Modern Bayesian Tools for Time Series Analysis

R in Finance 2016

Thomas P. Harte & R. Michael Weylandt

2016-05-20

Godolphin Capital Management & Rice University

Introduction

Disclaimers

Thomas P. Harte and R. Michael Weylandt ("the Authors") are providing this presentation and its contents ("the Content") for educational purposes only at the *R in Finance Conference*, 2016-05-20, Chicago, IL. Neither of the Authors is a registered investment advisor and neither purports to offer investment advice nor business advice.

You may use any of the Content under the terms of the MIT License (see below).

The Content is provided for informational and educational purposes only and should not be construed as investment or business advice. Accordingly, you should not rely on the Content in making any investment or business decision. Rather, you should use the Content only as a starting point for doing additional independent research in order to allow you to form your own opinion regarding investment or business decisions. You are encouraged to seek independent advice from a competent professional person if you require legal, financial, tax or other expert assistance.

The Content may contain factual or typographical errors: the Content should in no way be construed as a replacement for qualified, professional advice. There is no guarantee that use of the Content will be profitable. Equally, there is no guarantee that use of the Content will not result in losses.

THE AUTHORS SPECIFICALLY DISCLAIM ANY PERSONAL LIABILITY, LOSS OR RISK INCURRED AS A CONSEQUENCE OF THE USE AND APPLICATION, EITHER DIRECTLY OR INDIRECTLY, OF THE CONTENT. THE AUTHORS SPECIFICALLY DISCLAIM ANY REPRESENTATION, WHETHER EXPLICIT OR IMPLIED, THAT APPLYING THE CONTENT WILL LEAD TO SIMILAR RESULTS IN A BUSINESS SETTING. THE RESULTS PRESENTED IN THE CONTENT ARE NOT NECESSARILY TYPICAL AND SHOULD NOT DETERMINE EXPECTATIONS OF FINANCIAL OR BUSINESS RESULTS.

MIT License

Credits

We (TH, MW) are not developers of `Stan`, only happy users.

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, Matt Hoffman, Michael Malecki, Peter Li, Yuanjun Guo.

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

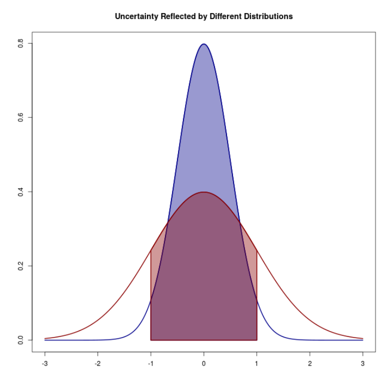
Table of Contents

Bayesian Inference with Stan

> *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.

Two normal distributions with different standard deviations. The blue distribution is, in some sense, more “precise” than the red one.[1]



[1] 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.

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].[2][3]

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

---

[2] 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].

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

Bayesian Inference

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.

Choosing Priors

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]

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].

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

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:

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

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.

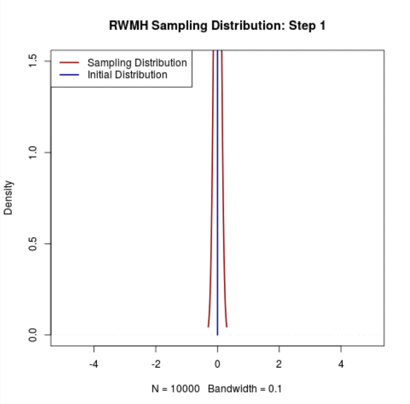
Given an initialization distribution (which may be a point mass) P_0, application of the transition kernel Π gives a new distribution P_1.

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 $\Pi\pi = \pi$.

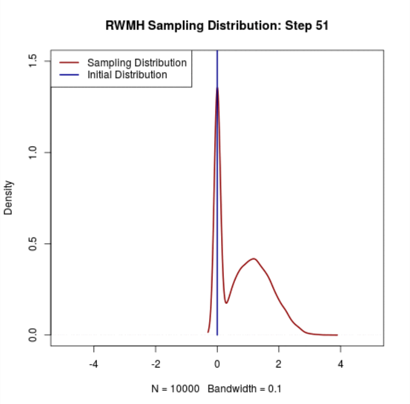
Example:

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



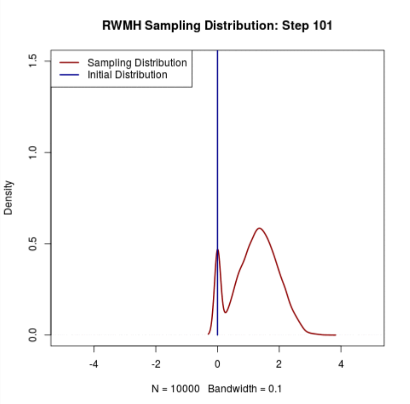
Example:

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



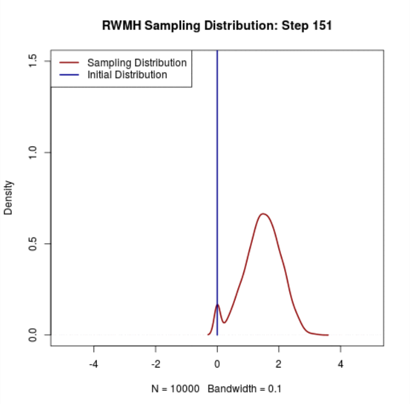
Example:

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



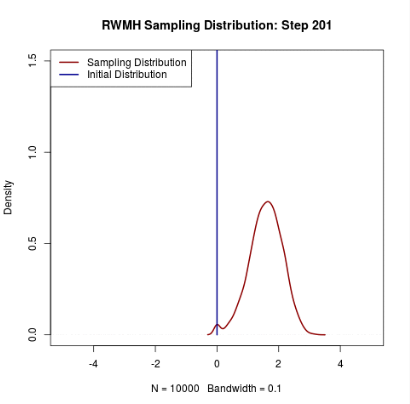
Example:

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



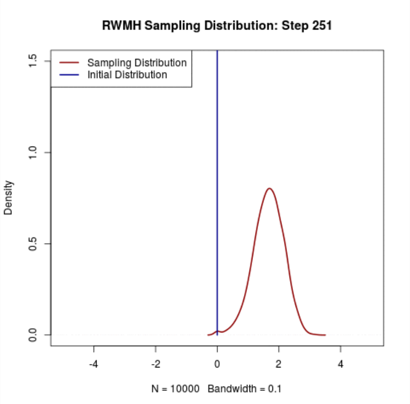
Example:

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



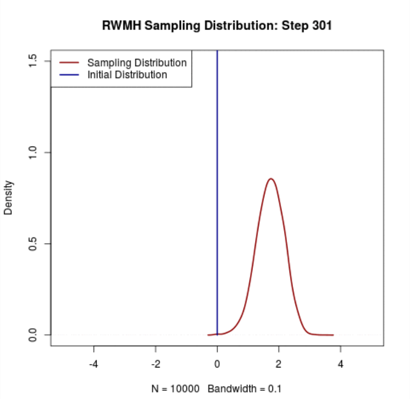
Example:

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



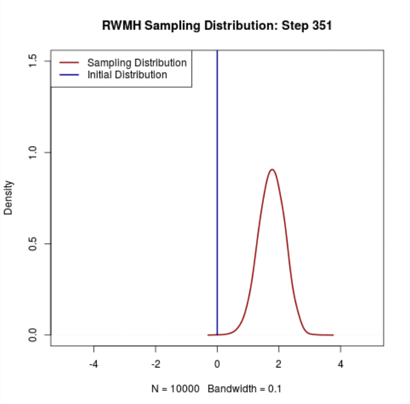
Example:

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



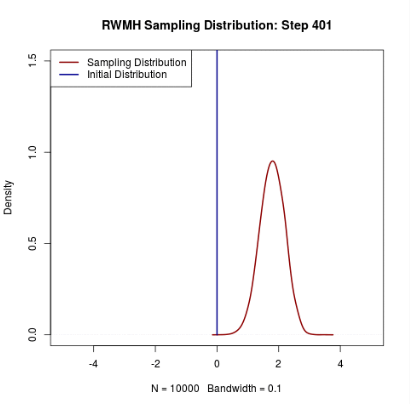
Example:

- $P_0 = \delta_0$;
- 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:

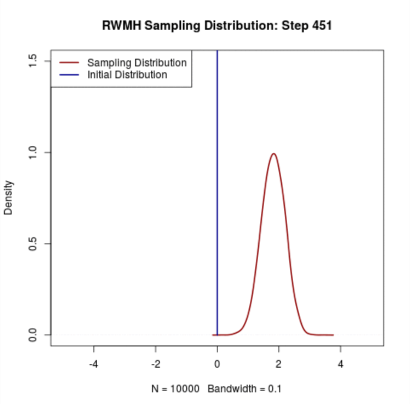
- $P_0 = \delta_0$;
- Π is a Random Walk Metropolis Hastings update (proposal distribution: $X_t \sim \mathcal{N}(X_{t-1}, 1)$);
- π is $\mathcal{N}(2, 5^2)$.



Convergence of MCMC

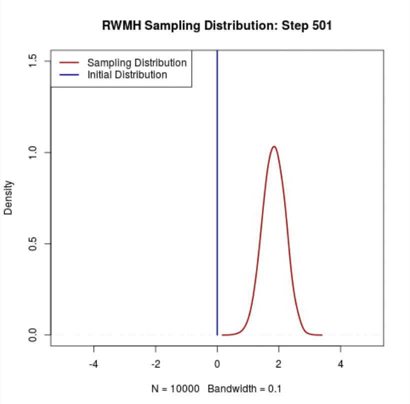
Example:

- $P_0 = \delta_0$;
- 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 $\mathcal{N}(2, 5^2)$.



Accuracy of MCMC: Effective Sample Size

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.[6]

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.

[6] 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*:[7]

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

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.[8]

[7] 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).

[8] 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.[9]

[9] 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

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.

Euclidean and Riemannian HMC

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.

AutoDiff

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.[10] Unlike numerical differentiation, AutoDiff provides exact gradient calculations with a single function call.

[10] 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`, $m = 1$.

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)[11]

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])

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

Stan

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 Compilation Model

`Stan` uses a two step compilation process: the user writes a model in pure `Stan` code[12] 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.

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

Stan Language Building Blocks

`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`

Stan Language Building Blocks

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.[13] 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, Section 56.1].

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

[13] `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])

Stan Language Model

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
}
```

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{
    increment_log_prob(beta_log(p, 1, 1)); // weak prior
    for(i in 1:N){ // likelihood
        increment_log_prob(bernoulli_log(p, y[i]));
    }
}
```

Time Series Analysis from a Bayesian Point-of-View

Demo time

Advanced Material

The power and universality of the Bayesian approach come from the ease with which we can build complex models out of simpler pieces.

Two particularly useful techniques for building realistic models are

- *mixture modeling* – modeling a complex population as a combination of simpler sub-populations
- *hierarchical modeling*[14] – exchangeable data with a set of known groups (*e.g.*, treating the returns of all stocks within the same sector as identically–but not independently–distributed)

Unfortunately, these can induce particularly tricky posteriors. In this section, we'll give some practical tips for dealing with these.

[14] Hierarchical modeling, in particular, is a statistical super-weapon. See [GH06] for an accessible introduction.

Note: The following models are not necessarily ideal (or even realistic) – they are just examples to show the flexibility and power of `Stan`. The Stan manual [Sta15d, Chapter 7] has good coverage of "standard" time series models (ARMA, HMM, *etc.*).

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:[15] we are estimating more quantities–$\{h_t\}_{t=1}^T, \mu, \phi, \sigma$–than we have observations.

[15] [vH14] is a nice introduction to the theory of high-dimensional probability; [BvdG11] is a comprehensive, though difficult, monograph on the challenges of and methods for non-Bayesian statistics in high-dimension.

The Stan manual [Sta15d, Section 7.5] 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);
    }
}
```

Stochastic Volatility

```
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));
}
```

Stochastic Volatility

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);
...
```

To get more realistic behavior, we may want to use a mixture distribution for returns.

Mixture distributions suffer from a identifiability issue: there's no (natural) way to distinguish:

$$A \sim \frac{1}{2}\mathcal{N}(0, 1) + \frac{1}{2}\mathcal{N}(0, 10)$$
$$B \sim \frac{1}{2}\mathcal{N}(0, 10) + \frac{1}{2}\mathcal{N}(0, 1)$$

because they are actually the same probability measure under different parameterizations. Since we check convergence of parameters, not measures, this can throw-off our diagnostics.[16]

For this simple case, we can force identifiability by requiring the SD of the first component to be lower than the second.[17] Stan makes this easy with the `positive_ordered` data type.

[16] This is a relatively harmless instance of the "label-switching" problem [Ste00].

[17] This works because the components inherit the ordering of $\mathbb{R}$; for more complex mixtures, this trick won't work. See [Sta15d, Section 20.2] for details.

Warning: It's easy to miscode mixtures.

Recall that Stan works by defining a *log probability density*, π. Mixture distributions add their densities on the linear scale, so the mixture density should be calculated as

$$\pi_{\text{mix}} = \log(w_1 e^{\pi_1} + w_2 e^{\pi_2})$$

The `log_sum_exp` and `log_mix` functions implement this calculation in a numerically stable way.

Finally, since we have the raw probability, we need to use the `increment_log_prob` function to change π directly, rather than using a sampling statement.

Around 2011, the finance press was taken with the idea of "risk-on/risk-off" dynamics.[18] Assuming that the latent "risk" state is IID, this naturally suggests a mixture of two return distributions with volatilities $\sigma^{(1)} < \sigma^{(2)}$:

$$\begin{aligned} w_t &\sim \mathcal{B}(1,1) \\ r_t &\sim w_t\mathcal{N}(0,(\sigma^{(1)})^2) + (1-w_t)\mathcal{N}(0,(\sigma^{(2)})^2) \end{aligned}$$

Note that, because we cannot vectorize the likelihood [Sta15d, Section 10.4], this model samples more slowly than the examples discussed so far.[19]

This model performs a "soft-clustering" of days by volatility. A more realistic model would include some sort of memory in the weight process (see, *e.g.* [Sta15d, Section 7.5]).

[18] *E.g.*, `http://www.economist.com/blogs/buttonwood/2010/11/financial_markets_and_economy`

[19] There are two causes of slowness in sampling: the time required to calculate an individual sample and the time required for the chain to mix effectively. Vectorization only helps with the former; reparamterization (or change of priors) are typically needed to address the latter.

Risk-On/Risk-Off

```
data {
    int<lower=0> T;
    vector[T] y;
}
parameters {
    positive_ordered[2] sigma;
    real<lower=0,upper=1> w[T];
}
model {
    sigma ~ cauchy(0, 5);
    w ~ beta(1, 1);
    for(t in 1:T){
        increment_log_prob(log_mix(w[t], normal_log(y[t], 0, sigma[1]),
                                         normal_log(y[t], 0, sigma[2])));
    }
}
```

Risk-On/Risk-Off Stochastic Volatility

We can combine the risk-on/risk-off mixture dynamics with stochastic volatility to have $\sigma^{(1)}, \sigma^{(2)}$ evolve over time.

$$\begin{aligned}
w_t &\sim \mathcal{B}(1,1) \\
h_{t+1}^{(1)} &\sim \mathcal{N}\left(\mu^{(1)} + \phi(h_t^{(1)} - \mu), \sigma^2\right) \\
h_{t+1}^{(2)} &\sim \mathcal{N}\left(\mu^{(2)} + \phi(h_t^{(2)} - \mu), \sigma^2\right) \\
r_t &\sim w_t \mathcal{N}(0, e^{h_t^{(1)}}) + (1 - w_t)\mathcal{N}(0, e^{h_t^{(2)}})
\end{aligned}$$

Here we force identifiability by ordering the long-run log-volatility of the two processes.

Risk-On/Risk-Off Stochastic Volatility

```
data {
   int<lower=0> T;
   vector[T] y;
}

parameters {
   ordered[2] mu; // Identify the mixture components by long run volatility
   real<lower=-1, upper=1> phi;
   real<lower=0> sigma;
   vector[T] h_std_low;
   vector[T] h_std_high;
   real<lower=0,upper=1> w[T];
}
```

Risk-On/Risk-Off Stochastic Volatility

```
transformed parameters {
   vector[T] h_low;
   vector[T] h_high;

   h_low <- h_std_low * sigma;
   h_low[1] <- h_low[1] / sqrt(1 - phi * phi);
   h_low <- h_low + mu[1];

   for(t in 2:T){
       h_low[t] <- h_low[t] + phi * (h_low[t-1] - mu[1]);
   }
   // Repeat for the second process
   // Could also wrap this in a function to have DRY-er code
   h_high <- h_std_high * sigma;
   h_high[1] <- h_high[1] / sqrt(1 - phi * phi);
   h_high <- h_high + mu[2];

   for(t in 2:T){
      h_high[t] <- h_high[t] + phi * (h_high[t-1] - mu[2]);
   }
}
```

Risk-On/Risk-Off Stochastic Volatility

```
model {
   phi ~ uniform(-1, 1);
   sigma ~ cauchy(0, 5);
   mu ~ cauchy(0, 10);
   h_std_low ~ normal(0, 1);
   h_std_high ~ normal(0, 1);
   w ~ beta(1, 1);

   for(i in 1:T){
      increment_log_prob(log_mix(w[i],
                                 normal_log(y[i], 0, exp(h_low[i]/2)),
                                 normal_log(y[i], 0, exp(h_high[i]/2))));
   }
}
```

In almost any problem where multiple units are observed (as opposed to observing one unit multiple times), the data is naturally grouped into a hierarchical structure: students within classes within schools within districts, fields within farms within states within regions, companies within subsectors within sectors, *etc.*

A hierarchical model will perform *partial pooling*: each class has its own mean, which is shrunk towards the overall school mean, each school has its own mean which is shrunk towards the district mean, *etc.*

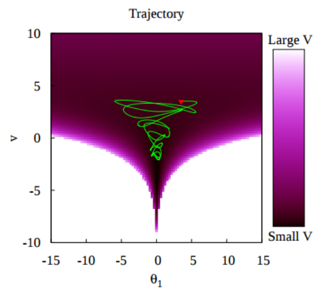
Hierarchical models allow the data to dictate how much shrinkage should occur: larger groups will be shrunk less than small groups.

A simple example: hierarchical probit regression.

```
data{
    int<lower=1> D; // number of groups
    int<lower=0> n; // Number of observations
    int<lower=0> p; // Number of predictors

    int<lower=1, upper=D> g[n]; // Group assignments
    matrix[n, p] X; // design matrix
    int<lower=0,upper=1> y[n]; // response 0/1 for probit
}
parameters{
    real mu[p]; // global means for each regression coefficient;
    real<lower=0> sigma[p]; // inter-group variability for each regression coefficient
    // here we have an matrix of p regression coefficients for the D different groups
    matrix[p, D] beta;
}
model{
    // Build a hierarchical prior on our betas
    for(p_ in 1:p){
        mu[p] ~ normal(0, 5); // Weakly regularizing priors on the grand mean coefficients
        sigma[p] ~ cauchy(0, 2.5);
        for(d in 1:D){
            // Individual regression coefficients pulled towards grand mean for that value
            beta[p_,d] ~ normal(mu[p], sigma[p]);
        }
    }
    // Likelihood: Pick appropriate betas from above
    for(i in 1:n){
        y[n] ~ bernoulli(Phi(X[i,] * beta[,g[i]]));
    }
}
```

While seemingly simple, hierarchical models can have difficult densities to sample:



(Figure 11 from [BG13]; funnel distribution described in [Nea03])

Samplers with a fixed step size have trouble exploring the large upper part of the "funnel" and the small crevices. In low-dimensions, we can use a small step size to explore these regions; in high-dimensions, a small step size may not mix rapidly.

While the "long-term" fix for this and similar issues is *Riemannian Hamiltonian Monte Carlo* [GC11, BG13], the use of "non-centered parametrizations" [PRS07] can also have a significant effect.

Equivalent parametrizations of the same problem can have very different posterior correlations:

$$y_i \sim \mathcal{N}(\mu + \alpha_i, \sigma_\epsilon^2) \qquad \Longrightarrow \text{corr}(\alpha_i, \alpha_j) = \left(1 + \frac{b\sigma_\epsilon^2}{\sigma_\alpha^2 \sigma_\mu^2}\right)^{-1}$$

$$y_i \sim \mathcal{N}(\eta_i, \sigma_\epsilon^2) \qquad \Longrightarrow \text{corr}(\alpha_i, \alpha_j) = \left(1 + \frac{b\sigma_\alpha^2}{\sigma_\epsilon^2 \sigma_\mu^2}\right)^{-1}$$

where $\eta_i = \mu + \alpha_i$ [GSC95] (more examples in [Gel04]).

Non-Centered Parameterizations

Centered:

```
beta_mean ~ cauchy(0, 5); beta_sd ~ cauchy(0, 5);
beta ~ normal(beta_mean, beta_sd);
y ~ normal(beta, 1);
```

Non-Centered:

```
beta_raw ~ normal(0, 1);
beta_mean ~ cauchy(0, 5); beta_sd ~ cauchy(0, 5);
beta <- beta_sd * beta_raw + beta_mean;
y ~ normal(beta, 1);
```

The latter tends to be more efficient.

We've already used this trick for efficient stochastic volatility models: `h` vs. `h_std`.

A Hierarchical Stochastic Volatility Model

One final example: a hierarchical stochastic volatility model: a multi-sector stochastic volatility model where the between-sector correlation evolves over time:

$$
\begin{aligned}
h_{t+1} &\sim \mathcal{N}\left(\mu_h + \phi_h(h_t - \mu_h), \sigma_h^2\right) \\
\nu_{t+1} &\sim \mathcal{N}\left(\mu_\nu + \phi_\nu(\nu_t - \mu_\nu), \sigma_\nu^2\right) \\
\Sigma_t &\sim \text{LKJ}(\Sigma, \nu_t) \\
R_t &\sim \mathcal{N}\left(0, \exp\{h_t\} * \Sigma_t\right)
\end{aligned}
$$

where

- $\exp(h_t)$ is the instantaneous volatility scale;
- $\sigma^{(i)}$ are the per-sector volatility multipliers;
- ν_t is a measure of the daily sector correlation;
- Σ_t is the instantaneous correlation;
- R_t is the daily return vector

We force $\mu_h = 1$ for identifiability here (else confounded with $\sigma^{(i)}$).

A Hierarchical Stochastic Volatility Model

```
data {
    int<lower=1> T;
    int<lower=2> K; // Sectors
    matrix[T, K] R; // Return data
}
parameters {
    real mu_nu;
    real<lower=-1, upper=1> phi_h; // Stationary volatility
    real<lower=-1, upper=1> phi_nu; // Stationary correlation
    real<lower=0> sigma_h; real<lower=0> sigma_nu;
    vector[T] h_std; vector[T] nu_std;
    corr_matrix Sigma; // Average correlation
    real<lower=0> sigma[K]; // Average sector vols
}
```

(continued)

A Hierarchical Stochastic Volatility Model

```
transformed parameters {
    vector[T] h;
    h <- h_std * sigma_h;
    h[1] <- h[1] / sqrt(1 - phi_h * phi_h);
    h <- h + 1;
    for(t in 2:T){
        h[t] <- h[t] + phi_h * (h[t-1] - 1);
    }

    vector[T] nu;
    nu <- nu_std * sigma;
    nu[1] <- nu[1] / sqrt(1 - phi_nu * phi_nu);
    nu <- nu + mu_nu;
    for(t in 2:T){
        nu[t] <- nu[t] + phi_nu * (nu[t-1] - nu);
    }
}
```

(continued)

A Hierarchical Stochastic Volatility Model

```
model {
    // Priors
    phi_h ~ uniform(-1, 1); phi_nu ~ uniform(-1, 1);
    sigma_h ~ cauchy(0, 5); sigma_nu ~ cauchy(0, 5);
    // Scaled Innovations in h and eta processes are IID N(0,1)
    h_std ~ normal(0, 1); nu_std ~ normal(0, 1);
    // Sector covariances get weak cauchy prior
    sigma ~ cauchy(0, 10);
    // Instantaneous covariance matrix for likelihood
    {
        for(t in 1:T){
            // Local variable
            Sigma_t ~ LKJcorr(nu[t]);
            R[t, ] ~ multi_normal(0, exp(h[t]/2) * sigma * Sigma_t);
        }
    }
}
```

Thank you

Learning More

If you're interested in learning more, start with Michael Betancourt's MLSS-2014 talks (HMC (link) and Stan (link)).

Bob Carpenter's MLSS-2015 talk (link) is a bit more "hands-on" with the Stan language. (Michael's talk goes 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.

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.

References II

[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.

[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.

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

[BvdG11] Peter Bühlmann and Sara van de Geer.
Statistics for High-Dimensional Data: Methods, Theory and Applications.
Springer Series in Statistics. Springer, 1st edition, 2011.

[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.

[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.

[Gel04] Andrew Gelman.
Parameterization and bayesian modeling.
Journal of the American Statistical Association, 99(466), 2004.

[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.

References VIII

[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`.

[GSC95] Alan E. Gelfand, Sujit K. Sahu, and Bradley P. Carlin.
Efficient parametrisations for normal linear mixed models.
Biometrika, 82(3):479–488, 1995.

[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.

[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.

[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.

[Nea03] Radford Neal.
Slice sampling.
The Annals of Statistics, 31(3):705–767, 2003.
https://projecteuclid.org/euclid.aos/1056562461.

[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.

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

[PRS07] Omiros Papaspiliopoulos, Gareth O. Roberts, and Martin Sköld.
A general framework for the parametrization of hierarchical models.
Statistical Science, 22(1):59–73, 2007.
https://projecteuclid.org/euclid.ss/1185975637.

[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 2.9.0, 2015.
http://mc-stan.org/pystan.html;
https://pypi.python.org/pypi/pystan.

[Sta15b] Stan Development Team.
rstan: the R interface to Stan, version 2.9.0, 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 2.9.0, 2015.
`http://mc-stan.org/`.

[Sta15d] Stan Development Team.
Stan Modeling Language Users Guide and Reference Manual, Version 2.9.0, 2015.
`http://mc-stan.org`.

[Ste00] Matthew Stephens.
Dealing with label switching in mixture models.
Journal of the Royal Statistical Society, Series B, 62(4):795–809, 2000.

[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.

[vH14] Ramon van Handel.
Probability in high dimension, 2014.
https://www.princeton.edu/~rvan/ORF570.pdf.