



KNITROTM 6.0 *for* MathematicaTM
User's Manual

KNITRO for *Mathematica* User's Manual

Version 6.0

Todd Plantenga
Ziena Optimization, Inc.
www.ziena.com

May 2009

©2006-2009 Ziena Optimization, Inc.

Contents

Contents	a
1 Introduction	1
1.1 Product Overview	1
1.2 Algorithms Overview	2
1.3 What's New in Version 6.0	3
1.4 Contact and Support Information	3
2 Installation	4
2.1 Windows	4
2.2 Unix (Linux, Mac OS X)	5
2.3 Linux Compatibility Issues	6
2.4 Running in <i>Mathematica</i>	7
3 Calling KNITRO from <i>Mathematica</i>	8
3.1 Simple examples	8
3.2 Interface definition	9
3.3 Options	11
3.4 Halting execution	12
3.5 Log files	12
4 Improving Performance	14
4.1 Example 1	15
4.2 Example 2	16
5 User options in KNITRO	18
5.1 Description of KNITRO user options	18
5.2 Loading dynamic libraries	31
6 KNITRO termination test and optimality	33
6.1 Continuous problems	33
6.2 Discrete or mixed integer problems	34
7 KNITRO output and solution information	36
7.1 Understanding KNITRO output for continuous problems	36
7.2 Understanding KNITRO output for discrete problems	39
8 Algorithm Options	43
8.1 Automatic	43
8.2 Interior/Direct	43
8.3 Interior/CG	43
8.4 Active Set	43

9 Other KNITRO special features	44
9.1 Second derivative options	44
9.2 Feasibility options	45
9.3 Honor Bounds	45
9.4 Crossover	45
9.5 Multi-start	46
10 Special problem classes	48
10.1 Linear programming problems (LPs)	48
10.2 Quadratic programming problems (QPs)	48
10.3 Systems of Nonlinear Equations	48
10.4 Least Squares Problems	48
10.5 Global optimization	49
10.6 Mixed integer programming (MIP)	49
References	50
Appendix A Solution Status Codes	51
Appendix B List of Output Files	55

1 Introduction

This chapter gives an overview of the KNITRO optimization software package for *Mathematica*[®], and details concerning contact and support information. KNITRO release 6.0 is available for *Mathematica* versions 5.2, 6.0, and 7.0. KNITRO is functionally identical in all versions; however, the KNITRO binary differs between *Mathematica 6.0* and *7.0* because it is linked with the newer *MathLink* library.

1.1 Product Overview

KNITRO for Math Optimization is a *Mathematica* application package for finding solutions of both continuous (smooth) optimization models (with or without constraints), as well as discrete optimization models with integer or binary variables (i.e. mixed integer programs). The Ziena KNITRO solver is implemented as a *MathLink* application, and computes numerical solutions to optimization problems composed from standard *Mathematica* functions. KNITRO is primarily designed for finding local solutions of large-scale, continuous nonlinear problems. Multi-start heuristics are provided for trying to locate the global solution. Although primarily designed for general, nonlinear optimization, KNITRO is efficient at solving all of the following classes of optimization problems:

- unconstrained
- bound constrained
- systems of nonlinear equations
- least squares problems, both linear and nonlinear
- linear programming problems (LPs)
- quadratic programming problems (QPs), both convex and nonconvex
- general nonlinear (smooth) constrained problems (NLP), both convex and nonconvex
- mixed integer linear programs (MILP) of moderate size
- mixed integer (convex) nonlinear programs (MINLP) of moderate size

The KNITRO package provides the following features:

- efficient and robust solution of small or large problems
- solvers for both continuous and discrete problems
- derivative-free, 1st derivative, and 2nd derivative options
- option to remain feasible throughout the optimization or not
- both interior-point (barrier) and active-set methods
- both iterative and direct approaches for computing steps
- support for Windows, Linux, and Mac OS X

1.2 Algorithms Overview

The problems solved by KNITRO have the form

$$\underset{x}{\text{minimize}} \quad f(x) \tag{1.1a}$$

$$\text{subject to} \quad c^L \leq c(x) \leq c^U \tag{1.1b}$$

$$b^L \leq x \leq b^U, \tag{1.1c}$$

where $x \in \mathbf{R}^n$ are the unknown variables (which can be specified as continuous, binary or integer), c^L and c^U are lower and upper bounds (possibly infinite) on the general constraints, and b^L and b^U are lower and upper simple bounds (possibly infinite) on the variables. This formulation allows many types of constraints, including equalities (if $c^L = c^U$), fixed variables (if $b^L = b^U$), and both single and double-sided inequality constraints or bounded variables. KNITRO assumes that the functions $f(x)$, and $c(x)$ are smooth, although problems with derivative discontinuities can often be solved successfully.

KNITRO implements three state-of-the-art interior-point and active-set methods for solving continuous, nonlinear optimization problems. Each algorithm possesses strong convergence properties and is coded for maximum efficiency and robustness. However, the algorithms have fundamental differences that lead to different behavior on nonlinear optimization problems. Together, the three methods provide a suite of different ways to attack difficult problems.

We encourage the user to try all algorithmic options to determine which one is more suitable for the application at hand. For guidance on choosing the best algorithm see Section 8.

Interior/Direct algorithm: Interior-point methods (also known as barrier methods) replace the nonlinear programming problem by a series of barrier subproblems controlled by a barrier parameter μ . Trust regions and a merit function are used to promote convergence. Interior-point methods perform one or more minimization steps on each barrier subproblem, then decrease the barrier parameter and repeat the process until the original problem (1.1) has been solved to the desired accuracy. The Interior/Direct method computes new iterates by solving the primal-dual KKT matrix using direct linear algebra. The method may temporarily switch to the Interior/CG algorithm if it encounters difficulties.

Interior/CG algorithm: This method is similar to the Interior/Direct algorithm, except the primal-dual KKT system is solved using a projected conjugate gradient iteration. This approach differs from most interior-point methods proposed in the literature. A projection matrix is factorized and conjugate gradient applied to approximately minimize a quadratic model of the barrier problem. The use of conjugate gradient on large-scale problems allows KNITRO to utilize exact second derivatives without forming the Hessian matrix.

Active Set algorithm: Active set methods solve a sequence of subproblems based on a quadratic model of the original problem. In contrast with interior-point methods, the algorithm seeks active inequalities and follows a more exterior path to the solution. KNITRO implements a sequential linear-quadratic programming (SLQP) algorithm, similar in nature to a sequential quadratic programming method but using linear programming subproblems to estimate the active set. This method may be preferable to interior-point algorithms when a good initial point can be provided; for example, when solving a sequence of related problems. KNITRO can also “crossover” from an interior-point method and apply Active Set to provide highly accurate active set and sensitivity information (see Section 9.4).

For mixed integer programs (MIPs), KNITRO provides two variants of the branch and bound algorithm. The first is a standard implementation, while the second is specialized for convex, mixed integer nonlinear problems.

For a detailed description of the algorithm implemented in Interior/CG see [4] and for the global convergence theory see [1]. The method implemented in Interior/Direct is described in [10]. The Active Set algorithm is described in [3] and the global convergence theory for this algorithm is in [2]. A summary of the algorithms and techniques implemented in the KNITRO software product is given in [6]. An important component of KNITRO is the HSL routine MA27 [7] which is used to solve the linear systems arising at every iteration of the algorithm. In addition, the Active Set algorithm in KNITRO may make use of the COIN-OR Clp linear programming solver module. The version used in KNITRO may be downloaded from <http://www.ziena.com/clp.html>.

1.3 What's New in Version 6.0

- KNITRO 6.0 introduces new features for solving optimization models (both linear and nonlinear) with binary or integer variables. The KNITRO mixed integer programming (MIP) code offers two algorithms for mixed-integer nonlinear programming (MINLP). The first is a nonlinear branch and bound method and the second implements the hybrid Quesada-Grossman method for convex MINLP. The KNITRO MINLP code is designed for convex mixed integer programming and is a heuristic for nonconvex problems. The MIP code also handles mixed integer linear programs (MILP) of moderate size. A MIP problem is defined and solved using the new functions `KnitroMipMinimize[]` and `KnitroMipMaximize[]`. In addition many new user options beginning with `mip_*` have been added to offer user control over the MIP methods. See more details in Sections 3.2 and 5.1.
- The multi-start generation of new start points is improved in KNITRO 6.0. The user option `ms_maxbndrange` for generating initial points, now only applies to unbounded variables, while the new user option `ms_startprange` establishes a maximum initial range for all variables. The default value of `ms_saveto1` was also increased. See Sections 5.1 and 9.5 for more details.
- KNITRO 6.0 offers improved performance of the SLQP active-set algorithm on simple bound-constrained optimization models. This will often be the most efficient algorithm for these types of problems.
- General performance improvements have been made for both the active-set and interior-point/barrier solvers in KNITRO 6.0.

1.4 Contact and Support Information

KNITRO is licensed and supported by Ziena Optimization, Inc. (<http://www.ziena.com/>). General information regarding KNITRO can be found at the KNITRO website:

<http://www.ziena.com/knitro.html>

For technical support or licensing information, contact Ziena directly at

info@ziena.com

2 Installation

Instructions for installing the KNITRO application package and license on supported platforms are given below. After installing, view the `INSTALL.txt` and `LICENSE.KNITRO.txt` files, then test the installation. Instructions for running KNITRO within *Mathematica* and installing the help documentation are in Section 2.4. These instructions apply for *Mathematica* versions 5.1, 5.2, 6.0, and 7.0.

If upgrading from KNITRO 5.0 for *Mathematica*, then you must first remove or rename the old version. We suggest renaming the installed application folder `Ziena` to `old.Ziena.50`. You should close *Mathematica*, find the `Ziena` folder where KNITRO 5.0 is installed (the instructions below will help), rename it, unzip the new KNITRO release, and then restart *Mathematica*.

2.1 Windows

KNITRO is supported on Windows 2003 and Windows XP SP2. There are compatibility problems with Windows XP SP1 system libraries – users should upgrade to Windows XP SP2. The KNITRO for *Mathematica* software package for Windows is delivered as a zipped file. Copy this file to its installation location in *Mathematica*. The choices are:

- `$InstallationDirectory\AddOns\Applications` - for all users. The default location of `$InstallationDirectory` on Windows is

```
c:\Program Files\Wolfram Research\Mathematica\5.2
```

- `$BaseDirectory\Applications` - for all users on the current machine. The default location of `$BaseDirectory` on Windows is

```
c:\Documents and Settings\All Users\Application Data\Mathematica
```

- `$UserBaseDirectory\Applications` - for just the current user of *Mathematica* on the current machine. The default location of `$UserBaseDirectory` on Windows is

```
c:\Documents and Settings\\Application Data\Mathematica
```

You may evaluate any of the above “\$ variables” in *Mathematica* to find out where they point in your installation.

Make sure *Mathematica* is not running. Double-click on the KNITRO file and extract the contents. Make sure the new folder is named `Ziena`. It should contain the following files:

- `INSTALL.txt`: A file containing installation instructions.
- `Documentation\English\`: A folder containing *Mathematica* Help documentation for KNITRO, and a copy of this manual. See Section 2.4 for details on installing the Help documentation.
- `Licensing\LICENSE.KNITRO.txt`: A file containing the KNITRO license agreement.
- `Licensing\get_machine.ID.exe`: An executable that identifies the machine ID, required for obtaining a `Ziena` license file.

knitro.mx:	The KNITRO for <i>Mathematica</i> application package.
knitro_ml.exe:	A <i>MathLink</i> executable containing the KNITRO solver.
kernel\init.m:	A <i>Mathematica</i> m-file that helps load KNITRO at runtime.

To activate KNITRO for your computer you will need a valid Ziena license file. If you purchased a floating network license, then refer to the *Ziena License Manager User's Manual*. For a stand-alone license, open a DOS-like command window (click **Start** → **Run**, and then type `cmd`). Change to the directory where you unzipped the distribution, and type `get_machine_ID.exe`, a program supplied with the distribution. This will generate a machine ID (five pairs of hexadecimal digits). Email the machine ID to `info@ziena.com`. Ziena will then send a license file containing the encrypted license text string. Ziena supports a variety of ways to install licenses. The simplest procedure is to copy each license into a file whose name begins with the characters "ziena_lic". Then place the file in the folder

```
C:\Program Files\Ziena\
```

For more installation options and general troubleshooting, read the *Ziena License Manager User's Manual*.

2.2 Unix (Linux, Mac OS X)

KNITRO for *Mathematica* is supported on Linux and Mac OS X (10.4 or higher). The KNITRO software package for Unix is delivered as a gzipped tar file. Copy this file to its installation location in *Mathematica*. The choices are:

- `$InstallationDirectory/AddOns/Applications` - for all users. The default location of `$InstallationDirectory` on Unix is

```
/usr/local/Wolfram/Mathematica/5.2
```

- `$BaseDirectory/Applications` - for all users on the current machine. The default location of `$BaseDirectory` on Unix is

```
/usr/share/Mathematica
```

- `$UserBaseDirectory/Applications` - for just the current user of *Mathematica* on the current machine. The default location of `$UserBaseDirectory` on Unix is

```
~/.Mathematica (note that this is a hidden file)
```

You may evaluate any of the above "\$ variables" in *Mathematica* to find out where they point in your installation.

Make sure *Mathematica* is not running. Unpack the distribution by typing the following commands:

```
gunzip Ziena-platformname.tar.gz
tar -xvf Ziena-platformname.tar
```

Unpacking will create a directory named **Ziena** containing the following files:

<code>INSTALL.txt:</code>	A file containing installation instructions.
<code>Documentation/English/:</code>	A folder containing <i>Mathematica</i> Help documentation for KNITRO, and a copy of this manual. See Section 2.4 for details on installing the Help documentation.
<code>Licensing/LICENSE.KNITRO.txt:</code>	A file containing the KNITRO license agreement.
<code>Licensing/get_machine.ID:</code>	An executable that identifies the machine ID, required for obtaining a Ziena license file.
<code>knitro.mx:</code>	The KNITRO for <i>Mathematica</i> application package.
<code>knitro.ml:</code>	A <i>MathLink</i> executable containing the KNITRO solver.
<code>kernel/init.m:</code>	A <i>Mathematica</i> m-file that helps load KNITRO at runtime.

To activate KNITRO for your computer you will need a valid Ziena license file. If you purchased a floating network license, then refer to the *Ziena License Manager User's Manual*. For a stand-alone license, execute `get_machine.ID`, a program supplied with the distribution. This will generate a machine ID (five pairs of hexadecimal digits). Email the machine ID to `info@ziena.com`. Ziena will then send a license file containing the encrypted license text string. Ziena supports a variety of ways to install licenses. The simplest procedure is to copy each license into a file whose name begins with the characters "ziena_lic" (please use lower-case letters). Then place the file in your `$HOME` directory, or the current directory of your *Mathematica* session (shown with the `Directory[]` command). For more installation options and general troubleshooting, read the *Ziena License Manager User's Manual*.

2.3 Linux Compatibility Issues

A Linux link error sometimes seen with user option "blasoption" is the following:

```
./callback1_dynamic: error while loading shared libraries: ../../lib/libmkl.so:
cannot restore segment prot after reloc: Permission denied
```

This is a security enhanced Linux (SELinux) error message. The Intel Math Kernel Library `lib/libmkl.so` shipped with KNITRO does not have the proper security identifiers for your distribution of SELinux (the library is loaded with user option "blasoption"). You could disable security enhancements, but a better fix is to change the security identifiers of the library to acceptable values. On Linux Fedora Core 4, an acceptable security type is "texrel_shlib_t"; other Linux distributions are probably similar. The fix is made by changing to the KNITRO `lib` directory and typing:

```
chcon -c -v -t texrel_shlib_t libmkl.so
```

2.4 Running in *Mathematica*

Launch *Mathematica* and load the KNITRO application by typing

```
<<Ziena'
```

Verify that the package loaded by viewing the built-in's description:

```
?KnitroMinimize
```

Verify that the KNITRO executable is active by solving a trivial optimization problem (type the command on the first line, and KNITRO should return the second line):

```
KnitroMinimize[x,{{x,1,2}}]
{1., {1.}}
```

If you see a license error message

```
KnitroMinimize[x,{{x,1,2}}]
$Failed -- Could not start the KNITRO solver -- please check
your license. status=-49
```

then you have probably placed the Ziena license file in the wrong location. Review Section 2.1 (Windows) or 2.2 (Unix), or consult the *Ziena License Manager User's Manual*, or email info@ziena.com for assistance.

The KNITRO installation includes `.nb` documentation for the *Mathematica* Help Browser. To make the documentation visible in the *Mathematica 5.1* or *5.2* Help Browser, you must rebuild the *Mathematica* Help index. This operation only needs to be done once after installation of KNITRO. Start *Mathematica*, click on "Help" in the title toolbar, then click "Rebuild Help Index". KNITRO documentation should appear in the Help Browser under Add-ons & Links.

To make the documentation visible in the *Mathematica 6.0* Help Browser, make sure the documentation and all its subfolders is copied to `$BaseDirectory`, and then restart *Mathematica*. The full pathname should be `$BaseDirectory\Applications\Ziena\Documentation\English\`. KNITRO documentation should appear in the Help Browser under Documentation Center → Installed Add-ons.

3 Calling KNITRO from *Mathematica*

3.1 Simple examples

A first simple example is an unconstrained optimization problem:

$$\underset{x,y}{\text{minimize}} \quad x^2 + y^2.$$

The solution is found by submitting to KNITRO:

```
KnitroMinimize[x^2 + y^2, {}, {x,y}]
{0., {0., 0.}}
```

The result is a list containing the minimum value, followed by a list of variable values that achieve the minimum. The solution can also be produced in rule form by calling with an option:

```
KnitroMinimize[x^2 + y^2, {}, {x, y}, ReturnAsRules -> True]
{0., {x -> 0., y -> 0.}}
```

The example becomes slightly more challenging with bound constraints on the variables:

$$\underset{x,y}{\text{minimize}} \quad x^2 + y^2 \quad \text{subject to} \quad x \geq 1.$$

Bounds are passed to KNITRO as a pair of lists, lower bounds first, then upper bounds. When a variable has no bound, use plus or minus Infinity:

```
KnitroMinimize[x^2 + y^2,
               {{1, -Infinity}, {Infinity, Infinity}}, {x, y}]
{1., {1., 0.}}
```

Now add a nonlinear constraint to the example:

$$\underset{x,y}{\text{minimize}} \quad x^2 + y^2$$

$$\text{subject to} \quad x^2 + 4y^2 = 1$$

$$x \geq 1.$$

Constraints can be equalities or inequalities, and they are listed after the objective function:

```
KnitroMinimize[{x^2 + y^2, {x^2 + 4y^2 == 4}},
               {{1, -Infinity}, {Infinity, Infinity}}, {x, y}]
{1.75, {1., -0.866025}}
```

This problem is symmetric in y , so another solution is at the point $(1, 0.866025)$. KNITRO finds a *local* optimum to nonlinear optimization problems. A local optimum attains an objective value better than any neighboring feasible point, but not necessarily the best objective value over the entire feasible region. Nonlinear problems with nonconvex objective or constraints can have more than one local solution point; in this example, there are two local solutions. KNITRO reaches a local optimum by iterating from an initial guess. In this case no initial guess was supplied, so KNITRO chose a random point within the variable bounds and arrived at $(1, -0.866205)$. KNITRO can be given an initial guess to help it reach a particular local solution:

```
KnitroMinimize[{x^2 + y^2, {x^2 + 4y^2 == 4}},
               {{1, -Infinity}, {Infinity, Infinity}},
               {{x,1}, {y,1}}]
{1.75, {1., 0.866025}}
```

Users are often interested in finding the very best local solution. This is called a *global* optimization problem, and in general requires searching for local solutions from many different start points. KNITRO provides a simple multi-start capability to assist in this search, described in Section 10.5.

Problems with a given initial start point or variable bounds can be submitted in an alternate form that may be more convenient. The bounds and start point are collected in one list. For the example above, the alternate form is:

```
KnitroMinimize[{x^2 + y^2, {x^2 + 4y^2 == 4}},
               {{x, 1, 1, Infinity}, {y, -Infinity, 1, Infinity}}]
{1.75, {1., 0.866025}}
```

This form can also be used without an initial guess:

```
KnitroMinimize[{x^2 + y^2, {x^2 + 4y^2 == 4}},
               {{x, 1, Infinity}, {y, -Infinity, Infinity}}]
{1.75, {1., -0.866025}}
```

Functions can also be specified symbolically. Here is the same example built up from *Mathematica* definitions:

```
f = x^2 + y^2;
c = {x^2 + 4y^2 == 4};
KnitroMinimize[{f, c}, {{x, 1, Infinity}, {y, -Infinity, Infinity}}]
{1.75, {1., -0.866025}}
```

Another way to construct optimization problems is to define vectors and matrices with Table commands and multiply them together. Here is a quadratic programming problem (quadratic objective function subject to linear constraints) with n unknowns and m constraints:

```
n = 5;
m = 2;
z = Table[a[i], {i,n}];
A = Table[Random[], {m}, {n}];
b = Table[Random[], {m}];
KnitroMinimize[{z.z, {A.z >= b}}, {}, z]
{0.687592, {0.463209, 0.182019, 0.0750876, 0.529216, 0.39267}}
```

3.2 Interface definition

Four built-ins are defined by the application package:

```
KnitroMinimize[]
KnitroMaximize[]
KnitroMipMinimize[]
KnitroMipMaximize[]
```

Both built-ins take the same arguments and options, except `Mip` calls include variable types. The first instructs KNITRO to minimize the objective over continuous variables, while the second asks it to maximize. The third and fourth built-ins minimize and maximize the objective over a mix of continuous and integer-valued variables.

Typing `?KnitroMinimize` displays the description of the basic calling forms, which have already been described in Section 3.1:

```
Minimize a function subject to constraints and bounds on the variables.
  The function and constraints can be linear or nonlinear. Return
  calculated values as {f_optimal, {optimal variable values}}.
The standard calling format, assuming f is a function of x and y:
KnitroMinimize[f, {}, {x,y}] minimizes an unconstrained function.
KnitroMinimize[f, {{x_min,y_min},{x_max,y_max}}, {x,y}]
  minimizes f subject to simple variable bounds.
KnitroMinimize[{f, {list of constraints}}, {}, {x,y}]
  minimizes f subject to general equalities and inequalities.
KnitroMinimize[{f, {list of constraints}}, {{x_min,y_min},{x_max,y_max}},
  {list of vars}]
  minimizes f subject to general constraints and simple variable bounds.
```

All variable bounds and initial values must be numeric (type Integer, Real, or Rational), or the *Mathematica* symbol Infinity (or -Infinity).

Constraint bounds must be numeric. This also implies that variables appear only on one side of a constraint. The following, in which `c(vars)` is some function of the variables, are acceptable constraint forms:

```
bnd == c(vars)
      c(vars) == bnd
lower_bnd <= c(vars)
      c(vars) <= upper_bnd
lower_bnd <= c(vars) <= upper_bnd
upper_bnd >= c(vars)
      c(vars) >= lower_bnd
upper_bnd >= c(vars) >= lower_bnd
```

Objective and constraint functions *must be differentiable* in the feasible region. For instance, the function `Abs[x]` is not differentiable at the point $x = 0$. If KNITRO iterates at this point, it will request evaluation of partial derivatives from *Mathematica*. Results are unpredictable; *Mathematica 5.1* has been observed to crash when derivatives are evaluated with `NumericalFunction` objects (see Section 4 for more about derivative evaluation). In general, nondifferentiable functions should be avoided. If a function is nondifferentiable at known points that can be excluded from possible solutions, then add inequalities to make those points infeasible and use the KNITRO option `{"bar_feasible", 1}` or `{"bar_feasible", 3}` to ensure iterates stay feasible (Section 5.1).

The `KnitroMipMinimize` interface is identical to `KnitroMinimize` except that variable types are included with the variables. Recognized variable types are “c” or “C” (continuous), and “i” or “I” (integer-valued). Each variable is passed as a sublist containing the variable and its type; for example, `{{x, "c"}, {y, "i"}}` instead of `{x,y}`. Bounds on integer variables are rounded to the nearest integer.

3.3 Options

The functions accept a number of options:

- Algorithm - “Automatic”, “Direct”, “CG”, or “Active” (see Section 8).
Default = “Automatic”.
- Hessian - “Exact”, “FiniteDiff”, “BFGS”, “SR1”, or “L-BFGS” (see Section 9.1).
Default = “Exact”.
- HonorBounds - True, False (see Section 9.3).
Default = False.
- MaxIterations - any nonnegative integer. (see Section 5.1).
Default = 1000.
- UseCompiledFns - True, False. (see Section 4).
Default = False.
- ReturnAsRules - True, False. (see below)
Default = False.
- ReturnLambda - True, False. (see below)
Default = False.
- TraceProgress - True, False. (see Section 4)
Default = False.

The first four options are KNITRO user options, described fully in other sections of this manual.

`ReturnAsRules` causes KNITRO to return the solution vector in rules format; for example:

```
KnitroMinimize[x^2 + y^2, {}, {x, y}, ReturnAsRules -> True]
{0., {x -> 0., y -> 0.}}
```

`ReturnLambda` causes KNITRO to return the Lagrange multipliers as a list after the solution vector; for example:

```
KnitroMinimize[{x^2 + y^2, {x^2 + 4y^2 == 4}},
               {{1, -Infinity}, {Infinity, Infinity}},
               {{x,1}, {y,1}}, ReturnLambda -> True]
{1.75, {1., 0.866025}, {-0.25, -1.5, 0.}}
```

KNITRO returns $m + n$ multipliers, where the first m are multipliers for constraints, and the last n are multipliers for variable bounds.

The KNITRO solver offers a large number of other options that are listed in Section 5.1. These can be passed to KNITRO by listing them under the single option name “`KnitroOptionList`”. For example, to select the Direct algorithm, use barrier rule 6, and require all iterates to remain feasible with respect to inequalities:

```
KnitroMinimize[{x^2 + y^2, {x^2 + 4y^2 == 4}},
               {{1, -Infinity}, {Infinity, Infinity}},
               {{x,1}, {y,1}}, Algorithm->"Direct",
               KnitroOptionList -> {"bar_murule",6}, {"bar_feasible",1}]]
```

3.4 Halting execution

The `KnitroMinimize` call can be aborted in the same way other *Mathematica* calls are stopped. If running in GUI mode, then type `<ALT><, >` and choose “Send Abort to link”. If running in console mode, then type `<CTL><C>` and choose “r”.

A better way to limit execution time in KNITRO is to use the “`maxtime_cpu`” or “`maxtime_real`” option. These are available in “`KnitroOptionList`” (see Section 3.3), and described in Section 5.1. If the multi-start capability is enabled to search for a global optimum, then use the “`ms_maxtime_cpu`” or “`ms_maxtime_real`” option, also described in Section 5.1.

3.5 Log files

Two different log files can be enabled by different options. The standard KNITRO output log file described in Section 7 is enabled with

```
KnitroOptionList -> {{"outmode",1},{“outlev”,5}}
```

The output file is named `knitro.log` and appears in the current directory of your *Mathematica* session. This location is shown with the `Directory[]` command in *Mathematica*. If you execute `SetDirectory[]` to change the current directory, then the file appears in this new location. (Note, in the early versions of *Mathematica 6.0* for Windows, files appear in the folder `C:\Documents and Settings\<user>\My Documents`.)

The other log file is an execution summary enabled by setting option “`log_knml`” to 1. For example:

```
KnitroMinimize[x^2 + y^2, {}, {x,y},
               KnitroOptionList->{{"log_knml",1}}]
```

creates a file named `knml.solve.info.log` in the same directory that `knitro.log` appears. The contents of the log file for this example are

```
02/09/2006 10:28:52 KnitroMLSolve called with a new problem
n=2, m=0, nnzJ=0, nnzH=2
length of xinit = 0
length of bnds = 0, 0
length of cbnds = 0, 0
length of Jacobi indices = 0, 0
length of Hessian indices = 2, 2
Optimization complete, KNITRO status = 0
Num FC callbacks = 2
Num GA callbacks = 2
Num W callbacks = 1
CPU time in KNITRO solve = 0.015625 secs
real time in KNITRO solve = 0.019000 secs
real time in callbacks = 0.003000 secs
```

The first line prints the number of unknowns `n` and constraints `m` passed to KNITRO, the number of sparse nonzeros `nnzJ` in the constraint Jacobian, and the number of sparse nonzeros `nnzH` in the Hessian of the Lagrangian. The lengths of arrays passed from *Mathematica* to the *MathLink*

executable are displayed. The statistics after completion provide the number of callbacks and total time spent. KNITRO makes callbacks to *Mathematica* when it needs numerical evaluation of functions (FC), gradients (GA), or second derivatives (W). The log file can provide helpful clues to speed up large-scale problems that execute slowly.

4 Improving Performance

The KNITRO algorithms use Newton-like methods to solve nonlinear optimization problems, and these require evaluation of first and second partial derivatives of the objective and constraint functions. Numerical evaluations by *Mathematica* are needed at every iterate. Newton-like methods solve linear algebra subproblems involving the Jacobian matrix of first derivatives and the Hessian matrix of second derivatives. KNITRO efficiently manages large-scale problems by exploiting sparsity in these matrices; therefore, *Mathematica* must use sparse matrices when evaluating the partial derivatives.

Mathematica provides two different techniques for numerical evaluation of sparse partial derivatives. The first method calls

```
Experimental'CreateNumericalFunction [
    variable list, {function list}, n,
    Gradient->{Automatic, Sparse->False},
    Hessian->{Automatic, Sparse->True}]
```

while the second method conceptually follows the sequence of operations

```
tmpSp = SparseArray[ D[{function list}, {variable list, 1}] ]
tmpR = (#[[1]])& /@ ArrayRules[tmpSp]
tmpE = Extract[ tmpSp, Most[tmpR] ]
spArray = Compile[ Evaluate[variable list], Evaluate[tmpE] ]
```

`CreateNumericalFunction` is included with standard *Mathematica* releases, and enables *Mathematica* to construct a sparse array representation of the first and second derivatives. After the `NumericalFunction` structures are created, KNITRO is started. When KNITRO needs to compute a new iterate, it calls back to *Mathematica* to evaluate numerical derivatives at a particular point, and *Mathematica* makes the computation using the `NumericalFunction` object.

The second method performs symbolic differentiation with respect to all variables and then collects the nonzero terms in a sparse array. The symbolic form is compiled for fast evaluation of numerical derivatives. Again, KNITRO calls back to *Mathematica* for numerical derivatives at new iterates, and in this case *Mathematica* makes the computation using the compiled object.

The `KnitroMinimize` function spends time in three different areas:

1. Construction of sparse partial derivative objects in *Mathematica*.
2. Numerical evaluation of functions and their derivatives in *Mathematica*.
3. Optimization calculations in KNITRO.

Execution time in the first two areas are strongly affected by the options “UseCompiledFns” and “Hessian” (see Section 3.3). Execution time in KNITRO is strongly affected by the options “Algorithm” and “Hessian” (Section 3.3), and to a lesser extent by the KNITRO user options described in Section 5.1. If you experience poor performance, use the “TraceProgress” and logging options to measure the time spent in each area, and then experiment with tuning options to speed up execution. It is difficult to predict the best choice of options in advance, since it depends on the sparsity of partial derivatives, degree of nonlinearity, and condition number of the Jacobian and Hessian matrices, among other factors. Two examples are provided below to illustrate different possibilities.

4.1 Example 1

The first example solves a nonlinear regression problem to find the best two-dimensional ellipse fitting a collection of data points. The example is available as a *Mathematica* notebook from Ziena and on the KNITRO page of *Wolfram's* web site at:

<http://www.wolfram.com/products/applications/knitro/examples.html>

Given p data points $\{(\bar{x}_1, \bar{y}_1), \dots, (\bar{x}_p, \bar{y}_p)\}$, the problem defines a pair of unknowns corresponding to each point. The orthogonal regression formulation makes each unknown as close as possible to its corresponding data point, but requires the unknown points to lie on an ellipse. Mathematically:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^p (x_i - \bar{x}_i)^2 + (y_i - \bar{y}_i)^2 \\ \text{subject to} & ax_i^2 + by_i^2 + cx_iy_i + dx_i + ey_i = 1, \text{ for } i = 1, \dots, p \end{array}$$

Thus, p data points give rise to an optimization problem with $n = 2p + 5$ unknowns and p quadratic equality constraints.

Data sets with up to a few hundred points are solved in just a few seconds; much faster than using the *Mathematica* built-ins `Minimize`, `NMinimize`, or (*Mathematica 6.0* only) `FindMinimum`. However, KNITRO performance becomes an issue for larger numbers of data points. To investigate, we set options “TraceProgress” and enable logging to see where execution time is spent:

```
KnitroMinimize[... , TraceProgress->True, KnitroOptionList->{{"log_knml",1}}]
```

The “TraceProgress” option displays messages like the following:

```
*Trace* KNM is processing arguments
*Trace* creating the objective, UseCompiledFns=0
*Trace* creating the constraints
*Trace* computing the sparse Jacobian
*Trace* computing the sparse Hessian
*Trace* calling KNITRO solver
```

and the log file `knml_solve_info.log` looks like (for $p = 300$):

```
11/07/2006 12:00:53 KnitroMLSolve called with a new problem
  n=605, m=300, nnzJ=2100, nnzH=2700
  length of xinit = 605
  length of bnds = 0, 0
  length of cbnds = 300, 300
  length of Jacobi indices = 2100, 2100
  length of Hessian indices = 2700, 2700
Optimization complete, KNITRO status = 0
  Num FC callbacks = 18
  Num GA callbacks = 11
  Num W callbacks = 10
  CPU time in KNITRO solve = 0.093750 secs
  real time in KNITRO solve = 1.295000 secs
  real time in callbacks = 1.123000 secs
```

The example with $p = 300$ is solved quickly, but performance slows significantly as the number of regression points reaches into the thousands (see the graphs below). The default `KnitroMinimize` options are `UseCompiledFns->False` and `Hessian->"Exact"`. Messages from “TraceProgress” indicate most of the time spent is in construction of the sparse Jacobian and Hessian (i.e., before the message “*Trace* calling KNITRO solver”). This suggests switching to `Hessian->"FiniteDiff"`. It turns out in this example that additional savings result from setting `UseCompiledFns->True`.

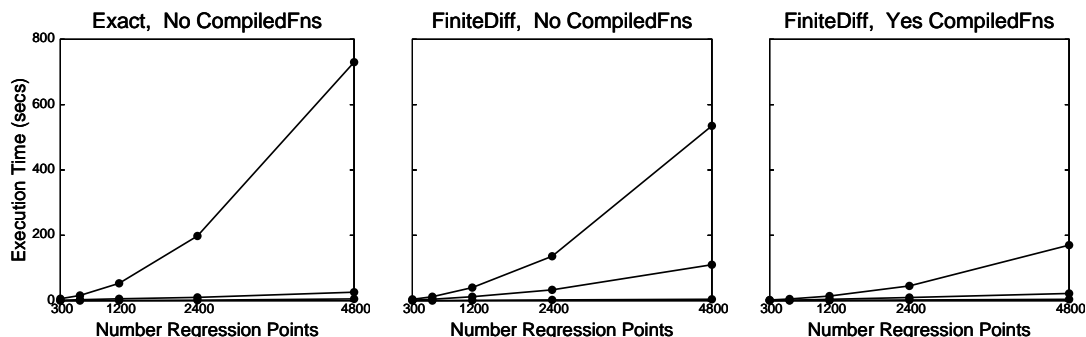
The graphs below show CPU time for various combinations of “Hessian” and “UseCompiledFns”. Each graph plots three lines showing execution time (scale is the same in all three graphs) versus the number of regression points p .

Top line: Total CPU time

Middle line: CPU time spent in KNITRO plus evaluation of numerical functions

Bottom line: CPU time spent in KNITRO (this line is very close to the x -axis)

The time gap between the top and middle lines therefore measures time that *Mathematica* spends constructing the objective, constraints, and sparse derivative matrices. The gap between the middle and bottom lines measures time spent evaluating numerical functions in *Mathematica*. The bottom line is time spent in KNITRO, which is quite small in this example (less than 5 seconds even for $p = 4800$).



The left and center graphs differ by changing `Hessian->"Exact"` to `Hessian->"FiniteDiff"`. They show a big reduction in *Mathematica* setup time, because it no longer constructs Hessian matrices. On the other hand, evaluation time increases because KNITRO requests many more first derivative computations to make a finite-difference approximation to the Hessians. Overall, there is a modest improvement in performance.

The center and right graphs differ by changing `UseCompiledFns->False` to `UseCompiledFns->True`. Evaluation time decreases substantially as expected, because evaluations are now performed by a compiled function. *Mathematica* setup time also decreases, so this set of options is superior. Note that data for these graphs was acquired using *Mathematica 5.2*. In *Mathematica 6.0*, the “UseCompiledFns” option makes almost no difference for this example problem.

4.2 Example 2

The second example is an artificial quadratic programming (QP) problem with a deliberately dense Jacobian matrix. The *Mathematica* code and result from KNITRO are as follows:

```

n = 100;
m = 50;
z = Table[a[i], {i,n}];
A = Table[Random[], {m}, {n}];
b = Table[Random[], {m}];

KnitroMinimize[{z.z, {A.z >= b}}, {}, z]
{0.00976074, {0.00331857, 0.00555646, 0.00559314, 0.00561975, ...

```

The problem has $m = 50$ constraints built from uniformly distributed random numbers. Hence, the constraints are linear, but the Jacobian matrix is dense (5,000 nonzeros in this case). The Hessian matrix is sparse (100 nonzeros) because the only nonzero second derivatives are in the objective function.

A problem of size $n = 100, m = 50$ is solved by `KnitroMinimize` in about one second, substantially faster than using the *Mathematica* built-ins `Minimize` or `NMinimize`. Execution time increases significantly for a problem with $n = 300, m = 150$, primarily because the dense Jacobian now has 45,000 nonzeros. Following the same procedure as Example 1 (Section 4.1), we try `Hessian->"FiniteDiff"` and `UseCompiledFns->True`.

	KNITRO time	Evaluation time	Total time
Exact Hessian, UseCompiledFns=False	8	3	20
Exact Hessian, UseCompiledFns=True	8	5	25
FiniteDiff Hessian, UseCompiledFns=True	35	39	80
FiniteDiff Hessian, UseCompiledFns=False	40	103	150

The table above shows that exact Hessians are much faster than finite-difference approximations. This is because the finite-difference computation makes many first derivative evaluations to estimate the Hessian, and first derivatives with the dense Jacobian are costly to compute. Another competitive option in this example is `Hessian->"L-BFGS"`. This instructs KNITRO to use a built-in approximation for the Hessians, avoiding the cost of constructing and evaluating second derivatives, and the cost of approximating by first derivatives finite-differences.

Mathematica 6.0 offers the built-in function `FindMinimum` with option `Method->InteriorPoint`. This is the fastest *Mathematica* built-in on this problem, but it is not as fast as KNITRO, with the time gap increasing as the problem size scales up. If we change the constraints to equalities instead of inequalities, then the problem becomes easier to solve (ill-conditioning is removed) and `FindMinimum` is competitive. KNITRO automatically applies advanced techniques to cope with ill-conditioning, which explains the performance difference.

The lesson to be learned from these two examples is that many options should be tried when performance on a large-scale problem needs to be improved.

5 User options in KNITRO

The KNITRO solver offers a large number of other options that are listed in this section. These options can be passed to KNITRO by listing them under the single option name “KnitroOptionList”. For example, to select the Direct algorithm, use barrier rule 6, and require all iterates to remain feasible with respect to inequalities:

```
KnitroMinimize[{x^2 + y^2, {x^2 + 4y^2 == 4}},
               {{1, -Infinity}, {Infinity, Infinity}},
               {{x,1}, {y,1}}, Algorithm->"Direct",
               KnitroOptionList -> {"bar_murule",6},{"bar_feasible",1}]
```

The most important options are made directly settable in the `KnitroMinimize` function: “Algorithm”, “Hessian”, “HonorBounds”, and “MaxIterations”. These are listed in Section 3.3.

KNITRO offers a number of user options, each option taking an integer or double precision value. Options are identified by a string name. The `KTR_PARAM_` descriptor is for programmatic interfaces and is not used with *Mathematica*.

5.1 Description of KNITRO user options

`algorithm` (`KTR_PARAM_ALG`): Indicates which algorithm to use to solve the problem (see Section 8).

- 0 (auto): Let KNITRO automatically choose an algorithm, based on the problem characteristics.
- 1 (direct): Use the Interior/Direct algorithm.
- 2 (cg): Use the Interior/CG algorithm.
- 3 (active): Use the Active Set algorithm.

Default value: 0

This option can also be specified as part of the `KnitroMinimize[]` built-in using string names; for example, `Algorithm->"Active"`. See Section 3.3 for details.

`bar_initmu` (`KTR_PARAM_BAR_INITMU`): Specifies the initial value for the barrier parameter μ used with the barrier algorithms. This option has no effect on the Active Set algorithm.

Default value: 1.0e-1

`bar_feasible` (`KTR_PARAM_BAR_FEASIBLE`): Specifies whether special emphasis is placed on getting and staying feasible in the interior-point algorithms.

- 0 (no): No special emphasis on feasibility.
- 1 (stay): Iterates must satisfy inequality constraints once they become sufficiently feasible.
- 2 (get): Special emphasis is placed on getting feasible before trying to optimize.
- 3 (get_stay): Implement both options 1 and 2 above.

Default value: 0

NOTE: This option can only be used with the Interior/Direct and Interior/CG algorithms.

If using {"bar_feasible",1} or {"bar_feasible",3}, this will activate the feasible version of KNITRO. The feasible version of KNITRO will force iterates to strictly satisfy inequalities, but does not require satisfaction of equality constraints at intermediate iterates (see Section 9.2). This option and the `honorbnds` option may be useful in applications where functions are undefined outside the region defined by inequalities. The initial point must satisfy inequalities to a *sufficient* degree; if not, KNITRO may generate infeasible iterates and does not switch to the feasible version until a sufficiently feasible point is found. *Sufficient* satisfaction occurs at a point x if it is true for all inequalities that

$$cl + tol \leq c(x) \leq cu - tol \quad (5.2)$$

The constant tol is determined by the option `bar_feasmodetol`.

If using {"bar_feasible",2} or {"bar_feasible",3}, KNITRO will place special emphasis on first trying to get feasible before trying to optimize.

See Section 9.2 for more details on this option.

`bar_feasmodetol` (KTR.PARAM.BAR_FEASMODETOL): Specifies the tolerance in equation (5.2) that determines whether KNITRO will force subsequent iterates to remain feasible. The tolerance applies to all inequality constraints in the problem. This option only has an effect if using {"bar_feasible",1} or {"bar_feasible",3}.

Default value: 1.0e-4

`bar_initpt` (KTR.PARAM.BAR_INITPT): Indicates whether an initial point strategy is used with barrier algorithms. This option has no effect on the Active Set algorithm.

0 (auto): Let KNITRO automatically choose the strategy.

1 (yes): Shift the initial point to improve barrier algorithm performance.

2 (no): Do no alter the initial point supplied by the user.

Default value: 0

`bar_maxbacktrack` (KTR.PARAM.BAR_MAXBACKTRACK): Indicates the maximum allowable number of backtracks during the linesearch of the Interior/Direct algorithm before reverting to a CG step. Increasing this value will make the Interior/Direct algorithm less likely to take CG steps. If the Interior/Direct algorithm is taking a large number of CG steps (as indicated by a positive value for "CGits" in the output), this may improve performance. This option has no effect on the Active Set algorithm.

Default value: 3

`bar_maxrefactor` (KTR.PARAM.BAR_MAXREFACTOR): Indicates the maximum number of refactorizations of the KKT system per iteration of the Interior/Direct algorithm before reverting to a CG step. These refactorizations are performed if negative curvature is detected in the model. Rather than reverting to a CG step, the Hessian matrix is modified in an attempt to make the subproblem convex and then the KKT system is refactorized. Increasing this value will make

the Interior/Direct algorithm less likely to take CG steps. If the Interior/Direct algorithm is taking a large number of CG steps (as indicated by a positive value for “*CGits*” in the output), this may improve performance. This option has no effect on the Active Set algorithm.

Default value: 0

bar_murule (KTR_PARAM_BAR_MURULE): Indicates which strategy to use for modifying the barrier parameter μ in the barrier algorithms (see Section 8). Not all strategies are available for both barrier algorithms, as described below. This option has no effect on the Active Set algorithm.

- 0 (auto): Let KNITRO automatically choose the strategy.
- 1 (monotone): Monotonically decrease the barrier parameter. Available for both barrier algorithms.
- 2 (adaptive): Use an adaptive rule based on the complementarity gap to determine the value of the barrier parameter. Available for both barrier algorithms.
- 3 (probing): Use a probing (affine-scaling) step to dynamically determine the barrier parameter. Available only for the Interior/Direct algorithm.
- 4 (dampmpc): Use a Mehrotra predictor-corrector type rule to determine the barrier parameter, with safeguards on the corrector step. Available only for the Interior/Direct algorithm.
- 5 (fullmpc): Use a Mehrotra predictor-corrector type rule to determine the barrier parameter, without safeguards on the corrector step. Available only for the Interior/Direct algorithm.
- 6 (quality): Minimize a *quality* function at each iteration to determine the barrier parameter. Available only for the Interior/Direct algorithm.

Default value: 0

bar_penaltycons (KTR_PARAM_BAR_PENCONS): Indicates whether a penalty approach is applied to the constraints. Using a penalty approach may be helpful when the problem has degenerate or difficult constraints. It may also help to more quickly identify infeasible problems, or achieve feasibility in problems with difficult constraints. This option has no effect on the Active Set algorithm.

- 0 (auto): Let KNITRO automatically choose the strategy.
- 1 (none): No constraints are penalized.
- 2 (all): A penalty approach is applied to all general constraints.

Default value: 0

bar_penaltyrule (KTR_PARAM_BAR_PENRULE): Indicates which penalty parameter strategy to use for determining whether or not to accept a trial iterate. This option has no effect on the Active Set algorithm.

- 0 (auto): Let KNITRO automatically choose the strategy.
- 1 (single): Use a single penalty parameter in the merit function to weight feasibility versus optimality.

- 2 (**flex**): Use a more tolerant and flexible step acceptance procedure based on a range of penalty parameter values.

Default value: 0

blasoption (KTR_PARAM_BLASOPTION): Specifies the BLAS/LAPACK function library to use for basic vector and matrix computations.

- 0 (**knitro**): Use KNITRO built-in functions.
 1 (**intel**): Use Intel Math Kernel Library (MKL) functions.
 2 (**dynamic**): Use the dynamic library specified with option **blasoptionlib**.

Default value: 0

NOTE: BLAS and LAPACK functions from Intel Math Kernel Library (MKL) 9.1 are provided with the KNITRO distribution. The MKL is available for Windows (32-bit and 64-bit), Linux (32-bit and 64-bit), and Mac OS X (32-bit x86); it is *not* available for Solaris or Mac OS X PowerPC. The MKL is not included with the free student edition of KNITRO.

BLAS (Basic Linear Algebra Subroutines) and LAPACK (Linear Algebra PACKage) functions are used throughout KNITRO for fundamental vector and matrix calculations. The CPU time spent in these operations can be measured by setting option **debug=1** and examining the output file **kdbg_summ*.txt**. Some optimization problems are observed to spend less than 1% of CPU time in BLAS/LAPACK operations, while others spend more than 50%. Be aware that the different function implementations can return slightly different answers due to roundoff errors in double precision arithmetic. Thus, changing the value of **blasoption** sometimes alters the iterates generated by KNITRO, or even the final solution point.

The **knitro** option uses built-in BLAS/LAPACK functions based on standard netlib routines (www.netlib.org). The **intel** option uses MKL functions written especially for x86 and x86_64 processor architectures. On a machine running an Intel processor (e.g., Pentium 4), testing indicates that the MKL functions can reduce the CPU time in BLAS/LAPACK operations by 20-30%. The **dynamic** option allows users to load any library that implements the functions declared in the file **include/blas_lapack.h**. Specify the library name with option **blasoptionlib**.

The Intel MKL is provided in the KNITRO **lib** directory and is loaded at runtime by KNITRO. The operating system's load path must be configured to find this directory or the MKL will fail to load. See Section 5.2 for details.

If your machine uses security enhanced Linux (SELinux), you may see errors when loading the Intel MKL. Refer to Section 2.3 for more information.

blasoptionlib (KTR_PARAM_BLASOPTIONLIB): Specifies a dynamic library name that contains object code for BLAS/LAPACK functions. The library must implement all the functions declared in the file **include/blas_lapack.h**. The source file **blasAcmlExample.c** in **examples/C** provides a wrapper for the AMD Core Math Library (ACML), suitable for machines with an AMD processor. Instructions are given in the file for creating a BLAS/LAPACK dynamic library from the ACML. The operating system's load path must be configured to find the dynamic library, as described in Section 5.2.

NOTE: This option has no effect unless using {"blasoption",2}.

`cplexlibname` (KTR.PARAM.CPLEXLIB): See option `lpsolver`.

`debug` (KTR.PARAM.DEBUG): Controls the level of debugging output. Debugging output can slow execution of KNITRO and should not be used in a production setting. All debugging output is suppressed if using option `{"outlev",0}`.

- 0 (none): No debugging output.
- 1 (problem): Print algorithm information to `kdbg*.log` output files.
- 2 (execution): Print program execution information.

Default value: 0

`delta` (KTR.PARAM.DELTA): Specifies the initial trust region radius scaling factor used to determine the initial trust region size.

Default value: 1.0e0

`feastol` (KTR.PARAM.FEASTOL): Specifies the final relative stopping tolerance for the feasibility error. Smaller values of `feastol` result in a higher degree of accuracy in the solution with respect to feasibility. See Section 6 for more information.

Default value: 1.0e-6

`feastol_abs` (KTR.PARAM.FEASTOLABS): Specifies the final absolute stopping tolerance for the feasibility error. Smaller values of `feastol_abs` result in a higher degree of accuracy in the solution with respect to feasibility. See Section 6 for more information.

Default value: 0.0e0

`hessopt` (KTR.PARAM.HESOPT): Specifies how to compute the (approximate) Hessian of the Lagrangian. See Section 9.1 for more information.

- 1 (exact): KNITRO calls *Mathematica* to numerically evaluate the exact Hessian.
- 2 (bfgs): KNITRO computes a (dense) quasi-Newton BFGS Hessian.
- 3 (sr1): KNITRO computes a (dense) quasi-Newton SR1 Hessian.
- 4 (finite_diff): KNITRO computes Hessian-vector products using finite-differences.
- 6 (lbfgs): KNITRO computes a limited-memory quasi-Newton BFGS Hessian (its size is determined by the option `lmsize`).

Default value: 1

NOTE: Option `{"hessopt",4}` is not available with the Interior/Direct algorithm. Option `{"hessopt",5}` is not available in KNITRO with *Mathematica*.

This option can also be specified as part of the `KnitroMinimize[]` built-in using string names; for example, `Hessian->"FiniteDiff"`. See Section 3.3 for details.

KNITRO usually performs best when the user provides exact Hessians (`{"hessopt",1}`). If neither can be provided, then `{"hessopt",4}` is recommended. This option is comparable in terms of robustness to the exact Hessian option and typically not much slower in terms of time, provided that gradient evaluations are not a dominant cost. Quasi-Newton options `{"hessopt",2}` and `{"hessopt",3}` are only recommended for small problems ($n < 1000$) since they require working with a dense Hessian approximation. Option `{"hessopt",6}` may be used for large problems. See Section 9.1 for more information.

honorbnds (KTR_PARAM_HONORBND): Indicates whether or not to enforce satisfaction of simple variable bounds throughout the optimization (see Section 9.3). This option and the **bar_feasible** option may be useful in applications where functions are undefined outside the region defined by inequalities.

- 0 (no): KNITRO does not require that the bounds on the variables be satisfied at intermediate iterates.
- 1 (always): KNITRO enforces that the initial point and all subsequent solution estimates satisfy the bounds on the variables.
- 2 (initpt): KNITRO enforces that the initial point satisfies the bounds on the variables.

Default value: 2

lmsize (KTR_PARAM_LMSIZE): Specifies the number of limited memory pairs stored when approximating the Hessian using the limited-memory quasi-Newton BFGS option. The value must be between 1 and 100 and is only used with {"hessopt",6}. Larger values may give a more accurate, but more expensive, Hessian approximation. Smaller values may give a less accurate, but faster, Hessian approximation. When using the limited memory BFGS approach it is recommended to experiment with different values of this parameter. See Section 9.1 for more details.

Default value: 10

lpsolver (KTR_PARAM_LPSOLVER): Indicates which linear programming simplex solver the KNITRO Active Set algorithm uses when solving internal LP subproblems. This option has no effect on the Interior/Direct and Interior/CG algorithms.

- 1 (internal): KNITRO uses its default LP solver.
- 2 (cplex): KNITRO uses IBM ILOG-CPLEX, provided the user has a valid CPLEX license. The CPLEX library is loaded dynamically after `KTR_solve()` is called.

Default value: 1

If using {"lpsolver",2} then the CPLEX shared object library or DLL must reside in the operating system's load path (see Section 5.2). If this option is selected, KNITRO will automatically look for (in order): CPLEX 11.2, CPLEX 11.1, CPLEX 11.0, CPLEX 10.2, CPLEX 10.1, CPLEX 10.0, CPLEX 9.1, CPLEX 9.0, or CPLEX 8.0.

maxcgit (KTR_PARAM_MAXCGIT): Determines the maximum allowable number of inner conjugate gradient (CG) iterations per KNITRO minor iteration.

- 0: Let KNITRO automatically choose a value based on the problem size.
- n*: At most $n > 0$ CG iterations may be performed during one minor iteration of KNITRO.

Default value: 0

maxcrossit (KTR_PARAM_MAXCROSSIT): Specifies the maximum number of crossover iterations before termination. If the value is positive and the algorithm in operation is Interior/Direct or Interior/CG, then KNITRO will crossover to the Active Set algorithm near the solution. The

Active Set algorithm will then perform at most `maxcrossit` iterations to get a more exact solution. If the value is 0, no Active Set crossover occurs and the interior-point solution is the final result.

If Active Set crossover is unable to improve the approximate interior-point solution, then KNITRO will restore the interior-point solution. In some cases (especially on large-scale problems or difficult degenerate problems) the cost of the crossover procedure may be significant – for this reason, crossover is disabled by default. Enabling crossover generally provides a more accurate solution than Interior/Direct or Interior/CG. See Section 9.4 for more information.

Default value: 0

`maxit` (KTR_PARAM_MAXIT): Specifies the maximum number of iterations before termination.

0: Let KNITRO automatically choose a value based on the problem type. Currently KNITRO sets this value to 10000 for LPs/NLPs and 3000 for MIP problems.

n : At most $n > 0$ iterations may be performed before terminating.

Default value: 0

`maxtime_cpu` (KTR_PARAM_MAXTIMECPU): Specifies, in seconds, the maximum allowable CPU time before termination.

Default value: 1.0e8

`maxtime_real` (KTR_PARAM_MAXTIMEREAL): Specifies, in seconds, the maximum allowable real time before termination.

Default value: 1.0e8

`mip_branchrule` (KTR_PARAM_MIP_BRANCHRULE): Specifies which branching rule to use for MIP branch and bound procedure.

0 (auto): Let KNITRO automatically choose the branching rule.

1 (most_frac): Use most fractional (most infeasible) branching.

2 (pseudocost): Use pseudo-cost branching.

3 (strong): Use strong branching (see options `mip_strong_candlim`, `mip_strong_level` and `mip_strong_maxit` for further control of strong branching procedure).

Default value: 0

`mip_debug` (KTR_PARAM_MIP_DEBUG): Specifies debugging level for MIP solution.

0 (none): No MIP debugging output created.

1 (all): Write MIP debugging output to the file `kdbg_mip.log`.

Default value: 0

`mip_gub_branch` (KTR_PARAM_MIP_GUB_BRANCH): Specifies whether or not to branch on generalized upper bounds (GUBs).

- 0 (no): Do not branch on GUBs.
- 1 (yes): Allow branching on GUBs.

Default value: 0

`mip_heuristic` (KTR.PARAM.MIP.HEURISTIC): Specifies which MIP heuristic search approach to apply to try to find an initial integer feasible point. If a heuristic search procedure is enabled, it will run for at most `mip_heuristic_maxit` iterations, before starting the branch and bound procedure.

- 0 (auto): Let KNITRO choose the heuristic to apply (if any).
- 1 (none): No heuristic search applied.
- 2 (feaspump): Apply feasibility pump heuristic.
- 3 (mpec): Apply heuristic based on MPEC formulation.

Default value: 0

`mip_heuristic_maxit` (KTR.PARAM.MIP.HEURISTIC.MAXIT): Specifies the maximum number of iterations to allow for MIP heuristic, if one is enabled.

Default value: 100

`mip_implications` (KTR.PARAM.MIP.IMPLICATNS): Specifies whether or not to add constraints to the MIP derived from logical implications.

- 0 (no): Do not add constraints from logical implications.
- 1 (yes): KNITRO adds constraints from logical implications.

Default value: 1

`mip_integer_tol` (KTR.PARAM.MIP.INTEGERTOL): This value specifies the threshold for deciding whether or not a variable is determined to be an integer.

Default value: 1.0e-8

`mip_integral_gap_abs` (KTR.PARAM.MIP.INTGAPABS): The absolute integrality gap stop tolerance for MIP. See Section 6.2 for more information.

Default value: 1.0e-6

`mip_integral_gap_rel` (KTR.PARAM.MIP.INTGAPREL): The relative integrality gap stop tolerance for MIP. See Section 6.2 for more information.

Default value: 1.0e-6

`mip_knapsack` (KTR.PARAM.MIP.KNAPSACK): Specifies rules for adding MIP knapsack cuts.

- 0 (none): Do not add knapsack cuts.
- 1 (ineqs): Add cuts derived from inequalities only.
- 2 (ineqs_eqs): Add cuts derived from both inequalities and equalities.

Default value: 1

`mip_lpalg` (`KTR_PARAM_MIP_LPALG`): Specifies which algorithm to use for any linear programming (LP) subproblem solves that may occur in the MIP branch and bound procedure. LP subproblems may arise if the problem is a mixed integer linear program (MILP), or if using {"mip_method",2}. (Nonlinear programming subproblems use the algorithm specified by the algorithm option.)

0 (auto): Let KNITRO automatically choose an algorithm, based on the problem characteristics.

1 (direct): Use the Interior/Direct (barrier) algorithm.

2 (cg): Use the Interior/CG (barrier) algorithm.

3 (active): Use the Active Set (simplex) algorithm.

Default value: 0

`mip_maxnodes` (`KTR_PARAM_MIP_MAXNODES`): Specifies the maximum number of nodes explored (0 means no limit).

Default value: 100000

`mip_maxsolves` (`KTR_PARAM_MIP_MAXSOLVES`): Specifies the maximum number of subproblem solves allowed (0 means no limit).

Default value: 200000

`mip_maxtime_cpu` (`KTR_PARAM_MIP_MAXTIMECPU`): Specifies the maximum allowable CPU time in seconds for the complete MIP solution. Use `maxtime_cpu` to additionally limit time spent per subproblem solve.

Default value: 1.0e8

`mip_maxtime_real` (`KTR_PARAM_MIP_MAXTIMEREAL`): Specifies the maximum allowable real time in seconds for the complete MIP solution. Use `maxtime_real` to additionally limit time spent per subproblem solve.

Default value: 1.0e8

`mip_method` (`KTR_PARAM_MIP_METHOD`): Specifies which MIP method to use.

0 (auto): Let KNITRO automatically choose the method.

1 (BB): Use the standard branch and bound method.

2 (HQG): Use the hybrid Quesada-Grossman method (for convex, nonlinear problems only).

Default value: 0

`mip_outinterval` (`KTR_PARAM_MIP_OUTINTERVAL`): Specifies node printing interval for `mip_outlevel` when `mip_outlevel`>0.

1: Print output every node.

2: Print output every 2nd node.

N: Print output every Nth node.

Default value: 10

`mip_outlevel` (KTR_PARAM_MIP_OUTLEVEL): Specifies how much MIP information to print.

- 0 (`none`): Do not print any MIP node information.
- 1 (`iters`): Print one line of output for every node.

Default value: 1

`mip_outsub` (KTR_PARAM_MIP_OUTSUB): Specifies MIP subproblem solve debug output control. This output is only produced if using `{"mip_debug", 1}` and appears in the file `kdbg_mip.log`

- 0: Do not print any debug output from subproblem solves.
- 1: Subproblem debug output enabled, controlled by option `outlev`.
- 2: Subproblem debug output enabled and print problem characteristics.

Default value: 0

`mip_pseudoinit` (KTR_PARAM_MIP_PSEUDOINIT): Specifies the method used to initialize pseudo-costs corresponding to variables that have not yet been branched on in the MIP method.

- 0: Let KNITRO automatically choose the method.
- 1: Initialize using the average value of computed pseudo-costs.
- 2: Initialize using strong branching.

Default value: 0

`mip_rootalg` (KTR_PARAM_MIP_ROOTALG): Specifies which algorithm to use for the root node solve in MIP (same options as `algorithm` user option). *Default value: 0*

`mip_rounding` (KTR_PARAM_MIP_ROUNDING): Specifies the MIP rounding rule to apply.

- 0 (`auto`): Let KNITRO choose the rounding rule.
- 1 (`none`): Do not round if a node is infeasible.
- 2 (`heur_only`): Round using a fast heuristic only.
- 3 (`nlp_sometimes`): Round and solve a subproblem if likely to succeed.
- 4 (`nlp_always`): Always round and solve a subproblem.

Default value: 0

`mip_selectrule` (KTR_PARAM_MIP_SELECTRULE): Specifies the MIP select rule for choosing the next node in the branch and bound tree.

- 0 (`auto`): Let KNITRO choose the node selection rule.
- 1 (`depth_first`): Search the tree using a depth first procedure.
- 2 (`best_bound`): Select the node with the best relaxation bound.
- 3 (`combo_1`): Use depth first unless pruned, then best bound.

Default value: 0

`mip_strong_candlim` (KTR_PARAM_MIP_STRONG_CANDLIM): Specifies the maximum number of candidates to explore for MIP strong branching. *Default value: 10*

`mip_strong_level` (KTR_PARAM_MIP_STRONG_LEVEL): Specifies the maximum number of tree levels on which to perform MIP strong branching. *Default value: 10*

`mip_strong_maxit` (KTR_PARAM_MIP_STRONG_MAXIT): Specifies the maximum number of iterations to allow for MIP strong branching solves. *Default value: 1000*

`mip_terminate` (KTR_PARAM_MIP_TERMINATE): Specifies conditions for terminating the MIP algorithm.

0 (optimal): Terminate at optimum (see Section 6 for more information).

1 (feasible): Terminate at first integer feasible point.

Default value: 0

`ms_enable` or `multistart` (KTR_PARAM_MULTISTART): Indicates whether KNITRO will solve from multiple start points to find a better local minimum. See Section 9.5 for details.

0 (no): KNITRO solves from a single initial point.

1 (yes): KNITRO solves using multiple start points.

Default value: 0

`ms_maxbndrange` (KTR_PARAM_MS_MAXBNDRANGE): Specifies the maximum range that an unbounded variable can take when determining new start points. If a variable is unbounded in one or both directions, then new start point values are restricted by the option. If x_i is such a variable, then all initial values satisfy

$$\max\{b_i^L, x_i^0 - \text{ms_maxbndrange}/2\} \leq x_i \leq \min\{b_i^U, x_i^0 + \text{ms_maxbndrange}/2\},$$

where x_i^0 is the initial value of x_i provided by the user, and b_i^L and b_i^U are the variable bounds (possibly infinite) on x_i . This option has no effect unless using {"ms_enable", 1}.

Default value: 1000.0

`ms_maxsolves` (KTR_PARAM_MS_MAXSOLVES): Specifies how many start points to try in multi-start. This option has no effect unless using {"ms_enable", 1}.

0: Let KNITRO automatically choose a value based on the problem size. The value is $\min(200, 10N)$, where N is the number of variables in the problem.

n : Try $n > 0$ start points.

Default value: 0

`ms_maxtime_cpu` (KTR_PARAM_MS_MAXTIMECPU): Specifies, in seconds, the maximum allowable CPU time before termination. The limit applies to the operation of KNITRO since multi-start began; in contrast, the value of `maxtime_cpu` limits how long KNITRO iterates from a single start point. Therefore, `ms_maxtime_cpu` should be greater than `maxtime_cpu`. This option has no effect unless using {"ms_enable", 1}.

Default value: 1.0e8

ms_maxtime_real (KTR_PARAM_MSMAXTIMEREAL): Specifies, in seconds, the maximum allowable real time before termination. The limit applies to the operation of KNITRO since multi-start began; in contrast, the value of **maxtime_real** limits how long KNITRO iterates from a single start point. Therefore, **ms_maxtime_real** should be greater than **maxtime_real**. This option has no effect unless using {"ms_enable",1}.

Default value: 1.0e8

ms_num_to_save (KTR_PARAM_MSNUMTOSAVE): Specifies the number of distinct feasible points to save in a file named **knitro_mspoints.log**. Each point results from a KNITRO solve from a different starting point, and must satisfy the absolute and relative feasibility tolerances. The file stores points in order from best objective to worst. Points are distinct if they differ in objective value or some component by the value of **ms_savetol** using a relative tolerance test (see Section 9.5). This option has no effect unless using {"ms_enable",1}.

Default value: 0

ms_savetol (KTR_PARAM_MSSAVETOL): Specifies the tolerance for deciding if two feasible points are distinct. Points are distinct if they differ in objective value or some component by the value of **ms_savetol** using a relative tolerance test (see Section 9.5). A large value can cause the saved feasible points in the file **knitro_mspoints.log** to cluster around more widely separated points. This option has no effect unless using {"ms_enable",1} and **ms_num_to_save** is positive.

Default value: 1.0e-6

ms_startptrange (KTR_PARAM_MSSTARTPTRANGE): Specifies the maximum range that each variable can take when determining new start points. If a variable has upper and lower bounds and the difference between them is less than **ms_startptrange**, then new start point values for the variable can be any number between its upper and lower bounds. If the variable is unbounded in one or both directions, or the difference between bounds is greater than the minimum of **ms_startptrange** and **ms_maxbndrange**, then new start point values are restricted by the option. If x_i is such a variable, then all initial values satisfy

$$\max\{b_i^L, x_i^0 - \tau\} \leq x_i \leq \min\{b_i^U, x_i^0 + \tau\},$$

$$\tau = \min\{\text{ms_startptrange}/2, \text{ms_maxbndrange}/2\}$$

where x_i^0 is the initial value of x_i provided by the user, and b_i^L and b_i^U are the variable bounds (possibly infinite) on x_i . This option has no effect unless using {"ms_enable",1}.

Default value: 1.0e20

ms_terminate (KTR_PARAM_MSTERMINATE): Specifies the condition for terminating multi-start. This option has no effect unless using {"ms_enable",1}.

- 0: Terminate after **ms_maxsolves**.
- 1: Terminate after the first local optimal solution is found or **ms_maxsolves**, whichever comes first.
- 2: Terminate after the first feasible solution estimate is found or **ms_maxsolves**, whichever comes first.

Default value: 0

objrange (KTR.PARAM.OBJRANGE): Specifies the extreme limits of the objective function for purposes of determining unboundedness. If the magnitude of the objective function becomes greater than **objrange** for a feasible iterate, then the problem is determined to be unbounded and KNITRO proceeds no further.

Default value: 1.0e20

opttol (KTR.PARAM.OPTTOL): Specifies the final relative stopping tolerance for the KKT (optimality) error. Smaller values of **opttol** result in a higher degree of accuracy in the solution with respect to optimality. See Section 6 for more information.

Default value: 1.0e-6

opttol_abs (KTR.PARAM.OPTTOLABS): Specifies the final absolute stopping tolerance for the KKT (optimality) error. Smaller values of **opttol_abs** result in a higher degree of accuracy in the solution with respect to optimality. See Section 6 for more information.

Default value: 0.0e0

outappend (KTR.PARAM.OUTAPPEND): Specifies whether output should be started in a new file, or appended to existing files. The option affects **knitro.log** and files produced when using {"debug",1}. It does not affect **knitro_newpoint.log**, which is controlled by option **newpoint**.

0 (no): Erase any existing files when opening for output.

1 (yes): Append output to any existing files.

Default value: 0

NOTE: The option should not be changed after calling **KTR_init_problem()**.

outdir (KTR.PARAM.OUTDIR): Specifies a single directory as the location to write all output files. The option should be a full pathname to the directory, and the directory must already exist.

NOTE: The option should not be changed after calling **KTR_init_problem()** or **KTR_mip_init_problem()**.

outlev (KTR.PARAM.OUTLEV): Controls the level of output produced by KNITRO.

0 (none): Printing of all output is suppressed.

1 (summary): Print only summary information.

2 (iter_10): Print basic information every 10 iterations.

3 (iter): Print basic information at each iteration.

4 (iter_verbose): Print basic information and the function count at each iteration.

5 (iter_x): Print all the above, and the values of the solution vector **x**.

6 (all): Print all the above, and the values of the constraints **c** at **x** and the Lagrange multipliers **lambda**.

Default value: 0

outmode (KTR.PARAM.OUTMODE): Specifies where to direct the output from KNITRO.

- 0 (**screen**): Output is directed to standard out (*Mathematica* may route output to a different window).
- 1 (**file**): Output is sent to a file named `knitro.log`.
- 2 (**both**): Output is directed to both the screen and file `knitro.log`.

Default value: 1

pivot (`KTR_PARAM_PIVOT`): Specifies the initial pivot threshold used in factorization routines. The value should be in the range $[0 .. 0.5]$ with higher values resulting in more pivoting (more stable factorizations). Values less than 0 will be set to 0 and values larger than 0.5 will be set to 0.5. If **pivot** is non-positive, initially no pivoting will be performed. Smaller values may improve the speed of the code but higher values are recommended for more stability (for example, if the problem appears to be very ill-conditioned).

Default value: $1.0\text{e-}8$

scale (`KTR_PARAM_SCALE`): Performs a scaling of the objective and constraint functions based on their values at the initial point. If scaling is performed, all internal computations, including the stopping tests, are based on the scaled values.

- 0 (**no**): No scaling is performed.
- 1 (**yes**): KNITRO is allowed to scale the objective function and constraints.

Default value: 1

soc (`KTR_PARAM_SOC`): Specifies whether or not to try second order corrections (SOC). A second order correction may be beneficial for problems with highly nonlinear constraints.

- 0 (**no**): No second order correction steps are attempted.
- 1 (**maybe**): Second order correction steps may be attempted on some iterations.
- 2 (**yes**): Second order correction steps are always attempted if the original step is rejected and there are nonlinear constraints.

Default value: 1

xtol (`KTR_PARAM_XTOL`): The optimization process will terminate if the relative change in all components of the solution point estimate is less than **xtol**. If using the Interior/Direct or Interior/CG algorithm and the barrier parameter is still large, KNITRO will first try decreasing the barrier parameter before terminating.

Default value: $1.0\text{e-}15$

5.2 Loading dynamic libraries

Some user options instruct KNITRO to load dynamic libraries at runtime. This will not work unless the executable can find the desired library using the operating system's *load path*. Usually this is done by appending the path to the directory that contains the library to an environment variable. For example, suppose the library to be loaded is in the system's "temp" directory. The instructions below will correctly modify the load path.

On Windows, type (assuming the directory is `c:\Temp`)

```
> set PATH=%PATH%;c:\Temp
```

On Mac OS X, type (assuming the directory is `/tmp`)

```
> export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:/tmp
```

If you run a Unix bash shell, then type (assuming the directory is `/tmp`)

```
> export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/tmp
```

If you run a Unix csh or tcsh shell, then type (assuming the directory is `/tmp`)

```
> setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:/tmp
```

6 KNITRO termination test and optimality

6.1 Continuous problems

The first-order conditions for identifying a locally optimal solution of the problem (1.1) are:

$$\nabla_x \mathcal{L}(x, \lambda) = \nabla f(x) + \sum_{i=1..m} \lambda_i^c \nabla c_i(x) + \sum_{j=1..n} \lambda_j^b = 0 \quad (6.3)$$

$$\lambda_i^c \min[(c_i(x) - c_i^L), (c_i^U - c_i(x))] = 0, \quad i = 1..m \quad (6.4)$$

$$\lambda_j^b \min[(x_j - b_j^L), (b_j^U - x_j)] = 0, \quad j = 1..n \quad (6.5)$$

$$c_i^L \leq c_i(x) \leq c_i^U, \quad i = 1..m \quad (6.6)$$

$$b_j^L \leq x_j \leq b_j^U, \quad j = 1..n \quad (6.7)$$

$$\lambda_i^c \geq 0, \quad i \in \mathcal{I}, \quad c_i^L \text{ infinite}, \quad c_i^U \text{ finite} \quad (6.8)$$

$$\lambda_i^c \leq 0, \quad i \in \mathcal{I}, \quad c_i^U \text{ infinite}, \quad c_i^L \text{ finite} \quad (6.9)$$

$$\lambda_j^b \geq 0, \quad j \in \mathcal{B}, \quad b_j^L \text{ infinite}, \quad b_j^U \text{ finite} \quad (6.10)$$

$$\lambda_j^b \leq 0, \quad j \in \mathcal{B}, \quad b_j^U \text{ infinite}, \quad b_j^L \text{ finite}. \quad (6.11)$$

Here \mathcal{I} and \mathcal{B} represent the sets of indices corresponding to the general inequality constraints and (non-fixed) variable bound constraints respectively. In the conditions above, λ_i^c is the Lagrange multiplier corresponding to constraint $c_i(x)$, and λ_j^b is the Lagrange multiplier corresponding to the simple bounds on the variable x_j . There is exactly one Lagrange multiplier for each constraint and variable. The Lagrange multiplier may be restricted to take on a particular sign depending on whether the corresponding constraint (or variable) is upper bounded or lower bounded as indicated in (6.8)–(6.11). If the constraint (or variable) has both a finite lower and upper bound, then the appropriate sign of the multiplier depends on which bound (if either) is binding (active) at the solution.

In KNITRO we define the feasibility error (**FeasErr**) at a point x^k to be the maximum violation of the constraints (6.6), (6.7), i.e.,

$$\text{FeasErr} = \max_{i=1..m, j=1..n} (0, (c_i^L - c_i(x^k)), (c_i(x^k) - c_i^U), (b_j^L - x_j^k), (x_j^k - b_j^U)), \quad (6.12)$$

while the optimality error (**OptErr**) is defined as the maximum violation of the first three conditions (6.3)–(6.5). The remaining conditions on the sign of the multipliers (6.8)–(6.11) are enforced explicitly throughout the optimization. In order to take into account problem scaling in the termination test, the following scaling factors are defined

$$\tau_1 = \max(1, (c_i^L - c_i(x^0)), (c_i(x^0) - c_i^U), (b_j^L - x_j^0), (x_j^0 - b_j^U)), \quad (6.13)$$

$$\tau_2 = \max(1, \|\nabla f(x^k)\|_\infty), \quad (6.14)$$

where x^0 represents the initial point.

For unconstrained problems, the scaling (6.14) is not effective since $\|\nabla f(x^k)\|_\infty \rightarrow 0$ as a solution is approached. Therefore, for unconstrained problems only, the following scaling is used in the termination test

$$\tau_2 = \max(1, \min(|f(x^k)|, \|\nabla f(x^0)\|_\infty)), \quad (6.15)$$

in place of (6.14).

KNITRO stops and declares `Locally optimal solution found` if the following stopping conditions are satisfied:

$$\text{FeasErr} \leq \max(\tau_1 * \text{feastol}, \text{feastol_abs}) \quad (6.16)$$

$$\text{OptErr} \leq \max(\tau_2 * \text{opttol}, \text{opttol_abs}) \quad (6.17)$$

where `feastol`, `opttol`, `feastol_abs`, and `opttol_abs` are constants defined by user options (see Section 5).

This stopping test is designed to give the user much flexibility in deciding when the solution returned by KNITRO is accurate enough. One can use a scaled stopping test (which is the recommended default option) by setting `feastol_abs` and `opttol_abs` equal to `0.0e0`. Likewise, an absolute stopping test can be enforced by setting `feastol` and `opttol` equal to `0.0e0`.

Note that the stopping conditions (6.16),(6.17) apply to the problem being solved internally by KNITRO. If the user option `scale=yes` (see Section 5.1), then the problem objective and constraint functions may first be scaled before the problem is sent to KNITRO for the optimization. In this case, the stopping conditions apply to the scaled form of the problem. If the accuracy achieved by KNITRO with the default settings is not satisfactory, the user may either decrease the tolerances described above, or try setting `scale=no`.

Unbounded problems

Since by default, KNITRO uses a relative/scaled stopping test it is possible for the optimality conditions to be satisfied within the tolerances given by (6.16)-(6.17) for an unbounded problem. For example, if $\tau_2 \rightarrow \infty$ while the optimality error stays bounded, condition (6.17) will eventually be satisfied for some `opttol`>0. If you suspect that your problem may be unbounded, using an absolute stopping test will allow KNITRO to detect this.

6.2 Discrete or mixed integer problems

Algorithms for solving versions of (1.1) where one or more of the variables are restricted to take on only discrete values, proceed by solving a sequence of continuous relaxations, where the discrete variables are *relaxed* such that they can take on any continuous value. The *global* solutions, $f(x_R)$, of these relaxed problems provide a lower bound on the optimal objective value for problem (1.1) (upper bound if maximizing). If a feasible point is found for problem (1.1) that satisfies the discrete restrictions on the variables, then this provides an upper bound on the optimal objective value of problem (1.1) (lower bound if maximizing). We will refer to these feasible points as *incumbent* points and denote the objective value at an incumbent point by $f(x_I)$. Assuming all the continuous subproblems have been solved to global optimality (if the problem is convex, all local solutions are global solutions), an optimal solution of problem (1.1) is verified when the lower bound and upper bound are equal.

KNITRO declares optimality for a discrete problem when the gap between the best (i.e., largest) lower bound $f^*(x_R)$ and the best (i.e., smallest) upper bound $f^*(x_I)$ is less than a threshold determined by the user options, `mip_integral_gap_abs` and `mip_integral_gap_rel`. Specifically, KNITRO declares optimality when either

$$f^*(x_I) - f^*(x_R) \leq \text{mip_integral_gap_abs}, \quad (6.18)$$

or

$$f^*(x_I) - f^*(x_R) \leq \text{mip_integral_gap_abs} * \max(1, |f^*(x_I)|), \quad (6.19)$$

where `mip_integral_gap_abs` and `mip_integral_gap_rel` are typically small positive numbers.

Since these termination conditions assume that the continuous subproblems are solved to global optimality and KNITRO only finds local solutions of nonconvex, continuous optimization problems, they are only reliable when solving convex, mixed integer problems. The integrality gap $f^*(x_I) - f^*(x_R)$ should be non-negative although it may become slightly negative from roundoff error, or if the continuous subproblems are not solved to sufficient accuracy. If the integrality gap becomes largely negative, this may be an indication that the model is nonconvex, in which case KNITRO may not converge to the optimal solution, and will be unable to verify optimality (even if it claims otherwise).

7 KNITRO output and solution information

This section provides information on understanding the KNITRO output and accessing solution information.

7.1 Understanding KNITRO output for continuous problems

If using `{"outlev",0}` then all printing of output is suppressed; this is the default with *Mathematica*. If `outlev` is positive, then KNITRO prints information about the solution of your optimization problem either to standard output (`{"outmode",0}`), to a file named `knitro.log` (`{"outmode",1}`), or to both (`{"outmode",2}`). Output to the file is recommended with *Mathematica*.

This section describes KNITRO outputs at various levels.

Display of Nondefault Options:

KNITRO first prints the banner displaying the Ziena license type and version of KNITRO that is installed. It then lists all user options which are different from their default values (see Section 5 for the default user option settings). If nothing is listed in this section then it must be that all user options are set to their default values. Lastly, KNITRO prints messages that describe how it resolved user options that were set to AUTOMATIC values. For example, if using option `{"algorithm",0}`, then KNITRO prints the algorithm that it chooses.

```

=====
      Commercial Ziena License
      KNITRO 6.0.0
      Ziena Optimization, Inc.
      website:  www.ziena.com
      email:    info@ziena.com
=====

outlev:      6
KNITRO changing algorithm from AUTO to 1.
KNITRO changing bar_murule from AUTO to 1.
KNITRO changing bar_initpt from AUTO to 2.
KNITRO changing bar_penaltyrule from AUTO to 1.
KNITRO changing bar_penaltycons from AUTO to 1.

```

In the example above, it is indicated that we are using a more verbose output level `{"outlev",6}` instead of the default value `{"outlev",2}`. KNITRO chose algorithm 1 (Interior/Direct), and then determined four other options related to the algorithm.

Display of Problem Characteristics:

KNITRO next prints a summary description of the problem characteristics including the number and type of variables and constraints and the number of nonzero elements in the Jacobian matrix and Hessian matrix (if providing the exact Hessian).

```

Problem Characteristics
-----

```

```

Objective goal: Minimize
Number of variables:      2
  bounded below:         0
  bounded above:         1
  bounded below and above: 0
  fixed:                  0
  free:                   1
Number of constraints:    2
  linear equalities:      0
  nonlinear equalities:   0
  linear inequalities:    0
  nonlinear inequalities: 2
  range:                  0
Number of nonzeros in Jacobian: 4
Number of nonzeros in Hessian: 3

```

Display of Iteration Information:

Next, if `outlev` is greater than 2, KNITRO prints columns of data reflecting detailed information about individual iterations during the solution process. An iteration is defined as a step which generates a new solution estimate (i.e., a successful step).

If using `{"outlev",2}`, summary data is printed every 10 iterations, and on the final iteration. If `{"outlev",3}`, summary data is printed every iteration. If `{"outlev",4}`, the most verbose iteration information is printed every iteration.

Iter	fCount	Objective	FeasError	OptError	Step	CGits
0	1	9.090000e+02	3.000e+00			
1	2	7.989784e+02	2.878e+00	9.096e+01	6.566e-02	0
2	3	4.232342e+02	2.554e+00	5.828e+01	2.356e-01	0
3	4	1.457686e+01	9.532e-01	3.088e+00	1.909e+00	0
4	9	1.235269e+02	7.860e-01	3.818e+00	7.601e-01	5
5	10	3.993788e+02	3.022e-02	1.795e+01	1.186e+00	0
6	11	3.924231e+02	2.924e-02	1.038e+01	1.856e-02	0
7	12	3.158787e+02	0.000e+00	6.905e-02	2.373e-01	0
8	13	3.075530e+02	0.000e+00	6.888e-03	2.255e-02	0
9	14	3.065107e+02	0.000e+00	6.397e-05	2.699e-03	0
10	15	3.065001e+02	0.000e+00	4.457e-07	2.714e-05	0

The meaning of each column is described below.

Iter: Iteration number.

fCount: The cumulative number of function evaluations. (This information is only printed if `outlev` is greater than 3).

Objective: Gives the value of the objective function at the current iterate.

FeasError: Gives a measure of the feasibility violation at the current iterate (see Section 6).

- OptError:** Gives a measure of the violation of the Karush-Kuhn-Tucker (KKT) (first-order) optimality conditions (not including feasibility) at the current iterate (see Section 6).
- ||Step||:** The 2-norm length of the step (i.e., the distance between the new iterate and the previous iterate).
- CGits:** The number of Projected Conjugate Gradient (CG) iterations required to compute the step.

Display of Termination Status:

At the end of the run a termination message is printed indicating whether or not the optimal solution was found and if not, why KNITRO stopped. The termination message typically starts with the word "EXIT:". If KNITRO was successful in satisfying the termination test (see Section 6), the message will look as follows:

```
EXIT: Locally optimal solution found.
```

See the appendix for a list of possible termination messages and a description of their meaning and the corresponding value returned by `KTR_solve()`.

Display of Final Statistics:

Following the termination message, a summary of some final statistics on the run are printed. Both relative and absolute error values are printed.

Final Statistics

```
-----
```

```
Final objective value           = 3.06500096351765e+02
Final feasibility error (abs / rel) = 0.00e+00 / 0.00e+00
Final optimality error (abs / rel) = 4.46e-07 / 3.06e-08
# of iterations                 = 10
# of CG iterations              = 5
# of function evaluations       = 15
# of gradient evaluations       = 11
# of Hessian evaluations        = 10
Total program time (secs)      = 0.00136 ( 0.000 CPU time)
Time spent in evaluations (sec) = 0.00012
```

Display of Solution Vector and Constraints:

If `outlev` equals 5 or 6, the values of the solution vector are printed after the final statistics. If `outlev` equals 6, the final constraint values are also printed, and the values of the Lagrange multipliers (or dual variables) are printed next to their corresponding constraint or bound.

```
Constraint Vector                Lagrange Multipliers
-----
c[ 0] = 1.00000006873e+00,    lambda[ 0] = -7.00000062964e+02
c[ 1] = 4.50000096310e+00,    lambda[ 1] = -1.07240081095e-05
```

Solution Vector

```

-----
x[      0] =  4.99999972449e-01,   lambda[      2] =  7.27764067199e+01
x[      1] =  2.00000024766e+00,   lambda[      3] =  0.00000000000e+00
=====

```

KNITRO can produce additional information which may be useful in debugging or analyzing performance. If `outlev` is positive and using `{"debug",1}`, then multiple files named `kdbg_*.log` are created which contain detailed information on performance. If `outlev` is positive and `{"debug",2}`, then KNITRO prints information useful for debugging program execution. The information produced by `debug` is primarily intended for developers, and should not be used in a production setting.

7.2 Understanding KNITRO output for discrete problems

If using `{"outlev",0}` then all printing of output is suppressed. If `outlev` is positive, then KNITRO prints information about the solution of your optimization problem either to standard output (`{"outmode",0}`), to a file named `knitro.log` (`{"outmode",1}`), or to both (`{"outmode",2}`). The option `outdir` controls the directory where output files are created (if any are) and the option `outappend` controls whether output is appended to existing files. When `outlev` is positive, the options `mip_outlevel`, `mip_debug`, `mip_outinterval` and `mip_outsub` control the amount and type of MIP output generated as described below. See Section 5 for more details.

This section describes KNITRO outputs at various levels for discrete or mixed integer problems. We examine the output that results from running `test_api_minlp` to solve the “Synthesis 1” nonlinear, mixed integer programming problem.

KNITRO first prints the banner displaying the Ziena license type and version of KNITRO that is installed. It then lists all user options which are different from their default values (see Section 5 for the default user option settings). If nothing is listed in this section then it must be that all user options are set to their default values. Lastly, KNITRO prints messages that describe how it resolved user options that were set to `AUTOMATIC` values. For example, if using option `{"mip_branchrule",0}`, then KNITRO prints the branching rule that it chooses.

```

=====
          Commercial Ziena License
          KNITRO 6.0.0
          Ziena Optimization, Inc.
          website:  www.ziena.com
          email:    info@ziena.com
=====

mip_method:          1
mip_outinterval:    1
KNITRO changing mip_rootalg from AUTO to 1.
KNITRO changing mip_lpalg from AUTO to 3.
KNITRO changing mip_branchrule from AUTO to 2.
KNITRO changing mip_selectrule from AUTO to 2.
KNITRO changing mip_rounding from AUTO to 3.
KNITRO changing mip_heuristic from AUTO to 1.
KNITRO changing mip_pseudoinit from AUTO to 1.

```

In the example above, it is indicated that we are using `{"mip_method",1}` which is the standard branch and bound method (see Section 5), and that we are printing output information at every node since `{"mip_outinterval",1}`. It then determined seven other options related to the MIP method.

Display of Problem Characteristics:

KNITRO next prints a summary description of the problem characteristics including the number and type of variables and constraints and the number of nonzero elements in the Jacobian matrix and Hessian matrix (if providing the exact Hessian).

If no initial point is provided by the user, KNITRO indicates that it is computing one. KNITRO also prints the results of any MIP preprocessing to detect special structure and indicates which MIP method it is using.

Problem Characteristics

```
-----
Objective goal: Minimize
Number of variables:           6
    bounded below:             0
    bounded above:             0
    bounded below and above:   6
    fixed:                     0
    free:                      0
Number of binary variables:    3
Number of integer variables:   0
Number of constraints:         6
    linear equalities:         0
    nonlinear equalities:      0
    linear inequalities:       4
    nonlinear inequalities:     2
    range:                    0
Number of nonzeros in Jacobian: 16
Number of nonzeros in Hessian: 3
```

No start point provided -- KNITRO computing one.

```
KNITRO detected 1 GUB constraints
KNITRO derived 0 knapsack covers after examining 3 constraints
KNITRO solving root node relaxation
KNITRO MIP using Branch and Bound method
```

Display of Node Information:

Next, if `{"mip_outlevel",1}`, KNITRO prints columns of data reflecting detailed information about individual nodes during the solution process. The frequency of this node information is controlled by the `mip_outinterval` parameter. For example, if using `{"mip_outinterval",100}`,

this node information is printed only for every 100th node (printing output less frequently may save significant CPU time in some cases). In the example below, {"mip_outinterval", 1}, so information about every node is printed.

Node	Left	Iinf	Objective		Best relaxatn	Best incumbent
1	0	2	7.592845e-01		7.592845e-01	
2	1	1	5.171320e+00		7.592845e-01	
* 2	1			r		7.671320e+00
* 3	2	0	6.009759e+00	f	5.171320e+00	6.009759e+00
4	1		1.000000e+01	pr	5.171320e+00	6.009759e+00
5	0		7.092732e+00	pr	6.009759e+00	6.009759e+00

The meaning of each column is described below.

Node: The node number. If an integer feasible point was found at a given node, then it is marked with a *

Left: The current number of active nodes left in the branch and bound tree.

Iinf: The number of integer infeasible variables at the current node solution.

Objective: Gives the value of the objective function at the solution of the relaxed subproblem solved at the current node. If the subproblem was infeasible or failed, this is indicated. Additional symbols may be printed at some nodes if the node was pruned (**pr**), integer feasible (**f**), or an integer feasible point was found through rounding (**r**).

Best relaxatn: The value of the current best relaxation (lower bound on the solution if minimizing) (see Section 6.2).

Best incumbent: The value of the current best integer feasible point (upper bound on the solution if minimizing) (see Section 6.2).

Display of Termination Status:

At the end of the run a termination message is printed indicating whether or not the optimal solution was found and if not, why KNITRO stopped. The termination message typically starts with the word "EXIT:". If KNITRO was successful in satisfying the termination test (see Section 6.2), the message will look as follows:

```
EXIT: Optimal solution found.
```

See the appendix for a list of possible termination messages and a description of their meaning and the corresponding value returned by `KTR_mip_solve()`.

Display of Final Statistics:

Following the termination message, a summary of some final statistics on the run are printed.

Final Statistics for MIP

Final objective value	=	6.00975890892825e+00
Final integrality gap (abs / rel)	=	0.00e+00 / 0.00e+00 (0.00%)
# of nodes processed	=	5
# of subproblems solved	=	6
Total program time (secs)	=	0.09930 (0.099 CPU time)
Time spent in evaluations (secs)	=	0.00117

Display of Solution Vector and Constraints:

If `outlev` equals 5 or 6, the values of the solution vector are printed after the final statistics.

Solution Vector

x[0]	=	1.30097589089e+00
x[1]	=	0.00000000000e+00
x[2]	=	1.00000000000e+00
x[3]	=	0.00000000000e+00 (binary variable)
x[4]	=	1.00000000000e+00 (binary variable)
x[5]	=	0.00000000000e+00 (binary variable)

=====

KNITRO can produce additional information which may be useful in debugging or analyzing MIP performance. If `outlev` is positive and `{"mip_debug",1}`, then the file named `kdbg_mip.log` is created which contains detailed information on the MIP performance. In addition, if `{"mip_outsub",1}`, this file will contain extensive output for each subproblem solve in the MIP solution process. The information produced by `mip_debug` is primarily intended for developers, and should not be used in a production setting.

8 Algorithm Options

8.1 Automatic

KNITRO provides three different algorithms for solving problems. See Section 1.2 for an overview of the methods. By default, KNITRO automatically tries to choose the best algorithm for a given problem based on problem characteristics.

We strongly encourage you to experiment with all the algorithms as it is difficult to predict which one will work best on any particular problem.

8.2 Interior/Direct

This algorithm often works best, and will automatically switch to Interior/CG if the direct step is suspected to be of poor quality, or if negative curvature is detected. Interior/Direct is recommended if the Hessian of the Lagrangian is ill-conditioned. The Interior/CG method in this case will often take an excessive number of conjugate gradient iterations. It may also work best when there are dependent or degenerate constraints. Choose this algorithm by setting user option `algorithm=1`.

We encourage you to experiment with different values of the `bar_murule` option when using the Interior/Direct or Interior/CG algorithm. It is difficult to predict which update rule will work best on a problem.

NOTE: Since the Interior/Direct algorithm in KNITRO requires the explicit storage of a Hessian matrix, this algorithm only works with Hessian options (`hessopt`) 1, 2, 3, or 6 (see Section 9.1). It may not be used with Hessian options 4 or 5, which do not supply a full Hessian matrix. The Interior/Direct algorithm may be used with the `bar_feasible` option.

8.3 Interior/CG

This algorithm is well-suited to large problems because it avoids forming and factorizing the Hessian matrix. Interior/CG is recommended if the Hessian is large and/or dense. It works with all Hessian options, and with the `bar_feasible` option. Choose this algorithm by setting user option `algorithm=2`.

We encourage you to experiment with different values of the `bar_murule` option when using the Interior/Direct or Interior/CG algorithm. It is difficult to predict which update rule will work best on a problem.

8.4 Active Set

This algorithm is fundamentally different from interior-point methods. The method is efficient and robust for small and medium-scale problems, but is typically less efficient than the Interior/Direct and Interior/CG algorithms on large-scale problems (many thousands of variables and constraints). Active Set is recommended when “warm starting” (i.e., when the user can provide a good initial solution estimate, for example, when solving a sequence of closely related problems). This algorithm is also best at rapid detection of infeasible problems. Choose this algorithm by setting user option `algorithm=3`.

NOTE: The `bar_feasible` option (see Section 9.2) is not available for use with the Active Set algorithm. The method works with all Hessian options.

9 Other KNITRO special features

This section describes in more detail some of the most important features of KNITRO. It provides some guidance on which features to use so that KNITRO runs most efficiently for the problem at hand.

9.1 Second derivative options

The default option of KNITRO uses analytic second derivatives from *Mathematica* to compute the Hessian of the Lagrangian function. Sometimes the cost of computing second derivatives is overly expensive; in this case, KNITRO offers other options which are described in detail below.

(Dense) Quasi-Newton BFGS

The quasi-Newton BFGS option uses gradient information to compute a symmetric, *positive-definite* approximation to the Hessian matrix. Typically this method requires more iterations to converge than the exact Hessian version. However, since it is only computing gradients rather than Hessians, this approach may be more efficient in some cases. This option stores a *dense* quasi-Newton Hessian approximation so it is only recommended for small to medium problems ($n < 1000$). The quasi-Newton BFGS option is chosen by setting user option `hessopt=2`.

(Dense) Quasi-Newton SR1

As with the BFGS approach, the quasi-Newton SR1 approach builds an approximate Hessian using gradient information. However, unlike the BFGS approximation, the SR1 Hessian approximation is not restricted to be positive-definite. Therefore the quasi-Newton SR1 approximation may be a better approach, compared to the BFGS method, if there is a lot of negative curvature in the problem since it may be able to maintain a better approximation to the true Hessian in this case. The quasi-Newton SR1 approximation maintains a *dense* Hessian approximation and so is only recommended for small to medium problems ($n < 1000$). The quasi-Newton SR1 option is chosen by setting user option `hessopt=3`.

Finite-difference Hessian-vector product option

If the problem is large and gradient evaluations are not a dominant cost, then KNITRO can internally compute Hessian-vector products using finite-differences. Each Hessian-vector product in this case requires one additional gradient evaluation. This option is chosen by setting user option `hessopt=4`. The option is only recommended if the exact gradients are provided.

NOTE: This option may not be used when `algorithm=1`.

Limited-memory Quasi-Newton BFGS

The limited-memory quasi-Newton BFGS option is similar to the dense quasi-Newton BFGS option described above. However, it is better suited for large-scale problems since, instead of storing a dense Hessian approximation, it stores only a limited number of gradient vectors used to approximate the Hessian. The number of gradient vectors used to approximate the Hessian is controlled by user option `lmsize`.

A larger value of `lmsize` may result in a more accurate, but also more expensive, Hessian approximation. A smaller value may give a less accurate, but faster, Hessian approximation. When using the limited memory BFGS approach it is recommended to experiment with different values of this parameter.

In general, the limited-memory BFGS option requires more iterations to converge than the dense quasi-Newton BFGS approach, but will be much more efficient on large-scale problems. The limited-memory quasi-Newton option is chosen by setting user option `hessopt=6`.

9.2 Feasibility options

KNITRO offers an option `bar_feasible` that can force iterates to stay feasible with respect to *inequality* constraints or can place special emphasis on trying to get feasible.

If `bar_feasible=1` or `bar_feasible=3` KNITRO satisfies inequalities by switching to a *feasible mode* of operation, which alters the manner in which iterates are computed. The option does not enforce feasibility with respect to *equality* constraints, as this would impact performance too much. The theory behind feasible mode is described in [5].

The initial point must satisfy inequalities to a sufficient degree; if not, KNITRO may generate infeasible iterates and does not switch to the feasible mode until a sufficiently feasible point is found. We say *sufficient* satisfaction occurs at a point x if it is true for all inequalities that

$$cl + tol \leq c(x) \leq cu - tol \quad (9.20)$$

The constant $tol > 0$ is determined by the option `bar_feasmodetol`; its default value is `1.0e-4`. Feasible mode becomes active once an iterate x satisfies (9.20) for all inequality constraints. If the initial point satisfies (9.20), then every iterate will be feasible with respect to the inequalities.

KNITRO can also place special emphasis on *getting* feasible (with respect to all constraints) through the option `bar_feasible`. If `bar_feasible=2` or `bar_feasible=3`, KNITRO will first place special emphasis on getting feasible before working on optimality. This option is not always guaranteed to accelerate the finding of a feasible point. However, it may do a better job of obtaining feasibility on difficult problems where the default version struggles.

NOTE: This option can only be used with the Interior/Direct and Interior/CG algorithms.

9.3 Honor Bounds

In some applications, the user may want to enforce that the initial point and all subsequent iterates satisfy the simple bounds $bl \leq x \leq bu$. For instance, if the objective function or a nonlinear constraint function is undefined at points outside the bounds, then the bounds should be enforced at all times.

By default, KNITRO enforces bounds on the variables only for the initial start point and the final solution (`honorbnds=2`). To enforce satisfaction at all iterates, set `honorbnds=1`. To allow execution from an initial point that violates the bounds, set `honorbnds=0`.

9.4 Crossover

Interior-point (or barrier) methods are a powerful tool for solving large-scale optimization problems. However, one drawback of these methods is that they do not always provide a clear picture of which constraints are active at the solution. In general they return a less exact solution and less exact sensitivity information. For this reason, KNITRO offers a *crossover* feature in which the interior-point method switches to the Active Set method at the interior-point solution estimate, in order to “clean up” the solution and provide more exact sensitivity and active set information.

The crossover procedure is controlled by the `maxcrossit` user option. If this parameter is greater than 0, then KNITRO will attempt to perform `maxcrossit` Active Set crossover iterations after the interior-point method has finished, to see if it can provide a more exact solution. This can be viewed as a form of post-processing. If `maxcrossit` is not positive, then no crossover iterations are attempted.

The crossover procedure will not always succeed in obtaining a more exact solution compared with the interior-point solution. If crossover is unable to improve the solution within `maxcrossit` crossover iterations, then it will restore the interior-point solution estimate and terminate. If `outlev` is greater than one, KNITRO will print a message indicating that it was unable to improve the solution. For example, if `maxcrossit=3`, and the crossover procedure did not succeed, the message will read:

```
Crossover mode unable to improve solution within 3 iterations.
```

In this case, you may want to increase the value of `maxcrossit` and try again. If KNITRO determines that the crossover procedure will not succeed, no matter how many iterations are tried, then a message of the form

```
Crossover mode unable to improve solution.
```

will be printed.

The extra cost of performing crossover is problem dependent. In most small or medium scale problems, the crossover cost is a small fraction of the total solve cost. In these cases it may be worth using the crossover procedure to obtain a more exact solution. On some large scale or difficult degenerate problems, however, the cost of performing crossover may be significant. It is recommended to experiment with this option to see whether improvement in the exactness of the solution is worth the additional cost.

9.5 Multi-start

Nonlinear optimization problems (1.1) are often nonconvex due to the objective function, constraint functions, or both. When this is true, there may be many points that satisfy the local optimality conditions described in Section 6. Default KNITRO behavior is to return the first locally optimal point found. KNITRO offers a simple *multi-start* feature that searches for a better optimal point by restarting KNITRO from different initial points. The feature is enabled by setting `ms_enable=1`.

The multi-start procedure generates new start points by randomly selecting components of x that satisfy lower and upper bounds on the variables. KNITRO finds a local optimum from each start point using the same problem definition and user options. The final solution returned from `KTR.solve()` is the local optimum with the best objective function value if any local optimum have been found. If no local optimum have been found, KNITRO will return the best feasible solution estimate it found. If no feasible solution estimate has been found, KNITRO will return the least infeasible point. If you wish to see details of the local optimization process for each start point, then set option `outlev` to at least 4.

The number of start points tried by multi-start is specified with the option `ms_maxsolves`. By default, KNITRO will try $\min\{200, 10n\}$, where n is the number of variables in the problem. Users may override the default by setting `ms_maxsolves` to a specific value.

The multi-start option is convenient for conducting a simple search for a better solution point. Search time is improved if the variable bounds are made as tight as possible, confining the search

to a region where a good solution is likely to be found. The user can restrict the multi-start search region without altering bounds by using the options `ms_maxbndrange` and `ms_startptrange`. The first option applies to variables unbounded in at least one direction (i.e., the upper or lower bound (or both) is infinite) and keeps new start points within a total range equal to the value of `ms_maxbndrange`. The second option applies to all variables and keeps new start points within a total range equal to the value of `ms_startptrange`, overruling `ms_maxbndrange` if it is a tighter bound. In general, use `ms_startptrange` to limit the multi-start search only if the initial start point supplied by the user is known to be the center of a desired search area. Use `ms_maxbndrange` as a surrogate bound to limit the multi-start search when a variable is unbounded. See Section 5.1 for details.

The `ms_num_to_save` option allows a specific number of distinct feasible points to be saved in a file named `knitro_mspoints.log`. Each point results from a KNITRO solve from a different starting point, and must satisfy the absolute and relative feasibility tolerances. Different start points may return the same feasible point, and the file contains only distinct points. The option `ms_savetol` determines that two points are distinct if their objectives or any solution components (including Lagrange multipliers) are separated by more than the value of `ms_savetol` using a relative tolerance test. More specifically, two values x and y are considered distinct if

$$|x - y| \geq \max(1, |x|, |y|) * \text{ms_savetol}. \quad (9.21)$$

The file stores points in order from best objective to worst. If objectives are the same (as defined by `ms_savetol`), then points are ordered from smallest feasibility error to largest. The file can be read manually, but conforms to a fixed property/value format for machine reading.

Instead of using multi-start to search for a global solution, a user may want to use multi-start as a mechanism for finding any locally optimal or feasible solution estimate of a nonconvex problem and terminate as soon as one such point is found. The `ms_terminate` option, provides the user more control over when to terminate the multi-start procedure. If `ms_terminate=optimal` the multi-start procedure will stop as soon as the first locally optimal solution is found or after `ms_maxsolves` – whichever comes first. If `ms_terminate=feasible` the multi-start procedure will instead stop as soon as the first feasible solution estimate is found or after `ms_maxsolves` – whichever comes first. If `ms_terminate=maxsolves`, it will only terminate after `ms_maxsolves`.

In most cases the user would like to obtain the *global optimum* to (1.1); that is, the local optimum with the very best objective function value. KNITRO cannot guarantee that multi-start will find the global optimum. In general, the global optimum can only be found with special knowledge of the objective and constraint functions; for example, the functions may need to be bounded by other piece-wise convex functions. KNITRO executes with very little information about functional form. Although no guarantee can be made, the probability of finding a better local solution improves if more start points are tried. See Section 10.5 for more discussion.

10 Special problem classes

This section describes specializations in KNITRO to deal with particular classes of optimization problems. We also provide guidance on how to best set user options and model your problem to get the best performance out of KNITRO for particular types of problems.

10.1 Linear programming problems (LPs)

A linear program (LP) is an optimization problem where the objective function and all the constraint functions are linear. KNITRO has built in specializations for efficiently solving LPs.

10.2 Quadratic programming problems (QPs)

A quadratic program (QP) is an optimization problem where the objective function is quadratic and all the constraint functions are linear. KNITRO has built in specializations for efficiently solving QPs. Typically, these specialization will only help on convex QPs.

10.3 Systems of Nonlinear Equations

KNITRO is effective at solving systems of nonlinear equations. To solve a square system of nonlinear equations using KNITRO one should specify the nonlinear equations as equality constraints (i.e., $c^L = c^U$ in (1.1b)), and specify the objective function (1.1a) as zero (i.e., $f(x) = 0$).

If KNITRO is converging to a stationary point for which the nonlinear equations are not satisfied, the multi-start option described in Section 9.5, may help in finding a solution by trying different starting points.

10.4 Least Squares Problems

There are two ways of using KNITRO for solving problems in which the objective function is a sum of squares of the form

$$f(x) = \frac{1}{2} \sum_{j=1}^q r_j(x)^2.$$

If the value of the objective function at the solution is not close to zero (the large residual case), the least squares structure of f can be ignored and the problem can be solved as any other optimization problem. Any of the KNITRO options can be used.

On the other hand, if the optimal objective function value is expected to be small (small residual case) then KNITRO can implement the Gauss-Newton or Levenberg-Marquardt methods which only require first derivatives of the residual functions, $r_j(x)$, and yet converge rapidly. To do so, the user need only define the Hessian of f to be

$$\nabla^2 f(x) = J(x)^T J(x),$$

where

$$J(x) = \begin{bmatrix} \frac{\partial r_j}{\partial x_i} \end{bmatrix} \begin{matrix} j = 1, 2, \dots, q \\ i = 1, 2, \dots, n \end{matrix} .$$

The actual Hessian is given by

$$\nabla^2 f(x) = J(x)^T J(x) + \sum_{j=1}^q r_j(x) \nabla^2 r_j(x);$$

the Gauss-Newton and Levenberg-Marquardt approaches consist of ignoring the last term in the Hessian.

KNITRO will behave like a Gauss-Newton method by setting `Algorithm->"Direct"`, and will be very similar to the classical Levenberg-Marquardt method with `Algorithm->"CG"`. For a discussion of these methods see, for example, [9].

10.5 Global optimization

KNITRO is designed for finding locally optimal solutions of continuous optimization problems. A local solution is a feasible point at which the objective function value at that point is as good or better than at any “nearby” feasible point. A globally optimal solution is one which gives the best (i.e., lowest if minimizing) value of the objective function out of all feasible points. If the problem is *convex* all locally optimal solutions are also globally optimal solutions. The ability to guarantee convergence to the global solution on large-scale *nonconvex* problems is a nearly impossible task on most problems unless the problem has some special structure or the person modeling the problem has some special knowledge about the geometry of the problem. Even finding local solutions to large-scale, nonlinear, nonconvex problems is quite challenging.

Although KNITRO is unable to guarantee convergence to global solutions it does provide a *multi-start* heuristic which attempts to find multiple local solutions in the hopes of locating the global solution. See Section 9.5 for information on trying to find the globally optimal solution using the KNITRO multi-start feature.

10.6 Mixed integer programming (MIP)

KNITRO provides tools for solving optimization models (both linear and nonlinear) with binary or integer variables. The KNITRO mixed integer programming (MIP) code offers two algorithms for mixed-integer nonlinear programming (MINLP). The first is a nonlinear branch and bound method and the second implements the hybrid Quesada-Grossman method for convex MINLP.

The KNITRO MINLP code is designed for convex mixed integer programming and is a heuristic for nonconvex problems. The MIP code also handles mixed integer linear programs (MILP) of moderate size. A MIP problem is defined and solved via the built-in functions `KnitroMipMinimize[]` and `KnitroMipMaximize[]`. The KNITRO MIP tools do not currently handle special ordered sets (SOS’s) or semi-continuous variables.

Many user options are provided for the MIP features to tune performance, including options for branching, node selection, rounding and heuristics for finding integer feasible points. User options specific to the MIP tools begin with `mip_` (see Section 5). It is recommended to experiment with several of these options as they often can make a significant difference in performance. In particular, if finding any integer feasible point is your highest priority, you should set the `mip_heuristic` option to search for an integer feasible point before beginning the branch and bound procedure (by default no heuristics are applied).

The MIP features are new in KNITRO 6.0.

References

- [1] R. H. Byrd, J.-Ch. Gilbert, and J. Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89(1):149–185, 2000.
- [2] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz. On the convergence of successive linear-quadratic programming algorithms. *SIAM Journal on Optimization*, 16(2):471–489, 2006.
- [3] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz. An algorithm for nonlinear optimization using linear programming and equality constrained subproblems. *Mathematical Programming, Series B*, 100(1):27–48, 2004.
- [4] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.
- [5] R. H. Byrd, J. Nocedal, and R. A. Waltz. Feasible interior methods using slacks for nonlinear optimization. *Computational Optimization and Applications*, 26(1):35–61, 2003.
- [6] R. H. Byrd, J. Nocedal, and R.A. Waltz. KNITRO: An integrated package for nonlinear optimization. In G. di Pillo and M. Roma, editors, *Large-Scale Nonlinear Optimization*, pages 35–59. Springer, 2006.
- [7] Harwell Subroutine Library. *A catalogue of subroutines (HSL 2002)*. AEA Technology, Harwell, Oxfordshire, England, 2002.
- [8] Hock, W. and Schittkowski, K. *Test Examples for Nonlinear Programming Codes*, volume 187 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, 1981.
- [9] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, 1999.
- [10] R. A. Waltz, J. L. Morales, J. Nocedal, and D. Orban. An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical Programming A*, 107(3):391–408, 2006.

Solution Status Codes

The solution status return codes are organized as follows.

- 0:** The final solution satisfies the termination conditions for verifying optimality.
- 100 to -199:** A feasible approximate solution was found.
- 200 to -299:** The code terminated at an infeasible point.
- 300:** The problem was determined to be unbounded.
- 400 to -499:** The code terminated because it reached a pre-defined limit.
- 500 to -599:** The code terminated with an input error or some non-standard error.

A more detailed description of individual return codes and their corresponding termination messages is provided below.

0 (KTR_RC_OPTIMAL):

Locally optimal solution found.

KNITRO found a locally optimal point which satisfies the stopping criterion (see Section 6 for more detail on how this is defined). If the problem is convex (for example, a linear program), then this point corresponds to a globally optimal solution.

-100 (KTR_RC_NEAR_OPT):

Primal feasible solution estimate cannot be improved. It appears to be optimal, but desired accuracy in dual feasibility could not be achieved.

No more progress can be made, but the stopping tests are close to being satisfied (within a factor of 100) and so the current approximate solution is believed to be optimal.

-101 (KTR_RC_FEAS_XTOL):

Primal feasible solution; terminate because the relative change in solution estimate < xtol. Decrease xtol to try for more accuracy.

The optimization terminated because the relative change in the solution estimate is less than that specified by the parameter xtol. To try to get more accuracy one may decrease xtol. If xtol is very small already, it is an indication that no more significant progress can be made. It's possible the approximate feasible solution is optimal, but perhaps the stopping tests cannot be satisfied because of degeneracy, ill-conditioning or bad scaling.

-102 (KTR_RC_FEAS_NO_IMPROVE):

Primal feasible solution estimate cannot be improved; desired accuracy in dual feasibility could not be achieved.

No further progress can be made. It's possible the approximate feasible solution is optimal, but perhaps the stopping tests cannot be satisfied because of degeneracy, ill-conditioning or bad scaling.

-200 (KTR_RC_INFEASIBLE):

Convergence to an infeasible point. Problem may be locally infeasible. If problem is believed to be feasible, try `multistart` to search for feasible points.

The algorithm has converged to an infeasible point from which it cannot further decrease the infeasibility measure. This happens when the problem is infeasible, but may also occur on occasion for feasible problems with nonlinear constraints or badly scaled problems. It is recommended to try various initial points with the multi-start feature described in Section 9.5. If this occurs for a variety of initial points, it is likely the problem is infeasible.

-201 (KTR_RC_INFEAS_XTOL):

Terminate at infeasible point because the relative change in solution estimate $<$ `xtol`. Decrease `xtol` to try for more accuracy.

The optimization terminated because the relative change in the solution estimate is less than that specified by the parameter `xtol`. To try to find a feasible point one may decrease `xtol`. If `xtol` is very small already, it is an indication that no more significant progress can be made. It is recommended to try various initial points with the multi-start feature described in Section 9.5. If this occurs for a variety of initial points, it is likely the problem is infeasible.

-202 (KTR_RC_INFEAS_NO_IMPROVE):

Current infeasible solution estimate cannot be improved. Problem may be badly scaled or perhaps infeasible. If problem is believed to be feasible, try `multistart` to search for feasible points.

No further progress can be made. It is recommended to try various initial points with the multi-start feature described in Section 9.5. If this occurs for a variety of initial points, it is likely the problem is infeasible.

-203 (KTR_RC_INFEAS_MULTISTART):

`MULTISTART`: No primal feasible point found.

The multi-start feature was unable to find a feasible point. If the problem is believed to be feasible, then increase the number of initial points tried in the multi-start feature and also perhaps increase the range from which random initial points are chosen. See Section 9.5 for more details about multi-start and Section 5.1 for various multi-start user options.

-300 (KTR_RC_UNBOUNDED):

Problem appears to be unbounded. Iterate is feasible and objective magnitude $>$ `objrange`.

The objective function appears to be decreasing without bound, while satisfying the constraints. If the problem really is bounded, increase the size of the parameter `objrange` to avoid terminating with this message.

-400 (KTR_RC_ITER_LIMIT):

Iteration limit reached.

The iteration limit was reached before being able to satisfy the required stopping criteria. The iteration limit can be increased through the user option `maxit`. See Section 5.1.

-401 (KTR_RC.TIME_LIMIT):

Time limit reached.

The time limit was reached before being able to satisfy the required stopping criteria. The time limit can be increased through the user options `maxtime_cpu` and `maxtime_real`. See Section 5.1.

-403 (KTR_RC.MIP_EXH):

All nodes have been explored.

The MIP optimality gap has not been reduced below the specified threshold, but there are no more nodes to explore in the branch and bound tree. If the problem is convex, this could occur if the gap tolerance is difficult to meet because of bad scaling or roundoff errors, or there was a failure at one or more of the subproblem nodes. This might also occur if the problem is nonconvex. In this case, KNITRO terminates and returns the best integer feasible point found.

-404 (KTR_RC.MIP_FEAS_TERM):

Terminating at first integer feasible point.

KNITRO has found an integer feasible point and is terminating because the user option `mip_terminate = feasible`. See Section 5.1.

-405 (KTR_RC.MIP_SOLVE_LIMIT):

Subproblem solve limit reached.

The MIP subproblem solve limit was reached before being able to satisfy the optimality gap tolerance. The subproblem solve limit can be increased through the user option `mip_maxsolves`. See Section 5.1.

-406 (KTR_RC.MIP_NODE_LIMIT):

Node limit reached.

The MIP node limit was reached before being able to satisfy the optimality gap tolerance. The node limit can be increased through the user option `mip_maxnodes`. See Section 5.1.

-500 (KTR_RC.CALLBACK_ERR):

Callback function error.

This termination value indicates that an error (i.e., negative return value) occurred in a user provided callback routine.

-501 (KTR_RC.LP_SOLVER_ERR):

LP solver error.

This termination value indicates that an unrecoverable error occurred in the LP solver used in the active-set algorithm preventing the optimization from continuing.

-502 (KTR_RC.EVAL_ERR):

Evaluation error.

This termination value indicates that an evaluation error occurred (e.g., divide by 0, taking the square root of a negative number), preventing the optimization from continuing.

-503 (KTR_RC_OUT_OF_MEMORY):

Not enough memory available to solve problem.

This termination value indicates that there was not enough memory available to solve the problem.

-505 to -600:

Termination values in this range imply some input error or other non-standard failure. If `outlev>0` details of this error will be printed to standard output or the file `knitro.log` depending on the value of `outmode`.

List of Output Files

knitro.log

This is the standard output from KNITRO. The file is created if `outmode=file` or `outmode=both`. See Section 7 for an explanation of the contents.

knitro_mspoints.log

This file contains a set of feasible points found by multi-start, each distinct, in order of best to worst. The file is created if `ms_enable=yes` and `ms_num_to_save` is greater than zero. See Section 9.5 for more information.

knitro_newpoint.log

This file contains a set of iterates generated by KNITRO. It is created if `newpoint` equals `saveone` or `saveall`.

knml_license_debug.txt

This file contains information to help debug Ziena license issues. It is created if environment variable `ZIENA_LICENSE_DEBUG=1`. See the *Ziena License Manager User's Manual* for more information.

kdbg_barrierIP.log

kdbg_directIP.log

kdbg_normalIP.log

kdbg_profileIP.log

kdbg_stepIP.log

kdbg_summIP.log

kdbg_tangIP.log

These files contain detailed debug information. The files are created if `debug=problem` and either barrier method (`Interior/Direct` or `Interior/CG`) executes. The `kdbg_directIP.log` file is created only for the `Interior/Direct` method.

kdbg_actsetAS.log

kdbg_eqpAS.log

kdbg_lpAS.log

kdbg_profileAS.log

kdbg_stepAS.log

kdbg_summAS.log

These files contain detailed debug information. The files are created if `debug=problem` and the `Active Set` method executes.

kdbg_mip.log

This file contains detailed debug information. The file is created if `mip_debug=all` and one of the MIP methods executes.