Statistical Computing in R

Report

Implementation of NMMSO in R

Submitted by

425699 Daniel Carriola
425599 Jannik Hoffjann

Under the guidance of
Dr. Mike Preuss

WESTFÄLISCHE WILHELMS-UNIVERSITÄT MÜNSTER

Information Systems and Statistics

ERCIS

Münster - NRW - Germany

Winter Semester 2015/16

Contents

1 Introduction

In the recent years, R has become the statistical programming language of choice for many scientists. The strength of R, being a domain specific language, has also become one of its weaknesses. Since new research findings in statistical computing are split up over several languages like R, Matlab or SciPy[1], it often becomes difficult to compare new methods with established ones. Because it is also hard to interface those languages due to different architectures and data storage mechanisms there is often no other way than to reimplement new methods in a different programming language to create a common scope.

An example for a well perceived new finding in statistical computing is the NMMSO-Algorithm by Jonathan E. Fieldsend (Fieldsend 2014). It won the niching competition in 2015 held by the CEC and is only written in Matlab. Since the chair 'Information Systems and Statistics' at the Westfälische Wilhelms-Universität Münster, Germany, is mainly concentrating its work on Statistical Computing in R, an implementation of this algorithm became interesting.

As part of this Seminar Project in the context of the Seminar 'Statistical Computing in R', a reimplementation of the NMMSO algorithm in R (nmmso.R) will be presented. During this technical documentation, the general function of the algorithm and the used test cases by the CEC will be shown. Afterwards the structure and used techniques and libraries, as well as problems and pitfalls due to the different behaviors of R and Matlab, will be shown. The documentation will be closed by the benchmarking results and several test cases.

It was the goal of this project to keep up high comparability with the original code, to ensure the correct functionality and easily implement changes to the original codebase in this program. To reach this, unit tests were used where possible and continuous comparison between interim results of the original implementation and nmmso.R where used to ensure functioning. Additionally a benchmarking suite, which builds on the CEC Benchmarking Suite for Niching Algorithm was implemented to evaluate and test the functioning of nmmso.R with the same characteristics as in the original implementation.

2 General Function

The starting point of the project was the paper provided by Dr. Jonathan E. Fieldsend (Fieldsend 2014) on the Niching Migratory Multi-Swarm Optimiser (NMMSO) algorithm. NMMSO is a multi-modal optimiser which relies heavily on multiple swarms that are generated on the landscape of a function in order to find the global optima. It is build around three main pillars: (1) dynamic in the numbers of dimensions, (2) self-adaptive without any special preparation and (3) exploitative local search to quickly find peak estimates (Fieldsend 2014, p. 1).

Multi-modal optimisation in general, does not differ very much from well known and widely discussed single-objective optimisation, but but unlike it, the goal of the algorithms in the multi-modal perspective is not to find just one single optimising point but all possible points

[1]SciPy is a common library for the Python Programming language which brings Statistical Computing capabilities to the language.

(Fieldsend 2014, p. 1). In order to do so, many early multi-modal optimisation algorithms needed defined parameters (Fieldsend 2014, p. 1).

Maybe write a bit more about multi-modal algorithms in general

Newer algorithms fall in the field of self-tuning and try to use different mathematical paradigms like nearest-best clustering with covariance matrices (Preuss 2012) and strategies like storing the so far best found global optima estimators to provide them as parameters for new optimisation runs (Epitropakis et al. 2013). Contradictory to that NMMSO goes the way of many early algorithms and uses the swarm strategy in order to find which store their current (Fieldsend 2014).

In order to do so, NMMSO follow a strict structure, which can be seen in the following pseudo-code

```
nmmso (max_evals, tol, n, max_inc, c_1, c_2, omega)
    S: initialise_swarm(1)
    evaluations := 1
    while evaluations < max_evals:
        while flagged_swarms(S) == true:
            {S, m} := attempt_merge(S, n, tol)
            evals := evals + m
        S := increment(S, n, max_inc, c_1, c_2, omega)
        evals := evals + min(|S|, max_inc)
        {S, k} := attempt_separation(S, tol)
        evals := evals + k
        S := add_new_swarm(S)
        evals := evals + 1
    {X*, Y*} := extract_gbest(S)
    return X*,Y*
```

This structure wasn't modified during the reimplementation of NMMSO to keep comparability and the possibility to fix bugs at a high level. The only newly introduced setting was the possibility to modify c_1, c_2 and omega as parameters from the outside. In the original version those parameters are part of the program code.

	standard value	used value
evaluations	0	0
max_evol	100	100
tol_val	10^-6	10^-6
c_1	2.0	2.0
c_2	2.0	2.0
omega	0.1	0.1

3 CEC Algorithms

3.1 CEC

The IEEE Congress of Evolutionary Computation (CEC) is one of the largest, most important and recognized conferences within Evolutionary Computation (EC). It is organised by the IEEE Computational Intelligence Society in cooperation with the Evolutionary Programming Society and covers most of the subtopics of the EC.

In order to validate the potential of the NMMSO algorithm, it was submitted to the IEEE CEC 2015 held in Sendai, Japan. Here, Dr. Fieldsend was provided with some multimodal benchmark test functions with different dimension sizes and characteristics, for evaluating niching algorithms developed by Dr. Xiaodong Li, Dr. Andries Engelbrecht and Dr. Michael G. Epitropakis (Epitropakis et al. 2013). They state that even if several niching methods have been around for many years, further advances in this area have been hindered by several obstacles; most of the studies focus on very low dimensional multi-modal problems (2 or 3 dimensions) making this more complicated to asses theses methods' scalability to high dimensions with better performance. The benchmark tool includes 20 test functions (in some cases the same function but with different dimension sizes), which includes 10 simple, well-known and widely used benchmark functions, based on recent studies, and more complex functions following the paradigm of composition functions. In the following section, they will be briefly explained:

- `F1: Five-Uneven-Peak Trap (1D)`
- `F2: Equal Maxima (1D)`
- `F3: Uneven Decreasing Maxima (1D)`
- `F4: Himmelblau (2D)`
- `F5: Six-Hump Camel Back (2D)`
- `F6: Shubert (2D, 3D)`
- `F7: Vincent (2D, 3D)`
- `F8: Modified Rastrigin - All Global Optima (2D)`
- `F9: Composition Function 1 (2D)`
- `F10: Composition Function 2 (2D)`
- `F11: Composition Function 3 (2D, 3D, 5D, 10D)`
- `F12: Composition Function 4 (3D, 5D, 10D, 20D)`

All of the test functions are formulated as maximisation problems. F1, F2 and F3 are simple 1D multimodal functions, while F4 and F5 are simple 2D functions and not scalable. F6 to F8 are scalable multimodal functions. The number of global optima for F6 and F7 are determined by the dimension. However, for F8, the number of global optima is independent of the dimension, therefore it can be controlled by the user. F9 to F12 are scalable multimodal functions constructed by several basic functions with different properties (Sphere function, Grienwank, Rastrigin, Weierstrass and the Expanded Griewank's plus Rosenbrock's function). F9 and F10 are separable, and non-symmetric, while F11 and F12 are non-separable, non-symmetric complex multimodal functions. The number of global optima in all of the composition functions is independent of the number of dimensions, therefore can be controlled by the user (Epitropakis et al. 2013).

Maybe write each math equation or the R code

Besides the 12 different functions, also, a $count_goptima$ function was included in order to find the optimal value for each evaluated function in each iteration. Here, some optimal values are already given as well as the number of each value to find. Together with an accuracy rate, the evaluation starts in a cycle and stores the possible optimal values in order to be compared with the expected values using the accuracy rates of 0.1, 0.01, 0.001, 0.0001 and 0.00001 for each different iteration. Once the optimal values are found and after comparing them with the same results of the Matlab implementation, it can be concluded that the new implementation works as expected and is ready for submission.

Explain maybe a little bit more, maybe a pseudocode

https://en.wikipedia.org/wiki/IEEE_Congress_on_Evolutionary_Computation

3.2 Implementation and Pitfalls

During the developing time, an issue raised with the CEC benchmark tool. In order to compare the R implementation of the NMMSO algorithm with the original one, it was mandatory to use this tool to test each of its functions with the new algorithm and compare results. After several complications with the original test suite (these complications will be addressed in the pitfalls' section), it was decided to recode each of the functions as an independent R package to avoid any further complication and having an easier and more trustworthy comparison of the NMMSO algorithm in R.

Pitfalls

When the decision of re-writing the whole benchmark tool in the R programming language was made, some issues came up regarding the way Matlab handles the matrices and vectors. After analysing the Matlab implementation,

Results in R
• f_ 1 : f(1. . . 1) = 120 • f_ 2 : f(1. . . 1) = 5.270904e-92 • f_ 3 : f(1. . . 1) = 0.02501472 • f_ 4 : f(1. . . 1) = 94 • f_ 5 : f(1. . . 1) = -3.233333 • f_ 6 : f(1. . . 1) = -3.180351 • f_ 7 : f(1. . . 1) = 0 • f_ 8 : f(1. . . 1) = 5.671692 • f_ 9 : f(1. . . 1) = 0 • f_ 10 : f(1. . . 1) = -38 • f_ 11 : f(1. . . 1) = -268.6638 • f_ 12 : f(1. . . 1) = -758.9333 • f_ 13 : f(1. . . 1) = -613.5412 • f_ 14 : f(1. . . 1) = -1838.556 • f_ 15 : f(1. . . 1) = -1049.86 • f_ 16 : f(1. . . 1) = -2149.558 • f_ 17 : f(1. . . 1) = -1238.211 • f_ 18 : f(1. . . 1) = -1683.184 • f_ 19 : f(1. . . 1) = -1342.819 • f_ 20 : f(1. . . 1) = -1337.852

---

4 The Implementation

4.1 Structure of the project

In difference to the original implementation it was chosen to split up all single functions of the nmmso algorithm into single files. These were bundled into the standard R package structure to give the possibility to make it available over CRAN[2] in the future. To and give the possibilty to collaborate the project was managed and versioned via Github. The package was tested with testthat[3] and documented with roxygen2[4]. To assure functioning the package was continuously tested with Travis CI.

After analysing the algorithm provided in Matlab by Dr. Fieldsend, it was decided to first translate each of the functions into the R programming language. At first instance, this task seemed to be simple because most of the functions were basically managing matrices and vectors, but later this became a problem that will be covered in the pitfalls' section (4.2) of this paper.

Once all the NMMSO functions existed in R and having the input data, the testing phase started. It has been said, that one of the biggest problems when coding an already existing program into another programming language, is the different behaviours corresponding to each object (in case of an object-oriented language) or its main structure. The first runs came with several errors regarding the matrix generation and handling, slowing down the project in a near future. Using GitHub, it was easier to attack these problems in parallel, having one developer reviewing different functions and the other one, fixing other bugs and continue the testing phase. Also, this was achieved in an easier way, thanks to that each function was coded in an independent R file, making easier and faster the debugging and the fixing of each problem.

4.2 Pitfalls and Problems

In the beginning of the project, after translating each function into the R programming language and in order to start testing each one of them, some data was missing. The algorithm runs with some parameters that had to be given by the CEC so it was necessary to contact the organisation and ask for the files including the benchmarking tool. Also, Dr. Fieldsend was contacted in order to have a better understanding of the general structure of his algorithm. Once the data was completed, the test phase started, trying to get the same results as the original one, but this was not possible at first instance because the output was not generalised and most of the data generated inside the algorithm were random values to be evaluated.

During the implementation of the NMMSO.R algorithm, some problems regarding the language structure came up immediately. After some first test cases, it was discovered how different Matlab and R work with matrices and vectors. In the case of Matlab, every time a value wanted to be added for an inexistent index, this was created automatically and added to the data structure, where no row was existing before, instead of throwing the common "index out of bounds" error. First, it was necessary to compare all these possible behaviors in Matlab so a general way to attack them could be implemented and after few tries, two function-files were created. "add_col.R", as its name says, is in charge to add a new column into a structure, it is just necessary to specify the original structure where it will be added to, the index and the new object containing the information to add. This function considers the cases if the new object is a vector, a matrix or just a single value, so the original matrix

[2]Comprehensive R Archive Network.

[3]https://cran.r-project.org/web/packages/testthat/index.html

[4]https://cran.r-project.org/web/packages/roxygen2/index.html

could be modified and returned as desired. “add_row.R”, on the other hand, simply imitates the behavior of “add_col.R” but as a transpose matrix in order to add the rows.

Once the data was completed in order to run the algorithm and get at least a similar output as the original version, the CEC benchmarking tool was intended to be used for testing. Thanks to the CEC organisation, these tools were provided in C++, Java and Matlab to check which one could be the best implementation for this case. After some trials with the C++ version, some issues regarding its implementation and missing documentation lead the development team to re-write the complete benchmarking tool in the R programming language, in order to avoid mixing code and be sure to get the results expected since the beginning of this project. Also, it was thought in a way to supply the research community with a new implementation of this tool for future projects.

After solving the previous issues and starting to run the whole algorithm, a memory problem came up. Mainly because of the continuous problem between R and Matlab, this was caused, because one matrix, originally containing only integer values, was added a float value in a certain point within the process. R automatically cloned the integer matrix, creating it with a float type in order to continue without errors for every iteration. In the end having an unnecessary number of matrices allocated in memory, making impossible the computation of the algorithm with a size of 7.9 GB during the iteration 700 out of 50,000 (depending on the function to be evaluated in CEC benchmark suite, the total number of iterations would vary). Even if this issue was solved only by changing the original matrix to contain float values since the beginning, it caused several problems and time consumption during the testing phase.

Finally, after completing and fixing the R implementation, and trying to test it with the real number of iterations for each of the twenty CEC benchmarking functions, a bigger computation power was needed. Having only students’ computers for development, it was necessary a bigger source of computation power and with the help of the R library “BatchJobs” together with the Palma Cluster, property of the Westfälische Wilhelms-Universität located in Münster, Germany, it was possible to run the algorithm in different batch jobs within the cluster and just printing output files in order to analyse the data in the end.

5 Benchmark and Comparison

To compare nmmso.R with the original NMMSO the CEC test cases were used to run the same benchmarks as in the original submission (Fieldsend 2014). There 4 different Ratios were used to measure the performance of certain algorithms. Three of those measures (Peak Ratio, Success Ratio and Convergence Speed) have been introduced in (Epitropakis et al. 2013, pp. 6–7) to create a common point of comparison. The fourth ratio is special for the nmmso algorithm since it tracks the number of swarms over the iterations of the algorithm. Nmmso.R uses the same measures to reach the highest comparability possible.

The first measure used is the Success Ratio (SR). The Success Ratio is defined as the percentage of successful runs (runs that found all global optima) over all runs (Li et al. 2013, p. 7). As for the other ratios this measure was taken over several independent runs and collectively evaluated. The taken measures for the Success Ratio can be found in Table 2.

$$\frac{successful\ runs}{NR} = SR$$

Here NR denotes the Number of runs done to reach this measure.

Table 2: Success Ratio over given runs (Measure of share of runs which found all global optima)

	0.1	0.01	0.001	0.0001	0.00001	runs
F1	1	1	1	1	1	45
F2	1	1	1	1	1	42
F3	1	1	1	1	1	44
F4	1	1	1	1	1	44
F5	1	1	1	1	1	41
F6	1	1	1	1	0	40
F7	1	1	1	1	1	41
F8	1	1	1	0.78	0.61	23
F9	0.96	0.96	0.96	0.93	0.93	27
F10	1	1	1	1	1	41
F11	1	1	1	1	1	40
F12	1	1	1	1	1	41
F13	1	1	1	1	1	41
F14	1	1	1	1	1	39
F15	0.97	0.97	0.97	0.97	0.97	31
F16	0	0	0	0	0	20
F17	0.21	0.11	0.11	0.11	0.11	19
F18	0.39	0.39	0.35	0.3	0.3	23
F19	0	0	0	0	0	18
F20	0	0	0	0	0	17

The second measure introduced by the CEC committee and also used by Dr. Fieldsend is the Convergence Rate. The Convergence Rate (CR) measures the needed evaluations per Accuracy and Function to find all global optima (Li et al. 2013, p. 7). This measure takes the mean of evaluations over all runs. The results of this measure can be found in Table 3.

$$\frac{\sum_{n=1}^{NR} evals_n}{NR} = CR$$

In this measure, $evals$ denotes the number of evaluations done.

Table 3: Convergence Rates over given runs (Mean of evaluations needed to find all global optima, if all optima have never been found the maximum allowed evaluations for that function were taken.)

	0.1	0.01	0.001	0.0001	0.00001	runs
F1	610	814	1010	1204	1434	45
F2	181	265	390	560	677	42
F3	33	164	269	382	504	44
F4	501	728	951	1202	1458	44
F5	80	194	312	514	748	41
F6	19124	24077	29970	41858	200001	40
F7	8453	9073	10539	12097	14434	41
F8	199028	239062	283957	331813	358588	23
F9	190205	197213	213486	227226	235417	27
F10	871	1309	1716	2252	2749	41
F11	3743	5652	7159	8309	9173	40
F12	17240	25441	36657	43765	50048	41
F13	10221	14763	18100	21782	26547	41
F14	28185	34579	47325	57497	65873	39
F15	106836	125918	144279	165564	180095	31
F16	400001	400001	400001	400001	400001	20
F17	352657	368393	372283	373597	374462	19
F18	299275	302915	313231	318487	320204	23
F19	400001	400001	400001	400001	400001	18
F20	400001	400001	400001	400001	400001	17

The third measure is the Peak Ratio (PR). It measures the share of found global optima over all runs (Li et al. 2013, p. 7). The results of this evaluation can be found in Table 4.

$$\frac{\sum_{n=1}^{NR} NOF_n}{NKO * NR} = PR$$

In this measure NOF denotes the number of found optima per run and NKO the number of known optima for the function.

Table 4: Peak Ratio over given runs (Share of found global optima over all runs)

	0.1	0.01	0.001	0.0001	0.00001	runs
F1	1	1	1	1	1	45
F2	1	1	1	1	1	42
F3	1	1	1	1	1	44
F4	1	1	1	1	1	44
F5	1	1	1	1	1	41
F6	1	1	1	1	0	40
F7	1	1	1	1	1	41
F8	1	1	1	0.99	0.97	23
F9	1	1	1	1	1	27
F10	1	1	1	1	1	41
F11	1	1	1	1	1	40
F12	1	1	1	1	1	41
F13	1	1	1	1	1	41
F14	1	1	1	1	1	39
F15	1	1	1	1	1	31
F16	0.01	0	0	0	0	20
F17	0.78	0.75	0.73	0.72	0.69	19
F18	0.84	0.83	0.83	0.79	0.79	23
F19	0.34	0.33	0.32	0.32	0.31	18
F20	0.15	0.15	0.15	0.15	0.13	17

As a fourth measure, which wasn't introduced by the CEC committee, but used in the original nmmso implementation (Fieldsend 2014), the Number of Swarms was chosen. Since this is a continuous measure and, therefore no calculation is needed, this measure is pictured as graphs. The graphs can be found in Figure 1. They show the development of *number of swarms* kept by nmmso.R over all iterations. Important to notice here is that *iterations* is different from the *evaluations* referenced in the other measures. Iterations are calls to start single runs of nmmso.R and, therefore, are different from the evaluations taken within the program.

Additionally, a fifth measure was introduced which denotes the runtime of nmmso.R for the single functions. These times were taken on the ZIVHPC, a scientific High Perfomance Computing Cluster by Westfälische Wilhelms-Universität Münster. Since the nmmso.R is a strictly sequential algorithm the runtimes for single runs will be comparable on common computers. The ZIVHPC was only used to parallelise the single runs.

When comparing those measures with the ones given in the original paper (Fieldsend 2014), it can be seen that the reimplementation nmmso.R is an overall good resemblance of the original algorithm. The three CEC measures are close to the original taken measures and the trend curves for the number of kept swarms have similar trends.

The biggest differences between the benchmarking results of the two implementations can be seen in the general results of function 14, 15, 16 and 18, as well as in the number of created swarms for the n-dimensional functions:

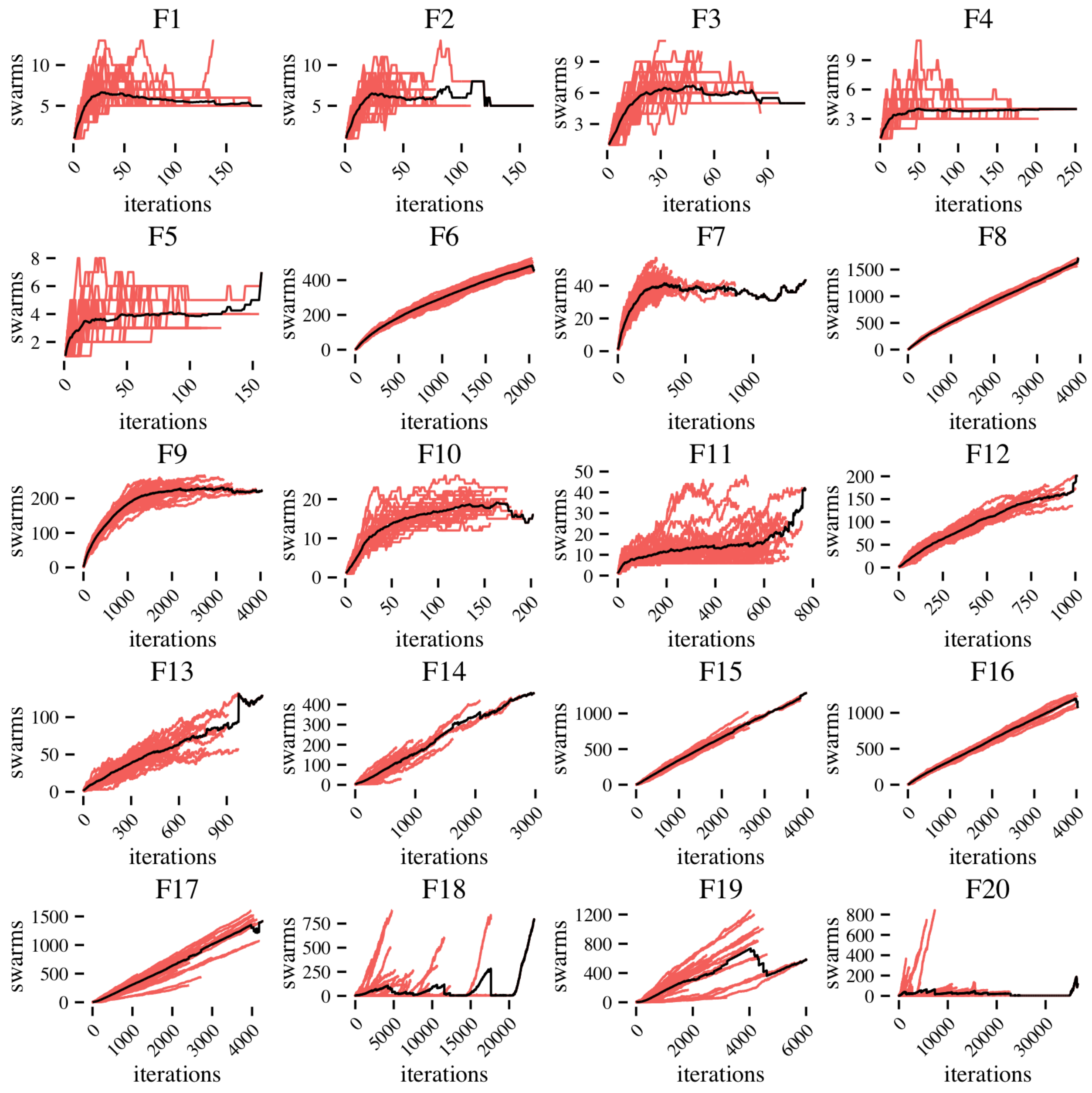

Figure 1: plot of chunk trend curve of kept swarms over all 20 functions. The red curves show the number of swarms kept for each single run. The black line shows the mean of kept swarms over these runs.

(1) Function 14 and 15 have a *Success Ratio* of 1 aswell as *Peak Ratio* of one 1 for all accuracy levels. Additionally nmmso.R sometimes found all global optima for Function 18. In contradiction of all three function almost never result in the finding of global optima in the evaluation of the original implementation. Only at the lowest accuracy, the original implementation is able to find all global optima for Function 14 (Fieldsend 2014, p. 16). It is hard to say if this difference is equal to an error in the implementation of nmmso.R or if an error in the original implementation was fixed. Also, this could be a difference in the reimplementation of the CEC Benchmarking Tool. Nevertheless, this is an interesting point of discussion and worth evaluating.

(2) nmmso.R performs noticeable worse for Function 16 than for the original function. While nmmso.R has a *Peak Ratio* of 0.01 for an accuracy of 0.1 and 0 for all others, the original implementation reaches a *Peak Ratio* of around 0.6 for all accuracies. This might be an implementation error in the CEC Benchmarking Tool and not in nmmso.R since it is so significantly worse that it is unlikely that this difference would only occur in one test function.

(3) Almost all algorithm runs on high-dimensional functions (F12-F20) result in a high number of swarms while all other results regarding this functions are comparable to the original results. This difference becomes very clear in the case of Functions 17-20. In the paper addressing the original implementation the x-axis rank from 0-40,000 iterations while for the reimplementation limit of 4,000 for Function 17, of 20,000 for Function 18, 6,000 for Function 19 and of 30,000 for Function 20 is enough to show all data sets. This is connected to the creation of much more swarms, which leads to an earlier depletion of the maximum allowed number of evaluations.

Additionally, the time of all algorithms was taken. Even though this measure widely varies depending on the computer's architecture it can show the different complexity of all 20 functions.

Table 5: Taken time of nmmso.R for all 20 functions. All times are in seconds.

	Mean	Standard Deviation
F1	13	3.8
F2	8.6	6.5
F3	4.8	2.6
F4	27	6.1
F5	10	4.4
F6	7341	807
F7	740	350
F8	32089	5379
F9	23513	4169
F10	123	26
F11	660	439
F12	2182	517
F13	1746	1359
F14	4519	1940

	Mean	Standard Deviation
F15	12282	7859
F16	40958	1837
F17	33240	12200
F18	61631	27491
F19	63224	15799
F20	97631	14844

6 Conclusion

In this project a reimplementation of the NMMSO algorithm in R was shown. As part of this project also a reimplementation of the CEC Benchmarking Suite was created. It was shown that it is possible to translate a Matlab program into R

7 Acknowledgements

We want to thank Dr. Jonathan Fieldsend for his continuous help via mail during this seminar. Also, the committee of the CEC was always available for questions and concerns during our work. Furthermore, a special thanks go to all employees of the chair for 'Information Systems and Statistics' including Dr. Mike Preuß, Jakob Bossek and Pascal Kerschke who were available for any questions regarding the implementation and this report at all times.

Epitropakis, M. G., Li, X., and Burke, E. K. 2013. “A dynamic archive niching differential evolution algorithm for multimodal optimization,” in *Evolutionary computation (cEC), 2013 iEEE congress on*, IEEE, pp. 79–86.

Fieldsend, J. E. 2014. “Running up those hills: Multi-modal search with the niching migratory multi-swarm optimiser,” in *Evolutionary computation (cEC), 2014 iEEE congress on*, IEEE, pp. 2593–2600.

Li, X., Engelbrecht, A., and Epitropakis, M. G. 2013. “Benchmark functions for cEC’2013 special session and competition on niching methods for multimodal function optimization,” *RMIT University, Evolutionary Computation and Machine Learning Group, Australia, Tech. Rep.*

Preuss, M. 2012. “Improved topological niching for real-valued global optimization,” in *Applications of evolutionary computation*, Springer, pp. 386–395.