An Introduction to Stan and RStan

Houston R Users Group

Michael Weylandt

2016-12-06

Rice University

Introduction

Credits

I (MW) am not a developer of `Stan`, only a very happy user.

Credit for `Stan` goes to the Stan Development Team: Andrew Gelman, Bob Carpenter, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Allen Riddell, Marco Inacio, Jeffrey Arnold, Rob J. Goedman, Brian Lau, Mitzi Morris, Rob Trangucci, Jonah Sol Gabry, Alp Kucukelbir, Robert. L. Grant, Dustin Tran, Krzysztof Sakrejda, Aki Vehtari, Rayleigh Lei, Sebastian Weber, Chalres Margossian, Thel Seraphim, Vincent Picaud, Matt Hoffman, Michael Malecki, Peter Li, Yuanjun Guo.

Much of the material in this presentation mirrors the excellent `Stan` manual. Any mistakes in exposition are solely the responsibility of MW.

Background

How familiar are you with the following concepts?

- Bayesian Statistics

Background

How familiar are you with the following concepts?

- Bayesian Statistics
- Bayesian Computation

How familiar are you with the following concepts?

- Bayesian Statistics
- Bayesian Computation
- Stan

Table of Contents

Bayesian Inference

> *Statistics is the science of learning from data, and of measuring, controlling, and communicating uncertainty. ([DL12])*

Bayesian Statistics emphasizes the use of *probability* as a language for describing uncertainty.

An All-Too-Brief Introduction to Bayesian Inference

Bayesian statistics uses the language of probability to quantify information. Anything which is (treated as) unknown has a probability distribution associated with it.

The tools of probability, in particular Bayes' Rule, give a mathematically coherent framework for updating beliefs in the presence of data. Classical works on this point-of-view are Ramsey, Savage, Jeffreys and Jaynes [Ram31, Sav54, Jef61, Jay03].[1][2]

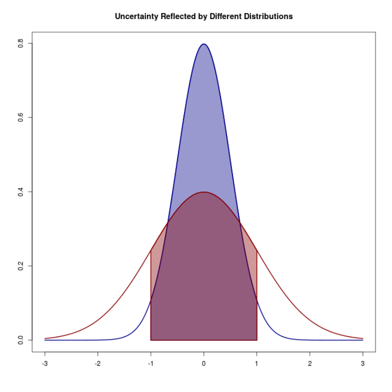
The probabilistic content of Bayes' Theorem is trivial. The statistical content is not. – Steve Huntsman (MathOverflow)

[1] While Bayesian inference is typically promoted on the basis of incorporating prior information and inferential flexibility, it can be shown to have good frequentist properties in a range of circumstances as well [Efr15].

[2] Two expositions of "subjective" probability of particular interest to finance are [Key21] and [SV01].

Two normal distributions with different standard deviations.

The blue distribution is more “precise” than the red one.[3]



[3] The connection between curvature and inferential precision is found in classical statistics as well: the Fisher information is a measure of curvature of the likelihood function. The field of *information geometry* uses differential geometry to develop this relationship in much more generality; see, *e.g.*, [ABNK+87] for more.

Basically all Bayesian inference problems[4] are of the canonical form:

- Given prior information about an unknown quantity θ, express that information as a probability distribution $p(\theta)$, conventionally known as the *prior*;
- Given data observed according to some random process depending on θ, construct an appropriate *likelihood* $p(X|\theta)$;
- Using Bayes' rule, calculate a new distribution of θ, conditioned on the data X; this distribution is conventionally known as the *posterior*:

$$\pi(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)\,\mathrm{d}\theta}$$

(Almost) All inference reduces to taking expectations under $\pi(\cdot)$.

[4] For an introduction to Bayesian methods see [McE15], [GH06], or [Hof09]; [GCS+14] is the Bayesian "Bible" for applied statistics. [Rob07] is an excellent text on Bayesian foundations.

The choice of prior has historically been one of the more controversial aspects of Bayesian analysis. Roughly, three classes of priors:

- Informative: Provide significant information and help guide analysis.
- Weakly Informative: Avoid pathologies, but let the data dominate. Similar to weak regularization in non-Bayesian analysis.
- Non-Informative. Attempt to provide no information: hard to achieve in practice.[5]

Technical Warning: If you don't provide a prior, `Stan` will implicitly use a uniform (flat) prior. For unbounded parameters, this gives an improper distribution and strange things can occur (*e.g.*, [HC96]). *Caveat emptor*

[5] See, *e.g.*, [KW96, BBS09].

Having defined our posterior distribution

$$\pi(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)\,\mathrm{d}\theta}$$

we want to actually perform statistical inference (make claims about the unobserved parameters).

Three major tasks:

Having defined our posterior distribution

$$\pi(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)\,\mathrm{d}\theta}$$

we want to actually perform statistical inference (make claims about the unobserved parameters).

Three major tasks:

- Point estimation: making a single "best guess" about a value of interest

Bayesian Inference

Having defined our posterior distribution

$$\pi(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)\,\mathrm{d}\theta}$$

we want to actually perform statistical inference (make claims about the unobserved parameters).

Three major tasks:

- Point estimation: making a single "best guess" about a value of interest
- Set/Interval estimation: selecting a set of values with a high probability of containing the parameter of interest

Bayesian Inference

Having defined our posterior distribution

$$\pi(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)\,\mathrm{d}\theta}$$

we want to actually perform statistical inference (make claims about the unobserved parameters).

Three major tasks:

- Point estimation: making a single "best guess" about a value of interest
- Set/Interval estimation: selecting a set of values with a high probability of containing the parameter of interest
- Prediction: making a guess about future observations

Having defined our posterior distribution

$$\pi(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)\,\mathrm{d}\theta}$$

we want to actually perform statistical inference (make claims about the unobserved parameters).

Three major tasks:

- Point estimation: making a single "best guess" about a value of interest
- Set/Interval estimation: selecting a set of values with a high probability of containing the parameter of interest
- Prediction: making a guess about future observations

In classical statistics, these often require three different sets of tools. In Bayesian statistics, all three can be accomplished using the posterior.

Bayesian Inference

Typically, the posterior is actually a pretty difficult distribution to work with directly, so we actually work with samples from the distribution. Historically, obtaining these samples has been the hardest part of Bayesian inference, but new tools like `Stan` have made it significantly easier.

Typically, the posterior is actually a pretty difficult distribution to work with directly, so we actually work with samples from the distribution. Historically, obtaining these samples has been the hardest part of Bayesian inference, but new tools like `Stan` have made it significantly easier.

Use of samples as opposed to the distribution itself is the heart of the *Monte Carlo* method.

Bayesian Inference

Typically, the posterior is actually a pretty difficult distribution to work with directly, so we actually work with samples from the distribution. Historically, obtaining these samples has been the hardest part of Bayesian inference, but new tools like `Stan` have made it significantly easier.

Use of samples as opposed to the distribution itself is the heart of the *Monte Carlo* method.

For point estimation, we typically use the posterior mean or median. With sufficient samples from the posterior, the sample mean and the posterior mean will be essentially the same.

Typically, the posterior is actually a pretty difficult distribution to work with directly, so we actually work with samples from the distribution. Historically, obtaining these samples has been the hardest part of Bayesian inference, but new tools like `Stan` have made it significantly easier.

Use of samples as opposed to the distribution itself is the heart of the *Monte Carlo* method.

For point estimation, we typically use the posterior mean or median. With sufficient samples from the posterior, the sample mean and the posterior mean will be essentially the same.

For set estimation, we simply construct a set that a given fraction of the posterior samples fall into.

Typically, the posterior is actually a pretty difficult distribution to work with directly, so we actually work with samples from the distribution. Historically, obtaining these samples has been the hardest part of Bayesian inference, but new tools like `Stan` have made it significantly easier.

Use of samples as opposed to the distribution itself is the heart of the *Monte Carlo* method.

For point estimation, we typically use the posterior mean or median. With sufficient samples from the posterior, the sample mean and the posterior mean will be essentially the same.

For set estimation, we simply construct a set that a given fraction of the posterior samples fall into.

Slides in the Computation with Stan section explain a bit more about how this is done. For now, we'll take it as a given.

Generalized Linear Mixed Effect Models

In "standard" linear regression, the error terms are considered to be *independent and identically distributed*.

$$y_i = \mathbf{x}_i^T \beta + \epsilon_i \qquad \epsilon_i \overset{\text{IID}}{\sim} \mathcal{N}(0, \sigma^2)$$

In "standard" linear regression, the error terms are considered to be *independent and identically distributed*.

$$y_i = \mathbf{x}_i^T \beta + \epsilon_i \qquad \epsilon_i \overset{\text{IID}}{\sim} \mathcal{N}(0, \sigma^2)$$

For many important problems, this is not reasonable. We may have repeated measurements of the same "unit," correlated errors, natural groupings that we want to capture in the data, *etc.*.

In "standard" linear regression, the error terms are considered to be *independent and identically distributed*.

$$y_i = \mathbf{x}_i^T \beta + \epsilon_i \qquad \epsilon_i \overset{\text{IID}}{\sim} \mathcal{N}(0, \sigma^2)$$

For many important problems, this is not reasonable. We may have repeated measurements of the same "unit," correlated errors, natural groupings that we want to capture in the data, *etc.*.

Mixed-Effects Models let us deal with all of these phenomena. Bayesians use these all of the time, but typically prefer the name "multilevel-" or "hierarchical-" models.

For mixed-effects models, data is typically arranged in a "hierarchy" of observational units: *e.g.*

Exams *within* students *within* classes *within* schools *within* districts *within* states

For mixed-effects models, data is typically arranged in a "hierarchy" of observational units: *e.g.*

Exams *within* students *within* classes *within* schools *within* districts *within* states

We model each grouping as an linear/additive effect:

$$\text{Score}_{ijklm} = \overbrace{\mu}^{\text{National Baseline}} + \overbrace{\nu_i}^{\text{State Effect}} + \overbrace{\kappa_{ij}}^{\text{District Effect}} + \cdots + \overbrace{\epsilon_{ijklm}}^{\text{Exam Specific Randomness}}$$

Notation can be a bit overwhelming, but key idea is that we are partitioning observed variance to different layers of the hierarchy

In `R`, mixed-effects models are typically fit with the `lme4` and `nlme` packages [PB00, BMBW15].

Fitting Mixed-Effects Models

In `R`, mixed-effects models are typically fit with the `lme4` and `nlme` packages [PB00, BMBW15].

A "drop-in" `Stan` based replacement is available through the `rstanarm` package [Sta16].

Fitting Mixed-Effects Models

In `R`, mixed-effects models are typically fit with the `lme4` and `nlme` packages [PB00, BMBW15].

A “drop-in” `Stan` based replacement is available through the `rstanarm` package [Sta16].

Let’s try this out an an example of tumor growth data from [HHW$^+$04] (example courtesy of Peter Hoff, [Hof09, Section 11.4]):

www.stat.washington.edu/people/pdhoff/Book/Data/XY.tumor

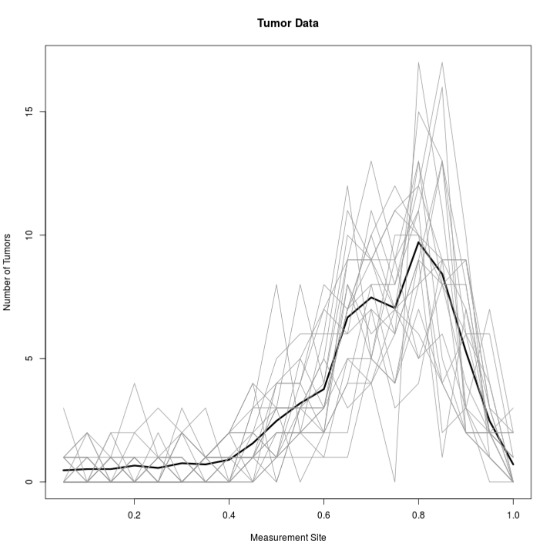
This data records the number of tumors found in different sections of the intestines of 21 different mice. The data follows a natural hierarchy (counts within mice) and suggests a mixed-effect model should be used to capture the "mouse effect".

If we look at the data, we see that certain mice have higher overall cancer incidence than others, but that there is still a consistent rise near the end of the intestine:

```
URL = "http://www.stat.washington.edu/people/pdhoff/Book/Data/data/XY.tumor"
TUMOR_DATA <- dget(URL)
plot(seq(0.05, 1, by=0.05), colMeans(TUMOR_DATA$Y), col="black", lwd=3,
     ylim=range(TUMOR_DATA$Y), type="l", main="Tumor Data",
     xlab="Measurement Site", ylab="Number of Tumors")

for(i in 1:21){
    lines(seq(0.05, 1, by=0.05), TUMOR_DATA$Y[i, ], col="grey80")
}
```

Tumor Data



Tumor Data
Number of Tumors
0
5
10
15
0.2
0.4
0.6
0.8
1.0
Measurement Site

Poisson GLMM

Since this is count data, we'll use Poisson regression.

The data is a bit messily formatted, but after some cleaning, we can fit the model with a mouse-effect and a 4th degree polynomial for the location effect using `rstanarm` as follows:

```
library(reshape2); library(rstanarm);
options(mc.cores=parallel::detectCores())
DATA <- cbind(expand.grid(mouse_id=1:21,
                          location=seq(0.05, 1, by=0.05)),
              count=melt(TUMOR_DATA$Y)$value)

M <- stan_glmer(count ~ poly(location, 4) + (1|mouse_id),
                family=poisson, data=DATA)
```

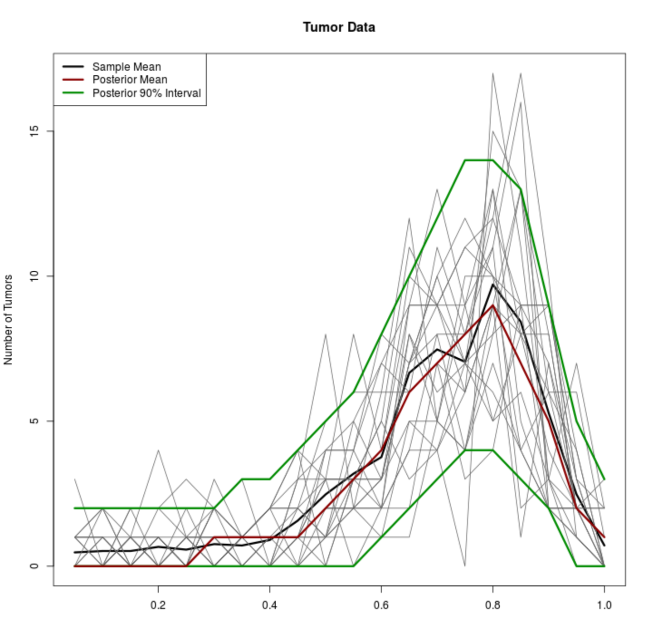
Takes about 15 seconds to get 4000 samples on this data set, which is more than enough for posterior inference.

Say we were interested in the prediction for the “typical” mouse. We make posterior predictions with the mouse-effect set to zero and examine the 5%, 50%, and 95% quantiles of the posterior (giving a 90% prediction interval):

```
X <- seq(0.05, 1, by=0.05)
PP <- posterior_predict(M, re.form=~0, newdata=data.frame(location=X))
apply(PP, 2, quantile, c(0.05, 0.50, 0.95))
```

The model appears to fit the data well:

Poisson GLMM



MCMC Convergence and Model Assessment

Checking Model Fit

In our Poisson GLMM example, the model fit the data well and sampling converged rapidly. We are not always so lucky. In general, we should always check convergence diagnostics before proceeding to inference.

In our Poisson GLMM example, the model fit the data well and sampling converged rapidly. We are not always so lucky. In general, we should always check convergence diagnostics before proceeding to inference.

The `shinystan` and `bayesplot` packages provide tools for visualizing and checking the results of MCMC inference.

Demo Time!

```
launch_shinystan(M)
```

Computation with Stan

Stan

Stan [Sta15c, CGH+, GLG15] is a probabilistic programming language superficially like `BUGS` or `JAGS` [LJB+03, Plu03]. Unlike BUGS and JAGS, not restricted to Gibbs sampling or conjugate (exponential family graphical) models.

Provides:

- Full Bayesian Inference (via Hamiltonian Monte Carlo)
- Variational Bayesian Inference (via ADVI [KRGB15, KTR+16, BKM16])
- Penalized MLE (Bayesian MAP)[6]

Best thought of as a DSL for specifying a distribution and sampling from it.

Named after *Stanislaw Ulam*, co-author, with N. Metropolis, of *The Monte Carlo Method* (JASA-1949, [MU49])

[6] Useful for quickly checking results against non-Bayesian software. MAP with uniform priors should recover the MLE (modulo optimization issues for non-convex problems).

Language- and platform-agnostic back-end ([Sta15c, Sta15d]) with a wide range of front-ends:

- Shell (“`CmdStan`”)
- `R` (“`RStan`”, [Sta15b]) ⋆
- `Python` (“`PyStan`”, [Sta15a]) ⋆
- MATLAB (“`MatlabStan`”, [Lau15])
- `Stata` (“`StataStan`”, [GS15])
- `Julia` (“`JuliaStan`”, [Goe15])

⋆: In process interface. The others “simply” wrap `CmdStan`.

`Stan` uses a two step compilation process: the user writes a model in pure `Stan` code[7] which is then translated to `C++` by the `stanc` compiler. The translated program is then compiled like any other `C++` program into a stand alone binary.

Once compiled, the model can be re-run with different inputs / parameters.

Requires a working `C++` compiler unlike the (interpreted) BUGS/JAGS to compile new models. Once compiled, the binary can be moved between machines (*modulo* standard linking and library constraints).

Higher level interfaces (*e.g.* `RStan`) can run the compilation in the background.

[7] It is possible to embed `Stan` directly within a `C++` program, but more advanced.

`Stan` provides a wide range of built-in data types:

- Data primitives: `real`, `int`
- Mathematical structures: `vector`, `matrix` – can hold `real` and `int`
- Programming structures: `array` – can hold *any* other `Stan` type
- Constrained structures: `ordered`, `positive_ordered`, `simplex`, `unit_vector`
- Matrix types: `cov_matrix`, `corr_matrix`, `cholesky_factor_cov`, `cholesky_factor_corr`

Constraints on data types are used to transform into an *unconstrained space* where `Stan` performs inference.

```
real<lower=0> sigma;
real<lower=0,upper=1> p;
```

`sigma` is log-transformed to be supported on $\mathbb{R}$; similarly `p` is logit^{-1}-transformed.[8] Since there is an change of variables in these transforms, `Stan` automatically adds a Jacobian to the target density. When you perform similar "left-hand-side" transformations, `Stan` will warn that a manual Jacobian adjustment may be necessary [Sta15d, Chapter 20].

Warning: Because `Stan` works on a (transformed) $\mathbb{R}$, *discrete parameters* are not directly supported. (Discrete data is fine.)

[8] `Stan` has a range of transformations into unconstrained space:

- Positivity constraints use a $\log(\cdot)$-transform
- Boundedness constraints use a (scaled) $\text{logit}(\cdot)$-transform
- Simplex constraints use a stick-breaking transform ($\mathbb{R}^K \to \mathbb{R}^{K-1}$)
- Matrix constraints (PD) use Cholesky-based transforms (see [PB96])

A `Stan` program is divided into *blocks*. The key blocks are:

- `data`: Defines the external data which `Stan` will read at the beginning of execution
- `parameters`: Defines the variables which will be inferred
- `model`: Defines the probability model relating the data and parameters. Both the prior and the likelihood are coded in this block

Additional blocks, *e.g.*, `transformed data`, `generated quantities` are useful for performing additional transformations within `Stan`. Less useful when using `Stan` through the interfaces.

Toy example (Beta-Bernoulli):

```
data{
    int<lower=0> N; // Number of observations
    int<lower=0,upper=1> y[N]; // observed 0/1 variables
}
parameters{
    real<lower=0,upper=1> p; // unknown p
}
model{
    p ~ beta(1, 1); // weak prior
    y ~ bernoulli(p); // vectorized across elements of y
}
```

Stan Language Model

The “sampling statements” in the model block are syntactic sugar for direct manipulation of the log-posterior.

Equivalent:

```
data{
    int<lower=0> N; // Number of observations
    int<lower=0,upper=1> y[N]; // observed 0/1 variables
}
parameters{
    real<lower=0,upper=1> p; // unknown p
}
model{
    target += beta_log(p, 1, 1); // weak prior
    for(i in 1:N){ // likelihood
        target += bernoulli_log(p, y[i]);
    }
}
```

$$\pi(\theta|X) = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)\,\mathrm{d}\theta}$$

The denominator of this quantity (the “partition function”) is often analytically intractable so we are left with

$$\pi(\theta|X) \propto p(X|\theta)p(\theta)$$

How can we calculate $\mathbb{E}[F(\theta)]$ for arbitrary (measurable) $F(\cdot)$ when we can only calculate π up to a normalizing constant?

In theory, we *sample* from π and invoke the Law of Large Numbers:

$$\frac{1}{N}\sum_{i=1}^{N} F(\theta_i) \overset{n\to\infty}{\longrightarrow} \mathbb{E}[F(\theta)]$$

In practice, we cannot sample directly from π either.

Not entirely true. We can (sort of) sample from π, but not IID.

We construct an (ergodic) Markov chain with transition kernel Π chosen to have the same stationary distribution as π (see, *e.g.*, [Tie94] for details). Then, samples from this Markov chain are samples from π if either:

- We initialize the chain with a draw from π;
- We run the chain long enough (infinitely long!) so that it converges to π.

The first is, again, impossible. Let's look more closely at the second.

A Markov chain is a map from the space of probability distributions onto itself.

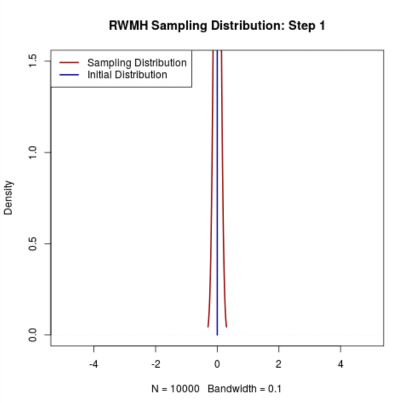
Given an initialization distribution (which may be a point mass) P_0, application of the transition kernel Π gives a new distribution P_1.

Repeated application gives $\{P_n\}$ which tend to π as $n \to \infty$. If P_0 is "close" to π, the convergence is rapid.

π is the stationary point of Π so $\Pi\pi = \pi$.

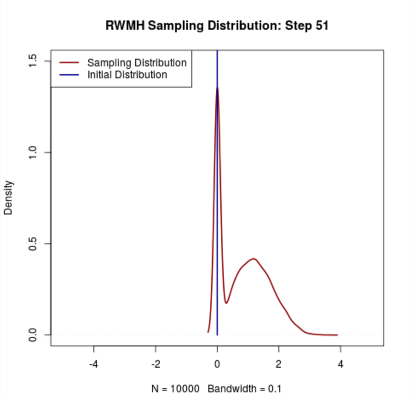
Example:

- $P_0 = \delta_0$;
- Π is a Random Walk Metropolis Hastings update (proposal distribution: $X_t \sim \mathcal{N}(X_{t-1}, 1)$);
- π is $\mathcal{N}(2, 5^2)$.



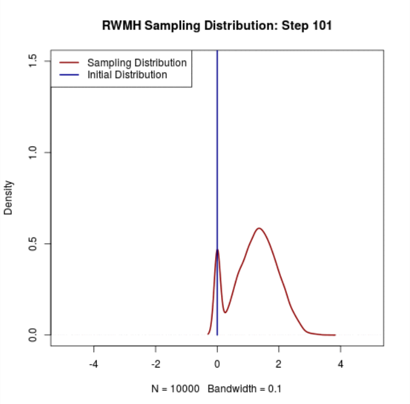
Example:

- $P_0 = \delta_0$;
- Π is a Random Walk Metropolis Hastings update (proposal distribution: $X_t \sim \mathcal{N}(X_{t-1}, 1)$);
- π is $\mathcal{N}(2, 5^2)$.



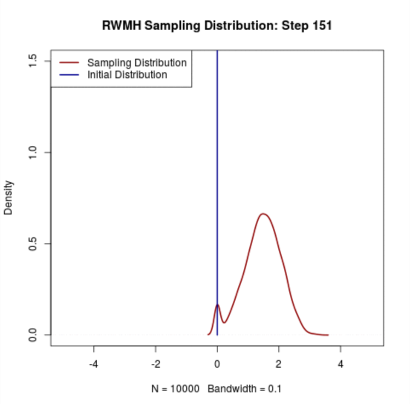
Example:

- $P_0 = \delta_0$;
- Π is a Random Walk Metropolis Hastings update (proposal distribution: $X_t \sim \mathcal{N}(X_{t-1}, 1)$);
- π is $\mathcal{N}(2, 5^2)$.



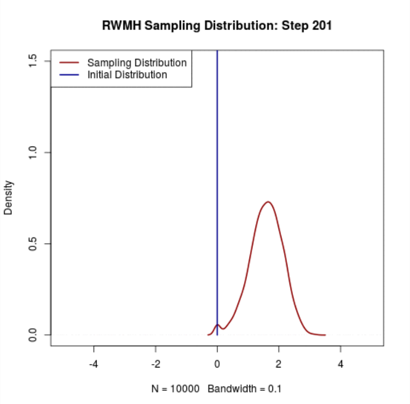
Example:

- $P_0 = \delta_0$;
- Π is a Random Walk Metropolis Hastings update (proposal distribution: $X_t \sim \mathcal{N}(X_{t-1}, 1)$);
- π is $\mathcal{N}(2, 5^2)$.



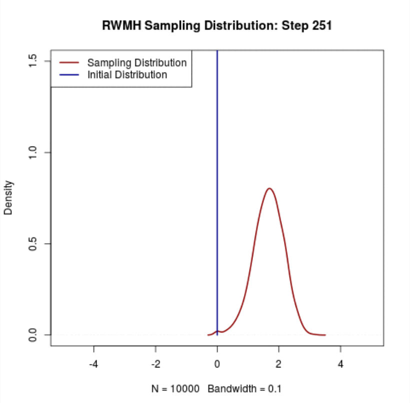
Example:

- $P_0 = \delta_0$;
- Π is a Random Walk Metropolis Hastings update (proposal distribution: $X_t \sim \mathcal{N}(X_{t-1}, 1)$);
- π is $\mathcal{N}(2, 5^2)$.



Example:

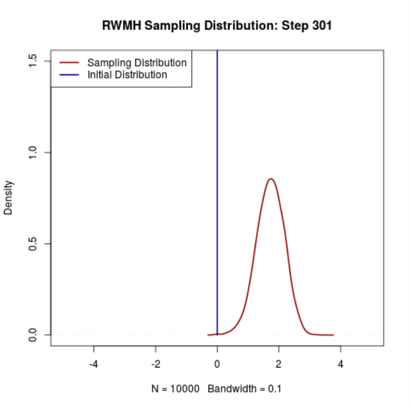
- $P_0 = \delta_0$;
- Π is a Random Walk Metropolis Hastings update (proposal distribution: $X_t \sim \mathcal{N}(X_{t-1}, 1)$);
- π is $\mathcal{N}(2, 5^2)$.



Convergence of MCMC

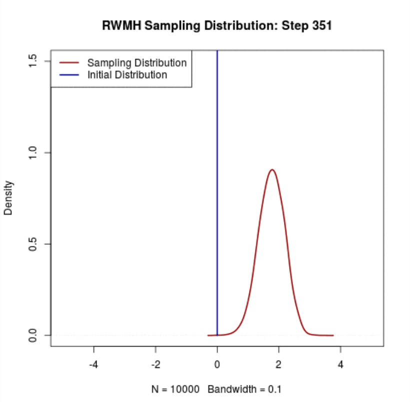
Example:

- $P_0 = \delta_0$;
- Π is a Random Walk Metropolis Hastings update (proposal distribution: $X_t \sim \mathcal{N}(X_{t-1}, 1)$);
- π is $\mathcal{N}(2, 5^2)$.



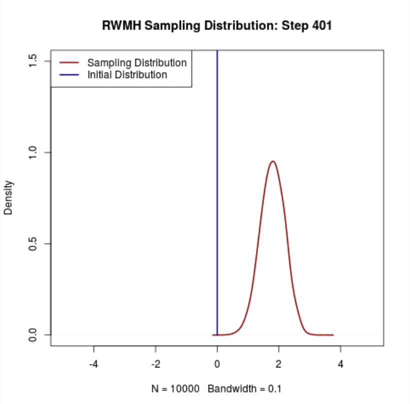
Example:

- $P_0 = \delta_0$;
- Π is a Random Walk Metropolis Hastings update (proposal distribution: $X_t \sim \mathcal{N}(X_{t-1}, 1)$);
- π is $\mathcal{N}(2, 5^2)$.



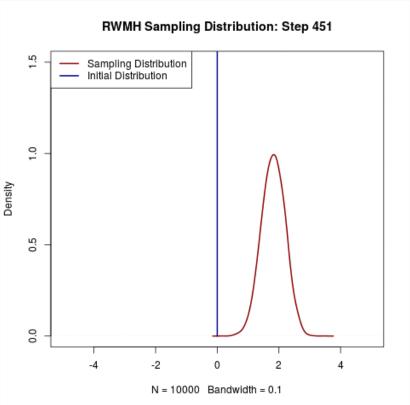
Example:

- $P_0 = \delta_0$;
- Π is a Random Walk Metropolis Hastings update (proposal distribution: $X_t \sim \mathcal{N}(X_{t-1}, 1)$);
- π is $\mathcal{N}(2, 5^2)$.



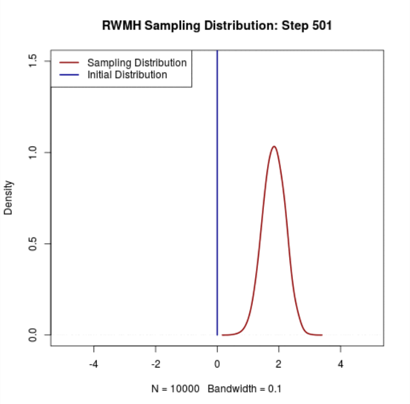
Example:

- $P_0 = \delta_0$;
- Π is a Random Walk Metropolis Hastings update (proposal distribution: $X_t \sim \mathcal{N}(X_{t-1}, 1)$);
- π is $\mathcal{N}(2, 5^2)$.



Example:

- $P_0 = \delta_0$;
- Π is a Random Walk Metropolis Hastings update (proposal distribution: $X_t \sim \mathcal{N}(X_{t-1}, 1)$);
- π is $\mathcal{N}(2, 5^2)$.



How many samples from π do we need?

Depends what we want to do: let's take calculating the posterior mean as a typical task.[9]

Two possible sources of variance:

- The inherent variance of the posterior;
- Additional variance from having a finite number of draws ("Monte Carlo error")

The first is unavoidable (and a key advantage of the Bayesian approach); the second can be reduced with more samples.

[9] See [Jon04] for a discussion of the *Markov Chain Central Limit Theorem*; see [RR04] for a discussion of the general conditions required for MCMC convergence.

Accuracy of MCMC: Effective Sample Size

If we have n IID samples from π, our Monte Carlo standard error (ratio of total variance to inherent variance) is approximately $\sqrt{1+n^{-1}}$ [GCS+14, Section 10.5].

With autocorrelation, we instead look at the *effective sample size*:[10]

$$\text{ESS} = \frac{n}{1 + \sum_{t=1}^{\infty} \rho_t}$$

See [GCS+14, Section 11.5] or [KCGN98, Section 2.9] for details.

Technically, there is a different ACF for each $\mathbb{E}[f(\cdot)]$ we estimate, but this isn't usually a big deal in practice.[11]

---

[10] The exact formula has $n^2/(n + \sum_{t=1}^{n}(n-t)\rho_t)$ but for large n this is approximately equal (and faster to calculate).

[11] There is a disconnect between practice and theory here. Theory establishes conditions for accurate inference for *all* possible f (see, *e.g.*, [LPW08]), but we usually only care about a few f. Some (very) recent work attempts to establish convergence rates for restricted classes of f [RRJW16].

Choice of Markov Kernel

Since ESS controls the quality of our inference, ESS/second is the appropriate metric for comparing samplers.

Different choices of the Markov transition kernel Π can give radically different ESS/second.

Standard Metropolis-Hastings or Gibbs Samplers struggle for complex (hierarchical) and high-dimensional (many parameters) models.

Hamiltonian Monte Carlo ([Nea11]) performs much more efficiently for a wide range of problems.[12]

[12] The Markov Chains constructed by HMC can be shown to be *geometrically ergodic* (quick mixing) under relatively weak conditions [LBBG16].

In its default mode, `Stan` uses a technique known as *Hamiltonian Monte Carlo* to sample from the posterior with minimal autocorrelation. These draws are typically more expensive than from other methods, *e.g.* Gibbs samplers, but the reduced correlation leads to a (much) higher ESS/second.

Very roughly: Metropolis-Hastings methods ([MRR+53, Has70]) move around the probability space randomly (without knowledge of the underlying geometry) and use a accept-reject step to adjust probabilities accordingly.

Hamiltonian Monte Carlo gives a particle a random "kick" and samples based on the path of the particle: uses *Hamiltonian mechanics* to simulate the path of the particle in an energy field induced by the target density π.

Hamiltonian dynamics (*a.k.a.* Hamiltonian mechanics) is an equivalent formulation of Newtonian mechanics. Let $\boldsymbol{p}$ be the momenta of all particles in the system and $\boldsymbol{q}$ be the position of the particles.

The evolution of the system over time is uniquely defined by:

$$\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{q}}$$
$$\frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t} = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{p}}$$

where $\mathcal{H}$ is the *Hamiltonian* of the system, a function which measures the total energy of the system.

Hamiltonian mechanics is easily extended to relativistic systems.

Once the Hamiltonian $\mathcal{H}$ and the initial conditions are specified, the time evolution of the system is known deterministically. In practice, the PDEs cannot be solved explicitly and a numerical integrator must be used.

A common choice of integrator is the *leapfrog integrator* which has the nice properties of being:

- time-reversibility
- symplectic (energy conserving)

See [LR05] for details. With step size ϵ (requiring $L\epsilon$ steps to integrate over a time interval of length L), the leapfrog integrator has ϵ^2 error.

Hamiltonian Monte Carlo (originally *Hybrid* Monte Carlo) [Nea11] builds a Hamiltonian system to sample from a target density π.

We let $\boldsymbol{q}$ (the "position") be the parameters of the target density and add an auxiliary momentum variable $\boldsymbol{p}$. The joint distribution of $\boldsymbol{p}, \boldsymbol{q}$ can be used to define a Hamiltonian $\mathcal{H}$:

$$\begin{aligned} H(\boldsymbol{p}, \boldsymbol{q}) &= -\log p(\boldsymbol{p}, \boldsymbol{q}) \\ &= -\log p(\boldsymbol{q}|\boldsymbol{p}) - \log p(\boldsymbol{p}) \\ &= \underbrace{T(\boldsymbol{q}|\boldsymbol{p})}_{\text{Kinetic energy}} + \underbrace{V(\boldsymbol{p})}_{\text{Potential energy}} \end{aligned}$$

Solving the Hamiltonian equations for this system, we find

$$\begin{aligned} \frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t} &= \frac{\partial T}{\partial \boldsymbol{p}} \\ \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} &= -\frac{\partial V}{\partial \boldsymbol{q}} \end{aligned}$$

We can (approximately) solve these equations using a leapfrog integrator. To introduce randomness, $\boldsymbol{p}_0$ is initialized from a $\mathcal{N}(0, \Sigma)$ matrix where Σ is independent of prior samples and of $\boldsymbol{q}_0$.

Leapfrog integration is then simply:

$$\rho \leftarrow \rho - \frac{\epsilon}{2} \frac{\partial V}{\partial \boldsymbol{q}}$$

$$\theta \leftarrow \theta + \epsilon \Sigma \boldsymbol{p}$$

$$\rho \leftarrow \rho - \frac{\epsilon}{2} \frac{\partial V}{\partial \boldsymbol{q}}$$

repeated L times.

If leapfrog integration were exact, we could directly accept the result of the leapfrog step. In reality, we have to use a Metropolis acceptance step [MRR+53] to account for the error. Thus, HMC *as implemented* is strictly a form of Metropolis MCMC, but with a *highly efficient* transition kernel Π which moves along the geometric contours of the target distribution π rather than randomly.

The form of Σ controls the implicit geometry of the the Hamiltonian dynamics [BS11, BBLG14]. In particular, Σ^{-1} is the mass matrix of the particle undergoing Hamiltonian evolution.

Fixed Σ (either diagonal or full) corresponds to a Euclidean metric on the parameter space and gives *Euclidean Hamiltonian Monte Carlo*.

Current research considers changing Σ for each sample: this corresponds to sampling on a Riemannian manifold and gives rise to *Riemannian Hamiltonian Monte Carlo* [GC11, Bet13]. By varying Σ, RHMC can adapt to the “funnels” found in complex hierarchical variables more efficiently [BG13].

The No-U-Turn Sampler (NUTS)

For large L, running HMC to termination is wasteful when the particle begins to retrace its steps. Early termination would save computation time but biases our sampling.

To avoid this, the *No-U-Turn Sampler* ("NUTS") enhances HMC by allowing time to intermittently run backwards: see [HG14] for details. The time-reversability of the leapfrog integrator is key for allowing NUTS to work properly.

NUTS is the default sampler used in `Stan` though "pure" HMC is also available.

Automatic Differentiation (“AutoDiff”) is a technique for automatically calculating the numerical gradient of a function at a fixed point.

AutoDiff expresses computations in terms of language primitives (addition, multiplication, and function calls) and uses the chain rule to calculate the gradient as part of regular function evaluation.

`Stan` uses autodiff to efficiently calculate the gradients necessary for HMC.

AutoDiff vs Other Gradient Calculation Techniques

AutoDiff is not

- Symbolic Differentiation
- Numerical Differentiation (finite difference approximations)

Unlike symbolic differentiation, AutoDiff has no knowledge about the function being evaluated: only the arithmetic primitives.[13] Unlike numerical differentiation, AutoDiff provides exact gradient calculations with a single function call.

[13] Consequently, AutoDiff provides an exact derivative for an *approximation* of the function of interest rather than an approximation to the exact function of interest.

`Stan` provides a fully AutoDiff-equipped math library ([CHB+15]) built on Boost and Eigen [Sch11, GJ+10].

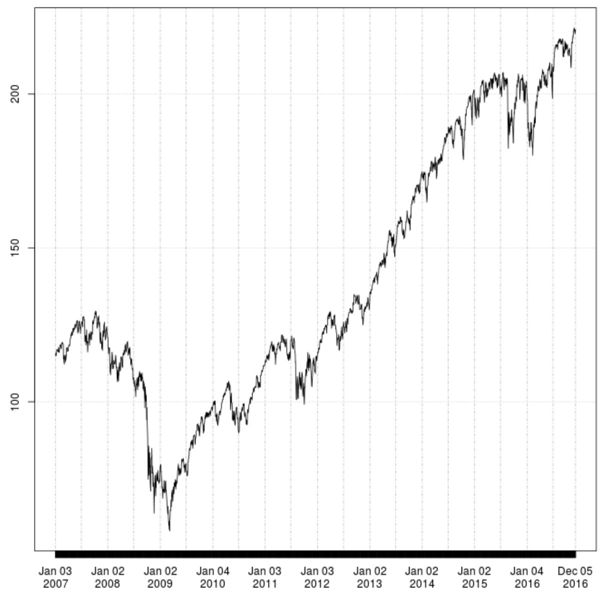
Currently `Stan` only uses *first-order* AutoDiff but *second-order* AutoDiff will be released soon.

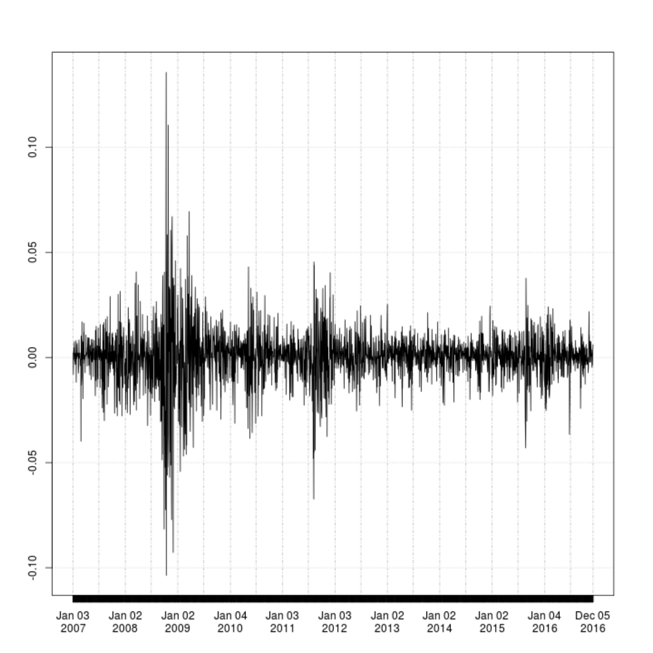
`Stan`'s AutoDiff is *reverse-mode* which means that it works "down" the function call chain:

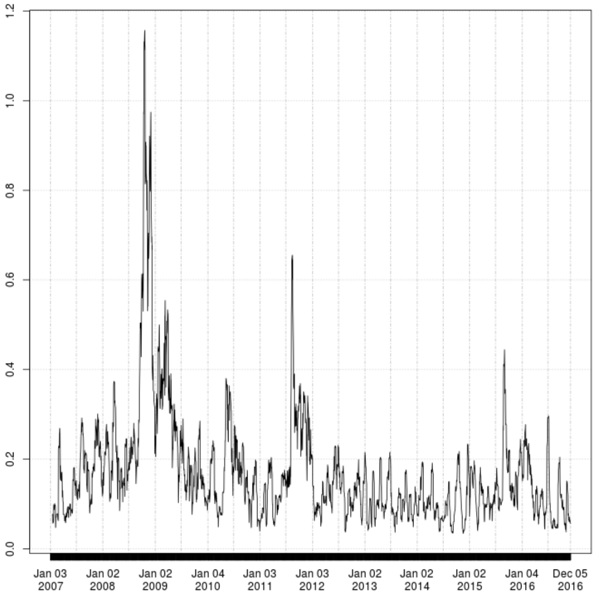
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x} = \frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\frac{\partial w_1}{\partial x} = \dots$$

When computing derivatives of functions $f : \mathbb{R}^n \to \mathbb{R}^m$, this is more efficient for $m \ll n$; for `Stan`, $m = 1$.

Financial Time Series: Stochastic Volatility Models


200
150
100
Jan 03 2007
Jan 02 2008
Jan 02 2009
Jan 04 2010
Jan 03 2011
Jan 03 2012
Jan 02 2013
Jan 02 2014
Jan 02 2015
Jan 04 2016
Dec 05 2016


0.10
0.05
0.00
-0.05
-0.10
Jan 03 2007
Jan 02 2008
Jan 02 2009
Jan 04 2010
Jan 03 2011
Jan 03 2012
Jan 02 2013
Jan 02 2014
Jan 02 2015
Jan 04 2016
Dec 05 2016



1.2
1.0
0.8
0.6
0.4
0.2
0.0
Jan 03 2007
Jan 02 2008
Jan 02 2009
Jan 04 2010
Jan 03 2011
Jan 03 2012
Jan 02 2013
Jan 02 2014
Jan 02 2015
Jan 04 2016
Dec 05 2016

Financial time series (*e.g.* stock market returns) exhibit *volatility clustering* – that is, there are periods of relative calm (small day-over-day changes) and periods of relative volatility (large day-over-day changes).

Financial time series (*e.g.* stock market returns) exhibit *volatility clustering* – that is, there are periods of relative calm (small day-over-day changes) and periods of relative volatility (large day-over-day changes).

A standard simple model of a financial time series is:

$$r_t \overset{\text{IID}}{\sim} \mathcal{N}(0, \sigma_t^2)$$

where σ_t^2 is the "instantaneous volatility."

Financial time series (*e.g.* stock market returns) exhibit *volatility clustering* – that is, there are periods of relative calm (small day-over-day changes) and periods of relative volatility (large day-over-day changes).

A standard simple model of a financial time series is:

$$r_t \overset{\text{IID}}{\sim} \mathcal{N}(0, \sigma_t^2)$$

where σ_t^2 is the "instantaneous volatility."

We would like to know the instantaneous volatility for each day, but it's essentially impossible to calculate without further assumptions (or higher frequency data) since we can't calculate a standard deviation with only a single sample.

Financial time series (*e.g.* stock market returns) exhibit *volatility clustering* – that is, there are periods of relative calm (small day-over-day changes) and periods of relative volatility (large day-over-day changes).

A standard simple model of a financial time series is:

$$r_t \overset{\text{IID}}{\sim} \mathcal{N}(0, \sigma_t^2)$$

where σ_t^2 is the "instantaneous volatility."

We would like to know the instantaneous volatility for each day, but it's essentially impossible to calculate without further assumptions (or higher frequency data) since we can't calculate a standard deviation with only a single sample.

Volatility models add additional structure to the time-dynamics of σ_t^2 so that we can estimate it from observed data.

The Stochastic Volatility model of [KSC98] models volatility as a latent mean-reverting AR(1) process.

$$h_{t+1} \sim \mathcal{N}\left(\mu + \phi(h_t - \mu), \sigma^2\right)$$
$$r_t \sim \mathcal{N}\left(0, \exp\{h_t\}\right)$$

Implemented in `stochvol` package for `R` [Kas16].

If we want the volatility at each time t, this is a *high-dimensional* model: quantities–$\{h_t\}_{t=1}^{T}, \mu, \phi, \sigma$–than we have observations.

Normally, this is impossible without further constraints or regularization, but the prior fulfills that role in the Bayesian context.

The Stan manual [Sta15d, Section 9.5] describes how to code this model efficiently:

```
data {
    int<lower=0> T;
    vector[T] y;
}
parameters {
    real mu;
    real<lower=-1, upper=1> phi; // Stationary volatility
    real<lower=0> sigma;
    vector[T] h_std;
}
```

(continued)

```
transformed parameters {
    vector[T] h;
    h = h_std * sigma;
    h[1] = h[1] / sqrt(1 - phi * phi);
    h = h + mu;
    for(t in 2:T){
        h[t] = h[t] + phi * (h[t-1] - mu);
    }
}
```

```
model {
    // Priors
    phi ~ uniform(-1, 1);
    sigma ~ cauchy(0, 5);
    mu ~ cauchy(0, 10);
    // Scaled Innovations in h process are IID N(0,1)
    h_std ~ normal(0, 1);
    // Observation likelihood.
    // Note exp(h/2) since Stan uses normal(mean, SD)
    y ~ normal(0, exp(h/2));
}
```

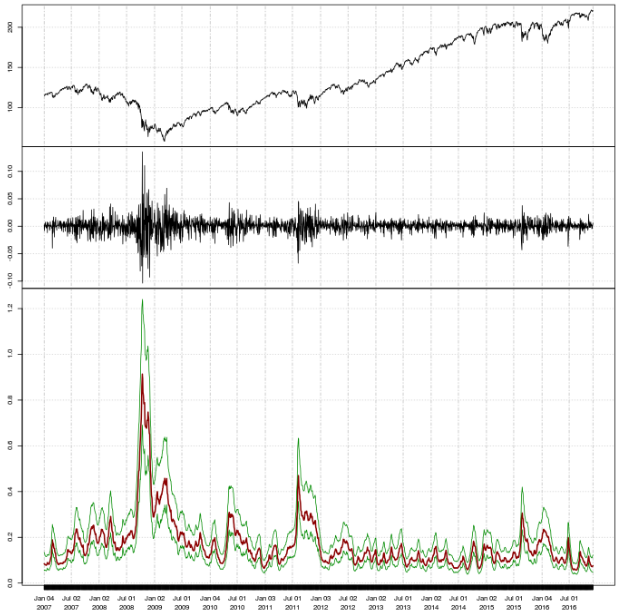
Running this model, we can plot the estimated volatility with its confidence interval over time:

```
library(quantmod)

SPY <- getSymbols("SPY", auto.assign=FALSE)
R <- na.omit(ROC(Ad(SPY)))

SAMPLES <- stan("sv.stan", data=list(y=as.vector(R), T=length(R)))
PP <- apply(exp(extract(SAMPLES, "h")[[1]]/2), 2,
            quantile, c(0.05, 0.50, 0.95))
```

SPY & Inferred Volatility



Once coded-up, adapting the model to use a heavy-tailed or skewed error process is straightforward:

t-errors (inferring the degrees of freedom)

```
...
    real<lower=0> nu;
...
    nu ~ cauchy(0, 5);
    y ~ student_t(nu, 0, exp(h/2));
...
```

Skew-normal errors (inferring the skewness parameter):

```
...
    real alpha;
...
    alpha ~ cauchy(0, 5);
    y ~ skew_normal(0, exp(h/2), alpha);
...
```

Thank you

If you're interested in learning more, start with Michael Betancourt's talks to the Tokyo Stan Users' Group: (Modeling (link) and HMC (link)).

Bob Carpenter's MLSS-2015 talk (link) is a bit more "hands-on" with the Stan language. (Michael's talks go into more MCMC and HMC theory)

The `Stan` manual (link) is remarkably readable.

The `stan-users` mailing list (link) is a good place to ask for help with more detailed issues.

Pierre-Antoine Kremp built a fully-open source political forecasting model using Stan. Check it out!

References

[ABNK+87] Shun-ichi Amari, O.E. Barndorff-Nielsen, Robert E. Kass, Steffen L. Lauritzen, and C.R. Rao.
Differential Geometry in Statistical Inference, volume 10 of *Lecture Notes-Monograph Series*.
Institute of Mathematical Statistics, 1987.
http://www.jstor.org/stable/4355557.

[BBLG14] Michael J. Betancourt, Simon Byrne, Samuel Livingstone, and Mark Girolami.
The geometric foundations of Hamiltonian Monte Carlo, 2014.
arXiv 1410.5110.

[BBS09] James O. Berger, José M. Bernardo, and Dongchu Sun.
The formal definition of reference priors.
Annals of Statistics, 37(2):905–938, 2009.
`https://projecteuclid.org/euclid.aos/1236693154`; arXiv 0904.0156.

[Bet13] Michael Betancourt.
A general metric for Riemannian manifold Hamiltonian Monte Carlo.
In Frank Nielsen and Frédéric Barbaresco, editors, *Geometric Science of Information: First International Conference (GSI 2013)*, volume 8085 of *Lecture Notes in Computer Science*, pages 327–334. Springer, 2013.
arXiv 1212.4693.

References III

[BG13] Michael Betancourt and Mark Girolami.
Hamiltonian Monte Carlo for hierarchical models, 2013.
arXiv 1312.0906.

[BKM16] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe.
Variational inference: A review for statisticians, 2016.
arXiv 1601.00670.

[BMBW15] Douglas Bates, Martin Mächler, Ben Bolker, and Steve Walker.
Fitting linear mixed-effects models using lme4.
Journal of Statistical Software, 67, 2015.
`https://www.jstatsoft.org/article/view/v067i01`.

[BS11] Michael Betancourt and Leo C. Stein.
The geometry of Hamiltonian Monte Carlo, 2011.
arXiv 1112.4118.

[CGH+] Bob Carpenter, Andrew Gelman, Matt Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell.
Stan: A probabilistic programming language.
Journal of Statistical Software.
Forthcoming; preprint available at
`http://www.stat.columbia.edu/~gelman/research/published/stan-paper-revision-feb2015.pdf`.

[CHB+15] Bob Carpenter, Matthew D. Hoffman, Marcus Brubaker, Daniel Lee, Peter Li, and Michael Betancourt.
The Stan math library: Reverse-mode automatic differentiation in C++, 2015.
arXiv 1059.07164.

[DL12] Marie Davidian and Thomas A. Louis.
Why statistics?
Science, 336(6077):12, 2012.

[Efr15] Bradley Efron.
Frequentist accuracy of bayesian estimates.
Journal of the Royal Statistical Society, Series B, 77(3):617–646, 2015.
Discussion at `https://www.youtube.com/watch?v=2oKw5HHAWs4`.

[GC11] Mark Girolami and Ben Calderhead.
Riemann manifold langevin and hamiltonian monte carlo methods.
Journal of the Royal Statistical Society, Series B, 73(2):123–214, 2011.

References VI

[GCS+14] Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin.
Bayesian Data Analysis.
Texts in Statistical Science. CRC Press, 3rd edition, 2014.

[GH06] Andrew Gelman and Jennifer Hill.
Data Analysis Using Regression and Multilevel/Hierarchical Models.
Analytical Methods for Social Research. Cambridge University Press, 1st edition, 2006.

[GJ+10] Gaël Guennebaud, Benoît Jacob, et al.
Eigen v3, 2010.
`http://eigen.tuxfamily.org`.

[GLG15] Andrew Gelman, Daniel Lee, and Jiqiang Guo.
Stan: A probabilistic programming language for Bayesian inference and optimization.
Journal of Educational and Behavioral Statistics, 40:530–543, 2015.
http://www.stat.columbia.edu/~gelman/research/published/stan_jebs_2.pdf.

[Goe15] Rob Goedman.
Stan.jl: the Julia interface to Stan, 2015.
http://mc-stan.org/julia-stan.html;
http://gitub.com/goedman/Stan.jl.

[GS15] Robert Grant and Stan Development Team.
StatStan: the Stata interface to Stan, 2015.
`http://mc-stan.org/stata-stan.html`;
`http://gitub.com/stan-dev/statastan`.

[Has70] W.K. Hastings.
Monte Carlo sampling methods using Markov chains and their applications.
Biometrika, 57:97–109, April 1970.
http://www.jstor.org/stable/2334940.

[HC96] James P. Hobert and George Casella.
The effect of improper priors on gibbs sampling in hierarchical linear mixed models.
Journal of the American Statistical Association, 91(436):1461–1473, 1996.
http://www.jstor.org/stable/2291572.

[HG14] Matthew D. Hoffman and Andrew Gelman.
The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo.
Journal of Machine Learning Research, 15:1593–1623, 2014.
http://jmlr.org/papers/v15/hoffman14a.html.

[HHW+04] Kevin M. Haigis, Peter D. Hoff, Alanna White, Alex R. Shoemaker, Richard B. Halberg, and William F. Dove. **Tumor regionality in the mouse intestine reflects the mechanism of loss of Apc function.** *Proceedings of the National Academy of Sciences of the United States of America*, 101(26):9769–9773, 2004.

[Hof09] Peter D. Hoff. *A First Course in Bayesian Statistical Methods.* Springer Texts in Statistics. Springer, 1st edition, 2009.

[Jay03] E.T. Jaynes. *Probability Theory: The Logic of Science.* Cambridge University Press, 2003.

[Jef61] Harold Jeffreys.
Theory of Probability.
Ofxford University Press, 3rd edition, 1961.

[Jon04] Galin L. Jones.
On the Markov chain central limit theorem.
Probability Surveys, 1:299–320, 2004.

[Kas16] Gregor Kastner.
Dealing with stochastic volatility in time series using the r package stochvol.
69(5), 2016.
https://www.jstatsoft.org/article/view/v069i05.

[KCGN98] Robert E. Kass, Bradley P. Carlin, Andrew Gelman, and Radford M. Neal.
Markov chain Monte Carlo in practice: A roundtable discussion.
The Americal Statistician, 52:93–100, 1998.
`http://dx.doi.org/10.1080/00031305.1998.10480547`.

[Key21] John Maynard Keynes.
A Treatise on Probability.
Macmillan & Co., 1921.

[KRGB15] Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David M. Blei.
Automatic variational inference in Stan, 2015.
arXiv 1506.03431.

[KSC98] Sangjoon Kim, Neil Shephard, and Siddhartha Chib.
Stochastic volatility: Likelihood inference and comparison with ARCH models.
The Review of Economic Studies, 65(3):361–393, 1998.
`http://www.jstor.org/stable/2566931`.

[KTR+16] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei.
Automatic differentiation variational inference, 2016.
arXiv 1603.00788.

[KW96] Robert E. Kass and Larry Wasserman.
The selection of prior distributions by formal rules.
Journal of the American Statistical Association, 91(435):1343–1370, 1996.
`http://www.jstor.org/stable/2291752`.

[Lau15] Brian Lau.
MatlabStan: the Matlab interface to Stan, 2015.
http://mc-stan.org/matlab-stan.html;
http://gitub.com/brian-lau/MatlabStan.

[LBBG16] Samuel Livingstone, Michael Betancourt, Simon Byrne, and Mark Girolami.
On the geometric ergodicity of hamiltonian monte carlo, 2016.
arXiv 1601.08057.

[LJB+03] David Lunn, Christopher Jackson, Nicky Best, Andrew Thomas, and David Spiegelhalter.
The BUGS Book: A Practical Introduction to Bayesian Analysis.
Texts in Statistical Science. CRC Press, 1st edition, 2003.

[LPW08] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer.
Markov Chains and Mixing Times.
American Mathematical Society, 1st edition, 2008.
http://research.microsoft.com/en-us/um/people/peres/markovmixing.pdf.

[LR05] Benedict Leimkuhler and Sebastian Reich.
Simulating Hamiltonian Dynamics.
Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2005.

References XVI

[McE15] Richard McElreath.
Statistical Rethinking: A Bayesian Course with Examples in R and Stan.
Texts in Statistical Science. CRC Press, 1st edition, 2015.
http://xcelab.net/rm/statistical-rethinking/; Early draft available at
http://xcelab.net/rmpubs/rethinking/bookOLD.pdf.

[MRR+53] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller.
Equation of state calculations by fast computing machines.
Journal of Chemical Physics, 21:1087–1092, 1953.
http://scitation.aip.org/content/aip/journal/jcp/21/6/10.1063/1.1699114.

[MU49] Nicholas Metropolis and Stanislaw Ulam.
The Monte Carlo method.
Journal of the American Statistical Association, 44:335–341, 1949.
http://www.jstor.org/stable/2280232.

[Nea11] Radford M. Neal.
MCMC using Hamiltonian dynamics.
Handbooks of Modern Statistical Methods, chapter 5, pages 113–162. Chapman & Hall/CRC, 1st edition, 2011.
http://www.mcmchandbook.net/HandbookChapter5.pdf; arXiv 1206.1901.

[PB96] Jose C. Pinheiro and Douglas M. Bates.
Unconstrained parametrizations for variance-covariance matrices.
Statistics and Computing, 6:289–296, 1996.

[PB00] Jose C. Pinheiro and Douglas M. Bates.
Mixed-Effects Models in S and S-PLUS.
Statistics and Computing. Springer, 1st edition, 2000.

[Plu03] Martyn Plummer.
JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling, 2003.
`http://mcmc-jags.sourceforge.net/`.

[Ram31] Frank P. Ramsey.
The Foundations of Mathematics and Other Logical Essays.
1931.

[Rob07] Christian Robert.
The Bayesian Choice: From Decision Theoretic Foundations to Computational Implementatin.
Springer Texts in Statistics. Springer, 2nd edition, 2007.

[RR04] Gareth O. Roberts and Jeffrey S. Rosenthal.
General state space Markov chains and MCMC algorithms.
Probability Surveys, 1:20–71, 2004.

[RRJW16] Maxim Rabinovich, Aaditya Ramdas, Michael I. Jordan, and Martin J. Wainwright.
Function-specific mixing times and concentration away from equilibrium, 2016.
arXiv 1605.02077.

[Sav54] Leonard J. Savage.
Foundations of Statistics.
1954.

[Sch11] Boris Schling.
The Boost C++ Libraries.
XML Press, 2011.
http://boost.org.

[Sta15a] Stan Development Team.
PyStan: the python interface to Stan, version 2.12.0, 2015.
http://mc-stan.org/pystan.html; https://pypi.python.org/pypi/pystan.

[Sta15b] Stan Development Team.
rstan: the R interface to Stan, version 2.12.0, 2015.
http://mc-stan.org/rstan.html; https://cran.r-project.org/web/packages/rstan/index.html.

[Sta15c] Stan Development Team.
Stan: A C++ library for probability and sampling, version 2.12.0, 2015.
http://mc-stan.org/.

[Sta15d] Stan Development Team.
Stan Modeling Language Users Guide and Reference Manual, Version 2.13.1, 2015.
http://mc-stan.org.

[Sta16] Stan Development Team.
rstanarm: Bayesian applied regression modeling via Stan, 2016.
https://cran.r-project.org/web/packages/rstanarm/index.html.

[SV01] Glenn Shafer and Vladimir Vovk.
Probability and Finance: It's Only a Game!
Wiley Series in Probability and Statistics. Wiley, 2001.

[Tie94] Luke Tierney.
Markov chains for exploring posterior distributions.
Annals of Statistics, 22(4):1701–1728, 1994.
https://projecteuclid.org/euclid.aos/1176325750.