BAT.jl - Upgrading the Bayesian Analysis Toolkit

technische universität dortmund

Cornelius Grunwald for the BAT collaboration

Funded by DFG Deutsche Forschungsgemeinschaft German Research Foundation

Studienstiftung des deutschen Volkes

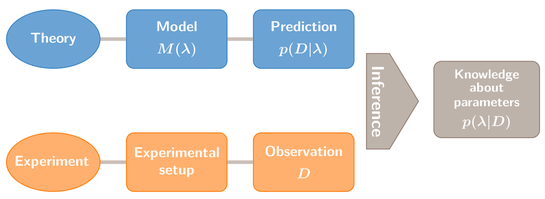
1. The Motivation

Statistical inference

goal: gain knowledge about model from data

typical analysis tasks:

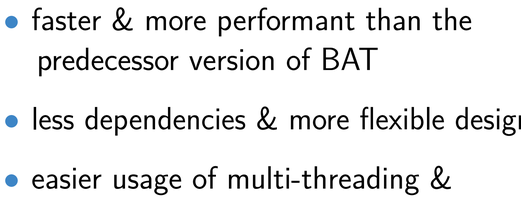
- estimation of model parameters
- model comparisons
- hypothesis testing
- limit setting
- goodness-of-fit tests



3. The Curse

Computation of posterior

- challenging task for real-world problems
  → curse of dimensionality



advanced algorithms are needed to perform high-dimensional:

- sampling of posterior
- optimization
- integration
- marginalization

5. The Rewrite

NEW

BAT.jl

- rewrite of BAT from scratch in the Julia programming language

design goals:

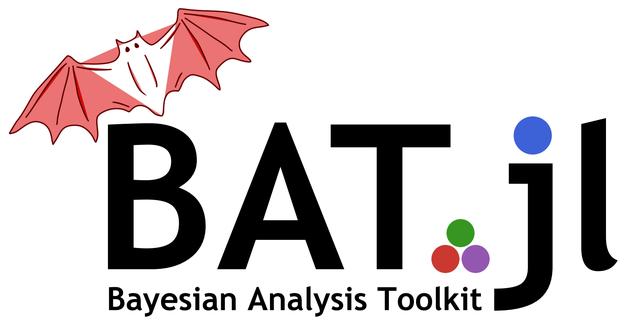
- faster & more performant than the predecessor version of BAT
- less dependencies & more flexible design
- easier usage of multi-threading & distributed computing
- open for further fields of research (become independent of ROOT)

→ new code structure & new algorithms

7. The Features

Current features of BAT.jl

- registered Julia package (v. 0.5)
- user-definable likelihoods & priors
- Metropolis-Hastings sampling algorithm
- native parallel running of Markov chains
- integrals with Adaptive Harmonic Mean Integration [1808.08051]
- call of likelihoods from other languages (directly or via binary RPC calls)
- formatted output & plotting recipes



github.com/bat/bat.jl

The How-To

Installation

install & setup BAT via Julia's built-in package manager

```
# installation via package manager (only first use)
using Pkg
pkg"add BAT"

# activate BAT
using BAT
```

Define model

define the statistical model in terms of the log-likelihood

- ✚ implement custom likelihoods
- ✚ use common distributions from *Distributions.jl*
- ✚ call code in python, C/C++, Fortran, R, mathematica, ... via Julia's interfaces
- ✚ call external likelihoods in any language via BAT.jl's RPC

```
log_likelihood = let μ = [15, 10], σ = [1.5, 2.5]
    params -> begin
        # custom implementation of normal distribution
        normal = ( -0.5 * log(2π*σ[1]^2)
                   -(params.λ1-μ[1])^2 / (2σ[1]^2) )

        # normal distribution from Distributions.jl
        n1 = pdf(Normal(μ[2]-5, σ[2]), params.λ2)
        n2 = pdf(Normal(μ[2]+5, σ[2]), params.λ2)

        return normal + log(n1 + 3*n2)
    end
end
```

Define prior

- ✚ implement custom priors
- ✚ use common distributions from *Distributions.jl*
- ✚ use histograms as priors

```
prior = NamedTupleDist(
    λ1 = Normal(6, 2.5), #for parameter1
    λ2 = -30.0..30.0     #for parameter2
)
```

Define prosterior

```
posterior = PosteriorDensity(log_likelihood, prior)
```

Setup sampler

choose sampling algorithm, number of chains & number of samples

- ✚ possible to adjust sampler settings for: proposal distribution, MCMC tuning algorithm, convergence test, chain initialization, tuning/burn-in strategy

```
# choose sampling algorithm
algorithm = MetropolisHastings()

nchains = 8       # number of Markov chains
nsamples = 10^6   # number of samples per chain
```

Generate samples

```
samples = bat_sample(
    posterior, (nsamples, nchains), algorithm
)
```

parameters	log_posterior	log_prior	weight
(λ1 = 11.063, λ2 = 17.904)	-14.160	-7.980	2
(λ1 = 11.113, λ2 = 18.104)	-14.209	-8.021	4
⋮	⋮	⋮	⋮

Show sample statistics

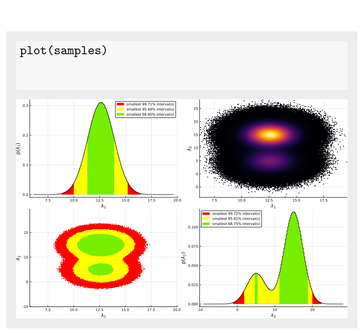
```
stats = bat_stats(samples)
```

```
Mode: [12.617, 15.000]    Mean: [12.618, 12.492]
Covariance: [1.650 -0.015; -0.015 25.063]
```

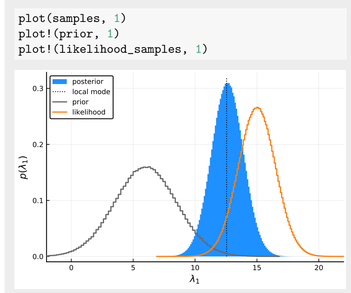
Visualize results

- ✚ predefined & customizable plot recipes
- ✚ individual plotting

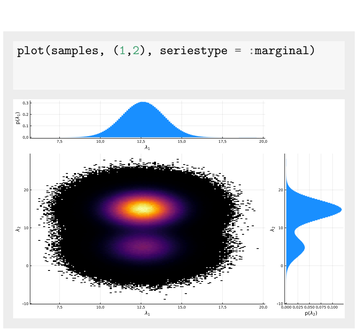
```
plot(samples)
```



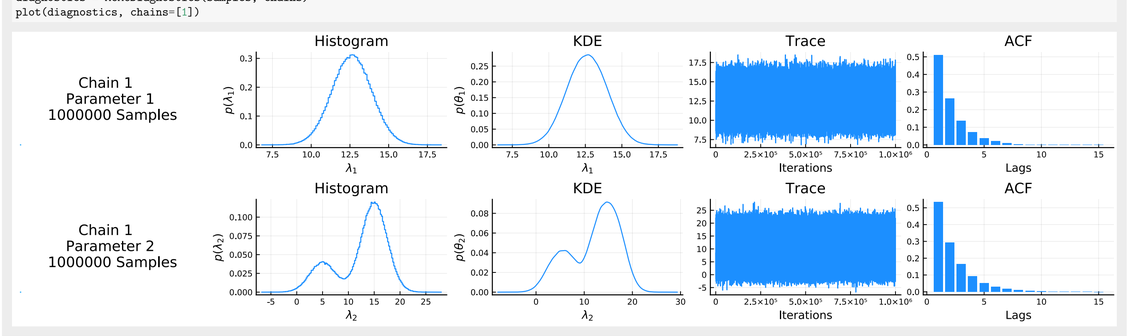
```
plot(samples, 1)
plot!(prior, 1)
plot!(likelihood_samples, 1)
```




```
plot(samples, (1,2), seriestype = :marginal)
```



```
diagnostics = MCMCDiagnostics(samples, chains)
plot(diagnostics, chains=[1])
```



Calculate integral from samples

- ✚ Adaptive Harmonic Mean Integration (AHMI)

```
evidence = bat_integrate(samples)
```

```
evidence: 7.8e-5 ± 2.1e-7
```

Export samples to file

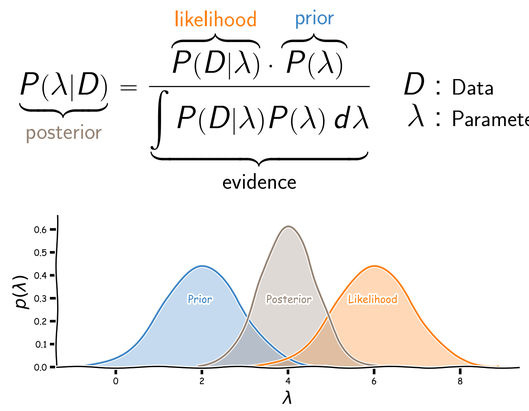
```
bat_write("samples.hdf5", samples)
```

2. The Theory

Bayesian inference

- use Bayes' theorem to update knowledge about model parameters λ from data D
- Bayes' theorem:

$$\underbrace{P(\lambda|D)}_{\text{posterior}} = \frac{\overbrace{P(D|\lambda)}^{\text{likelihood}} \cdot \overbrace{P(\lambda)}^{\text{prior}}}{\underbrace{\int P(D|\lambda)P(\lambda)\,d\lambda}_{\text{evidence}}} \qquad \begin{matrix} D: \text{Data} \\ \lambda: \text{Parameters} \end{matrix}$$



- posterior contains the knowledge about the parameters for given data

4. The Tool

Bayesian Analysis Toolkit - BAT

Comput. Phys. Commun. 180 (2009) 2197

- collection of algorithms & methods for Bayesian inference
- user-defined statistical models in a multi-purpose language (no tool-specific modeling language)
- use of Markov chain Monte Carlo algorithms for sampling of posterior
- toolbox-character: choose from different methods to solve custom problems
- first version released in 2008 (written in C++, depending on ROOT)

6. The Language

The Julia language

- modern general-purpose language (v. 1.0 released 2018)
- focus on high-performance numerical & scientific computing
- built-in package-management system
- large scientific community & growing ecosystem (> 3000 registered packages)
- features: multiple dispatch, dynamic typing, meta-programming & macros, ...
- natively possible to call code in other languages, e.g. python, C, Fortran, ...

8. The Future Prospects

Further development of BAT.jl

- release of v. 1.0 planned for this month
- finalizing main features & API design
- new sampling algorithms (WIP)
- support for distributed computing (WIP)
- development of examples, tutorials & templates for common analysis tasks (WIP)
- much more to come...
  → stay tuned!