From Your Algorithm to Stan: BridgeStan

2023-03-16

From your algorithm to Stan

1. Specify your algorithm (e.g., BayesComp 2023 talk/poster)
2. Develop prototype in BridgeStan
3. Evaluate using PosteriorDb

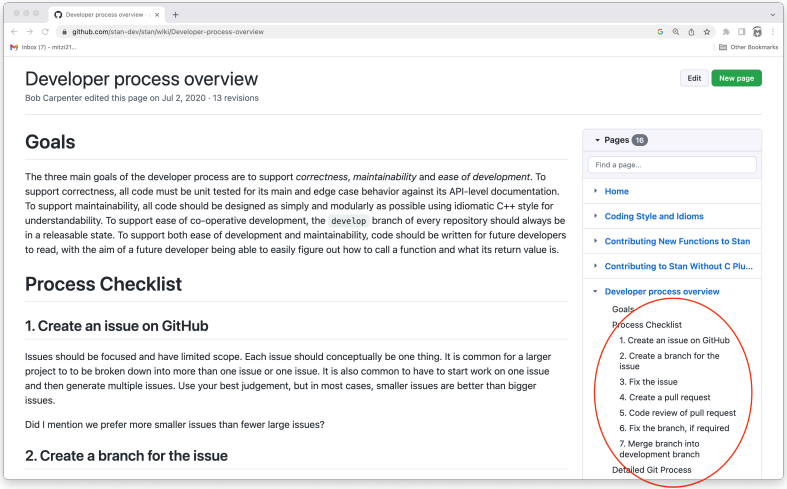
Stan - Key Components

- Stan probabilistic programming language
  - Stan program specifies a joint probability density
  - `stanc` compiler translates Stan code to efficient C++
- Math library
  - forward and reverse auto-diff - fastest auto-diff for non-GPU/TPU machines
  - higher-order functions - ODEs, algebraic solvers, forward or adjoint methods
  - distributions - bernoulli, bessel, ..., weibull, wiener, wishart
  - linear algebra - cholesky, QR decomposition, broadcast operators
  - vectorized operators, distributions
- Stan base model class object

Inference Algorithms

- Available via command line, Julia, Python, and R wrapper interfaces
  - exact Bayesian inference using NUTS-HMC sampler
  - approximate Bayesian inference - ADVI
  - point estimation - MLE, MAP
  - Laplace sample from MAP stimate
  - Standalone Generated Quantities - use existing sample to generate new quantities of interest
  - (LogProb - extract log probabilities and gradients)
  - (Diagnostics - compare initial gradients with finite-differences)
- Increasingly difficult to add new algorithms
  - mature code base has accumulated technical debt

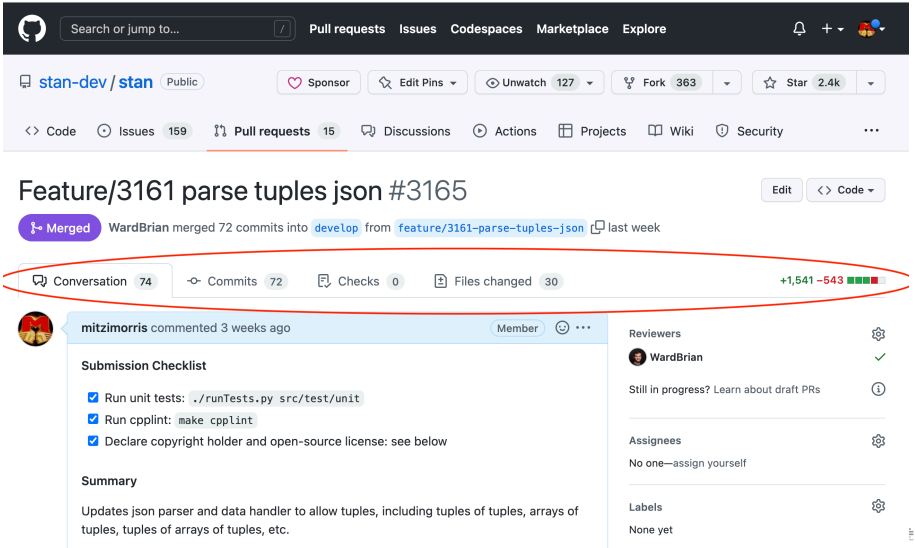
Stan Developer Process



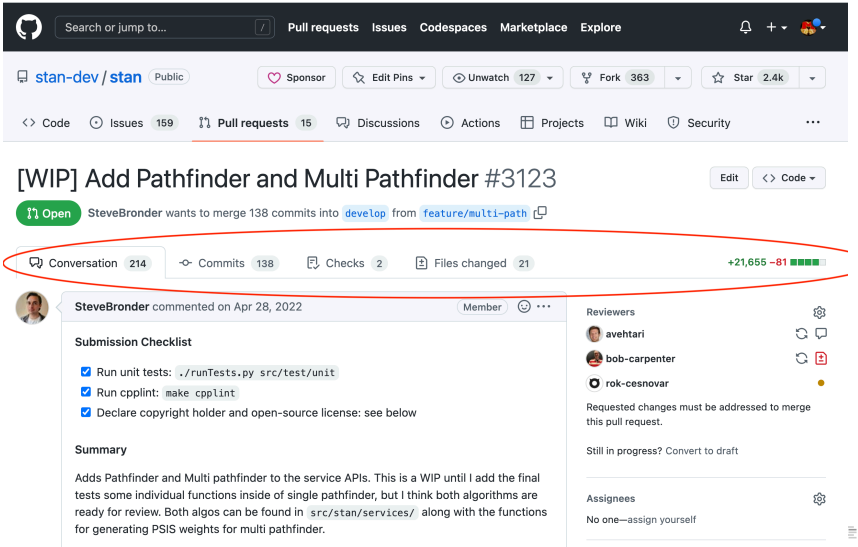
But

- It's never that straightforward - any new feature involves
  - multiple iterations
  - successive refinements
  - extensive testing
  - developer review and consensus

Example PR: JSON parser



Example PR: Pathfinder

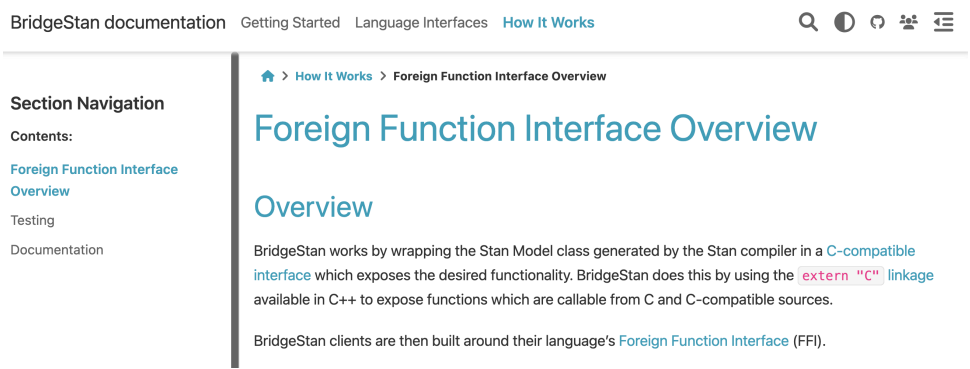


BridgeStan - https://roualdes.github.io/bridgestan/

- (No longer missing) Stan interface for algorithm developers
  - BridgeStan instantiates model given data
  - code in your comfort zone
  - common Stan math backend - compare algorithm efficiency
- Efficient in-memory access to the methods of the Stan model class
  - log densities, gradients, Hessians
  - constraining and unconstraining transforms
  - if methods of the Stan model class change, BridgeStan will too.
- Interfaces for Python, Julia, and R
- Release v1.0.1, Feb 2023

BridgeStan - How it Works

In-memory access is a foreign-function call into compiled model `.so` file



Bernoulli Example

Model bernoulli.stan

```
data {
  int<lower=0> N;
  array[N] int<lower=0,upper=1> y;
}
parameters {
  real<lower=0,upper=1> theta;
}
model {
  theta ~ beta(1,1);  // uniform prior on interval 0,1
  y ~ bernoulli(theta);
}
```

Data bernoulli.data.json

```
{ "N" : 10, "y" : [0,1,0,0,0,0,0,0,0,1] }
```

BridgeStan Python Example - Hello, World!

HelloBridgestan.ipynb

Code

[1]:
```
import os
import bridgestan as bs
import numpy as np
bs.set_bridgestan_path(os.path.join(os.path.expanduser('~'), 'github', 'bridgestan'))
```

[2]:
```
stan = './stan/bernoulli.stan'
data = './data/bernoulli.data.json'
hello_berni = bs.StanModel.from_stan_file(stan, data, seed=12345)
```

[3]:
```
print(f"This model's name is {hello_berni.name()}.")
print(f"It has {hello_berni.param_num()} parameters.")
```

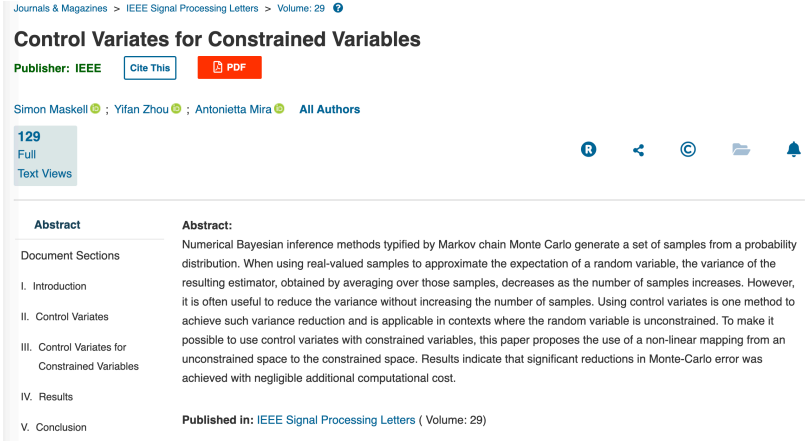
```
This model's name is bernoulli_model.
It has 1 parameters.
```

[4]:
```
x = np.random.random(hello_berni.param_unc_num())
q = np.log(x / (1 - x))  # unconstrained scale
lp, grad = hello_berni.log_density_gradient(q, jacobian=False)
print(f"log_density and gradient of Bernoulli model: {(lp, grad)}")
```

```
log_density and gradient of Bernoulli model: (-32.98525010174007, array([-7.83740174]))
```

BridgeStan Python Example - Control Variates

https://github.com/zhouyifan233/ControlVariates



Journals & Magazines > IEEE Signal Processing Letters > Volume: 29

Control Variates for Constrained Variables

Publisher: IEEE | Cite This | PDF

Simon Maskell ; Yifan Zhou ; Antonietta Mira All Authors

129 Full Text Views

Abstract

Document Sections

I. Introduction

II. Control Variates

III. Control Variates for Constrained Variables

IV. Results

V. Conclusion

Abstract:
Numerical Bayesian inference methods typified by Markov chain Monte Carlo generate a set of samples from a probability distribution. When using real-valued samples to approximate the expectation of a random variable, the variance of the resulting estimator, obtained by averaging over those samples, decreases as the number of samples increases. However, it is often useful to reduce the variance without increasing the number of samples. Using control variates is one method to achieve such variance reduction and is applicable in contexts where the random variable is unconstrained. To make it possible to use control variates with constrained variables, this paper proposes the use of a non-linear mapping from an unconstrained space to the constrained space. Results indicate that significant reductions in Monte-Carlo error was achieved with negligible additional computational cost.

Published in: IEEE Signal Processing Letters (Volume: 29)

BridgeStan Python Example - Control Variates

Model

https://github.com/zhouyifan233/ControlVariates/blob/main/controlvariates/postprocess_bs.py

```python
def run_postprocess(samples, model_bs, cv_mode='linear', output_squared_samples=False, output_runtime=False):
    num_samples = samples.shape[0]

    # Unconstraint mcmc samples.
    unconstrained_samples = []
    for i in range(num_samples):
        unconstrained_samples.append(model_bs.param_unconstrain(copy.copy(samples[i])))
    unconstrained_samples = np.array(unconstrained_samples)

    # Calculate gradients of the log-probability
    grad_start_time = time.time()
    grad_log_prob_vals = []
    for i in range(num_samples):
        log_p, grad = model_bs.log_density_gradient(copy.copy(unconstrained_samples[i]), propto=True, jacobian=True)
        grad_log_prob_vals.append(grad)
    grad_log_prob_vals = np.array(grad_log_prob_vals)
    grad_runtime = time.time() - grad_start_time

    # Run control variates
    cv_start_time = time.time()
    if cv_mode == 'linear':
        cv_samples = linear_control_variates(samples, grad_log_prob_vals)
        cv_runtime = time.time() - cv_start_time
        # print('Gradient time: {:.05f} --- Linear control variate time: {:.05f}.'.format(grad_runtime, cv_runtime))
```

New sampling algorithms

To name a few:

- MEADS - Tuning-Free Generalized HMC - Hoffman and Sounstov
  https://proceedings.mlr.press/v151/hoffman22a/hoffman22a.pdf
  (Prototype:
  https://github.com/roualdes/MCBayes.jl/blob/main/test/meads.jl)
- MALT - Metropolis Adjusted Langevin Trajectories - Riou-Durand and Vogrinc
  https://arxiv.org/pdf/2202.13230

Developer Bob Carpenter:

"I think the combo of all these ideas will be the way forward for sampling."

Discussion: Which algorithms?

- Stan program `model` block fits the joint log prob of all parameters
  - no distinction between prior and likelihood statements
- Data is static
  - no natural way to stream new observations
- No discrete parameters
  - interface has `params_i` arg; the appendix in the guts of Stan

References

- BridgeStan announcement
- BridgeStan Discussion thread on Stan Forums
- Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A probabilistic programming language. *Journal of Statistical Software* 76(1). 10.18637/jss.v076.i01
- Bob Carpenter, Matthew D. Hoffman, Marcus Brubaker, Daniel Lee, Peter Li, and Michael J. Betancourt. 2015. The Stan Math Library: Reverse-Mode Automatic Differentiation in C++. *arXiv*:1509.07164