Parameter estimation for nonlinear dynamical models.

GSoC 2018 Proposal

Vaibhav Kumar Dixit

March 13, 2018

1 Abstract

A differential equation model of a dynamical system is a nonlinear parameterized model that is created to match realistic scenarios and occasionally it might be associated with some data, obtained from the system or through simulation. Once there is reasonable level of confidence in the correctness of the model, the task that remains is to estimate the parameters of the model. Due to the structure of parameter estimation problems in dynamical models, statistics and machine learning techniques are an ideal choice for determining the parameters. During the course of the summer I will be implementing some statistical algorithms, including Stochastic Approximation Expectation Maximization(SAEM) and Maximum A Posteriori Estimation (MAP), for parameter estimation of a dynamic model. I will also work on extending support for parameter estimation in Stochastic Differential Equations (SDEs) by adding first differences distribution to generalized Log-Likelihood. These would be quite important additions to the suite of methods in JuliaDiffEq and would be of great use to scientific community involved in systems biology, HIV-AIDS study, and drug dosage estimation.

2 Synopsis

Most of the scientific community deals with the basic problem of trying to mathematically model the reality around them and this often involves dynamic systems. The general trend to model complex dynamical systems is through the use of differential equations. These differential equation models often have non-measurable parameters. The popular "forward-problem" of simulation consists of solving the differential equations for a given set of parameters, the "inverse problem" to simulation, known as parameter estimation, is the process of utilizing data to determine these model parameters. The inverse problem has not been covered as extensively as the forward problem and thus it presents a good avenue for further research primarily because of its applications in fields mentioned above.

Parameter estimation aims to find the unknown parameters of the model which give the best fit to a set of experimental data. In this way, parameters which cannot be measured directly will be determined in order to ensure the best fit of the model with the experimental results. The purpose of this project is to utilize the growing array of statistical, optimization, and machine learning tools

in the Julia ecosystem to build library functions with primary focus on Bayesian Inference. These will make it easier for scientists to perform this parameter estimation with the most high-powered and robust methodologies.

Currently JuliaDiffEq has DiffEqParamEstim.jl for the Optimization Methods and DiffEqBayes.jl for Bayesian Methods of parameter estimation. DiffEqParamEstim.jl supports the optimization algorithms available in BlackBoxOptim.jl, NLopt.jl and Optim.jl. Given my recent work leading up to the GSoC, DiffEqBayes.jl currently supports Stan.jl, Turing.jl and DynamicHMC.jl as the backends for the MCMC step. I propose the following.

- To add Stochastic Approximation Expectation-Maximization and Maximum a Posteriori Estimation methods (#62) for parameter estimation.
- I would also work on extending support for parameter estimation in Stochastic Differential equations by adding first differences distribution to generalized Log-Likelihood (#64).
- Also there isn't much work done on a good heuristic to measure the performance in Parameter Estimation problems, this would be another avenue to pursue as benchmarking is among the most important aspects of such packages and therefore some concrete way of comparative analysis will be required.
- Further during the course of the summer I would be actively involved in creating new benchmarks using the current methods and for the methods to be developed by me (#30).

3 The Project

3.1 Background

Julia has a diverse set of differential equation solvers and an active community around DifferentialEquations.jl. It also has a great array of optimization and statistical packages, therefore most of the basic infrastructure required for the development of new parameter estimation methodologies, are already in place. The primary task would be to implement the mentioned methods using these tools in a generalized manner so that all the problem types such as multiple parameter cases or multi-variable differential equations can be covered.

3.2 Goals

3.2.1 Theoretical description

1. SAEM

We start off with a problem of fitting linear and nonlinear ODEs with random effects in the structural parameters under situations in which the initial conditions of the ODEs may be unknown. The parameters can be considered a mix of fixed β and random effects b, the standard Expectation-Maximization (EM) procedures involves obtaining the Maximum Likelihood Estimate by cycling iteratively through an expectation (E)-step which involves analytically computing terms that appear in a pseudo-loglikelihood function given by the conditional expectation of the complete-data log-likelihood function with respect to the distribution $\Pr(b \mid Y; \theta)$ followed by the maximization (M)-step in which the parameter estimates are updated by using analytic formulas that serve to maximize the pseudo-loglikelihood function.

E step -

$$Q(\theta^{n+1} \mid \theta^n) = E_{b|Y;\theta}[\log(L(Y, b; \theta^n))]$$

M step -

$$\theta^{n+1} = argmax_\theta(Q(\theta^{n+1} \mid \theta^n))$$

SAEM differs from conventional E–M algorithm because we use a stochastic approximation procedure in the E-step, coupled with a gradient-type updating procedure (e.g. the Newton–Raphson) in the M-step. In the E-step we handle the intractable integrals of expectations by instead using Markov Chain Monte Carlo sampling techniques to obtain summary statistics of $\Pr(b \mid Y; \theta)$ and thus replacing the analytical expectation by an "approximation", then in the M-step a Newton–Raphson algorithm is used to obtain MLEs of θ, here a sequence of gain constants γ is used as a weightage factor for the degree to which the parameter estimates are updated in the subsequent iterations.

To use any gradient-type algorithm such as the Newton–Raphson we require the score vector and the information matrix of the loglikelihood function, denoted as $s_Y(\theta; Y)$ and $I_Y(\theta; Y)$, respectively. These are then used to update the parameter vector as defined by

$$\theta^m = \theta^{m-1} + [I_Y(\theta^{(m-1)}; Y)]^{-1} s_Y(\theta^{(m-1)}; Y)$$

Thus in the E step the samples obtained from MCMC on $\Pr(b \mid Y; \theta)$ are then used to obtain the score and Fisher information matrix. Then in the M Step updated parameter estimates are obtained using the above obtained score and information and the gain constant γ which is useful in moderating the estimation algorithm from settling too quickly into local minima in earlier iterations and also helps speed convergence toward a final set of parameter estimates during the later iterations. The SAEM algorithm alternates between the E-step and the M-step until some predefined convergence criteria have been met.

2. MAP

The usual Maximum A Posteriori algorithm involves prediction of parameters of a model, given the data. We assume a prior distribution for the parameter θ with a probability density function, say, $g(\theta)$ and the distribution of data y given θ as $f(Y \mid \theta)$. Then θ is estimated as the mode of the posterior distribution obtained using the Bayes Theorem from the prior distribution and the likelihood as follows.

$$\theta_{MAP}(x) = argmax_\theta(f(x \mid \theta)g(\theta))$$

In our case of Dynamical Models, the parameters, as mentioned in the SAEM description are considered to be a mixture of fixed and random effects and we assume the parameter vector as θ and the random effect as u. Thus the observed log-likelihood for the subject i as

$$L_\theta^i = log(\int_{R^p} L^\theta_{y^i|u^i}(Y^i \mid u)\phi(u)du)$$

We denote $L_\theta = L_\theta^i + ...L_\theta^n$. Now further we get

$$\theta_{MAP} = argmax_\theta[L_\theta^{MAP}] \quad where\ L_\theta^{MAP} = L_\theta + \log(\pi(\theta))$$

This gives us our parameter estimate.

3. First differences distribution to loglikelihood

The idea here is to provide an option for using a cost function that uses the likelihood of the first differences of the data against a distribution as the loss. In the the current implementation the distribution of the data points is measured at every time point but this can lead to problems as sometimes we may obtain the correct distribution but that won't give us the correct trajectory of individual particles. Such solutions are called "weak" solutions, the evolution of distributions is a deterministic process but to obtain the correct trajectory we need the "strong" solutions. Suppose you have a model where the initial value ($u0$) is a standard normal distribution and has to stay as a standard normal distribution over time. If every SDE's solution is actually constant, then it gets the overall distribution correct but the actual stochastic dynamics would be completely wrong measuring the loss in the first differences helps us to avoid such situations and information such as that constant doesn't match the data is not lost.

4. New benchmarks and heuristic for comparative analysis of parameter estimation methods

There is a lot of scope for expanding the current set of benchmarks with new problems, this will require some familiarity with academic literature to obtain new dynamical models and then to convert them to DifferentialEquations.jl's form. Also currently the parameter estimation benchmarking present a not so straightforward problem of comparison between the various available optimization and bayesian approaches available, this arises because there isn't a concrete set of properties that can be evaluated to make that comparison and also because the results are very sensitive to initialization both in accuracy and the time taken for obtaining the result. Hence using some standard problems with similar initializations of the parameters and obtaining more informative results from the algorithms will be required for future work, but this would require some theoretical guarantees to ensure reproducibility and correctness.

3.2.2 Implementation Details

1. SAEM

I aim to start off the SAEM method from a version which would handle a single parameter problem, using Lotka-Volterra equation as the example model. Initially I will begin with using DynamicHMC.jl as the sampling package. The rest of the algorithm the update and loglikelihood creation part can be implemented in a straightforward manner using Distributions.jl and hard-coding some of the steps.

This can then be generalized to include variety of extendability, something I have gained some experience in by my contributions to DiffEqBayes.jl. The main hurdle will be to get the score and Fisher information matrix for the loglikelihoods, decision will have to be made whether to use external packages for these or to implement them on our own.

2. MAP

The MAP implementation will be a bit more straightforward, the initial step involves creating the log-likelihood of the data given the priors on the parameters, this will involve creating a function like build_loss_objective which will be used to create the cost function which can then be used with an optimization package such as Optim.jl to get the maximum of the loglikelihood, which gives us the estimate of the parameter (θ_{MAP}). There is already some amount of framework set up in the existing implementations in DiffEqBayes.jl which can be used as a reference in calculating the posterior likelihood. The rest would involve substituting the MCMC sampling step with a MAP estimate calculation.

Passing the correct bounds and initial values to the optimizer would be very important here and it will need some time devoted to the fine tuning of them from the problem's variables.

3. First Difference to loglikelihood.

The cost function currently supports a loglikelihood of the data points but that does not properly catch all the information possible to supply to the optimizer in the cost function, more information can be packed in a loglikelihood of the first differences, I have presented a narrowed down version of the existing implementation to a specific case in which it supports this method, as given below first the existing implementation and then the narrow proposed version. The existing implementation of the cost function will be expanded to first difference and then the first difference method can be weighed against the original implementation and thus instead of one or the other both can be used.

Current implementation

```
function (f::LogLikeLoss)(sol::DESolution)
    distributions = f.distributions
    fill_length = length(f.t)-length(sol)
    for i in 1:fill_length
       push!(sol.u,fill(Inf,size(sol[1])))
    end
    ll = 0.0
    if eltype(distributions) <: UnivariateDistribution
        for j in 1:length(f.t), i in 1:length(sol[1][1])
            # i is the number of time points
            # j is the size of the system
            # corresponds to distributions[i,j]
            ll -= logpdf(f.distributions[i,j],sol[i,j])
        end
    else # MultivariateDistribution
        for j in 1:length(f.t), i in 1:length(sol[1][1])
            # i is the number of time points
            # j is the size of the system
            # corresponds to distributions[i,j]
            ll -= logpdf(f.distributions[i],sol[i])
        end
    end
    ll
end
```

New version with limited support

```
function (f::LogLikeLoss)(sol::DESolution)
    distributions = f.distributions
    is_normal_or_diff = f.ifdiffrence
    if is_normal_or_diff == True
        fill_length = length(f.t)-length(sol)
        for i in 1:fill_length
            push!(sol.u, fill(Inf,size(sol[1])))
        end
        diff_data = sol.u[:,2:end] - sol.u[:,1:end-1]
        ll = 0.0
        if eltype(distributions) <: UnivariateDistribution
        for j in 1:length(f.t-1), i in 1:length(diff_data[1][1])
            # i is the number of time points
            # j is the size of the system
            # corresponds to distributions[i,j]
            ll -= logpdf(f.distributions[i,j],diff_data[i,j])
        end
    end
    ll
end
```

4. Benchmarking

I am pretty familiar with the benchmarking process for parameter estimation problems in JuliaDiffEq, it has need of expansion to new problems and some re-structuring. I will be continuously involved in this throughout the summer. This will initially start with conversion of some standard benchmarks namely BioPreDyn from the available MATLAB code to Julia implementation. Another thing to explore in this area would be to develop a sound heuristic for comparative analysis among different algorithms within DiffEqBayes.jl and DiffEqParamEstim.jl

3.3 Timeline

Community Bonding Period (April 24- May 13)

This period will be spent in reading up the papers given in references below and developing a concrete plan how to implement them and finalizing the packages to be used, such as Stan.jl, Turing.jl or DynamicHMC.jl by the help of the concerned package maintainers and some experiments on our own part.

First and Second week (May 14 - May 28)

This period will be spent on implementing the SAEM algorithm described above.

Third week (May 29 - June 5)

This period will be appropriate to create tests and benchmarks for the SAEM algorithm and have some ideas regarding its performance against the existing algorithms in the package.

Fourth and Fifth week (June 6 - June 20)

We will use this period for the implementation of MAP algorithm.

Sixth week (June 21 - June 28)

The main agenda would be to get ready the benchmarks and tests for the MAP method and to establish some analysis of the relative performance with SAEM and the already available algorithms.

Seventh week (June 29 - July 5)

I would use this week as a buffer for catching up on any previous work left and to begin the discussion and get an idea about how to develop a formal heuristic for measuring comparative performance of different algorithms for parameter estimation.

Eighth week (July 6 - July 13)

This week would be spent on continuing the development of the heuristic.

Ninth and Tenth week (July 13 - July 20)

This would spent on refining the parameter estimation for SDE problems as described above.

Eleventh week (July 21 - July 28)

In this period I would like to explore SDE problems in more depth and, as currently there are no benchmarks for them, on creating some benchmarks for them.

Twelfth week (July 29 - August 5)

This week would go in the last of the benchmarks and also updating any documentation left.

Thirteenth week (August 6 - August 13)

Wrap up

I would like to mention that even though sufficient time has been given to everything right now but there is possibility of some unexpected or irresolvable issues coming so that should be taken into account. Most of the benchmarking work mentioned here involves updating the existing benchmarks but there is another aspect to it, that of adding the BioPreDyn benchmarks and some more if required, this will be done parallelly throughout the summer.

3.4 Previous Contributions

3.4.1 Major PRs

- JuliaDiffEq/DiffEqBayes.jl (#17) - Implemented stan_string method to expand stan_inference's compatibility with priors other than Normal.
- JuliaDiffEq/DiffEqBayes.jl (#23) - Implemented generate_theta, function to add individual bounds to parameter's members in stan_inference and updated the test cases.
- JuliaDiffEq/DiffEqBayes.jl (#26) - Implemented plot_chains function for plotting the trace_plots of results from stan_inference (expanded to turing_inference too in a later PR).
- JuliaDiffEq/DiffEqBenchmarks.jl (#11) - Added notebooks of LotkaVolterra and Lorenz equations' parameter estimation using DiffEqBayes.jl.
- JuliaDiffEq/DiffEqBayes.jl (#28) - Expanded turing_inference to support more than 2 variable equations.
- JuliaDiffEq/DiffEqBayes.jl (#31) - Created dynamichmc_inference as a wrapper over DynamicHMC.jl for parameter estimation using DynamicHMC.jl as the backend.
- JuliaDiffEq/DiffEqBenchmarks.jl (#19) - Added notebook for Lotka Volterra parameter estimation with optimization methods.
- JuliaDiffEq/DiffEqBenchmarks.jl (#18) - Added FitzHugh-Nagumo benchmark for DiffEqBayes.
- JuliaDiffEq/DiffEqParamEstim.jl (#66) - Fixed issue 65 regarding syntax problem with build_lsoptim_objective.
- JuliaDiffEq/DiffEqDocs.jl (#77) - Docs for dynamichmc_inference (DiffEqBayes.jl) for parameter estimation.

3.4.2 Other PRs

- JuliaLang/julia (#232248) - Float16/32 inconsistent compact NaN printing
- JuliaLang/julia (unmerged #25285) - Fixing #24214
- JuliaLang/julia (unmerged #24192) - Added initial docs for SingularException
- JuliaDiffEq/DiffEqBayes.jl (#19)
- JuliaDiffEq/DiffEqBayes.jl (#21)
- JuliaDiffEq/DiffEqBayes.jl (#25)
- JuliaDiffEq/DiffEqBenchmarks.jl (#12)
- JuliaDiffEq/DiffEqBayes.jl (#29)
- JuliaDiffEq/DiffEqBayes.jl (#32)
- JuliaDiffEq/DiffEqBenchmarks.jl (#16)

- JuliaDiffEq/DiffEqBayes.jl (#34)
- JuliaDiffEq/DiffEqDocs.jl (#79)
- JuliaDiffEq/DiffEqBayes.jl (#37)
- JuliaDiffEq/DiffEqBenchmarks.jl (#20)
- JuliaDiffEq/DiffEqBenchmarks.jl (#21)
- JuliaDiffEq/DiffEqDocs.jl (#90)
- JuliaDiffEq/DiffEqDocs.jl (#93)

3.5 About me

I am an undergraduate sophomore at Indian Institute of Technology (B.H.U.), Varanasi pursuing Mathematics and Computing. Using mathematics to understand and model the real world has been my passion since high school. I came across Julia when I started exploring open source in my freshman year and have been following its Discourse and Slack for some time now, of all the open source projects I have looked at the thing that has set Julia apart is the great community always available to help out. I plan on devoting as much of my time as possible to contributing as I have found it to be much more of a learning experience than most of the classroom teaching I have come across.

My interest areas include Statistics and Machine Learning, my work with JuliaDiffEq has also got me very interested in Differential Equations in the past couple of months. I have had some experience with Statistical Modeling of Time Series through a semester project in my third semester. I have had some other self projects, such as a Big Integer library with basic arithmetic in C++, some android apps and some work in Python to improve my programming skills, which can be found on my Github page.

3.5.1 Contact

Github: Vaibhavdixit02
Email: vaibhavyashdixit@gmail.com
Slack Id: vaibhavdixit02
Website: https://vaibhavdixit02.github.io/

3.6 References

- DynamicHMC.jl
- Turing.jl
- Stan.jl
- DiffEqBayes.jl
- Chow, Sy-Miin, et al. "Fitting Nonlinear Ordinary Differential Equation Models with Random Effects and Unknown Initial Conditions Using the Stochastic Approximation Expectation–Maximization (SAEM) Algorithm." Psychometrika, vol. 81, no. 1, 2014, pp. 102–134., doi:10.1007/s11336-014-9431-z.

- Drylewicz, Julia, Daniel Commenges and Rodolphe Thiebaut. "Maximum a Posteriori Estimation in Dynamical Models of Primary HIV Infection" Statistical Communications in Infectious Diseases, 4.1 (2012): -. Retrieved 13 Mar. 2018, from doi:10.1515/1948-4690.1040
- Rackauckas, Christopher, and Qing Nie. "DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia." Journal of Open Research Software, vol. 5, 2017, doi:10.5334/jors.151.