Referee report

Title: Challenges in Developing Great Quasi-Monte Carlo Software
Authors: Choi, Ding, Hickernell, Rathinavel, and Sorokin
Outlet: MCQMC 2022 Proceedings

General Assessment

I like this paper. It does not really contain new results, but it provides relevant and useful guidelines for the development of QMC software, which I think is very important. It is well-written and clear. I do not have much else to say, aside from the following details. I recommend acceptance after a minor revisions.

Details

1. The example on page 6 is not very clear to me. What is the meaning of the "discretize beam position" here? Where can we see that? What is f? I think there should be a bit more explanations.

2. The paper is essentially all about estimating the expectation of a function by QMC sampling. But (R)QMC is also used for function approximation, simulating Markov chains, density estimation, etc. This would required extra (connected) software tools. For example to maintain and sort and array of Markov chains in Array-RQMC [102, 103]. A lot of this is actually implemented in the SSJ library [101], which also implements many QMC point sets, sequences, and randomizations in the "hups" and "mcqmctools" packages. Maybe this could be mentioned briefly somewhere.

3. Refer. [9]: Maybe the authors can give the reference to the MCQMC 2020 paper on LatNet Builder instead, because it better explains what the software is doing.

4. Section 5: Although Python is very popular and easy to use, perhaps it should be recognized explicitly that it is not the best tool if we want speed. I made the following simple experiment to compare the speeds of qmcpy with that of SSJ. We want to integrate the simple function $f(u_1, u_2) = u_1 + u_2 - 1$ over the unit square, using RQMC with Sobol points + a linear scrambling + a random digital shift, say with $n = 2^{14}$ points. I computed 20,000 independent replicates of the RQMC estimator, with independent randomizations. With SSJ, this took 4.7 seconds on my Thinkpad laptop. With qmcpy, it took about two minutes, which is about 25 times more. I am not asking that the authors report this experiment in their paper, but I think they should mention that Python can be significantly slower than a good Java or C++ implementation. One possibility would be to put a Python layer over a fast implementation.

References

[101] P. L'Ecuyer. SSJ: Stochastic simulation in Java. `https://github.com/umontreal-simul/ssj`, 2023.

[102] P. L'Ecuyer, D. Munger, C. Lécot, and B. Tuffin. Sorting methods and convergence rates for Array-RQMC: Some empirical comparisons. *Mathematics and Computers in Simulation*, 143:191–201, 2018.

[103] F. Puchhammer, A. Ben Abdellah, and P. L'Ecuyer. Variance reduction with Array-RQMC for tau-leaping simulation of stochastic biological and chemical reaction networks. *Bulletin of Mathematical Biology*, 83(Article 91), 2021.