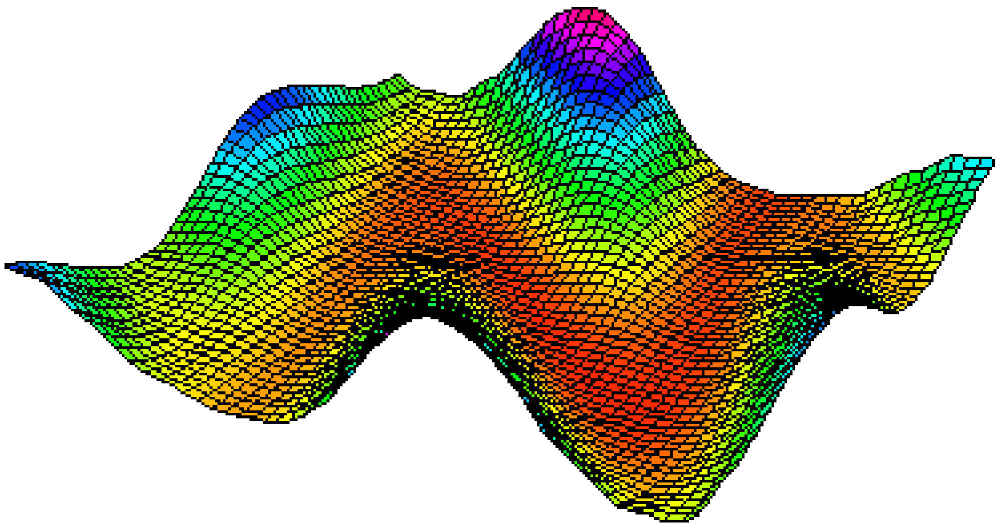
# MathOptimizer

## *An Advanced Modeling and Optimization System for Mathematica Users*



## User Guide

***Version 1.0***

## Product Development and Support

Pintér Consulting Services, Inc.

129 Glenforest Drive,   Halifax, NS   Canada B3M 1J2

|  |  |
|---|---|
| Telephone: | +1-902-443-5910 |
| Fax: | +1-902-431-5100 |
| E-mail: | jdpinter@hfx.eastlink.ca |
| URL: | http://is.dal.ca/~jdpinter |

## Copyright

## Trademarks

## **Summary**

The objective of *MathOptimizer* is to find the best solution of a general class of real-valued, constrained optimization problems. In numerical practice, this means the generation of an approximate solution(s), on the basis of a finite number of model function evaluations, possibly including the use of higher-order information.

The primary current area of *MathOptimizer* applications is nonlinear programming, specifically including global (multiextremal) optimization models. *MathOptimizer* can also be applied to solve linear programming problems (which in principle belong to its more general scope), and certain — as a rule, smaller — integer programming models.

The following standardized model form is considered:

minimize f(x)

subject to the constraints

g(x) = 0

h(x) ≤ 0

xlb ≤ x ≤ xub

Here x is a real vector, f is a scalar objective function, the vector functions g and h are the (equality and inequality) constraints.

The finiteness of the component-wise bounds xlb and xub is assumed. All model functions f, g , h are assumed to be real-valued and (at least) continuous. The individual function lists for f, g and h may be empty, but at least one of these lists is supposed to be non-empty. This means that, for instance, systems of nonlinear equations and model feasibility problems naturally belong to the scope of this general model.

Note additionally that all well-posed, but seemingly more general optimization models can be brought to the above form by elementary transformations. This formulation also includes the broad class of combinatorial (discrete) optimization models. However, *MathOptimizer* currently is not targeted to handle such models, at least not with competitive efficiency. (A related extension is planned soon.)

*MathOptimizer* provides a state-of-art combination of global and local search procedures to solve — in principle, 'all possible' — instances of the general optimization model stated above. The built-in solvers can be used in a variety of operational modes, and they can be flexibly tuned by setting and adjusting their documented options.

This *User Guide* (following technical and legal notes) briefly reviews the background of *MathOptimizer.* This is followed by a detailed description of its usage, a concise discussion of modeling tips, and a fairly extensive exposition related to the solution of various test problems.  Some important aspects of the modeling process are also explained, and new users are advised to work through the examples provided, quite possibly including model changes, as well as your own 'favorite' tests. The last section of the *Guide* contains a few illustrative applications (more will be added), and the bibliography section provides pointers towards further details for the interested Reader.

One of the great advantages of using *Mathematica* (and, hence, of *MathOptimizer*) is that the entire application development process can be documented in a single 'live' notebook that enables sophisticated project documentation (with embedded multimedia components as deemed necessary), while it also fully supports the actual software development and computational work. *Mathematica* notebooks (being ordinary text files) are portable across all supported platforms: therefore they enable convenient and compact information exchange with research colleagues and clients.

The *MathOptimizer User Guide* assumes that the Reader is familiar with the essential concepts of nonlinear and global optimization and these are discussed only briefly. The algorithmic options are explained in sufficient detail, to support their intelligent usage; and a list of reference books is provided to assist the Reader.

It is also assumed that the Reader is reasonably comfortable working with *Mathematica*. If this is not the case, then an introductory training course, or an in-depth self-study course — using *simultaneously* an introductory level text and a computer, with the *Mathematica Book* at arms-length — is strongly recommended. Several books on *Mathematica* (from introductory to advanced level) are also listed in the bibliography section. In addition, there is a huge amount of useful information at the Web site of Wolfram Research, http://www.wri.com.

## Acknowledgements

This work significantly benefited from advice, books and software, and a visiting scholar grant, all generously provided by Wolfram Research, Inc. Several colleagues at WRI were most helpful in answering my beginner's questions.

The *MathOptimizer* software development project (including detailed testing and documentation) were made possible by a grant received from the National Research Council of Canada, and a subsequent research contract from DRDC-Atlantic (Dartmouth, NS, Canada). The valuable advice and suggestions received from Christopher Purcell all through this development work are gratefully acknowledged.

I also wish to thank Anton Rowe and Frank Kampas for useful comments and advice.

## System Requirements and Installation

### Hardware and Software Requirements

*MathOptimizer* will run on all hardware platforms for which a (current) *Mathematica* implementation and support is available from Wolfram Research. *MathOptimizer* has been developed and tested mostly using *Mathematica* Versions 4.0 and 4.1, but earlier versions (3.5, 3.0, and even 2.2) should also be sufficient. A full installation of *Mathematica*, as well as the usage of at least Version 3.0 is recommended.

Most of the examples included in this *Guide* typically can be solved in seconds, or at most in a few minutes on a Pentium II class personal computer with 128 Mbytes of RAM, when using *Mathematica* 4.0 (or later versions). The solution of large, complex optimization models will obviously benefit from more RAM and a faster processor, while — at least for demonstration purposes — any Pentium class PC with 64 Mbytes of RAM should work without problems.

The illustrative examples discussed below were solved (often in a fraction of a second) on a Pentium IV 1.6 GHz processor based machine with 256 MBytes of RAM, but a Celeron 433 MHz, 192 MBytes RAM laptop — with the estimated computing power of a Pentium II 300 MHz machine — was also successfully used during the tests (solving the same models about 4 to 6 times slower).

### Installation

The current *MathOptimizer* installation file system can be obtained on CD or diskette(s) media, or it can be obtained via e-mail (following a proper licensing procedure). The directory structure on the media (using standard PC notation) is as follows:

\MathOptimizer: this folder (directory) contains the current set of *MathOptimizer* distribution packages (.m files, typically in encoded format).

\MathOptimizer\Documentation\English: this subdirectory contains the current *User Guide* (a notebook file), and the corresponding browser categories (an .m file), possibly in zipped format. (Use WinZip or a similar utility for zipped file delivery systems, after copying them into the directory specified below.)

Additional delivery files (technical notes etc.) may also be present.

### *Version 4.x*

To install *MathOptimizer,* place (copy) the MathOptimizer folder directly in the Mathematica 4.x Files\-AddOns\Applications folder. (The name of the actual main *Mathematica* folder may be different on your own machine, but the subdirectory structure should be the same as above.) Then start *Mathematica* and select **Help→Rebuild Help Index** from the front end menu to install the *MathOptimizer* documentation in the help system.

Upon installation, the AddOns\Applications\MathOptimizer folder will contain, the package distribution files, and the \Documentation\English folder.

The package files (possibly in operating system specific executable only versions) are intended for loading into *Mathematica* using the Needs or Get command. The documentation consists of standard notebook (plain text) files, and it is the same for all operating systems.

### *Versions 3.0 and 3.5*

The procedure is essentially identical to the above. To install *MathOptimizer,* place the MathOptimizer folder in the Mathematica 3.x Files\AddOns\Applications folder. Then start *Mathematica* and select **Help→Rebuild Help Index** from the front end menu to install the *MathOptimizer* documentation in the help system.

# Technical Support and Legal Issues

## Technical Support and Contact Information

Licensed users are entitled to technical support. *MathOptimizer* is developed and supported by

Pintér Consulting Services, Inc.

129 Glenforest Drive, Halifax, NS, Canada B3M 1J2.

Tel. 1-902-443-5910     (24-hours message service available)

E-mail: jdpinter@hfx.eastlink.ca

Please note that e-mail is the preferred way of communication, except in cases of extreme urgency; consulting fees may be charged for direct assistance via telephone.

Please let us know of your opinion, questions and problems related to using *MathOptimizer* and this *User Guide*.  New modeling challenges are also of interest.

Consulting, workshops and tutorials, as well as customized software development — related to *MathOptimizer* and to other advanced system modeling and optimization projects — are also offered by Pintér Consulting Services.

## Copyright Notice

## License Agreement

*Software License*

The *MathOptimizer* program system and its entire documentation (referred to below as the 'Product') is the property of Pintér Consulting Services, Inc., Halifax, NS, Canada (abbreviated below as 'PCS').

PCS grants you, an individual or an organization (referred to below as 'User'), the right to use the *MathOptimizer* software that you have received from an authorized dealer, following a proper licensing procedure.

The Product may be stored and used on one or several computers, as long as you have a correesponding valid license and you are its only User.

The licensed User may not make available for use the software to any other party, in any format, without a proper licensing agreement. Multiple simultaneous (or separate) licenses are available from PCS and its authorized partners. Please contact PCs and its partners for such licensing details.

*Limited Warranty*

The *MathOptimizer* software is provided 'as is', without warranty of any kind, regarding its usability for a specific purpose. The entire risk as to the results and the performance of the Product is assumed by the User. PCS disclaims all warranties, either express or implied, including but not limited to implied warranties of merchantability, and fitness for a particular purpose, with respect to the entire Product.

In no event will PCS be liable for any business related damages (including but not limited to: loss of profits, business interruption, loss of information, and the like) arising out of the use or inability to use the Product.

PCS guarantees that all shipped Products are free from defects in materials and workmanship under normal use and service for a period of 90 days.

The functionality of the Product is also guaranteed to the specific details and examples, exactly as documented by this *User Guide.*

*Product Replacement and Returns*

PCS offers replacement of the Product, if it arrives in unusable form (due to damage caused during shipment). Only shipping and handling charges apply in such cases of replacement.

Product returns are accepted within 30 days with a full refund, except shipping and handling charges that apply also in such cases. Return expenses are also to be covered by the User: All returned items have to be in resellable (as if brand new) condition.

A brief explanation regarding the reason for return will be much appreciated: PCS will make every reasonable effort to keep all *MathOptimizer* users satisfied.

*Product Upgrades*

It is planned to add features to the MathOptimizer product: users will be offered the opportunity to purchase upgrades, at a full deduction of the price already paid. Shipping and handling charges will typically be payable also in such cases.

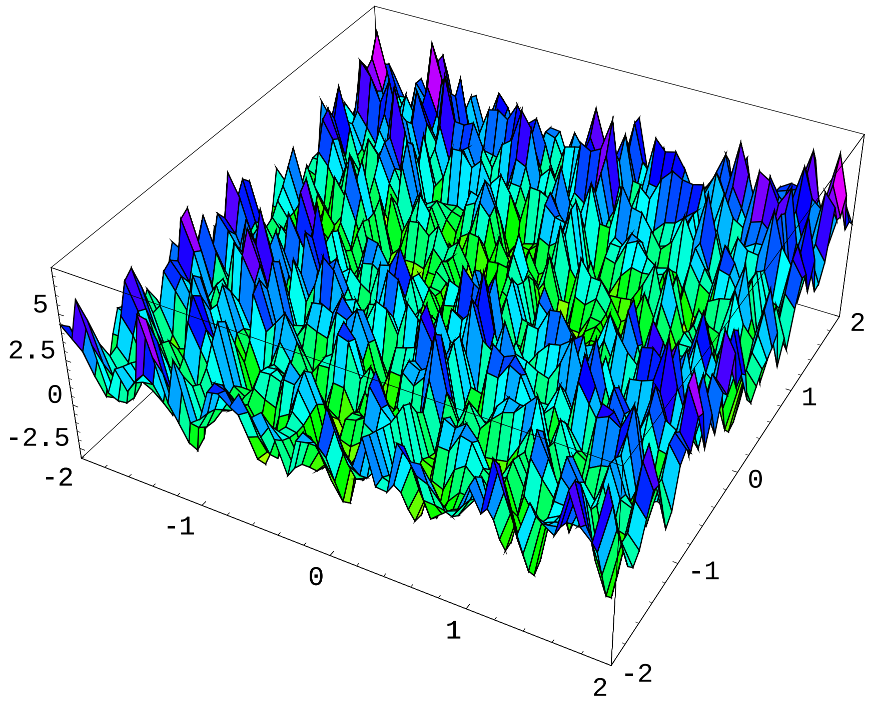
# Mathematical Background Notes

## Scope of Application

*MathOptimizer* is primarily aimed at the solution of the most general (solvable) class of constrained optimization problems. It provides a combination of advanced global and local search procedures for optimizing the value of a given objective function subject to a finite collection of equality and inequality constraints.

Although, at least in principle, *MathOptimizer* can be used to solve arbitrary — linear or nonlinear — optimization models, the current main emphasis is on the continuous nonlinear case. This very general class of models includes 'traditional' (convex) nonlinear optimization. In addition, it includes the far more general case of global optimization problems: these are typically defined by a multimodal objective function on a simple 'box' region, on a linearly constrained set, or on a possibly far more complicated (nonconvex and even disconnected) feasible solution set.

The picture below illustrates the potential difficulty of continuous optimization models encompassed by *MathOptimizer*. Although the feasible set is just a two-dimensional 'box', the objective function is highly multimodal. Hence, the application of a purely local scope solver — as a general rule — will not enable the finding of the best solution (even in much simpler cases). This illustrative example actually will be discussed later on, as one of the numerical challenges.



## Methodology and Illustrative References

*MathOptimizer* combines several local and global scope algorithmic approaches, to support the robust and efficient solution of global as well as local (linear or nonlinear) optimization models.

The global solver approach is based on state-of-art research. The core of the current implementation is a stochastic search algorithm combined with statistically based reasoning. (Theoretically, this method will be globally convergent with probabilty one, as the sample size tends to infinity.)

The current local solver is based on Lagrangian duality theory in convex optimization. In its operations, it uses the built-in function FindMinimum that serves for unconstrained (local) optimization. (Theoretically, this method will be locally convergent for sufficiently smooth models.)

Due to the primary objectives and intended scope of this *User Guide*, there is no reason and room here to discuss the underlying mathematical optimization theory that leads to the strategies embedded in *MathOptimizer*. A few illustrative references are cited below, a more complete list is provided at the end of this *Guide*. Several of the references also include an extensive discussion of nonlinear / global optimization applications.

---

Bazaraa, M.S., Sherali, H.D. and Shetty, C.M. (1993) *Nonlinear Programming: Theory and Algorithms.* Wiley, New York.

Bertsekas, D.P. (1999) *Nonlinear Programming.* (2nd edn.) Athena Scientific, Cambridge, MA.

Edgar, T.F., Himmelblau, D.M, and Lasdon, L.S. (2001) *Optimization of Chemical Processes.* McGraw-Hill, New York.

Hillier, F.S. and Lieberman, G.J. (1986) *Introduction to Operations Research.* (4th edn.) Holden Day, CA.

Horst, R. and Pardalos, P.M., eds. (1995) *Handbook of Global Optimization, Vol. 1.* Kluwer Academic Publishers, Dordrecht / Boston / London.

Papalambros, P.Y. and Wilde, D.J. (2000) *Principles of Optimal Design*, Cambridge University Press, UK.

Pardalos, P.M. and Romeijn, H.E., eds. (2002) *Handbook of Global Optimization, Vol. 2.* Kluwer Academic Publishers, Dordrecht / Boston / London.

Pintér, J.D. (1996) *Global Optimization in Action*, Kluwer Academic Publishers, Dordrecht / Boston / London.

Roos, C. and Terlaky, T. (1998) *Nonlinear Optimization*, TU Delft, The Netherlands.

## Model Formulation, Input and Output Information

| **Input Information** | | **Standard Names** (optional, used in most examples in this *User Guide*) |
|---|---|---|
| x | vector of decision variables | vars |
| xlb | finite lower bound vector of x | varlb |
| xub | finite upper bound vector of x | varub |
| xnom | (or xinit) nominal (or initial) value of x | varnom |
| f(x) | (scalar) objective function | objf |
| g(x) | vector of equality constraints | eqs |
| h(x) | vector of inequality constraints | ineqs |

**Model Formulation**

The following standardized model form is considered throughout this *User Guide*:

minimize f(x)

subject to the constraints

$g(x) = 0$

$h(x) \leq 0$

$xlb \leq x \leq xub$

Note that all (formally more general) constrained optimization models can be brought to the above form by elementary transformations. The individual function lists for f, g and h may be empty, but at least one of these lists is non-empty.

**Basic Assumptions and Explanatory Notes**

All model functions f, g, h are assumed to be at least computable and continuous over the domain [xlb,xub]: this guarantees the theoretical convergence of the global (sub)algorithms used. Note that in the context of the built-in local scope search methodology, the (local) smoothness of the model functions is also tacitly postulated.

The explicit consideration of lower and upper bounds xlb and xub is necessary, since the existence of the global solution (set) is guaranteed only by the existence of such known, finite bounds. It is assumed that xlb < xub component-wise, and that xlb ≤ xnom ≤ xub.

**Output of Results**

- the optimal solution vector found (i.e., numerically estimated) by the algorithm

- the estimated optimum value

- constraint function values at optimum estimate

- aggregated merit (exact penalty) function value at optimum estimate (following a global search phase)

- statistically established global bound for optimum value (following a global search phase)

- constraint satisfaction/violation level  (following a local search phase)

- Kuhn-Tucker conditions: satisfaction/violation level (following a local search phase)

- complementary slackness conditions: satisfaction/violation level (following a local search phase)

Depending on the solver options and combinations used, some of this information may be suppressed in the final result. By selecting a detailed reporting option, the full set of solver messages may be reported to the Messages window, in addition to the final result that appears in the user's own application note-book.

## *MathOptimizer* Usage Definitions and Options

### ■ Activate *MathOptimizer*

In order to invoke *MathOptimizer*, the user simply enters

```
Needs["MathOptimizer`Optimize`"];
$ContextPath
```

```
{MathOptimizer`Optimize`, MathOptimizer`CNLP`,
 MathOptimizer`MS`, Global`, System`}
```

This yields the output shown above. (Make sure to select 'No', if *Mathematica* asks you about evaluating all initialization cells.)

Note that the context path query shown above serves only for verifying the correct context settings: although it is not essential, it may be wise to routinely use this simple check, to avoid seemingly 'puzzling' behaviour later.

## ■ Optimize Function Definition and Options

The package **Optimize** serves to integrate the currently available solver options. Basic information regarding its usage can be invoked as shown below.

```
?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(x)
  subject to g(x)=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 stand-alone 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 possiblities: 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.
```

The options of the package **Optimize** can be queried as shown below.

```
Options[Optimize]
```

```
{Version → 1., GlobalSolverMode → 1,
 LocalSolverMode → 1, ReportLevel → 0}
```

```
?GlobalSolverMode
```

```
GlobalSolverMode is an Optimize option. Its default setting
  is 1: in this case the global scope search mode option
  1 (currently set to the algorithm MS) is applied.  The
  global search mode can also be used on its own, by setting
  GlobalSolverMode -> 1 and LocalSolverMode -> 0. For further
  information regarding the global solver mode MS, type ?MS.
```

**?LocalSolverMode**

```
LocalSolverMode is an Optimize option. Its default setting
  is 1: in this case the local scope search mode option 1
  (currently set to the algorithm CNLP) is applied.  The
  local search mode can also be used on itw own, by setting
  GlobalSolverMode -> 0 and LocalSolverMode -> 1. For further
  information regarding the local solver mode CNLP, type ?CNLP.
```

**?ReportLevel**

```
ReportLevel is an Optimize option. If it is set to 1, then all
  results are reported in the form of itemized messages.
  For the default setting ReportLevel -> 0, the solution
  information is provided only in a concise list format.
```

## ■ MS Function Definition and Options

The options of the solver package **MS** can be queried as shown below.

**?MS**

```
MS[f, g, h, x, xinit, xlb, xub, opts] is an adaptive
  stochastic (MultiStart based) search procedure which
  numerically approximates the global solution of the
  following general, possibly multiextremal constrained
  optimization problem: minimize f(x) subject to g(x)=0 and
  h(x)<=0. It is assumed that the real vector x belongs to
  a given finite interval range [xlb, xub], xinit is a list
  of initial (nominal) values for x. The objective f and the
  constraints g and h are assumed to be continuous, g and h
  are lists of Mathematica expressions. The collection of
  real-valued model functions {f, g, h} is aggregated by
  a merit (exact penalty) function. MS returns the global
  estimate of the solution vector, the corresponding merit
  function value and its (stochastic) global lower bound
  estimate, the individual model function values, and the
  total number of model function evaluations. Although MS
  can be used in stand-alone mode (especially to handle
  pure box-constrained models), it is recommended to apply
  subsequently a more precise local search method such as
  CNLP. (Seamless solver mode combinations are supported by
  the package Optimize.) The argument 'opts' denotes a list
  of algorithm features: use Options[MS] to display these.
```

**Options[MS]**

```
{Version → 1., ReportLevelMS → 0, PenaltyMultiplier → 1,
 MaxIterations → 0, MaxSample → 0, RandomSeed → 0}
```

```
?ReportLevelMS
```

```
ReportLevelMS is an option in MS. If it is set to 1,
  then all results  are reported in the form of itemized
  messages. For the default setting ReportLevelMS -> 0, the
  solution information is provided only in a list format.
```

```
?PenaltyMultiplier
```

```
PenaltyMultiplier is an option in MS: it is the penalty
  factor 'p' used in the definition of the merit function.
  The latter is defined as meritfct = f(x) + p*(||g(
  x)|| + ||Max[h(x),0]||); here the absolute value (l1)-
  norm is used. PenaltyMultiplier can be set to any non-
  negative value, its default setting is 1. Note that
  this setting assumes that the objective and constraint
  functions are well-scaled (i.e. that all functions
  are approximately of the same order of magnitude). It
  is recommended to pre-scale the functions as much as
  possible, to attain good scaling. Larger PenaltyMultiplier
  values - as a rule - will enforce model feasibility,
  but their use may also lead to numerical difficulties.
```

Note that the option MaxIterations below has an identical name to one of the *Mathematica* function option names. (This could have been avoided, but would lead to more complicated notations.) Therefore the More... hyperlink — see below — will lead to a feature of *Mathematica* itself (and hence there is no need to follow up on it in the present context).

```
?MaxIterations
```

```
MaxIterations is an option in MS: it determines the number of
   multistart iteration cycles. MaxIterations can be set
   to any non-negative integer value. It is recommended to
   set its value as an increasing function of the model
   size, considering both the number of variables and
   constraints. Note that larger values will increase the
   runtime linearly. If MaxIterations is set to 0, then the
   default internal setting will take place: the latter
   is set in line with the recommendation above. More…
```

```
?MaxSample
```

```
MaxSample is an option in MS: it determines the number of
  sample points in each multistart iteration cycle (up to
  a rounding operation; a minimal value for this parameter
  is set internally). MaxSample can be set to any non-
  negative integer value, the default setting being simply
  0. It is recommended to set its value as an increasing
  function of the model size, considering both the number
  of variables and constraints. If MaxSample is set to
  0, then the default internal setting will take place:
  the latter is set in line with the recommendation above.
```

```
?RandomSeed
```

```
RandomSeed is an option in MS: it determines the way the
  intrinsic random number generator is initialized. Its value
  can be set to an arbitrary integer which then will be used
  to seed the generator: this option supports the generation
  of possibly different numerical results in different runs,
  even if all other option parameters are the same. The
  default setting is 0, in which case the random seed is set
  internally to a fixed value: this option guarantees that
  identically parameterized runs will lead to identical results.
```

## ■ CNLP Function Definition and Options

The options of the solver package **CNLP** can be queried as shown below.

```
?CNLP
```

```
CNLP[f, g, h, x, xinit, xlb, xub, opts] numerically
  approximates a local solution to the following general
  constrained optimization problem: minimize f(x) subject
  to g(x)=0 and h(x)<=0. The real vector x and the given
  initial solution xinit are assumed to belong to the
  finite interval range [xlb, xub]. The objective f and the
  constraints g and h are arbitrary, possibly nonlinear
  real-valued functions, g and h are lists of Mathematica
  expressions. The local C1-continuity (smoothness) of all
  model functions is assumed. CNLP is based on a locally
  convergent Constrained Non-Linear Programming algorithm.
  CNLP returns the local solution vector, the corresponding
  local optimum estimate and constraint function values, and
  additional model information (related to the satisfaction
  of feasibility, Kuhn-Tucker, and complementary slackness
  conditions). In order to solve multiextremal models, it is
  recommended to apply first a global scope method such as
  MS. (Seamless solver mode combinations are supported by
  the package Optimize.) The argument 'opts' denotes a list
  of algorithm features: use Options[CNLP] to display these.
```

```
Options[CNLP]
```

```
{Version → 1., ReportLevelCNLP → 0, ConvergenceLevel → 6,
 PrecisionLevel → 6, UnconstrainedModel → False}
```

```
?ReportLevelCNLP
```

```
ReportLevelCNLP is an option in CNLP. If it is set to 1,
  then all results are reported in the form of itemized
  messages. For the default setting ReportLevelCNLP -> 0,
  the solution information is provided only in a list format.
```

```
?ConvergenceLevel
```

```
ConvergenceLevel is an option in CNLP. It defines the 10-based
  negative exponent parameter used in the convergence test in
  CNLP iterations. (Example: ConvergenceLevel -> 6 sets the
  convergence test parameter to 10^-6.) For well-scaled models,
  this default setting should be appropriate. Increasing
  values can be expected to lead to higher precision,
  but runtimes may become longer, and - due to numerical
  errors - such precisions may not always be attainable.
```

```
?PrecisionLevel
```

```
PrecisionLevel is an option in CNLP. It defines the 10-based
  negative exponent parameter used in the constraint
  feasibility, Kuhn-Tucker conditions and complementary
  slackness tests. (Example: PrecisionLevel -> 6 sets the
  required precision to 10^-6.) For well-scaled models, this
  default setting should be appropriate. Increasing values
  can be expected to lead to higher precision, but runtimes
  may become longer, and - due to modeling or numerical
  errors - such precisions may not always be attainable.
```

```
? UnconstrainedModel
```

```
UnconstrainedModel is an option in CNLP. If it is set to True (
  and the number of constraint functions is zero indeed),
  then the built-in function FindMinimum is launched from
  within CNLP. Its default setting is False: in this case CNLP
  is applied also to unconstrained models. This may lead to
  longer execution times, and in some cases to higher precision.
```

## Some General Model Formulation Tips

### ■ Explicit Bound Settings and Variable Range Scaling

By results from classical analysis, the existence of finite bounds xlb ≤ x ≤ xub is needed to assure that the general optimization model form considered indeed has a finite (globally optimal) solution. One can think of minimizing or maximizing the trivially simple univariate function f(x)=x over the interval (-∞, 0) to see that the closedness of the variable range [xlb, xub] is also essential.

It is always good practice to select 'safe', but 'not exaggerated' bounds. In general, global and local search methods will benefit from well-chosen, reasonable bounds and a good quality initial (nominal) solution vector. If the bounds are possibly or sometimes binding, then one should add the corresponding bound constraints also to the explicit list of inequality constraints, at least when also applying local search methods. (Let us note here that the global search methods embedded in *MathOptimizer* will never leave the interval [xlb, xub]; at the same time, in general they also will run slower when larger intervals and/or more model constraints are considered).

Evidently, the — generally speaking, multidimensional — interval [xlb,xub] can be simply scaled to the unit 'box' [0,1] of corresponding dimensionality, by the transformation

$x_i = \mathrm{xlb}_i + x_{s,i}(\mathrm{xub}_i - \mathrm{xlb}_i) \quad i=1,...,n$

Here $n$ is the model dimensionality, $i$ denotes the indices of the vector components, and $x_s \in [0,1]$ is the scaled vector.

Again, it is often beneficial and hence advisable to make such a scaling before using *MathOptimizer* (or other solvers), to improve algorithmic search efficiency.

## ■ Model Function Scaling

Let us recall the standardized model statement and assume that the model variables are already scaled, if that was deemed necessary. Below we shall briefly address the numerically important issue of model function scaling.

To this end, we apply first the following equivalent model transformation: we shall minimize z, and add $f(x) \leq z$ to the set of constraints. Furthermore, all constraints will be considered individually. Two cases will be distinguished regarding whether the constraint considered has a non-zero constant term in it, or not. For those constraints that do not have such a constant term, a given (preset) tolerance level $t > 0$ is introduced. (This parameter should naturally correspond to the parameter PrecisionLevel defined in connection with the local search component(s) of *MathOptimizer*.) All equality constraints will be replaced by the componentwise pair of inequalities $-t \leq g_j(x) \leq t$; a similar (but only 'one-sided') transformation is applied to all constraints that do not have a non-zero constant term.

For simplicity (after all such transformations are completed) we shall denote such an individually considered constraint function by $c(x) \leq k$, instead of using indexed notations for the components of g and h.

Now, constraint $c(x) \leq k$ can be simply replaced by $(c(x) - k)/k \leq 0$, if k is 'sufficently large', that is (say) at least $k > 100*t$ holds. For those constraints in which this is not the case (this will specifically include all equality relations), one can use instead of the original constraint the approximation $(c(x) - k)/(1+k) \leq 0$.

Let us emphasize here that there is no guaranteed recipe regarding the 'best possible' model scaling, and in many cases some experimentation will help to find good quality numerical solutions even in difficult models.

# Local Nonlinear (and Linear) Optimization: Test Examples

## ■ Getting Started: A Simple Example

The following small-scale example has all ingredients of the model type considered in the framework of *MathOptimizer*. (The optimum value in this model is zero at x1 = x2 = 1.)

The main steps of the modeling and solution procedure are illustrated below: comments set in blue serve only for explanation..

```
(* Define optimization model *)

vars = {x1, x2}; (* decision variables *)

varnom = {8., -14.}; (* nominal (initial) values of variables
*)

varlb = {-10., -15.}; (* lower bounds of variables *)

varub = {20., 10.}; (* upper bounds of variables *)

objf = 10.*(x1^2 - x2)^2 + (x1 - 1)^2; (* objective function,
to minimize *)

eqs = {x1 - x1*x2}; (* equality constraints *)

ineqs = {3.*x1 + 4.*x2 - 25.}; (* inequality constraints,
stated in <=0 form *)

(* Solve model *)

Optimize[objf, eqs, ineqs, vars, varnom, varlb, varub,
GlobalSolverMode -> 1, LocalSolverMode -> 1, ReportLevel -> 1]
```

$\{\{1., 1.\}, 6.91861\times10^{-19}, \{-1.68423\times10^{-9}\}, \{-18.\}, \{1.68423\times10^{-9}, 7.33364\times10^{-15}, 0.\}\}$

Setting ReportLevel → 1 allows tracing the solution procedure in the Messages window: the Reader may like to see these messages now, to familiarize yourself with the output information. This setting is especially advisable during model development and testing, while ReportLevel → 0 suits better the 'routine' solution of well-tested models.

## ■ Extracting Result Information

It is easy to extract components from the result list, assuming that we know the list entry position of the component in question. (Again, consult the itemized sequence of messages, when ReportLevel is set to 1.)

```
result = Optimize[objf, eqs, ineqs, vars, varnom, varlb, varub,
GlobalSolverMode -> 1, LocalSolverMode -> 1, ReportLevel -> 0]
```

$\{\{1., 1.\}, 6.91861\times10^{-19}, \{-1.68423\times10^{-9}\}, \{-18.\}, \{1.68423\times10^{-9}, 7.33364\times10^{-15}, 0.\}\}$

Notice passing by the identical results of the two runs shown above, due to the default seed (re)setting of the built-in random number generator.

```
varopt = result[[1]];
Print["estimated optimal solution vector: ", varopt];
```

```
estimated optimal solution vector: {1., 1.}
```

```
fopt = result[[2]];
Print["optimum estimate: ", fopt];
```

```
optimum estimate: 6.91861×10^-19
```

```
eqsopt = result[[3]];
Print["equality constraint values at optimum estimate: ",
eqsopt];
```

```
equality constraint values at optimum estimate: {-1.68423×10^-9}
```

```
ineqsopt = result[[4]];
Print["inequality constraint values at optimum estimate: ",
ineqsopt];
```

```
inequality constraint values at optimum estimate: {-18.}
```

```
violsopt = result[[5]];
Print["violation levels of optimality conditions at optimum
estimate: ", violsopt];
```

```
violation levels of optimality conditions at optimum estimate:
 {1.68423×10^-9, 7.33364×10^-15, 0.}
```

The entries of the last list above correspond to the maximal constraint feasibility violation, the violation of the Kuhn-Tucker conditions, and the violation of the complementary slackness condition (at the optimum estimate found).

## ■ Unconstrained Nonlinear Optimization

If the constraint (vector) functions g and h are absent, and the bound constraints xlb ≤ x ≤ xub are not 'tight' — i.e., the optimal solution is known to be in the interior of [xlb, xub] — then the model is essentially unconstrained. Recall, however that the existence of bounds is needed to assure that the model indeed has a finite globally optimal solution.

Unconstrained optimization models can be solved in the local sense also by the built-in function FindMinimum which, as a  rule, should work faster than the more complex CNLP procedure. Therefore FindMinimum is made available internally by the CNLP function as an option: this feature will be illustrated below.

### Rosenbrock Test Problem

The following standard test problem is rather easy, although it is not convex. (The optimum value is zero at x1 = x2 = 1.)

Let us solve this model first by using only the local search model via the package Optimize.

```
vars = {x1, x2};
varnom = {8, -4};
varlb = {-10, -10};
varub = {10, 10};
objf = 100*(x1^2 - x2)^2 + (x1 - 1)^2;
eqs = {};
ineqs = {};
result = Optimize[objf, eqs, ineqs, vars, varnom, varlb, varub,
GlobalSolverMode -> 0, LocalSolverMode -> 1, ReportLevel -> 1]
```

$\{\{1., 1.\}, 4.94271\times10^{-30}, \{\}, \{\}, \{-\infty, 9.95\times10^{-14}, 0\}\}$

Observe that since the model is essentially unconstrained (except the stated, non-binding lower and upper bounds), the maximal constraint violation is returned as -∞. In presence of bounds, this value will be non-negative: namely, it is expected to be (approximately) zero for feasible solutions and a 'non-acceptably large' positive value for infeasible solutions found.

Let us solve the same model, using only CNLP, and the information that the model is unconstrained.

```
CNLP[objf, eqs, ineqs, vars, varnom, varlb, varub,
 UnconstrainedModel → True, ReportLevelCNLP → 1]
```

```
{0., {x1 → 1., x2 → 1.}}
```

One can compare the above operations to those of the built-in function FindMinimum.

```
FindMinimum[objf, {x1, 8}, {x2, -4}]
```

$\{4.19981\times10^{-12}, \{x1\to0.999999, x2\to0.999998\}\}$

The Reader may notice that — in this particular example, as well as in numerous others — CNLP produces a more precise result then FindMinimum on its own: this is due to model-dependent accuracy settings in CNLP. In general, this strategy may also lead to (somewhat or more noticeably) slower solution procedures: there is an obvious trade-off between solution speed and quality.

One can also notice the difference between the form of results reported by Optimize and FindMinimum. We think that — at least in the present context — it is simpler to use the resulting output list returned by Optimize rather than the list of rules returned by FindMinimum.

## Internal Verification of Unconstrained Structure

Note that if the model is, in fact, constrained (i.e. the function lists g and/or h are not empty), then CNLP will be used even if UnconstrainedModel is misspecified. We shall illustrate this feature, by adding a constraint to the model stated above. (Notice the invalid True setting below.)

```
ineqs = {x1 +x2 - 10};
result = CNLP[objf, eqs, ineqs, vars, varnom, varlb, varub,
UnconstrainedModel→True, ReportLevelCNLP→1]
```

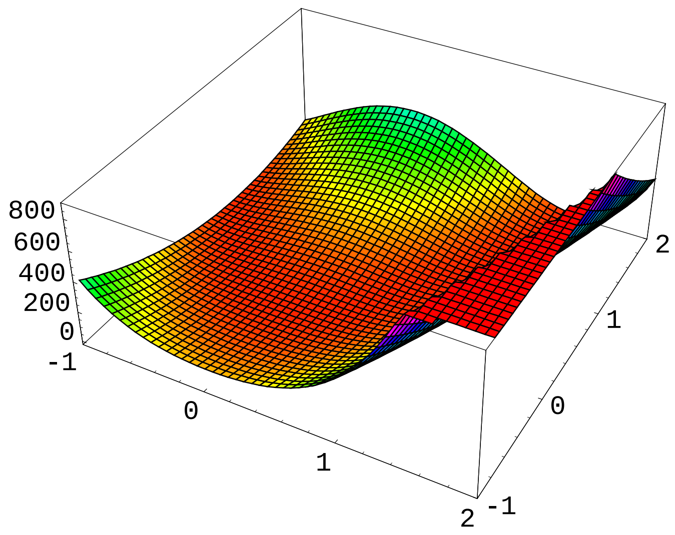
$\{\{1., 1.\}, 2.27534\times10^{-23}, \{\}, \{-8.\}, \{0., 1.60561\times10^{-11}, 0.\}\}$

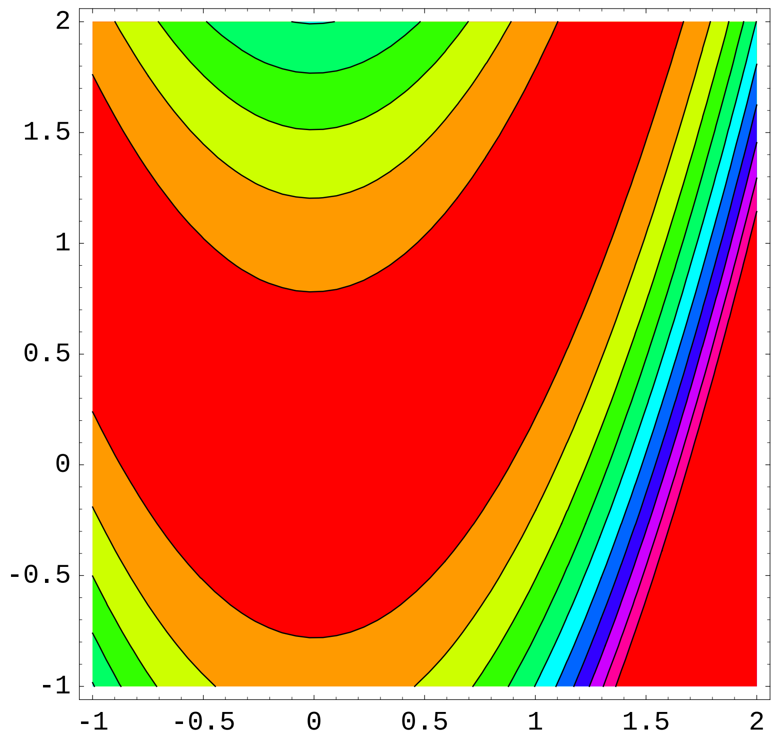
## ■ Model Visualization

The excellent visualization capabilities of *Mathematica* can become helpful tools, assisting model development, analysis, corrections and changes: their use is especially justified in case of nonlinear models.

For instance, three-dimensional and contour plot pictures can be generated at any given input vector x (including the nominal value and/or the solution point found). The projection subspaces can be interactively selected by the user. (Of course, the pictures shown below are related to the objective function of the most recently analysed test problem.)

```
Plot3D[objf, {x1,-1,2}, {x2,-1,2}, ColorFunction→Hue,
PlotPoints→50];
ContourPlot[objf, {x1,-1,2}, {x2,-1,2}, ColorFunction→Hue,
PlotPoints→50];
```





## ■ Linear Programming

Let us consider next a 6-variable, 9-constraint linear programming model. Note that the built-in LP solver (ConstrainedMin) is most likely to work faster for such models, but Optimize or CNLP also works. Since linear programming models are convex, there is no need to apply a global search mode.

(The optimum value is at a location that is perhaps not seen easily even for this small model. )

```
vars = {x1, x2, x3, x4, x5, x6};
varnom = {1., -2., 3., 2., -3., -1.};
varlb = Table [0, {Length[vars]}];                  (* a simple
way to set bounds *)
varub = Table [1000, {Length[vars]}];
objf = -(3x1 + 2*x2 + 1*x3 + 1*x4 + 2*x5 + 1*x6);  (*
transformed from a maximization problem *)
eqs = {};
ineqs = {
    -x1, -x2, -x3, -x4, -x5, -x6,                   (* lower
bounds included as constraints *)
    x1 + x2 + x3 + x4 + x5 + x6 - 100,
    1*x1 - 2*x2 + 3*x3 - 4*x4 + 5*x5 - 6*x6 - 200,
    6*x1 + 5*x2 + 4*x3 + 3*x4 + 2*x5 + 1*x6 - 200};
CNLP[objf, eqs, ineqs, vars, varnom, varlb, varub]
```

$\{\{6.25, -1.39548\times10^{-8}, 1.28622\times10^{-9}, -2.25678\times10^{-8}, 68.75, 25.\},$
$-181.25, \{\}, \{-6.25, 1.39548\times10^{-8}, -1.28622\times10^{-9}, 2.25678\times10^{-8},$
$-68.75, -25., -1.85248\times10^{-8}, 1.94493\times10^{-9}, 4.07945\times10^{-9}\},$
$\{2.25678\times10^{-8}, 1.81375\times10^{-7}, 3.68772\times10^{-9}\}\}$

Let us solve now the same model by applying ConstrainedMin: this requires the corresponding constraint set format as shown below.

```
ineqsCM = { -x1 < 0, -x2 < 0, -x3 < 0, -x4 < 0, -x5 < 0, -x6
<0,
    x1 + x2 + x3 + x4 + x5 + x6 - 100 <0,
    1*x1 - 2*x2 + 3*x3 - 4*x4 + 5*x5 - 6*x6 - 200 <0,
    6*x1 + 5*x2 + 4*x3 + 3*x4 + 2*x5 + 1*x6 - 200 <0};

ConstrainedMin[objf, ineqsCM, vars]
```

$$\left\{-\frac{725}{4}, \left\{x1\to\frac{25}{4}, x2\to0, x3\to0, x4\to0, x5\to\frac{275}{4}, x6\to25\right\}\right\}$$

The numerical values obtained are the same, up to a small error in precision; see below. (This is due to the difference in the algorithmic approaches used by ConstrainedMin and CNLP).

```
N[%]
```

```
{-181.25, {x1 → 6.25, x2 → 0., x3 → 0., x4 → 0., x5 → 68.75, x6 → 25.}}
```

Observe again the difference in the style of reporting between the *MathOptimizer* packages and ConstrainedMin. If one needs to solve only linear models, then in general — at least currently — the use of the built-in, faster ConstrainedMin can be recommended. On the other hand, if nonlinear models (versions) are also considered, then the *MathOptimizer* packages produce a set of internally consistent results for all such cases.

## ■ Constrained Nonlinear Optimization: A Small Model

The following is still a fairly simple constrained model which has two variables, two equality and two inequality constraints. (The optimum value is zero at the vector x = 0.)

```
vars = {x1, x2};
varnom = {10, -10};
varlb = Table [-100, {Length[vars]}];
varub = Table [100, {Length[vars]}];
objf = (2*x1^2-x2)^2 + (x2 - 6*x1^2)^2;
eqs = {x1 - 10x2 - x1*x2, x1 - 3*x2};
ineqs = {x2 + x1 - 1, x2 - x1 -2};
result = CNLP[objf, eqs, ineqs, vars, varnom, varlb, varub,
ReportLevel->0]
```

```
{{-1.26272×10^-8, -1.94303×10^-9},
 7.55072×10^-18, {6.8031×10^-9, -6.7981×10^-9},
 {-1., -2.}, {6.8031×10^-9, 7.36977×10^-11, 0.}}
```

## ■ Model Sensitivity to Parameter Changes

Changing the currently formulated model (as stated above) just a little bit may lead to a rather different solution, or even to infeasible models. This observation indicates that even small NLP models can be very difficult.

For example, let us change the blue parameter below a few times, and observe the solution changes! (Some recommended parameter values are: 1, 5, 10, 20, 50, 100,...)

For instance, at parameter value 10 the solution found is infeasible. (The Reader can verify this by setting the blue parameter to 10.) This finding shows that local search *per se* may not be sufficient to handle the resulting model.

```
eqs = {x1 - 10x2 - x1*x2 - 10., x1 - 3*x2};
result = CNLP[objf, eqs, ineqs, vars, varnom, varlb, varub]
```

```
{{-1.17537, -1.17239}, 105.006, {-0.829487, 2.34179},
 {-3.34776, -1.99701}, {2.34179, 2.90436×10^-6, 0.}}
```

## ■ Constrained Nonlinear Optimization: A More Difficult Small Model

In the model formulated below, the objective function as well as the equality constraint lead to non-convex structure. (The global optimum value is zero at x1 = x2 = 0.)

Try to change the initial (nominal) solution as suggested below, see comments in blue.

```
vars = {x1, x2};
varlb = Table [-10, {Length[vars]}];
varub = Table [10, {Length[vars]}];
varnom = {-4, -2}; (* this leads to an infeasible solution... *)
objf = (2*x1^2-x2^2)^2 + (x2 - 6*x1^2)^2;
eqs = {x1 - 10x2 - 100*Sin[2x1+3x2]};
ineqs = {x2 + x1 - 2};

result = CNLP[objf, eqs, ineqs, vars, varnom, varlb, varub]
```

```
{{-0.897538, -3.4753}, 178.585, {-7.44338×10^-8},
 {-6.37284}, {7.44338×10^-8, 5.88178×10^-10, 0.}}
```

```
varnom = {-2, -1}; (* this leads to a local optimum... *)
result = CNLP[objf, eqs, ineqs, vars, varnom, varlb, varub]
```

```
{{-0.391571, -1.77519}, 15.3558, {9.73077×10^-9},
 {-4.16676}, {9.73077×10^-9, 4.73715×10^-12, 0.}}
```

```
varnom = {1, -1}; (* this 'sufficiently close guess' leads to
the global optimum... *)
result = CNLP[objf, eqs, ineqs, vars, varnom, varlb, varub,
ReportLevel->1]
```

```
{{-1.5432×10^-11, -7.05692×10^-20}, 5.18394×10^-39,
 {3.07097×10^-9}, {-2.}, {3.07097×10^-9, 5.82826×10^-14, 0.}}
```

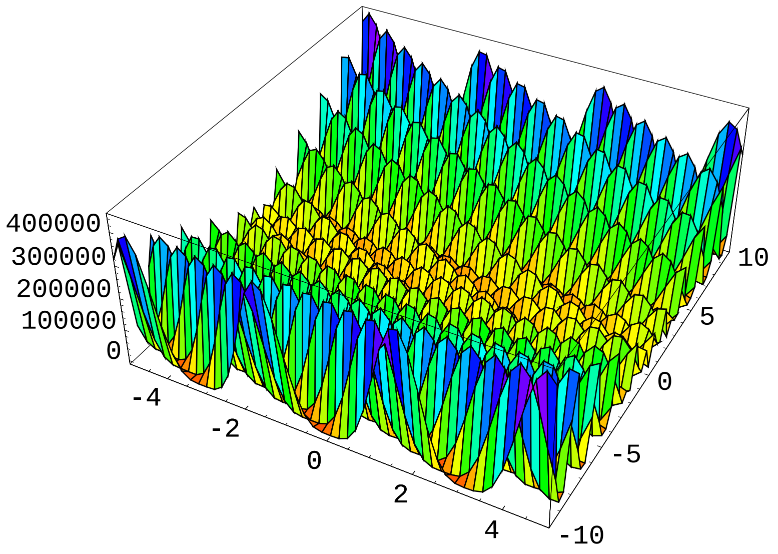
## ■ Merit Function: Model Visualization and Penalty Scaling Effects

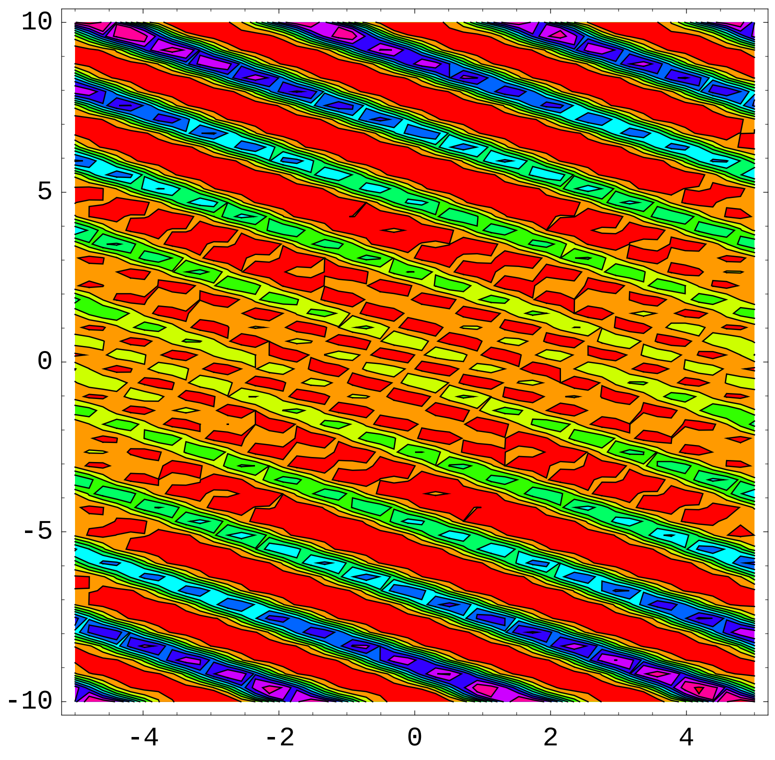
This section illustrates the aggregation of model information in a so-called 'merit function'. This is an exact penalty function, as shown by its definition below. Such a function is used internally by the global solver MS.

```
(* one can introduce a penalty factor, to try better
'enforcing' feasible solutions *)
(* a 'good' parameter choice may need some subjective
experimentation... try a few values. *)
penmult = 10;
(* aggregated merit function *)
meritfct = objf + penmult*(eqs.eqs + Map[Max,{ineqs,
0}].Map[Max,{ineqs, 0}]);
```

Pictures of the merit function may serve to indicate the potential difficulty of the resulting (approximate, parameterized) unconstrained optimization problem.To solve this optimization problem, often genuine global search is needed.

```
Plot3D[meritfct, {x1,-5,5}, {x2,-10,10}, ColorFunction→Hue,
PlotPoints→50];
ContourPlot[meritfct, {x1,-5,5}, {x2,-10,10}, ColorFunction→
Hue, PlotPoints→50];
```





## ■ Constrained Nonlinear Optimization: A 4-Variable, 4-Constraint Model

This is a bit more 'tricky' model: both the objective and the equality constraints are non-convex. The global solution is found (by local search only), when the search is started from a good initial 'guess'. (The optimum value is zero at x = 0.)

```
vars = {x1, x2, x3, x4};
varnom = {4., -2., 3., 5.};
varlb = Table [-100, {Length[vars]}];
varub = Table [100, {Length[vars]}];
objf = 2*(x1 + x3 - x4)^2 + (x2 - x1 + x3 - x4)^2 + (x1*x2 -
x3*x4)^2;
eqs = {x1^2 - Sin[x2] - x4, x1*x3 - x2*x4*x1};
ineqs = {2*x1 + 5*x2 + x3 + x4 - 10, 3*x1 - 2*x2 + x3 - 4*x4 -
5};
result = CNLP[objf, eqs, ineqs, vars, varnom, varlb, varub]
```

$\{\{3.39568\times10^{-13}, 6.56941\times10^{-13}, 1.00604\times10^{-12}, 1.32411\times10^{-12}\},$
$9.24767\times10^{-28}, \{-1.98105\times10^{-12}, 3.41619\times10^{-25}\},$
$\{-10., -5.\}, \{1.98105\times10^{-12}, 1.0265\times10^{-13}, 0.\}\}$

## ■ Constrained Nonlinear Optimization: A 6-Variable, 7-Constraint Model

This model has a non-convex objective function, and four non-convex constraints, as well as three simple linear inequality constraints. Again, the global solution is found by local search, when starting from a good initial 'guess'. (The optimum value is zero at x = 0.)

```
vars = {x1, x2, x3, x4, x5, x6};
varnom = {1., -2., 1., 2., 1., -1.};
varlb = Table [-100, {Length[vars]}];
varub = Table [100, {Length[vars]}];
objf = (x1 + x2)^2 + (x3 - x5)^2 + (x6 - x4)^2 + 2*(x1 + x3 -
x4)^2 + (x2 - x1 + x3 - x4)^2 +
    10*Sin[x5 - x6 + x1]^2;
eqs = {
    x1^2 - Sin[x2] - x4 + x5 + x6,
    x1*x3 - x2*x4*x1 - x5 - Sin[x6 - x1 - x3],
    x2*x6*Cos[x5] - Sin[x3*x4] + x2 - x5,
    x1*x2 -x3^2 - x4*x5 - x6^2
    };
ineqs = {
    2*x1 + 5*x2 + x3 + x4 - 1,
    3*x1 - 2*x2 + x3 - 4*x4,
    x1 + x2 + x3 + x4 + x5 + x6 - 2
    };
CNLP[objf, eqs, ineqs, vars, varnom, varlb, varub]
```

```
{{2.03184×10^-7, 1.41507×10^-7, 2.32282×10^-7,
  1.43425×10^-7, 5.47783×10^-8, 3.2716×10^-7}, 4.03276×10^-13,
 {9.70055×10^-8, 5.35285×10^-8, 8.67291×10^-8, -1.40093×10^-13},
 {-0.999999, -1.4881×10^-8, -2.},
 {9.70055×10^-8, 4.21914×10^-8, 8.74061×10^-15}}
```

Starting from a different initial point, one obtains another (suboptimal, i.e. locally optimal) result. Observe that the solution is feasible, and the Kuhn-Tucker conditions are satisfied to sufficient — that is, to the prescribed — precision; however, the solution found is inferior to the previously found one (which actually is the global optimum).

```
varnom = {10., -10., 10., 10., 10., -10.};
CNLP[objf, eqs, ineqs, vars, varnom, varlb, varub]
```

```
{{-0.472344, -0.848764, 0.469209,
  0.660639, -1.34795, 1.03502}, 7.18862,
 {-2.08203×10^-8, -5.70788×10^-8, 2.68579×10^-8, -8.2118×10^-9},
 {-5.05866, -1.89285, -2.50419}, {5.70788×10^-8, 9.5235×10^-8, 0.}}
```

A few of the examples shown above already indicate the need for including a suitable global scope search approach: in the next section we will follow that route.

# Global Optimization: Test Examples

## ■ Standard Univariate Global Optimization Tests

These models are often used in the global optimization literature. In spite of their (typical) simplicity, the test functions below cover a reasonable variety of model forms met in practice, and hence may serve as primary illustrations of GO algorithm workings. (For further examples, see for instance, Dixon and Szegö, 1975; Pintér, 1996; Floudas et al., 1999.) Note that for better — or for more interesting — visualization, several known test models have been modified for the present purposes.

### Illustrative Model Instances

In all simple examples stated in this section, only box-constraints are given.

```
eqs = {}; ineqs = {};
```

In the following test function, the global (approximate) solution value is -0.999612 (approximately) at 7.85411; notice that the exponential term decreases as x becomes larger.

```
f1= Exp[-x] - (Sin[x])^3; lb1 = -5; ub1 = 10; xnom1 = 1;
```

The next one is a standard test model (due to Rastrigin): the global solution value is -1 at x=0.

```
f2= x^2 - Cos[18x]; lb2 = -5; ub2 = 5; xnom2= 2;
```

In the following test function, the global solution value is -1.61642 at x=0.5.22406.

```
f3= Sin[x](Cos[x] - Sin[x])^2; lb3 = 0; ub3 = 10; xnom3 = 3;
```

The following test function is a polynomial: such functions are often used in tests. (The global solution value is 0 at x=1.)

```
f4= (2x^4 - 13x^3 + 18x^2 - 10x + 3)^2; lb4 = -1; ub4 = 4;
xnom4 = 3;
```

The following is a standard test model (due to Shubert): it is heavily oscillating, and the global solution value is -14.838 at x=-7.39728.

```
f5= Sum[k*Sin[(k+1)x+k], {k,1,5}]; lb5 = -10; ub5 = 10; xnom5 =
5;
```

The next test function has a number of local solutions, improving 'right to left': the global solution value is -0.943045 at x=-9.44999.

```
f6= 0.1x + Sqrt[Abs[x]]*Sin[x]^2; lb6 = -10; ub6 = 5; xnom6 =
-6;
```

The following test function is again a polynomial, with a quite simple (though non-convex) shape on the given optimization range. The global optimum is at 6.32565, its value is -4.43673.

```
f7= 0.01*(0.2*x^5 - 1.6995*x^4 + 0.998266*x^3 - 0.0218343*x^2 -
0.000089248*x); lb7 = 0; ub7 = 8; xnom7 = 2;
```

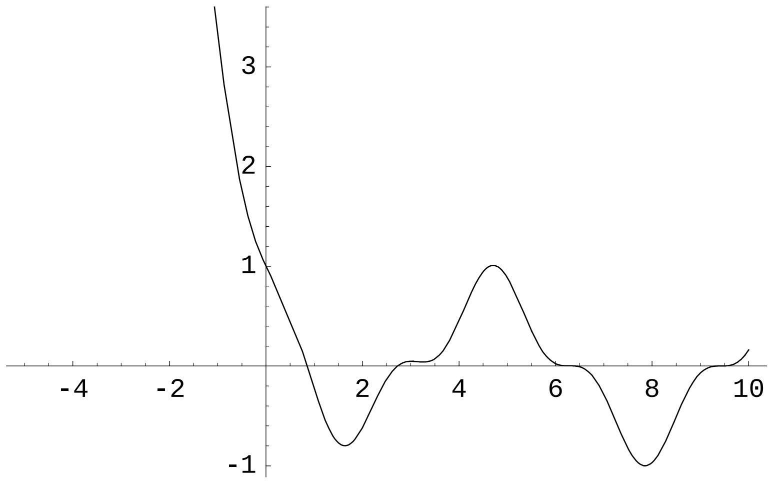
Here is another polynomial, with several local optima. The global optimum is at 1.75767, its value is -0.686072.

```
f8= 2*x^2 - 1.05*x^4 + 0.1666667*x^6 - x; lb8 = -2; ub8 = 2.5;
xnom8 = 1;
```

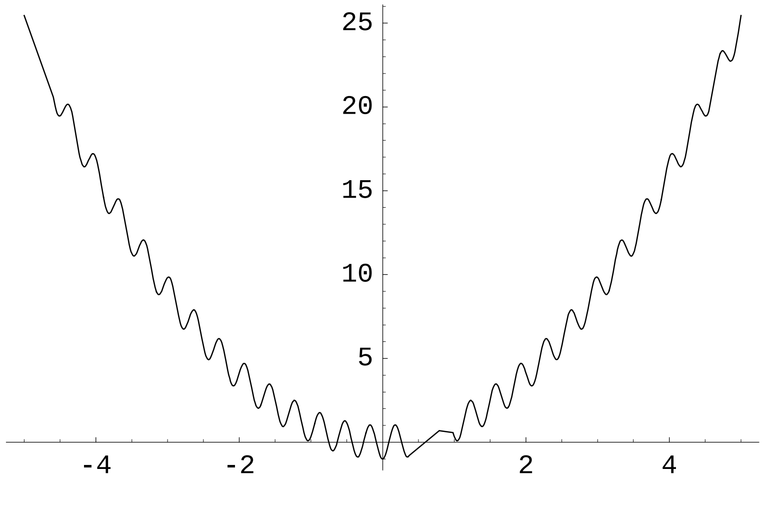
## Model Visualization

This is easy in the one-dimensional case. (Again, these models often serve to make the first step in verifying algorithm concepts.)

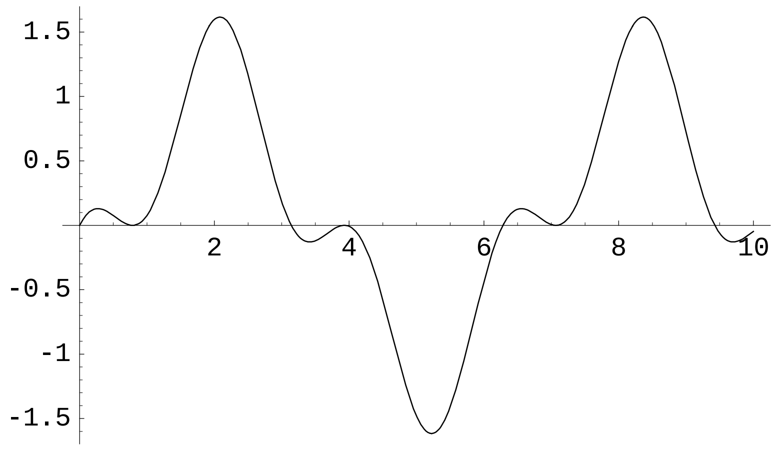
```
Plot[f1, {x, lb1, ub1}];
```



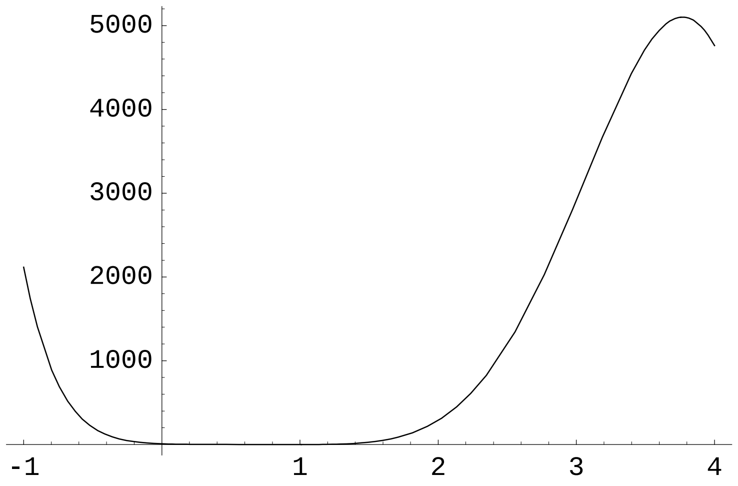
```
Plot[f2, {x, lb2, ub2}];
```



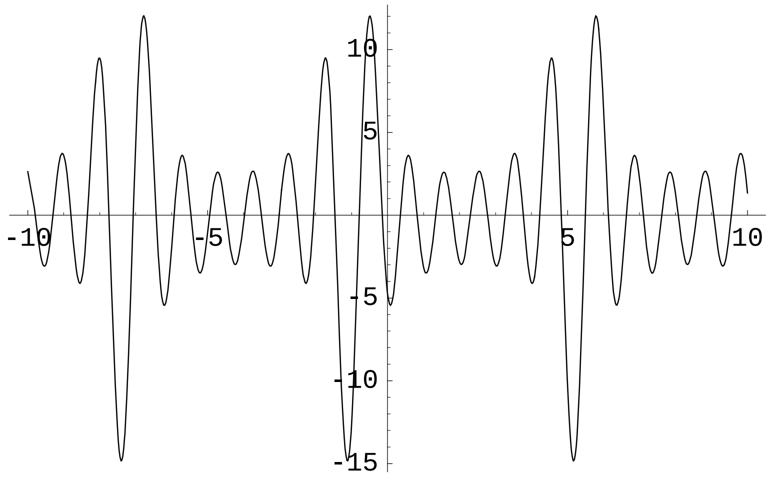
```
Plot[f3, {x, lb3, ub3}];
```



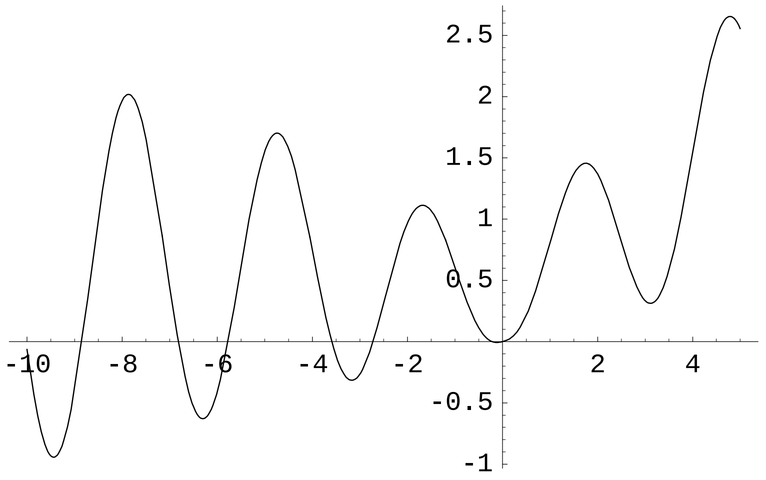
```
Plot[f4, {x, lb4, ub4}];
```



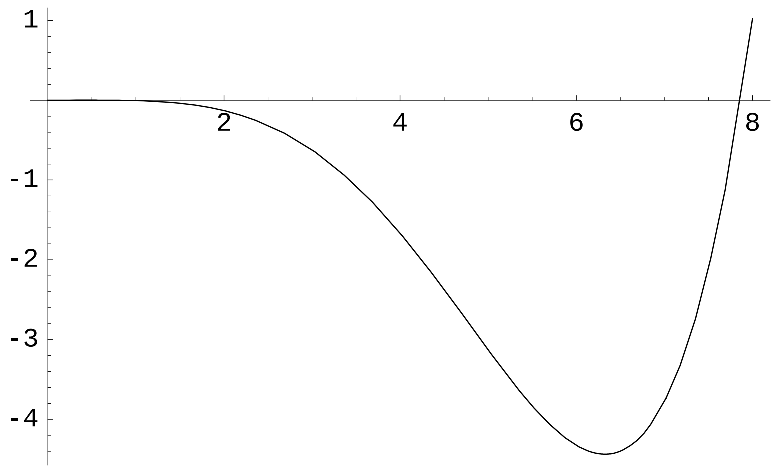
```
Plot[f5, {x, lb5, ub5}];
```



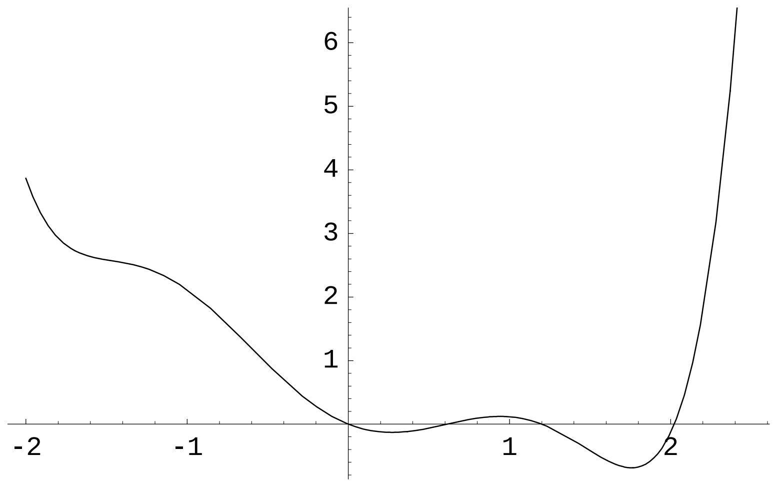
```
Plot[f6, {x, lb6, ub6}];
```



```
Plot[f7, {x, lb7, ub7}];
```



```
Plot[f8, {x, lb8, ub8}];
```



## Solving Illustrative Univariate Models by Local and Global Search Procedures

In all the models stated above, we set (an arbitrary) nominal solution; it could also be simply assumed to be the centre of the search region. (This is often a reasonable choice, if there is no additional information available, except the stated variable range.)

Below we also show timing information (as an exception), to indicate the relative runtimes.

```
result1 = CNLP[f1, {}, {}, {x}, {xnom1}, {lb1}, {ub1}]//Timing
result1 = MS[f1, {}, {}, {x}, {xnom1}, {lb1}, {ub1}]//Timing
result1 = Optimize[f1, {}, {}, {x}, {xnom1}, {lb1},
{ub1}]//Timing
```

```
{0.05 Second,
 {{1.63604}, -0.798881, {}, {}, {-∞, 1.98452×10^-14, 0}}}
{0.17 Second,
 {{7.84478}, -0.999481, -0.999578, -0.999481, {}, {}, 490}}
{0.22 Second,
 {{7.85411}, -0.999612, {}, {}, {-∞, 2.80557×10^-11, 0}}}
```

From the results shown above, one can see that (in this example)

- the local search per se is not sufficient (it misses the global solution)

- the global search phase already gives a reasonably good approximate solution (notice the small gap between the solution found and the estimated global bound)

- the combined search effort gives a more precise solution (with a relatively small added computational effort).

These findings are, in general terms, valid for many other global optimization problems. We shall illustrate this point, by solving also the other one-dimensional test models. (The Reader may like to revisit the corresponding pictures, to visually check the solutions found.)

```
result2 = CNLP[f2, {}, {}, {x}, {(lb2+ub2)/2}, {lb2}, {ub2}]
result2 = MS[f2, {}, {}, {x}, {(lb2+ub2)/2}, {lb2}, {ub2}]
result2 = Optimize[f2, {}, {}, {x}, {(lb2+ub2)/2}, {lb2},
{ub12}]
```

```
{{0.}, -1., {}, {}, {-∞, 0., 0}}
{{0}, -1, -1, -1, {}, {}, 490}
{{0.}, -1., {}, {}, {-∞, 0., 0}}
```

```
result3 = CNLP[f3, {}, {}, {x}, {(lb3+ub3)/2}, {lb3}, {ub3}]
result3 = MS[f3, {}, {}, {x}, {(lb3+ub3)/2}, {lb3}, {ub3}]
result3 = Optimize[f3, {}, {}, {x}, {(lb3+ub3)/2}, {lb3}, {ub3}]
```

```
{{5.22406}, -1.61642, {}, {}, {-∞, 4.49254×10^-8, 0}}
{{5.22191}, -1.61641, -1.61705, -1.61641, {}, {}, 490}
{{5.22406}, -1.61642, {}, {}, {-∞, 3.39045×10^-10, 0}}
```

```
result4 = Optimize[f4, {}, {}, {x}, {(lb4 + ub4) / 2}, {lb4}, {ub4}]
```

{{1.}, $3.5412\times10^{-27}$, {}, {}, {-∞, $5.9508\times10^{-13}$, 0}}

```
result5 = Optimize[f5, {}, {}, {x}, {(lb5 + ub5) / 2}, {lb5}, {ub5}]
```

{{-7.39728}, -14.838, {}, {}, {-∞, $1.07154\times10^{-6}$, 0}}

```
result6 = Optimize[f6, {}, {}, {x}, {(lb6 + ub6) / 2}, {lb6}, {ub6}]
```

{{-9.44999}, -0.943045, {}, {},
{-∞, $\sqrt{(-0.0549709 + 0.000103404\,\mathrm{Abs}'[-9.44999])^2}$, 0}}

```
result7 = Optimize[f7, {}, {}, {x}, {(lb7 + ub7) / 2}, {lb7}, {ub7}]
```

{{6.32565}, -4.43673, {}, {}, {-∞, $4.78973\times10^{-8}$, 0}}

```
result8 = Optimize[f8, {}, {}, {x}, {(lb8 + ub8) / 2}, {lb8}, {ub8}]
```

{{1.75767}, -0.686072, {}, {}, {-∞, $4.46512\times10^{-8}$, 0}}

One can conclude that at least these simple one-dimensional models do not pose a difficult challenge to *MathOptimizer*. Some more (or far more) difficult models will follow soon below.

## ■ Global and Local Search Approaches in Constrained Nonlinear Optimization

### Model Formulation

This is a non-convex, 2-variable, 2-constraint model. (The global optimum value is zero at x = 0.)

```
vars = {x1, x2};
varlb = Table [-10, {Length[vars]}];
varub = Table [20, {Length[vars]}];
varnom = {1, -1};  (* this 'sufficiently close guess' leads to
the global optimum... *)
objf = (2*x1^2-x2^2)^2 + (x2 - 3*x1^2)^2;
eqs = {x1 - 4*x2 - 5*Sin[2x1+3x2]};
ineqs = {x2 + x1 - 1};
```

### Appplication of Local Search

Let us first try to solve this model using (only) the local search method CNLP: if we have a good 'guess' of the solution, then — as a rule — a fast local scope search leads to the corresponding (local or maybe even global) optimum.

Again, solely for the purpose of illustrative comparison, we will also generate timing information.

```
resultCNLP = CNLP[objf, eqs, ineqs, vars, varnom, varlb,
varub]//Timing
```

```
{0.11 Second, {{6.54931×10^-16, -1.17658×10^-17}, 1.38434×10^-34,
  {-5.67083×10^-15}, {-1.}, {5.67083×10^-15, 2.87933×10^-14, 0.}}}
```

### Global Search Approach, Without Local Search

Let us try to solve now the same example by using only the global search method MS.

```
resultMS = MS[objf, eqs, ineqs, vars, varnom, varlb,
varub]//Timing
```

```
{1.43 Second, {{0.146901, -0.0710351}, 0.0206712,
  -0.0304424, 0.0198875, {0.0279942}, {-0.924134}, 3080}}
```

The solution found by MS (in general) does not have the same precision as the one attained by the local search method CNLP. Let us try to attain higher precision by enabling a more thorough global search. This is apparently quite succesful, but the solution time can also be a lot longer...

```
resultMS = MS[objf, eqs, ineqs, vars, varnom, varlb, varub,
MaxIterations->100, MaxSample->100 ]//Timing
```

```
{12.69 Second, {{-0.0489743, 0.0236219}, 0.000352839,
  -0.00772462, 0.000287799, {-0.00806477}, {-1.02535}, 27500}}
```

One can observe the following points.

1. As a general rule, the global scope solution procedure is slower, often much slower (depending on parameter settings).

2. More search effort does not necessarily lead to better solutions due to the stochastic search component in MS, since the sample points will be different in each run. Note that this negative feature is remedied, by using the RandomSeed default option (in which case the random number generator is pre-seeded).

3. Unless a significant (perhaps inordinate) amount of search effort is allocated to the global solution method, the accuracy of the solution is less than by using local search from a 'good' initial point. In fact, the global search method does not rely crucially on the initial solution 'guess', as shown below.

### Model Sensitivity to Initial Solution Changes: The Importance of Global Search

Let us change the initial solution guess, and observe the solution changes!

```
varnom = {-2, 3}; (* this leads to a local optimum, when using
CNLP... *)
resultCNLP2 = CNLP[objf, eqs, ineqs, vars, varnom, varlb,
varub]//Timing
```

```
{1.48 Second, {{0.65855, 0.774751}, 0.34837, {-0.0443739},
  {0.433301}, {0.433301, 4.06583×10^-6, 49.1793}}}
```

```
varnom = {5, -8}; (* this leads to an infeasible solution, when
using CNLP... *)
resultCNLP3 = CNLP[objf, eqs, ineqs, vars, varnom, varlb,
varub]//Timing
```

```
{1.43 Second, {{-4.28779, -2.81537}, 4192.59,
  {2.13799}, {-8.10316}, {2.13799, 0.000269123, 0.}}}
```

At the same time, the global search method will return 'reasonable' solutions (as a rule, at least for well-scaled models), since in fact it does not depend essentiially on the starting solution.

Let us try to solve the model again, with the current 'bad' starting solution: the solution quality is typically similar to the one found by a similar global search effort. ( (Actually, the results will be identical, if the same search options, including the default RandomSeed option are used.)

```
resultMS2 = MS[objf, eqs, ineqs, vars, varnom, varlb,
varub]//Timing
```

```
{1.37 Second, {{0.146901, -0.0710351}, 0.0206712,
  -0.0304424, 0.0198875, {0.0279942}, {-0.924134}, 3080}}
```

### Combined Global and Local Search

Let us activate now first the MS global optimization procedure, followed by the local search procedure CNLP. (Default usage options are used in both cases).

```
Optimize[objf, eqs, ineqs, vars, varnom, varlb, varub]//Timing
```

{1.48 Second, {{$1.0985\times10^{-11}$, $-3.92067\times10^{-18}$}, $1.53745\times10^{-35}$,
{$-9.88652\times10^{-11}$}, {-1.}, {$9.88652\times10^{-11}$, $5.5676\times10^{-15}$, 0.}}}

As this example illustrates, prudent combinations of good quality global and local solvers will typically lead to superior performance (in terms of overall robustness and efficiency) when compared to either pure global or pure local scope solvers.

## ■ Randomly Generated Global Optimization Test Models

### Function Class Definition

It is often useful to test optimization algorithms on randomly generated models with known (preset or randomly set) solutions. The following example shows such a model class.

```
(* decision variables *)
vars = {x1, x2, x3};

(* number of decision variables; needed only for easier model
setup *)
nvars = Length[vars];

(* finite lower bounds; component-wise scaling to 0 is
recommended *)
varlb = Table[0, {nvars}];

(* finite upper bounds; component-wise scaling to 1 is
recommended  *)
varub = Table[1, {nvars}];

(* randomly generated solution to test problem *)
sol = Table[Random[], {nvars}];

(* objective function *)
objf = (x1 - sol[[1]])^2 + 2*(x2 - sol[[2]])^2 + 3*(x3 -
sol[[3]])^2;

(* list of equality constraints *)
eqs = {
    Sin[5*(x1 - sol[[1]])] - 12*(x3 - sol[[3]])*(x2 -
sol[[2]]),
    Exp[(x1 - sol[[1]])*(x2 - sol[[2]])*(x3 - sol[[3]])]-1
    };

(* list of inequality constraints *)
ineqs = {
    ((x1-sol[[1]])*(x2-sol[[2]])*(x3-sol[[3]]))^2 +
    Sin[ 15*(x1*x2*x3 - sol[[1]]*sol[[2]]*sol[[3]]) ] - 0.05
    };

(* define nominal solution; default setting: search range
centre *)
varnom = (varlb+varub)/2;

(* penalty multiplier *)
penmult = 1;

(* aggregated merit (exact penalty) function *)
meritfct = objf + penmult * Sum[ eqs[[i]]^2, {i, Length[eqs]}]
+ penmult * Sum[ Max[ineqs[[i]],0]^2, {i, Length[ineqs]}];
```

## Model Verification

Model checking is an important step in the development process, since most of us make mistakes when building complex models.

The simplest check is to see that the model functions evaluate correctly at least at the given nominal solution(s). It is also highly recommended to verify model evaluations at the lower and upper bounds, and at least at a few other 'arbitrary' interior points of the embedding 'box' region.

```
(* determine merit function value at nominal optimum estimate *)
meritfctnom = meritfct /. Thread[vars->varnom];

(* test only: check optimum value at test model solution sol *)
optval = meritfct /. Thread[vars -> sol];
(* test printouts *)
Print["randomly generated optimal solution ", sol];
Print["initial nominal (centre) solution ", varnom];
Print["initial nominal function value ", meritfctnom];
Print["globally optimal function value ", optval];
```

```
randomly generated optimal solution
 {0.370966, 0.782398, 0.597549}
```

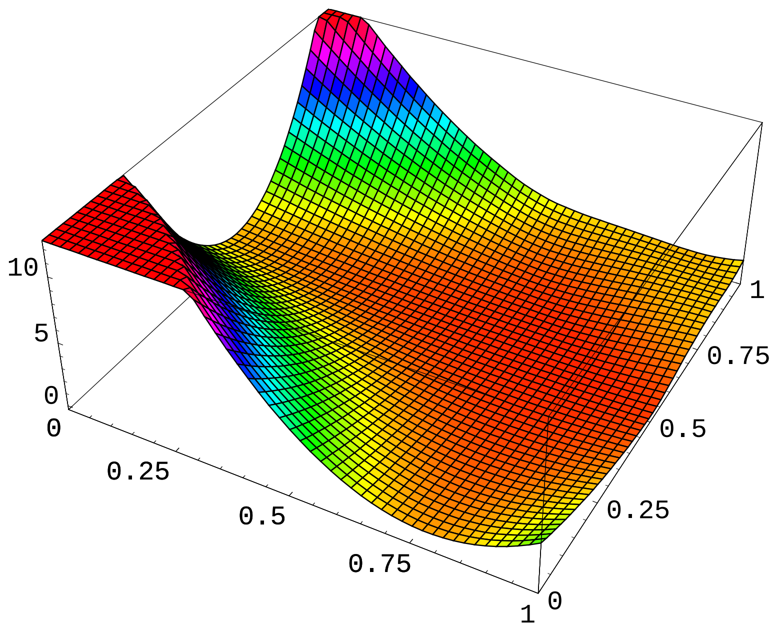
initial nominal (centre) solution $\left\{\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right\}$

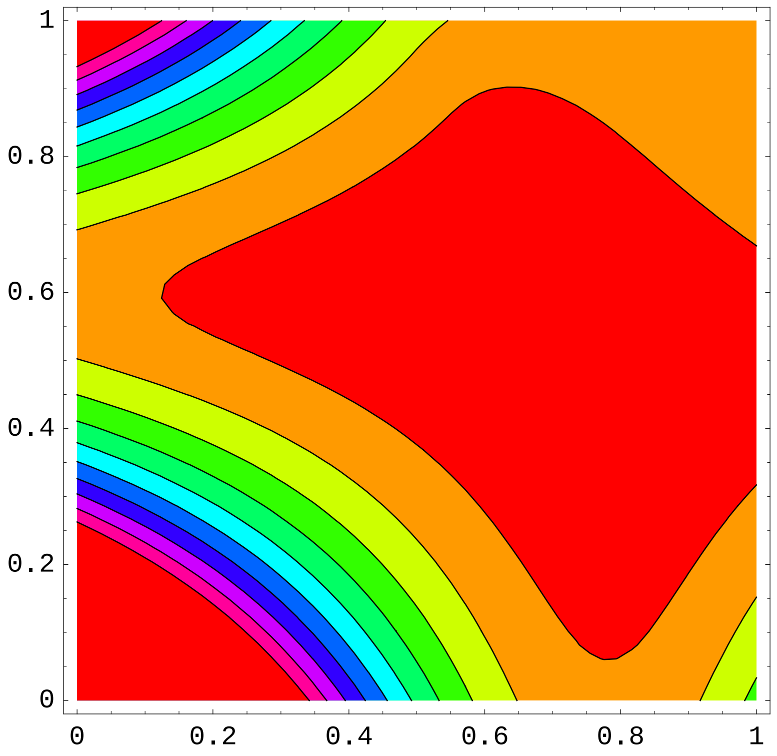
```
initial nominal function value 0.278018

globally optimal function value 0.
```

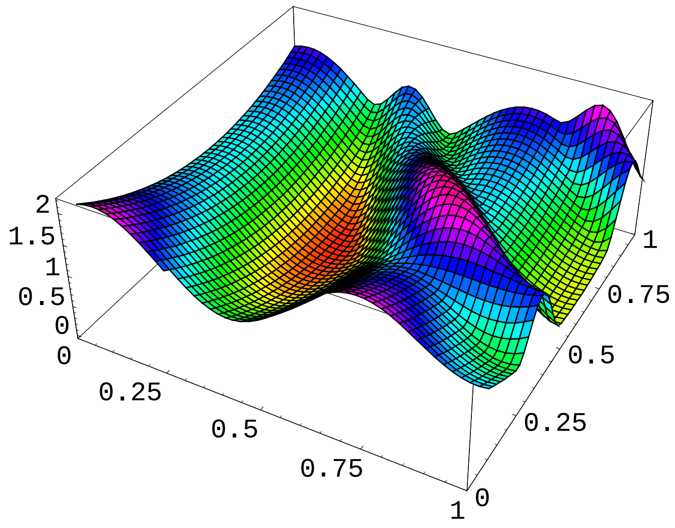
## Model Visualization

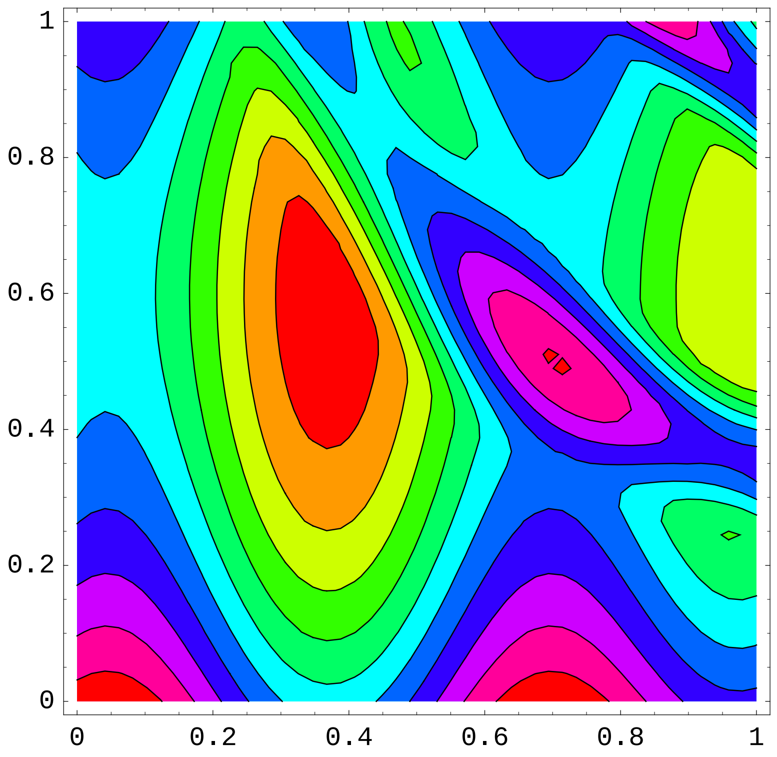
Models can be visualized, to analyze expected model complexity. Subspace projections are generated by the next statements.

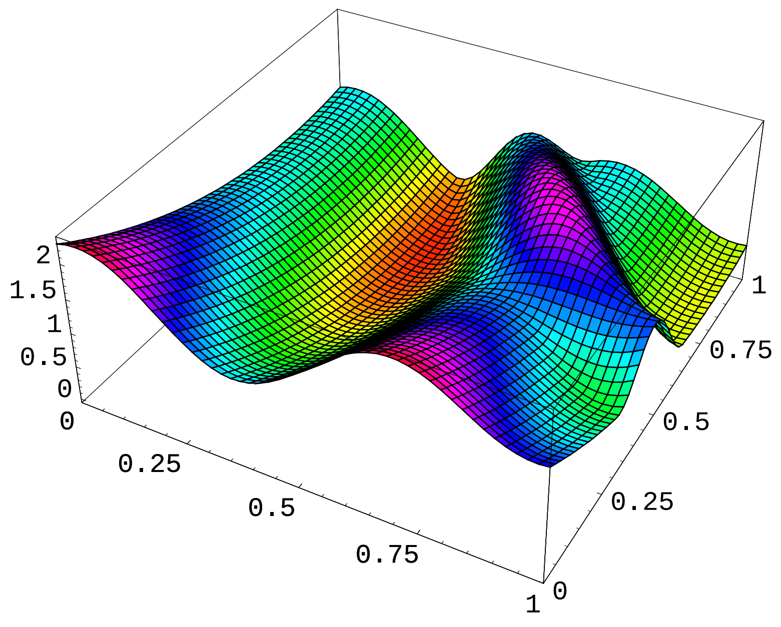
```
Plot3D[meritfct /. x1->sol[[1]], {x2,0,1}, {x3,0,1},
PlotPoints->50, ColorFunction -> Hue];
ContourPlot[meritfct /. x1->sol[[1]], {x2,0,1}, {x3,0,1},
PlotPoints->50, ColorFunction ->Hue];

Plot3D[meritfct /. x2->sol[[2]], {x1,0,1}, {x3,0,1},
PlotPoints->50, ColorFunction -> Hue];
ContourPlot[meritfct /. x2->sol[[2]], {x1,0,1}, {x3,0,1},
PlotPoints->50, ColorFunction -> Hue];

Plot3D[meritfct /. x3->sol[[3]], {x1,0,1}, {x2,0,1},
PlotPoints->50, ColorFunction -> Hue];
ContourPlot[meritfct /. x3->sol[[3]], {x1,0,1}, {x2,0,1},
PlotPoints->50, ColorFunction -> Hue];
```

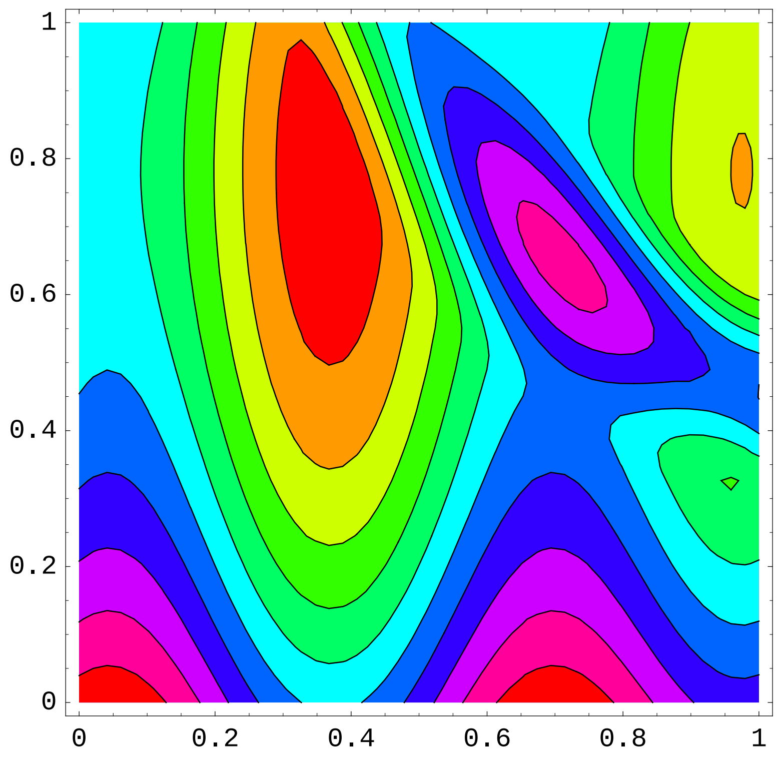

10
5
0
0
0.25
0.5
0.75
1
0
0.25
0.5
0.75
1


1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1


2
1.5
1
0.5
0
0
0.25
0.5
0.75
1
0
0.25
0.5
0.75
1


1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1





## Solving Random Test Examples by Combined Global and Local Search

Although the random models above are not too complicated, they are not trivial either.

Let us solve now the current model instance. A comparison of the result obtained (see below) with the randomly generated solution (displayed above, in the Model Verification subsection) indicates that the methods in *MathOptimizer* seem to work well, at least for these small randomly generated test models.

```
Optimize[objf, eqs, ineqs, vars, varnom, varlb, varub]
```

```
{{0.370966, 0.782398, 0.597549}, 1.53084×10^-20,
 {6.18636×10^-10, 0.}, {-0.05}, {6.18636×10^-10, 5.77464×10^-14, 0.}}
```

## ■ Constrained Global Optimization: Further Examples

### Hock & Schittkowski, Problem 62

The following model is test problem 62 from the collection by Hock and Schittkowski (1981), cited also in other test collections and studies.

```
vars = {x1, x2, x3};
varlb = Table [0, {Length[vars]}];
varub = Table [1, {Length[vars]}];
varnom = {0.5, 0.5, 0.5};
objf = -32.174*(255.*Log[(x1+x2+x3+0.03)/(0.09*x1+x2+x3+0.03)]
+ 280*Log[(x2+x3+0.03)/(0.07*x2+x3+0.03)] +
290*Log[(x3+0.03)/(0.13*x3+0.03)]);
eqs = {100*(x1 + x2 + x3 - 1)};  (* explicit constraint scaling
is used, to force feasibility *)
ineqs = {};
```

```
Optimize[objf, eqs, ineqs, vars, varnom, varlb, varub]
```

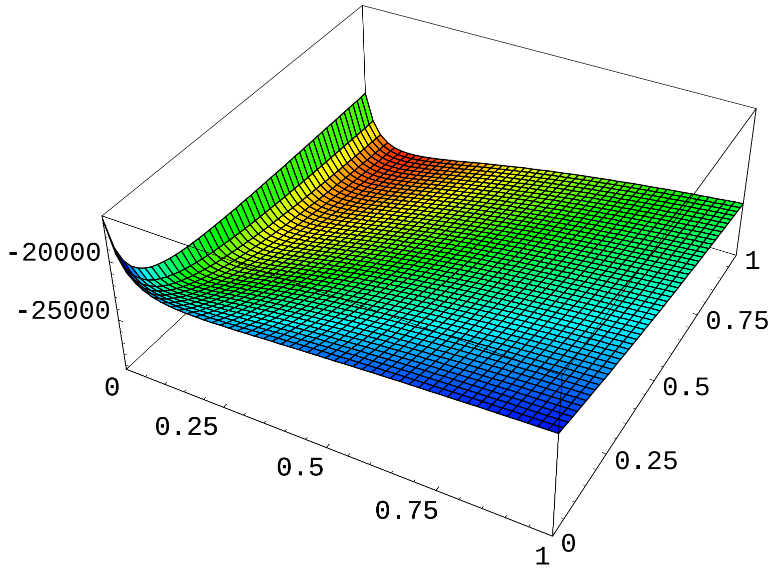
```
{{0.617813, 0.328202, 0.0539851}, -26272.5,
 {-1.00975×10^-7}, {}, {1.00975×10^-7, 0.000120306, 0}}
```

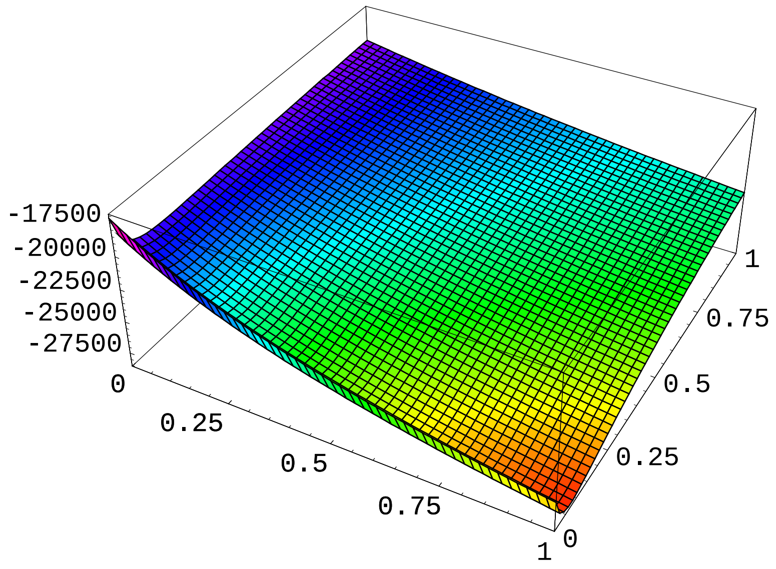
The result found above is in good accordance with the solution cited by Hock and Schittkowski. (The relative difference of the two results is about 10^-6, in terms of the objective function values found. As the model visualization exercise below shows, the model is not changing rapidly' around the optimum estimate: this can also make a precise numerical solution rather difficult.)

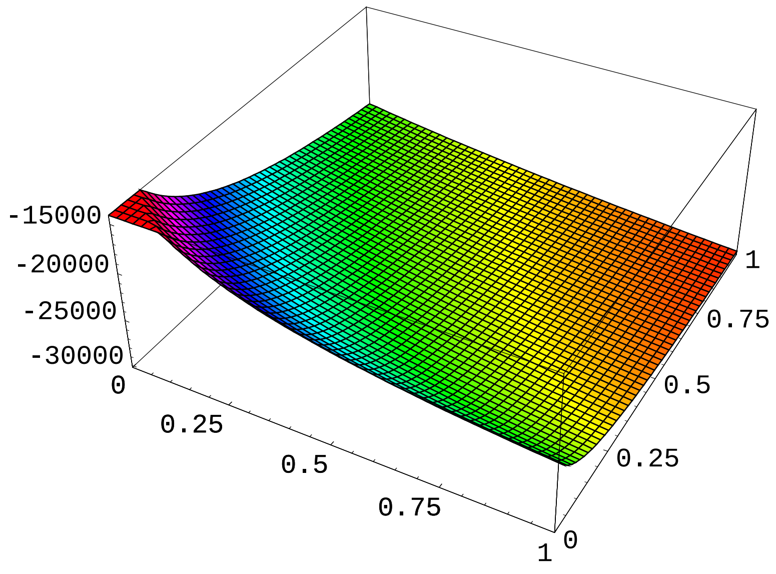
### Model Visualization

The pictures of an appropriately 'back-scaled' merit function ar shown below.

```
penmult = 0.01;
meritfct = objf + penmult * Sum[ eqs[[i]]^2, {i, Length[eqs]}];
xsol ={0.617813, 0.328202, 0.0539851};

Plot3D[meritfct /. x1->xsol[[1]], {x3,0,1}, {x2,0,1},
PlotPoints->50, ColorFunction -> Hue];
Plot3D[meritfct /. x2->xsol[[2]], {x1,0,1}, {x3,0,1},
PlotPoints->50, ColorFunction -> Hue];
Plot3D[meritfct /. x3->xsol[[3]], {x1,0,1}, {x2,0,1},
PlotPoints->50, ColorFunction -> Hue];
```


-20000
-25000
0
0.25
0.5
0.75
1
0
0.25
0.5
0.75
1


-17500
-20000
-22500
-25000
-27500
0
0.25
0.5
0.75
1
0
0.25
0.5
0.75
1



## A Concave Quadratic Programming Model

This classical test problem is due to Hesse (see also, e.g., Floudas et al., p. 24.) The model has 6 variables and 6 constraints, in addition to the range constraints which need to be explicitly considered in this case (since — due to the model structure — several solution vector components are at their lower or upper bounds).

The global solution value is -310 which is attained on the boundary of the feasible region; there are also other (local) solutions on the boundary.

Note the penalty multiplier (penmult) used with respect to the set of all constraints, to 'scale them up' to the objective function: this helps to avoid 'runaway' solutions in the local search phase.

```
vars = {x1, x2, x3, x4, x5, x6};
varlb = {0, 0, 1, 0, 1, 0};
varub = {10, 10, 5, 6, 5, 10};
varnom = {1, 1, 1, 1, 1, 1};
objf = -25*(x1-2)^2 - (x2-2)^2 - (x3-1)^2 - (x4-4)^2 - (x5-1)^2
- (x6-4)^2;
eqs = {};
ineqs = {-x1, x1-6, -x2, x2-6, 1-x3, x3-5, -x4, x4-6, 1-x5,
x5-5, -x6, x6-10,
(-(x3-3)^2-x4+4)/4,
(-(x5-3)^2-x6+4)/4,
(x1-3x2-2)/2,
(-x1+x2-2)/2,
(x1+x2-6)/6,
(-x1-x2+2)/2};
penmult = 100;
ineqs = penmult*ineqs;
meritfct = objf + penmult * Sum[ eqs[[i]]^2, {i, Length[eqs]}]
+ penmult * Sum[ Max[ineqs[[i]],0]^2, {i, Length[ineqs]}];
```

As the solution obtained below by CNLP illustrates, local search per se is not sufficient.

```
varnom = {1, 1, 1, 1, 1, 1};
CNLP[objf, eqs, ineqs, vars, varnom, varlb, varub]
```

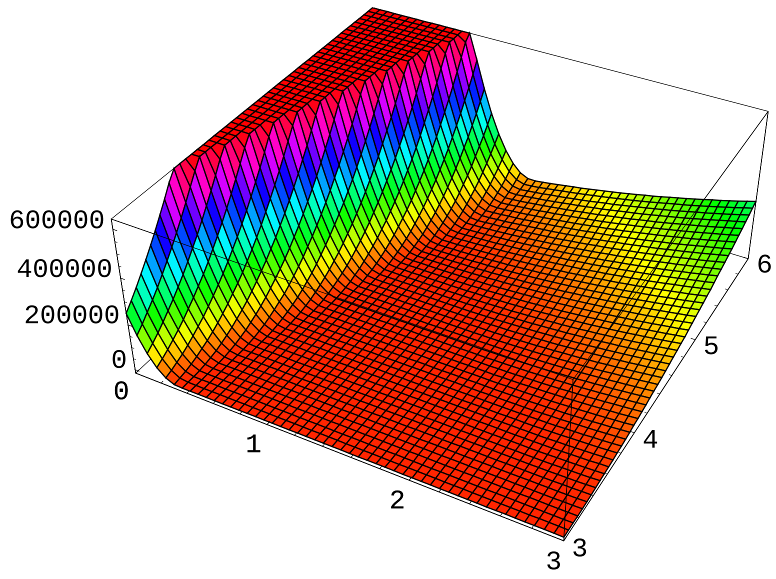
```
{{-8.67393×10^-13, 2., 1., -2.07525×10^-12, 1., -2.07525×10^-12},
 -132., {}, {8.67393×10^-11, -600., -200., -400., 5.30231×10^-10,
  -400., 2.07525×10^-10, -600., 5.30231×10^-10, -400.,
  2.07525×10^-10, -1000., -4.78361×10^-10, -4.78361×10^-10,
  -400., -5.3535×10^-10, -66.6667, 6.2208×10^-10},
 {6.2208×10^-10, 1.24847×10^-6, 1.19936×10^-10}}
```

```
Optimize[objf, eqs, ineqs, vars, varnom, varlb, varub]
```

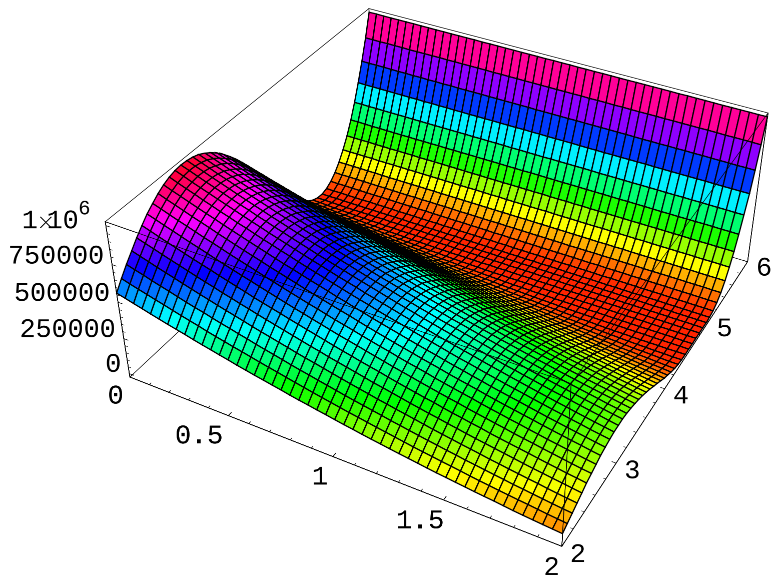
```
{{5., 1., 5., -5.19599×10^-12, 5., 10.},
 -310., {}, {-500., -100., -100., -500., -400.,
  2.41752×10^-8, 5.19599×10^-10, -600., -400.,
  -2.51621×10^-9, -1000., -4.26255×10^-9, -2.40453×10^-8,
  -250., 3.95306×10^-9, -300., 9.66259×10^-8, -200.},
 {9.66259×10^-8, 7.41437×10^-10, 6.53593×10^-7}}
```

**Model Visualization**

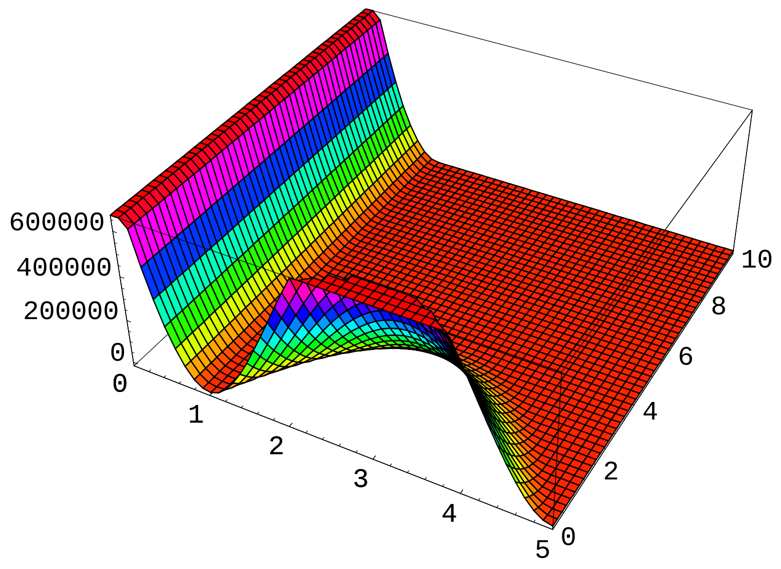
```
xsol ={5., 1., 5.,0., 5., 10.};
Plot3D[meritfct /. Thread[vars->xsol], {x2,0,3}, {x1,3,6},
PlotPoints->50, ColorFunction -> Hue];
```



```
Plot3D[meritfct /. Thread[vars->xsol], {x4,0,2}, {x3,2,6},
PlotPoints->50, ColorFunction -> Hue];
```



```
Plot3D[meritfct /. Thread[vars->xsol], {x5,0,5}, {x6,0,10},
PlotPoints->50, ColorFunction -> Hue];
```



## Global Optimization: Some More Difficult Challenges and Other Illustrative Applications

## A Simple Industrial Design Problem

### ■ Model Formulation

This is a standard nonlinear programming test model: consult, for instance, the LINGO Model Library, maintained by LINDO Systems, Inc.

Our task is to design a box at minimum cost that meets (expected functionality related) area and volume constraints, as well as certain aesthetic requirements.

Decision variables:

d --- depth of box

w --- width of box

h --- height of box

Cost min = 0.1*(d*w + d*h) + 0.2*w*h; (by assumption, the material cost is twice as much for two of the sides, than for all the others)

|  |  |  |
|---|---|---|
| Surface | 2*(h*d + h*w + d*w) >= 888; | (should be sufficiently large) |
| Volume | h*d*w >= 1512; | (should be sufficiently large) |
| Aesthetics | h/w <= 0.718; | (proportionality bound) |
|  | h/w >= 0.518; | (proportionality bound) |
|  | d*w <= 252; | (sufficiently small footprint required) |
|  | 10 <= d <= 300; | ('safe' variable range constraints are suggested here: |
|  | 10 <= w <= 300; | it is easy to give more narrow ranges, but these |
|  | 10 <= h <= 300; | are also sufficiently 'tight' for successful solution...) |

Note that several constraints could be simplified (namely, the proportionality constrains can be linearized); we simply follow here the original model form as given in the LINGO Model Library.

## ■ Numerical Solution

It is straightforward to set up the model, using variables as above.

```
Clear[d, h, w];
vars = {d, h, w};
varnom = Table[50., {3}];
varlb = Table[10., {3}];
varub = Table[300., {3}];
eqs = {};
ineqs = {-2*(h*d + h*w + d*w) + 888, -h*d*w + 1512., h/w -
0.718, -h/w + 0.518, d*w - 252.};
objf = 0.1*(d*w + d*h) + 0.2*w*h;
result = Optimize[objf,eqs,ineqs,vars,varnom,varlb,varub,
    GlobalSolverMode -> 1, LocalSolverMode -> 1, ReportLevel ->
1]
```

```
{{23.031, 6.86566, 9.56219}, 50.9651, {},
 {-1.36165×10^-8, -3.10472×10^-8, 1.10654×10^-7, -0.2, -31.7734},
 {1.10654×10^-7, 0.0000888699, 2.53612×10^-7}}
```

The more exact optimal objective function value found above is 50.965074987088876.

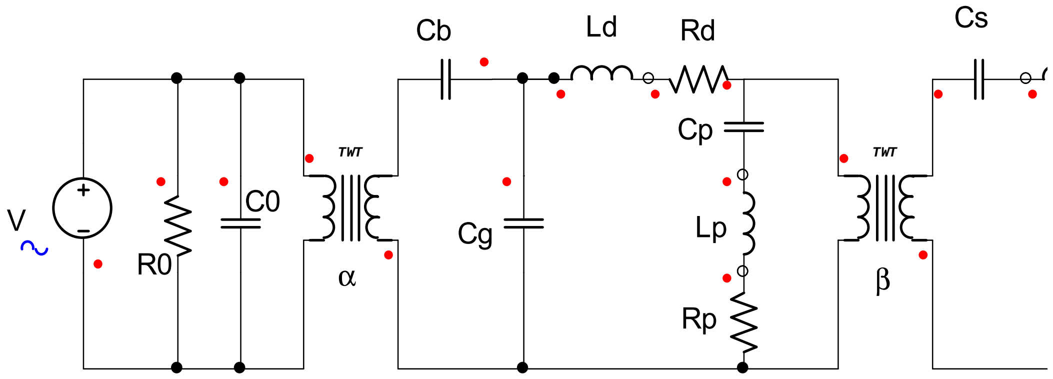
The result obtained is essentially identical (to great numerical precision) to the result found by using the LINGO solver.

# A Sonar Transducer Design Problem

## ■ Summary Description

The electric circuit shown below simulates a piezoelectric flextensional sonar projector. The circuit was developed by Moffett et al. (1992), to describe the barrel stave projector.

This is an equivalent circuit model: that is, electric circuit components simulate the mass, compliance and damping of mechanical components by corresponding inductances, capacitances, and resistances.



The conversion from mechanical to electrical degrees of freedom is handled by a transformer (the parameter $\alpha$ represents its transformer turns ratio); two other transformers (modelled by the turns ratios $\beta$ and $\gamma$) represent the flextensional mechanical transformers.

The optimization problem consists of finding a set of design parameters that gives a 'nice' broad efficiency (versus frequency), measured as the ratio of radiated acoustic power (delivered to the radiation resistance Rr) to input electric power provided by the voltage source V. The ideal efficiency curve is taken to be given by a Hamming window function, a distribution commonly seen in signal processing applications.

For the purposes of this illustrative example, some of the design parameters have been fixed, but 7 parameters have been left 'free', and hence these have to be optimized. The design parameters have been normalized so that the frequency range of interest is assumed to be between 0 and 1 Hz. The parameters belong to given finite ranges; there are no other (equality or inequality) constraints.

The objective function is defined by a somewhat complicated 'black box' system model given below. The *Mathematica* code for this model was written by C.J. Purcell, DRDC-Atlantic. Note that — for easier understanding — some of the key components of the code are shown in blue.

## ■ Model Formulation

```
Clear[goal,objf,objective,C0,R0,Rs,Rl,Rd,Rr,Lr,Cb,Cg,α,β,γ,Cs,Ls
,Ll,Cl,Cd,Ld];
```

Maximized Hamming window function is our goal for the ideal broad band efficiency distribution; this is combined with the efficiency calculated in the 'black box' module below in an overall objective function.

Since the standard *MathOptimizer* model form is aimed at minimizing the objective function, the maximization objective is simply multiplied by -1.

```
goal[x_]:=N[1-Cos[Pi x]^2];

objective[β_,Cs_,Ls_,Ll_,Cl_,Cd_,Ld_]:=
- Sum[goal[f] * Module
[{Y,w=N[2 Pi f],Zr,Z1,Z2,Z3,V1,V2,V3,I0,I1,V0=1,
powerout,powerin},(  (* Begin Module calculations *)
Zr=1/(1/Rr+1/(w I Lr));    (* impedance to gnd at Rr *)
Z1=1/(w I Cs) + w I Ls + Rs+ Zr/γ^2; (* impedance to gnd at
input to Cs *)
(* impedance to gnd at input to beta transformer *)
Z2=1/(1/(Rl+w I Ll+1/(w I Cl)) + β^2/Z1);
Z3=1/(w I Cg)+ 1/(w I Cb+ 1/(Rd+1/(w I Cd)+ w I Ld + Z2));
(* impedance at input to Cg *)
Y=(1/R0 + w I C0 +α^2/Z3); (* input admittance *)
I0= Y * V0; (* the input current *)
V1= α V0(1-1/(I w Cg Z3));
V2= V1 * Z2/(Z2+Rd+1/(w I Cd)+w I Ld);
V3=γ β V2 *(Zr/γ^2)/(Zr/γ^2+Rs+w I Ls+1/(w I Cs));
powerout=Re[V3 Conjugate[V3/Rr]]; (* acoustic power Watts *)
powerin=Re[V0 Conjugate[I0]]; (* input electrical power Watts *)
powerout/powerin ) (* return value: relative efficiency *)
],{f, 0.1, 1., 0.01}]; (* End of Module calculations *)

(* Constants *)
{C0,R0}={.5, 10^5};
{Rs,Rl,Rd}={.01, .01, .01};
{Rr,Lr}={.01, .2};
{Cb,Cg}={20., 20.};
{α,γ}={.01,.06};

(* Optimization parameters: β, Cs, Ls, Ll , Cl, Cd, Ld. *)
vars = {β,Cs,Ls,Ll,Cl,Cd,Ld};
varnom = Table[.1,{Length[vars]}];
varlb = Table[0.001,{Length[vars]}];
varub = Table[5.0,{Length[vars]}];

(* Objective function *)
objf = objective[Apply[Sequence,vars]];
eqs = {};
ineqs = {};
```

## ■ Model Verification

It is always good practice to check the model structure by providing first some test input (at least trying the nominal setting as well as the bounds), just to see that calculations coded lead to meaningful results. This is especially important in somewhat more complicated models.

```
objective[Apply[Sequence,varnom]]

-0.0526404

objective[Apply[Sequence, varlb]]

-9.68541×10^-10

objective[Apply[Sequence, varub]]

-0.0885071
```

## ■ Numerical Solution

In order to avoid numerical issues with respect to this particular model (due to symbolic gradient manipulations within the Module above), for illustration we apply here only the (gradient-free) global search mode MS.

The solution of this numerically rather intensive model takes about 12 minutes on an Intel P4 1.6 GHz processor based machine. According to the result returned (see below), 7952 MS search steps were completed during runtime: this means about 10.73 objective function evaluations per second.

```
resultMS = Optimize[objf, eqs, ineqs, vars, varnom, varlb, varub,
   GlobalSolverMode → 1, LocalSolverMode → 0] // Timing

{740.765 Second,
 {{4.43451, 3.08985, 0.539311, 3.09438, 3.27806, 3.58907,
   0.00103298}, -42.794, -43.2231, -42.794, {}, {}, 7952}}
```

The globally optimized performance indicator value (changing its sign, since we aimed at maximizing this value) equals 42.794: note that this is a very significant improvement over the settings tried at the model verification step. The statistically generated (in the present context, upper) bound value 43.2231 shows that — most likely — the result obtained is close to the global optimum.

A more precise analysis should also incorporate a local search in which numerical gradient approximations could be utilized.

# A Chemical Equilibrium Problem

## ■ Model Formulation

This model was proposed by Peters, Hayes and Hieftje, and it is often used as a test problem: consult, for instance, the LINGO Model Library, maintained by LINDO Systems, Inc.

In chemical equilibrium problems, one needs to determine the contribution rates of various components to mixtures which have given (known or  prescribed) chemical characteristics.

In the example below we want to determine the concentrations of several components of phosphorus acid such that the resulting pH value equals 8, and the total phosphate concentration is 0.1.

Note that such models are often are (very) poorly scaled. For scaling reasons it is often better to take logarithms of the equations: however, the suitability of various — theoretically equivalent — formulations may depend significantly also on the solvers applied.

The equilibrium equations are:

Log[H2P]+Log[H]-Log[H3P]=Log[0075];

Log[HP]+Log[H]-Log[H2P]=Log[6.2*10^(-8)];

Log[H]+ Log[P]-Log[HP]=Log[4.8*10^-13];

Log[H]=Log[10^-8];

H3P+H2P+HP+P=0.1;

The approximate solution of this model (according to LINGO Model Library information) is known to be

Log[H]= -18.4207

Log[P] = -12.3965

Log[HP] = - 2.4522

Log[H2P] = -4.2767

Log[H3P] = -17.8045

## ■ Numerical Solution

All variable names will be started (in fact, entirely denoted) by small letters, following *Mathematica* conventions.

```
Clear[h, p, hp, h2p, h3p];

vars = {p, hp, h2p, h3p};
varnom = Table[.01, {4}];
varlb = Table[10^-20, {4}];
varub = Table[0.1, {4}];

h = 10^-8;   (* deduced directly from the fourth equation *)

(* Logarithmic model form: leads to numerical errors due to
poor scaling...
eqs = {
Log[h2p] + Log[h] - Log[h3p] - Log[.0075],
Log[hp] + Log[h] - Log[h2p] - Log[6.2 * 10^-8],
Log[h] + Log[p] - Log[hp] - Log[4.8 * 10^-13],
h3p + h2p + hp + p - 0.1
};
*)

(* Natural scale model form: seems to work better for our
current set of solvers... *)
eqs =
{h2p*h - 0.0075*h3p,
hp*h - 6.2*10^(-8)*h2p,
h*p - 4.8*10^(-13)*hp,
h3p + h2p + hp + p - 0.1
};

(* ineqs = Flatten[{varlb - vars, vars - varub}]; *)
(* One has to add these constraints, to avoid numerical errors
due to poor model-scaling... *)
ineqs = {varlb - vars};

(* The objective is to satisfy all equations simultaneously *)
objf = Sum [eqs[[i]]^2, {i,4}];
```

```
Optimize[objf,eqs,ineqs,vars,varnom,varlb,varub,
    GlobalSolverMode -> 1, LocalSolverMode -> 1, ReportLevel ->
1,
    ConvergenceLevel->12, PrecisionLevel->20]
```

```
{{0.0393242, 0.0508666, 0.00980916, 1.27395×10^-8}, 1.6462×10^-19,
 {2.54529×10^-12, -9.95018×10^-11, 3.93218×10^-10, -9.62085×10^-12},
 {{-0.0393242, -0.0508666, -0.00980916, -1.27395×10^-8}},
 {3.93218×10^-10,
  √((-8.32932×10^-13 + {{{-1, 0, 0, 0}}, {{0, -1, 0, 0}},
          {{0, 0, -1, 0}}, {{0, 0, 0, -1}}}.{0.})^2 +
     (-6.05524×10^-13 + {{{-1, 0, 0, 0}}, {{0, -1, 0, 0}},
          {{0, 0, -1, 0}}, {{0, 0, 0, -1}}}.{0.})^2 +
     (-6.05415×10^-13 + {{{-1, 0, 0, 0}}, {{0, -1, 0, 0}},
          {{0, 0, -1, 0}}, {{0, 0, 0, -1}}}.{0.})^2 +
     (-6.05366×10^-13 + {{{-1, 0, 0, 0}}, {{0, -1, 0, 0}},
          {{0, 0, -1, 0}}, {{0, 0, 0, -1}}}.{0.})^2), 0.}}
```

For comparison with the values from the LINGO Library, take the logarithm of the values found.

```
Log[{0.017960613045001034, 0.047525366787566775,
  0.03451401759870644, 2.5710327551048025*^-9}]
```

```
{-4.01957, -3.04649, -3.36639, -19.779}
```

One can see that the result is visibly (but not qualitatively) different from the values cited above from the literature. However, as the detailed list returned by the *MathOptimizer* solvers shows (see above), the chemical equilibrium equations are satisfied to high precision. This indicates that the model could have a multitude of 'near-solutions' of similar quality. The individual scaling of the constraints would also most likely help to improve the precision even further. A potential energy model based genuine objective function can assist further — and, in fact, is often used — to select the 'absolutely best' solution.

# The Hundred-dollar, Hundred-digit Challenge Problems: Problem 4

## ■ Problem Statement and Preliminary Analysis

This challenge was posted by N. Trefethen, Oxford University, UK in the SIAM News January - February 2002; page 3.

**Find the global minimum of the two-variable real function f(x,y) defined below as**

$$\frac{1}{4}(x^2 + y^2) + e^{\sin(50\,x)} - \sin(10\,(x + y)) + \sin(60\,e^{y}) + \sin(70\sin(x)) + \sin(\sin(80\,y))$$

**No explicit variable bounds are provided.**

One can immediately observe that since the function f(x,y) is made up by two non-negative terms and four trigonometric terms, its global minimum has to be greater than -4.

A second elementary observation is that the value of this function tends to infinity if the absolute value of x and/or y becomes large. Therefore the optimum shall be located in a suitable neighbourhood of the origin.

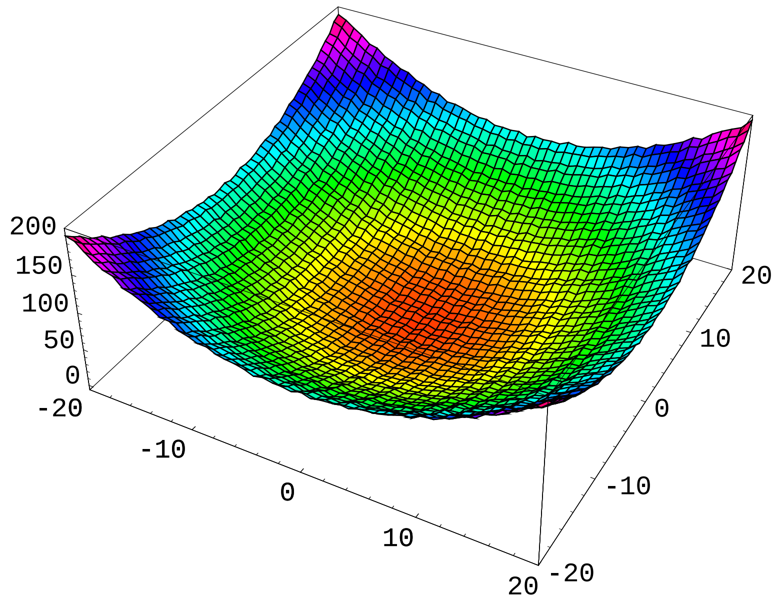
Let us define our objective function:

```
Clear[x, y, objf];

objf = 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;
```

In an attempt to find reasonable bounds for x and y, we shall generate a few pictures of this function.

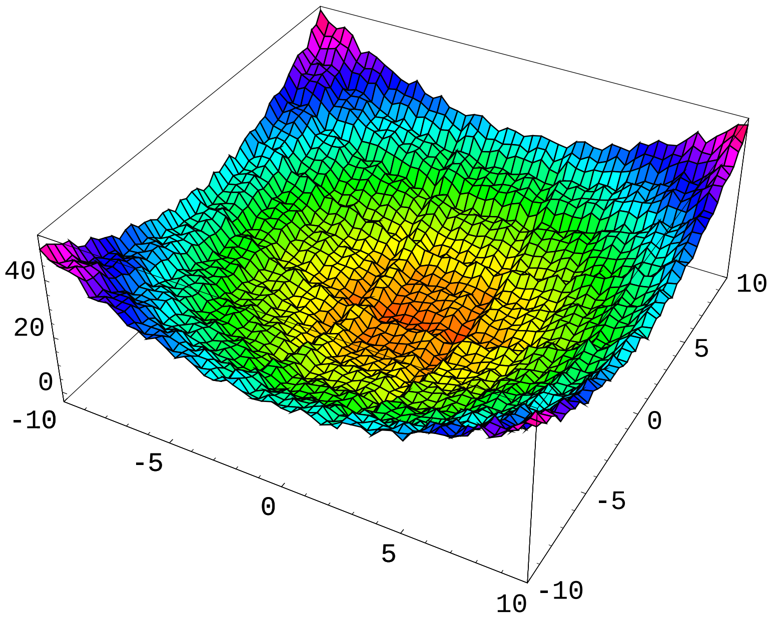
From a distance, it does not seem to be too difficult...

```
Plot3D[objf, {x, -20, 20}, {y, -20, 20},
  PlotPoints → 50, ColorFunction → Hue];
```

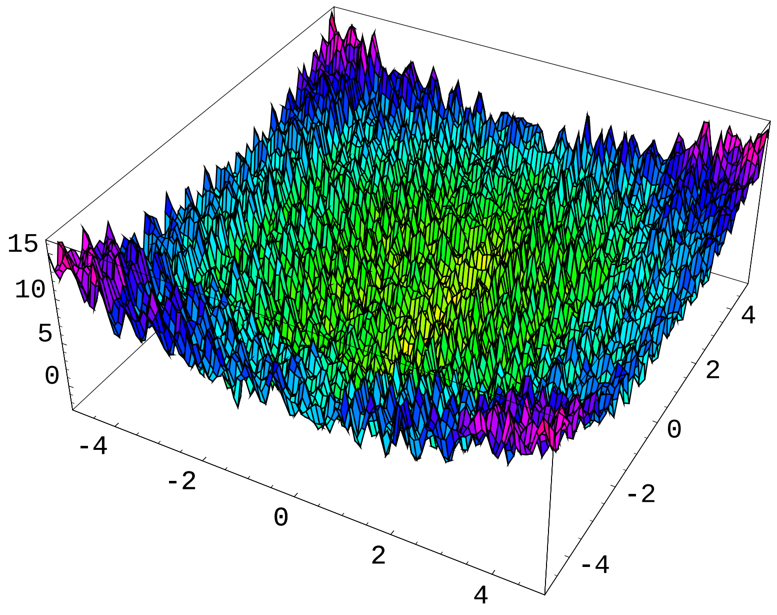


The 'fine print', however, shows a very complicated surface.  This is a 'rather serious toy problem' — standard nonlinear optimizers would most likely face a hopeless task, and even for a proper general purpose global search approach the task is formidable...

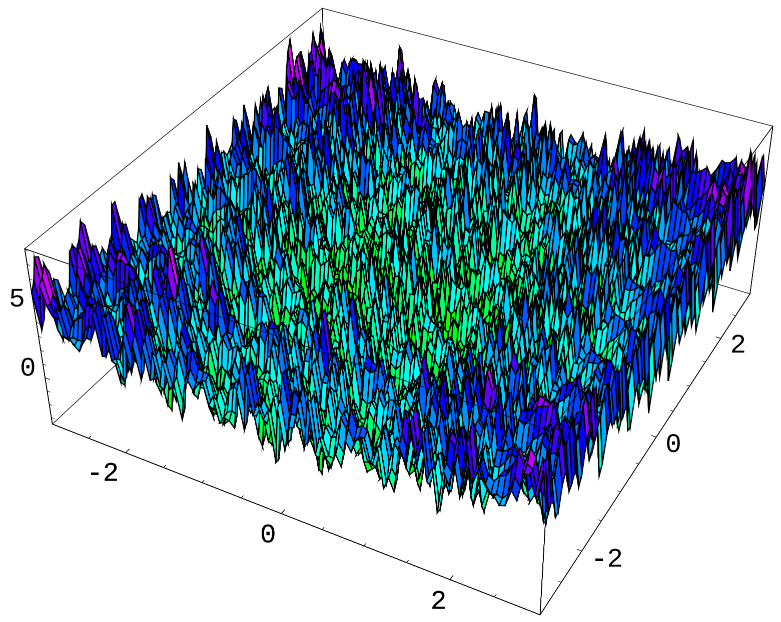
```
Plot3D[objf, {x, -10, 10}, {y, -10, 10},
  PlotPoints → 50, ColorFunction → Hue];
```



```
Plot3D[objf, {x, -5, 5}, {y, -5, 5},
  PlotPoints → 100, ColorFunction → Hue];
```



```
Plot3D[objf, {x, -3, 3}, {y, -3, 3},
  PlotPoints → 100, ColorFunction → Hue];
```



The exact solution could be found, for instance, by using interval-arithmetic tools; however, the associated computational effort is likely to be rather significant, or even prohibitive for most such solver implementations...

For illustrative purposes, below we provide an (approximate) numerical solution by using *MathOptimizer.*

## ■ Numerical Solution

```
vars ={x,y};
varlb = {-3,-3};  (* The bounds are chosen on the basis of the
pictures generated above *)
varub = {3,3};
varnom = {0,0};
eqs = {};
ineqs = {};
objf = 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;
```

Verification of code at the given (selected) nominal solution:

```
N[objf /. Thread[vars->varnom]]
```

```
0.695189
```

Let us see first what local search can do on its own.

```
resultCNLP = CNLP[objf, eqs, ineqs, vars, varnom, varlb, varub]
{{-0.0223022, -0.00472762},
 -0.713075, {}, {}, {-∞, 1.31233×10^-12, 0}}
```

As expected, it gets easily 'trapped' at a local optimum very near the origin. (The objective function value is -0.713075.)

Let us try next the global scope solver MS on its own:

```
resultMS = MS[objf, eqs, ineqs, vars, varnom, varlb, varub]
{{-0.0271588, 0.2191}, -2.96668, -3.71271, -2.96668, {}, {}, 1161}
```

The objective function value found (-2.96668) is much better than the one found earlier by local search.

Solution with standard solver settings leads to the following result:

```
resultOpt = Optimize[objf, eqs, ineqs, vars, varnom, varlb,
varub]
{{-0.0244031, 0.210612}, -3.30687, {}, {}, {-∞, 3.90871×10^-11, 0}}
optval = resultOpt[[2]]
-3.30687
```

The more precise optimum value found equals

```
SetPrecision[optval, 10]
-3.306868647
```

Next, we increase the global search phase sample size, in order to find a (quite possibly) better starting point, as well as a more reliable lower bound estimate.

As the results (below) show, both the starting solution became better and the bound estimate improved significantly.

```
resultMS = MS[objf, eqs, ineqs, vars, varnom, varlb, varub,
  MaxIterations → 1000, MaxSample → 1000]
{{-0.024627, 0.211789}, -3.3, -3.30906, -3.3, {}, {}, 999000}
```

Now we shall feed this solution into the local scope solver.

```
varnom = {-0.02462697605211417, 0.21178911411097368};
resultCNLP = CNLP[objf, eqs, ineqs, vars, varnom, varlb, varub]
{{-0.0244031, 0.210612}, -3.30687, {}, {}, {-∞, 3.17515×10^-7, 0}}
```

The more precise value of the best solution vector found is

```
varopt = {-0.024403079644709905, 0.21061242716594214};
```

```
N[objf /. Thread[vars -> varopt]]
-3.30687
```

One can compare this value to the (statistically established) global bound estimate -3.30906. The latter is based on 999,000 function evaluations on a relatively small two-dimensional interval region; hence, it can be expected to be reasonably tight...

Finally, we calculate the ratio of the optimum estimate found and the lower bound estimate as:

```
-3.30687 / -3.30906
0.999338
```

Hence, one can conjecture that the quality of the solution found by *MathOptimizer* (in a few minutes, using a not exceedingly powerful laptop PC) is estimated to be within 99.93386 % of the — strictly speaking, unknown to us — 'true' solution of this small-scale, but rather significant numerical challenge.

Note added upon revision of the User Guide (August 5, 2002)

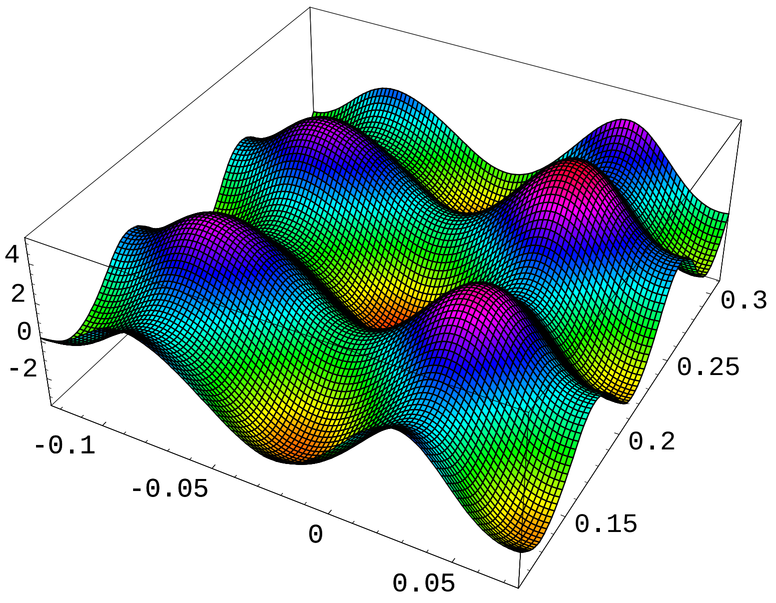
The true optimum value in this problem, to 10-digit accuracy, is -3.306868647, as cited at the Wolfram Research web site http://mathworld.wolfram.com/news/2002-05-25_challenge/.

This implies that the optimum estimate found by *MathOptimizer* is, in fact, precise to at least 10 digits. (Recall that the lower bound value stated above has been based on a statistical estimate).

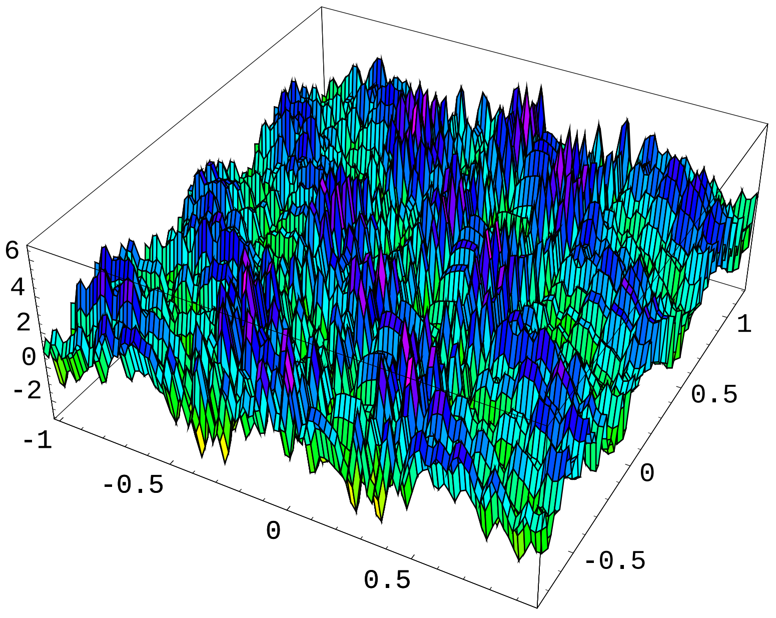
## ■ Posterior Visual Analysis and Concluding Notes

The pictures generated around the optimum estimate (see below) show that the problem is indeed tremendously difficult, at least when perceived as a 'pure' numerical global optimization problem... The first picture also seems to verify the (local) optimality of the numerical solution found, surrounded by fairly 'similar' solutions.

```
Plot3D[objf,
{x,varopt[[1]]-0.1,varopt[[1]]+0.1},{y,varopt[[2]]-0.1,varopt[[2
]]+0.1},PlotPoints→100, ColorFunction→Hue];
```



```
Plot3D[objf,
{x,varopt[[1]]-1.,varopt[[1]]+1.},{y,varopt[[2]]-1.,varopt[[2]]+
1.},PlotPoints→100, ColorFunction→Hue];
```



Note finally that the above analysis serves for illustration only, and that *MathOptimizer*, in fact, could be applied to solving this model with (even) higher precision.

# The Hundred-dollar, Hundred-digit Challenge Problems: Problem 9

## ■ Problem Statement and Visual Analysis

This global optimization challenge was also posted by Trefethen, in the SIAM News January - February 2002; page 3.

**Find the parameter value a that maximizes the value of**

$$(2+\mathrm{Sin}[10\,a])\int_0^2 x^a\,\mathrm{Sin}\left[\frac{a}{2-x}\right]dx \qquad\qquad 0. \le a \le 5.$$

**(The value of the parameter should be determined as precisely as possible.)**

```
Clear[f, a, x];
f[a_, x_]:= (2+Sin[10*a]) * x^a * Sin[a/(2-x)];
Integrate[f[a, x], {x,0,2}]
```

$$\left(\int_0^2 x^a\,\mathrm{Sin}\left[\frac{a}{2-x}\right]dx\right)(2+\mathrm{Sin}[10\,a])$$

This integral is actually known in *Mathematica*, as a special case of the the Meijer G function family (see MeijerG in *Mathematica*'s Help system).

For illustrating the use of a global optimization based approach, numerical integration will be applied below, to obtain (approximate) integral values.

First a simple test: numerical integration at a=0 results in zero, as expected. Observe also the warning messages in the console window these indicate a (possibly) highly oscillatory integrand.

```
NIntegrate[f[0., x], {x,0,2}]
```

```
0.
```

Two other integral values:

```
NIntegrate[f[5., x], {x,0,2}]
```

```
0.0907006
```

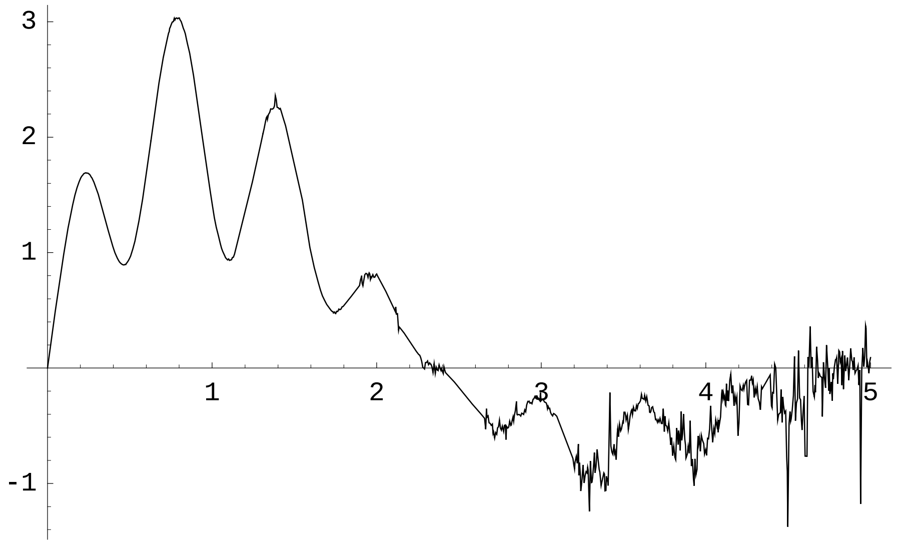
```
NIntegrate[f[2.5, x], {x,0,2}]
```

```
-0.167812
```

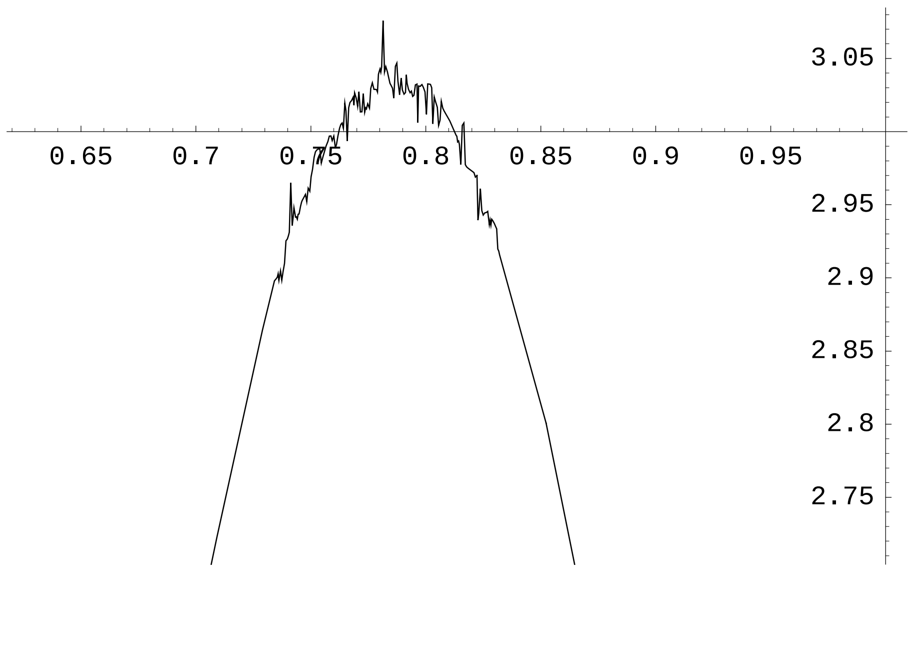
Next, we shall display a sequence of plots of this parametric integral as a function of its parameter **a**, on adaptively chosen, shrinking intervals that we postulate (by simple visual inspection) to enclose the solution.

One shall notice the perhaps unexpected behaviour of this seemingly not too complicated function: this eventually leads to an interesting global optimization challenge...

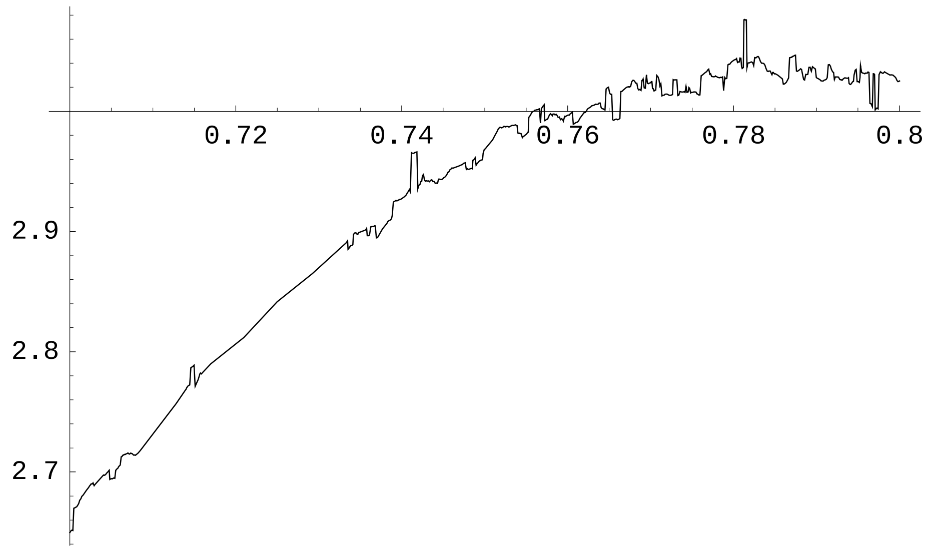
```
Clear[g];
g[a_] := NIntegrate[f[a, x], {x,0,2}];

Plot[g[a], {a, 0., 5.}];
```



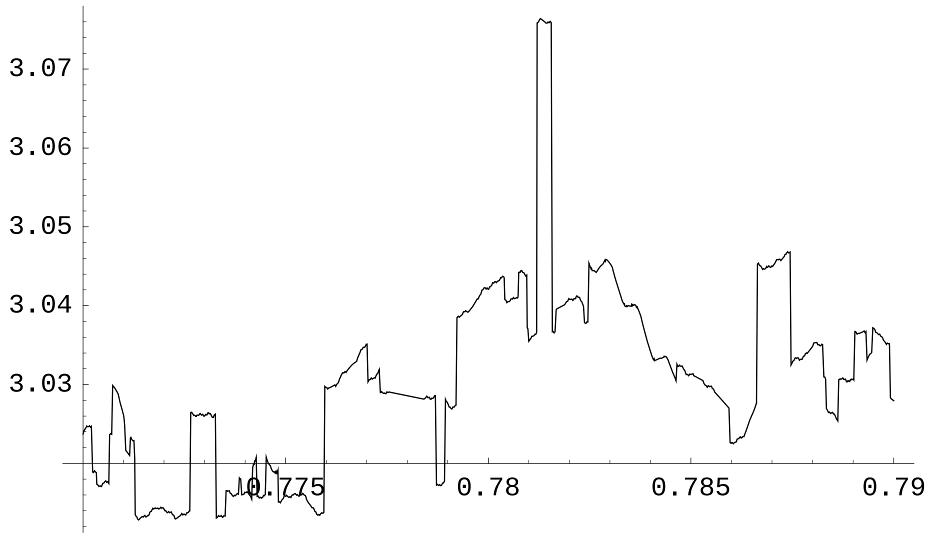
```
Plot[g[a], {a, 0.5, 1.}];
```



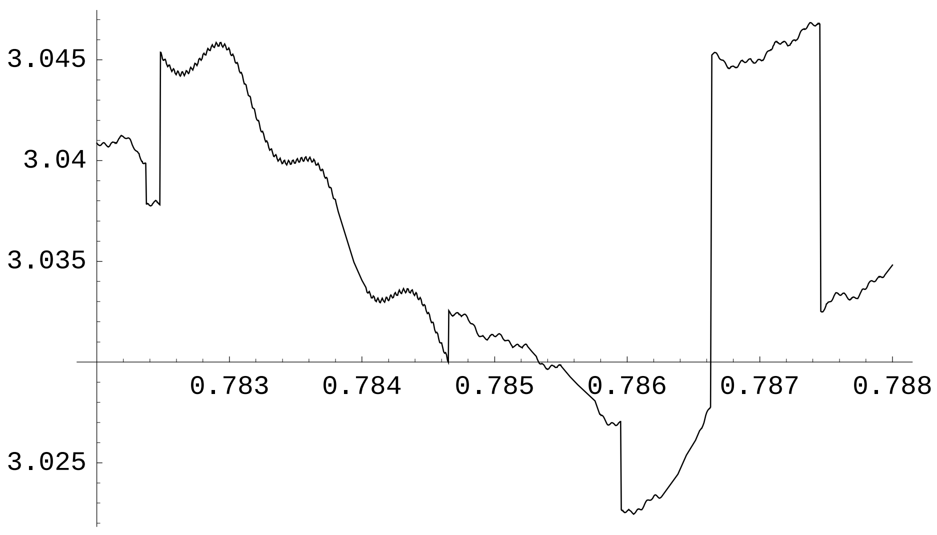
```
Plot[g[a], {a, 0.7, 0.8}];
```



```
Plot[g[a], {a, 0.77, 0.79}];
```



```
Plot[g[a], {a, 0.782, 0.788}];
```



As one can see, the value of the parametric integral heavily oscillates, even on a rather small scale: this fact indicates the difficulty of this (merely one-dimensional) global optimization problem.

## ■ Numerical Solution

```
vars = {a};
varlb = {0.};
varub = {5.};
varnom = {2.5};
eqs = {};
ineqs = {};
(* Change of sign, for maximization problem *)
Clear[objf];
objf = - NIntegrate[(2+Sin[10*a]) * x^a * Sin[a/(2-x)], {x, 0,
2}];

resultOpt = Optimize[objf, eqs, ineqs, vars, varnom, varlb,
varub]
```

```
{{0.778601}, -3.02822, {}, {}, {-∞, 22.7387, 0}}
```

Note that the result above is generated together with a large number of warning messages. These messages are again related to the failure of convergence of the numerical integration routine, due to possible singularity and oscillatory behaviour. The optimization process takes only a few seconds: the computational burden mainly comes from the embedded numerical integration, done for all sampled parameter values.

Observe that the Kuhn-Tucker conditions at the solution found are satisfied only to a (definitely unacceptable) precision of ~ 22.7387: this is also a direct consequence of the poor numerical behaviour of the model (objective) function.

Dr. Frank Kampas suggested to use a modified definition of the objective function, and he observed that this leads to a more precise solution.

```
Clear[objf];
objf[a_?NumberQ] :=
  -NIntegrate[(2 + Sin[10 * a]) * x^a * Sin[a / (2 - x)], {x, 0, 2}];

resultOpt =
 Optimize[objf[a], eqs, ineqs, vars, varnom, varlb, varub]

{{0.786714}, -3.04498, {}, {}, {-∞, 0.0109141, 0}}
```

The true solution argument of the optimization model is **a*** ~ 0.7859336743... as cited in the July - August 2002 issue of SIAM News, p. 3.

Let us evaluate the relative difference between this exact and the approximate solution found by *MathOptimizer*:

```
(0.785934 - 0.786714) / 0.785934

-0.00099245
```

The corresponding objective function values are also close: compare the (imprecise) result below to the (imprecise) result of the optimization runs, 3.02822 and 3.04498.

```
NIntegrate[f[0.785934, x], {x, 0, 2}]

3.02697
```

This shows that in spite of the significant numerical difficulties outlined — which result in some rather imprecise function values during the entire optimization process — *MathOptimizer* generates an approximate solution that is precise to about 99.9 percent in the argument.

This numerical example can serve as another motivation to use a proper, globally scoped search approach, even in unavoidably 'noisy' models.

## ■ Concluding Notes

In an attempt to solve this interesting example, one could find an approximate solution simply by visual 'bracketing', clearly, this would be much harder or impossible to do in higher dimensions...

Interval arithmetic-based methods would provide an exact answer, but the cost of applying such methods easily becomes prohibitive as model dimensionality increases. In presence of numerical errors such as the integration issues mentioned above, such methods may also fail to produce exact answers, unless special remedies are (or can be) implemented.

As this — perhaps not even too complicated — example also indicates, the need for global optimization may occur in 'surprising' contexts.

A large variety of global optimization applications in the sciences and engineering are discussed in some of the references listed in the next section.

# Bibliography

This selected list includes a few 'classical' textbooks, more books from recent years, as well as several World Wide Web sites. The current reference list predominantly contains works that discuss nonlinear modeling and programming, with an added emphasis on global optimization. (The list will be correspondingly expanded, whenever further solver modes are added to *MathOptimizer*.)

Several of the works cited focus on mathematical concepts, theory and algorithms, or software issues pertinent to the subject, while others discuss also a number of interesting applications.

In addition to the *Mathematica Book*, several excellent books on *Mathematica* (consulted by this author also during the *MathOptimizer* development work) are also listed below.

---

Bazaraa, M.S., Sherali, H.D. and Shetty, C.M. (1993) *Nonlinear Programming: Theory and Algorithms.* Wiley, New York.

Bhatti, M.A. (2000) *Practical Optimization Methods With Mathematica Applications*. Springer-Verlag, New York / Berlin / Heidelberg.

Bertsekas, D.P. (1999) *Nonlinear Programming.* (2nd edn.) Athena Scientific, Cambridge, MA.

Bomze, I.M., Csendes, T., Horst, R., and Pardalos, P.M., eds. (1997) *Developments in Global Optimization*. Kluwer Academic Publishers, Dordrecht / Boston / London.

De Leone, R., Murli, A., Pardalos, P.M. and Toraldo, G., eds. (1998) *High Performance Software for Nonlinear Optimization.* Kluwer Academic Publishers, Dordrecht / Boston / London.

Dixon, L.C.W. and Szegö, G.P., eds. (1975, 1978) *Towards Global Optimisation. Vols. 1-2.* North-Holland, Amsterdam.

Edgar, T.F., Himmelblau, D.M, and Lasdon, L.S. (2001) *Optimization of Chemical Processes.* McGraw-Hill, New York.

Floudas et al. (1999) *Handbook of Test Problems in Local and Global Optimization.* Kluwer Academic Publishers, Dordrecht / Boston / London.

Fourer, R. (2002) *Nonlinear Programming FAQ.* http://www-unix.mcs.anl.gov/otc/ Guide/faq/nonlinear-programming-faq.html.

Gaylord, R.J., Kamin, S.N. and Wellin, P.R. (1996) *An Introduction to Programming with Mathematica*. (2nd edn.) Springer-Verlag, New York / Berlin / Heidelberg.

Greenberg, H.J. (1998) *Mathematical Programming Glossary.* http://www-math. cudenver.edu/~hgreenbe/glossary/glossary.html.

Grossmann, I.E., ed. (1996) *Global Optimization in Engineering Design*. Kluwer Academic Publishers, Dordrecht / Boston / London.

Hendrix, E.M.T. (1998) *Global Optimization at Work.* Ph.D. Dissertation, Agricultural University, Wageningen.

Hillier, F.S. and Lieberman, G.J. (1986) *Introduction to Operations Research.* (4th edn.) Holden Day, CA.

Hock, W. and Schittkowski, K. (1981) *Test Examples for Nonlinear Programming Codes.* Springer-Verlag, Berlin / Heidelberg / New York.

Horst, R. and Pardalos, P.M., eds. (1995) *Handbook of Global Optimization, Vol. 1.* Kluwer Academic Publishers, Dordrecht / Boston / London.

*Journal of Global Optimization* (published since 1991). Kluwer Academic Publishers, Dordrecht / Boston / London.

*Journal of Heuristics* (published since 1995). Kluwer Academic Publishers, Dordrecht / Boston / London.

Kearfott, R.B. (1996) *Rigorous Global Search: Continuous Problems.* Kluwer Academic Publishers, Dordrecht / Boston / London.

Maeder, R.E. (1997) *Programming in Mathematica.* (3rd edn.) Addison-Wesley, Reading, MA.

Mittelmann, H.D. and Spelucci, P. (2002) *Decision Tree for Optimization Software.* http://plato.la.asu.edu/guide.html.

Moffett, M.B., Lindberg, J.F., McLaughlin, E.A., and Powers, J.M. (1992) An Equivalent Circuit Model for Barrel-Stave Flextensional Transducers, pp. 170-180 in *Proc. Transducers for Sonics and Ultrasonics*, McCollum, M.D., Hamonic, B.F. and Wilson, O.B., eds. Technomic, Lancaster, Pennsylvania.

Neumaier, A. (2002) *Global Optimization.* http://solon.cma.univie.ac.at/~neum/glopt. html.

Papalambros, P.Y. and Wilde, D.J. (2000) *Principles of Optimal Design*, Cambridge University Press, UK.

Pardalos, P.M. and Romeijn, H.E., eds. (2002) *Handbook of Global Optimization, Vol. 2.* Kluwer Academic Publishers, Dordrecht / Boston / London.

Pintér, J.D. (1996a) *Global Optimization in Action (Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications).* Kluwer Academic Publishers, Dordrecht / Boston / London.

Pintér, J.D. (1999) *LGO - A Model Development System for Continuous Global Optimization. User's Guide.* Pintér Consulting Services, Halifax, NS.

Pintér, J.D. (2001) *Computational Global Optimization in Nonlinear Systems — An Interactive Tutorial.* Lionheart Publishing, Inc., Atlanta, GA.

Pintér, J.D. (2002a) Global Optimization: Software, Tests and Applications. Chapter 15 (pp. 515-569) in: Pardalos and Romeijn, eds.

Pintér, J.D., ed. (2002b) *Global Optimization — Selected Case Studies.* Kluwer Academic Publishers, Dordrecht / Boston / London. To appear (as of September 2002: please check the Kluwer web site for up-to-date information).

Roos, C. and Terlaky, T. (1998) *Nonlinear Optimization*, TU Delft, The Netherlands.

Schrage, L. (2001) *Optimization Modeling with LINGO*. LINDO Systems, Inc. Chicago, IL.

Wagner, D.B. (1996) *Power Programming with Mathematica — The Kernel*, McGraw-Hill, New York.

Wolfram, S. (1999) *The Mathematica Book.* (4th edn.) Wolfram Media, Champaign, IL, and Cambridge University Press, Cambridge, UK.