MathOptimizer: A nonlinear optimization package for *Mathematica* users

Frank J. Kampas[1] and János D. Pintér[2]

1: WAM Systems, Inc, 600 W Germantown Pike, Plymouth Meeting, PA 19462 USA
fkampas@msn.com
2: Özyeğin University, Istanbul, 34662 Turkey, and PCS, Inc., Canada
janos.pinter@ozyegin.edu.tr, www.pinterconsulting.com

Technical Report, Özyeğin University, Istanbul
Submitted for publication: November 2009

Abstract

Mathematica is an advanced software system that enables symbolic computing, numerics, program code development, model visualization and professional documentation in a unified framework. Our *MathOptimizer* software package serves to solve global and local optimization models developed using *Mathematica*. We introduce *MathOptimizer*'s key features and discuss its usage options that support a range of operational modes. The numerical capabilities of the package are illustrated by simple and more advanced examples, pointing towards a broad range of potential applications.

MSC: 90C30; 65K05

Keywords: *Mathematica*; *MathOptimizer*; Global and local optimization; Model development and solution; Illustrative examples and applications.

---

1. Introduction

Mathematica, by Wolfram Research (www.wolfram.com), is a state-of-the-art software package for scientific and technical computing. One of the great advantages of using *Mathematic*a is that the entire application development process – model development, computing, visualization and professional level documentation – can be presented incrementally in an interactive *Mathematica* notebook document (or in a set of such documents if preferred). *Mathematica* notebooks, being ordinary text files, are portable across all major hardware and operating system platforms: this feature enables convenient information exchange within a business, research, or academic-educational context.

MathOptimizer is a global-local optimization software package for solving constrained nonlinear optimization problems formulated in the *Mathematica* environment. In this work we shall consider the following (terse, but general) nonlinear optimization model form:

$$\text{minimize } f(x) \text{ subject to the constraints } x \in D := \{x: g(x) \le 0, xl \le x \le xu\}. \quad (1.1)$$

In (1.1) $x \in \boldsymbol{R}^n$ is an n-dimensional real vector, f: $\boldsymbol{R}^n \to \boldsymbol{R}$ is the objective function; g: $\boldsymbol{R}^n \to \boldsymbol{R}^m$ is an m-component real-valued constraint vector function. Correspondingly, in the model formulation shown

above $0 \in \boldsymbol{R}^m$, and $[xl,xu]$ defines an n-dimensional interval bound for x, all inequalities being properly interpreted component-wise.

We shall assume that the components of the vector bounds xl and xu are finite, D is non-empty, and that the model functions f and g (the latter component-wise) are continuous. The existence of the global solution (set) is obviously guaranteed by these conditions. At the same time, the symbolic solution of many instances of model (1.1) is impossible, and such models can also be difficult to solve numerically. Significant difficulties can be caused by the − possible or verifiable − multimodality of f, and/or by the possibly complicated − non-convex, perhaps even disjoint − feasible set D, implicitly determined by the functions g.

Without going into technical details, let us remark that all well-posed, but seemingly more general (finite-dimensional, continuous) constrained optimization models can be brought to the terse canonical model form (1.1) by elementary or more advanced transformations. Although this also includes the transformation of combinatorial optimization models to the form (1.1), the focus of the present work is on nonlinear optimization with continuous decision variables.

MathOptimizer is aimed at finding the globally or locally optimal solution(s) of nonlinear models that could have a multitude of such optima. Since *MathOptimizer* is a native *Mathematica* package, optimization models can be developed exploiting the significant repertoire of *Mathematica* features and functionality.

To address the general model-type stated by (1.1), *MathOptimizer* can be used in a variety of operational modes, controlled by option settings. In this article, we first review the technical background of *MathOptimizer*. This is followed by a detailed description of *MathOptimizer*'s options, illustrated by solving simple to more advanced optimization problems.

We assume that the Reader is familiar with – or that s/he will understand from our article – the essential concepts and usage of *Mathematica* and *MathOptimizer* for the purposes of nonlinear optimization. All *Mathematica* input commands (set using **`Courier Bold`** fonts) are explained and illustrated in sufficient detail, and we provide references for technical details not discussed here.

MathOptimizer will work across all hardware and operating system platforms for which a current *Mathematica* implementation is available from Wolfram Research. At the time of this writing (2009) the current *Mathematica* version is 7.0.1. *MathOptimizer* has been originally developed and tested using *Mathematica* 4.0 (in 2002). Since then it has been updated and significantly revised: our article discusses the current version that has numerous added features compared to earlier versions. The test results and optional timings reported below were obtained using our "average capability" personal computers. Most of the examples presented here can be solved well within a second, unless explicitly noted otherwise.

For technical background and details not discussed here, we refer to the extensive topical literature related to nonlinear optimization and to *Mathematica*: here only a few illustrative references will be given. Classical (local) nonlinear optimization is discussed e.g. by Bertsekas [1], Boyd and Vandenberghe [2], Hillier and Lieberman [4], while the subject of global optimization is discussed e.g. by Horst and Pardalos [5], Pardalos and Romeijn [9], and Pintér [10,11,12]. *Mathematica* itself is described by its standard reference Wolfram [17], as well as e.g. by Gaylord, Kamin and Wellin [3], Maeder [8], Trott [14], and Wagner [15]. Finally, *MathOptimizer* is described in detail by its technical documentation [7].

2. *MathOptimizer* Installation

The current *MathOptimizer* installation file system consists of the integrated global-local solver package (file MO.m), and its documentation (file UserGuide.nb). MO.m is intended for loading into *Mathematica* using the standard **`Needs`** command. To guarantee this − assuming that *Mathematica* is installed in the directory C:\Mathematica\7.0 − MO.m is to be placed in C:\Mathematica\7.0\AddOns\Applications\MathOptimizer. Following *Mathematica*'s earlier convention, the *MathOptimizer* documentation can be put into the C:\...\MathOptimizer\Documentation\English subdirectory. (This location has supported the direct invocation of the User Guide through *Mathematica*'s Help system, for earlier versions of the software. Now the User Guide can be simply opened as a *Mathematica* notebook document whenever needed.)

Based on the recommended standard installation directory structure, *MathOptimizer* can be directly invoked for use by the following *Mathematica* command:

```
Needs["MathOptimizer`MO`"]
```

Note that, in order to follow *Mathematica*'s syntax, we will not end input commands and outputs by the period (.) symbol.

The file MO.m contains the Optimize package that integrates all *MathOptimizer* solvers and their options. The functionality of Optimize can be queried by the command

```
?Optimize
```

In reply, **Optimize** returns the following *Mathematica* output:

Optimize[objective, constraints_List, varswithbounds_List, options___] minimizes the given objective function under a given set (list) of constraint functions, and given variable bounds. The varswithbounds list is defined in the form {{var1, var1 lower bound, var1 upper bound}, {var2, var2 lower bound, var2 upper bound}...}.

Type Options[Optimize] to see the options and their default settings.

Type ?optionname for more information on each individual option.

(All *Mathematica* output will be typeset in Times New Roman fonts, occasionally and slightly formatted for the purposes of this article.)

As shown above, the **Optimize** function requires three key arguments. The first argument is the model objective function, the second is the possibly empty list of model constraints, and the third is the list of decision variables with corresponding bounds. A number of options can be selected and added to the basic input list of Optimize: we will discuss these options later on.

3. Using *MathOptimizer*: A Model Development Template

Our first example introduces a conceptual model development template. Following the general model form (1.1) and the corresponding key input information requirements of *MathOptimizer*, we can define standard symbols for the model components:

varswithbounds	the decision variables with given lower and upper bounds; nominal values (representing an initial solution guess) can be optionally added
objective	the model objective function that is minimized by default
constraints	the model constraints: these can be given in ≤0, =0, or ≥0 format.

For illustration, let us consider the model

$$\begin{aligned} &\text{minimize } (x_1^2 - 2\,x_2)^2 + (x_1 - 2)^2 \\ &\sin(x_1^2 - 3\,x_1 x_2) = 0 \\ &x_1^4 + 5\,x_2^3 - 40 \le 0 \\ &-5 \le x_1 \le 3 \\ &-5 \le x_2 \le 4 \end{aligned} \tag{3.1}$$

In *Mathematica*, lists of objects are denoted by entries placed between curly brackets **{}**. The list of **Optimize**'s input arguments corresponding to (3.1) is

```
varswithbounds = {{x1,-5, 3},{x2,-5,4}}
objective = (x1^2-2*x2)^2+(x1-2)^2
constraints = {Sin[x1^2-3*x1*x2]==0,x1^4+5*x2^3-40≤0}
```

Now we can call *MathOptimizer* to solve (3.1) by the command

`Optimize[objective,constraints,varswithbounds]`

The result returned by the package (immediately following the **`Optimize`** call shown above) is the following *Mathematica* list:

{0.0166185, {x1→1.87448, x2→1.74215}, Maximal Constraint Violation→1.80328×10^{-10}}.

The first entry of the return list is the numerical global optimum estimate found f* ~ 0.0166185; the second entry shows the elements of the corresponding solution vector x_1* ~ 1.87448, x_2* ~ 1.74215; and the last entry is the maximal constraint violation (MCV) value ~ $1.80328\cdot10^{-10}$ at (x_1*, x_2*). The MCV value refers to the violation of the general constraints. (The box constraints must always be satisfied by assumption except when deliberately relaxed: we will discuss this point later on.)

Let us note that numerical values can be displayed with higher accuracy if needed, using standard *Mathematica* functionality: the built-in function **`N[expression,nd]`** returns (more precisely, attempts to return) the numerical value of an arbitrary suitable *Mathematica* **`expression`** with **`nd`**-digit precision.

In spite of its small size, model (3.1) is not trivial. For comparison, we have solved the same model using the built-in *Mathematica* function **`NMinimize`** that serves for numerical nonlinear optimization. The solution found by **`NMinimize`** applying its default settings is not as good as *MathOptimizer*'s solution: see below. (Notice that **`NMinimize`** does not return a value similar to the MCV indicator of *MathOptimizer*: hence, we do not get direct feedback regarding the level of feasibility of the solution found.)

{1.03074, {x1→1.13774, x2→0.379247}}

Of course, we will not claim that this is always the case. However, our example illustrates a practically important point: using different algorithms and corresponding software implementations to solve difficult numerical problems can be useful.

4. *MathOptimizer* Options

4.1. List of Options

The combined global-local search approach implemented in *MathOptimizer* theoretically guarantees stochastic convergence to the global solution (set): consult Pintér [10] for the theoretical background. *MathOptimizer*'s default option settings are frequently – but not always – sufficient to find the (numerical) global solution. Generally speaking, in "relatively easy" optimization problems one should always obtain the same solution, except perhaps small numerical differences, even when reasonably changing some *MathOptimizer* options. In "more difficult" models – that are quite typical in the global optimization context – it often makes sense to change one or several of the options, to possibly explore alternative solutions. Although a universal recipe for option settings cannot be expected, a prudent selection of options can lead to improved numerical solutions in difficult models, with a moderate amount of experimentation.

The list of *MathOptimizer* options is queried by the command **`Options[Optimize]`**. This leads to the following return value list, formatted for better readability:

{PenaltyMultiplier→1,
MultiStarts→10,
Samples→1000,
RandomSeed→0,
SamplingMethod→Random,
ListableExpressions→True,

LocalSearches→4,
ShowGlobalResults→False,
ShowDetailedGlobalResults→False,
ShowLocalResults→False,
ShowDetailedLocalResults→False,
LocalSearchMethod→FindMinimum,
BoundedLocalSearch→True,
RepeatedLocalSearches→True}

This list displays the *MathOptimizer* option names and their assigned default settings. The option names follow the recommended *Mathematica* style for function names and similar option symbols. All default option settings can be changed by using standard *Mathematica* conventions.

In this article, it would be excessive to describe in detail each of these options. Instead, we offer a short commentary related to each of them, and briefly illustrate some of the key options by simple examples. For numerous further examples, we refer again to [7].

PenaltyMultiplier

PenaltyMultiplier is the penalty factor p used in the definition of the so-called merit function. If the constraint vectors are separated into equalities $g(x)=0$ and inequalities $h(x)\leq 0$, then the merit function is defined as $f(x) + p\cdot(\|g(x)\| + \|\max[h(x),0]\|)$; here the absolute value (l1)-norm is used.

Here is an illustrative example, with a slight entertainment flavour: we attempt to find a Pythagorean triplet by using global optimization.

```
Optimize[Abs[x^2+y^2-z^2],{Sin[πx]==0,Sin[πy]==0,Sin[πz]==0},
    {{x,1,5},{y,1,5},{z,1,5}},PenaltyMultiplier→3]
```

{5.86198*10^-12, {x→3., y→4., z→5.}, Maximal Constraint Violation→1.84269*10^-12}

MultiStarts

If used in its global and local search modes, then *MathOptimizer* first performs MultiStarts (number of) global searches in a stochastic multistart based sampling framework, and then it performs local searches starting from the best result or several of the best results found by the global searches. The default setting of MultiStarts is 10; it may be advisable to increase this number for higher-dimensional or otherwise more difficult models. A suggested heuristic setting, in line with (1.1), is $m+1+n$, where $m+1$ is number of model functions, and n is the number of decision variables. It is important to point out that MultiStarts→0 leads to local search only, without a preceding global search phase. This local search is started from a given initial point or − if such information is absent, then − from the midpoint of the search range defined by the lower and upper bounds.

The next example illustrates that in difficult global optimization problems the quality of solution could improve by increasing the number of global scope multistarts. The example chosen is Trefethen's Problem 4: for details, consult e.g. Weisstein [16]. We shall consider the objective function

```
objective=Exp[Sin[50*x]]+Sin[60*Exp[y]]+Sin[70*Sin[x]]+
          Sin[Sin[80*y]]-Sin[10*(x+y)]+(x^2+y^2)/4;
```

This function is highly multimodal as shown by Figure 1, and hence it often has served as a test challenge for global solvers in recent years.

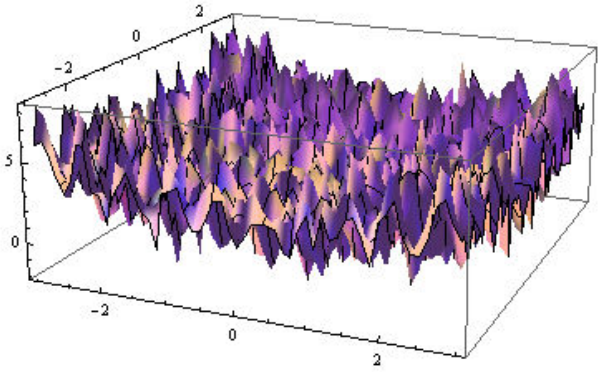

Figure 1. The objective function in Trefethen's Problem 4, for -3≤x≤3, -3≤y≤3.

The next command attempts to minimize this function over the interval region -3≤x≤3, -3≤y≤3, from a given initial point x=1, y=2. Notice the empty list **{}** in the **Optimize** call that indicates the absence of general constraints (the box constraints always have to be defined).

```
Optimize[objective,{},{{x,-3,1,3},{y,-3,2,3}},MultiStarts→0]
{2.21767,{x→0.944705,y→2.00307},Maximal Constraint Violation→0}
```

The solution found is only one of the great many local optima, see Figure 1. By contrast, even a small number (3) of global scope multistart phases – each followed by local search from the best point found in the given global search phase – leads to the global solution (that is known for this well-studied test problem).

```
Optimize[objective,{},{{x,-3,1,3},{y,-3,2,3}},MultiStarts→3]
{-3.30687,{x→-0.0244031,y→0.210612},Maximal Constraint Violation→0}
```

In fact, the solution directly retrieved from *MathOptimizer*'s result on the next *Mathematica* input line

```
{-3.306868647475235,{x→-0.024403079694365632,y→0.21061242715536216}
```

meets the 10-digit precision requirement of the originally posted challenge.

Samples

The option Samples determines the total number of sample points in each multistart iteration. Samples can be set to any non-negative integer value, at least in principle. It is recommended to set its value as an increasing function of the model size, considering both the number of variables and constraints. The next example shows that a moderate amount of multistart based sampling (global scope search phases followed by corresponding local searches) suffices to handle Trefethen's Problem 4:

```
Optimize[objective,{},{{x,-3,-2,3},{y,-3,-1,3}},
MultiStarts→10, Samples→100]
{-3.30687,{x→-0.0244031,y→0.210612},Maximal Constraint Violation→0}
```

RandomSeed

The default setting for RandomSeed is 0. Changing the random seed to an arbitrary positive integer may lead to a numerically different solution, and thus the option can serve as an easy mechanism to generate a sequence of alternative solutions in difficult multimodal problems.

To illustrate this point, consider the following optimization problem that has two global solutions; the corresponding optimized decision variables have opposite signs. By using different random seeds in an automated sequence of four optimization runs, both global solutions can be obtained. The **Table** and **TableForm** commands jointly lead to reporting the results in a tabular form. (Again, the output is slightly formatted for better readability.)

```
Table[Optimize[x^2-y^2,{Cos[x-y]≥0.5},{{x,-5,5},{y,-5,5}},
RandomSeed → i], {i,4}]//TableForm
```

{-24.9443, {x→0.235988,y→-5.}, Maximal Constraint Violation→0.},
{-24.9443, {x→0.235988,y→-5.}, Maximal Constraint Violation→0.},
{-24.9443, {x→-0.235988,y→5.}, Maximal Constraint Violation→0.},
{-24.9443, {x→-0.235988,y→5.}, Maximal Constraint Violation→0.}

SamplingMethod

The default setting (Random) leads to generating a sequence of pseudo-random global search points. The alternative low discrepancy generator (QuasiRandom) setting can produce a more uniform coverage of the box search region (with a greater computational effort), especially when the sample size is rather limited. Hence, changing the sampling method may lead to numerically different solutions in difficult multimodal problems. Trefethen's Problem 4 with a limited search effort is used for illustration:

```
objective=Exp[Sin[50*x]]+Sin[60*Exp[y]]+Sin[70*Sin[x]]+
          Sin[Sin[80*y]]-Sin[10*(x+y)]+(x^2+y^2)/4;

Optimize[objective,{},{{x,-3,1,3},{y,-3,2,3}},
MultiStarts→1,Samples→500,SamplingMethod→"Random"]
```

{-2.78312,{x→-0.0254078,y→-0.339479},Maximal Constraint Violation→0}

```
Optimize[objective,{},{{x,-3,1,3},{y,-3,2,3}},
MultiStarts→1,Samples→500,SamplingMethod→"QuasiRandom"]
```

{-3.06263,{x→0.34493,y→0.368019},Maximal Constraint Violation→0}

ListableExpressions

MathOptimizer takes advantage of the fact that many mathematical functions such as **x²** or **Sin[x]** are listable. Listability means that such functions automatically thread over array arguments. If the optimization model is defined by listable functions, then one should use the default setting True for this option: in general this leads to somewhat faster program runs. If the objective function or constraints include expressions that are not listable (this can be verified within *Mathematica*), then ListableExpressions should be set to False.

LocalSearches

Following the global search phase, a local search is performed starting from the best LocalSearches number of points (those with the lowest merit function values). In general, setting larger values in this option could improve the solution quality in difficult optimization problems.

ShowGlobalResults

MathOptimizer performs a number of global searches in a multistart framework, and then uses a local search to improve the result of the best global search. Although theoretically this approach guarantees global convergence, in numerical practice the best result from the global search phase may not give the best final result after a local search is performed. ShowGlobalResults enables the user to see the merit function values returned by all the global searches, in order to decide if local searches should be performed starting from other global search results. In other words, setting this parameter to True (default is False) can help to explore alternative solutions based on a range of global search based initial solution "guesses".

ShowDetailedGlobalResults

If this options is set to True, then a sorted list of the objective function values and associated variable values found by the global searches is returned, ordered by their quality. This list also includes the (additional) results evaluated for the nominal variable values.

This option can be useful, when the model function evaluations are expensive, and we would like to get some useful information about the best solutions found in a resource-limited run. Another good reason to use it can be when the local search could run into numerical difficulties, due e.g. to local non-differentiability of some model functions. In such cases, one can just do a (limited or detailed) global search and omit the local search option, by setting the options LocalSearches to 0, and RepeatedLocalSearches to False. (The option RepeatedLocalSearches will be discussed shortly.)

For illustration, we use again Trefethen's Problem 4:

```
objective=Exp[Sin[50*x]]+Sin[60*Exp[y]]+Sin[70*Sin[x]]+
Sin[Sin[80*y]]-Sin[10*(x+y)]+(x^2+y^2)/4;

Optimize[objective,{},{{x,-3,3},{y,-3,3}},
MultiStarts→5,Samples→200,ShowDetailedGlobalResults→True,
LocalSearches→0,RepeatedLocalSearches→False]
```

```
{{Detailed Global Search Results→
{
{-2.40972,{0.341628,0.293711}},
{-2.17049,{-1.16677,0.209861}},
{-1.61935,{1.35706,-1.18785}},
{-1.32988,{-0.947472,-0.0914035}},
{-1.30179,{1.07655,-0.249177}},
{0.695189,{0,0}}
}}}
```

ShowLocalResults

If this option is set to True (used in conjunction with ShowGlobalResults→True), then a sorted list of the local search results (objective function values only) is returned, in the same order as the corresponding global results. The best solution found is always fully displayed, however.

ShowDetailedLocalResults

If this option is set to True, then a sorted list of the detailed local search results (objective function values and variable values) is returned, in the same order as the corresponding global results.

LocalSearchMethod

This option selects the optimization method used in the local search phase. The default method is *Mathematica*'s built-in **FindMinimum** function. The alternative choice is AugmentedLagrangian that

invokes an implementation of the augmented Lagrangian optimization method. It may be advisable to try both local searches in solving difficult models.

```
Optimize[x+y+z,{x^2+y^2+z^2==1,x(y+1)==1/2},{{x,-2,2},{y,-2,2},
{z,-2,2}}, LocalSearchMethod→"FindMinimum"]
```
{-1.75583,{x→-0.515919,y→-1.2202,z→-0.019716},
Maximal Constraint Violation→0.755445}

```
Optimize[x+y+z,{x^2+y^2+z^2==1,x(y+1)==1/2},{{x,-2,2},{y,-2,2},
{z,-2,2}}, LocalSearchMethod→" AugmentedLagrangian"]
```
{-0.372761,{x→0.532935,y→-0.0618001,z→-0.843896},
MaximalConstraintViolation→4.25963×10^{-8}}

BoundedLocalSearch

The global search phase (by its algorithm design) always satisfies the explicitly given variable bounds. However, the local search could leave the feasible region – by violating the preset variable bounds – if BoundedLocalSearch is set to False. Applying this option may lead to better results, if the bounds were perhaps misspecified. The next example illustrates this possibility.

```
Optimize[(x-6)^2+(y-2)^2,{},{{x,-5,5},{y,-5,5}}]
```
{1.,{x→5.,y→2.},Maximal Constraint Violation→0}

```
Optimize[(x-6)^2+(y-2)^2,{},{{x,-5,5},{y,-5,5}},
BoundedLocalSearch→False]
```
{0.,{x→6.,y→2.},Maximal Constraint Violation→0}

RepeatedLocalSearches

The default setting is True, which means that local search is performed at several times during the global search phase. This setting typically results in faster optimization and better overall results, but may not do so for all problems. As always, it could make sense to test both option settings in solving difficult models.

5. Optimization Models with (Arbitrary) Continuous *Mathematica* Functions

MathOptimizer can handle a very broad range of user-defined model functions, including many of *Mathematica*'s built-in functions and their various (programmed) extensions. In principle, one could use "all" continuous *Mathematica* functions – if suitable – as model components. We will illustrate this key point by relatively simple examples in the next two subsections.

A small, but important technical consideration is to define the model functions so that they are only evaluated numerically, and for general safety (if in doubt, then) to set the option ListableExpressions to False.

5.1. Example 1: Optimizing a Parametric Integral Expression

In the first example we wish to find the parameter $0 \le a \le 3$ in the parametric integral

$$\int_0^{\pi} \cos(ax^2)\sin(ax)dx \tag{5.1}$$

so that the corresponding integral is minimal. This problem can be directly handled by the next two *Mathematica* and *MathOptimizer* commands: the first one defines the objective function, and second one solves the resulting optimization problem.

```
f[a_?NumberQ]:= NIntegrate[Cos[a*x^2]*Sin[a*x],{x,0,Pi}]

Optimize[f[a],{},{{a,0,3}},ListableExpressions→False,
MultiStarts→1]
```
{-0.293604,{a→0.465361},Maximal Constraint Violation→0}}

Due to the embedded numerical evaluation of the integral – for algorithmically selected values of the parameter *a* – the runtime is about two minutes. *MathOptimizer* has found the correct solution as confirmed by Figure 2 below.

```
Plot[f[a],{a,0,3}]
```

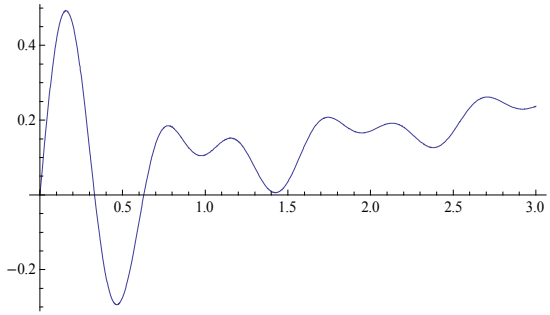

Figure 2. The objective function in the optimization problem induced by (5.1).

Let us note that the built-in function **NMinimize** finds the suboptimal solution a ~ 1.42484 with the corresponding local optimum value f[a] ~ 0.00601389: see Figure 2. Again, this example can serve as a motivation to use several alternative high-quality solver tools to handle difficult nonlinear optimization models – whenever this is possible.

5.2. Example 2: Minimizing Composite Bessel Functions

The Bessel function **BesselJ[*n*,*z*]** satisfies the parametric differential equation

$z^2 y'' + z y' + (z^2 - n^2) y = 0$

Due to their oscillating behaviour, Bessel functions can be used also in global optimization tests. For example, consider the optimization problem formulated, solved, and visualized below.

```
objective = BesselJ[3,x]+BesselJ[4,x];

Optimize[objective,{},{{x,-50,100}},MultiStarts→1]
```
{-0.473448,{x→8.62573},Maximal Constraint Violation→0}

This is the correct numerical solution, see Figure 3:

```
Plot[objective,{x,-50,100}, PlotRange→All]
```

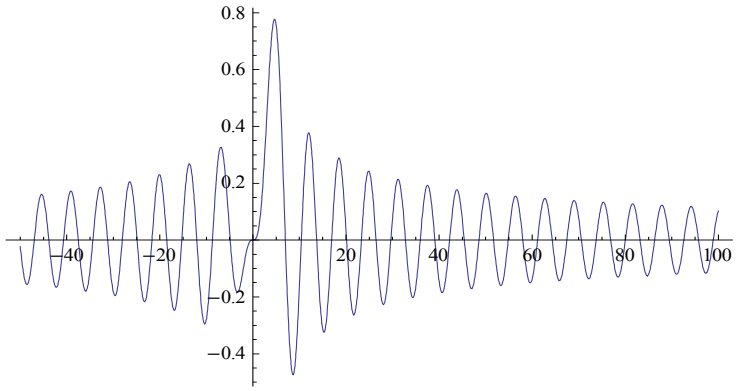

Figure 3. The objective function BesselJ[3,x]+BesselJ[4,x], -50≤x≤100.

It is easy to extend this example to higher dimensions, and thereby to create further test models with a known solution. Next, we solve and visualize a direct two-dimensional model extension:

```
objective2=BesselJ[3,x]+BesselJ[4,x]+BesselJ[3,y]+BesselJ[4,y];

Optimize[objective2,{},{{x,-50,100},{y,-50,100}},MultiStarts→1]
```

{-0.946897,{x→8.62573,y→8.62573},Maximal Constraint Violation→0}

```
Plot3D[objective2,{x,-50,100},{y,-50,100},PlotPoints→100,
PlotRange→All]
```

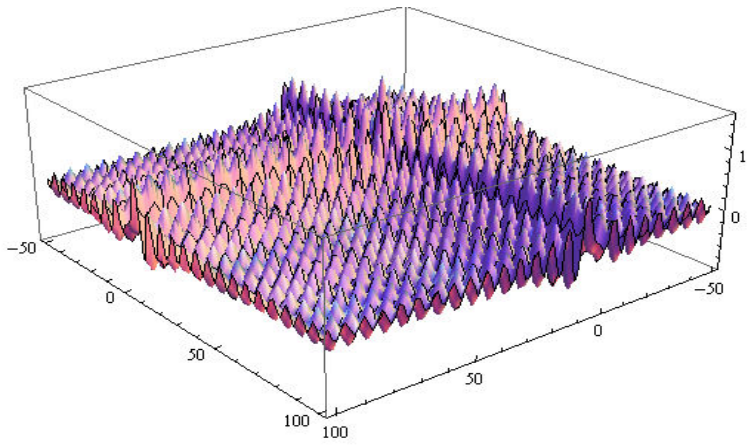

Figure 4. The objective function BesselJ[3,x]+BesselJ[4,x]+BesselJ[3,y]+BesselJ[4,y], -50≤x≤100, -50≤y≤100.

Based on the result of the one-dimensional model version, we can directly conclude that *MathOptimizer* has found the correct (numerical global) solution: Figure 4 shows the many local optima in this test problem.

6. Optimization of Non-uniform Circle Packings

Optimized object packings (configurations or arrangements) are important in various engineering and scientific fields such as numerical integration, potential energy models, experimental design, and others. Next, we will illustrate *MathOptimizer* performance by solving circle packing problems. Specifically, we consider the following general problem-type: given an arbitrary collection of circles,

find the smallest circumscribing circle that includes all circles in a "tight" non-overlapping arrangement. This and similar packing problems have been studied recently by a number of researchers (including the authors). In general, the global solution in such problems is often unknown: hence only putative optima are published, except for special cases and/or for very small model instances.

In our illustrative numerical study, we have taken circles with radii $r_i=1/i^{1/2}$, for i=1,…,i_{max}; here i_{max} determines the problem size. We will omit the discussion of the related *Mathematica* code development [6,7], and only summarize here some of our illustrative results obtained by using *MathOptimizer*, for 5,10,15, and 20 circles. Let us remark that for i_{max}=20 the corresponding optimization model includes 190 non-convex constraints (that express the pairwise "no overlap" relation): thus it is far from trivial. We also emphasize that in our numerical experiments we do not exploit any insight into the possible structure of such packings: *MathOptimizer* is used as a "blind" (completely automatic) solver tool. The results are summarized in Table 1.

Number of circles	Optimum value found	Runtime in seconds
5	1.75155	4.212
10	1.94642	21.466
15	2.04702	54.694
20	2.13901	206.889

Table 1. Non-uniform circle packings: illustrative results.

For illustration, the 20-circle configuration found by *MathOptimizer* is shown below.

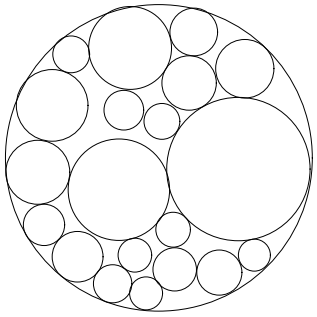

Figure 5. The 20-circle configuration found by *MathOptimizer*.

Although the runtimes are increasing (as it can be expected), numerical performance seems to scale reasonably well as the model size increases – at least for the problem-instances considered.

For comparison, we also ran **`NMinimize`** on these problems. In our tests, both packages found the same numerical solution for 5 circles, while in the larger test model instances considered *MathOptimizer* surpassed the quality of the solution (circumscribed circle radius) found by **`NMinimize`** by up to 2 percent.

7. Conclusions

The *MathOptimizer* package serves to solve nonlinear optimization models formulated using *Mathematica*. *Mathematica*'s advanced model development capabilities in combination with *MathOptimizer* offer a powerful platform for nonlinear systems modeling and optimization. Our article illustrates some of the advantages of using a single software platform platform for application development; many further examples are discussed e.g. in [6,7,13].

Let us mention finally that *MathOptimizer* has been used by industry, research organizations and in academia to solve optimization problems since 2002. We expect that the new – more efficient and flexible – package version described here will also assist many researchers in their work.

Acknowledgements

The initial *MathOptimizer* development work (by JDP) benefited from advice, books and software, and a visiting scholar grant provided by Wolfram Research. The first *MathOptimizer* software development project was also supported by the National Research Council of Canada, and by a subsequent research contract from DRDC, Dartmouth, NS, Canada. The valuable advice and suggestions received from Dr. Christopher Purcell (DRDC) throughout this early development work are also gratefully acknowledged.

References

[1] Bertsekas, D.P. (1999) *Nonlinear Programming*. (2nd Edition) Athena Scientific, Cambridge, MA.
[2] Boyd, S. and Vandenberghe, L. (2004) *Convex Optimization.* Cambridge University Press, New York.
[3] Gaylord, R.J., Kamin, S.N. and Wellin, P.R. (1996) *An Introduction to Programming with Mathematica*. (2nd Edition) Springer-Verlag, New York / Berlin / Heidelberg.
[4] Hillier, F.S. and Lieberman, G.J. (2005) *Introduction to Operations Research*. (8th edn.) McGraw-Hill, New York.
[5] Horst, R. and Pardalos, P.M., Eds. (1995) *Handbook of Global Optimization, Vol. 1*. Kluwer Academic Publishers, Dordrecht.
[6] Kampas, F.J. and Pintér, J.D. (2006) Configuration analysis and design by using optimization tools in *Mathematica*. *The Mathematica Journal* 10 (1), 128-154.
[7] Kampas, F.J. and Pintér, J.D. (2009) *MathOptimizer − An advanced nonlinear optimization program system for Mathematica users. User Guide.* (Current edition) Published and distributed by PCS, Inc.
[8] Maeder, R.E. (1997) *Programming in Mathematica*. (3rd Edition) Addison-Wesley, Reading, MA.
[9] Pardalos, P.M. and Romeijn, H.E., Eds. (2002) *Handbook of Global Optimization, Vol. 2.* Kluwer Academic Publishers, Dordrecht.
[10] Pintér, J.D. (1996) Global *Optimization in Action (Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications)*. Kluwer Academic Publishers, Dordrecht.
[11] Pintér, J.D. (2002) Global Optimization: Software, Tests and Applications. Chapter 15 (pp. 515-569) in: Pardalos and Romeijn, Eds., *Handbook of Global Optimization, Vol. 2.* Kluwer Academic Publishers, Dordrecht.
[12] Pintér, J.D., Ed. (2006) *Global Optimization: Scientific and Engineering Case Studies.* Springer Science + Business Media, New York.
[13] Pintér, J.D. and Kampas, F.J. (2005) Nonlinear optimization in *Mathematica* with *MathOptimizer Professional*. *Mathematica in Education and Research* 10 (2), 1-18.
[14] Trott, M. (2004, 2006) *The Mathematica GuideBooks, Volumes 1-4: Programming, Graphics, Numerics, Symbolics*. Springer Science + Business Media, New York.
[15] Wagner, D.B. (1996) *Power Programming with Mathematica − The Kernel*. McGraw-Hill, New York.
[16] Weisstein, Eric W. (2009) "Hundred-Dollar, Hundred-Digit Challenge Problems." From MathWorld − A Wolfram Web Resource, http://mathworld.wolfram.com/Hundred-DollarHundred-DigitChallengeProblems.html.
[17] Wolfram, S. (2005) *The Mathematica Book*. (5th Edition) Wolfram Media, Champaign, IL, and Cambridge University Press, Cambridge, UK.