



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, bessell, ..., 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 estimate
 - 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

Developer process overview · x +

github.com/stan-dev/stan/wiki/Developer-process-overview

Inbox (7) - mitzi21...

Developer process overview

Bob Carpenter edited this page on Jul 2, 2020 · 13 revisions

Edit New page

Goals

The three main goals of the developer process are to support *correctness*, *maintainability* and *ease of development*. To support correctness, all code must be unit tested for its main and edge case behavior against its API-level documentation. To support maintainability, all code should be designed as simply and modularly as possible using idiomatic C++ style for understandability. To support ease of co-operative development, the `develop` branch of every repository should always be in a releasable state. To support both ease of development and maintainability, code should be written for future developers to read, with the aim of a future developer being able to easily figure out how to call a function and what its return value is.

Process Checklist

1. Create an issue on GitHub

Issues should be focused and have limited scope. Each issue should conceptually be one thing. It is common for a larger project to be broken down into more than one issue or one issue. It is also common to have to start work on one issue and then generate multiple issues. Use your best judgement, but in most cases, smaller issues are better than bigger issues.

Did I mention we prefer more smaller issues than fewer large issues?

2. Create a branch for the issue

Pages 16

Find a page...

- Home
- Coding Style and Idioms
- Contributing New Functions to Stan
- Contributing to Stan Without C Plu...
- Developer process overview**
 - Goals
 - Process Checklist
 - 1. Create an issue on GitHub
 - 2. Create a branch for the issue
 - 3. Fix the issue
 - 4. Create a pull request
 - 5. Code review of pull request
 - 6. Fix the branch, if required
 - 7. Merge branch into development branch
 - Detailed Git 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



[Pull requests](#) [Issues](#) [Codespaces](#) [Marketplace](#) [Explore](#)

 [stan-dev / stan](#) Public

[Sponsor](#) [Edit Pins](#) [Unwatch](#) 127 [Fork](#) 363 [Star](#) 2.4k


[Code](#) [Issues](#) 159 [Pull requests](#) 15 [Discussions](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#)

Feature/3161 parse tuples json #3165

[Edit](#) [Code](#)

[Merged](#) WardBrian merged 72 commits into [develop](#) from [feature/3161-parse-tuples-json](#) last week

[Conversation](#) 74 [Commits](#) 72 [Checks](#) 0 [Files changed](#) 30 +1,541 -543



mitzimorris commented 3 weeks ago

Member



Submission Checklist

- ☒ Run unit tests: `./runTests.py src/test/unit`
- ☒ Run cplint: `make cplint`
- ☒ Declare copyright holder and open-source license: see below

Summary

Updates json parser and data handler to allow tuples, including tuples of tuples, arrays of tuples, tuples of arrays of tuples, etc.

Reviewers

 WardBrian 

Still in progress? [Learn about draft PRs](#)

Assignees


No one—assign yourself

Labels

None yet



Example PR: Pathfinder

 Search or jump to... Pull requests Issues Codespaces Marketplace Explore

stan-dev / stan Public Sponsor Edit Pins Unwatch 127 Fork 363 Star 2.4k

<> Code Issues 159 Pull requests 15 Discussions Actions Projects Wiki Security

[WIP] Add Pathfinder and Multi Pathfinder #3123

Edit <> Code

Open

 SteveBronder wants to merge 138 commits into develop from feature/multi-path


Conversation 214

Commits 138

Checks 2

Files changed 21

+21,655 -81

 SteveBronder commented on Apr 28, 2022 Member


Submission Checklist


- Run unit tests: `./runTests.py src/test/unit`
- Run cpplint: `make cpplint`
- Declare copyright holder and open-source license: see below


Summary

Adds Pathfinder and Multi pathfinder to the service APIs. This is a WIP until I add the final tests some individual functions inside of single pathfinder, but I think both algorithms are ready for review. Both algos can be found in `src/stan/services/` along with the functions for generating PSIS weights for multi pathfinder.

Reviewers

 avehtari

 bob-carpenter

 rok-cesnovar

Requested changes must be addressed to merge this pull request.

Still in progress? Convert to draft

Assignees

No one—assign yourself



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

BridgeStan documentation Getting Started Language Interfaces [How It Works](#)



Section Navigation

Contents:

[Foreign Function Interface Overview](#)

Testing

Documentation

[Home](#) > [How It Works](#) > Foreign Function Interface Overview

Foreign Function Interface Overview

Overview

BridgeStan works by wrapping the Stan Model class generated by the Stan compiler in a [C-compatible interface](#) which exposes the desired functionality. BridgeStan does this by using the `extern "C"` linkage available in C++ to expose functions which are callable from C and C-compatible sources.

BridgeStan clients are then built around their language's [Foreign Function Interface](#) (FFI).



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  X  +

[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_bern_i = bs.StanModel.from_stan_file(stan, data, seed=12345)

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

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

[4]: x = np.random.random(hello_bern_i.param_unc_num())
q = np.log(x / (1 - x)) # unconstrained scale
lp, grad = hello_bern_i.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

```
34 def run_postprocess(samples, model_bs, cv_mode='linear', output_squared_samples=False, output_runtime=False):
35     num_samples = samples.shape[0]
36
37     # Unconstraint mcmc samples.
38     unconstrained_samples = []
39     for i in range(num_samples):
40         unconstrained_samples.append(model_bs.param_unconstrain(copy.copy(samples[i])))
41     unconstrained_samples = np.array(unconstrained_samples)
42
43     # Calculate gradients of the log-probability
44     grad_start_time = time.time()
45     grad_log_prob_vals = []
46     for i in range(num_samples):
47         log_p, grad = model_bs.log_density_gradient(copy.copy(unconstrained_samples[i]), propto=True, jacobian=True)
48         grad_log_prob_vals.append(grad)
49     grad_log_prob_vals = np.array(grad_log_prob_vals)
50     grad_runtime = time.time() - grad_start_time
51
52     # Run control variates
53     cv_start_time = time.time()
54     if cv_mode == 'linear':
55         cv_samples = linear_control_variates(samples, grad_log_prob_vals)
56         cv_runtime = time.time() - cv_start_time
57         # print('Gradient time: {:.05f} --- Linear control variate time: {:.05f}'.format(grad_runtime, cv_runtime))
58     ...
```



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*