STAT243_final_project

Weijie Yuan, Franziska Schmidt, Jennifer Wiederspahn

12/10/2018

Adaptive Rejection Sampling

STAT243 Fall 2018 Final Project

Group members: Franziska Schmidt, Weijie Yuan, Jennifer Wiederspahn Github: schfranz

Introduction

Here, we describe a method for adaptive rejection sampling from any univariate log-concave probability density function based on Gilks et al. (1992). The method works without determination of the mode by making use of an envelope and a squeezing function which converge to the underlying density $f(x)$ as sampling proceeds. The assumption of log-concavity of $f(x)$ avoids locating the supremum of the (possibly unnormalized) input function $g(x)$, where $g(x) = cf(x)$. In addition, the need to evaluate $g(x)$ is reduced by using the recently acquired information about $g(x)$, thus reducing the number of evaluations of $g(x)$ even further. For derivative-based Adaptive Rejection Sampling, we assume that $g(x)$ is continuous and differentiable everywhere in domain D and that $h(x) = lng(x)$ exists, s.t. $h(x)$ is concave everywhere in D. Generally speaking, the algorithm can be divided up into the following steps: To initialize the sampling, a set of fixed points is evaluated and the log-density $h(x)$, as well as its derivative are evaluated on the fixed points. Next, these function evaluations are used to construct a piecewise-linear, upper bound h^+ for the log-density function via supporting tangent lines of the log-density at the fixed points. Assuming that $g^+ = exp(h^+)$, sampling $Y\ g^+$ is straightforward because g^+ is piecewise-exponential. More specifically, after having picked $U\ Unif(0, 1)$, Y is accepted if $U \leq exp(h(Y) - h^+(Y))$. Otherwise, another sample is drawn from g^+ and the rejected Y can be added to the initial set of fixed points and the piecewise-linear upper bound h^+, allowing for an adaptive update.

Approach

1. *Main function*

- main adaptive rejection sampling function
- log of the original function
- find starting x_k using local maximum of 'h' function
- initialize output variable
- iterate until we have enough points
- calculate h_k and derivative of h_k
- generate sample points from $s_k(x)$
- carry out rejection test to determine whether we should accept these points and whether we should update these points into original x_k
- cumulative envelop: Calculate areas under exponential upper bound function for normalization purposes, Normalize, Sampling: Generate seeds for Inverse CDF method, Rejection testing, update accpeted points to sample, update x_k

2. *Supporting functions*

- generate intersect z_j
- initialization: check different cases whether lb or ub is Inf and set different initialization - create upper hull in a vectorized fashion and take exponential of it for further sampling from inverse CDF.

- create lower hull in a vectorized fashion
- sample from the envelope $s_k(x)$ using inverse CDF.
- Sample from uniform random distribution.
- Rescale sample value w to area of the selected segment, since area under segment may not equal to 1. Besides, for those unnormalized distribution, this process will normalize it.
- Use inverse CDF of selected segment to generate a sample.
- rejection test: Generate random seed from uniform distribution, carry out squeeze and reject tests to filter sample points, return updateIndicator and acceptIndicator for adding and accepting points in boolean form.
- Return boolean indicator whether to accept candidate sample point
- Update x_k and sample points.

3. *Testing*

- check whether f is positive in range from var_lower to var_upper
- f is continuous
- choose a test point in interval
- check if the sign of boundary values differ
- calculate derivative of a function instead of "grad"
- if limit doesn't exist then we need to stop
- check h(x) is concave
- test for log-concavity
- something to mention: the random number generator iterates over results after 626 unique values which can pose a problem if the user tries to generate a large sample size. We have noticed this but have not implemented a solution since it is default behaivor of R

4. *Breaking Points*

- when input g is not log-concave, continuous and differentiable.
- when input arguments of ars() is not validate
- when users input some extreme distribution, i.e. normal distribution with large mean, it will cause error using self-constructed 'cal_grad' function.
- when input bounds are not validate, i.e. function is not differentiable and finite at some points between lower bound and upper bound.

Repository Location and User Instructions

The ars package resides in the Github repository "schfranz/ars" and can be installed in R using `devtools::install_github('schfranz/ars')` and made available with `library(ars)`. The package can be tested with `library(testthat); test_package('ars')`. You can get additional information by typing `?ars` or `help(ars)`.

The development repository is called "schfranz/STAT243-Final-Project" if you are interested.

Package Overview

This is an overview over the most relevant files and directories in the `ars` package:

- ars/
  - R/
    - arsFunction.R
    - supportingFunctions.R
  - inst/tests/ - testthat/

```
      - testArsInputs.R
  + testArsInputs.R
  + testArsOutputs.R
```

+ `testSuppFunctions.R`

– man/ - ars.Rd

The package directory `ars` contains three relevant folders: `R`, which contains the main function and supporting functions, `inst`, which forces the installation of all tests contained in `tests/testthat`, and `man`, which contains information for the package's help functions.

Contributions

Weijie Yuan: main function, supporting functions and some test samples *Franziska Schmidt:* Github support, unit testing *Jennifer Wiederspahn:* R package and report writing

All team members contributed to the development of their own and other member's parts via Slack and during meetings.