ODE Solvers Wrap Up

Zhanhao Zhang

May 7, 2019

We often encounter the situation where we have a chart of data for the time series of some variables, and we are well informed of the model that can describe the data, for instance, an ordinary differential equation system. However, among the parameters in the model, some of them are missing, for some reason. Then one of our goals will be to figure out the best estimation of those missing values based on the model and our data.

I tried three different machine learned based approaches to estimate those missing values, which are Iterated Filtering (in pomp package in R), Least Square Method, and (Particle) Monte Carlo Markov Chain. In the sections below, I will describe how each algorithm works, what their runtime approximately is, how well they fit the data, what their advantages and disadvantages are, and what we should be cautious about if we use them as tools.

1 Iterated Filtering (provided in pomp)

The Iterated Filtering algorithm is an Expectation Maximization (EM) based algorithm. It first generates a list of particles. For each of the particle, it starts with an initial guess (within a specified range) of the values for the missing parameters. Then, for each subsequent iterations, it first figure out the estimated values of the time series data using the current guess of the parameter values; then it calculates the likelihood of this set of parameter values based on the original data and the estimated values. Finally, weighting on the likelihood of each particle, it samples the parameter values again for each particle, and then go through the same process. The number of iterations has to be specified by users as well. Usually if the number of iterations is large enough, then all of the particles will converge to the set of parameter values that fit the data best. The output will be the set of parameter values that have the highest likelihood (a.k.a the MLE).

Iterated Filtering algorithm allows us to give the step function in both deterministic and stochastic ways. Deterministic means writing out the explicit expression for the differential equation system that is used to describe our data, while the stochastic way is to express the terms in differential equation systems using random variables like rbinom, rpois, rnorm, etc. However, if we are converting the differential equation system into a stochastic version, then we need to be cautious that the when we set each rate to be 0, we can get the same expression for each variable in stochastic version as in deterministic version, otherwise we will get a biased fitted curve, which is either too high or too low from the original data.

An example can be illustrated as follow, where f(X), g(X) are deterministic functions of X, $R_1(X)$, $R_2(X)$ are random variables generated using X: $\frac{dX}{dt} = f(X) + g(X)$ $\frac{dX}{dt} = R_1(X) + R_2(X)$ If we solve for $\frac{dX}{dt} = 0$, we should get a same expression for X from $f(X) + g(X) = 0$ as from $R_1(X) + R_2(X) = 0$

The runtime of this algorithm is significantly longer than the other two. For a dataset with 100 time steps, we will need an hour for the algorithm to finish, using 100 particles and 100 iterations. If we increase the number of particles to 1000, keeping everything else the same, then we will need around 10 hours, which is a task that is good to be left alone at the bed time.

The performance of this algorithm is also affected by the range of initial guesses for each missing parameter. With a narrower range, we can get a good fit using less particles (a good fit here means getting a fitted curve that falls right in the middle of the data points, or at least we can't witness a biased trend with eyes). For instance, if we can be sure that a parameter falls into [0, 1], then we had better not make the initial guess of this parameter in the range of [0, 100]. Otherwise, if the number of particles is not large enough, the initial guesses from the particles will be so sparse in the parameters' domains that they cannot give useful information for how to sample the parameter values in the next step.

In addition, we also need to be cautious about the Cscippet() function in pomp, which is used to speed up the program by running the program in C instead of R. However, Cscippet() only allows us to declare or pass in a parameter in a primitive type. We cannot pass in a vector as a parameter, nor can we allocate an array whose values can be accessed during all of the time steps. If we have to use an array or a vector as an auxiliary storage for our step function, then we should use the normal R function in lieu of Cscippet().

Though I have not got the chance to verify it, Iterated Filtering in pomp is designing for the ordinary differential equation systems that have random components. If we only have a system of differential equations that only has deterministic components, then we may want to move on to more efficient algorithms.

2 Least Square Method

The Least Square Method is a more straight forward algorithm. Its basic idea is to come up with a set of parameters, using which we can generate the estimated values for each variable in a time series that have the least square error from the original data.

We can generate the estimated values based on the parameters using either discrete time step (using a step function designed by clients) or continuous time step (using ode() function in R). But if we are using continuous time step, we should be cautious about the input of equations. The equations have to be of the form: $\frac{dX}{dt}$ = something independent of dt. When dt is very very small, it is valid to approximate $1 - e^{-p \cdot dt}$ with $p \cdot dt$. However, the time step in ode() function cannot be specified by clients, so we should always be aware of the issue that the time step may not be small enough for our data to use this kind of approximate.

The tool we can use to find the least square is nls.lm() function, which is available in minpack.lm package in R. The issue we should address here is that for a non-linear optimization problem, its least square loss function is usually not a convex one, and this algorithm can only guarantee a solution at a local minimum. Hence, what I do is to generate 100 initial guesses of the parameter values, and run all of them using the least square method, and finally pick the output that has the least deviance from the original data.

The runtime for this algorithm is much shorter, we can get a good fit of a data with 5000 time steps in around 1 to 2 hours.

3 (Particle) Monte Carlo Markov Chain

The Monte Carlo Markov Chain (MCMC), is based on rejection sampling. For a single chain, it starts at an initial guesses for the missing parameters. Then, it samples another set of parameters center around the previous set. Then it compares their likelihood. If the likelihood is higher than the previous one, then it updates the parameters to the current set. If the likelihood is lower than the previous one, then it adopts the current likelihood with the probability of $\frac{\text{current likelihood}}{\text{previous likelihood}}$. It continuous until it has gone through a user-specified number of iterations. The output is the set of parameter values that has the highest likelihood, among all these iterations. Particle Monte Carlo Markov Chain (pMCMC) is the same as MCMC, except that we have multiple chains instead of one, and the outcome will be the set of parameter values that has the highest likelihood among all these chains.

The likelihood function can be defined in at least two ways. The first way is to compute the mean and standard deviation for each variable, and assume the estimated data should follow a normal distribution with that mean and standard deviation. The second way is to measure the deviation between estimated values and original data, where the difference of their values follows a normal distribution with mean = 0, standard deviation = the standard deviation of that variable in the original data.

Each one has their pros and cons. The first one is good for data that have reached the stability, but if the original data has an increasing or decreasing trend, then the average value will be pulled down or up from the stability value, and the fitted curve will be biased. For the second method, it does not require the data to have reached its stability state, but it is not robust to outliers or randomness of the original data.

The runtime is much longer than the least square method, but it doesn't require as long time to get a good fit as the iterated filtering algorithm. For a dataset with 5000 time steps, we can generally get a good fit using several chains of length 1000, which takes 4 hours to execute. The number of chains is usually determined by the computing power available, for instance, how many cores there are in the computer.