August 14, 2023
Hyrum Hansen

Introduction:

This report outlines progress made on my thesis. I have applied gloptipoly3 and SeDuMi – two MATLAB subroutines designed to work in conjunction to find global optima of multivariate semidefinite polynomials – to the problem of finding highly G-efficient experimental designs.

1 Cursory Problems:

1.1 D-Optimal Calculations

To further embed myself in the problem space, I used my implementation of CEXCH in conjunction with MATLAB's implementation of Brent's optimizer to find highly D-efficient designs. The algorithm was quick to find designs on-par with those from Borkowski's 2003 paper, *Using a Genetic Algorithm to Generate Small Exact Response Surface Designs*.

In general, the D-optimal problem is to find the design to

$$\max |F'F|.$$

Designs can then be used to compute

$$D(F) = 100 * |F'F|^{(1/p)}/N$$

where p is the number of columns in F (also calculated as $\binom{K+2}{2}$ with K the number of parameters) and N is the number of trials.

An additional termination criteria was imposed on the CEXCH algoithm to prevent perpetual execution. Rather than execute until X passes through the algorithm with *no* swaps, I ran the algorithm until each entry of X after CEXCH was within one ten-thousandth of its value before CEXCH.
*** THIS MAY NOT NEED TO BE, THE SWITCH TO NELDER-MEAD ALLOWS INCLUSION OF INITIAL CONDITION WHICH MAY SOLVE THIS ISSUE

1.1.1 Case 1: K=2, N=8

DO 2000 ITERATIONS HERE

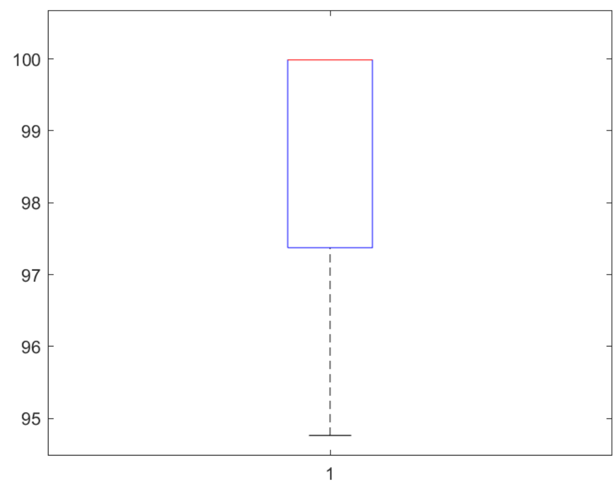

Figure 1: D-efficiencies for CEXCH + Brent's optimizer were near 100% efficient most of the time

1.1.2 Case 2: K=2, N=10

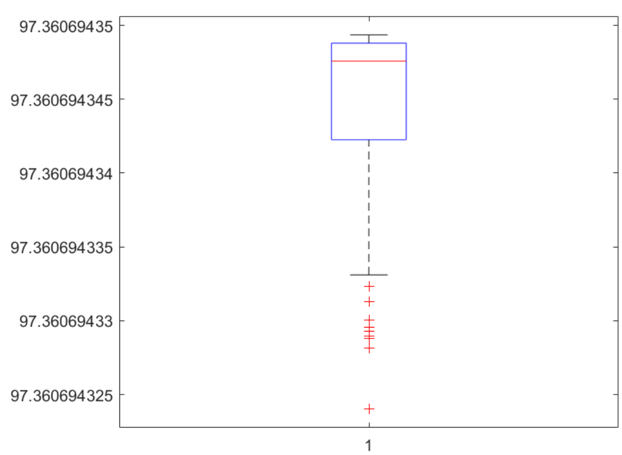

Figure 2: Only really produced one design (up to some rounding errors).

1.1.3 Case 3: K=3, N=12

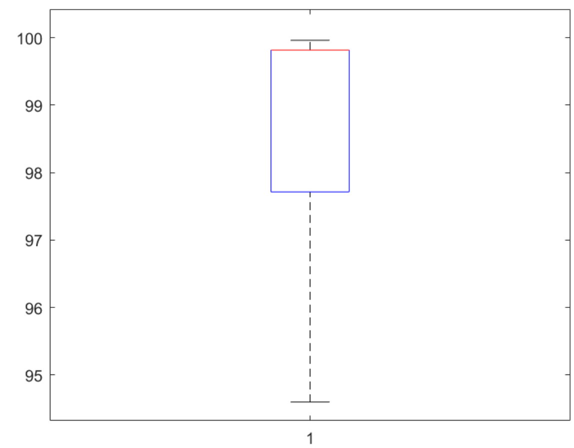

Figure 3: Algorithm performed very well, finding the optimal D design.

1.1.4 Case 4: K=3, N=16

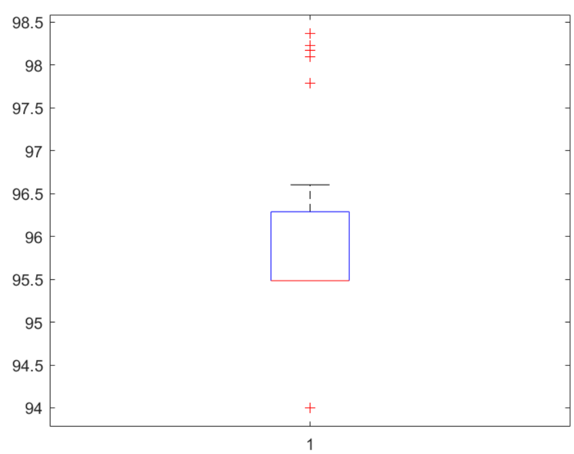

Figure 4: 100 iterations were insufficient to find the optimal design, but most were still highly efficient

1.2 G-score with Gloptipoly

To assess the efficacy of gloptipoly on computing G-scores, I used a dataset of highly G-efficient designs generated using particle swarm optimization (PSO) from the 2022 paper *Improved G-Optimal Designs for Small Exact Response Surface Scenarios: Fast and Efficient Generation via Particle Swarm Optimization.* The algorithm was able to score these designs as expected. The grid search used in conjunction with PSO was never more than one percent away from the true g-efficiency.

K	N	PSO_efficiency	gloptipoly_efficiency	absolute_difference
1	3	100	100	2.142e-07
1	4	82.918	82.918	2.1523e-07
1	5	80.5763	80.5763	1.557e-06
1	6	100	100	4.0706e-07
1	7	91.1669	91.1669	2.1584e-06
1	8	89.1259	89.1259	7.8554e-07
1	9	100	100	6.198e-08
2	6	75.0304	74.3909	0.63959
2	7	80.2387	80.0443	0.19442
2	8	87.943	87.943	1.3584e-06
2	9	86.6336	84.0311	2.6024
2	10	87.4032	86.2993	1.1039
2	11	87.0703	86.6567	0.41355
2	12	88.1719	88.1135	0.05839
3	10	71.4253	70.3796	1.0457
3	11	80.5095	79.5433	0.96612
3	12	83.349	83.1261	0.22287
3	13	86.4558	85.819	0.63673
3	14	89.7063	89.0927	0.61354
3	15	85.993	85.7733	0.21966
3	16	85.7876	85.3949	0.39269
4	15	71.0864	70.6416	0.4448
4	17	73.9012	73.6594	0.24188
4	20	80.1998	79.3124	0.8874
4	24	85.948	85.8531	0.094888
5	21	68.6703	67.8414	0.8289
5	23	73.1925	72.674	0.51859
5	26	75.3124	74.8438	0.46851
5	30	76.1637	75.7131	0.45055

2 Searching for Optimal Designs with CEXCH + Gloptipoly

The search for candidate designs began with the design matrix for a two-factor, n-trial experimental setting. A few challenges were immediate.

1. **Problem:** Perpetual execution with Brent's optimizer. After letting one candidate design pass through CEXCH about 250 times it was determined that a new stopping criteria would have to be defined. Swaps were still being made, even if it was something like

   $$X_{ij} \to X_{ij} + 0.00001$$

   The issue here is that $N \times K$ outer optimizations are needed for each pass which takes a long time, especially when considering that Brent's optimizer calls gloptipoly multiple times for each execution.

   Solution: Criteria for termination would be $|G(X_n) - G(X_{n+1})| < 0.01$. This criteria gave better results, but some matrices still passed through CEXCH nearly 100 times before criteria was met.

   - I tried rounding the matrices to the nearest one-ten-thousandth as well but the above solution seemed to offer quicker execution for comparable performance.

2. **Problem:** Optimizers not arriving to the solutions we intended.

   Solution: I've tried a few things.

   - Lower the tolerance of Brent's optimizer. Currently it sits at 1×10^{-10} which is as fine as the implementation will allow. I also tested 1×10^{-8} and 1×10^{-6}
   - Increase gloptipoly order of relaxation. Currently it sits at 4, no performance gain was found by increasing to 5 and 6. The increase slows it down dramatically.

3. **Problem:** Brent's optimizer in MATLAB doesn't allow an initial condition parameter.

   Solution: Change to a Nelder-Mead simplex approach

2.1 Preliminary Results with Gloptipoly:

$K = 1$ cases. 300 iterations per case were run to minimize computation time. 2000+ should probably be run to ensure optimal design is found.

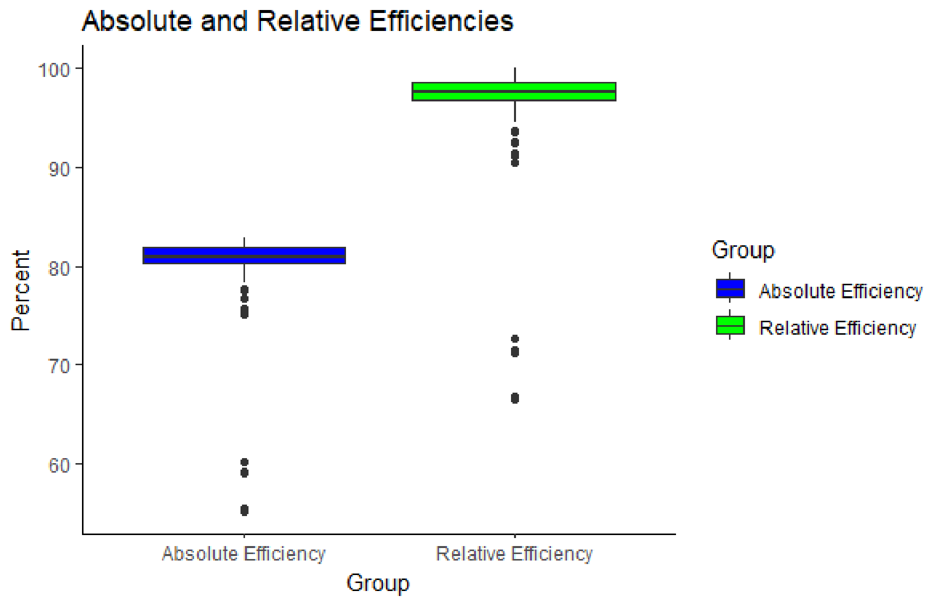

Figure 5: $K = 1$, $N = 4$. The Maximum relative efficiency achieved was 99.99%

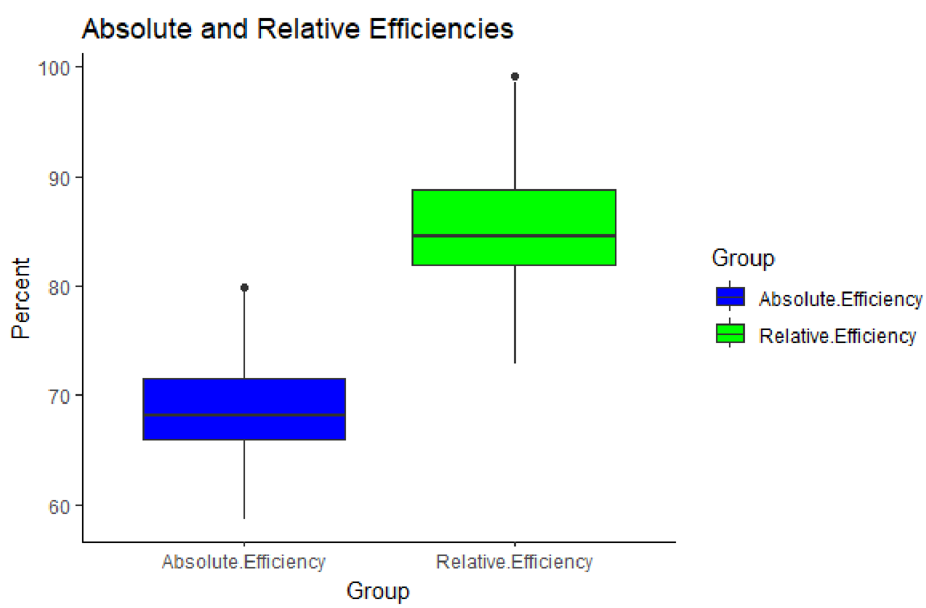

Figure 6: $K = 1$, $N = 5$. The Maximum relative efficiency achieved was 99.15%

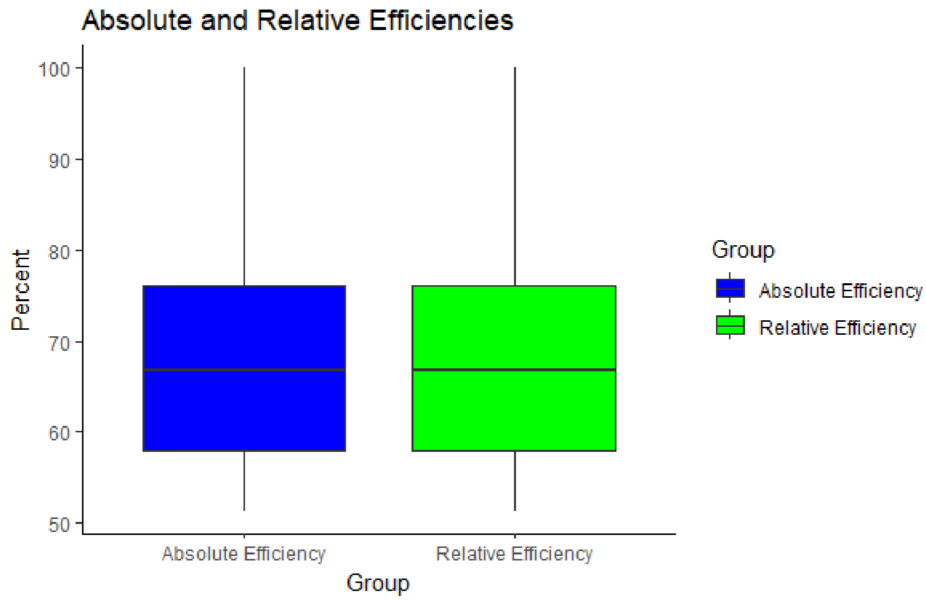

Figure 7: $K = 1$, $N = 6$. The plots appear identical because they are – the best found optimal design has 100 absolute efficiency. The Maximum relative efficiency achieved was 99.97%.

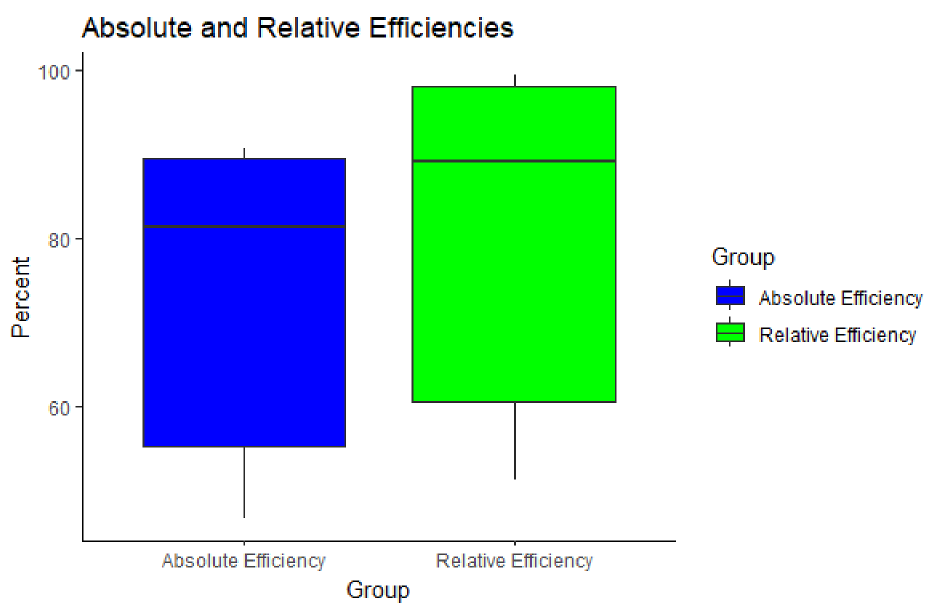

Figure 8: $K = 1$, $N = 7$. Maximum relative efficiency 99.32%.

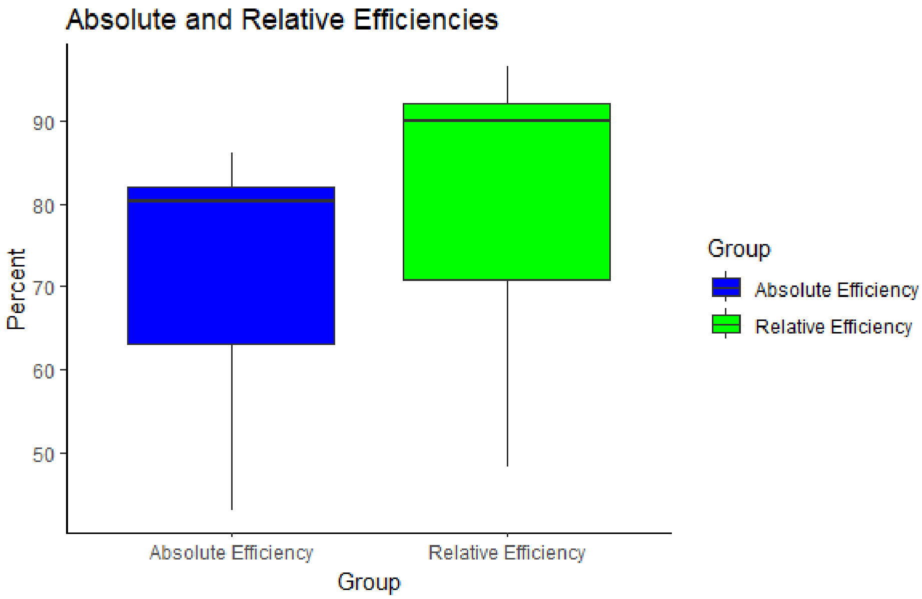

Figure 9: $K = 1$, $N = 8$. Maximum relative efficiency 96.59%.

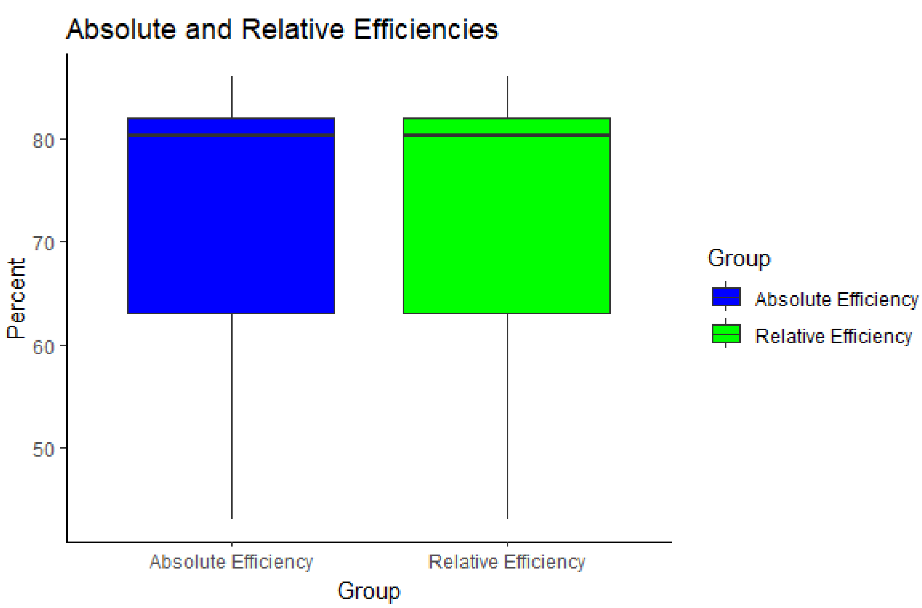

Figure 10: $K = 1$, $N = 9$. Maximum relative efficiency 86.09%.