Lotka-Volterra Bayesian Parameter Estimation Benchmarks

Vaibhav Dixit, Chris Rackauckas

March 15, 2019

0.1 Parameter Estimation of Lotka-Volterra Equation using DiffEqBayes.jl

```
using DiffEqBayes

using Distributions
using OrdinaryDiffEq, RecursiveArrayTools, ParameterizedFunctions, DiffEqUncertainity

Error: ArgumentError: Package DiffEqUncertainity not found in current path:
- Run `import Pkg; Pkg.add("DiffEqUncertainity")` to install the DiffEqUnce
rtainity package.

using Plots
using DiffEqMonteCarlo

Error: ArgumentError: Package DiffEqMonteCarlo not found in current path:
- Run `import Pkg; Pkg.add("DiffEqMonteCarlo")` to install the DiffEqMonteC
arlo package.

gr(fmt=:png)

Plots.GRBackend()

f = @ode_def_nohes LotkaVolterraTest begin
    dx = a*x - b*x*y
    dy = -c*y + d*x*y
end a b c d

Error: LoadError: UndefVarError: @ode_def_nohes not defined
in expression starting at none:1

u0 = [1.0,1.0]
tspan = (0.0,10.0)
p = [1.5,1.0,3.0,1,0]

5-element Array{Float64,1}:
 1.5
 1.0
 3.0
 1.0
 0.0
```

```
prob = ODEProblem(f,u0,tspan,p)
```

```
Error: UndefVarError: f not defined
```

```
sol = solve(prob,Tsit5())
```

```
Error: UndefVarError: prob not defined
```

```
t = collect(range(1,stop=10,length=10))
sig = 0.49
data = convert(Array, VectorOfArray([(sol(t[i]) + sig*randn(2)) for i in 1:length(t)]))
```

```
Error: UndefVarError: sol not defined
```

```
scatter(t, data[1,:], lab="#prey (data)")
```

```
Error: UndefVarError: data not defined
```

```
scatter!(t, data[2,:], lab="#predator (data)")
```

```
Error: UndefVarError: data not defined
```

```
plot!(sol)
```

```
Error: UndefVarError: sol not defined
```

```
priors =
    [Truncated(Normal(1.5,0.5),0.5,2.5),Truncated(Normal(1.2,0.5),0,2),Truncated(Normal(3.0,0.5),1,4),
```

```
4-element Array{Distributions.Truncated{Distributions.Normal{Float64},Distr
ibutions.Continuous},1}:
 Truncated(Distributions.Normal{Float64}(μ=1.5, σ=0.5), range=(0.5, 2.5))
 Truncated(Distributions.Normal{Float64}(μ=1.2, σ=0.5), range=(0.0, 2.0))
 Truncated(Distributions.Normal{Float64}(μ=3.0, σ=0.5), range=(1.0, 4.0))
 Truncated(Distributions.Normal{Float64}(μ=1.0, σ=0.5), range=(0.0, 2.0))
```

```
cb = AdaptiveProbIntsUncertainty(5)
```

```
Error: UndefVarError: AdaptiveProbIntsUncertainty not defined
```

```
monte_prob = MonteCarloProblem(prob)
```

```
Error: UndefVarError: prob not defined
```

```
sim = solve(monte_prob,Tsit5(),num_monte=100,callback=cb,reltol=1e-5,abstol=1e-5)
```

```
Error: UndefVarError: cb not defined
```

```
plot(sim,vars=(0,1),linealpha=0.4)
```

```
Error: UndefVarError: sim not defined
```

0.1.1 Parameter estimation with Stan.jl backend

The solution converges for tolerance values lower than 1e-3, lower tolerance leads to better accuracy in result but is accompanied by longer warmup and sampling time, truncated normal priors are used for preventing Stan from stepping into negative values.

```
@time bayesian_result_stan =
    stan_inference(prob,t,data,priors;reltol=1e-5,abstol=1e-5,vars
    =(StanODEData(),InverseGamma(3,2)))
```

```
Error: UndefVarError: prob not defined
```

Plots of the chains generated to show convergence.

```
plot_chain(bayesian_result_stan)
```

```
Error: UndefVarError: bayesian_result_stan not defined
```

0.1.2 Parameter estimation with Turing.jl backend

```
@time bayesian_result_turing = turing_inference(prob,Tsit5(),t,data,priors)
```

```
Error: UndefVarError: prob not defined
```

The chains seem to have not converged and require longer chains but there isn't anyway to pass warmup samples to Turing.jl's HMC sampler which has been used as the sampler in the implementation.

```
plot_chain(bayesian_result_turing)
```

```
Error: UndefVarError: bayesian_result_turing not defined
```

0.2 Conclusion

Lotka-Volterra Equation is a "predator-prey" model, it models population of two species in which one is the predator (wolf) and the other is the prey (rabbit). It depicts a cyclic behaviour, which is also seen in its Uncertainity Quantification Plots. This behaviour makes it easy to estimate even at very high tolerance values (1e-3).

In case of Stan.jl backend we obtain quite accurate values by setting sufficiently low tolerance at 1e-5 and passing 500 warmup samples, because as evident from the plots as it didn't converge before it which was observed from multiple runs, which ensures both high accuracy within 1.84 minutes, 1.7 minutes for warmup sampling and 14 seconds for sampling. Decreasing the tolerance leads to more accurate results but at the cost of significant increase in time taken.

Turing.jl backend implementation doesn't seem to have converged, inability to pass warmup samples is one of the drawbacks, the results obtained are quite accurate and it recorded 33.33 seconds.

Using DynamicHMC.jl as the backend gives good accuracy and takes 197 seconds but the exploration of the domain seems to be more constrained as compared to other backends as

evident from the plots which is due to the lower stepsize, this can be adjusted by passing the kwarg ϵ.