, \delta_{new})$. Otherwise set $(\beta_{old}, \sigma_{old}^2, \delta_{old}) = (\beta_{old}, \sigma_{old}^2, \delta_{old})$
5. Increment r and return to step 2.

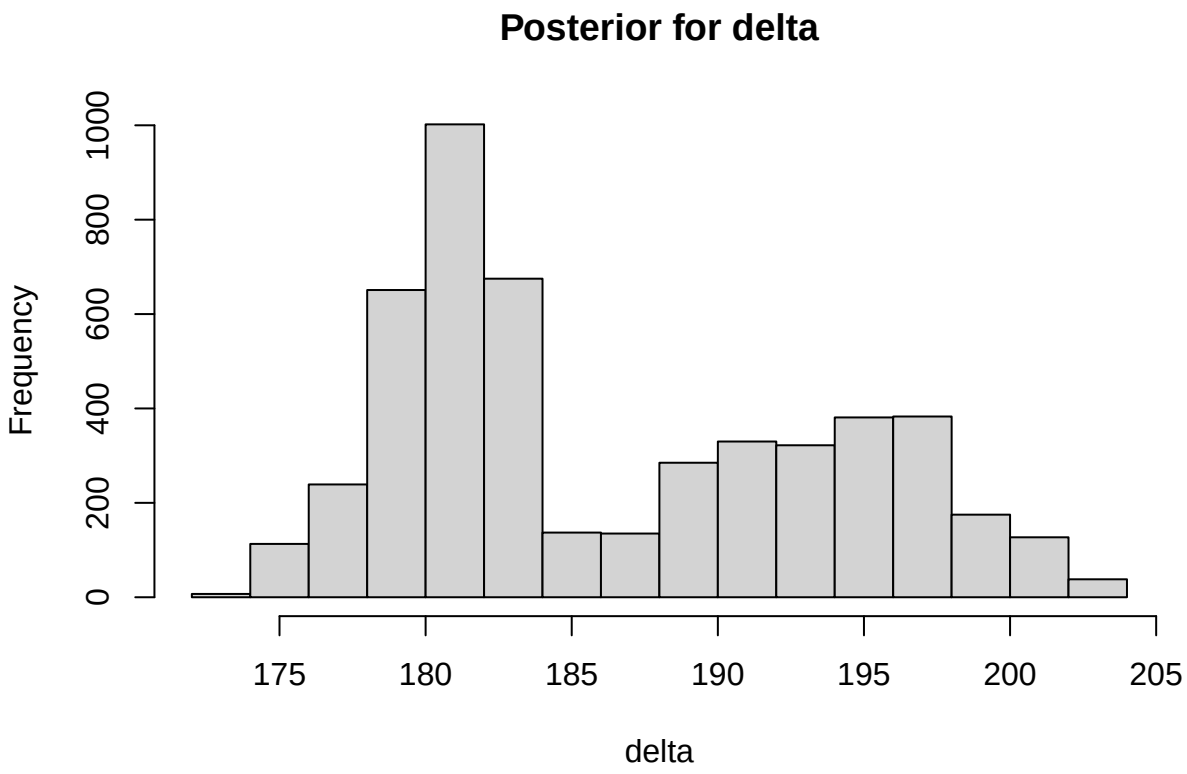
The code of this implementation will be provided in the appendix.

Data generation and application:

For the data generation, we will use the true values of $\beta = 5, \sigma^2 = 20$. We use 100 times points $t = 1, 2, 3...100$. For the initial values, we set $\beta = 1, \sigma^2 = 1, \delta = 200$. For the MCMC chain, 10000 iterations were run, with the first 5000 dropped as the burn in period. The following are the posteriors for β, σ^2, δ


Posterior for beta
Frequency
0
200
400
600
800
4.85
4.90
4.95
5.00
5.05
5.10
5.15
beta


Posterior for sigma2
Frequency
0
100
300
500
12
14
16
18
20
sigma2



We can see that, although the posterior for β is somewhat close to the true value, the posterior for σ^2 seems to be quite off. This may be due to an improper selection of summary statistic for this simplified version. In the paper, they recommend building a linear regression model to build a conditional expectation rather than just taking the marginal expectation as done here. There may also be other complications such as the selection of initial value for δ, as it is noted to be quite important in the paper, but here I just chose a value that would not cause a $\frac{0}{0}$ error in the computation, as the differences in the true mean and simulated means tended to be quite large, which would result in 0's when taking the kernel. Finally, the random walk parameter was chosen to be 1 for all of the parameters, and this may require tuning as well. But all in all, it does seem like this method works and is very fast. I am just happy that I got something that worked and was not completely disastrous in terms of results.

Appendix:

```
set.seed=1124234248174

##create data
beta=5
sigma2=20

t=seq(1,100,by=1)

x_true=exp(beta-.5)*t+rnorm(100,0,1)
y_true=x_true+rnorm(100,0,sqrt(sigma2))

##MCMC algorithm

MCMC=function(beta_initial,sigma2_initial,d_initial,reps){
```

```
  x=exp(beta_initial-.5)*t+rnorm(100,0,1)
  y=x+rnorm(100,0,sqrt(sigma2_initial))
  beta=beta_initial
  sigma2=sigma2_initial
  d=d_initial

  beta_vector=rep(0,reps)
  sigma2_vector=rep(0,reps)
  d_vector=rep(0,reps)

  for(i in 1:reps){
    beta_new=beta+runif(1,-.3,.3)
    sigma2_new=sigma2+runif(1,-.3,.3)
    d_new=d+runif(1,-1,1)

    x_new=exp(beta_new-.5)*t+rnorm(100,0,1)
    y_new=x_new+rnorm(100,0,sqrt(sigma2_new))

    y_new_bar=mean(y_new)
    y_old_bar=mean(y)

    alpha_num=dnorm(abs(y_new_bar-mean(y_true))/d_new)/d_new
    alpha_den=dnorm(abs(y_old_bar-mean(y_true))/d)/d

    alpha=alpha_num/alpha_den

    u=runif(1)
    if(alpha>u){
      beta=beta_new
      sigma2=sigma2_new
      d=d_new
      x=x_new
      y=y_new
    }

    beta_vector[i]=beta
    sigma2_vector[i]=sigma2
    d_vector[i]=d
  }

  return(list(beta_vector,sigma2_vector,d_vector))
}



##
set.seed=1143414231
test=MCMC(1,10,200,10000)
test[[1]]=test[[1]][-seq(1,5000)]
test[[2]]=test[[2]][-seq(1,5000)]
test[[3]]=test[[3]][-seq(1,5000)]
hist(test[[1]],main="Posterior for beta",xlab="beta")
hist(test[[2]],main="Posterior for sigma2",xlab="sigma2")
```

```r
hist(test[[3]],main="Posterior for delta",xlab="delta")
```