Implementing distributions in STAN

An easy* tutorial on how to do it

Anna Freni-Sterrantino
The Alan Turing Institute

22nd May, 2025

Credits

Zhi Ling and Oliver Ratmann

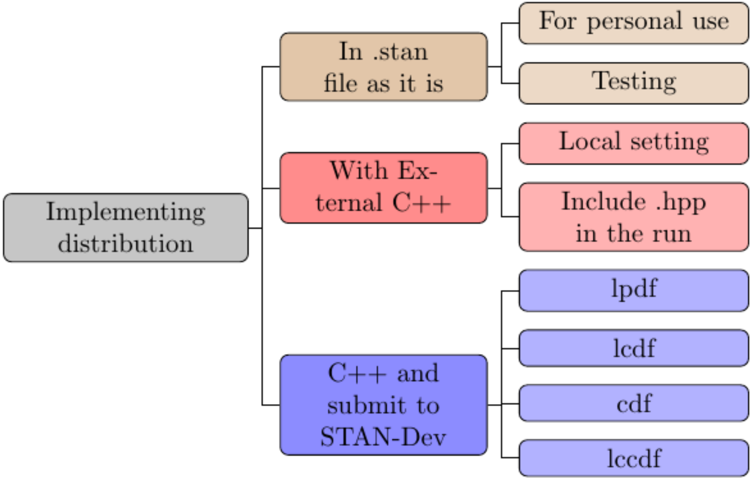
Overview:

1. Introduction
2. How to simply code in .stan
3. Express log-probability density function(pdf), cumulative distribution function (cdf), complementary cdf(ccdf) and random number generator (rng) Compute for all the above the derivatives
4. Code in C++ using the provided template

Recommended approach (from Stan-math page)

- Code the distribution in Stan syntax
- Write down/calculate the partial derivatives for each input
- Code C++ distribution
- Testing disitrbution

Different ways for different reasons



What do you need

- A distribution of your choice
- Stan & co. installed
- Libraries: GCC, Boost, Sundials
- Editor: Emacs or your favourite
- Useful: Maplesoft/ ChatGPT(!)/matrixcalculus.org (has no digamma)

What do you need

- Secret Ingredient: Patience

Roadmap

1. Yule-Simon distribution
2. Code it in Stan-ishland
3. Code as external files (C++) the random number generator
4. You code the C++ log pmf

Life-savings hacks (Thank me later!)

1. Install Stan-math Library and follow the instructions to make sure everything works fine
2. Check all your C++ libraries are updated
3. Run some examples to assess whether your configuration is working

The Yule-Simon distribution

A probability distribution function (pdf):

$$Y \sim \alpha B(y, \alpha+1) = \alpha \frac{\Gamma(y)\Gamma(\alpha+1)}{\Gamma(y+\alpha+1)} = \frac{\alpha\alpha!(y-1)!}{(y+\alpha)!}$$

where $y \geq 1$ is in integer, $\alpha > 0$ is a shape parameter, B is the Beta function, Γ is a the gamma function.
The log pdf:

$$lpmf = \log(\alpha) + \log\left(\Gamma(y)\right) + \log\left(\Gamma(\alpha+1)\right) - \log\left(\Gamma(y+\alpha+1)\right)$$

The cumulative distribution function (cdf):

$$F(x) = P(Y \leq y) = 1 - yB(y, \alpha+1) = 1 - y \cdot \frac{\Gamma(y)\Gamma(\alpha+1)}{\Gamma(y+\alpha+1)}$$

Into .stan file as it is

In .stan file as it is

Or

```
functions{
  real yule_simon_lpmf(int y, real a) {
   real lprobs = log(a) + lgamma(y) + lgamma(a+1) - lgamma(y+1+a);
   return lprobs;
}
```

`_lpmf` suffix allows the function to act as a density function in the program.

Into .stan- cont.d

Improving the code for good practices:

```
functions{
    real yule_simon_lpmf(array[] int y, real a) {
     int N = size(y);
     vector[N] lprobs;
        for (i in 1:N) {
        lprobs[i] = lgamma(a) + lgamma(y[i])+ lgamma(a+1) - lgamma(y[i]+1+a);
}
        return  sum(lprobs);
}}
....
model{
  a ~gamma(0.001, 0.001);
  y ~yule_simon(a);
}
```

Hands on → **Folder** `Basic_stan`

Intermediate step

With External C++

- We code in C++
- Save in hpp file
- We 'compile' adding the 'new-distribution.hpp' in the `cmdstan_model`

Example

Suppose we want to write a function to compute odds in C++

```
#include <iostream>
% namespace bernoulli_example_model_namespace {
  double make_odds(const double& theta, std::ostream *pstream__) {
    return theta / (1 - theta);
  }
}

```

saved into a file named `external.hpp`

In stan file

```
functions {
  real make_odds(data real theta);
}
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);
}
generated quantities {
  real odds;
  odds = make_odds(theta);
}
```

Local C++ hpp file

```r
mod <- cmdstan_model('bernoulli_example.stan',
     include_paths=getwd(),
     cpp_options=list(USER_HEADER='external.hpp'),
     stanc_options = list("allow-undefined")
)


```

Hands on → **Folder** `Intermediate/Bernoulli`

C++ external new distribution file

```
template <bool propto, typename T_y, typename T_a>
stan::return_type_t<T_a> yule_simon_lpmf(const T_y &y,
const T_a &a,
std::ostream *pstream__) {
  using stan::math::lbeta;
  using stan::math::log;
  auto lpmf = lbeta(y, a + 1.0);
  if constexpr (stan::math::include_summand<propto>::value) {
    lpmf += log(a);
  }
  return lpmf;
}
```

To call the .hpp file in R/stan

```r
mod <- cmdstan_model('ys_example.stan',
                     include_paths=getwd(),
                     cpp_options=list(USER_HEADER='external.hpp'),
                     stanc_options = list("allow-undefined")
)

```

Hands on → Folder `Intermediate/YS_ZL`

Let's move to the next layer: C++

C++ and submit to STAN-Dev

- Write the logarithmic pdf, cdf and ccdf, compute the derivative and code in C++
- `https://github.com/stan-dev/math/blob/develop/stan/math/prim/prob/`
- Stan uses automatic differentiation to define gradients of the log density function.

C++ and Stan-math files

- Because C++'s double doesn't track gradients. It's just a number.
- Stan introduces a custom type: `stan::math::var, a class that wraps a double and builds the graph during evaluation.`
- `Provide the partial derivatives - DONE`
- `Use C++ template function`
- `https://stan-dev.r-universe.dev/articles/StanHeaders/stanmath.html`

Preliminary notions

To fully implement a distribution in Stan, it is often desirable to mathematically derive certain derivatives and include them as well. by $f(y, \boldsymbol{\theta})$, $F(y, \boldsymbol{\theta})$, and $C(y, \boldsymbol{\theta})$, respectively, where $\boldsymbol{\theta}$ is the parameter vector.

$$\log f(y, \boldsymbol{\theta}) \tag{1}$$

We aim to calculate the gradients of this log-likelihood function with respect to the distribution parameters $\boldsymbol{\theta}$.

$$\nabla_{\boldsymbol{\theta}} \log f(y, \boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}} \log F(y, \boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}} \log C(y, \boldsymbol{\theta}) \tag{2}$$

where $\nabla_{\boldsymbol{\theta}} \stackrel{\text{def}}{=} \left[\frac{\partial}{\partial \theta_1}, \frac{\partial}{\partial \theta_2}, \cdots, \frac{\partial}{\partial \theta_n}\right] = \frac{\partial}{\partial \boldsymbol{\theta}}$.

Useful results

Knowing that

$$\frac{d}{dz} log \Gamma(z) = \frac{\Gamma(z)'}{\Gamma(z)} = \psi(z)(digamma)$$

$$\frac{\partial \log B(\alpha, \beta)}{\partial \beta} = \psi(\beta) - \psi(\alpha + \beta),$$

Useful relationships

$$P(Y \le y) = F(r, \alpha, \beta) = 1 - C(\alpha,)$$

The partial derivative of the $F(\alpha)$ w.r.t. α is

$$\begin{aligned}
\frac{\partial \log F(\alpha)}{\partial \alpha} &= \frac{\partial \log[1 - C(\alpha)]}{\partial \alpha} \\
&= -\frac{1}{1 - C(r, \alpha, \beta)} \frac{\partial C(r, \alpha, \beta)}{\partial r} \\
&= -\frac{1}{1 - C(\alpha)} \frac{\partial \log C(\alpha)}{\partial \alpha} C(\alpha).
\end{aligned}$$

This is to say, to know $\frac{\partial \log F(\alpha)}{\partial \alpha}$, we only need to know $\frac{\partial \log C(\alpha)}{\partial \alpha}$

Derivative of the log pmf (lpmf)

$$
\begin{aligned}
&\frac{d}{d\alpha}\left[\log(\alpha) + \log\left(\Gamma(y)\right) + \log\left(\Gamma(\alpha+1)\right) - \log\left(\Gamma(y+\alpha+1)\right)\right] \\
&= \frac{1}{\alpha} + \psi(\alpha+1) - \psi(y+\alpha+1)
\end{aligned}
$$

Derivative of the log cdf (lcdf)

$$P(Y \leq y) = F(y, \alpha) = 1 - C(y, \alpha)$$

The partial derivative of the $F(r, \alpha, \beta)$ w.r.t. α is

$$\begin{aligned}
\frac{\partial \log F(y, \alpha)}{\partial \alpha} &= \frac{\partial \log[1 - C(y, \alpha)]}{\partial \alpha} \\
&= -\frac{1}{1 - C(y, \alpha)} \frac{\partial C(y, \alpha)}{\partial \alpha} \\
&= -\frac{1}{1 - C(y, \alpha)} \frac{\partial \log C(y, \alpha)}{\partial \alpha} C(y, \alpha).
\end{aligned}$$

* to know $\frac{\partial \log F(y,\alpha)}{\partial \alpha}$, we only need to know $\frac{\partial \log C(y,\alpha)}{\partial \alpha}$

Least but not last: log ccdf (lccdf)

For C++ implementation, also the complementary cumulative distribution

$$P(Y > y) = 1 - F(y, \alpha) = C(y, \alpha)$$

Random Generator Number - it is easier

To code `yule_simon_rng` for $Y \sim YS(\alpha)$

- Code to generate YS random numbers needs suffix `_rng`
- The `_rng` function allows the use of other `_rng` functions such `uniform_rng`, `exponential_rng` or `neg_binomial_rng` as a hack for geometric distribution.

Potential RNG functions

```
VGAM::ryules
function (n, shape)
{
    rgeom(n, prob = exp(-rexp(n, rate = shape))) + 1
}

```

Another option is to approach it using an inverse method

```
  ryule_simon <- function(n, rho) {
  if (rho <= 0) stop("rho must be > 0")
  u <- runif(n)
  v <- rbeta(n, rho, 1)
  x <- 1 + floor(log(u) / log(v))
  return(x) }

```

Use `uniform_rng` and `beta_rng`

Hands on → **Folder** `Dev_style`

Testing our C++ function- Dev type

To test that the C++ functions are correctly written

- `https://github.com/stan-dev/math/tree/develop/test/prob`
- `mydist_test.hpp` containing a class inheriting from `AgradDistributionTest` containing methods
  - `valid_values()` which fills the [in/out] params argument with valued testing values for your distribution and the [in/out] `log_prob`argument with the log probability given those parameters.
  - `invalid_values()` which fills the [in/out] value argument with testing values that should fail.
  - Two `log_prob()` methods which call your distribution, one with `propto` and the other without.
  - `log_prob_function()` which is the log probability density function's simple implementation much like the one you wrote in Stan.

Summary

- A better idea on how to navigate the different aspects of coding new distribution in STAN
- Understand the benefit of coding it in C++/stan-math library
- A starting point to implement your own distribution or modify
- Slides and extra documents will be sent/shared

Background on the Yule-Simon distribution

- YS arises as a limiting distribution of a particular model studied by Udny Yule in 1925 to analyse the growth in the number of species per genus in some higher taxa of biotic organisms.
- The distribution also arises as a compound distribution, in which the parameter of a geometric distribution is treated as a function of random variable having an exponential distribution.
- The Yule distribution is a special case of the beta-geometric distribution, when $b = 1$(King, M, 2017).
- The Waring distribution is a generalization of the Yule distribution.
- For large x-values, the Zipf distribution and the Yule-Simon distribution are indistinguishable. In other words, the Zipf distribution models the tail end of the Yule.