Stan: A Platform for Scalable Bayesian Inference

The Stan Development Team
www.mc-stan.org, @mcmc_stan

Stan is a collection of state-of-the-art computational tools to facilitate the specification and fitting of the complex Bayesian models that arise at the frontiers of applied statistics. A high-performance math and statistics library written in C++ is exposed to users via interfaces to the command line, R, Python, and a rapidly growing list of other popular analysis environments. These interfaces are also available on all three common operating systems: Mac OS X, Linux, and Windows.

By automating statistical computation, Stan allows users to focus their efforts on exploiting their domain expertise to build more sophisticated models and better analyses. Academic fields as diverse as psychology, political science, ecology, astronomy, and physics, amongst many others, have used Stan to build analyses novel to their fields. Similarly, Stan is an increasingly powerful tool in industries spanning pharmacology and medicine to social media and entertainment to real estate and finance. The large scope of computing environments supported by Stan allow it to be quickly integrated into new applications across cinch and industry.

Stan is freedom-respecting, open-source software (new BSD core, some interfaces in GPLv3) and is affiliated with NumFOCUS, a 501(c)(3) nonprofit supporting open code and reproducible science.

Bayesian Inference

Bayesian inference is a powerful statistical methodology for learning from large, noisy, and often corrupted data. A Bayesian model is compromised of a *likelihood* that models the complex measurement process,

$$\pi(\mathcal{D} \mid \theta),$$

and a *prior distribution* that quantifies our knowledge of the underlying system before the measurement is made,

$$\pi(\theta).$$

Together these two components yield the *posterior distribution* which quantifies all of our information about the system being studied after the measurement,

$$\pi(\theta \mid \mathcal{D}) \propto \pi(\mathcal{D} \mid \theta)\,\pi(\theta).$$

Once the posterior distribution has been summarized we can extract principled inferences by computing expectations,

$$\mathbb{E}_{\pi}[f] = \int \mathrm{d}\theta\, \pi(\theta \mid \mathcal{D}) f(\theta),$$

such as means, variances, quantiles, and expected utility functions for making robust decisions.

Consequently the practical challenge of implementing Bayesian inference reduces to first being able to specify complex and bespoke posterior distributions and then being able to accurately compute posterior expectation values. These challenges are facilitated with the Stan Modeling Language and Stan Core Library, respectively.

The Stan Modeling Language

The Stan Modeling Language is a probabilistic programming language designed for specifying complex posterior distributions. Each Stan program is organized by programming blocks. The data block defines the measurement,

$\pi(\theta \mid \mathcal{D}) \quad \rightarrow$

```
data {
  int<lower=1> N;
  real x[N];
  real y[N];
}
parameters {
  real beta;
  real alpha;
  real<lower=0> sigma;
}
model {
  beta ~ normal(0, 1);
  alpha ~ normal(0, 1);
  sigma ~ cauchy(0, 2.5);
  y ~ normal(beta * x + alpha, sigma);
}
```

while the parameter block defines the degrees of freedom in the model,

$\pi(\theta \mid \mathcal{D}) \quad \rightarrow$

```
data {
  int<lower=1> N;
  real x[N];
  real y[N];
}
parameters {
  real beta;
  real alpha;
  real<lower=0> sigma;
}
model {
  beta ~ normal(0, 1);
  alpha ~ normal(0, 1);
  sigma ~ cauchy(0, 2.5);
  y ~ normal(beta * x + alpha, sigma);
}
```

Finally the model block defines the posterior density itself,

$\pi(\theta \mid \mathcal{D}) \quad \rightarrow$

```
data {
  int<lower=1> N;
  real x[N];
  real y[N];
}
parameters {
  real beta;
  real alpha;
  real<lower=0> sigma;
}
model {
  beta ~ normal(0, 1);
  alpha ~ normal(0, 1);
  sigma ~ cauchy(0, 2.5);
  y ~ normal(beta * x + alpha, sigma);
}
```

The language is extensive, supporting linear algebra, ordinary differential equations, and a large library of classic and modern probability density functions.

The Stan Core Library

Once a Bayesian model has been specified as a Stan program it is parsed into C++ code that can utilized by the Stan Core Library.

The first component of the Stan Core Library is the Stan Math Library, which admits high-performance evaluation of the posterior density while providing *automatic differentiation,* a algorithmic method for computing fast and exact derivatives of the posterior density.

The availability of the posterior density and its derivatives enables the implementation of state-of-the-art computational algorithms in the Stan Algorithm Library. These algorithms include

Optimization

The parameter values that optimize the posterior density can be found with an an implementation of the L-BFGS algorithm, allowing Stan to compute penalized maximum likelihood estimates.

Sampling

Stan utilizes a state-of-the-art implementation of Hamiltonian Monte Carlo to accurately estimate posterior expectations with Markov chain Monte Carlo. This implementation also features a suite of powerful diagnostics that inform the user when estimates are inaccurate or otherwise should not be trusted in a serious analysis.

Variational Inference

Automatic Differentiation Variational Inference is a algorithm currently under development to automate the application of variational inference for any model specified in Stan. The algorithm aims to provide less precise but faster estimates of posterior expectations than Markov chain Monte Carlo.

These latter two algorithms also demonstrate the power of Stan in not only supporting Bayesian inference but also promoting research in statistical computation. The Stan Core Library provides the building blocks for developing new algorithms while the Stan Programming Language allows these algorithms to be validated against the models actually be used in statistical practice.

Moreover, this interaction between algorithms and models facilitates the development of automated adaptation procedures that remove the burdensome task of algorithm tuning from users.