# Implementations of Global Sensitivity Analysis Methods in Julia

**Daniel Alfonsetti**
alfonset@mit.edu
18.337J/6.338J Final Project
18 Dec. 2020

## Abstract

Global sensitivity analysis (GSA) seeks to understand how distributions of the input parameters relate to distributions of the output parameters, and is thus different from local sensitivity analysis which is concerned with measuring sensitivity of the output at specific points of the input. In this project we seek to implement four global sensitivity methods in Julia that have not been implemented previously: fractional factorial sampling, RBD-FAST, EASI, and the delta moment-independent measure. Time and memory benchmarking are performed for each, and numerical experiments from the methods' corresponding research papers are recreated to help assert correctness when applicable. The code for this project can be found at: https://github.com/danielalfonsetti/18337FinalProject

## 1 Fractional Factorial

Fractional factorial is a principled method to sample the input space without having to test all possible combinations of the levels of parameters. For example, if we have k parameters and we wanted to test 2 levels for each parameter, a so-called "full factorial design" would entail running $2^k$ simulations. Running this many simulations for a model with many parameters can often be too expensive, and so the fractional factorial method allows us to only use a fraction of these.

In this project, only a 2-level design is concerned. The implementation is based on the theory presented in Andrea Saltelli's *Global Sensitivity Analysis: The Primer*. For a 2-level design, the design matrix generated by a fractional factorial sampling scheme is composed of -1s and 1s. Each row corresponds to one simulation, each column corresponds to a factor, and a -1 entry encodes the 'low-value' level while the 1s encode the 'high-value' level. For each parameter, the user will have to specify what numerical value a 'low' and 'high' value corresponds to so the design matrix can be converted to an actual input matrix for the model.

### 1.1 Fractional Factorial Correctness

In order to verify the correctness of the Julia implementation, the method was tested on the function of Sobol', a common benchmark for global sensitivity analysis. The exact form of the function taken was one with 8 input factors, and was parameterized by the vector [0, 1, 4.5, 9, 99, 99, 99, 99]. Each number is inversely related to how important a factor is.

Using a 'low-value' of 0.1 and a high value of '0.8', the estimated main effects for these parameters were: [-0.272, -0.159, -0.065, -0.037, -0.004, -0.004, -0.004, -0.004], The observed magnitudes of these estimated main effects are consistent with the importance of their corresponding factors in the function of Sobol'.

### 1.2 Fractional Factorial Benchmarking

The generation of the design matrix can be defined recursively as the concatenation of Hadamard matrices. A naive implementation may build up the design matrix by allocating smaller Hadamard matrices at each recursive step, and then concatenating them together. However, a more performance oriented approach would be to allocate the size of the final matrix ahead of time, and then index into the parts of the matrix that should be updated at each step of the recursion. We will call this method the "expanding window" method, in contrast to the "recursive method". A performance comparison between the two approaches is shown in figure 1.

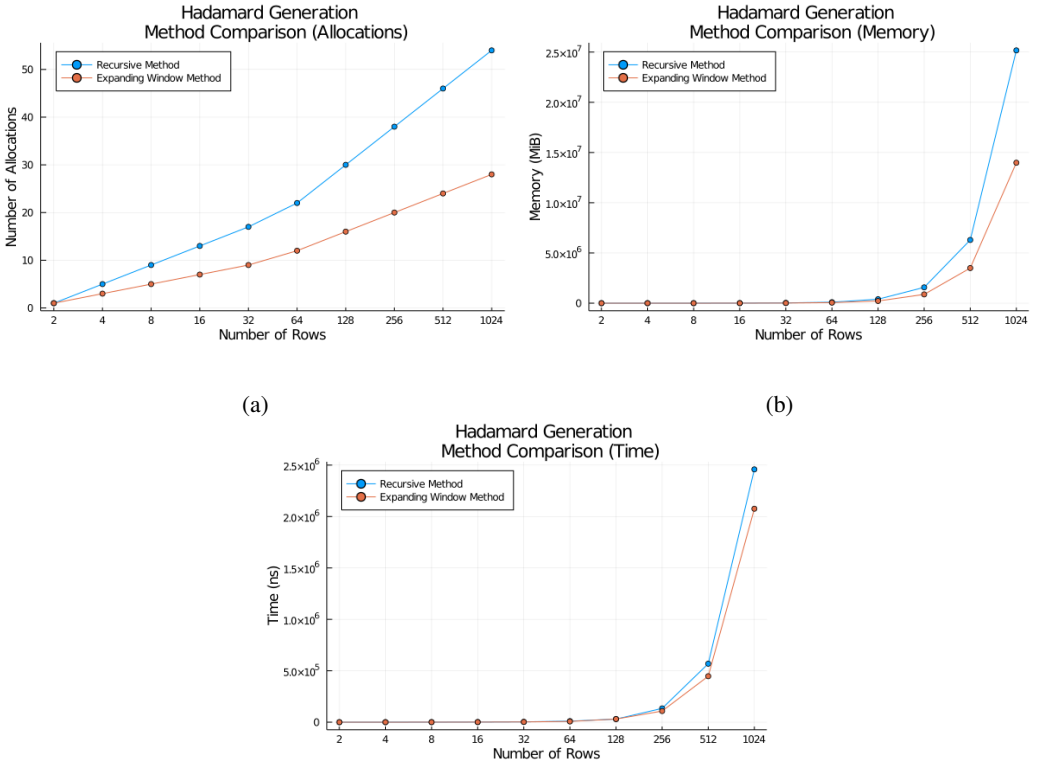



Figure 1: Benchmarks for Hadamard Matrix Generation

For a small number of rows being generated, the difference between the two implementations is negligible. However, as the number of rows increases, the performance benefits clearly increase as well. The expanding window method's advantage stem's from its reduced number of heap allocations. This reduced number of allocations also has the consequence of reducing the amount of garbage collection that has to be performed, as observed through flame graph profiling.

## 2 RBD-FAST

Instead of creating a design matrix, a different class of GSA methods attempt to quantify the fraction of variance that each input factor contributes to the total variance observed in the output by using spectral methods. One common way to do this is through the Fourier Amplitude Sensitivity Test (FAST) method, where model input factors are sampled in such a way that unique frequencies are imposed on each factor. In this way, these frequencies are carried through to the output of the model, and spectral analysis can be done to derive the influence of each input on the output.

However, in the FAST method, the algorithm to choose the frequencies to be independent is computationally expensive. Moreover, in the standard FAST method, the model needs to be evaluated N*K times, where N is the number of samples and K is the number factors. This is compared to

| Factor Number | Analytic Value | N=500 | | N=1000 | | N=2000 | |
|---|---|---|---|---|---|---|---|
| | | Mean | Std. Dev | Mean | Std. Dev | Mean | Std. Dev |
| Factor 1 | 0.7160 | 0.704 | 0.024 | 0.708 | 0.019 | 0.706 | 0.024 |
| Factor 2 | 0.1790 | 0.184 | 0.030 | 0.181 | 0.032 | 0.187 | 0.025 |
| Factor 3 | 0.0240 | 0.051 | 0.112 | 0.038 | 0.055 | 0.042 | 0.058 |
| Factor 4 | 0.0072 | 0.011 | 0.017 | 0.009 | 0.012 | 0.010 | 0.011 |
| Factor 5 | 0.0001 | -0.005 | 0.01 | -0.002 | 0.006 | -0.001 | 0.002 |
| Factor 6 | 0.0001 | -0.008 | 0.009 | -0.003 | 0.005 | -0.001 | 0.003 |
| Factor 7 | 0.0001 | -0.009 | 0.009 | -0.002 | 0.005 | -0.001 | 0.002 |
| Factor 8 | 0.0001 | -0.003 | 0.013 | -0.003 | 0.006 | -0.002 | 0.002 |

Table 1: Recreation of Table 2 in Tarantola *et al*.

only N model evaluations being needed in Random Balance Design FAST (RBD-FAST), which is the method implemented in this project. In RBD-FAST, instead of applying a different frequency to each factor, a (likely) different random permutation is associated with each factor. Sensitivity indices are then determined for each factor based on rearranging the same set of N output values based on the permutations associated with each factor and then running spectral analysis on the rearranged outputs.

The Julia code for RBD-FAST implemented in this project is based on the theory presented in Tarantola *et al*. (2006) and *Global Sensitivity Analysis: The Primer* by Andrea Saltelli. In Tarantola *et al*. (2006), the authors propose two methods; the method implemented here is the one they refer to as "Hybrid FAST-RBD." The code currently only supports the sampling of inputs that are uniformly distributed on [0,1].

### 2.1 RBD-FAST Correctness

In order to verify correctness of the Julia implementation, an attempt was made to recreate numerical experiments in the Tarantola *et al*. (2006) paper. The function used to test the implementation was the function of Sobol'.

For the first test, the function of Sobol was parameterized by the vector [0, 1, 4.5, 9, 99, 99, 99, 99]. In their paper, the authors estimate main effects for each of the factors across different samples sizes. They run their experiment 50 times for each sample size, and record the mean and standard deviation of the estimated main effects for each factor over the 50 runs. They report results in table 2 of their paper, and the results of the analysis of the Julia implementation performed in this project are shown in table 1. While the means here appear consistent with those observed in the Tarantola *et al*. paper, the standard deviations here are larger and do not decrease with increasing sample sizes (N) like they appear to in the paper.

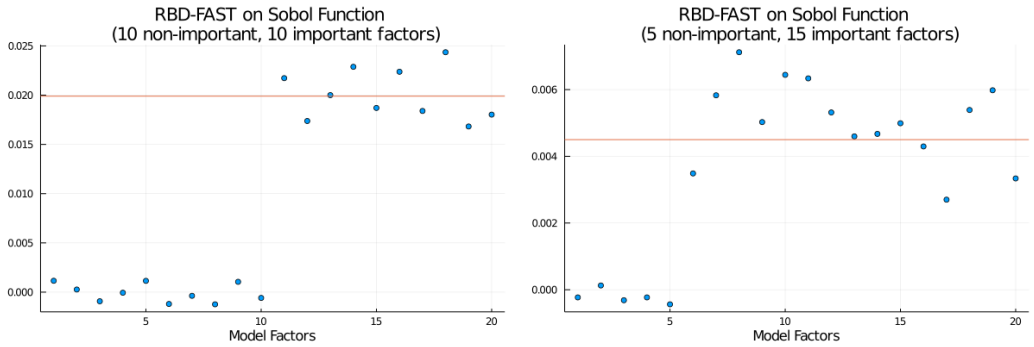
Further analysis of the standard deviations observed in the Julia implementation verified that they did not decrease with increasing sample sizes, which seems incorrect. The code was thoroughly checked for errors, but the reason for this apparent inconsistency was never found.

Regardless, we moved on to try to recreate the other experiments in the Tarantola *et al* paper. These again involved the function of Sobol', but now parameterized with considerably more factors that are important. In figures 2(a) and 2(b), we observe that our RBD-FAST implementation does an adequate job of differentiating the non-important factors from the important factors even when there are many important factors, just as it does in Tarantola *et al*. However, the variance issue seems to persist here, as a non-negligible fraction of subsequent runs of the test summarized in the plots of figure 2 will have incorrect sensitivity estimates.

Finally, we wanted to test convergence to the true analytic values as the number of samples increased. In figure 3, we can see that this convergence property does appear to hold.

### 2.2 RBD-FAST Benchmarking

Profiling analysis shows an extremely small amount of garbage collection occurring, indicating that the implementation is highly optimized. This was made possible through avoiding needless




(a) (b)

Figure 2: RBD-FAST on function of Sobol' with many important factors

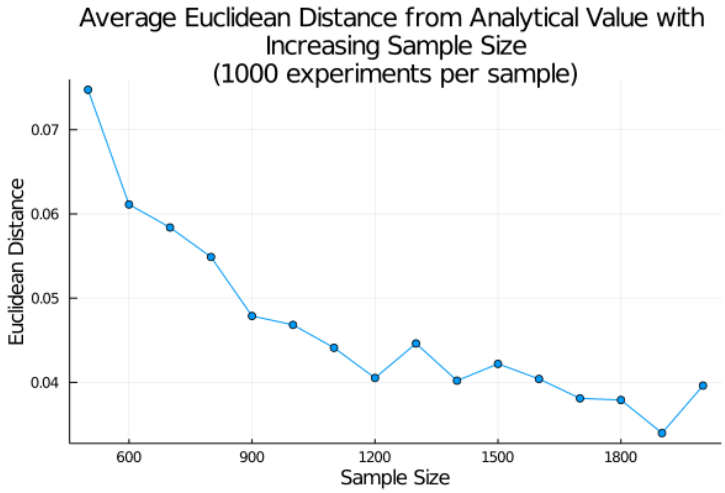


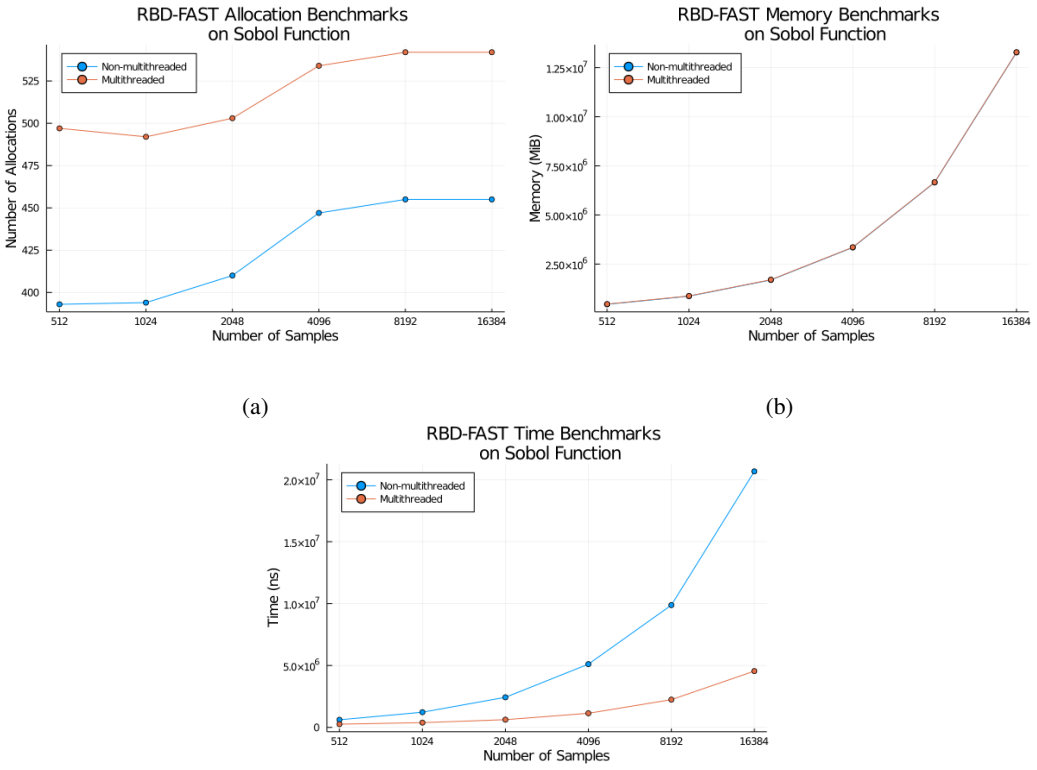

Figure 3: Convergence of the RBD-FAST sensitivity estimates to analytical values with increasing sample size.

allocations and using "views" wherever possible. When the model being used is the function of Sobol', it is observed through profiling analysis that the most time intensive steps are the generation of the input samples, the generation of the permutation vector that puts the parametric variable vector back in sorted order, and the fast fourier transform needed for the spectral density analysis. When a more complicated model is used, the model evaluation for each of the N samples would almost surely be the most time intensive step. Fortunately, all four of these steps can be parallelized via multithreading, since the sensitivity index calculations for each factor are independent. A comparison between the multithreading and non multithreading versions are shown in figure 4.

We can clearly see that as the number of samples increases, the time advantage of multithreading becomes more apparent. Moreover, it is shown that the total amount of memory used is not significantly affected by multithreading.

# 3 EASI

EASI is a method of estimating sensitivity indices from pre-computed model evaluations, as described by Elmar Plischke (2010). It is very similar to RBD-FAST, except that instead of input sampling and model evaluation being part of the process, pre-computed data is sorted and shuffled to impose frequencies on the input data. In this way, frequency analysis can be performed on the analogously sorted output without having to evaluate the model. Thus EASI is a post-processing method that




(a) (b)

(c)

Figure 4: Benchmarks for RBD-FAST

can be extremely useful in cases where someone else has already run many evaluations and/or each evaluation is computationally expensive.

The Julia code implemented in this project for EASI is based on the analogous code in the "Sensitivity Analysis Library" ("SALib") in Python, where it is named "RBD-FAST", despite it being different from the algorithm of the same name discussed in the previous section.

### 3.1 EASI Correctness

To help assert correctness of our implementation, we attempted to recreate results from Plischke's paper; in particular, figure 4 from Plischke. We use the function of Sobol' parameterized by the vector [0, 1, 4.5, 9, 99, 99, 99, 99] as the model.

Since EASI requires precomputed model evaluations, we first generated sample data and then ran it through the model. We generated sample sizes of 100, 300, 1000, 3000, 10000, and 30000. Each sample is a set of vectors, where each vector has its components drawn from a uniform distribution on [0,1]. For each vector, the function of Sobol' returns a scalar output.

We perform 150 runs per sample size. Thus, for each sample size, we get a distribution of estimated variance contributions for each factor. In figure 5 are the boxplots showing the distribution of estimates for the 1st and 4th factors. The red line is the true analytical value.

The plots in figure 5 are consistent with what Plischke observed, and we can see that the estimates are clearly converging to the true analytical values as the sample size increases. Moreover, the variance of the estimates decreases, as expected. Also of note is the fact that the bias corrected estimates are behaving as expected, as the means of the bias corrected estimates are much closer to the true analytical value than the mean of the unbiased estimates, for any given sample size.

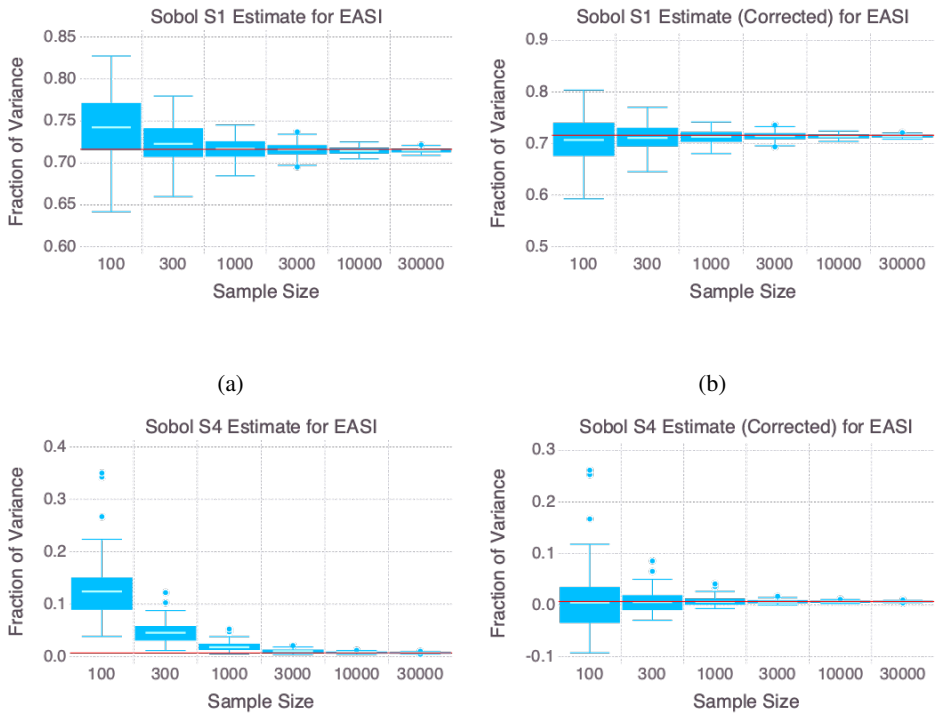



Figure 5: Recreation of figure 4 in Plischke (2013) showing convergence to analytical values as sample size increases, as well as the effectiveness of the bias correction.

### 3.2 EASI Benchmarking

The EASI algorithm can easily be parallelized via multithreading on each input factor. This allows for the shuffling and spectral analysis (the most expensive parts of the algorithm) for each factor to happen simultaneously. A comparison between the multithreaded version and the non-multithreaded version are shown in figure 6; the multithreaded version saves a significant amount of time, without using significantly more memory.

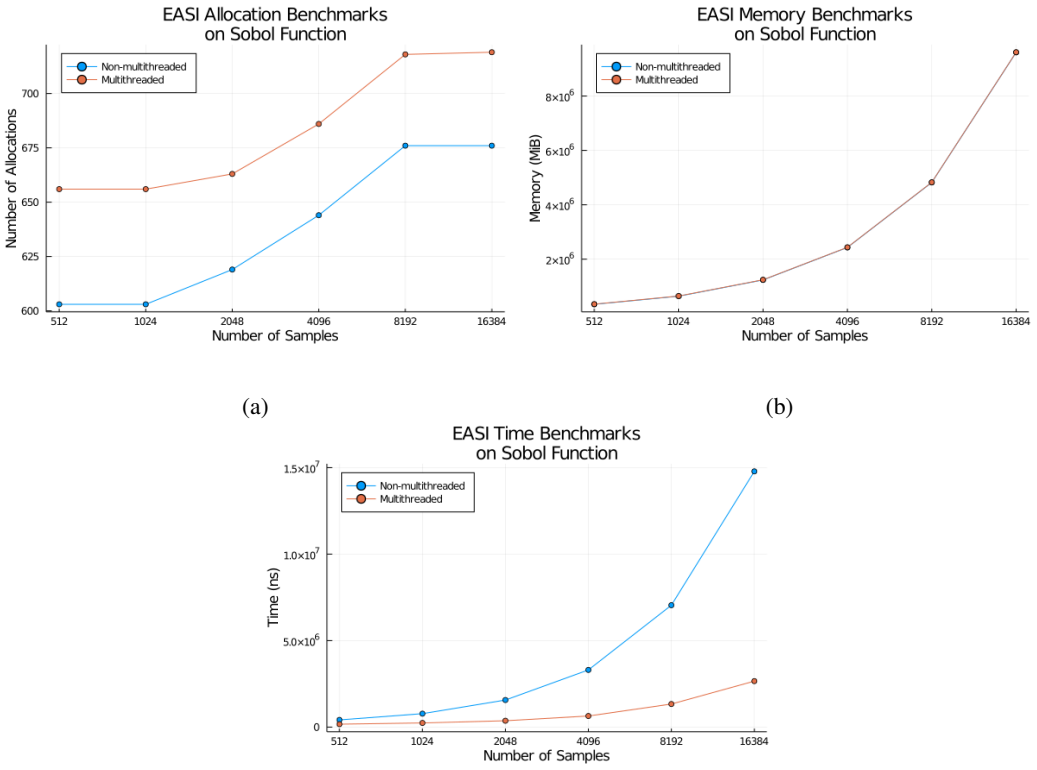
## 4 Delta Moment-Independent Measures

Like EASI, Delta Moment-Independent Analysis is another way to estimate sensitivity indices from pre-computed model evaluations, this time using so-called “delta-indices.” The intuition behind the method is that, given two random variables X and Y, the importance of X on Y can be measured by taking the distance between the distribution of Y and the distribution of Y conditioned on X=x. The Julia implementation made for this project is based on the corresponding code in Python’s “Sensitivity Analysis Library” ("SALib") and the work of Plischke *et al.* (2013).

### 4.1 Delta Method Correctness

In order to verify correctness of the implementation, we attempted to recreate a result from Plischke *et al.* (2013). In their paper, they use the Ishigami function as a benchmark. Like the function of Sobol’, the Ishigami function is a commonly used function for benchmarking GSA methods. It takes in four inputs, and returns a scalar. The fourth input is ‘inactive’ (i.e. it has no effect on the scalar output), and thus the analytical delta sensitivity index is 0.

In figure 7, we attempt to recreate figure 3 of their paper, which shows how the delta sensitivity measures become more accurate with increasing sample sizes for the inactive variable.




(a) (b)

(c)

Figure 6: Benchmarks for EASI

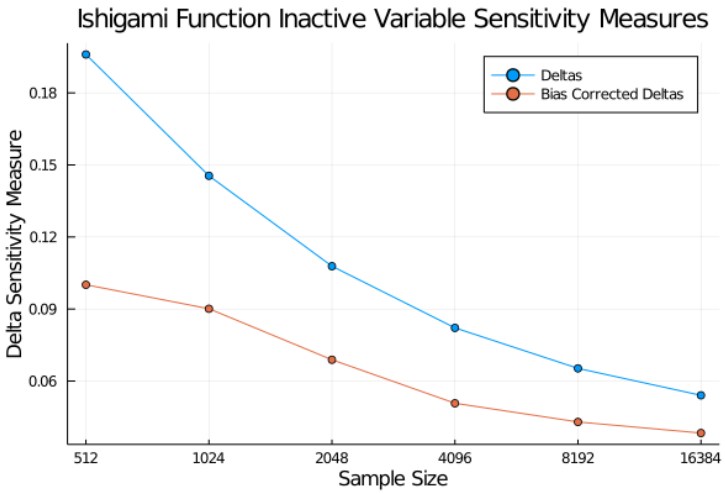



Figure 7

Unfortunately however, there are discrepancies between figure 3 in Plischke *et al.* (2013) and figure 7 here which could not be resolved. While the decreasing trend is the same, the delta indices in Plischke *et al.* (2013) started at a lower value (0.04 instead of 0.18) and the adjusted deltas don't appear to be correcting enough here. The reason for this discrepancy remains unclear despite attempts at debugging. Plischke *et al* (2013). did not submit code with their paper, so cross-referencing was not possible.

After some small numerical experiments comparing the implementation in Python's SALib library to the Julia implementation presented here, it was observed that the underlying kernel density estimation (KDE) methods varied, even when given the same quadrature points and told to use the same bandwidth estimation methods (Silverman). Due to this, it may be possible that the resulting delta indices observed by the Julia implementation presented here and the results of the Plischke *et al*. paper vary due to numerical differences in the underlying kernel estimation methods used. These differences could then be amplified while computing the delta indices, since multiple kernel estimations are performed while calculating each delta index. However, it is impossible to know exactly why there is a discrepancy, since, as stated previously, the authors did not provide code with their paper.

### 4.2 Delta Method Benchmarking

Each delta calculation is independent of the others, so parallel computation of these deltas via multithreading may prove advantageous. Figure 8 contains graphs validating this hypothesis. To create the data, sets of samples of increasing sizes were generated. Each set consisted of vectors of the form $X = [X_1, X_2, X_3, X_4]$ where each $X_i$ is a uniform random variable on the interval $[-\pi, \pi]$. For each input vector, corresponding output scalar values were generated from the Ishigami function to create a vector of outputs $Y$. For each set of samples, the delta index was estimated for each of the four input factors with respect to the output.

From the resulting plots in figure 8, we see that multithreading uses about the same amount of memory but uses less time, even at small sample sizes. Moreover, this time advantage is amplified as the sample size increases. This is as expected, since the overhead of maintaining multiple threads becomes relatively smaller as the work that each thread has to do increases.

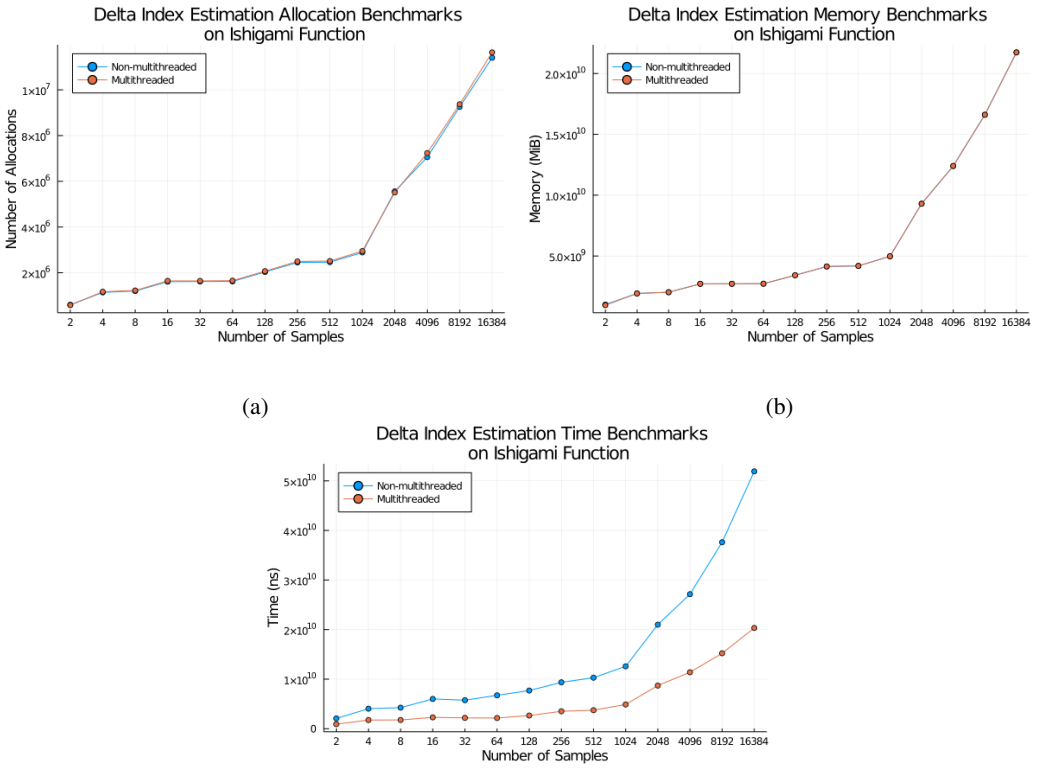



Figure 8: Benchmarks for Delta Index Estimation

One thing to note is that increasing the number of bootstrapping samples (which are used to calculate confidence intervals around the adjusted delta estimate) is expensive, since for each extra bootstrap

sample, you are performing K*(M+1) more kernel density estimations and integration steps (where K is the number of factors and M is the number of partitions). For example, in figure 9 we see that, when the algorithm is run on the Ishigami function (4 factors) and we only partition each factor into 5 classes, increasing the number of bootstrap samples by 50 increases the execution time by approximately 1 second.

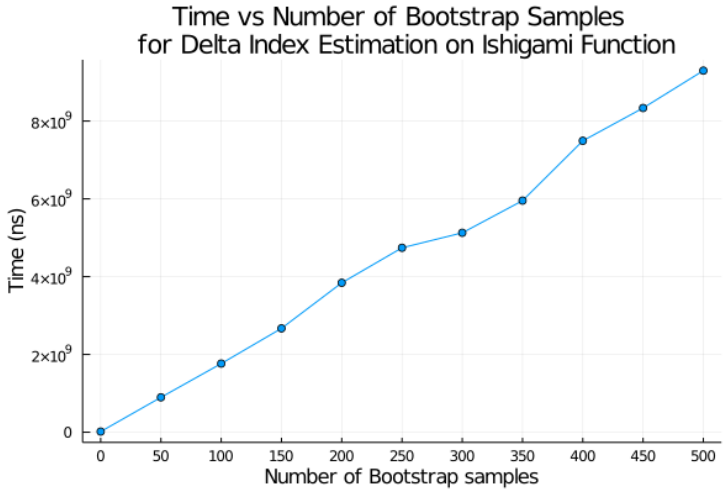



Figure 9

## 5 Conclusion

In this project we have implemented four different GSA methods in Julia: fractional factorial, RBD-FAST, EASI, and Delta Moment Independent Measures. Each method was parallelized where applicable and was made to use few allocations. Unfortunately, for RBD-FAST and the delta measures, some of the experiments in their respective original research papers failed to be reproduced. The EASI implementation proved to be more successful, as the experimental results were reproducible in Julia. We have also successfully implemented a simple - but potentially novel - memory efficient method for the generation of Hadamard matrices for the fractional factorial sampling method.

## References

[1] Saltelli, A. (2008). *Global Sensitivity Analysis: The primer*. Chichester: Wiley.

[2] Tarantola, S., D. Gatelli and T. Mara (2006). "Random Balance Designs for the Estimation of First Order Global Sensitivity Indices." Reliability Engineering and System Safety, 91:6, 717-727.

[3] Plischke, E. (2010). "An effective algorithm for computing global sensitivity indices (EASI)." Reliability Engineering System Safety, 95:4, 354-360.

[4] Plischke, E., E. Borgonovo, and C. L. Smith (2013). "Global sensitivity measures from given data." European Journal of Operational Research, 226:3, 536-550.

[5] Herman, J., W. Usher (2019). *Sensitivity Analysis Library* (SALib).