Team DIRECT-L Optimization Report

Optimization lecture by Martin Zaefferer

Vladyslav Borysenko
11152151
vladyslav.borysenko@smail.th-koeln.de

Pavitra Murugesan
11152523
pavitra.murugesan@smail.th-koeln.de

Adilkhan Kussainov
11152332
adilkhan.kussainov@smail.th-koeln.de

Iurii Skorokhod
11152152
skorokhod.iurii@gmail.com

1. INTRODUCTION

After learning the basics of the optimization and how some of the algorithms work, we were faced with the facts that not all the algorithms are capable of a global search. However, we found out that often this issue can be avoided by using random restarts. Therefore, one of the prevalent questions was which algorithms should we use in various situations, how to configure them, as well as, to see how their performance is affected by such things as random restarts, objective function, random seeds, boundaries and number of iterations.

Basically, our motivation was to investigate the performance of algorithms and compare them against a chosen one. As much as we wanted to test all of them, taking more than one algorithm was not possible in terms of available time. Hence, we were split in the groups, each was to test different algorithms and then reach a conclusion about their performance and whether they are appropriate for our set of objective functions.

Thus, DIRECT-L was selected as a testing subject due to us liking the idea and benefits of the original DIRECT. Our motivation was to explore a locally-biased version of DIRECT in terms of its capabilities to do local search in order to find global extrema.

1.1 DIRECT-L algorithm description

DIRECT algorithm has shown effectiveness on low-dimensional, real-world problems. Those issues are coming from a very wide range of fields such as surgery, genetics, air transport, astronomy, power generation, hydrogen fuel cell management, hard disk design, photovoltaic systems, and many others. There is one drawback of DIRECT, it is less competitive on higher-dimensional problems. Scientists state that DIRECT calculates the global optimum quickly, but it needs much time for high accuracy solution [4].

- The DIRECT (DIviding RECTangles) is a deterministic, gradient-free algorithm based on division of search space into smaller hyper-rectangles to find optima.
- The DIRECT algorithm works fine in both local and global search to efficiently find a global optimizer.
- The DIRECT-L is a form of DIRECT algorithm which is biased towards local search.
- This approach of locally-biased DIRECT was given by Gablonsky [3] and he proposed the following two changes to the existing DIRECT algorithm:
  1. Rather than choosing potentially optimal rectangles, selecting only one of several rectangles arbitrarily that were said to be potentially optimal.
  2. Rather than using the Euclidean 2 norm he introduced the use of infinity norm for selecting the size of rectangles.
- From Gablonsky's paper [3] we are able to understand that:
  - The DIRECT-L method is a variant of the DIRECT algorithm that favors exploration near local minima.
  - The DIRECT-L algorithm is suitable for low dimensional problem with only few global minima.
  - In such conditions, it could be observed that even though in some cases marginally the DIRECT-L outperforms DIRECT algorithm by comparatively requiring less function evaluations.
  - DIRECT-L performs well when the termination budget of function evaluations is set low.

2. METHODS AND MATERIALS

In order to have meaningful results and being able to compare them against the results of other groups, we were given with a default optimizer that is the ***Uniform random search algorithm*** which could be used as something to compare against. Also, we selected ***Differential Evolution*** as a supporting optimizer to better see the variation in the obtained results and performance. Then all the three algorithms were used in ***Gaussian Kriging process regression*** with a maximum (log) likelihood estimation under the hood of it.

2.1 Uniform random search (URS)

A random search algorithm stands for one that uses some sort of randomness in it. This can be represented by either

some sort of pseudo-random number generator or probability representation. Additionally, randomness can be introduced via random sampling, noise, random steps and permutations. Most notably such algorithms are called either *stochastic* or *metaheuristic*, as well as, *Monte Carlo methods* [7].

In our case we have a uniform random search due to the fact that we are using a `runif` function which generates random deviates between the given boundaries [1].

Below is the representation of our implementation of uniform random search optimization algorithm.

```
uniformRandomSearch <- function (x = NULL,
 fun, lower, upper,
 control = list(), ...) {
  con <- list(funEvals = 200)
  # default limit on function evaluations
  con[names(control)] <- control
  control <- con

  npar <- length(lower)
  # number of parameters
  xtest <- matrix(runif(control$funEvals * npar,
  lower,upper), npar, byrow = TRUE)
  ytest <- matrix(fun(xtest, ...), 1)
  best_index <- which.min(ytest)
  list(
    xbest = xtest[best_index, ],
    ybest = ytest[best_index],
    count = nrow(xtest)
  )
}
```

2.2 Differential Evolution

Differential evolution (DE) is yet another one stochastic method that is based on the operations with a population. It also is usually used for global optimization problems. Being stochastic means that DE does not employ any gradient methods. By performing more (compared to gradient-based techniques) evaluations the algorithm can, however, search large areas of candidate space to find the minimum. With each iteration the DE mutates each candidate solution, mixes it with other solutions and checks for its fitness and improvements. If none are observed, the solution is discarded and the process is then repeated [5].

In our case there was no need to write the algorithm manually and instead we used it directly within the *SPOT* R-library [2]. In this package the DE algorithm is called `optimDE` and it is, in fact, a default optimization algorithm for Kriging process regression from the same package. It is using the wrapped version of DE algorithm adopted from `DEoptim` R-library [2].

2.3 DIRECT-L

DIRECT-L and DIRECT both start by rescaling the bound constraints to a hypercube. By doing this all dimensions are assigned an equal weight in the search procedure. DIRECT-L was imported from a library called `NLopt R` without needing to create it from scratch. After that it was successfully used within the Kriging model with adjustments to the following parameters [6]:

- `original`: This method decides whether the algorithm will utilize the original implementation by Gablonsky. It is stated that the performance is mostly similar. If is was similar or not, would be revealed in the next chapters of the report. Additionally, using this option disregards `randomized` and `scaled` options.
- `randomized`: Decides if some randomization will be applied to determine which dimension to halve in the case of near-ties. At first, this option was assumed as mutable for our experiments, but it was discarded later on due to the fact it introduced too much randomness as it was also used with random seeds. Therefore, in our experiments randomized=FALSE.
- `nl.info`: Shows internal information of parameters that are used for DIRECT-L algorithm execution (e.g. number of iterations, stop-conditions for algorithm, stored value of objective function, etc). Each internal parameter can be changed/modified using control=list().

2.4 Kriging model

Gaussian Kriging process regression used in our experiments was also imported directly from SPOT package and it is represented by `buildKriging` function. As the documentation states this fucntion builds a Kriging model based on code by Forrester et al. The default settings set the exponents to the value of two and utilize a nugget while using re-interpolation to compensate for the uncertainty estimates of said nugget [2].

Some crucial settings within `buildKriging` are the following [2]:

- `algTheta`: this defines the optimizer used within the Kriging process. By default it is set to work with `optimeDE`, but in our case we were also passing `Uniform random search` and, naturally, `DIRECT-L` to it.
- `budgetAlgTheta`: defines the budget for the algorithm and has a default value of 200. This value is effectively multiplied with the length of the model parameter vector.
- `optimizeP`: a parameter that indicates whether the exponents should be optimized of left at a default value of two.

Of course, there are other parameters, but the above-mentioned ones were the most vital for our experiments.

3. DEMONSTRATION

The main goal of this project is to investigate which of the algorithms (URS, optimDE, directL) performs better. For this purpose data was created by using the following code below:

```
set.seed(1)
n <- 70
lower <- c(-2.5,-1.5)
upper <- c(1.5,2.5)
```

```
x <- cbind(runif(n,lower[1],upper[1]),
           runif(n,lower[2],upper[2]))
f <- function(x){
  20 + x[,1]^2 + x[,2]^2 -
    10*(cos(2*pi*x[,1]) + cos(2*pi*x[,2]))
}
y <- f(x)
```

For consistent results seed() was set to 1; 70 data points were generated.

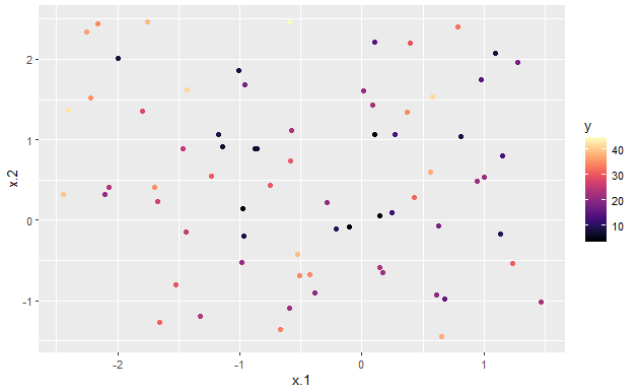

Figure 1: Generated x-y data points

The following generated data is used as input into Kriging model. As it was presented in previous chapters, Kriging model (buildKriging() in SPOT-package for R) as a default parameter for algTheta uses built-in optimDE algorithm. The budgetAlgTheta parameter set to default of 200 units (the further increase could cause performance drop), all further runs/iterations were done without budget modification.

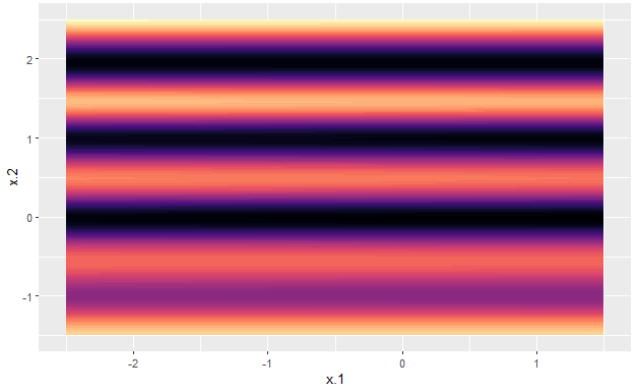

Figure 2: Graphical representation of the model with optimDE

The output of the MLE value with optimDE is 307.9285. Let's run the model with DIRECT-L algorithm. It is defined with such code:

```
Direct_L <- function (x = NULL,
                      fun,
                      lower, upper,
                      control = list(), ...) {

  result <- nloptr::directL(fn=fun,lower = lower,
                            upper=upper,
                            randomized = TRUE,
                            original = FALSE,
                            nl.info = FALSE,...)
  list(
    xbest = result$par,
    ybest = result$value,
    count = 0
  )
}
```

The code for Kriging model with DIRECT-L and default budget:

```
DirectL <- buildKriging(x,matrix(y,,1),control=list(
  algTheta=Direct_L, budgetAlgTheta = 200
))
```

The output of the MLE value with DIRECT-L is 307.7537. The result is almost similar to the optimDE.

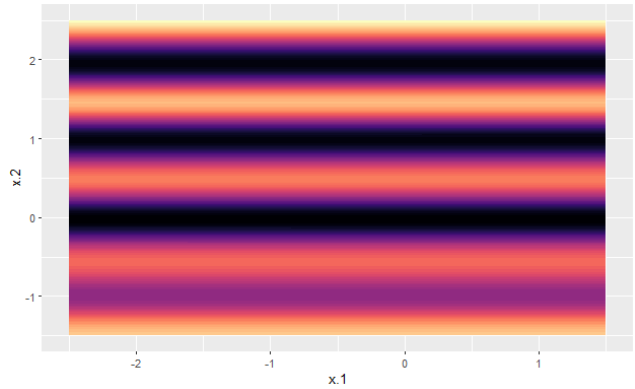

Figure 3: Graphical representation of the model with directL

Same operation is done for URS. The received result: 303.9694. Graphical representation can be seen at Figure 4.

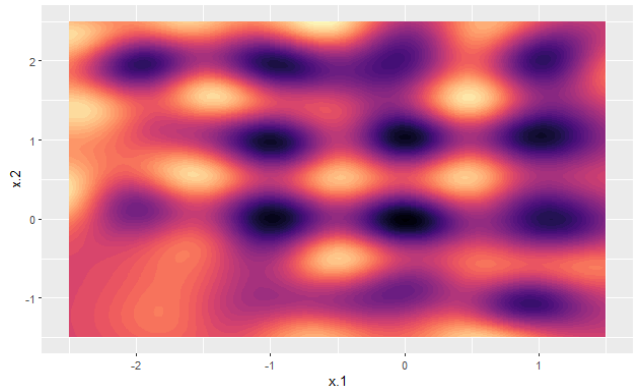

Figure 4: Graphical representation of the model with URS

For better understanding how the algorithms perform in this run we can compare them with the graph of the objective function.

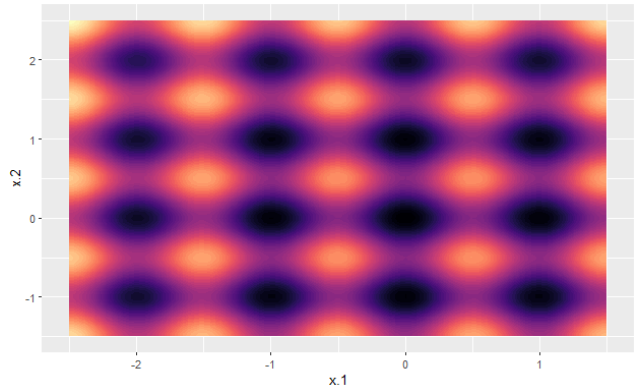

Figure 5: Graphical representation of the objective function

In comparison between Figure 3, Figure 2 and Figure 4, the graph for Uniform random search looks more similar to our searched objective function at Figure 5. Results for a single run of each algorithm at seed 1 are shown at the Table 1.

Algorithm	MLE Values
optimDE	307.9285
URS	303.9694
Direct-L	307.7537

Table 1: MLE Results for URS, optimDE, Direct-L

For the single run URS algorithm performs the best among the others algorithms. OptimDE and DIRECT-L, for this specific run show almost similar results. The main issue for the demonstrated approach: results have its unique range of values that has degree of randomness within the range. For each repeated run we received inconsistent results. At this stage it is hard to make conclusion which algorithm performs the best.

4. RESULTS

4.1 Performance comparison of URS, Direct-L, OptimDE with set.seed(1) and value acquisition of 20 iterations.

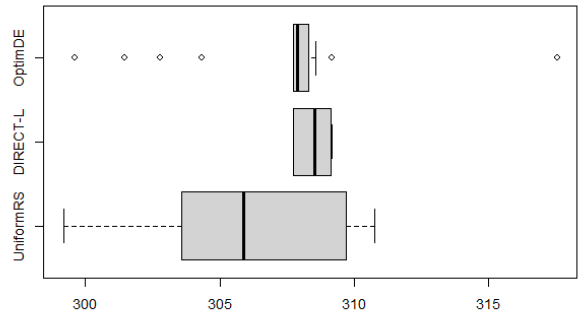

Figure 6: Comparison of URS, DL and DE using boxplot

The approach for data acquisition was changed. Each algorithm was performed in a loop of 20 repetitions. Data was recorded and plotted in a boxplot form to the Figure 6 and Figure 7.

The Figure 6 shows boxplot of the likelihood values of optimDE, DIRECT-L and Uniform random search. We can observe that the spread of Uniform random search is better than that of optimDE and DIRECT-L. And a lot of outliers are observed in optimDE. On comparing minimum values we observe that the minimum value of URS is $\leq$ 299 and in this box plot minimum value is not clearly observable for both optimDE and DIRECT-L. We can assume minimum of optimDE around 307.5 and of DIRECT-L around 308. But in reality we know, that maximum likelihood value of optimDE is 307.9285 and of DIRECT-L is 307.7537. And we are not able to accurately reach these values. Also, among optimDE and DIRECT-L it is hard to determine which one gives the lowest MLE value. So, we will plot the mean values of of all the functions by subtracting the mean values for each function instance.

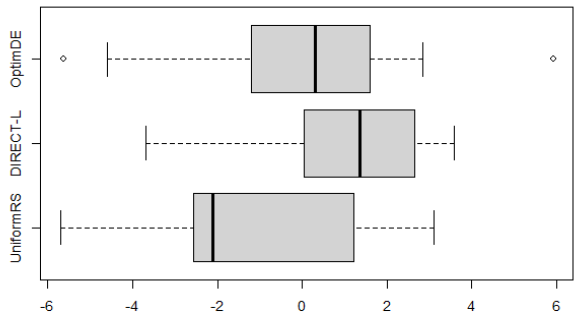

Figure 7: Comparison of mean values for URS, DL and DE using boxplot

At the Figure 7 we can observe that Uniform random search has lowest MLE than optimDE and DIRECT-L. On comparing the median values, the median of optimDE is almost equal to DIRECT-L's minimum value. So about 50% of optimDE's likelihood values were less than likelihood of DIRECT-L's.

4.2 Comparison of means values for URS, DL and DE.

The next step was to investigate if there any change in results for different seed setup. The approach from previous chapter was modified. We had loop run for 100 seeds (from 1 to 100); for each seed run there was a nested loop with 20 iteration for each algorithm. Afterwards, mean values for each nested loop run (mean value for the list of 20 values for every algorithm) were plotted to the Figure 8.

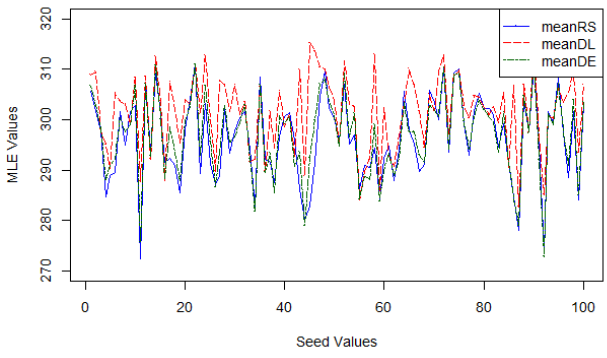

Figure 8: Mean values for URS, DL and DE (100 seed run)

As it was mentioned before, all runs were performed with default parameters for buildKriging (e.g budgetAlgTheta = 200), same applies to the algorithms for algTheta parameter. Additionally, it is worth noticing that 100 seed run with 20 nested iteration (2000 for each algorithm, 6000 in total) was time consuming. Overall 1.5 hours were spent to get the data for the Figure 8. As we can see from the Figure 8, without data sorting it is hard to make conclusion about the behavior of the algorithms with different seed setup. Data was sorted with regards to DIRECT-L values in ascending manner.

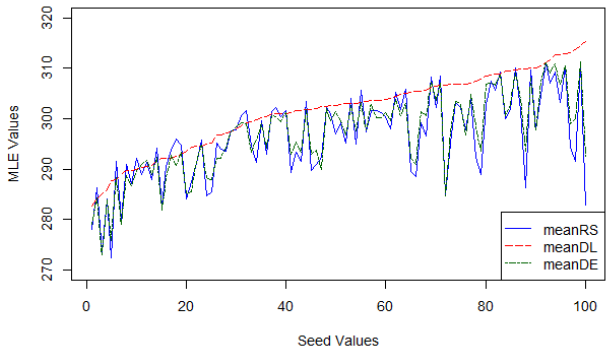

Figure 9: Mean values for URS, DL and DE (100 seed run) - values sorted with regards of meanDL

After analyzing Figure 9, we can say that in most cases optimDE and URS outperform DIRECT-L algorithm. In some instances DIRECT-L performs better, but the overall trend is not in favor of DIRECT-L. For the more clear picture, the cubic regression model was added and plotted to the Figure 10.

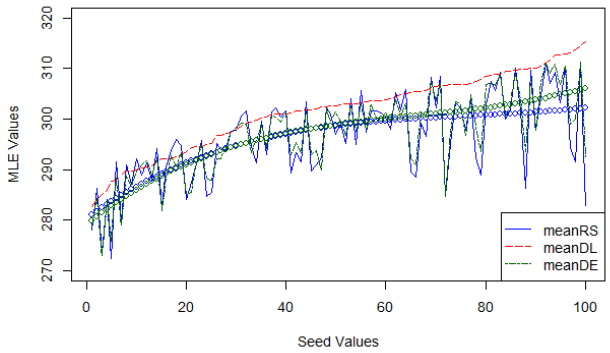

Figure 10: Mean values for URS, DL and DE (100 seed run) - values sorted with regards of meanDL, cubic regression added to DE and URS

The cubic regression model was not added to the meanDL, because it follows the actual line and does not provide any useful information. Furthermore, we can say that for the most part Uniform random search and OptimeDE shows similar results, only with the the high values URS slightly outperforms. So, basically, in normal conditions, URS performs the best of all three algorithms.

4.3 Mean values with optimizeP enabled

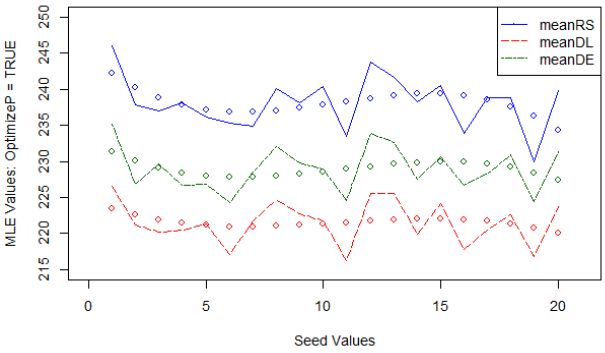

Figure 11: Comparison of mean values for URS, DL and DE with optimizeP = TRUE

Figure 11 shows us that with `opimizeP` enabled DIRECT-L performs significantly better and even wins over the other two algorithms when comparing log likelihood. The data presented was collected from simulations with 20 random seeds due to the fact that, in this case, doing 100 simulaions was taking significantly more time (estimated time around 5 hours). `optimizeP` provides the optimization in terms of exponents as well thus tailoring the algorithm to use the best fitting parameters for the model.

As it was already mentioned the picture is completely flipped: DIRECT-L which previously showed the worst performance of all three optimizers, now outperforms them. Not only that, but even URS which, seemingly, was leading before, loses its position to DE. Differential evolution, on the other hand, takes a position in the middle.

Another interesting point is that when looking at the scale we can see that with `optimizeP` enabled even mean values themselves are smaller that in simulations before, effectively dropping from roughly 300 to roughly 235 now. Which in our case with log likelihood means that a significant performance improvement is observed.

4.4 DIRECT-L while changing states of "original" and "optimizeP"

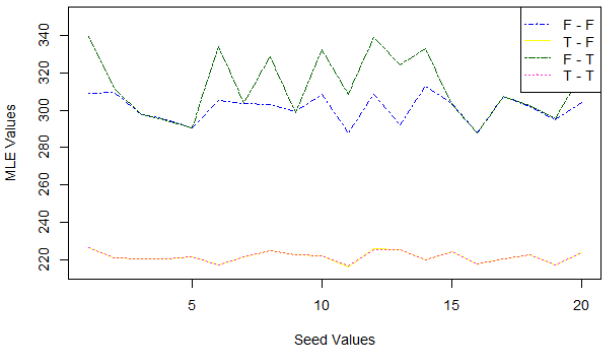

Figure 12: Playing with original and optimizeP within DIRECT-L

The next stage of experiment was to compare the performance of DIRECT-L while changing *original* and *optimizeP* parameters. Figure 12 shows us the results acquired while

iterating within 20 random seeds and changing the corresponding parameters. The left column (column 1) represents a status of *optimizeP* parameter, while the right column (column 2) represents the *original* option. Naturally, **T** means `True` and **F** means `False`.

First of all, it can be derived from the graph that, just like in the previous stage of experiment, the performance of DIRECT-L is significantly worse with *optimizeP = FALSE* having its mean MLE values around 315. Also, in this case, we see that the *original* setting has an influence on the final values. While in some seeds DIRECT-L performs identically with or without *original* enabled, in other cases enabling *original* increases MLE, which is undesirable.

At the same time, as expected, *optimizeP* drastically improves the performance of the optimizer. Not only that, but it also negates any influence from the *original* setting. It can be seen from the lines at the bottom. While not so obvious, there are actually two lines there, which are almost perfectly merged into one.

5. DISCUSSION

Finally, we can say that under the initial experimentation setup Uniform random search and optimDE outperform DIRECT-L by giving the lowest log likelihood values. Furthermore, when we set optimizeP = TRUE, DIRECT-L's performance improved and gave significanlty lower MLE values than Uniform random search and optimDE.

6. REFERENCES

[1] Uniform function - RDocumentation.

[2] Bartz-Beielstein, T. et al. 2021. In a Nutshell – The Sequential Parameter Optimization Toolbox. *arXiv:1712.04076 [cs, math]*. (Mar. 2021).

[3] Gablonsky, J.M. and Kelley, C.T. 2001. A Locally-Biased form of the DIRECT Algorithm. *Journal of Global Optimization*. 21, 1 (2001), 27–37.

[4] Jones, D.R. and Martins, J.R.R.A. 2020. The DIRECT algorithm: 25 years later. *Journal of Global Optimization*. 79, 3 (Oct. 2020), 521–566.

[5] Storn, R. and Price, K. 1997. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization*. 11, 4 (1997), 341–359.

[6] Ypma, J. et al. 2021. Nloptr: R Interface to NLopt.

[7] Zabinsky, Z.B. 2011. Random Search Algorithms. *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.