Configuration Analysis and Design by Using Optimization Tools in Mathematica

Frank J. Kampas
János D. Pintér

In engineering, economic, and scientific studies, decisions are frequently modeled by applying optimization concepts and techniques. This article discusses global optimization in multietremal models and tools to handle such models in Mathematica. Since we assume that not all readers are familiar with optimization models and methods, a general modeling framework is presented. We also review several built-in Mathematica optimization functions and then introduce MathOptimizer, an application package for continuous nonlinear (convex and global) optimization. To illustrate the usage of MathOptimizer, several configuration analysis and design models are formulated and solved. We also provide some comparative notes related to current Mathematica optimization functionality (namely, the function NMinimize) and to the recently introduced MathOptimizer Professional package.

Introduction

Quantitative decisions related to various engineering, economic, and scientific investigations are frequently modeled by applying optimization concepts and tools. In mathematical terminology, the decision maker or modeler wants to find the “absolutely best” decision vector that corresponds to the minimum (or maximum) of a suitable objective function, while it satisfies a given collection of feasibility constraint functions. The objective function expresses an overall—scalar or scalarized—performance measure, such as profit, utility, loss, risk, or error. The model constraints originate from physical, technical, economic, or possibly some other, considerations.

A large number of such models naturally belong to the realm of “traditional” continuous optimization, notably, linear and convex nonlinear programming.
These subjects are covered by most operations research/management science texts [1–21].

There exist numerous practically important optimization problems in which the phenomena and processes modeled are highly nonlinear and/or are evaluated computationally. In many cases, the necessary structural (convexity) requirements of traditional optimization are not satisfied or may not be simply verifiable. Nonconvex objective functions often possess a multitude of local optima; nonconvex feasible sets also may lead to a similar situation, even for convex objective functions. The subject of global optimization is the theory and application of seeking the “absolutely best” solution in models that have a number of local optima. For discussions of highly nonlinear models and of global optimization, consult, for example, [22–44].

Global optimization (GO) models and strategies have been analyzed since the 1950s. For example, think of the origins of “naturally inspired” genetic and evolutionary searches, simulated annealing, or random search methods. To our best knowledge, the first systematic collection of English GO papers is found in the volumes edited by Dixon and Szegö [45]. Around the same time, several topical books in Russian also became available. See, for example, the related review and references in [35].

As of 2004, there exist well over one hundred books, as well as many thousands of research articles, that discuss GO theory, models, methods, software, and applications. In addition to the works listed earlier, we refer the reader to the Handbook of Global Optimization volumes edited by Horst and Pardalos [46] and by Pardalos and Romeijn [47]. The Journal of Global Optimization, which has been published since 1991, is devoted entirely to the subject.

The Continuous Global Optimization Model

For the purposes of this article, we assume that discrete optimization variables are absent and that all model functions are continuous. This leads to the corresponding specific case of the mathematical programming (MP) model, referred to as the continuous global optimization (CGO) problem. In order to introduce this model form, we use the following notations.

\[ R^n \quad n \text{-dimensional Euclidean vector space} \]

\[ x \quad \text{vector of continuous decision variables} \ (x \in R^n) \]

\[ xl, xu \quad \text{explicit, component-wise lower- and upper-bound vectors on } x \]

\[ f (x) \quad \text{objective function of the model} \]

\[ g (x) \quad \text{vector function of equality constraints} \]

\[ b (x) \quad \text{vector function of inequality constraints} \]
Applying this notation, the CGO model can be expressed as

\[
\begin{align*}
\min f(x) & \quad f : D_0 \to \mathbb{R}^1 \\
g(x) &= 0 \quad g : D_0 \to \mathbb{R}^m \\
b(x) &\leq 0 \quad b : D_0 \to \mathbb{R}^m \\
x \in D_0 = [x_l, x_u] & \quad x_l, x_u \in \mathbb{R}^n
\end{align*}
\]

The set of feasible solutions to this model is defined by

\[
D = \{g(x) = 0, \ b(x) \leq 0, \ x_l \leq x \leq x_u\}.
\]

Note that \(D\) could be empty: for example, one may think of a system of nonlinear equations \(g(x) = 0\) that has no solutions within the range specified by \(x_l \leq x \leq x_u\). In many practical models, however, \(D\) is nonempty, and this property is explicitly postulated in most theoretical analyses of optimization models.

The globally best solution of the CGO model is an \(x^* \in D\), such that \(f(x^*) \leq f(x)\) for all \(x \in D\).

The locally best solution of CGO is an \(x^l \in D\), such that \(f(x^l) \leq f(x)\) for all \(x \in D, \|x - x^l\| < \delta\). Here \(\delta > 0\) is a suitable constant defining the \(\delta\)-neighborhood of \(x^l\). (Here typically the Euclidean norm is used, but other norms could be used as well.)

Obviously, \(f(x^*) \leq f(x^l)\), and the relation becomes equality only when \(x^l\) is also a globally optimal solution.

According to the classical theorem of Weierstrass, a continuous function \(f(x)\) attains its minimum on a closed, bounded set \(D\). This fact is often used to guarantee the existence of the global solution in various special cases of the CGO model. Observe at the same time that, in its full generality, this model encompasses a large class of difficult instances.

Figure 1 illustrates the potential difficulty of continuous optimization models. Although the feasible set is just a two-dimensional interval, the objective function is highly multimodal. Hence, purely local scope search methods, as a general rule, even in much simpler cases, will not find the best solution. Note that this model is due to Trefethen, and it has been used also in our software tests: consult, for example, [38].
\begin{itemize}
\item Optimization in \textit{Mathematica}
\end{itemize}

In this section, we briefly review \texttt{NMinimize}, a built-in general numerical optimization tool. This is followed by a somewhat more detailed summary of \texttt{MathOptimizer}, an application package.

Note that in addition to \texttt{NMinimize}, there are several other optimization-related functions in \textit{Mathematica}, for example, \texttt{ConstrainedMin} and \texttt{ConstrainedMax}, which will also be discussed shortly. We entirely omit the discussion of the underlying theory and algorithms and refer the reader to \textit{The Mathematica Book} [48], the current documentation [49, 50], and the optimization references listed earlier for details.

\begin{itemize}
\item Built-in Optimization Tools
\end{itemize}

The following brief description of \texttt{NMinimize} is based on \textit{Mathematica}'s online help system.

\texttt{NMinimize}(\texttt{\{f, cons\}, \texttt{vars}}) numerically finds (approximates) the global minimum of an arbitrary linear or nonlinear function \texttt{f} subject to the arbitrary constraints \texttt{cons}. \texttt{NMinimize} implements several algorithms for finding global optima. According to the documentation for \texttt{NMinimize}, “The methods are flexible enough to cope with functions that are not differentiable or continuous, and are
not easily trapped by local optima. Keep in mind, however, that finding a global optimum can be arbitrarily difficult, even without constraints, and so the methods used may fail. It may frequently be useful to optimize the function several times with different starting conditions and take the best of the results.” This caveat is of obvious relevance with respect to all general numerical (global) optimization approaches.

The following simple example illustrates the usage of this function. First, load the package if necessary (in Mathematica 5, NMinimize is autoloaded).

\begin{verbatim}
In[2]:= If[$VersionNumber < 5, 
   << NumericalMath`NMinimize']
\end{verbatim}

\begin{verbatim}
In[3]:= NMinimize[
   {Sin[5 x - 3 y] + x*y + (x - y)^2, 
    -2 <= x <= 5 && -3 <= y <= 4 && x + y <= 5}, 
   {x, y}]
\end{verbatim}

\begin{verbatim}
Out[3]= {-0.909737, {x -> -0.268141, y -> 0.0383059}}
\end{verbatim}

Figure 2 shows the objective function over the interval defined by the explicit bounds. Note that the added $x + y \leq 5$ constraint cuts off a part of this box region.

\begin{verbatim}
In[4]:= Plot3D[Sin[5 x - 3 y] + x*y + (x - y)^2, 
   {x, -2, 5}, {y, -3, 4}, 
   PlotPoints -> 50, Mesh -> False, ColorFunction -> Hue]
\end{verbatim}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure2.png}
\caption{The function $\sin(5x - 3y) + xy + (x - y)^2$ in the two-dimensional “box” region $x \in [-2, 5]$, $y \in [-3, 4]$.}
\end{figure}

\textbf{MathOptimizer}

MathOptimizer [38] is a third-party native Mathematica product that enables the global and local solution of the general class of optimization problems encompassed by the CGO model form. The key features of this package are summarized next. For theoretical background, refer to [35] and the other GO textbooks listed earlier.
MathOptimizer currently consists of two core solver packages and a solver integrator package. One of these solvers is used for the typically approximate global optimization of an aggregated “merit” (exact penalty) function on the given interval range \([xl, xu]\), in the possible presence of added constraints \(g\) and \(h\). This package is based on an efficient adaptive stochastic search method, combined with a statistical bounding procedure. The latter provides an estimate of the global optimum value, based on the global sampling results.

The second solver package implements a convex optimization approach aimed at finding a solution vector that satisfies the Karush–Kuhn–Tucker local optimality conditions. This solver option can be used for “precise” local optimization, based on the initial solution produced by the global search phase or provided by the user. (Note that, at least theoretically, all CGO model functions should be smooth, in order to support the application of this solver component.)

The solver integrator package enables the individual or combined use of these two solver packages; the solvers can also be used separately. (Further solver modules are under development and will be made available.)

The MathOptimizer User Guide [38] is a Mathematica notebook that can be directly invoked through the online Help Browser. The user guide includes installation and technical notes and provides concise mathematical background information and modeling tips. In addition to a basic usage description, it also discusses a number of simple and more challenging test problems and a few detailed application examples.

MathOptimizer can be installed and used on all hardware platforms that are suitable to run Mathematica 3.0 or later. It is available for all Microsoft Windows (95 and later) operating systems, as well as all other platforms that have corresponding current Mathematica implementations. Several prominent OS examples are Linux, Macintosh (including Mac OS X), HP-UX, SGI, Solaris, and Sun (Unix flavor) platforms. MathOptimizer software installation and tests with identical file systems were successfully completed using over half a dozen different hardware platforms and operating systems (at the time of this writing, including all the aforementioned platforms).

**MathOptimizer Usage Definitions and Options**

Assuming that MathOptimizer is set up using the Mathematica standard directory structure, it can be invoked by the following statement.

```mathematica
Needs["MathOptimizer`Optimize"]
```

Here is the concise information regarding how to use Optimize.
Optimize[f, g, h, x, xinit, xlb, xub, opts] integrates the current
solver options of the MathOptimizer system of packages. These
packages together serve to find — that is, to numerically
approximate — the solution of the following general constrained
optimization problem: minimize f subject to g=0 and h( x)≤0, x is a real vector from a given finite interval range [ xlb, xub], xinit is a list of initial (nominal) values for x.
All model functions are assumed to be continuous, g and h are
lists of Mathematica expressions. MathOptimizer is based on a
combination of global and local scope search procedures: these
can be used via calling Optimize in a flexible manner, in suitable
combinations or in a standalone mode, with corresponding sets of
options. Optimize returns the solution vector, and additional
model information (as described in detail with respect to the
individual solver options). The symbolic argument opts denotes
a list of usage possibilities: please use Options[Optimize]
to display these. The current solver option packages are MS
and CNLP: for further information on these, type ?MS or ?CNLP.

We will not go into further, quite extensive, details of the package options and refer the reader to the MathOptimizer User Guide [38] for more information.
Instead, we shall illustrate the usage of the function Optimize on the last problem discussed in the preceding section.

Here are the main steps of the modeling and solution procedure. Comments set in blue serve only for explanation.

vars = {x, y}; (* decision variables *)
varnom = {0, 0}; (* nominal (initial) values of variables *)
varlb = {-2, -3}; (* lower bounds of variables *)
varub = {5, 4}; (* upper bounds of variables *)
objf = Sin[5 x - 3 y] + x*y + (x-y)^2; (* objective function, to minimize *)
eqs = {}; (* equality constraints, absent *)
ineqs = {x+y-5}; (* inequality constraints *)

Optimize[objf, eqs, ineqs, vars, varnom, varlb, varub, 
GlobalSolverMode→1, LocalSolverMode→1, ReportLevel→0]

{{-0.268141, 0.0383059}, {-0.909737, 0.}, {5.22984, 0.}, {0., 2.03692×10^-10, 0.}}

The output of MathOptimizer consists of the numerically estimated list of optimal solution components, the corresponding objective function value, the lists of the equality and inequality model function values at the solution (including an empty list of equality constraints in this example), the list of the maximal feasibility violation level at the solution, the violation level of the Karush–Kuhn–Tucker equation at the solution, and the violation level of the complementary slackness condition at the solution.
Without going into further details, let us note that the output indicates that the local optimality conditions are satisfied to a fairly high precision and that the solution agrees with the one found by `NMinimize` to at least $10^{-6}$ precision, in terms of both the optimized arguments and the optimum value.

In the next (Contact and Collision Problems) section, we shall illustrate the usage of `MathOptimizer` and `NMinimize` in the context of several nontrivial tests that point towards interesting application areas. An extensive collection of further numerical examples and illustrative applications formulated in `Mathematica` is discussed in [29, 39].

### Contact and Collision Problems

#### Introductory Notes

There are many situations in engineering analysis and design in which one would like to know whether two stationary objects are in contact or, if not, what the minimum distance is between them. Similarly, for moving objects, one may wish to know whether or not they collide, the collision time if they do, or the minimum distance if they do not.

If each object can be analytically described as a set of points satisfying certain constraints, then finding the minimum distance can be formulated as a minimization problem in which one minimizes the distance between two points, each constrained to be in one of the objects. If one is only interested in whether or not the objects are in contact, the problem can be solved by determining if there are any points that satisfy the constraints describing both objects.

#### Intersecting Stationary Objects

For a simple example of determining whether or not two stationary objects intersect, let us consider two triangles described by the following lists of constraints.

```mathematica
tri1[x_, y_] := {y >= 0, x + y <= 3/2, y - x <= 1};
tri2[x_, y_] := {y >= x/2 + 1/4, y <= -2 x + 4, y <= 3 x - 1};
```

The points satisfying one or the other list of constraints can be shown using the `InequalityPlot` function. It is easily seen that the two triangles do intersect.

```mathematica
<< Graphics`InequalityGraphics`
```
The shape of the intersection, when nonempty, can be easily produced by looking at points satisfying both lists of constraints.

We can use the Minimize function to determine some target extremal point of the intersection of the triangles. In the following example, we want to determine the location of the point with the smallest $x + y$ value (see the preceding plot).

Had the triangles not overlapped, there would have been no solution to the optimization model.

As another simple example, we can also determine the point in the intersection that has the maximal $l_1$-norm (Manhattan) distance from the origin.
Nonintersecting Stationary Objects

To see an example of nonintersecting stationary objects, we modify triangle 1.

\[ \text{tril}[x, y] = \{ y \geq 0, x + y \leq \frac{3}{4}, y - x \leq 1 \} \]

In this case, \text{Minimize} gives no solution, as there are no points that satisfy the constraints of both triangles.

\[ \text{Minimize}::\text{matt} : \text{The minimum is not attained at any point satisfying the given constraints. More...} \]

Next, we shall take a nonlinear optimization model form to analyze the current configuration of the triangles. \text{Minimize} can be used to find the shortest line joining the two triangles by minimizing the distance between two points \((x_1, y_1)\) and \((x_2, y_2)\), with the first point constrained to be in triangle 1 and the second point constrained to be in triangle 2.

\[ \text{Timing}[\text{NMinimize}[\text{Join}[\{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}\}, \text{tril}[x_1, y_1], \text{tril}[x_2, y_2]], \{x_1, y_1, x_2, y_2\}]] \]

\[ \{0.219 \text{ Second}, \{x_1 \rightarrow 0.375, x_2 \rightarrow 0.5, y_1 \rightarrow 0.375, y_2 \rightarrow 0.5\}\} \]

\text{MathOptimizer} can be used as well to handle this optimization model. Note that in order to use \text{MathOptimizer}, it is necessary to rewrite the inequalities in the form \(\text{ineq}(x, y) \leq 0\), and then give as input only a list of the left-hand side of the inequalities.

\[ \text{tril}[x, y] = \{-y, x + y - \frac{3}{4}, y - x - 1\}; \]
\[ \text{tril}[x, y] = \{\frac{x}{2} - y + \frac{1}{4}, 2x + y - 4, y - 3x + 1\}; \]

\[ \text{obj} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}; \]
\[ \text{equalCons} = \{\}; \]
\[ \text{inequalCons} = \text{Join}[@\text{tril}[x_1, y_1], \text{tril}[x_2, y_2]]; \]
\[ \text{vars} = \{x_1, y_1, x_2, y_2\}; \]
\[ \text{inits} = \{1, 2, 2, 1\}; \]
\[ \text{lbs} = \{-2, -2, -2, -2\}; \]
\[ \text{ubs} = \{2, 2, 2, 2\}; \]
Timing[
soln = Optimize[obj, equalCons, inequalCons, vars, inits, lbs, ubs]]

FindMinimum::lstol:
The line search decreased the step size to within tolerance specified by
AccuracyGoal and PrecisionGoal but was unable to find a sufficient
decrease in the function. You may need more than MachinePrecision
digits of working precision to meet these tolerances. More...

Out[28]= 1.844 Second, {0.375, 0.375, 0.5, 0.5}, 0.176777,

The shortest line can be shown superimposed on a plot of the triangles.

In[29]:= p1 = InequalityPlot[Apply[And, Thread[tri1[x, y] \[LessEqual] 0]],
{\{x\}, \{y\}, DisplayFunction \[Rule] Identity};

In[30]:= p2 = Graphics[soln[[1]] /.
{x1_, y1_, x2_, y2_} \[Rule] 
{Thickness[.01], Line[\{\{x1, y1\}, \{x2, y2\}\}]}];

In[31]:= Show[p1, p2, DisplayFunction \[Rule] $DisplayFunction]

Analysis of Moving Objects

In the following example, we shall analyze the configuration of two objects, one
of them moving along a trajectory given analytically. Let us define

\[\text{sqr}[x_, y_] = \left\{ y - \frac{1}{2}, -\frac{1}{2} - y, x - \frac{1}{2}, -\frac{1}{2} - x \right\} ; \]

Then \[\text{Thread[sqr}[x, y] \[LessEqual] 0] \] defines a square with edge 1 centered on the
origin.

Next, we define the rotation \[\text{rot}[\theta] \] that rotates two-dimensional points clockwise by angle \(\theta\).
\text{In[33]} \text{=} \text{rot[θ]}[x_, y_] = 
\{x \mapsto x \cdot \cos[θ] + y \cdot \sin[θ], y \mapsto -x \cdot \sin[θ] + y \cdot \cos[θ]\};

The function \text{disp}[dx, dy] displaces a point by \((-dx, -dy)\).

\text{In[34]} \text{=} \text{disp}[dx_, dy_][x_, y_] = \{x \mapsto x - dx, y \mapsto y - dy\};

Finally, \text{newsqr}[dx, dy, θ] takes variables \(x\) and \(y\) (specifically, denoting a point from the square), and rotates and displaces it back to the starting position so the original constraints can be applied.

\text{In[35]} \text{=} \text{newsqr}[dx_, dy_, θ_][x_, y_] :=
\{\text{sqr}[x, y] \cdot \text{rot}[θ][x, y] \cdot \text{disp}[dx, dy][x, y]\};

The other object in our illustrative example is a static disk centered on the origin with radius \(1/2\).

\text{In[36]} \text{=} \text{disk}[x_, y_] = x^2 + y^2 - \frac{1}{4};

Now, as a starting position, we center the square at \((-2, 0)\), rotated clockwise by \(\pi/10\) at \(t = 0\), and then have it move with velocity \((1.0, 0.5)\) and rotate with angular velocity \(0.3\). At any time moment \(t\), the position of the disk and the square can be plotted using the following function.

\text{In[37]} \text{=} \text{p[t_]} := \text{InequalityPlot}\left[
\text{And @@ Thread[newsqr[-2 + t, 0.5 t, 0.3 t - \frac{π}{10}][x, y] \leq 0] || \text{disk}[x, y] \leq 0, \{x\}, \{y\}, \text{PlotRange} \rightarrow \{\{-3, 1\}, \{-1, 2\}\}}\right];

Note that the function \text{p[t]} can be used to generate a series of plots, which can be animated. Here are the initial object positions.

\text{In[38]} \text{=} \text{p[0]}

Now let us find the minimum distance between the square and the disk. First, we apply \text{NMinimize} and then \text{MathOptimizer}.
\textbf{In[39]} = \texttt{obj} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2};

equalCons = \{}
inequalCons = \texttt{Join}[
\texttt{newsqr}[-2 + t, .5* t, .3 t - \frac{\pi}{10}][x_1, y_1], \{\texttt{disk}[x_2, y_2]\}];

vars = \{x_1, y_1, x_2, y_2, t\};
inits = \{0, 1, 1, 0, 1\};
lbs = \{-2, -2, -2, -2, 0\};
ubs = \{1, 1, 1, 1, 5\};

\textbf{In[40]} = \texttt{Timing[ \texttt{NMinimize}[
\texttt{Join}[
\texttt{obj}, \texttt{Thread[equalCons == 0]}, \texttt{Thread[inequalCons \leq 0]]}, \texttt{vars}]\]} = \{0.266 \text{ Second}, \{1.88758 \times 10^{-16}, \{t \rightarrow 1.17315, 
\quad x_1 \rightarrow -0.312285, x_2 \rightarrow -0.312285, y_1 \rightarrow 0.106061, y_2 \rightarrow 0.106061\}\}\}

\textbf{In[41]} = \texttt{Timing[ \texttt{Optimize}[
\texttt{obj}, \texttt{equalCons}, \texttt{inequalCons}, \texttt{vars}, \texttt{inits}, \texttt{lbs}, \texttt{ubs}]\]}

\begin{verbatim}
FindMinimum::lstol:
The line search decreased the step size to within tolerance specified by
AccuracyGoal and PrecisionGoal but was unable to find a sufficient
decrease in the function. You may need more than MachinePrecision
digits of working precision to meet these tolerances. More...
\end{verbatim}

\textbf{Out[42]} = \{3.812 \text{ Second}, 
\{\{-0.340309, 0.35257, -0.340309, 0.35257, 1.67855\}, 1.56419 \times 10^{-15}, 
\}, \{-0.97445, -0.02555, -0.610154, -0.389846, -0.00988422\}, 
\{0., 1.41421, 0.\}\}\}

The minimum distance is (numerically very close to) 0, so the objects must have collided. A graph of the solution confirms that the square and disk had collided, actually by the earlier of the two times found. (As indicated by the plot, at \texttt{tsoln} the objects already overlap.)

\textbf{In[43]} = \texttt{tsoln} = \texttt{t /. solnNM[[2]]}

\textbf{Out[44]} = 1.17315

\textbf{In[45]} = \texttt{p[tsoln]}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{collision_graph.png}
\caption{Graph of the solution showing the collision of the square and disk.}
\end{figure}
Next, let us find the collision time by minimizing $t$ with a point in the square constrained to be equal to a point in the disk. \texttt{NMinimize} and \texttt{MathOptimizer} give the same solution.

\texttt{In[50]=}
\begin{verbatim}
  obj = t;
equalCons = \{x1 - x2, y1 - y2\};
  inequalCons =
    Join[newsqr[-2 + t, .5*t, .3 t - \frac{\pi}{10}]\[x1, y1], \{disk[x2, y2]\}];
  vars = \{x1, y1, x2, y2, t\};
  inits = \{0, 0, 0, 0, 1\};
  lbs = \{-2, -2, -2, -2, 0\};
  ubs = \{1, 1, 1, 1, 1.5\};
\end{verbatim}

\texttt{In[51]=}
\begin{verbatim}
  Timing[\texttt{NMinimize[}
        Join[\{obj\}, \texttt{Thread[equalCons = 0]}, \texttt{Thread[inequalCons \leq 0]}, vars]]]
\end{verbatim}

\texttt{NMinimize::incst :}
\texttt{NMinimize was unable to generate any initial points satisfying}
\texttt{the inequality constraints \{-\frac{1}{4} + x^2 + y^2 \leq 0, \langle\langle 3\rangle\rangle,}
\texttt{-\frac{1}{2} - \langle\langle 1\rangle\rangle + (-0.5 t + 1. y2) Sin[\langle\langle 1\rangle\rangle] \leq 0\}. The initial region specified}
\texttt{may not contain any feasible points. Changing the initial region or}
\texttt{specifying explicit initial points may provide a better solution. More...}

\texttt{Out[51]=}
\begin{verbatim}
{0.172 Second, \{1.00612, \{t \to 1.00612, x1 \to -0.499962,}
  x2 \to -0.499962, y1 \to 0.00616099, y2 \to 0.00616099\}}
\end{verbatim}

\texttt{In[52]=}
\begin{verbatim}
  Timing[\texttt{FindMinimum[}
        \texttt{soln = \texttt{Optimize[}obj, \texttt{equalCons, inequalCons, vars, inits, lbs, ubs]}\texttt{]}}]
\end{verbatim}

\texttt{FindMinimum::lstol :}
\texttt{The line search decreased the step size to within tolerance specified by}
\texttt{AccuracyGoal and PrecisionGoal but was unable to find a sufficient}
\texttt{decrease in the function. You may need more than MachinePrecision}
\texttt{digits of working precision to meet these tolerances. More...}

\texttt{Out[52]=}
\begin{verbatim}
{4.89 Second, \{-0.499962, 0.00616099, -0.499962, 0.00616099, 1.00612\},
  1.00612, \{9.34605 \times 10^{-10}, -2.03102 \times 10^{-9}\},
  \{-0.990777, -0.0092231, -6.49468 \times 10^{-11}, -1., 1.87505 \times 10^{-8}\},
  \{1.87505 \times 10^{-9}, 5.75711 \times 10^{-8}, 1.63765 \times 10^{-8}\}}
\end{verbatim}

\texttt{In[53]=}
\begin{verbatim}
  tsoln = soln[[1, 5]]
\end{verbatim}

\texttt{Out[53]=}
\begin{verbatim}
1.00612
\end{verbatim}

The next plot visually confirms that the solution found is a good numerical approximation of the collision time.
Obviously, these simple illustrative examples could be extended and modified to model issues of practical interest, for example, robot or machine movements in manufacturing and ergonomic contexts. Further related examples are discussed in [29].

Non-Uniform Size Circle Packings

Introductory Notes

Point arrangements and closely related uniform circle packings within given bodies (the latter most typically being \( n \)-dimensional intervals and circles for \( n = 2, 3 \)) have been intensively studied by the mathematical community for several decades. Such models are not only of pure academic interest, but are also closely related to important problems in numerical mathematics (integration, experimental design), physics and chemistry (potential models, crystal and molecule structures), and biology (viral morphology), just to name a few application areas.

There is a considerable body of literature devoted to these subjects [51–59]. Several of these works provide extensive lists of further references.

The applicability of global optimization tools to analyze and solve instances of various point arrangement (potential) models has been demonstrated [60–62]. We emphasize here that the issue of non-uniform size circle (or more general object) packings is entirely outside of the scope of “traditional” (purely analytical) studies, while GO can still be applied to such, as well as to many other, model variants and extensions.

For illustration, in this section we use global optimization to pack different-size circles into the smallest possible circle. Since this model formulation typically has infinitely many solutions per se, we will additionally try to bring the circles as close together as possible. The primary objective is to find the embedding circle with the smallest radius; the secondary objective—bringing the circles close
together—is added as a second criterion function. A linear combination of the two objectives is used in an aggregate objective function.

Note that alternative formulations are also possible and that rotational symmetries of solutions can also be avoided by added constraints, thereby making the solution of a specific model formulation essentially unique.

Model Formulation and Numerical Examples Using NMinimize, MathOptimizer, and MathOptimizer Professional

The solution is demonstrated using three different optimizers: NMinimize, MathOptimizer, and MathOptimizer Professional. MathOptimizer Professional is a recently introduced product [39] in which the actual optimization is performed by an external executable—the Lipschitz Global Optimizer (LGO) solver engine—on a Windows dynamic link library (dll) model function. The latter is automatically generated from a model formulated in Mathematica.

Without going into details, which are outside of the scope of this article, we should mention that LGO is a global/local solver suite that has been discussed, for example, in [35–37]. The LGO engine can also be used to handle models set up in Excel [63] and GAMS [64], in addition to its C or Fortran compiler-based implementations. Let us note that [65] presents a detailed and fully reproducible comparative assessment of LGO versus several state-of-art local nonlinear solvers: this comparison is based on solving an extensive set of GAMS models, and it was done by the GAMS Development Corporation (thus it can be considered as reasonably objective). These tests demonstrate that in a significant percentage of nonlinear optimization (GAMS) models, global scope search is necessary indeed; furthermore, that LGO compares favorably to the solvers considered (CONOPT, MINOS, and SNOPT).

In the following model equations, the center of circle \( i \) is denoted by \( \{x[i], y[i]\} \) and the radius of circle \( i \) is \( r[i] \). The distance from the origin to the most distant point in a circle is given by the radius of the circle plus the distance of the center of the circle from the origin. For circle \( i \), this distance is given by \( dd[i] \), as formulated here.

\[
\text{In}[61] := \quad \text{dd}[i\_\text{Integer}] = r[i] + \sqrt{x[i]^2 + y[i]^2} ;
\]

The distance between the centers of the circles \( i \) and \( j \) is given by \( p[i,j] \). Two circles do not overlap, if the sum of their radii minus the distance between their centers is not greater than 0: the corresponding term is given by \( dd[i,j] \) for circles \( i \) and \( j \).

\[
\text{In}[62] := \quad p[i\_\text{Integer}, j\_\text{Integer}] = \sqrt{(x[i] - x[j])^2 + (y[i] - y[j])^2} ;
\quad \text{dd}[i\_\text{Integer}, j\_\text{Integer}] = r[i] + r[j] - p[i, j] ;
\]

The optimization variables for describing the position of \( n \) circles are the coordinates of the centers of the circles.

\[
\text{In}[64] := \quad \text{Clear}[\text{vars}, \text{inequalCons}] ;
\]
The first objective, defined by \texttt{obj1}, is to minimize the radius of a circle circumscribing the packed circles, which is the maximum of the values \(dd[i]\) for the \(n\) circles. \textit{MathOptimizer Professional} only evaluates \texttt{Max} for two arguments, requiring the use of a table of constraints to define the first objective.

The second objective, defined by \texttt{obj2}, is to minimize the average distance between the centers of the \(n\) circles.

The overall objective function is defined now as the weighted average of the two objective functions \texttt{obj1} and \texttt{obj2}, with \(0 \leq \lambda \leq 1\) being the weight parameter. Note that the second objective function component is normalized to be the average distance between the centers of the circles, so that the two objective functions will remain of comparable magnitude. For \(\lambda = 0\), the objective function is aimed at finding the smallest embedding circle. For \(\lambda = 1\), the objective function expresses solely the average distance between the circle centers, providing the “tightest possible” packing.

The requirement that the packed circles not overlap means that \(dd[i, j] \leq 0\) for all pairs of \(i\) and \(j\). The input form differs somewhat for the different optimizers. For \textit{MathOptimizer Professional}, it is necessary to add the constraints that define \texttt{rmax}.

The problem as formulated has considerable symmetry. To reduce the symmetry, several constraints are added (to set the first circle object position and the relative location of the second and third circle objects), so that the results of the different optimizers can be more easily compared. Note that adding further similar constraints could increase the overall model complexity significantly, and thus it is avoided.
MathOptimizer and MathOptimizer Professional require bounds on the variables, which are expressed somewhat differently. Since the formulation for MathOptimizer Professional requires an extra variable, the variables with bounds are constructed in one function.

\begin{verbatim}
In[79]:= boundsMO[n_] := Apply[Sequence, Transpose[Table[{0, -3, 3}, {2*n}]]];
varsIn[80]:= boundsofmax[n_] := Join[{{rmax, 0, 3}},
    Transpose[{{vars[n], Table[-3, {2*n}], Table[3, {2*n}]]}}];
\end{verbatim}

The function in[i] produces a constraint describing circle i for use in the InequalityPlot function.

\begin{verbatim}
In[79]:= in[i_Integer] = (x - x[i])^2 + (y - y[i])^2 <= r[i]^2;
\end{verbatim}

The results of the calculations are shown as the combination of three plots: p1 uses InequalityPlot to show the circles with a table of inequalities defining the circles; p2 is a plot of the circumscribing circle; p3 is a plot of the centers of the circles. show combines the plots.

\begin{verbatim}
In[82]:= Clear[p1, p2, p3, show];
\end{verbatim}

\begin{verbatim}
In[83]:= p1[solnrule_, n_] :=
    InequalityPlot[Apply[Or, Table[in[i], {i, n}]] /. solnrule,
    {x, y}, DisplayFunction -> Identity];
p2[solnrule_, n_] := Graphics [Circle[{0, 0}, obj1[n] /. solnrule]]; p3[solnrule_, n_] := Graphics [
    Table[{{PointSize[.025], Point[{x[i], y[i]}]}, {i, n} /. solnrule];
    show[solnrule_, n_] := Block[{rr = obj1[n] /. solnrule},
    Show[p1[solnrule, n], p2[solnrule, n],
    p3[solnrule, n], DisplayFunction -> $DisplayFunction, 
    PlotRange -> {{-rr, rr}, {-rr, rr}}];
\end{verbatim}

Having defined the variables, objective function, and constraints, we now define functions for performing the circle packing and displaying the results using MathOptimizer, MathOptimizer Professional, and NMinimize. The numerical outputs of the following three functions are the values of the two objective functions and the complete solution.

\begin{verbatim}
In[87]:= packMathOptimizer[n_Integer, λ_] :=
    Block[{{soln, solnrule}},
    soln = Optimize[obj[λ, n], symbreakEqualConsMO, Join[
        symbreakInequalConsMO, inequalCons[n]], vars[n], boundsMO[n]];
    solnrule = Thread[vars[n] \\
        soln1]]];
    show[solnrule, n];
    {{obj1[n], obj2[n] /. solnrule, soln}]
\end{verbatim}

\begin{verbatim}
In[88]:= << MathOptimizerPro'callLGO'
\end{verbatim}
In our illustrative examples included here, five circles generated with radii 0.2 to 1.0 are used, together with a \( \lambda \) value of 0.5, as the key problem input.
Timing @ packMathOptimizer[n, 0.5]

FindMinimum::lstol:
The line search decreased the step size to within tolerance specified by
AccuracyGoal and PrecisionGoal but was unable to find a sufficient
decrease in the function. You may need more than MachinePrecision
digits of working precision to meet these tolerances. More...

General::stop: Further output of FindMinimum::lstol will
be suppressed during this calculation. More...

Out[92]= {79.609 Second,
  {1.83936, 1.34321}, {{0.0180293, -0.839165, 0.625977, 0.829711, -0.792936, 0.951527, -1.10627, -0.0172762, -0.318036, 0.315019}, 1.59128, {0.0180293, 0.000193655}, {-0.607948, -1.66888, -0.810966, -1.79069, 0.0238389, -0.365768, 0.0073256, -0.00211565, -0.0241332, -0.728226, -0.0752063, -0.0182119, 0.00585091, -0.25541}, {0.0238389, 0.707359, 0.00162173}}}

□ Numerical Example Using MathOptimizer Professional

AbsoluteTiming is used in this example since Timing will not include the time
required by the compiler and the LGO executable.
Comparison of the MathOptimizer and MathOptimizer Professional results shows that they are very similar. The MathOptimizer Professional result has somewhat lower values for both objective functions, and it also leads to smaller constraint violations: the maximal overlap of the circles equals approximately $1 \times 10^{-11}$. The calculation time for MathOptimizer Professional is approximately 30 times smaller, due its use of a compiled executable for minimization and a compiled dll for objective function and constraint evaluation. In further numerical tests, we have used MathOptimizer Professional to solve similar models up to $n = 40$ circles, without any other prior circle object arrangement specifications: these results are reported elsewhere [66].

**Numerical Example Using NMinimize**

Next, we also completed a few experimental runs to see how NMinimize performs on the same circle packing test problems. The circle packing obtained was not as good as in the result produced by MathOptimizer (or by MathOptimizer Professional).
In[94]:= Timing[packNMinimize[n, 0.5] - 1.5 - 1 - 0.5 - 0.5 1 1.5 - 1.5 - 1 - 0.5 - 0.5 1 1.5]


Note that we also did further extensive tests with respect to all three solvers. The corresponding—far more detailed—numerical results will be reported and discussed elsewhere, for example, in our forthcoming book [29].

## Conclusion

In this article, the subject of optimization using Mathematica tools is discussed, with an emphasis on models that could have multiple local optima of various quality: in such cases, we wish to find the best possible (global) solution. After introducing a general optimization modeling framework, several built-in Mathematica optimization functions and the MathOptimizer package are reviewed. To illustrate the usage of MathOptimizer, several point and multibody configuration analysis and design models are formulated and solved. In an illustrative test related to “best” circle packings, we also used NMinimize and MathOptimizer Professional.

As our results demonstrate, the usage of proper GO tools is both necessary and possible in order to provide globally established numerical solutions to difficult models of theoretical interest and practical relevance. To illustrate the practical use of our optimizers, let us mention here that MathOptimizer has been recently used to design high-quality acoustic equipment: consult, for example, [67]. The LGO solver system embedded in MathOptimizer Professional has already been used in a variety of advanced applications, including, for example, cancer therapy...
planning, laser equipment design, robotics design, industrial (shape) design, chemical product (material composition) design, water quality modeling, wastewater systems design and operations, computational chemistry, econometrics, financial modeling, and others. Most of these results are reported elsewhere: consult, for example, [29, 35–39]. We expect that a significant range of advanced optimization models developed using Mathematica will be successfully analyzed and solved using our packages.

Acknowledgments

The development of MathOptimizer (by the second author) has profited significantly from the quality software, literature, and advice provided by Wolfram Research staff in recent years, and from a project funded in part by Defence R&D Canada under Contract W7707-01-0746/001/HAL.

János Pintér also wishes to thank Christopher J. Purcell (DRDC Atlantic Region, Dartmouth, NS) for many useful comments related to the development of MathOptimizer. Pintér’s research has also been partially supported by the National Research Council of Canada (NRC IRAP Project 362093) and by the Hungarian Scientific Research Fund (OTKA Grant T 034350).

We also wish to thank Mark Sofroniou at Wolfram Research, Inc. for his kind permission to use and customize the Format.m Mathematica package in our MathOptimizer Professional development work.

References


---

### About the Authors

Frank J. Kampas is a senior developer at WAM Systems, Inc., where he is responsible for adding optimization capabilities to the company’s supply-chain management software. He holds a Ph.D. in physics from Stanford University and an M.B.A. from Temple University.

János D. Pintér is a consultant (owner of Pintér Consulting Services, Inc.); he is also an adjunct professor at Dalhousie University. He has authored books, articles, and professional software in the area of advanced nonlinear optimization. He holds a Ph.D. (Moscow State University) and a D.Sc. (Hungarian Academy of Sciences), both in mathematics. For further information, visit myweb.dal.ca/jdpinter.
Frank J. Kampas  
WAM Systems, Inc.  
600 West Germantown Pike, Suite 230  
Plymouth Meeting, Pennsylvania 19462  
fkampas@wamsystems.com

János D. Pintér  
Pintér Consulting Services, Inc.  
129 Glenforest Drive  
Halifax, Nova Scotia  
Canada, B3M 1J2  
jdpinter@hfx.eastlink.ca  
www.pinterconsulting.com