Acta Cybernetica **22** (2016) 715–733.

Nonlinear Symbolic Transformations for Simplifying Optimization Problems[*]

Elvira Antal[†‡] and Tibor Csendes[†]

Abstract

The theory of nonlinear optimization traditionally studies numeric computations. However, increasing attention is being paid to involve computer algebra into mathematical programming. One can identify two possibilities of applying symbolic techniques in this field. Computer algebra can help the modeling phase by producing alternate mathematical models via symbolic transformations. The present paper concentrates on this direction. On the other hand, modern nonlinear solvers use more and more information about the structure of the problem through the optimization process leading to hybrid symbolic-numeric nonlinear solvers.

This paper presents a new implementation of a symbolic simplification algorithm for unconstrained nonlinear optimization problems. The program can automatically recognize helpful transformations of the mathematical model and detect implicit redundancy in the objective function.

We report computational results obtained for standard global optimization test problems and for other artificially constructed instances. Our results show that a heuristic (multistart) numerical solver takes advantage of the automatically produced transformations.

New theoretical results will also be presented, which help the underlying method to achieve more complicated transformations.

Keywords: nonlinear optimization, reformulation, Mathematica

1 Introduction

Application of symbolic techniques to rewrite or solve optimization problems are a promising and emerging field of mathematical programming. Symbolic preprocessing of linear programming problems [11] is the classic example, this kind of transformation was implemented in the AMPL processor about twenty years ago

[*]This work was partially supported by the Grant TÁMOP-4.2.2.A-11/1/KONV-2012-0073.

[†]University of Szeged, Institute of Informatics, H-6720 Szeged, Árpád tér 2, Hungary, E-mail: `antale@inf.u-szeged.hu, csendes@inf.u-szeged.hu`

[‡]Kecskemét College, Faculty of Mechanical Engineering and Automation, Hungary

DOI: 10.14232/actacyb.22.4.2016.1

as an automatic “presolving” mechanism [7, 8]. A more recent example is the assistance of (mixed-) integer nonlinear programming solvers, as in the Reformulation-Optimization Software Engine of Liberti et al. [10]. In this field, the relaxation of some constraints or increasing the dimension of the problem could be reasonable to achieve feasibility.

However, as the work of Csendes and Rapcsák [5, 14] shows, it is also possible to produce equivalent models of an unconstrained nonlinear optimization problem via symbolic transformations automatically, while bijective transformations between the optima of the models are constructed. This method is capable of eliminating redundant variables or simplifying the problem in other ways.

The present paper is organized as follows. Section 2 summarizes briefly the results of Csendes and Rapcsák [5, 14] and presents a new, proper Mathematica implementation of their method together with a comparison with the earlier reported Maple prototype [2]. Section 3 presents computational results to show that the automatically produced transformations can help a traditional heuristic numeric solver, namely Global [4], to reduce computation times and function evaluations. Section 4 extends the theory of the mentioned method in two directions: describes parallel substitutions and introduces constraints into the model.

2 The Simplifier Method

2.1 Theoretical Background

We concentrate on unconstrained nonlinear optimization problems of the form

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \quad f(\boldsymbol{x}), \tag{1}$$

where $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is a smooth function given by a formula, i.e. a symbolic expression. “Expression” denotes a well-formed, finite combination of symbols (constants, variables, operators, function names and brackets), usually realized in computer algebra systems with a list (for example, a nested list of pointers in Mathematica [20]), or a directed acyclic graph [15]. In the subsequent description, vectors are denoted by boldface letters, sets by capital letters, and functions by small letters. The meaning of z_i depends on context: it denotes the i^{th} element of a vector $\boldsymbol{z}$ or an ordered set Z. We will use the function notation $v(\boldsymbol{z})$ to represent a v expression, which can contain any variable z_i, any real constant, and any function name.

The simplifier method aims to recognize, whether (1) could be transformed into an equivalent formulation, which is better in the following senses: the new formulation has fewer arithmetic operations to execute during evaluation, the dimension of the problem is less, or it is simpler to solve for another reason. Equivalent means here, that a bijective transformation can be given between the optima of the original and those of the transformed problem.

Csendes and Rapcsák [5] showed that an objective function $g(\boldsymbol{y})$ is equivalent to $f(\boldsymbol{x})$ in (1), if we get $g(\boldsymbol{y})$ by the following transformation:

- apply a substitution in $f(\boldsymbol{x})$:

$$y_i := h(\boldsymbol{x}) \,, \quad 1 \le i \le n,$$

  where $h(\boldsymbol{x})$ is a smooth function with a range $\mathbb{R}$, and $h(\boldsymbol{x})$ is strictly monotonic as a function of at least one variable x_i,

- rename the remaining variables:

$$y_j := x_j \,, \quad j = 1, \dots, i-1, i+1, \dots, n,$$

  and

- omit the variables y_i without presence in the evolving objective function.

The term *appropriate substitution* will refer to an $y_i = h(\boldsymbol{x})$ substitution, where

- $h(\boldsymbol{x})$ satisfies the criteria being smooth, monotonic in at least one variable x_i, and its range is equal to $\mathbb{R}$,
- $h(\boldsymbol{x})$ covers (characterizes all occurrences of) at least one variable x_i, that is, x_i could be removed totally from the optimization problem by substituting $h(\boldsymbol{x})$ by y_i, and
- $y_i = h(\boldsymbol{x})$ is not a simple renaming, that is, $h(\boldsymbol{x}) \neq x_i$, $i = 1, \dots, n$.

After applying a transformation with an appropriate substitution $y_i = h(\boldsymbol{x})$, $\boldsymbol{y}$ has at most the same dimension as $\boldsymbol{x}$. Redundant variables can be eliminated, if $h(\boldsymbol{x})$ covers two or more variables. In other words, we have the possibility to recognize whether the model can be formalized with a smaller set of variables. However, these are sufficient, but not necessary conditions for simplifier transformations.

For example, consider $f(x_1, x_2) = (x_1 + x_2)^2$. It is equivalent to minimize $g(y_1) = y_1^2$, and the optimal values of the original variables x_1 and x_2 can be set by the symbolic equation $y_1 = x_1 + x_2$, which is an appropriate substitution. In fact, we can handle an infinite number of global optimum points in this way, which is impossible for any numerical solver.

One of the main goals of the simplifier method is to find appropriate substitutions which would eliminate variables. Csendes and Rapcsák [5] with their Assertion 2 suggest to compute the partial derivatives $\partial f(\boldsymbol{x})/\partial x_i$, factorize them, and search for appropriate substitutions in the factors.

Assertion 2 [5]. *If the variables x_i and x_j appear everywhere in the expression of a smooth function $f(\boldsymbol{x})$ in a term $h(\boldsymbol{x})$, then the partial derivatives $\partial f(\boldsymbol{x})/\partial x_i$ and $\partial f(\boldsymbol{x})/\partial x_j$ can be factorized in the forms $(\partial h(\boldsymbol{x})/\partial x_i)\, p(\boldsymbol{x})$ and $(\partial h(\boldsymbol{x})/\partial x_j)\, q(\boldsymbol{x})$, respectively, and $p(\boldsymbol{x}) = q(\boldsymbol{x})$.*

If $\partial f(\boldsymbol{x})/\partial x_i$ cannot be factorized, then any appropriate substitution that is monotonic as a function of x_i is linear as a function of x_i.

For illustrative purposes, let us consider the following problem:

$$\min_{\boldsymbol{x}\in\mathbb{R}^3} \ 30\cdot(5x_1+e^{1+x_2})+20\cdot x_3,$$
$$s.t.\ \ln(5x_1+e^{1+x_2})+x_3\geq 5.$$

This example can be a tiny part of a process synthesis problem. Automatic model generator tools in the field of process synthesis produce several types of multiplicity and redundancy [6]. Among other redundancies, it is possible for some variables denoting chemical elements to appear exclusively in the chemical formula of a given material. In our example, x_1 and x_2 appear everywhere in the term $h(\boldsymbol{x})=5x_1+e^{1+x_2}$. For the sake of simplicity, the constraint can be reformulated by adding a penalty term [9] to the objective, so we need to minimize

$$f(\boldsymbol{x})=30\cdot(5x_1+e^{1+x_2})+20\cdot x_3+\sigma\left(\ln(5x_1+e^{1+x_2})+x_3-5-x_4\right)^2,$$

where x_1,x_2,x_3 are real variables based on physical parameters, σ is a penalty constant and x_4 is a slack variable. Due to Assertion 2, $\partial f(\boldsymbol{x})/\partial x_1$ can be transformed into the form $(\partial h(\boldsymbol{x})/\partial x_1)\cdot p(\boldsymbol{x})$, similarly $\partial f(\boldsymbol{x})/\partial x_2=(\partial h(\boldsymbol{x})/\partial x_2)\cdot q(\boldsymbol{x})$, while $p(\boldsymbol{x})=q(\boldsymbol{x})$. In our example $\partial h(\boldsymbol{x})/\partial x_1=5$, $\partial h(\boldsymbol{x})/\partial x_2=e^{1+x_2}$, and

$$p(\boldsymbol{x})=q(\boldsymbol{x})=\frac{2\left(75x_1+15e^{1+x_2}+\sigma\left(\ln(5x_1+e^{1+x_2})+x_3-5-x_4\right)\right)}{5x_1+e^{1+x_2}}.$$

Based on the above result, we created a computer program to produce equivalent transformations automatically for the simplification of unconstrained nonlinear optimization problems. The naive implementation would realize the following steps:

1. Compute the gradient of the objective function.
2. Factorize the partial derivatives.
3. Collect appropriate substitutions which contain x_i, into a list l_i:
   a) Initialize l_i to be the empty set.
   b) If the factorization was successful for $\partial f(\boldsymbol{x})/\partial x_i$, then extend l_i with the respective integrals of the factors.
   c) Extend l_i with the subexpressions of $f(\boldsymbol{x})$ that are linear in x_i.
   d) Drop the elements of l_i which do not fulfill the conditions of an appropriate substitution (the elements of l_i need to be monotonic in x_i).
4. Create a list S by applying all proper combinations of the appropriate substitutions from $L=\bigcup l_i,\ i=1,\dots,n$ to $f(\boldsymbol{x})$.
5. Choose the least complex element of S to be the simplified objective function.
6. Solve the problem with the simplified objective function (if possible).

7. Give the solution of the original problem by executing inverse transformations.

Most of the required steps of the algorithm (partial differentiation, factorization, symbolic integration and substitution) are realized in modern computer algebra systems as reliable symbolic computation methods. On the other hand, our first implementation in Maple showed that even one of the market-leader computer algebra systems has serious deficiency to our requirements in point of substitution capability and interval arithmetic with infinite intervals [2].

Actually, the exact range calculation for a nonlinear function has the same complexity as computing the global minimum and maximum. However, naive interval inclusion can be applied to verify whether the range of a subexpression is equal to $\mathbb{R}$. Naive interval inclusion is exact for a single use expression (SUE, an expression that contains any variable at most once), but it might produce overestimation for more complex expressions [12]. The possible overestimation can lead to false-positive answers in the range calculation. In other words, L can contain some substitutions with a range which is not equal to $\mathbb{R}$. It means that an additional verification for the range of the produced non-SUE substitutions would be required. On the other hand, most of the substitutions produced in our tests were SUEs. As an alternative, real quantifier elimination [17, 19] would be an applicable symbolic technique for range calculation.

Naive interval inclusion can also be used for the monotonicity test. A real function $f : \mathbb{R}^n \to \mathbb{R}$ is monotone if any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ such that $\boldsymbol{x} \leq \boldsymbol{y}$ satisfy $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$. Let us use the product order here: $(x_1, \ldots, x_n) \leq (y_1, \ldots, y_n)$ if $x_i \leq y_i$, $i = 1, \ldots, n$. In the discussed application we need to test whether a function $h_i(\boldsymbol{x})$ is strictly monotonic as a function of a variable x_i. Therefore we compute, whether the naive interval inclusion of the partial derivative $\partial h_i(\boldsymbol{x})/\partial x_i$ contains zero. This approach fits the mathematical definition of monotonicity and is expressive, as a strictly monotone function has a single region of attraction. Unfortunately, overestimation of the naive interval inclusion for non-SUEs can produce false-negative answers in the monotonicity test, so even some monotonic substitutions can be dropped.

Step 3 and Step 4 can be combined to speed up the process and also to ensure that a proper substitution set is applied to $f(\boldsymbol{x})$. We call a well ordered set H of appropriate substitutions *proper* if all the formulas $h_i(\boldsymbol{x}) \in H$ can be substituted by new variables at the same time in a function $f(\boldsymbol{x})$. That is, the expressions $\forall h_i(\boldsymbol{x}) \in H$ do not overlap in the computation tree of $f(\boldsymbol{x})$. Without this property, not all substitutions $y_i = h_i(\boldsymbol{x})$, $h_i(\boldsymbol{x}) \in H$ could be applied. For example, in $f(\boldsymbol{x}) = (x_1 + x_2 + x_3)^2$, the substitutions $y_1 = x_1 + x_2$ and $y_2 = x_2 + x_3$ would be also appropriate, but $H = \{x_1 + x_2, x_2 + x_3\}$ is not a proper substitution set because $x_1 + x_2$ and $x_2 + x_3$ both refer to the same occurrence of x_2. In fact, we prefer to choose the most complex $h(\boldsymbol{x})$ formula to eliminate a variable, so in this example, the substitution $y_3 = x_1 + x_2 + x_3$ should be accepted. At this point, and in Step 5, an easily applicable complexity definition for expressions is needed. In our implementation, an expression is said to be more complex than an other one if its representation (a list of pointers in Mathematica) is longer.

2.2 Implementation in Mathematica

Compared to our first implementation in the computer algebra system Maple [2], the new platform Mathematica has several advantages. First of all, the substitution procedures are much better, since the Mathematica programming language is based on term-rewriting. In other words, the capabilities of the basic substitution routine of Mathematica can be extended with regular expression based term-rewriting rules. We have written a specialized substitution routine in about 50 program lines. A dozen (delayed) rules are introduced, and we have combined four different ways for evaluating them, using the expanded, simplified, and also the factorized form of the formula. It is probably the most important part of the program, as simple refinements could have substantial influence on the result quality of the whole simplification process.

Mathematica has also better interval arithmetic implementation: this was crucial for quick and reliable range calculation on the expressions to be substituted. Naive interval inclusion for the enclosure of the ranges have been realized with the standard range arithmetic of Mathematica.

Furthermore, our new program supports the enumeration of all possible substitutions in Step 3, and it still keeps up in running time with the simpler Maple version, which has used simple heuristics to choose one possible substitution. The reasons for that are the application of adequate programming techniques, and some nice properties of the Mathematica system, such as automatic parallelization on list operations. However, a further study on an efficient substitution selection strategy would be welcome.

Let us mention that Mathematica tends to include more and more expert tools to ensure the possibility of writing efficient program code. For example, it has provided utilities to exploit the parallel computing abilities of the graphics processing unit (GPU) using CUDA or OpenCL technology since 2010.

Demonstration of the presented algorithm and our newest program codes will be available at the following homepage:

`http://www.inf.u-szeged.hu/~csendes/symbsimp/`

2.3 Improvements Compared to the Maple Version

Some simple examples follow to demonstrate the advantages of our new Mathematica program compared to the Maple version.

These examples were generated by us and were the problematic ones out of our self-made collection in the test phase of the Maple implementation, so it was obvious to use them to test the new program. See Table 1 for the Maple-based results and Table 2 for the results obtained by the Mathematica version. For the test cases not discussed, the two implementations gave the same output.

Remark, that the renaming convention of Csendes and Rapcsák [5] is slightly modified in the following tables. Simple renaming ($y_j := x_j$) is not applied in the hope that more elaborated substitutions are emphasized in this way.

Table 1: Results of some synthetic tests solved by our old, Maple based implementation

ID	Function f	Function g	Substitutions	Problem type	Result type
Sin2	$2x_3 \cdot \sin(2x_1 + x_2)$	$2y_1$	$y_1 = x_3 \cdot \sin(2x_1 + x_2)$	A	3
Exp1	$e^{x_1+x_2}$	e^{y_1}	$y_1 = x_1 + x_2$	A	1
Exp2	$2e^{x_1+x_2}$	$2y_1$	$y_1 = e^{x_1+x_2}$	A	3
Sq1	$x_1^2 x_2^2$	none	none	D	2
Sq2	$(x_1x_2 + x_3)^2$	y_1^2	$y_1 = x_1x_2+x_3$	A	1
SqCos1	$(x_1x_2 + x_3)^2 - \cos(x_1x_2)$	$y_3^2 - \cos(y_1)$	$y_1 = x_1x_2$, $y_3 = y_1 + x_3$	A	1,3
SqExp2	$(x_1 + x_2)^2 +2e^1e^{x_1+x_2}$	$y_1^2 + 2e^1e^{y_1}$	$y_1 = x_1 + x_2$	A	1
SqExp3	$(x_1 + x_2)^2 +2e^{1+x_1+x_2}$	none	none	A	2

Table 2: Results of some synthetic tests solved by our new, Mathematica based implementation

ID	Function f	Function g	Substitutions	Problem type	Result type
Sin2	$2x_3 \cdot \sin(2x_1 + x_2)$	$2x_3 \sin(y_1)$	$y_1 = 2x_1 + x_2$	A	1
Exp1	$e^{x_1+x_2}$	e^{y_1}	$y_1 = x_1 + x_2$	A	1
Exp2	$2e^{x_1+x_2}$	$2e^{y_1}$	$y_1 = x_1 + x_2$	A	1
Sq1	$x_1^2 x_2^2$	none	none	D	2
Sq2	$(x_1x_2 + x_3)^2$	y_1^2	$y_1 = x_1x_2+x_3$	A	1
SqCos1	$(x_1x_2 + x_3)^2 - \cos(x_1x_2)$	$y_1^2 - \cos(x_1x_2)$	$y_1 = x_1x_2+x_3$	A	1
SqExp2	$(x_1 + x_2)^2 +2e^1e^{x_1+x_2}$	$y_1^2 + 2e^{1+y_1}$	$y_1 = x_1 + x_2$	A	1
SqExp3	$(x_1 + x_2)^2 +2e^{1+x_1+x_2}$	$y_1^2 + 2e^{1+y_1}$	$y_1 = x_1 + x_2$	A	1

We apply again the evaluation codes from [2], which describe the quality of the results and the nature of the problem. These codes appear in the "Problem type" and "Result type" columns of Tables 1 and 2. The letters characterize the actual problem:

(A). Simplifying transformations are possible according to the presented theory.

(B). Simplifying transformations could be possible by extension of the presented theory.

(C). Some helpful transformations could be possible by extension of the presented theory, but they do not necessarily simplify the problem (e.g., since they increase dimensionality).

(D). We do not expect any helpful transformation.

The results are described by the second indicator: our program produced

1. proper substitutions,
2. no substitutions, or
3. incorrect substitutions.

As we had discussed earlier [2], Maple provides incorrect interval arithmetic, so we needed to use heuristics in the Maple implementation for range calculation. Also the algebraic substitution capabilities of Maple are weak. Almost all mistakes of the early implementation originated from these two reasons.

Code "2" is also used when only constant multipliers are eliminated.

In short, A1 means that proper substitution is possible and it has been found by the algorithm, while D2 means that as far as we know, no substitution is possible, and this was the conclusion of the program as well. The unsuccessful cases are those denoted by other codes.

The differences between the obtained results are explained next. For the *Sin2* test problem, a proper simplification was obtained by the new implementation, while the old one conveyed a more complex, but non-monotonic substitution.

In *Exp2*, $e^{x_1+x_2}$ does not have a range equal to $\mathbb{R}$, but the heuristic range calculation method used in Maple recognized it as an appropriate substitution. The range calculation subroutine in Mathematica proved to be better in this case.

In *Sq1*, the Maple implementation was not able to recognize the subexpression x_1x_2 in the expression $x_1^2x_2^2$, but was able to recognize $x_1x_2 + x_3$ in its square in *Sq2*. Since the second is not a multiplication type expression but a sum, it is represented in a different way. In Mathematica, regular expressions can be used to produce good substitutions, and our specialized substitution routine worked well for this problem. On the other hand, x_1x_2 is not monotone as a function of x_1 or x_2 for the whole search interval (supposed to be $\mathbb{R}$), so it cannot be chosen as an appropriate substitution.

Also for the *SqCos1* test problem, the new, Mathematica-based method applied a routine to check whether the substitution expression is monotone, so $y_1 = x_1x_2$ was eliminated from the substitution list.

In the cases *SqExp2-3*, also the weakness of the expression matching capability of Maple can be observed, as it was not able to recognize $x_1 + x_2$ in $e^{1+x_1+x_2}$, only in $e^1 e^{x_1+x_2}$.

2.4 Computational Test Results on Global Optimization Problems

Standard and frequently used global optimization test problems were used to study the capabilities and limitations of the symbolic simplification algorithm. The test set was extended in comparison to our earlier paper [2]. The description and even various implementations of the examined problems can be found in several resources and online collections. For example, the compact mathematical formulation and known optima of all of the mentioned problems can be found in Appendix A at Pal [13]. Our computational results are summarized in Table 3.

ID	Dim.	New variables	Substitutions	Problem type	Result type	Transform. time
BR	2	$\boldsymbol{y} = [x_1, y_1]$	$y_1 = x_2$ $-6 + (5/\pi)x_1$ $-0.129185x_1^2$	A	1	0.1092
Easom	2	$\boldsymbol{y} = \boldsymbol{x}$	none	D	2	0.1404
G5	5	$\boldsymbol{y} = \boldsymbol{x}$	none	D	2	2.7456
G7	7	$\boldsymbol{y} = \boldsymbol{x}$	none	D	2	36.5821
GP	2	$\boldsymbol{y} = \boldsymbol{x}$	none	D	2	0.4212
H3	3	$\boldsymbol{y} = \boldsymbol{x}$	none	D	2	16.3488
H6	6	Stopped.	Stopped.	D	2	1800 <
L1	1	$\boldsymbol{y} = \boldsymbol{x}$	none	D	2	0.0312
L2	1	$\boldsymbol{y} = \boldsymbol{x}$	none	D	2	0.6552
L3	2	$\boldsymbol{y} = \boldsymbol{x}$	none	D	2	2.3088
L5	2	$\boldsymbol{y} = \boldsymbol{x}$	none	D	2	15.3348
L8	3	$\boldsymbol{y} = [y_1, y_2, y_3]$	$y_1 = (x_1 - 1)/4$, $y_2 = (x_2 - 1)/4$, $y_3 = (x_3 - 1)/4$	A	1	13.1820
L9	4	$\boldsymbol{y} = [y_1, y_2, y_3, y_4]$	$y_1 = (x_1 - 1)/4$, $y_2 = (x_2 - 1)/4$, $y_3 = (x_3 - 1)/4$, $y_4 = (x_4 - 1)/4$	A	1	174.7047
L10	5	Stopped.	Stopped.	A	1	1800 <
L11	8	Stopped.	Stopped.	A	1	1800 <
L12	10	Stopped.	Stopped.	A	2	1800 <
L13	2	$\boldsymbol{y} = \boldsymbol{x}$	none	D	2	0.4992
L14	3	$\boldsymbol{y} = \boldsymbol{x}$	none	D	2	0.7800
L15	4	$\boldsymbol{y} = \boldsymbol{x}$	none	D	2	1.0296
L16	5	$\boldsymbol{y} = \boldsymbol{x}$	none	D	2	2.0904
L18	7	$\boldsymbol{y} = \boldsymbol{x}$	none	D	2	32.1517
Sch21 (Beale)	2	$\boldsymbol{y} = \boldsymbol{x}$	none	C	2	0.1248
Sch214						

(Powell)	4	$\boldsymbol{y}=\boldsymbol{x}$	none	D	2	0.0936
Sch218 (Matyas)	2	$\boldsymbol{y}=\boldsymbol{x}$	none	D	2	0.0156
Sch227	2	$\boldsymbol{y}=\boldsymbol{x}$	none	D	2	0.0624
Sch25 (Booth)	2	$\boldsymbol{y}=\boldsymbol{x}$	none	C	2	0.0312
Sch31	3	$\boldsymbol{y}=\boldsymbol{x}$	none	D	2	0.0468
Sch31	5	$\boldsymbol{y}=\boldsymbol{x}$	none	D	2	0.0936
Sch32	2	$\boldsymbol{y}=[y_1,y_2]$	$y_1=x_1-x_2^2$, $y_2=x_2-1$	A	1	0.0468
Sch32	3	$\boldsymbol{y}=\boldsymbol{x}$	none	D	2	0.0312
Sch37	5	$\boldsymbol{y}=\boldsymbol{x}$	none	D	2	0.0936
Sch37	10	$\boldsymbol{y}=\boldsymbol{x}$	none	D	2	0.2808
SHCB	2	$\boldsymbol{y}=\boldsymbol{x}$	none	D	2	0.0156
THCB	2	$\boldsymbol{y}=\boldsymbol{x}$	none	D	2	0.0156
Rastrigin	2	$\boldsymbol{y}=\boldsymbol{x}$	none	C	2	0.0936
RB	2	$\boldsymbol{y}=[y_1,y_2]$	$y_1=x_1^2-x_2$, $y_2=1-x_1$	A	1	0.0440
RB5	5	$\boldsymbol{y}=[x_1,x_2,x_3,x_4,y_1]$	$y_1=x_4^2-x_5$	A	1	0.3080
S5	4	$\boldsymbol{y}=\boldsymbol{x}$	none	D	2	225.5018
S7	4	$\boldsymbol{y}=\boldsymbol{x}$	none	D	2	1,010.2431
S10	4	Stopped.	Stopped.	D	2	1800 <
R4	2	$\boldsymbol{y}=\boldsymbol{x}$	none	C	2	0.2964
R5	3	$\boldsymbol{y}=[x_1,x_2,y_2]$	$y_1=3+x_3$, $y_2=(\pi/4)y_1$	A	1	13.8216
R6	5	$\boldsymbol{y}=[x_1,x_2,x_3,x_4,y_2]$	$y_1=3+x_5$, $y_2=(\pi/4)y_1$	A	2	995.6883
R7	7	Stopped.	Stopped.	A	2	1800 <
R8	9	Stopped.	Stopped.	A	2	1800 <

Table 3: Results for the standard global optimization test functions

In the common cases, most of our new results are identical to what we have obtained with the earlier, Maple-based implementation. The two differences are reported here. For the *Schwefel-227* test problem, the Maple version gave the substitution $y_1 = x_1^2 + x_2^2 - 2x_1$. This expression characterizes all occurrences of x_2, but it is not monotonic in any variable, so the Mathematica version had not suggested it for substitution. For *Schwefel-32 (n=2)*, Mathematica found a good substitution, while Maple did not.

All transformations were performed with Mathematica 9.0 under time constraints. In those cases, in which the complete simplifier program had not stopped in 1800 seconds, the message "Stopped." was written to the New variables and Substitutions columns and "1800 <" to the Transformation time column of Table 3. The numerical tests ran on a computer with an Intel i5-3470 processor, 8 GB RAM and 64-bit operating system.

Most of the running time was used by the symbolic formula transformations of the extended substitution routine. In the problematic cases, usually symbolic factorization consumed 1800 seconds. While every transformation in Table 2 finished in less than 0.2 seconds, the running time for the standard test cases vary more. 24 of 45 test cases ran in one second, further 10 analysis required less than one minute, but 7 test cases would require more than a half hour to finish.

Altogether, 45 well-known global optimization test problems were examined, and our Mathematica program offered equivalent transformations for 8 cases. In other words, our method suggested some simplification for 18% of this extended standard global optimization test set. The next section presents numerical results to demonstrate that these transformations could be useful for a global optimization solver.

3 On the Advantages of the Transformations

This section presents numerical test results to verify the usefulness of the transformations of Tables 2 and 3. We compare the results of a numerical global optimization solver for the minimization of the original and the transformed problem forms, for every cases of Tables 2 and 3 where our algorithm produced an equivalent transformation. The numerical indicators, as reached global optima values, running times, and function evaluation numbers are presented in Tables 4, 5, and 6. Boldface denotes the better options of related numbers.

Table 4: Optimal function values found by Global

	Original problem			Transformed problem		
ID	Fbest	Fmean	Fvar	Fbest	Fmean	Fvar
Exp1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Exp2	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Sq2	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
SqCos1	−1.0000	−0.7671	0.1899	−1.0000	**-0.9922**	**0.0700**
SqExp2	3.0000	3.0000	0.0000	3.0000	3.0000	0.0000
SqExp3	3.0000	3.0000	0.0000	3.0000	3.0000	0.0000
CosExp	−2.0000	**-1.6166**	0.4397	−2.0000	−1.5896	**0.2781**
BR	0.3979	0.3979	0.0000	0.3979	0.3979	0.0000
L8	0.0000	**2.1386**	**5.6861**	0.0000	2.2651	6.8558
L9	0.0000	2.4410	**8.7591**	0.0000	**2.3897**	10.1134
RB	0.0000	19.7318	1094.1167	0.0000	**0.0000**	**0.0000**
RB5	0.0000	1.8510	3.8703	0.0000	**1.8400**	**3.8677**
R5	0.0000	1.9017	26.3097	0.0000	**0.0000**	**0.0000**
R6	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Sch3.2	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Table 5: Number of function evaluations of Global

	Original problem		Transformed problem	
ID	NumEvalmean	NumEvalvar	NumEvalmean	NumEvalvar
Exp1	87	2,280	**51**	**188**
Exp2	102	2,489	**55**	**327**
Sq2	478	57,927	**52**	**38**
SqCos1	1467	463,242	**1,131**	**279,915**
SqExp2	155	4,903	**61**	**142**
SqExp3	151	5,282	**61**	**142**
CosExp	1,110	1,805,418	**631**	**154,691**
BR	136	1,504	**115**	**769**
L8	**785**	266,138	797	**242,593**
L9	2,606	1,838,627	**2,371**	**1,343,151**
RB	749	71,762	**127**	**976**
RB5	3,162	**693,878**	**2,634**	709,652
R5	**1,908**	1,644,365	2,957	**581,382**
R6	**6,001**	**223,269**	6,069	269,551
Sch3.2	119	1,290	**59**	**121**

Table 6: Running times of Global (seconds)

	Original problem		Transformed problem	
ID	Tmean	Tvar	Tmean	Tvar
Exp1	0.0289	0.0004	**0.0113**	**0.0000**
Exp2	0.0346	0.0005	**0.0124**	**0.0000**
Sq2	0.0919	0.0029	**0.0072**	**0.0000**
SqCos1	0.1822	0.0083	**0.1406**	**0.0047**
SqExp2	0.0270	0.0002	**0.0088**	**0.0000**
SqExp3	0.0264	0.0002	**0.0088**	**0.0000**
CosExp	0.2139	0.0429	**0.1623**	**0.0134**
BR	0.0241	0.0001	**0.0195**	**0.0000**
L8	0.0992	0.0046	**0.0990**	**0.0040**
L9	0.2771	0.0220	**0.2445**	**0.0150**
RB	0.0857	0.0009	**0.0241**	**0.0001**
RB5	0.2705	0.0050	**0.2089**	**0.0045**
R5	**0.2407**	0.0286	0.4567	**0.0159**
R6	0.7830	0.0136	**0.7785**	**0.0047**
Sch3.2	0.0187	0.0001	**0.0116**	**0.0000**

We performed 100 independent runs for every test cases, with the Matlab implementation of a general multi-start solver with a quasi-Newton type local search method, Global with the BFGS local search [4]. The tests were run on the same computer, which was described in Subsection 2.4, with 64-bit MATLAB R2011b. The parameters of Global were set to the defaults.

The solver needs an initial search interval for every variable, so we set the $[-100, 100]$ interval as initial box of every variable in our self-made test cases (Table 2), and the usual boxes for the standard problems (see for example Appendix A in [13]). In the transformed cases, the bounds were transformed appropriately.

Table 4 shows the optimal function values reached by Global with the above mentioned parameters. *Fbest* denotes the minimal optimum value, *Fmean* is the mean of the reached optimum values in average of the 100 runs, and *Fvar* denotes the variance of the reached minimum values. The real global optimum values were reached in every independent run for both of the original and the transformed forms in 8 of 15 cases. In the cases *RB* and *R5*, only the transformed form helped the solver to reach the minimum value in all 100 runs. Totally in 5 of 15 cases, the transformed form was easier to solve with Global, the two forms were equivalently difficult to solve in 8 cases, but in 2 cases the original form was slightly more favorable.

Let us compare the number of function evaluations required by the solver in a run (Table 5). *NumEvalmean* refers to the mean of the number of function evaluations, and *NumEvalvar* refers to the variance of the same indicator. The transformed problem needed fewer function evaluations in 12 of 15 cases, and in some cases it outperformed the original form very well. For example, Global needs only 17% of the original function evaluations for finding the optimum of the *Rosenbrock* problem in the transformed form, 80% of the original with the transformed *Branin*, and 11% of the original function evaluations with the transformed *Sq2* problem. The original form of *L8* and *R6* was slightly better for the solver in terms of function evaluations, and at *L8* also in founding minimum values. The original form of *R5* needed fewer function evaluations, but did not enable the solver to find the global optima in every run, while the transformed form did.

Regarding running times (Table 6), the transformed problem form allows Global to run a bit quicker than on the original problem form in almost every case. The average relative improvement in the running times of the whole test set is 31.5%. However, this indicator is better for our self-made problems (56.9%) and worse for the standard problems (9.3%).

We can conclude that the equivalent transformations of Table 2 and Table 3, which seem to be very simple, have a big influence on the performance of a traditional global optimization solver.

4 Theoretical Extension

We generalize the theoretical results of the papers of Csendes and Rapcsák [5, 14] to allow parallel substitutions and to cover constrained nonlinear optimization problems.

Let us start with an example for parallel substitutions. Consider the following objective function:

$$f(x_1, x_2, x_3) = (x_1 + x_2 + x_3)^2 + (x_1 - 2x_2 - 3x_3)^2.$$

It is equivalent to minimize $g(y_1, y_2) = y_1^2 + y_2^2$, which is a two-dimensional problem against the original three-dimensional one. Neither $y_1 = x_1 + x_2 + x_3$ nor $y_2 = x_1 - 2x_2 - 3x_3$ is appropriate in the earlier meaning, as they are smooth and monotonic in every variable, but none of the variables are covered by y_1 or y_2. However, y_1 together with y_2 characterizes all occurrences of x_1, x_2, and x_3, and $H = \{x_1+x_2+x_3, x_1-2x_2-3x_3\}$ is a proper set of substitutions, resulting dimension reduction of the aforementioned problem. The theoretical extension aims to handle this kind of parallel substitutions.

The original nonlinear optimization problem that will be simplified automatically now can include constraints, too:

$$\begin{array}{ll} \min\limits_{x \in \mathbb{R}^n} & f(\boldsymbol{x}) \\ c_i(\boldsymbol{x}) & = 0 \\ c_j(\boldsymbol{x}) & \le 0, \end{array} \tag{2}$$

where $f(\boldsymbol{x}), c_i(\boldsymbol{x}), c_j(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ are smooth real functions, given by a formula, and $i = 1, \ldots, p_1$ and $j = p_1 + 1, \ldots, p_2$ are integer indexes.

The transformed constrained optimization problem will be

$$\begin{array}{ll} \min\limits_{y \in \mathbb{R}^m} & g(\boldsymbol{y}) \\ d_i(\boldsymbol{y}) & = 0 \\ d_j(\boldsymbol{y}) & \le 0, \end{array} \tag{3}$$

where $g(\boldsymbol{y}) : \mathbb{R}^m \to \mathbb{R}$ is the transformed form of $f(\boldsymbol{x})$, and $d_i(\boldsymbol{y}), d_j(\boldsymbol{y}) : \mathbb{R}^m \to \mathbb{R}$ are again smooth real functions, the transformations of the constraints $c_i(\boldsymbol{x}), c_j(\boldsymbol{x})$, $i = 1, \ldots, p_1$ and $j = p_1 + 1, \ldots, p_2$.

Denote by X the set of variable symbols in the objective function $f(\boldsymbol{x})$ and in the constraint functions $c_k(\boldsymbol{x}), k = 1, \ldots, p_2$. Y will be the set of variable symbols in the transformed functions $g(\boldsymbol{y})$, and $d_k(\boldsymbol{y})$, $k = 1, \ldots, p_2$. Remark, that dimension increase is not allowed for the transformation steps, so $m \le n$ and $|Y| \le |X|$. At the beginning of the algorithm, $Y := X$.

Denote the set of the expressions on the left-hand side of the original constraints by C:

$$C := \{c_k(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}\,,\ k = 1, \ldots, p_2\}.$$

Denote by F the expression set of all subexpressions (all well-formed expressions that are part of the original expressions) of $f(\boldsymbol{x})$ and $c_k(\boldsymbol{x}) \in C$.

The crucial part of our algorithm is the *transformation step*. If an $H \subset F$ expression set covers a $V \subseteq X$ variable set (that is, none of $v \in V$ happens out of H in the original expression of $f(\boldsymbol{x})$ or $c_k(\boldsymbol{x}) \in C$), and $|H| \leq |V|$, then apply every substitutions, related to H, to $f(\boldsymbol{x})$ and C as follows. Substitute a new variable y_i in place of $h_i(\boldsymbol{x})$ for all $h_i(\boldsymbol{x}) \in H$ in $f(\boldsymbol{x})$ and also in every $c_k(\boldsymbol{x}) \in C$. Furthermore, let us update the variable set of the transformed problem: $Y := (Y \cup y_i) \setminus V$.

This will be referred to as a transformation step (corresponding to H). The special case $|H| = |V| = 1$, $p_1 = p_2 = 0$ belongs to the algorithm given by Csendes and Rapcsák [5] for the unconstrained case.

Further on, the notation $\boldsymbol{y} := H(\boldsymbol{x})$ will be used as an abbreviation for the following: $y_i := h_i(\boldsymbol{x})$, $i = 1, \ldots, |H|$.

The following assertion is a straightforward generalization of Assertion 1 in [5].

Assertion 1. *If a variable x_i appears in exactly one term, namely in $h(\boldsymbol{x})$, everywhere in the expressions of the smooth functions $f(\boldsymbol{x})$ and $c_k(\boldsymbol{x})$, $k = 1, \ldots, p_2$, then the partial derivatives of these functions related to x_i all can be written in the form $(\partial h(\boldsymbol{x})/\partial x_i)p(\boldsymbol{x})$, where $p(\boldsymbol{x})$ is continuously differentiable.*

Recall, that an ordered set H of substitutions is called *proper*, if all expressions $h_i(\boldsymbol{x}) \in H$ are such that they can be substituted by new variables at the same time. Ordering is required only for univocal indexing of the substitutions.

Theorem 1. *If H is proper and all $h_i(\boldsymbol{x}) \in H$ expressions are smooth and strictly monotonic as a function of every variable $v \in V \subseteq X$, the cardinality of H is less than or equal to the cardinality of V, and the domain of $h_i(\boldsymbol{x})$ is equal to $\mathbb{R}$ for all $h_i(\boldsymbol{x}) \in H$, then the transformation step corresponding to H simplifies the original problem in such a way that every local minimizer (maximizer) point $\boldsymbol{x}^*$ of the original problem is transformed to a local minimizer (maximizer) point $\boldsymbol{y}^*$ of the transformed problem.*

Proof. Consider first the sets of feasibility for the two problems. The substitution equations ensure that if a point $\boldsymbol{x}$ was feasible for the problem (2), then it remains feasible after the transformations for the new, simplified problem (3). The same is true for infeasible points.

Denote now a local minimizer point of $f(\boldsymbol{x})$ by $\boldsymbol{x}^*$, and let $\boldsymbol{y}^* := H(\boldsymbol{x}^*)$ be the transformed form of $\boldsymbol{x}^*$. As each $h_i(\boldsymbol{x}) \in H$ is strictly monotonic in at least one variable, all points from the $a = N(\boldsymbol{x}^*, \delta)$ neighborhood of $\boldsymbol{x}^*$ will be transformed into a $b = N(\boldsymbol{y}^*, \epsilon)$ neighborhood of $\boldsymbol{y}^*$, and $\forall x_j \notin a : \ y_j \notin b$. Both the objective functions $f(\boldsymbol{x})$ and $g(\boldsymbol{y})$, and the old and transformed constraint functions have the same value before and after the transformation. This fact ensures that each local minimizer point $\boldsymbol{x}^*$ will be transformed into a local minimizer point $\boldsymbol{y}^*$ of $g(\boldsymbol{y})$. The same is true for local maximizer points, by similar reasoning.

Additionally, $|H| \leq |V|$, so the construction of the transformation step ensures that the application of every substitution of H eliminates at least as many x_i

variables from the optimization model as the number of the new variables in every iteration. □

In contrast to Theorem 1, which gives sufficient conditions to have such a simplification transformation that will bring local minimizer points of the old problem to local minimizer points of the new one, the following theorem provides sufficient conditions to have a one-to-one mapping of the minimizer points.

Theorem 2. *If H is proper, and all $h_i(\boldsymbol{x}) \in H$ expressions are smooth, strictly monotonic as a function of every variable $v \in V \subseteq X$, the cardinality of H is less than or equal to the cardinality of V, and the domain and range of $h_i(\boldsymbol{x})$ are equal to $\mathbb{R}$ for all $h_i(\boldsymbol{x}) \in H$, then the transformation step corresponding to H simplifies the original problem in such a way that every local minimizer (maximizer) point $\boldsymbol{y}^*$ of the transformed problem can be transformed back to a local minimizer (maximizer) point $\boldsymbol{x}^*$ of the original problem.*

Proof. Since the substitution equations preserve the values of the constraint functions, each point $\boldsymbol{y}$ of the feasible set of the transformed problem (3) must be mapped from a feasible point $\boldsymbol{x}$ of the original problem (2): $\boldsymbol{y} = H(\boldsymbol{x})$. The same holds for infeasible points.

Denote a local minimizer point of (3) by $\boldsymbol{y}^*$. Now, since the ranges of the transformation functions in H are equal to $\mathbb{R}$, every local minimizer point $\boldsymbol{y}^*$ of the transformed problem (3) is necessarily a result of a transformation: let $\boldsymbol{x}^*$ be the back-transformed point: $\boldsymbol{y}^* = H(\boldsymbol{x}^*)$. Consider a neighborhood $N(\boldsymbol{y}^*, \delta)$ of $\boldsymbol{y}^*$, where every feasible point $\boldsymbol{y}$ has a greater or equal function value: $g(\boldsymbol{y}) \geq g(\boldsymbol{y}^*)$, and those feasible points $\boldsymbol{x}$ of (2), for which $H(\boldsymbol{x}) = \boldsymbol{y} \in N(\boldsymbol{y}^*, \delta)$. The latter set may be infinite if the simplification transformation decreases the dimensionality of the optimization problem. Anyway, there is a suitable neighborhood $N(\boldsymbol{x}^*, \delta')$ of $\boldsymbol{x}^*$ inside this set, for which the relation $f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*)$ holds for all $\boldsymbol{x} \in N(\boldsymbol{x}^*, \delta')$ that satisfies the constraints of (2). In other words, $\boldsymbol{x}^*$ is a local minimum point of (2).

The argument for local maximizers is similar. □

Corollary 1 is an immediate consequence of Theorem 1:

Corollary 1. *If H is proper, all the $h_i(\boldsymbol{x}) \in H$ expressions are smooth and invertible as a function of every variable $v \in V \subseteq X$, and the cardinality of H is less than or equal to the cardinality of V, then the transformation step corresponding to H simplifies the original problem in such a way that every local optimum point $\boldsymbol{x}^*$ of the original problem is transformed to a local optimum point $\boldsymbol{y}^*$ of the transformed problem.*

5 Summary

This paper examines the possibility and ability of implementing equivalent transformations for nonlinear optimization problems as an automatic presolving phase of numerical global optimization methods.

An extensive computational test has been completed on standard global optimization test problems and on other often used global optimization test functions together with some custom made problems designed especially to test the capabilities of symbolic simplification algorithms. Maple and Mathematica based implementations were compared.

The test results show that most of the simplifiable cases were recognized by our new, Mathematica-based algorithm, and the substitutions were correct. Tests with a numerical solver, namely Global, were performed to check the usefulness of the produced transformations. The results show that the produced substitutions can improve the performance of this multi-start solver.

We have presented some new theoretical results on automatic symbolic transformations to simplify constrained nonlinear optimization problems. However, further investigations would be necessary to build an efficient branch and bound strategy into the algorithm at Step 3–4 to realize good running times for the described parallel substitutions.

As a natural extension of the present application, symbolic reformulations are promising for speeding up interval methods of global optimization. The overestimation sizes for interval arithmetic [1] based inclusion functions were investigated in optimization models [18]. Symbolic transformations seem to be appropriate for a proper reformulation. Obvious improvement possibilities in this field are the use of the square function instead of the usual multiplication (where it is suitable), the transformation along the subdistributivity law, and finding SUE forms. In fact, such transformations usually are performed by the default expression simplification mechanism [16] of an ordinary computer algebra system. The domain of calculation has an important role in this presolve approach, since important features of functions such as monotonicity change substantially within the domain where a function is defined.

References

[1] Alefeld, G., Herzberger, J. *Introduction to Interval Computation.* Academic Press, New York, 1983.

[2] Antal, E., Csendes, T., Virágh, J. Nonlinear Transformations for the Simplification of Unconstrained Nonlinear Optimization Problems. *Cent. Eur. J. Oper. Res.* 21(4):665–684, 2013.

[3] Avanzolini, G., Barbini, P. Comment on "Estimating Respiratory Mechanical Parameters in Parallel Compartment Models". *IEEE Trans. Biomed. Eng.* 29:772–774, 1982.

[4] Csendes, T., Pál, L., Sendín, J. O. H., Banga, J. R. The GLOBAL Optimization Method Revisited. *Optimization Letters* 2:445–454, 2008.

[5] Csendes, T., Rapcsák, T. Nonlinear Coordinate Transformations for Unconstrained Optimization. I. Basic Transformations. *J. Global Optim.* 3(2):213–221, 1993.

[6] Farkas, T., Rev, E., Lelkes, Z. Process flowsheet superstructures: Structural multiplicity and redundancy Part 1: Basic GDP and MINLP representations. *Computers and Chemical Engineering*, 29:2180–2197, 2005.

[7] Fourer, R., and Gay, D. M. Experience with a Primal Presolve Algorithm. In Hager, W. W., Hearn, D. W. and Pardalos, P. M., editors, *Large Scale Optimization: State of the Art*, pages 135–154. Kluwer Academic Publishers, Dordrecht, 1994.

[8] Gay, D. M. Symbolic-Algebraic Computations in a Modeling Language for Mathematical Programming. In Alefeld, G., Rohn, J. and Yamamoto, T., editors. *Symbolic Algebraic Methods and Verification Methods*, pages 99–106. Springer-Verlag, 2001.

[9] Grossmann, I. E. MINLP optimization strategies and algorithms for process synthesis. In *Proc. 3rd Int. Conf. on Foundations of Computer-Aided Process Design*, page 105., 1990.

[10] Liberti, L., Cafieri, S., Savourey, D. The Reformulation-Optimization Software Engine. *Mathematical Software – ICMS 2010, LNCS 6327*, pages 303–314, 2010.

[11] Mészáros, Cs., Suhl, U. H. Advanced preprocessing techniques for linear and quadratic programming. *OR Spectrum* 25(4):575–595, 2003.

[12] Neumaier, A. Improving interval enclosures. Manuscript. Available at `http://www.mat.univie.ac.at/~neum/ms/encl.pdf`

[13] Pál, L. *Global optimization algorithms for bound constrained problems.* Ph.D. thesis, University of Szeged, 2011.

[14] Rapcsák, T., Csendes, T. Nonlinear Coordinate Transformations for Unconstrained Optimization. II. Theoretical Background. *J. Global Optim.* 3(3):359–375, 1993.

[15] Schichl, H., Neumaier, A. Interval Analysis on Directed Acyclic Graphs for Global Optimization. *J. Global Optim.* 33(4):541–562, 2005.

[16] Stoutemyer, D. R. Ten commandments for good default expression simplification. *J. Symb. Comput.* 46:859–887, 2011.

[17] Sturm, T., Tiwari A. Verification and synthesis using real quantifier elimination. In *Proceedings of the 36th international symposium on Symbolic and algebraic computation (ISSAC '11)*, ACM, New York, NY, USA, 329–336, 2011.

[18] Tóth, B., Csendes, T. Empirical investigation of the convergence speed of inclusion functions. *Reliable Computing*, 11:253–273, 2005.

[19] Vajda, R. Effective Real Quantifier Elimination. In *J. Karsai, R. Vajda (eds.), Interesting Mathematical Problems in Sciences and Everyday Life - 2011*, University of Szeged, ISBN:978-963-306-109-1

[20] Wolfram Mathematica 9 Documentation Center, Mathematica Tutorial: Basic Internal Architecture. Available at `https://reference.wolfram.com/mathematica/tutorial/BasicInternalArchitecture.html`

Received 15th June 2015