SBC basic

Hyunji Moon, Shinyoung Kim

2021-01-10

SBC

Introduction

SBC aims to extend `rstan::sbc()` to include support for new Simulation Based Calibration algorithms without the user having to hassle with sampling or standardizing input/output. It provides utility functions which can ease sampling prior/posterior predictive distributions, along with generic SBC plots, experimental features such as bootstrap/jacknife sampling, and integral probability metric based tests.

This vignette will demonstrate how to generate rank and fitted parameter outputs, which are used within SBC to plot graphs and calculate metrics.

Setting up the stan model - Conventions

Much like `rstan::sbc()`, SBC requires a couple of naming conventions within your stan model. Each convention will be explained thoroughly during the example walkthrough:

1. If you wish to sample from the prior predictive distribution directly within your stan model, they must be defined within the `generated quantities` block, with an underscore after the parameter name. For example, if your model has a parameter `mu ~ normal(0, 1);` within the model block, `mu_ ~ normal_rng(0, 1);` must be defined within `generaged quantities`.
2. In order to sample from the posterior predictive distribution, a generative model `y_ = P(theta)` must be defined within the `generated quantities` block. There is no specific naming convention enforced, but should be recognizable to the user(`rstan::sbc()` enforces the name "y_", and is therefore recommended).

SBC should automatically detect multidimensional parameters and appropriately access them.

Example model - Simple Poisson Regression

Example model - model setup

We will be running SBC against a model that defines `y ~ Poisson(lambda)`, where `lambda ~ Gamma(15, 5)`. The stan code for the model would be the following:

```
stan_code <- "
data{
  int n_datasets;
  int y[n_datasets];
}
parameters{
  real<lower = 0> lambda;
}
model{
  lambda ~ gamma(15, 5);
  y ~ poisson(lambda);
}

generated quantities{
  real lambda_ = gamma_rng(15, 5);  // optional
  int y_[n_datasets];
  real log_lik[n_datasets];
  for (i in 1:n_datasets){
    y_[i] = poisson_rng(lambda);
    log_lik[i] = poisson_lpmf(y_[i] | lambda);
  }
}
"
```

We have defined a model with posterior predictive sampling and posterior log probability defined in `generated quantities`. Note that the variable `lambda_`, which is equal to the prior predictive distribution, is also defined. However, this variable is optional, and instead can be replaced within R's `rexp()`.

We can then compile the model using either `rstan` or `cmdstan`, and create a `SBC::SBCModel` object. Note that using either `rstan` or `cmdstan` has no functional differences.

```
# set hyperprior and iteration numbers
n_datasets = 100 # total number of simulated parameter sets from theta
thin = 3

data <- list(n_datasets=n_datasets, y=as.vector(1:n_datasets))

model <- cmdstan_model(write_stan_file(stan_code))
```

```
## Compiling Stan program...
```

```
# alternatively, use rstan:
# model <- stan_model(model_code = stan_code)

sbc_obj <- SBCModel$new(name = "poisson", stan_model = model)
```

Example model - prior predictive sampling

Once we have the model compiled and `SBCModel` ready, we'll sample from the prior predictive distribution.

If the prior predictive distributions aren't defined within the stan model, you'll need to create a named list containing functions which specify them:

```
hyperpriors <- list("lambda"=function(){rgamma(1, shape=15, scale=0.2)})
```

Once that's done, you'll be able to sample from the prior predictive distribution with either `SBCModel$sample_theta_tilde()` or `SBCModel$sample_theta_tilde_stan()`. We'll be sampling a total of `n_datasets` individual samples of the parameter `lambda`:

```
theta_prior <- sbc_obj$sample_theta_tilde_stan(list("lambda"), n_datasets, data =
        data)
# or alternatively, use the hyperpriors list:
# theta_prior <- sbc_obj$sample_theta_tilde_stan(list("lambda), n_datasets,
        hyperpriors)
```

Example model - posterior predictive sampling

After we obtained the realizations for each parameter, we'll now sample from the posterior predictive distribution, conditioned on the sampled parameter draws. SBC will identify the number of samples you draw and extract them into an array:

```
sampled_y <- sbc_obj$sample_y_tilde(theta_prior , data=data)
```

Note that data passed into the model has no effect on the results.

Example model - refit theta with posterior predictive samples

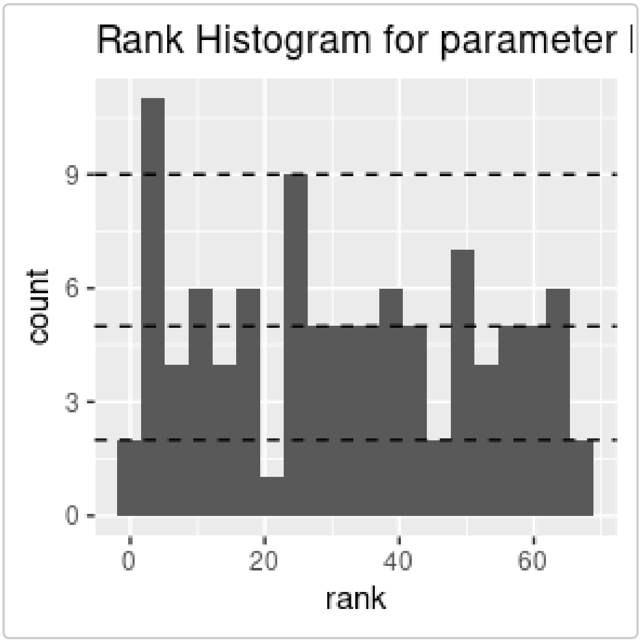
Once we have posterior predictive samples of y, we can retrieve parameters by refitting on the simulated y data. We'll run 200 iterations, therefore `200 * thin` samples for each parameter. This time, the simulated y data will be substituted as `y` into the data list:

```
# retrieve parameters by refitting on simulated y data
theta_post <- sbc_obj$sample_theta_bar_y(sampled_y, data=data,
        pars=list("lambda"), fit_iter = 200)
```

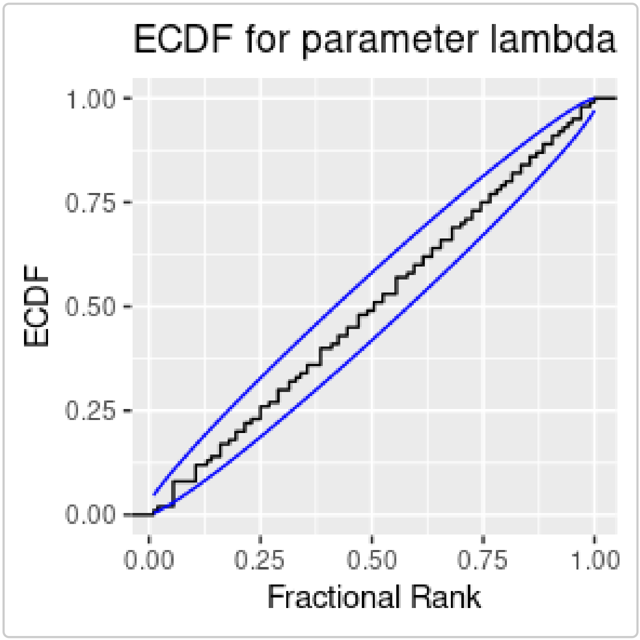
Example model - calculate ranks and plot results

Once we have the retrieved theta samples, we can use them in conjunction with prior predictive samples of theta to calculate rank statistics, and plot results:

```
rank <- calculate_rank(theta_prior, theta_post, thin = thin)  # thinning factor of
        3
plot_hist(rank, "lambda")
```



```
plot_ecdf(rank, "lambda")
```



`sample_all()`, a single function that does all the sampling for you

If you don't find the need to save intermediate sampling results, `SBCModel` provides a `SBCModel$sample_all()` function which will return only rank statistics.

```
model <- cmdstan_model(write_stan_file(stan_code))
```

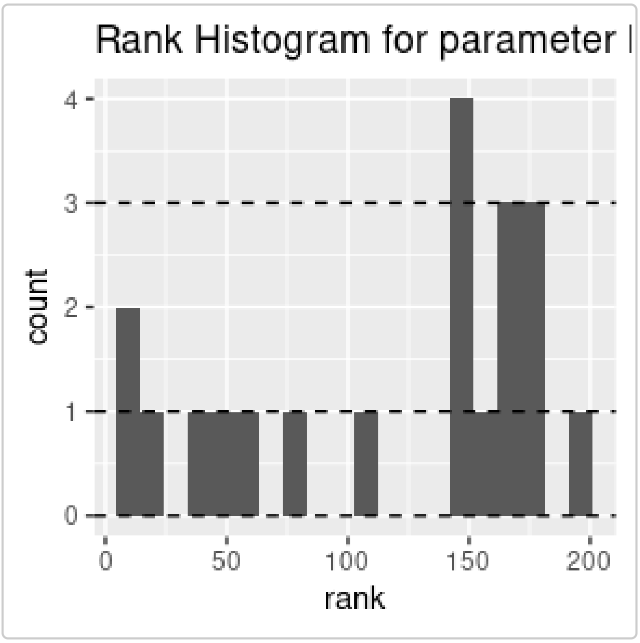
```
## Compiling Stan program...
```

```
# alternatively, use rstan:
# model <- stan_model(model_code = stan_code)
```

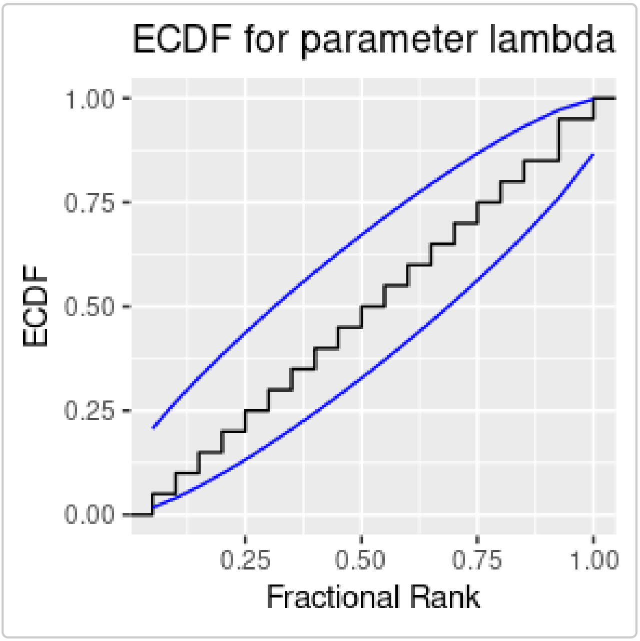
```
single_sbc_obj <- SBCModel$new(name = "poisson", stan_model = model)
easy_ranks <- single_sbc_obj$sample_all(priors=list(), pars=list("lambda"),
        n_iters=20, n_fits=200, data=data, thin=3)
```

If you supply an empty list as `priors` arguments, SBC will sample from the stan model. Otherwise, `priors` will contain the prior distribution functions, which will be used instead.

```
plot_hist(easy_ranks, "lambda")
```



```
plot_ecdf(easy_ranks, "lambda")
```



Experimental feature - Bootstrap sampling

For faster computation, bootstrap could be used to decrease the number of refittings by reusing a single posterior predictive sample vector. This is an experimental feature.

```r
bootstrap_y <- sbc_obj$sample_bootstrap_y_tilde(sampled_y[1, ], 10)  # create 10
        sample vectors from a single sample vector
post <- sbc_obj$sample_theta_bar_y(bootstrap_y, data=data, pars=list("lambda"),
        fit_iter=200)
# Do the rank magic here
```