Noname manuscript No.
(will be inserted by the editor)

Nonlinear Transformations for the Simplification of Unconstrained Nonlinear Optimization Problems

Elvira Antal · Tibor Csendes · János Virágh

Received: 24.12.2011 / Accepted: date

Abstract Formalization decisions in mathematical programming could significantly influence the complexity of the problem, and so the efficiency of the applied solver methods. This widely accepted statement induced investigations for the reformulation of optimization problems in the hope of getting easier to solve problem forms, e.g. in integer programming. These transformations usually go hand in hand with relaxation of some constraints and with the increase in the number of the variables. However, the quick evolution and the widespread use of computer algebra systems in the last few years motivated us to use symbolic computation techniques also in the field of global optimization.

We are interested in potential simplifications generated by symbolic transformations in global optimization, and especially in automatic mechanisms producing equivalent expressions that possibly decrease the dimension of the problem. As it was pointed out by Csendes and Rapcsák (1993), it is possible in some cases to simplify the unconstrained nonlinear objective function by nonlinear coordinate transformations. That means mostly symbolic replacement of redundant subexpressions expecting less computation, while the simplified task remains equivalent to the original in the sense that a conversion between the solutions of the two forms is possible.

We present a proper implementation of the referred theoretical algorithm in a modern symbolic programming environment, and testing on some examples both from the original publications and from the set of standard global optimization test problems to illustrate the capabilities of the method.

Keywords unconstrained nonlinear optimization · symbolic computation · reformulation · Maple

E. Antal, T. Csendes, and J. Virágh
University of Szeged, Institute of Informatics
H-6720 Szeged, Árpád tér 2, Hungary
Tel.: +36-62-544 305
Fax: +36-62-546 397
E-mail: csendes@inf.u-szeged.hu

1 Introduction

Global optimization problems are mostly solved by numerical algorithms, but the evolution of symbolic computation methodology and the accordingly powerful computer algebra systems (CAS) allow effective symbolic computation techniques also in this field.

Consider the unconstrained nonlinear optimization problem

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

where $f(x) : \mathbb{R}^n \to \mathbb{R}$ is a nonlinear, twice continuously differentiable function, given by symbolic expression, a formula. Our aim is to produce an equivalent problem form:

$$\min_{y \in \mathbb{R}^m} g(y), \tag{2}$$

where $g(y) : \mathbb{R}^m \to \mathbb{R}$ is simpler than $f(x)$. We mean a problem is simpler that an other one, if the former is easier to solve by optimization methods, while a direct transformation between the optimal solutions y^* and x^* is possible.

This kind of reformulation is usually done by hand in the stage of stating the problem, but that is not the only possibility. In the next section we will show several techniques, that do automatic manipulations on optimization problems in order to increase the efficiency of the solver. However, such a usage of symbolic computation is applied better just for linear and integer programming.

Section 3 presents our reformulation method for unconstrained nonlinear optimization problems, based on nonlinear coordinate transformations described in the paper by Csendes and Rapcsák (1993). The Maple implementation details are given in Section 4. Then Section 5 demonstrates the capabilities of this method on a set of standard global optimization test problems and on some other test functions.

2 Related works

This section gives a short outline of relevant manipulation techniques developed to simplify an optimization problem in some sense. We mention here the “presolving” mechanism of the AMPL processor, LP preprocessing, reformulation with relaxation for IP/MINLP solving, an interval branch and bound method speeded up by symbolic computation, and also unusual problem solving approaches that use algebraic techniques as quantifier elimination (QE), Gröbner bases, etc. for symbolic optimization.

Our approach is different, because it is not a relaxation, and we do not want to solve the complete problem with algebraic techniques (since it can be quite slow and may be not even successful), just to make our problem easier for the solver. We are interested in mechanisms producing equivalent transformations and such that possibly decrease the dimension of the problem. Now we present briefly the following manipulation techniques to provide a perspective.

AMPL is one of the most popular algebraic modeling languages, designed for describing optimization problems in a form close to the usual mathematical notation.

The AMPL processor can provide the problem and further information about it (the first, or the second derivatives etc.) to several solvers via the AMPL/solver interface library. So the AMPL processor is not a solver, but one can call CPLEX, MINOS, LANCELOT or other well-known third party solvers in a unified way for a problem, what is formulated in this modeling language.

Furthermore, the AMPL processor implements a preprocessing mechanism (called "presolving") what is executed before conveying the problem to the selected solver, and in fortunate cases it can detect infeasibility or it is able to reduce the size of the problem, which results in shorter transaction times and is advantageous mainly for the solvers that do not have its own preprocessor.

Its presolving mechanism was based on the method of Brearley et al. (1975), and applies only to linear objectives and constraints. Still in some cases it is able to fix nonlinear elements of a function, if the value of the variable can be determined by some other constraints. For further information about symbolic manipulations in AMPL, see the paper of Gay (2001).

In linear programming (LP) it is widely known, that applying various preprocessing methods before the solver phase can significantly improve the efficiency of both the simplex and the interior point methods. Possible steps in LP preprocessing are

– fixing some variables or reducing redundant constraints by primal/dual feasibility tests,
– recognizing the linearly dependent rows, tightening bounds and reducing redundant constraints or variables by elimination,
– reducing the constraint matrix to be as sparse as possible.

As the majority of the solving techniques in integer programming (IP) are based on relaxing the IP to a convenient LP problem, the efficient reconstruction of LP problems could benefit also in this field. The reformulation of IP to LP itself is an other interesting application of symbolic computation in optimization. The possible automatic transformations can enable the relaxation of constraints and even increasing the number of the variables with the aim of getting a solvable form of the given, otherwise hopelessly difficult problem.

There exists a freely available software package in the development stage for collecting useful reformulation algorithms in IP and mixed-integer nonlinear programming: Liberti et al. (2010). Preprocessing techniques of LP and a comparison on the results of large-scale problem preprocessing are summarized shortly by Mészáros and Suhl (2003) primarily for the execution of a simplex and an interior point method, but also with possible applications for IP and quadratic programming (QP).

Interval branch and bound speeded up by reformulation with relaxation (Byrne and Bogle 1999) is an interesting hybrid method combining symbolic reformulation with the numerical solver. The main idea is to rearrange the non-convex terms of the target function to the constraints and then simplify the complicated parts by interval bound relaxations with linear under-estimators.

Gröbner bases theory, QE and other algebraic techniques can also be used in some special cases to reformulate and solve optimization problems. Kanno et al. (2008) suggest symbolic optimization for a class of signal processing problems, i.e. using symbolic computation for the solution of the complete problem. It is not necessarily the most effective way, and the referred theory is available just for algebraic functions. However, the exploration of applications in connection with optimization could be an interesting field of symbolic computation.

3 Nonlinear coordinate transformations

As it was pointed out by Csendes and Rapcsák (1993), it is possible in some cases to simplify the unconstrained nonlinear objective function by nonlinear coordinate transformations. That means mostly symbolic replacement of redundant subexpressions expecting to relax the computing effort of the solver, while the simplified task still remains equivalent to the original one, in the sense that a direct transformation between the solutions of the two form is possible.

The published algorithm was demonstrated to be capable of solving unconstrained nonlinear optimization problems in fortunate cases by bringing them to a unimodal form, or simplifying them enough to make a substantial improvement for other optimization methods. In the latter case it acts as preprocessing.

The present section discusses the theoretical method, and Section 4 presents our implemented automatic simplifier program based on nonlinear coordinate transformations.

3.1 Motivation: a parameter estimation problem

Consider the parameter estimation problem discussed in Hantos et al. (1990), minimization of a sum-of-squares form objective function:

$$F\left(R_{aw}, I_{aw}, B, \tau\right) = \left[\frac{1}{m}\sum_{i=1}^{m}\left|Z_L(\omega_i) - Z_L'(\omega_i)\right|^2\right]^{1/2},$$

where $Z_L(\omega_i) \in \mathbb{C}$ is the measured impedance value, $Z_L'(\omega_i)$ is the modeled impedance at frequencies ω_i for $i = 1, 2, \ldots, m$ and R_{aw}, I_{aw}, B, and τ are model parameters. Parameter estimation problems form an important part of optimization models (e.g. just as in Anholcer (2011)), hence it is one of the main target area of our technique.

The original nonlinear model function is based on obvious physical parameters:

$$Z_L'(\omega) = R_{aw} + \frac{B\pi}{4.6\omega} - \imath\left(I_{aw}\omega + \frac{B\log(\gamma\tau\omega)}{\omega}\right),$$

where $\gamma = 10^{1/4}$ and $\imath$ is the imaginary unit. The study of Csendes and Rapcsák (1993) was motivated by the existence of a simplified and equivalent model function, that is linear in the model parameters:

$$Z_L'(\omega) = R_{aw} + \frac{B\pi}{4.6\omega} - \imath\left(I_{aw}\omega + \frac{A + 0.25B + B\log(\omega)}{\omega}\right).$$

The applied successful variable transformation was $A = B\log(\tau)$ that changed the problem characteristic from nonlinear to a linear least squares problem. In the same time, we obtained a proof that all local minimizer points of the original problem are global.

It is obvious, that such simplifications could be found for several nonlinear optimization problems, and in addition to that, even an automatic simplifier method could produce that kind of transformations.

3.2 Theoretical background

Problem (1) is easy to solve when f is unimodal, that is, f has a single region of attraction and so there is only one local minimizer point (and no local maximizer point) in the given set of feasibility $X \in \mathbb{R}^n$. One may recognize this property intuitively on one-dimensional functions, but in higher dimensions the decision is certainly not trivial in general. In this subsection we briefly present the results of Csendes and Rapcsák (1993) on how to observe the generalized unimodal property of n-dimensional functions by using nonlinear coordinate transformations. The proofs of the following theorems and assertions could be found in the original paper.

Let us define unimodality in the following way.

Definition 1 The n-dimensional $f(x)$ continuous function is *unimodal* on an open set $X \subseteq \mathbb{R}^n$ if there exists a set of infinite continuous curves such that the curve system is a homeomorphic mapping of the polar coordinate system of the n-dimensional space, and the function $f(x)$ grows strictly monotonically along the curves.

In this sense multimodal is synonymous with multiextremal, what is in accordance with common optimization usage. The mentioned curves can be regarded as those that belong to the trajectories of local searches. Remark, that an n-dimensional unimodal function is not necessarily unimodal along every line in the space. However, the definition suggests that we could introduce a simpler function (we will call it g) that suits to f along the mentioned curves, and f and g will have the same region of attraction.

In conclusion, if $f(x)$ is the objective function of an unconstrained nonlinear optimization problem, and we can recognize that $f(x)$ is unimodal in the sense of Definition 1, then it is possible to transform $f(x)$ to a simpler to solve one $g(y)$, minimize (maximize) $g(y)$ and get the extremum of $f(x)$ by simple back-transformations.

The implicit unimodality of a function can be formulated by variable transformations:

Theorem 1 *The continuous function $f(x)$ is unimodal in the n-dimensional real space if and only if there exists a homeomorph variable transformation $y = h(x)$ such that $f(x) = f(h^{-1}(y)) = y^T y + c$, where c is a real constant, and the origin is in the range S of $h(x)$.*

But how can we find such an h transformation function? In fact, we can construct possible substitutions that simplify f, and apply a sequence of them to reach a unimodal form in fortunate cases. The following theorems discuss the properties of substitutions that are needed to simplify the original function.

Theorem 2 *If $h(x)$ is smooth and strictly monotonic in x_i, then the corresponding transformation simplifies the function in the sense that each occurrence of $h(x)$ in the expression of $f(x)$ is padded by a variable in the transformed function $g(y)$, while every local minimizer (or maximizer) point of $f(x)$ is transformed to a local minimizer (maximizer) point of the function $g(y)$.*

So far the properties of h do not guarantee that f and g will be equivalent, but a mapping of all x^* to an appropriate y^* is given. Therefore we should test each transformed x^* whether that is indeed a solution of the original problem. The other option is to have further investigations before the substitution.

Theorem 3 *If $h(x)$ is smooth, strictly monotonic as a function of x_i, and its range is equal to $\mathbb{R}$, then for every local minimizer (or maximizer) point y^* of the transformed function $g(y)$ there exists an x^* such that y^* is the transform of x^*, and x^* is a local minimizer (maximizer) point of $f(x)$.*

The described simplifier substitutions could even eliminate some variables. In other words, we can recognize redundancies. For example, for the two-dimensional function $f(x_1,x_2) = (x_1 + x_2)^2$ the transformation $y_1 = x_1 + x_2$ satisfies the conditions of Theorem 3 for both x_1 and x_2, and it results in a one-dimensional objective function. The recognition of such redundant variables is beneficial and usually by far not trivial. In this way the outcome of the transformation sequence could be favorable even if a unimodal problem form could not be reached.

The following two theoretical statements provide additional hints on how to identify eliminable variables:

Assertion 1 *If a variable x_i appears everywhere in the expression of a smooth function $f(x)$ in a term $h(x)$, then the partial derivative $\partial f(x)/\partial x_i$ can be written in the form $(\partial h(x)/\partial x_i)\, p(x)$, where $p(x)$ is continuously differentiable.*

Consequently, if $\partial f(x)/\partial x_i$ is not factorizable, then $h(x)$ is linear in the variable x_i for the possible transformations (satisfying the conditions of Theorem 2).

Obviously, finding the appropriate $(\partial h(x)/\partial x_i)\, p(x)$ factorization is not necessarily easy, as many other factorizations can exist beside the one mentioned in Assertion 1. Yet a canonical form can be derived for a wide class of functions, e.g. for a polynomial $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ which have n roots $(x_1, \ldots, x_n)$ a standard factorization $a_n (x - x_1)(x - x_2) \cdots (x - x_n)$ always exists. Even if $\partial h(x)/\partial x_i$ can be determined, it may be difficult to find a good transformation function $h(x)$.

The condition of the existence of a proper variable transformation $h(x)$ that decreases the number of variables of an unconstrained nonlinear optimization problem is given in the following assertion.

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

These theoretical statements are the basis for the constructive symbolic algorithm, the implementation of which is described in the next section.

4 Implementation of a simplifier method for unconstrained nonlinear optimization problems

The aim of the present study was to create a proper implementation of the referred theoretical algorithm to be able to observe the limitations of the method in practice, and to document advantages and disadvantages of the algorithm on a wide set of global optimization problems. It was realized that a typical computer algebra system (CAS), such as Maple provides most of the required symbolic manipulations: partial differentiation, factorization, (symbolic) integration, list and string manipulation, substitution of expressions, solution of equations, etc. The fundamentals of the program were established by János Virágh in Maple. That implementation was reprogrammed and extended by Csilla Urbán, who wrote her masters thesis in this topic (Urbán 2009). Code review and further substantial improvements of the latter program resulted in the Maple program presented here.

4.1 Transformation method

The possible aims of an automatic simplifier method for unconstrained nonlinear optimization problems are as follows:

substitute some subexpressions in the objective function in order to

- eliminate parts of the computation tree,
- recognize unimodality,
- get an equivalent simpler form of the problem requiring less computation, and
- reduce (or at least not extend) the dimension of the problem.

The pseudo-code of our implementation is presented in Algorithm 1. This is a draft of the main function `symbsimp(x,f)` that has two input parameters: `f` contains the formula of the objective function of the given unconstrained nonlinear optimization problem and `x` represents the variable vector in the form of a Maple list.

According to Algorithm 1, the basic transformation steps are:

1. compute the gradient of the original function,
2. factorize the partial derivatives,
3. determine the substitutable subexpressions and substitute them:
   (a) if factorization was successful, then explore the subexpressions that can be obtained by integration of the factors,
   (b) if factorization was not possible, then explore the subexpressions that are linear in the related variables,
4. solve the simplified problem if possible, and give the solution of the original problem by transformation, and
5. verify the obtained results.

The most important necessary subroutines of the program are listed in Table 1 with their possible solutions in Maple.

The remaining questions that need further examination are detailed in the next subsection.

Algorithm 1 symbsimp(x, f) // f: function formula, x: variable list

```
g ← f
subslist ← {}
subsnumber ← 0
for $i = 1$ to Dimension(x) do
    dx ← $\bigtriangledown$g
    factordx ← Factor(dx)
    $h_i$ ← Null
    if factordx $\neq$ dx then
        h_list ← Null
        for $j \in$ factordx do
            p ← $\int$ factordx $dx_i$
            if NumbOccur(g, p) = NumbOccur(g, $x_i$) then
                repeat
                    h_temp ← Sort({$q$ | $q$ ∈Decompose(g, p) and IsType($q$, '+')}, p).Pop
                    if Test(h_temp, $x_i$) then
                        h_list.Push(h_temp)
                    end if
                until Test(h_temp, $x_i$) or h_temp = Null
            end if
        end for
        $h_i$ ← Sort(h_list, $x_i$).Front
    end if
    if $h_i$ = Null then
        repeat
            h_temp ← Sort({$q$ | $q$ ∈Decompose(g, $x_i$) and IsLinear($q$, $x_i$)}, $x_i$).Pop
        until Test(h_temp, $x_i$) or h_temp = Null
        if Test(h_temp, $x_i$) then
            $h_i$ ← h_temp
            subsnumber++
        else
            $h_i$ ← $x_i$
        end if
    end if
    g ← Subs(g, $h_i$, '$y_i$')
    subslist.Push("$y_i$ =$h_i$")
end for
if subsnumber $\neq$ 0 then
    $g^*$ ← Solve[g, y]
    $f^*$ ← Transform($g^*$, subslist)
    Verify(f, g)
end if
```

4.2 Open questions

4.2.1 Proper algebraic substitutions

Maple offers two types of substitutions: the syntactic substitution `subs`, and the algebraic one `algsubs`. The `simplify` command provides also a related service, as it is able to apply user-defined rules for an expression, as long as the side relations in the rules are polynomials in the variables. We collected some trivial examples to show the abilities of this three commands side-by-side in Maple 15:

Table 1 Required subroutines of the implementation and Maple solutions for them

`Factor(expr)`	give a list of the factors of `expr`, or give back `expr` if it is not factorisable Maple: `factor`, `afactor` (built-in functions)
`NumbOccur(expr,y)`	calculate the number of occurrences of `y` in `expr` Maple: `numboccur` (built-in function) with some problems (cf. Subsubsection 4.2.1)
`Decompose(expr,y)`	provide a list of the part expressions of `expr` that contain `y` Maple: realized by the recursive use of the built-in function `convert(expr,list)`
`IsType(expr,'+')`	set true, if the operator with highest precedence in the expression `expr` is addition or subtraction Maple: `whattype` (built-in function)
`IsLinear(expr,y)`	set true, if the expression `expr` is linear in `y` Maple: `type` and `linear` (built-in functions)
`Sort(exprlist,y)`	order the list `exprlist` of possible substitutions for variable `y` in the decreasing order by the usefulness of the substitution (it is not trivial, we prefer polynomials, even if it is a more complex expression) Maple: realized by list operations and built-in functions `sort`, `PolynomialTools[Sort]`
`Test(expr,y)`	give the value true, if `expr` satisfies all special conditions for simplification by `y` if it is substituted by a new variable (now we test whether `expr` characterizes all occurrences of `y`, and the range of `expr` is equal to $\mathbb{R}$) Maple: not trivial (cf. Subsubsection 4.2.2)
`Subs(expr,x,y)`	substitute `y` in the place of `x` in `expr` Maple: `subs`, `algsubs` (built-in functions) with some problems (cf. Subsubsection 4.2.1)
`Solve(expr,y)`	try to minimize `expr` according to the variables `y` Maple: not trivial, for example the built-in function `minimize`
`Transform(sollist,subslist)`	give the solutions of the original function, if `sollist` contains the minima of the transformed function and regarding the transformations given in `subslist` Maple: `subs` and `solve` (built-in function)
`Verify(x,y)`	check whether `x` and `y` are equivalent as objective functions (we apply range calculation for `x` and `y`) Maple: not trivial (cf. Subsubsection 4.2.3)

Command	Result	Is it sufficient?
`subs(a*b = d, a*b*c)`	abc	F
`algsubs(a*b = d, a*b*c)`	cd	T
`simplify(a*b*c, {a*b = d})`	cd	T
`subs(a = b, 1/a)`	$\frac{1}{b}$	T
`algsubs(a = b, 1/a)`	$\frac{1}{a}$	F
`simplify(1/a, {a = b})`	$\frac{1}{b}$	T
`subs(sqrt(a + b) = d, sqrt(a + b) + c)`	$c+d$	T
`algsubs(sqrt(a + b) = d, sqrt(a + b) + c)`	error	F
`simplify(sqrt(a + b) + c, {sqrt(a + b) = d})`	error	F
`subs(2^(a + b) = d, 2^(a + b))`	d	T
`algsubs(2^(a + b) = d, 2^(a + b))`	error	F
`simplify(2^(a + b), {2^(a + b) = d})`	error	F

It seems, that neither of the mentioned possibilities are perfect. However, `subs` is unambiguously the most robust procedure. This is the reason, why we use this in our implementation.

Perhaps the inconvenient substitutions cause that `numboccur` produces false output in some cases:

Command	Result	Is it sufficient?
`numboccur(a*b+a*b,a*b)`	1	T
`numboccur(a*b*c,a*b)`	0	F
`numboccur(1/a,a)`	1	T
`numboccur(1/(a*b),a*b)`	0	F
`numboccur(Exp(Sin(a/b))*Exp(1),Sin(a/b))`	1	T
`numboccur(Exp(Sin(a/b)+1),Sin(a/b))`	0	F

4.2.2 Assumptions for the substituted expressions

As we mentioned before, we need to check whether an $y = h(x)$ substitution satisfies some special conditions, that guarantee it will in fact simplify the original function expression. It is required to ensure the completeness and soundness of the whole reformulation process, i.e. to prove that the original and the finally transformed problem forms are equivalent, no solution is lost and no false solution is produced.

Theorem 3 states that an expression $h(x)$ is suitable for such a proper substitution, if

- the range of it is the whole set of real numbers: $R(h) = \mathbb{R}$,
- $h(x)$ is smooth, and
- $h(x)$ is strictly monotonic.

However, these are not necessarily the tightest possible conditions for such proper substitutions, and the computer realization of these tests is not easy. Generally speaking, the exact range calculation for a function is just as a hard problem as to find the global minimum and maximum, and therefore we need some tricks to estimate it approximately, as you can follow in the next subsubsection. Further investigations are needed to specify appropriate, and still easy to verify properties to check in the computer implementation.

4.2.3 Range calculation

For this purpose, the most obvious approach would be the usage of natural interval extension. We found two approximately adequate solutions for using interval operations in Maple. The first is the built-in “range arithmetic”, and the second is the external package intpakX (Grimmer 2003). Both tend to behave incorrectly in some cases.

Maple itself is able to handle simple interval computations by evaluating specially defined interval expressions with the function `evalr`. However, as it was stated by

Grimmer et al. (2004), the proper outward rounding and other necessary features fail in this range arithmetic to produce reliable results. We mention here just the fact, that the automatic simplifier mechanism often goes wrong with the built-in interval type, so one should use results given by Maple range arithmetic with caution. As an example, see the code:

```
> a := INTERVAL(-1 .. 1);
INTERVAL(-1 .. 1)
> b := INTERVAL(-1 .. 1);
INTERVAL(-1 .. 1)
> evalr(a*b);
INTERVAL(0 .. 1)
```

The first two commands declare variables $a = [-1,1]$ and $b = [-1,1]$ having the interval data type. The next line computes the product $a \cdot b$, and Maple gives the result $[0,1]$ as if it would be just a^2.

The external package intpakX defines types, operators, and special applications for real intervals and complex disc intervals. Now, we are interested just in the first group of features, i.e. for the calculation with real intervals in Maple. The intpakX interval operations are implemented with names different from the Maple interval operations to ensure separation. Outward rounding is also done in a separate way. However, standard Maple operators and functions are commonly used by intpakX, and hence if the arithmetic of the CAS produces errors greater than one unit of least precision (1 ULP), it can also appear in the results calculated by intpakX (Grimmer et al. 2004). IntpakX aims to implement correct interval arithmetic in Maple, but there are still some deficiency in this package. For example, the extended division for unbounded intervals (intervals with infinite endpoints) is impossible:

```
> ext_int_div(construct(-infinity, -1), construct(1, 2)):
Error, (in intpakX:-ext_int_div) first arg must be a finite
 interval or a numeric
```

Unfortunately, unbounded intervals are quite common in unconstrained nonlinear optimization, and the documentation does not prohibit the use of them. However, it is possible to recompile the package for using extended divisions for unbounded intervals (I. Bársony, personal communication), but we were not able to eliminate completely the emerging uncertainty of the operations with unbounded intervals yet.

In fact, intpakX contains also range enclosure algorithms, but only for 2D and 3D functions (Grimmer 2003; Grimmer et al. 2004).

Remark, that if a function is continuous and strictly monotonic in the variables (as Theorem 3 requires), then the range calculation on a computer could be quick and precise with interval arithmetic, cf. Neumaier (2008). Still, there is no easy way to check in Maple whether a user-defined function is monotonic or not.

Because of the mentioned problems of range-calculation in Maple, our implementation applies only the second derivative test. However, there is a way in Maple to use complete interval packages of other computational systems. An obvious approach would be the usage of INTLAB (Rump 1999) under MATLAB, as Maple is able to build a connection to MATLAB in running time, when both programs are installed on the same computer. First we need to set up INTLAB for MATLAB, then it is usable with the `Matlab` package of Maple. Automatically one MATLAB session is assigned to one Maple worksheet, which starts it. The user can open and close MATLAB sessions manually by `Matlab[openlink]` and `Matlab[closelink]` functions. This line create an `x` interval variable in INTLAB:

```
> Matlab[evalM]("x=infsup(1,2)");
```

Naturally, in the same way we can evaluate any other commands in MATLAB. Then we can get an interval like this:

```
> Matlab[evalM]("[a]=[x.inf,x.sup]");
> Matlab[getvar]("a");
```

It seems, that the available connection packages offer the best way to apply interval calculations from a Maple worksheet.

5 Computational results

We tested our implementation on all the examples both from the original publication (Csendes and Rapcsák 1993) and from the set of standard global optimization test problems. We also constructed some additional test functions to observe specific features of our method.

5.1 A successful example

We have tested our method on the Rosenbrock function from the original paper (Csendes and Rapcsák 1993). This is one of the standard global optimization test functions also, that will be discussed in Subsection 5.3 too. The objective function is:

$$f(x) = 100\left(x_1^2 - x_2\right)^2 + (1-x_1)^2.$$

We run the simplifier algorithm with the procedure call:

```
symbsimp([x2, x1], 100*(x1^2-x2)^2+(1-x1)^2);
```

In the first step, the algorithm determines the partial differentials:

$$\texttt{dx(1)} = -200x_1^2 + 200x_2$$

$$\texttt{dx(2)} = 400(x_1^2 - x_2)x_1 - 2 + 2x_1$$

Here the order of the variables given in the first input parameter is not irrelevant with respect of the execution sequence. This is why we give them in a list, not in a set. Hence `dx(1)` is the partial derivative of the Rosenbrock function with respect to x_2.

Then the factorized forms of the partial derivatives are computed:

$$\texttt{factor(dx(1))} = -200x_1^2 + 200x_2,$$

$$\texttt{factor(dx(2))} = 400x_1^3 - 400x_1x_2 - 2 + 2x_1.$$

As the Maple function `factor` is not capable to factorize any of the two partial derivatives, we obtain the original expression (except from multiplying out the brackets by the automatic simplifier mechanism of Maple).

The list of the subexpressions of f ordered by the complexity in x_2 is the following:

$$\{100(x_1^2-x_2)^2,(x_1^2-x_2)^2,x_1^2-x_2,-x_2,x_2,(1-x_1)^2,x_1^2,100,2,-1\}.$$

This list is built up by recursive decomposition of f and contains the simplest multipliers of x_2 also, however, we will never use the constant terms on the last positions. As the factorization of `dx(1)` was not successful, we examine the first element of the list what is linear in x_2: that is $x_1^2-x_2$. The range of this expression is $(-\infty,+\infty)$, and it characterize all occurrences of x_2, so the substitution $y_1=x_1^2-x_2$ is applicable.

The transformed function at this point of the algorithm is $g=100y_1^2+(1-x_1)^2$.

Now compute again the partial derivatives and its factorization:

$$\texttt{factor(dx(1))}=\texttt{dx(1)}=200y_1,$$

$$\texttt{factor(dx(2))}=\texttt{dx(2)}=-2+2x_1.$$

The subexpressions of g ordered by the complexity in x_1 are

$$\{(1-x_1)^2,1-x_1,-x_1,x_1,100y_1^2,2,1,-1\}.$$

We are looking again for the first subexpression, what is linear in the actual variable x_1. Our test shows, that $1-x_1$ have all the necessary properties needed by a good substitution (the range is $(-\infty,+\infty)$, and the characterization of all occurrences of x_1). So substitute $y_2=1-x_1$.

The final simplified function, what our automatic simplifier method produced is

$$g=100y_1^2+y_2^2.$$

The extreme value of the function g can be calculated easily also by the build-in Maple function `minimize`: $y_1=0,y_2=0$. The minimum of the original function, f can be determined by the respective inverse transformations: $x_1=1,x_2=1$.

Since the ranges of all substitutions are equal to the set of real numbers, all minima of the original function are found. As the range of the simplified function is a subset of the range of the original function ($[0,+\infty)\subseteq[0,+\infty)$), all transformed minima should be valid solutions of the original problem.

5.2 An unsuccessful example

We have also tried to simplify the more complex parameter estimation problem of Csendes and Rapcsák (1993):

$$F\left(R_{aw}, I_{aw}, B, \tau\right) = \left[\frac{1}{m}\sum_{i=1}^{m}\left|Z_L(\omega_i) - Z_L'(\omega_i)\right|^2\right]^{1/2}.$$

It was disappointing to found that Maple does not handle the formula of F correctly with an undefined m, so in our tests we have to fix it to a constant (for this example, we chose $m = 3$). The original form of the model function is

$$Z_L'(\omega) = R_{aw} + \frac{B\pi}{4.6\omega} - \imath\left(I_{aw}\omega + \frac{B\log(\gamma\tau\omega)}{\omega}\right),$$

and a non-trivial substitution is hoped in τ correspondingly to the description in Subsection 3.1.

Actually, the partial derivative with respect to τ is factorable, and an indefinite integral of the factor $1/\tau$ with respect to τ is found to be $\ln(\tau)$, but the program do not explore the expressions that contain this subexpression because the number of occurrences of τ and $\ln(\tau)$ in the original function calculated to be unequal. The inadequate substitution possibilities of Maple mislead our algorithm in this case. So the only substitution that was found is $y_2 = -R_{aw}$, while renaming ω to y_1 produced the strange $y_1[1], y_1[2], y_1[3]$ terms instead of $\omega_1, \omega_2, \omega_3$). Results for this problem are summarized in Table 2 as ParamEst1, where g_1 refers to the final transformed target function, almost equal to the original one:

$$\begin{aligned} g_1 = & 0.5773502693\left(\left|Z_L\left(y_1[1]\right) + y_2 - \frac{0.6829549246 y_4}{y_1[1]}\right.\right. \\ & + \imath\left(y_3 y_1[1] + \frac{0.4342944819 y_4 \ln(1.778279410 y_5 y_1[1])}{y_1[1]}\right)\Bigg|^2 \\ & + \left|Z_L(y_1[2]) + y_2 - \frac{0.6829549246 y_4}{y_1[2]}\right. \\ & + \imath\left(y_3 y_1[2] + \frac{0.4342944819 y_4 \ln(1.778279410 y_5 y_1[2])}{y_1[2]}\right)\Bigg|^2 \\ & + \left|Z_L(y_1[3]) + y_2 - \frac{0.6829549246 y_4}{y_1[3]}\right. \\ & \left. + \imath\left(y_3 y_1[3] + \frac{0.4342944819 y_4 \ln(1.778279410 y_5 y_1[3])}{y_1[3]}\right)\Bigg|^2\right)^{1/2} \end{aligned}$$

At this point we need to explain the codes, what we use in the "Result type" column of the Tables 2 – 5 to characterize the results:

For the actual problem …

Table 2 Our results on the examples of the original publication

ID	Function f	Function g	Substitutions	Result type
Cos	$\cos(e^{x_1}+x_2)+\cos(x_2)$	$\cos(y_1)+\cos(y_2)$	$y_1 = e^{x_1}+x_2, y_2 = x_2$	A1
ParamEst1	$[\frac{1}{3}\sum_{i=1}^{3} \lvert Z_L(\omega_i) - Z'_L(\omega_i)\rvert^2]^{1/2}$	g_1	$y_1 = \omega, y_2 = -R_{aw}, y_3 = I_{aw}, y_4 = B, y_5 = \tau$	A2a
ParamEst2	$[\frac{1}{3}\sum_{i=1}^{3} \lvert Z_L(\omega_i) - Z''_L(\omega_i)\rvert^2]^{1/2}$	$0.5773502693y_5^{1/2}$	$y_1 = \omega, y_2 = -R_{aw}, y_3 = I_{aw}, y_4 = B, y_5$	A3ab
ParamEst3	$[\frac{1}{3}\sum_{i=1}^{3} \lvert Z_L(\omega_i) - Z'''_L(\omega_i)\rvert^2]^{1/2}$	$0.5773502693y_5^{1/2}$	$y_1 = \omega, y_2 = -R_{aw}, y_3 = I_{aw}, y_4 = B, y_5$	A3b
Otis	$(\lvert Z_L(s) - Z_m(s)\rvert^2)^{1/2}$	$(\lvert -Z_L[1] + 1.\frac{y_2}{y_4}\rvert^2)^{1/2}$	$y_1 = s, y_2 = I_C(R_1+R_2)C_1C_2y_1^3 + (I_C(C_1+C_2)+(R_C(R_1+R_2)+R_1R_2)C_1C_2)y_1^2 + (R_C(C_1+C_2)+R_1C_1+R_2C_2)y_1+1, y_4 = (R_1+R_2)C_1C_2y_1^2+(C_1+C_2)y_1$	B3

A: simplifying transformations are possible according to the presented theory,
B: simplifying transformations are possible with the extension of the presented theory,
C: some useful transformations could be possible with the extension of the presented theory, but they do not necessarily simplify the problem (e.g. since they increase the dimensionality),
D: we do not expect any useful transformation.

Our program produced …

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

The mistake is due to the incomplete …

a: algebraic substitution,
b: range calculation.

We will use the code 2 also when only a constant multiplier is eliminated, as above in the case of y_2.

We use the Maple notation in Table 2 where "1." represents a numeric value, a floating point number what is very close to the integer 1.

After the first unsuccessful try we have reformed our problem to have separate log terms in the model function:

$$Z''_L(\omega) = R_{aw} + \frac{B\pi}{4.6\omega} - \iota\left(I_{aw}\omega + \frac{B(\log(\gamma)+\log(\tau)+\log(\omega))}{\omega}\right),$$

and the respective problem version is called ParamEst2. In this form the program will inspect the subexpressions that contain $\ln(\tau)$, but unfortunately the inappropriate range calculation enables this unhelpful substitution:

$$\begin{aligned}
y_5 = \Bigg| & Z_L(y_1[1]) + y_2 - \frac{0.6829549246 y_4}{y_1[1]} + \imath\Bigg(y_3 y_1[1] \\
&+ \frac{y_4(0.2500000000 + 0.4342944819\ln(\tau) + 0.4342944819\ln(y_1[1]))}{y_1[1]}\Bigg)\Bigg|^2 \\
&+ \Bigg| Z_L(y_1[2]) + y_2 - \frac{0.6829549246 y_4}{y_1[2]} + \imath\Bigg(y_3 y_1[2] \\
&+ \frac{y_4(0.2500000000 + 0.4342944819\ln(\tau) + 0.4342944819\ln(y_1[2]))}{y_1[2]}\Bigg)\Bigg|^2 \\
&+ \Bigg| Z_L(y_1[3]) + y_2 - \frac{0.6829549246 y_4}{y_1[3]} + \imath\Bigg(y_3 y_1[3] \\
&+ \frac{y_4(0.2500000000 + 0.4342944819\ln(\tau) + 0.4342944819\ln(y_1[3]))}{y_1[3]}\Bigg)\Bigg|^2 .
\end{aligned}$$

If we examine the list of the potential subexpressions managed by the program, we can find, that the most favorable option is $0.4342944819\ln(\tau)$, so the mistake of the range calculation is not the only reason, why we do not get the expected substitution. A new test with altered model function is needed to clarify all technical difficulty, what causes the failure of the simplification for this example.

The third form of the original parameter estimation problem has the multiplication by B one by one in the numerator (called as ParamEst3):

$$Z_L'''(\omega) = R_{aw} + \frac{B\pi}{4.6\omega} - \imath\left(I_{aw}\omega + \frac{B\log(\gamma) + B\log(\tau) + B\log(\omega)}{\omega}\right).$$

The complicated substitution is this time:

$$\begin{aligned}
y_5 = \Bigg| & Z_L(y_1[1]) + y_2 - \frac{0.6829549246 y_4}{y_1[1]} + \imath\Bigg(y_3 y_1[1] \\
&+ \frac{0.2500000000 y_4 + 0.4342944819 y_4\ln(\tau) + 0.4342944819 y_4\ln(y_1[1])}{y_1[1]}\Bigg)\Bigg|^2 \\
&+ \Bigg| Z_L(y_1[2]) + y_2 - \frac{0.6829549246 y_4}{y_1[2]} + \imath\Bigg(y_3 y_1[2] \\
&+ \frac{0.2500000000 y_4 + 0.4342944819 y_4\ln(\tau) + 0.4342944819 y_4\ln(y_1[2])}{y_1[2]}\Bigg)\Bigg|^2 \\
&+ \Bigg| Z_L(y_1[3]) + y_2 - \frac{0.6829549246 y_4}{y_1[3]} + \imath\Bigg(y_3 y_1[3] \\
&+ \frac{0.2500000000 y_4 + 0.4342944819 y_4\ln(\tau) + 0.4342944819 y_4\ln(y_1[3])}{y_1[3]}\Bigg)\Bigg|^2 .
\end{aligned}$$

It differs form the previous y_5 only in the bracketing of the numerator. Finally the list of the potential subexpressions will contain $0.4342944819 y_4 \ln(\tau)$ (and $y_4 = B$ from an earlier renaming). However, we cannot reach it because of the similar reason as in the case of ParamEst2.

The last problem we have studied was the Otis modell from the original paper. Its objective function is

$$Z_m(s) = \frac{Ds^3 + Cs^2 + Bs + 1}{Gs^2 + Fs},$$

where $s = j\omega$ and

$$\begin{aligned}
B &= R_C(C_1 + C_2) + R_1 C_1 + R_2 C_2, \\
C &= I_C(C_1 + C_2) + [R_C(R_1 + R_2) + R_1 R_2] C_1 C_2, \\
D &= I_C(R_1 + R_2) C_1 C_2, \\
F &= C_1 + C_2, \\
G &= (R_1 + R_2) C_1 C_2.
\end{aligned}$$

It is quite easy to see how to simplify it: the original parameters of the model are in the order of appearance R_C, C_1, C_2, R_1, R_2, and I_C. A natural way to simplify it is to use the definition equations, and hence the new parameters of B, C, D, F, and G. What is more, it means in the same time a dimension reduction as well. As it was stated in the original paper, and as it is easy to see, this type of simplification is not to be obtained by the suggested method. Table 2 indicates just the same.

5.3 Standard and other global optimization test problems

The most obvious question about our method is how it works on well known nonlinear optimization test functions. The optimization experts know these problems well enough to have a picture on what is doable in terms of simplification. The results of our program are comprised in Table 3.

As it was already reported, the Rosenbrock function is easy to simplify. It is by no means a surprise, since it was obviously constructed to have a simply to understand structure, and the difficulty in its solution was better just for gradient and to a less extent for quasi-Newton methods. The simplification of the RCOS function is in a way a surprise. All in all, the results for the standard global optimization problems mean a full success: all simplifiable problems were recognized as such, as solved, while the more complex ones were correctly identified as not to be simplified.

Next we have investigated some other often used global optimization test problems (see Table 4). In this set only problem Schwefel-227 could be simplified, while the others were found to be too difficult to simplify (correctly).

Finally we demonstrate the capabilities and limitations of the presented automatic simplification tool on custom made test functions as listed in Table 5. According to our experiences, most of the emerging implementation problems could be resolved in a more favorable computational environment. The range of the simplifiable problems could be extended by further improvement on the theory.

Table 3 Standard global optimization test functions

ID	Function g	Substitutions	Result type
Rosenbrock	$100y_2^2+(1-y_1)^2$	$y_1=x_1, y_2=y_1^2-x_2$	A1
Shekel-5	memory error	none	D2
Hartman-3	none	none	D2
Hartman-6	none	none	D2
Goldstein-Prize	none	none	D2
RCOS	$y_2^2+10(1-1/8/\pi)*\cos(y_1)+10$	$y_1=x_1, y_2=5/\pi y_1-1.275000000y_1^2/\pi^2+x_2-6$	A1
Six-Hump-Camel-Back	none	none	D2

Table 4 Other often used global optimization test functions

ID	Function g	Substitutions	Result type
Levy-1	none	none	D2
Levy-2	none	none	D2
Levy-3	none	none	D2
Booth	none	none	C2
Beale	none	none	C2
Powell	$(y_1+10y_2)^2+5(y_3+y_4)^2+(y_2-2y_3)^4+10(y_1+y_4)^4$	$y_1=x_1, y_2=x_2, y_3=x_3, y_4=-x_4$	D2
Matyas	none	none	D2
Schwefel ($n=2$)	none	none	C2
Schwefel-227	$y_2^2+0.25y_1$	$y_1=x_1, y_2=y_1^2+x_2^2-2y_1$	A1
Schwefel-31 ($n=5$)	none	none	D2
Schwefel32 ($n=2$)	none	none	D2
Rastrigin ($n=2$)	none	none	C2
Ratz-4	none	none	C2
Easom	none	none	D2
Griewank-5	none	none	D2

6 Summary

We implemented a symbolic algorithm for the simplification of nonlinear optimization problems in a popular software environment. Our test results show, that this kind of preprocessing of an unconstrained optimization problem could be done in seconds on usual computers, and the result is favorable in many cases. Although just a portion of the studied problems could be simplified by our technique, the presented method could substantially improve the efficiency of optimization software. In some cases the dimension reduction or the recognition of redundant model parameters to be achieved in this way are invaluable for the experts who analyze optimization models.

Table 5 Some additional simple test functions to demonstrate the capabilities of our technique

ID	Function f	Function g	Substitutions	Result type
Cos1	$100\cos(x_1+x_2)$	$100\cos(y_1)$	$y_1 = x_1+x_2$	A1
Sin1	$\sin(2x_1+x_2)$	$\sin(y_1)$	$y_1 = 2x_1+x_2$	A1
Sin2	$2x_3\sin(2x_1+x_2)$	$2y_1$	$y_1 = x_3\sin(2x_1+x_2)$	A1
Abs1	$\lvert x_1/x_2\rvert$	$\lvert y1\rvert$	$y_1 = x_1/x_2$	B1
Exp1	$e^{x_1+x_2}$	e^{y_1}	$y_1 = x_1+x_2$	A1
Exp2	$2e^{x_1+x_2}$	$2y_1$	$y_1 = e^{x_1+x_2}$	A3b
Sq1	$x_1^2x_2^2$	none	none	A2a
Sq2	$(x_1x_2+x_3)^2$	y_1^2	$y_1 = x_1x_2+x_3$	A1
SqSin1	$(x_1+x_2)^4+$ $26\sin(x_1+x_2)$	$y_1^4+26\sin(y_1)$	$y_1 = x_1+x_2$	A1
SqCos1	$(x_1x_2+x_3)^2-$ $\cos(x_1x_2)$	$y_3^2-\cos(y_1)$	$y_1 = x_1x_2, y_3 = y_1+x_3$	A1
SqExp1	$(x_1+x_2)^2+e^{x_1+x_2}$	$y_1^2+e^{y_1}$	$y_1 = x_1+x_2$	A1
SqExp2	$(x_1+x_2)^2+2e^1e^{x_1+x_2}$	$y_1^2+2e^1e^{y_1}$	$y_1 = x_1+x_2$	A1
SqExp3	$(x_1+x_2)^2+2e^{1+x_1+x_2}$	none	none	A2a

Acknowledgements The presented work was supported in part by the Grant TÁMOP-4.2.2/08/1/2008-0008.

References

Anholcer, M., V. Babiy, S. Bozoki, and W.W. Koczkodaj (2011) A simplified implementation of the least squares solution for pairwise comparison matrices. *Central European J. of Operations Research* 19(4):439-444

Brearley, A.L., G. Mitra, H.P. Williams (1975) Analysis of mathematical programming problems prior to applying the simplex algorithm. *Mathematical Programming* 8(1):54–83

Byrne, R.P., I.D.L. Bogle (1999) Global optimisation of constrained non-convex programs using reformulation and interval analysis. *Computers and Chemical Engineering* 23:1341–1350

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

Gay, D.M. (2001) Symbolic-Algebraic Computations in a Modeling Language for Mathematical Programming. *In Symbolic Algebraic Methods and Verification Methods, G. Alefeld, J. Rohn, and T. Yamamoto, eds, Springer-Verlag*, pp. 99–106

Grimmer, M. (2003) Interval Arithmetic in Maple with intpakX. *PAMM, Wiley-InterScience* 2(1): 442–443

Grimmer, M., K. Petras, and N. Revol (2004) Multiple Precision Interval Packages: Comparing Different Approaches. *Num. Software with Result Verification, LNCS* 2991:64–90

Hantos, Z., B. Daróczy, T. Csendes, B. Suki, and S. Nagy (1990) Modeling of Low-frequency Pulmonary Impedance in the Dog, *J. of Applied Physiology* 68: 849–860

Kanno, M., K. Yokoyama, H. Anai, and S. Hara (2008) Symbolic Optimization of Algebraic Fuctions. *ISSAC'08*, Hagenberg, Austria

Liberti, L., S. Cafieri, and D. Savourey (2010) The Reformulation-Optimization Software Engine. *Mathematical Software - ICMS 2010, LNCS* 6327:303–314

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

Neumaier, A. (2008) Improving interval enclosures. *Manuscript*

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

Rump, S.M. (1999) INTLAB - INTerval LABoratory. *In Developments in Reliable Computing, T. Csendes, ed, Kluwer Academic Publishers, Dordrecht*, pp. 77–104

Urbán, Cs. (2009) Simplification of nonlinear functions in Maple (in Hungarian) *Masters Thesis, University of Szeged*