The code for finding profile likelihood was written in Julia which can be installed at <https://julialang.org/downloads/>. I recommend using the VSCode IDE at <https://code.visualstudio.com/> with the Julia extension at <https://www.julia-vscode.org/>.

**The code will eventually be reuploaded with documentation and explanation on how to use it at** [**https://github.com/ph-kev/ProfileLikelihood**](https://github.com/ph-kev/ProfileLikelihood)**.** This will be done sometime next week (August 8, 2022 to August 12, 2022).

All code relating to profile likelihood is uploaded in the ProfileLikelihood folder in Google Drive. This *includes* **outdated code and code that may possibly not work** due to the constant changes to the functions used to produce the code. Code of interest will mostly be in the Saved Scripts folder which include the Julia code (.jl), saved data (.jld2), and the plots produced from the code. There are also various files scattered around that are used to find the confidence intervals and calculate R\_0 values.

The code for finding profile likelihood extensively used the DifferentialEquations.jl package at <https://diffeq.sciml.ai/stable/>. A tutorial for using the DifferentialEquations.jl package can be found at <https://diffeq.sciml.ai/stable/tutorials/ode_example/>.

We provide brief documentation of some functions.

generateData(index, seed, dist, prob, times, solver\_opts; incidenceStatus = false)

generateData takes in a differential equations problem prob and generates perfect data and noisy data at each value in the times array and the observation is based on the index which uses the order that the differential equations is written in. solver\_opts is a set of solver options. incidenceStatus generates incidence data and the index must be cumulative data. generateData returns perfectData and noisyData.

relativeError(data, sol, noiseLevel)

poissonError(data, sol)

constVarianceError(data, sol, sigma)

The code includes some objective functions where the measurement error follows relative error, the data follow the Poisson distribution, and the measurement error follows a normal distribution with mean 0 and known variance sigma^2.

likelihood(paramsCur, data, solObserved, prob, solver\_opts, times, objArr; incidenceObserved = [], paramIndex=0, paramEval=0)

likelihood find the likelihood using the objective function for the parameter value paramsCur and observations data. The objective functions are stored in the array objArr. solObserved is the same as the index when used to generate the data. If incidence data is being observed, then it is included in the array incidenceObserved. The parameters paramIndex and paramEval are used to fix a parameter at index paramIndex to the value paramEval.

likelihoodConst(obj; noiseLevel=0.01, times=Vector{Float64}(), data=Vector{Float64}(), sigma = 1)

likelihoodConst is the constant that is dropped when finding MLEs. obj is a string that have "relativeError", "poissonError", and "constVarianceError" as the available inputs. Some likelihood constants need additional information which could be specified in times, data, and sigma.

estimateParams(p0, fitter\_opts, data, solObserved, prob, solver\_opts, times, obj; incidenceObserved = [], paramIndex=0, paramEval=0, order = 4, maxRange = 1e-4, status = "local")

estimateParams are used to estimate the parameters using the objective function stored in the array obj. Most parameters are the same as before. order and maxRange are likely to be removed when the code is updated. status indicates the optimization algorithm used. Currently, status could be “local”, “global”, and “globalBB”. “local” used the NLOpt’s Newton algorithm (<https://nlopt.readthedocs.io/en/latest/NLopt_Algorithms/>) which find the local minimum is not recommended for use. “global” uses any algorithm from the Metaheuristics package (<https://github.com/jmejia8/Metaheuristics.jl>) depending on the settings in fitter\_opts. “globalBB” uses any algorithm from the BlackBoxOptim.jl package (<https://github.com/robertfeldt/BlackBoxOptim.jl>) depending on the settings in fitter\_opts. estimateParams return the value of the loss and the fitted parameters.

findThresholdOptimization(confidence, numsParams, loss)

findThresholdOptmization find the threshold given the confidence level, number of parameters, and loss value from estimateParams.

PL(stepSize, maxSteps, paramIndex, parametersFitted, data, solObserved, upperBound, loss, prob, solver\_opts, times, fitter\_opts, obj; incidenceObserved = [], order = 4, maxRange = 1e-4, status = "local")

PL return arrays of the fixed parameters and \chi^2\_{\rm PL} for plotting. stepSize is the stepsize taken when constructing the plots and maxSteps is the number of steps taken in one direction. upperBound sets a termination criteria where if \chi^2\_{\rm PL} is greater than upperBound, then the code does not stop in that direction.