Using Jaatha with a Custom Simulation Method

Lisha Mathew, Paul R. Staab and Dirk Metzler

December 10, 2013

1 Introduction

We originally designed Jaatha for Demographic Inference in Population Genetics[1]. As the algorithms turned out to work quite well there, we think that it might be useful in other situations as well. Jaatha should work in the following scenario:

- You have data that is (assumed to be) generated by a parametric model and is – at least approximately – a sample from independent Poisson variables under that model.
- You want maximum likelihood estimates for the parameter values generating your data, but the likelihood function is analytically untraceable.
- You can simulate data for different parameter values under your model.

In this document, we will explain how you can use Jaatha in such a situation and provide step-by-step instructions for a toy example. We recommend that you first read “The Jaatha HowTo” or the current publication about Jaatha (Mathew et al. [2013]) to understand how the algorithm works.

The shown example code was produced with Jaatha version 2.2. Please either update this manual or Jaatha if you are using a different version. Please note that the custom simulation interface changed after version 2.1 due to major internal restructurations. We are optimistic that it will remain stable in the future.

2 Toy example

For the sake of simplicity, we will assume that the toy model consists of 30 independent Poisson variables, where the first ten have mean x, the second ten have mean y and the last ten have mean z, with $x, y, z \in (0, \infty)$. For given values of x, y and z we can simulate data under this model with the function:

```
sampleFromModel <- function(x, y, z) {
    return(c(rpois(10, x), rpois(10, y), rpois(10, z)))
}
```

[1] Please consult Jaatha’s other vignette, “The Jaatha HowTo”, if you want to use Jaatha for Demographic Inference.

Assume we have observed data that originated from a model with true but unknown parameters $x = 3$, $y = 5$ and $z = 7.5$. Lets try to estimate these values again from the data.

```
set.seed(5)
data.observed <- sampleFromModel(3, 5, 7.5)
data.observed

##  [1]  2  4  6  2  1  4  3  4  6  1  4  5  4  5  3  3  4  8
## [19]  5  7 11  9  5  5  5  7  7 13  5 12
```

Of course, the arithmetic mean is a well-known unbiased, maximum likelihood estimator for x, y and z

```
c(x = mean(data.observed[1:10]), y = mean(data.observed[11:20]),
    z = mean(data.observed[21:30]))

##   x   y   z
## 3.3 4.8 7.9
```

but for the sake of this example, we will try using Jaatha for the estimation.

3 Configuring Jaatha

We need three objects to run Jaatha with our model:

- A list of observed summary statistics `sum.stats`,
- a function `sim.func` that simulates data according to our model and
- A $n \times 2$-Matrix `par.ranges` that gives the minimal and maximal values for the n model parameters.

3.1 Observed summary statistics

Since version 2.2 Jaatha supports using different independent groups summary statistics, where each group either is an array of independent Poisson variables or a vector valued transformation of an array, where the result vector again consists of independent Poisson variables. The two types of summary statistics are call `poisson.independent` and `poisson.transformed`, respectively. In our case, we can just stick to the first case.

To import the summary statistics in Jaatha, we create a list for each summary statistic, in which `method` gives on the two types above, and `value` gives the observed values

```
poisson.vector <- list(method = "poisson.independent", value = data.observed)
```

and combine all summary statistics into a list, indexed by a name

```
sum.stats <- list(poisson.vector = poisson.vector)
```

If we wanted to use the transformed type, say we wanted just to use the sum of all 30 vectors as summary statistics, we additionally need to give the transformation

```
poisson.sum <- list(method = "poisson.transformation", transformation = sum,
    value = data.observed)
```

and add this list to `sum.stats` instead.

3.2 Simulation function

The function for the simulation must take exactly two arguments, first the jaatha object and second a complete set of the n model parameters. The function should do the simulation and return the simulated summary statistics as a list, again indexed by the same names as in `sum.stats`. In our example $n = 3$ and we can write a simple wrapper functions for `sampleFromModel` to give it the required form:

```
sim.func <- function(sim.pars, jaatha) {
    list(poisson.vector = sampleFromModel(sim.pars[1], sim.pars[2],
        sim.pars[3]))
}
# An example call
sim.func(sim.pars = 1:3)

## $poisson.vector
##  [1] 1 0 0 0 0 1 1 1 1 1 3 1 3 2 4 2 3 0 3 2 2 3 6 7 6 2 2 2
## [29] 1 1
```

Here we don't need the `jaatha` parameter. It could be used for passing additional parameters to the simulation function. We will explain how to do this in a moment.

3.3 Parameter ranges

The matrix that gives the parameter ranges must be of dimension $m \times 2$. Each row consists of the minimal and maximal values of the range that the parameter can take. Restricting the range of the parameters is required at the moment. Jaatha also reads the names of the parameters from this matrix, so providing row names makes the output of Jaatha easier to read, but is not required for it to run. In our example, the matrix could look like this:

```
par.ranges <- matrix(c(0.1, 0.1, 0.1, 10, 10, 10), 3, 2)
rownames(par.ranges) <- c("x", "y", "z")
colnames(par.ranges) <- c("min", "max")
par.ranges
```

```
##   min max
## x 0.1  10
## y 0.1  10
## z 0.1  10
```

3.4 Initialization

Now, we can use these three objects to initialize Jaatha:

```
library(jaatha, quietly = TRUE)
jaatha <- new("Jaatha", sim.func, par.ranges, sum.stats)
```

From this point on, we can continue as described in "The Jaatha HowTo" by calling `Jaatha.initialSearch` and a `Jaatha.refinedSearch`. We will do so in a moment, but first cover a few open points.

First, if you want pass additional variables to the simulation function, you can use the `opts`-slot of the jaatha object, which holds a normal list. So say you don't want to hard code the ten variables per parameter in our model, you could call

```
jaatha@opts[["variable.number"]] <- 10
```

after you created the jaatha object with `new`, and use

```
jaatha@opts[["variable.number"]]

## [1] 10
```

in `sim.func` to access it again. Note that this values must be the same for all simulations. It is better to save variables need for the simulation in the Jaatha object rather than in the normal R-Workspace, as this ensures that Jaatha call are reproducible as long as the same Jaatha-Object is used.

Second, if your simulation requires temporary files, we strongly recommend to use the function `jaatha::getTempFile(``some.identifier'')` to generate a file name. This will makes sure that the different threats of Jaatha don't interact if the program is run on multiple cores in parallel.

Finally, the options `use.shm` for placing temporary files in memory, and `cores` and `sim.packes.size` for parallelization are implement in the base algorithm an can also be used along with your custom simulation function. Just add the options to the `new` call. These options are described in `?Jaatha.initialize`.

4 Running Jaatha

So, lets see how Jaatha performs in your toy example:

```
jaatha <- Jaatha.initialSearch(jaatha, 100, 2)
```

```
## *** Searching starting positions ***
## Creating initial blocks ...
## *** Block 1 : 0.1-1 x 0.1-1 x 0.1-1
## Best parameters 1 1 1 with estimated log-likelihood -229
##
## *** Block 2 : 0.1-1 x 0.1-1 x 1-10
## Best parameters 1 1 8.343 with estimated log-likelihood -129.7
##
## *** Block 3 : 0.1-1 x 1-10 x 0.1-1
## Best parameters 1 4.105 1 with estimated log-likelihood -175.1
##
## *** Block 4 : 0.1-1 x 1-10 x 1-10
## Best parameters 1 4.774 7.445 with estimated log-likelihood -84.82
##
## *** Block 5 : 1-10 x 0.1-1 x 0.1-1
## Best parameters 2.343 1 1 with estimated log-likelihood -203
##
## *** Block 6 : 1-10 x 0.1-1 x 1-10
## Best parameters 3.416 1 8.482 with estimated log-likelihood -96.39
##
## *** Block 7 : 1-10 x 1-10 x 0.1-1
## Best parameters 3.736 5.722 1 with estimated log-likelihood -154.4
##
## *** Block 8 : 1-10 x 1-10 x 1-10
## Best parameters 3.504 4.588 7.948 with estimated log-likelihood -64.52
##
##      log.likelihood     x     y     z
## [1,]         -64.52 3.504 4.588 7.948
## [2,]         -84.82 1.000 4.774 7.445
## [3,]         -96.39 3.416 1.000 8.482
## [4,]        -129.72 1.000 1.000 8.343
## [5,]        -154.40 3.736 5.722 1.000
## [6,]        -175.06 1.000 4.105 1.000
## [7,]        -203.04 2.343 1.000 1.000
## [8,]        -228.98 1.000 1.000 1.000
```

```
jaatha <- Jaatha.refinedSearch(jaatha, 2, 100)
```

```
## *** Search with starting Point in Block 1 of 2 ****
## ------------------
## Step No 1
## Using 108 Simulations
## Best parameters 3.439 4.384 8.319 with estimated log-likelihood -63.63
## No sigificant score changes in the last 1 Step(s)
##
## ------------------
## Step No 2
## Using 174 Simulations
## Best parameters 3.311 4.04 8.711 with estimated log-likelihood -63.68
## No sigificant score changes in the last 2 Step(s)
```

```
##
## -----------------
## Step No 3
## Using 177 Simulations
## Best parameters 3.21 3.976 7.763 with estimated log-likelihood -64.33
## No sigificant score changes in the last 3 Step(s)
##
## -----------------
## Step No 4
## Using 191 Simulations
## Best parameters 3.351 4.117 7.742 with estimated log-likelihood -63.61
## No sigificant score changes in the last 4 Step(s)
##
## -----------------
## Step No 5
## Using 250 Simulations
## Best parameters 3.403 4.281 7.739 with estimated log-likelihood -63.63
## No sigificant score changes in the last 5 Step(s)
##
## *** Finished search ***
## Score has not change much in the last 5 steps.
## Seems we have converged.
##
## Calculating log-composite-likelihoods for best estimates:
## * Parameter combination 1 of 6
## * Parameter combination 2 of 6
## * Parameter combination 3 of 6
## * Parameter combination 4 of 6
## * Parameter combination 5 of 6
## * Parameter combination 6 of 6
##
## *** Search with starting Point in Block 2 of 2 ****
## -----------------
## Step No 1
## Using 108 Simulations
## Best parameters 1.122 4.255 7.699 with estimated log-likelihood -75.88
##
## -----------------
## Step No 2
## Using 129 Simulations
## Best parameters 1.184 4.185 8.048 with estimated log-likelihood -78.03
## No sigificant score changes in the last 1 Step(s)
##
## -----------------
## Step No 3
## Using 171 Simulations
## Best parameters 1.329 4.696 8.121 with estimated log-likelihood -72.74
##
## -----------------
## Step No 4
## Using 136 Simulations
## Best parameters 1.491 4.674 8.094 with estimated log-likelihood -69.74
##
## -----------------
## Step No 5
## Using 161 Simulations
## Best parameters 1.673 4.166 8.345 with estimated log-likelihood -68.89
##
## -----------------
## Step No 6
## Using 128 Simulations
## Best parameters 1.877 4.147 8.77 with estimated log-likelihood -70.27
## No sigificant score changes in the last 1 Step(s)
##
## -----------------
## Step No 7
## Using 149 Simulations
## Best parameters 1.992 4.653 8.107 with estimated log-likelihood -67.05
## No sigificant score changes in the last 2 Step(s)
##
## -----------------
## Step No 8
## Using 139 Simulations
## Best parameters 2.235 4.147 8.261 with estimated log-likelihood -64.08
##
## -----------------
## Step No 9
## Using 132 Simulations
## Best parameters 2.301 4.653 8.047 with estimated log-likelihood -65.32
## No sigificant score changes in the last 1 Step(s)
##
## -----------------
## Step No 10
## Using 161 Simulations
## Best parameters 2.582 4.575 7.916 with estimated log-likelihood -64.98
## No sigificant score changes in the last 2 Step(s)
##
## -----------------
## Step No 11
## Using 176 Simulations
## Best parameters 2.897 4.716 7.767 with estimated log-likelihood -62.55
## No sigificant score changes in the last 3 Step(s)
##
## -----------------
## Step No 12
## Using 146 Simulations
## Best parameters 2.942 4.964 7.251 with estimated log-likelihood -63.08
## No sigificant score changes in the last 4 Step(s)
```

```
##
## ------------------
## Step No 13
## Using 175 Simulations
## Best parameters 2.86 5.088 7.462 with estimated log-likelihood -64.42
## No sigificant score changes in the last 5 Step(s)
##
## *** Finished search ***
## Score has not change much in the last 5 steps.
## Seems we have converged.
##
## Calculating log-composite-likelihoods for best estimates:
## * Parameter combination 1 of 10
## * Parameter combination 2 of 10
## * Parameter combination 3 of 10
## * Parameter combination 4 of 10
## * Parameter combination 5 of 10
## * Parameter combination 6 of 10
## * Parameter combination 7 of 10
## * Parameter combination 8 of 10
## * Parameter combination 9 of 10
## * Parameter combination 10 of 10
##
##
## Best log-composite-likelihood values are:
##   log.cl block     x     y     z
## 3 -63.77     2 2.942 4.964 7.251
## 1 -63.78     1 3.439 4.384 8.319
## 5 -63.90     1 3.403 4.281 7.739
## 6 -64.10     2 2.860 5.088 7.462
## 6 -64.26     1 3.403 4.281 7.739
```

Hence, Jaatha's best estimates are quite comparable to the direct maximum likelihoods estimates for this simple model.

References

Lisha A. Mathew, Paul R. Staab, Laura E. Rose, and Dirk Metzler. Why to account for finite sites in population genetic studies and how to do this with jaatha 2.0. *Ecology and Evolution*, 2013. doi: 10.1002/ece3.722. URL http://onlinelibrary.wiley.com/doi/10.1002/ece3.722/abstract.