GSoC 2017 Application Ayush Pandey- Optimization and Bayesian Techniques for the parameter estimation of Differential Equations

April 3, 2017

1 Contact

Name : Ayush Pandey
University : Indian Institute of Technology (IIT), Kharagpur
Email : ayushpandey.iitkgp@gmail.com
IRC Handle : KrishnaKanhaiya at freenode.net
Github Username : Ayush-iitkgp
Mentor : Christopher Rackauckas

2 The Project

2.1 The Problem and Motivations

The aim of the project is to add different optimization and bayesian techniques to estimate the parameters of the differential equation in DiffEqParamEstim.jl package and do a comprehensive study of the pros and cons of the different algorithms to add it to the documentation.

Differential equation models are widely used in many scientific fields that include engineering, physics, and biomedical sciences. The so-called "forward problem" that is the problem of solving differential equations for given parameter values in the differential equation models has been extensively studied by mathematicians, physicists, engineers, and scientists.

However, the **inverse problem**, the problem of parameter estimation based on the measurements of output variables, has not been well explored using modern optimization and statistical methods. 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. This will be done by globally minimizing an objective function which measures the quality of the fit. This inverse problem usually considers a cost function to be optimized (such as maximum likelihood). This problem has applications in systems biology, HIV-AIDS study, and drug dosage estimation.

3 The Plan

I propose to implement different algorithms for estimation of the parameters of the differential equations such as:

1. Implement support for more Kernels for two-stage method
2. Implement support for different distance measures such as (log-)likelihood and expectation maximization algorithm
3. Implement regularization techniques such as Tikhonov regularization, LASSO regularization and integrate with PenaltyFunctions.jl
4. Provide support for deterministic and heuristic algorithms for global optimum such as BlackBoxOptim, genetic algorithm
5. Implement Generalized Profiling Approach using Model Relaxation method for parameter estimation
6. Bayesian techniques for parameter estimation using Stan.jl

Stretch Goals

1. Implementing support for parameter estimation for stochastic differential equations
2. Writing native Julia bayesian computational engine using Mamba.jl and Klara.jl

I also aim to do a detailed study of the methods and write a proper documentation on where to use the algorithms, precautions to take while using them and the pros and cons of the above methods.

3.1 Mathematical Formulation

First of all, we need to understand the mathematical formulation of the inverse parameter estimation problem before we get into the implementation details.

3.1.1 General Form of Parameter Estimation Problem

$$\frac{dx(t,\theta)}{dt} = f(t, x(t,\theta), u(t), \theta), \tag{1}$$

$$y(x,\theta) = g(x(t,\theta),\theta), \tag{2}$$

$$x(t_0) = x_0(\theta), \quad t \in [\, t_0,\, t_f\,] \ , \tag{3}$$

where $x \in \mathbf{R}^n$ is the observed state vector, $\theta \in R^m$ is the vector of unknown parameters, f(.) is a known linear or non-linear function vector. In general, we may not observe x directly but we can observe a function of x, here g(.) is the observation functions that maps the state variable to a vector of observable quantities $y \in R^n$, these are the signals that can be measured in the experiments.

3.1.2 Maximum likelihood and cost function

Maximum likelihood and cost function Assuming that the transformed measurements y are contaminated by additive normally distributed uncorrelated random measurement errors i.e. $y_{ijk} = y_{ijk}(x(t_i), \theta) + e_{ijk}$ where $e_{ijk} \sim \mathrm{N}(0, \sigma_{ijk}^2)$ is the random error with standard deviation σ_{ijk} and y_{ijk} is the measured value, the estimation of the model parameters is formulated as the maximization of the likelihood of the data:

$$\mathcal{L}(\tilde{y} \mid \theta) = \prod_{k=1}^{N_e} \prod_{j=1}^{N_{y,k}} \prod_{i=1}^{N_{t,k,j}} \frac{1}{\sqrt{2\pi\sigma_{ijk}^2}} \times \exp\left(-\frac{1}{2} \frac{(y_{ijk}(x(t_i,\theta),\theta) - \tilde{y}_{ijk})^2}{\sigma_{ijk}^2}\right) , \tag{4}$$

where N_e is the number of experiments, $N_{y,k}$ is the number of observed compounds in the kth experiment, and $N_{t,k,j}$ is the number of measurement time points of the jth observed quantity in the kth experiment.

From the theory of maximum likelihood, **the objective is to maximize the likelihood function** and estimate the parameters for which the likelihood function is maximized.

Since the likelihood function consists of product terms, we take the log of the likelihood function and negate it to ease our calculation, thus we get log-likelihood function as follows:

$$Q_{\mathrm{LS}}(\theta) = \sum_{k=1}^{N_e} \sum_{j=1}^{N_{y,k}} \sum_{i=1}^{N_{t,k,j}} \left(\frac{y_{ijk}(x(t_i,\theta),\theta) - \tilde{y}_{ijk}}{\sigma_{ijk}}\right)^2 = R(\theta)^T R(\theta) , \tag{5}$$

Note - The maximization of the likelihood function (4) is equivalent to the minimization of the weighted least squares cost function (5).

where the residual vector $\mathrm{R}(\cdot) : R^{N_\theta} \to R^{N_D}$ is constructed from the squared terms by arranging them to a vector. With this, the model calibration problem can be stated as the well-known nonlinear least-squares (NLS) optimization problem:

A θ vector that solves this optimization problem is called the maximum-likelihood estimates of model parameters.

Therefore, the objective of the "**inverse problem**" is to pose the problem as the optimization problem and get the optimum value to estimate the parameters.

$$
\begin{aligned}
\underset{\theta}{\text{minimize}} \quad & Q_{\text{LS}}(\theta) = R(\theta)^T R(\theta) \\
\text{subject to} \quad & \theta_{\min} \leq \theta \leq \theta_{\max} \ , \\
& \frac{dx(t,\theta)}{dt} = f(u(t), x(t,\theta), \theta) \ , \\
& y(x,\theta) = g(x(t,\theta),\theta) \ , \\
& x(t_0) = x_0(\theta), \quad t \in [\, t_0,\, t_f\,] \ .
\end{aligned} \tag{6}
$$

4 Execution

4.1 Implement support for more Kernels for two-stage method

This method is inspired by the research paper Parameter estimation for Differential Equation Models using a Framework of Measurement Error in Regression Models where the idea is to estimate the parameters of the Ordinary Differential Equations (ODE) using local smoothing approach and a pseudo-least square.

Note - This method only works for ordinary differential equations models.

Let's rewrite the differential equation written above in another form:

$$
\frac{d\mathbf{X}(t)}{dt} = F\{\mathbf{X}(t), \beta\},
$$

where X(t) = $X1(t), \ldots, Xk(t)^T$ is an unobserved state vector, $\beta = (\beta 1, \ldots, \beta m)^T$ is a vector of unknown parameters, and F(·) = $F1(), \ldots, Fk()^T$ is a known linear or nonlinear function vector. In practice, we may not observe X(t) directly, but we can observe its surrogate Y(t). For simplicity, assume an additive measurement error model to relate X(t) to the surrogate Y(t), i.e.,

$$
\boldsymbol{Y}(t) = \mathbf{X}(t) + e(t),
$$

where the measurement error e(t) is independent of X(t) with a covariance matrix Σ_e.

Suppose X^′(t) is an estimator of X′(t). Substituting the estimates X^′(ti), i = 1, . . . , n, in the ODE equation above, we obtain a regression model:

where $\Delta(t_i)$ denotes the substitution error vector, that is $\Delta(t_i)$ = X^′(t_i)− X′(t_i). If X′(t_i) is an

$$\widehat{\mathbf{X}}'(t_i)=F\{\mathbf{X}(t_i),\beta\}+\Delta(t_i),$$

unbiased estimator of X′(t_i), $\Delta(t_i)$ are errors with mean zero but are not independent. However, if the estimator X′(t_i) is a biased estimator.

The idea is to estimate X(ti) and X′(ti) using regression techniques such as local polynomial regression, smoothing spline and the regression spline.

I have already implemented, a two-stage method which uses local linear regression to estimate X(t) and local quadratic regression to estimate X′(t).

As a consequence, the estimators Xˆ(t) and Xˆ′(t) can be expressed as

$$\widehat{X}(t)=\xi_1^{\mathrm{T}}(\mathbf{T}_{1,t}^{\mathrm{T}}\mathbf{W}_t\mathbf{T}_{1,t})^{-1}\mathbf{T}_{1,t}^{\mathrm{T}}\mathbf{W}_tY,$$
$$\widehat{X}'(t)=\xi_2^{\mathrm{T}}(\mathbf{T}_{2,t}^{\mathrm{T}}\mathbf{W}_t\mathbf{T}_{2,t})^{-1}\mathbf{T}_{2,t}\mathbf{W}_tY,$$

where

$$\mathbf{W}_t=\mathrm{diag}\{K_h(t_1-t),\ldots,K_h(t_n-t)\}$$

and K(.) is a symmetric kernel function K_h(.) = K(./h)/h and h is a proper bandwidth.

Currently, the implementation supports **Epanechnikov**, **Uniform** and **Triangular** kernel functions. I propose to include more kernels such as Quartic, Triweight, Tricube, Gaussian, Cosine, Logistic, Sigmoid function, and Silverman.

I also plan to include the advantages and the disadvantages of this method such as :

Advantages

1. Computational efficiency
2. Easing of the convergence problem
3. The initial values of the state variables of the differential equations not required
4. Providing good initial estimates of the unknown parameters for other computationally-intensive methods to further refine the estimates rapidly

Disadvantage

1. This method does not converge to the global/local minima as the Non-Linear regression does in the objective function is convex/concave.

4.2 Implement support for different distance measures such as (log-)likelihood and expectation maximization (EM) algorithm

In the present implementation the "build_loss_objective" function only covers the Least-Square objective function and the function defined in LossFunctions.jl. It also assumes that the errors in

measurement are not correlated i.e

$$y = f(x) + N(0, I)$$

Then the maximum likelihood estimate of x given observations y is the solution to the non-linear least squares problem:

x^* = **min** $||f(x)||^2$

This type of objective function can easily be constructed using LossFunction.jl. However, there are cases when the error is correlated such as:

$$y = f(x) + N(0, S)$$

then the maximum likelihood problem to be solved is:

x^* = **min** f'(x)S^{-1}f(x)

Thus, I plan to include the functionality to provide the user with the option to state whether the errors are correlated or not and accordingly build the cost/objective function.Also, the present implementation assumes that the variance of the error terms is same which may always be not the case. I will also implement an option for the **weighted least square cost function**.

The log-likelihood function reduces to weighted least squares when the error is normally distributed which may not be the case always. Thus, the idea way to go about it would be to provide the option to construct the likelihood function as the cost function. Log-likelihood should also support different versions depending on the type of experimental noise:

1. 'homo' homoscedastic noise with constant variance
2. 'homo_var' homoscedastic noise with known non-constant variance
3. 'hetero' heteroscedastic noise with variance proportional to the observations

However, when we have lots of missing data or the *latent variable* for which no data has been observed, the marginal log-likelihood cost function is multi-modal and hence very difficult to maximize. Thus, the need for *expectation maximization* algorithm.

** Expectation Maximization Algorithm**

Assumption - The data is drawn from the population following the *exponential family of distributions*.

The observations (X_1, X_2) are drawn from the population having probability density function P_θ for some unknown θ

Given - Only the observations of X_1 is known x = ($x_1, x_2, ...$)

Algorithm

1. Initialize $\theta = \theta_0$
2. for i in 0,1,2,.....n

E-step - $Q(\theta, \theta_i)$ = $E_{\theta_i}(logP_\theta(X1, X2)|X_1 = x)$

M-step - θ_{i+1} = **max** $Q(\theta, \theta_i)$

Repeat step 2 till the resonable convergence is obtained.

4.3 Implement regularization techniques such as Tikhonov regularization, LASSO regularization and integrate with PenaltyFunctions.jl

Regularization aims to make the problem less complex (more regular), i.e. to ensure the uniqueness of the solution, to reduce the ill-conditioning and to avoid model overfitting. These techniques are ways to surmount ill-posedness and ill-conditioning.

For non-linear cost function, there is no general recipe for the selection of the regularization method. The idea is to provide the users with the options to choose from. PenaltyFunctions.jl provides two main families of penalty functions namely **Element Penalties** and **Array Penalties** which can be used as the regularization function.

A comprehensive list of penalty functions provided by PenaltyFunctions.jl are:

Element Penalties

Penalty	value on element
`NoPenalty()`	`g(θ) = 0`
`L1Penalty()`	`g(θ) = abs(θ)`
`L2Penalty()`	`g(θ) = 0.5 * θ ^ 2`
`ElasticNetPenalty(α = 0.5)`	`g(θ) = (1 - α) * abs(θ) + α * .5 * θ ^ 2`
`SCADPenalty(a = 3.7, γ = 1.0)`	`L1Penalty that blends to constant`
`MCPPenalty(γ = 2.0)`	`g(θ) = abs(θ) < γ ? abs(θ) - θ ^ 2 / 2γ : γ / 2`
`LogPenalty(η = 1.0)`	`g(θ) = log(1 + η * abs(θ))`

Array Penalties

Penalty	value on array
`NuclearNormPenalty()`	`sum of singular values of x`
`MahalanobisPenalty(C)`	`g(x) = x' * C' * C * x`
`GroupLassoPenalty()`	`g(x) = vecnorm(x)`

I propose to integrate the PenaltyFunctions.jl with DiffeEqParamEstim to support the different penalty functions as regularization terms in the cost function.

There are 2 other penalty type regularization techniques which should be supported, these are:

Tikhonov regularization

$$\text{L}(\theta) = (\theta - \theta^{ref})^T W^T W (\theta - \theta^{ref})$$

where W $\in R^{N\theta \times N\theta}$ is a diagonal scaling matrix and $\theta^{ref} \in R^{N\theta}$ is a reference parameter vector.

LASSO regularization

$$\sum \theta_i <= \text{t}$$

where t is the value provided by the user.

4.4 Provide support for deterministic and heuristic algorithms for global optimum such as BlackBoxOptim, genetic algorithm

It is well-known that the cost function (5) can be highly nonlinear and nonconvex in the model parameters. Many efficient local optimization algorithms have been developed to find the solution of nonlinear least squares problems, including Gauss-Newton, Levenberg-Marquardt and trust-region methods. These local methods are especially efficient when provided with high quality first (gradient, Jacobian) and second order (Hessian) information via parametric sensitivities. However, in this type of problems they will likely converge to local solutions close to the initial guess of the parameters. Thus, there is a need to integrate global optimization algorithm packages with DiffEqParamEstim. JuliaOpt currently supports the following global optimization algorithms:

1. BlackBoxOptim - The present cost function supports optimization using BlackBoXOptim. I will add an example to illustrate the users on how to use it.
2. Evolutionary algorithms
3. GeneticAlgorithms
4. StochasticSearch such as Tabu Search and Simulated Annealing

The idea is to mould the cost function (least square or likelihood) generated during parameter estimation to be in the format accepted by the above solvers.

Note - Deterministic global optimization methods can guarantee global optimality but their computationally cost increases exponentially with the number of estimated parameters. Alternatively, stochastic and heuristic methods can be used as more practical alternatives, usually obtaining adequate solutions in reasonable computation times, although at the price of no guarantees. In such context, metaheuristics, hybrids (i.e. combinations) with efficient local search methods have been particularly successful, I would also like to look at an approach to combine the deterministic and the heuristic techniques to converge to the exact global optimum in a reasonable time.

4.5 Implement Generalized Profiling Approach using Model Relaxation method for parameter estimation

The likelihood, least squares and the two-stage methods to estimate the parameters of the differential equations from noisy data are computationally intensive and are often poorly suited to statistical techniques such as inference and interval estimation. The generalized smoothing approach is is based on the modification of data smoothing methods along with a generalization of profiled estimation and have been successfully deployed to find the interval estimates of the parameter.

4.5.1 Methodology

Given the differential equation of the form:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}, \mathbf{u}, t|\boldsymbol{\theta}).$$

The idea is to estimate the solution $\hat{x(t)}$ as the linear combination of the basis functions ϕ_i(t) as:

$$x_i(t) = \sum_{k}^{K_i} c_{ik}\phi_{ik}(t) = \mathbf{c}_i'\boldsymbol{\phi}_i(t),$$

where the number K_i of basis functions in vector ϕ_i is chosen so as to ensure enough exibility to capture the variation in xi and its derivatives that is required to satisfy the above differential equation.

Note - Although the original collocation methods used polynomial bases, the choice of ϕ_i(t) is taken to be spline because of it's computational efficiency, also because it allows control over the smoothness of the solution at specic values of t.

The first step is to find c_i' in terms of the parameter θ and the smoothing parameter λ using model fitting and data fitting simultaneously as:

$$\text{J(c} \mid \theta,\sigma,\lambda) = \sum w_i||y_i - \hat{x}_i(t_i)||^2 + PEN(x|\lambda)$$

where PEN(x $\mid \lambda$) is given by

$$PEN(x|L_\theta, \lambda) = \sum \lambda_i PEN_i(x)$$

and

$$L_{i,\theta}(x_i) = \dot{x_i} - f_i(x, u, t|\theta) = 0$$

Once, we have the c_i' the next step is to find the estimates of the parameter θ using data fitting as:

$$\textbf{Min} \sum (y(t_i) - \hat{x(t_i)})^2$$

Note - On the computational side, this method is as fast or faster than NLS and other approaches, and much faster than the Bayesian-MCMC method, which has comparable estimation efficiency. Unlike MCMC, the generalized profiling approach is relatively straightforward to deploy to a wide range of applications.

4.6 Bayesian techniques for parameter estimation using Stan.jl

Stan is an open-source probabilistic programming language that performs Bayesian inference for user-specifed models. Stan.jl is the Julia interface to the stan library.

Stan provides a built-in mechanism for specifying and solving systems of ordinary differential equations. Stan provides two different integrators, one tuned for solving non-stiff systems and one for stiff systems.

- rk45: a fourth and fifth order Runge-Kutta method for non-stiff systems
- bdf: a variable-step, variable-order, backward-differentiation formula implementation for stiff systems

Note - The stiff solvers are slower, but more robust.

Now, consider the ODE:

dy1 = y2 dy2 = -y1 - θ*y2

The above ODE can be translated into DifferentialEquations format as:

```
In [ ]: using DifferentialEquations
        f = @ode_def Harmonic begin
          dy1 = y2
          dy2 = -y1 - a*y2
```

```
end a=>1.5

u0 = [1.0;1.0]
tspan = (0.0,10.0)
prob = ODEProblem(f,u0,tspan)
```

The above ODE is also translated into Stan program given below:

In [ ]:
```
functions {
    real[] sho(real t,
    real[] y,
    real[] theta,
    real[] x_r,
    int[] x_i) {
        real dydt[2];
        dydt[1] = y[2];
        dydt[2] = -y[1] - theta[1] * y[2];
    return dydt;
        }
    }
data {
    int<lower=1> T;
    vector[2] y0;
    real ts[T];
    real theta[1];
    }
transformed data {
    real x_r[0];
    int x_i[0];
    }

model {
    }
generated quantities {
    vector[2] y_hat[T];
    matrix[2, 2] A;
    A[1, 1] = 0;
    A[1, 2] = 1;
    A[2, 1] = -1;
    A[2, 2] = -theta[1];
    for (t in 1:T)
    y_hat[t] = matrix_exp((t - 1) * A) * y0;
    // add measurement error
    for (t in 1:T) {
        y_hat[t, 1] = y_hat[t, 1] + normal_rng(0, 0.1);
        y_hat[t, 2] = y_hat[t, 2] + normal_rng(0, 0.1);
        }
    }
```

Our objective is to translate the ODE described in DifferentialEquations.jl using ParameterizedFunctions.jl into the corresponding Stan above and use Stan.jl to find the Bayesian estimates of the parameters.

Fortunately, ParametrizedFunctions.jl already provides us the option to extract the information the ODE model such as:

```
In [7]: println(f.origex)

begin  # In[6], line 3:
    dy1 = y2 # In[6], line 4:
    dy2 = -y1 - a * y2
end


In [8]: println(f.params) # prints the parameter of the differential equation

Symbol[:a]
Expr[quote
    du[1] = 1 * u[2]
    du[2] = -(u[1]) - a * u[2]
    nothing
end]
```

To summarize the above process in 3 steps: 1. Express the ODE model in Julia using `ParametrizedFunctions.jl`. 2. Convert the ParametrizedFunctions ODE model to Stan model using the functionalities provided by ParametrizedFunctions 3. Use Stan.jl to generate posterior distribution of parameters and return it to the user

5 About Me

5.1 Personal Background

I am a final year graduate student pursuing an Integrated Master of Science(MS) degree in Mathematics and Computing Sciences with specialization in Optimization at IIT Kharagpur, India. I am proficient in C, C++, Python, and Julia.

5.2 Previous Relevant Experience

I am a former Google Summer of the Code student with the Julia Language where I worked on the project titled **Support for complex numbers within Convex.jl**. As a result of our work, we became the first open-source DCP package to support optimization with complex-variables. I have also worked on different projects related to optimization and machine learning. Please find all my projects here. I have taken courses on differential equations such as Partial Differential Equations, Numerical solution of ODE and PDE and Advanced NUmnerical Techniques(Theory and Lab) where I have written MATLAB/C code for different numerical techniques used in solving ODEs and PDEs.

5.3 Relevant Courses

- Partial Differential Equations
- Numerical solution of ODE and PDE
- Advanced Numerical Techniques
- Stochastic Processes
- Non-Linear Programming
- Convex Optimization
- Probability & Statistics, Regression, Generalized Linear Models
- Linear Algebra, Programming and Data Structures, Object Oriented System Design

5.4 Answers of listed questions

`1. What do you want to have completed by the end of the program?`

By the end of the program, I want to support different optimization and bayesian techniques used to estimate the parameters of the differential equations given the experimental data. Presently, DiffEqParamEstim only supports Non-Linear Regression technique to estimate the parameters. I aim to implement numerous techniques such as two-stage method, model relaxation technique, regularization, log-likelihood estimation, meta-heuristic techniques for global optimum and bayesian techniques using Stan.jl.

`2. Who's interested in the work, and how will it benefit them?`

The problem of parameter estimation has extensive use in the following fields: 1. HIV-AIDS viral dynamics 2. Systems biology 3. Drug dosage estimation

`3. What are the potential hurdles you might encounter, and how can you resolve them?`

In order to link Stan with DiffEqParamEstim, I will have to understand how Stan works. I will also need to go through the theory of Markov Chain Monte Carlo and Hamiltonian Monte Carlo to write in-situ Bayesian estimator for parameters of the differential equations. Lastly, I would need to understand the theory behind the stochastic differential equations to write an implementation to find it's parameters.

`4. How will you prioritize different aspects of the project like features, API usability, documentation, and robustness?`

Please refer to the "Timeline" section where I have described in details about my plans to tackle different aspects of the project.

`5. Does your project have any milestones that you can target throughout the period?`

Yes, before JuliaCon 2017, I would have implemented the various optimization and regularization algorithms to estimate the parameters and would be ready for testing for the Julia community.

`6. Are there any stretch goals you can make if the main project goes smoothly?`

Yes, I would try to implement native Julia computational engine for Bayesian estimation using tools like Mamba.jl or Klara.jl and the inverse problem for Stochastic Differential Equations based on the paper Control of Stochastic and Induced Switching in Biophysical Networks.

`7. What other time commitments, such as summer courses, other jobs, planned vacations, etc., will you have over the summer?`

I expect to work full time on the project that is 30 or more hours a week.

5.5 Contribution to Open-Source Projects

- List of all Pull Requests to Convex.jl
- Added two-stage method #6
- Improved DiffEqDocs.
- Implemented Genetic Algorithmfor heuristic optimization
- Elaborated installation instruction for SCIP.jl #35

5.6 Experience with Julia

I have been using Julia for last one year. In terms of functionality, I like Julia because of its **multiple dispatch** feature as it lets me overload operators with a lot of ease than other programming languages.

But the most astonishing feature of Julia is that it is empowering. In other high-level languages, the users can not be developers because developing new packages in those languages require the users to know the intricacies of low-level language whereas, in Julia, users can develop packages for their needs in Julia itself without compromising with the speed.

6 Timeline (tentative)

6.0.1 Community Bonding period (May 5 - 30, 2017)

My summer vacation will start from 6th of May. During this period, I would want to get myself more familiarized with the other similar projects in languages like MATLAB, python. The idea to take an inspiration from the similar mature projects so that we understand the potential user's requirements, have an idea of the structure of the API we provide to the users so that it is very easy to switch to out implementation once we are ready to get released.

I would also like to use this time to understand the intricacies of Stochastic Differential Equation Models and probabilistic programming language Stan to help me figure out how to provide a support for it in DiffEqParamEstim.

6.0.2 Week 1

Goal: *Implement support for more Kernels for two-stage method*

I plan to implement the different techniques and algorithms described in the "Execution" section on weekly basis. We can merge the code in the main project after each method is implemented. The end outcome of this week would be that the users would be able to provide an option for more kernel functions while using the two-stage method.

6.0.3 Week 2, 3 and 4

Goal: *Implement support for different distance measures such as (log-)likelihood and expectation*

In this week, I plan to implement the support for log-likelihood cost function which is expected to take 1 week at maximum. Week 3 and 4 would be utilized to implement the second part which is **Expectation maximization algorithm**.

6.0.4 Week 5 and 6

Goal: *Integrate with PenaltyFunctions.jl and implement Tikhonov and Lasso regularization*

These 2 weeks would be required to implement regularization techniques such as Tikhonov regularization, LASSO regularization and integrate DiffEqParamEstim with PenaltyFunctions.

6.0.5 Week 7 and 8

Goal: *Documentation for JuliaCon and attending JuliaCon*

By this time, we would have a general set-up for finding the parameters of the differential equations using optimization techniques. If selected for JuliaCon 2017, I would like to take this time to write an easy to understand documentation of the methods implemented so far and travel to Berkeley to present my work.

6.0.6 Week 8

Goal: *Provide support for deterministic and heuristic algorithms for global optimum such as BlackBoxOptim, genetic algorithm*

During this week, I plan to add the support for the 4 packages mentioned in my proposal to find the global optimum for the parameter estimation.

6.0.7 Week 9 and 10

Goal: *Implement Generalized Profiling Approach using Model Relaxation method for parameter estimation*

6.0.8 Week 11 and 12

Goal: *Link with Stan.jl to estimate the parameters using Bayesian inference*

6.0.9 End-Term evaluation

Goal: *Working to implement the strech goals*

Buffer period for any lagging work. I also aim to work towards implementing the stretch goals such as to implementing support for parameter estimation for stochastic differential equations and write native Julia bayesian computational engine using Mamba.jl and Klara.jl.

7 References

[1]. Parameter Estimation for Differential Equation Models Using a Framework of Measurement Error in Regression Models

[2]. Parameter estimation: the build_loss_objective #5

[3]. PenaltyFunctions.jl

[4]. Robust and efficient parameter estimation in dynamic models of biological systems

[5]. Parameter Estimation for Differential Equations: A Generalized Smoothing Approach

[6]. Stan: A probabilistic programming language for Bayesian inference and optimization

[7]. Linking with Stan project #135