# KNITRO

## Contents

## 1 Introduction

Knitro is a software package for finding local solutions of both continuous (i.e. smooth) optimization problems, with or without constraints, and discrete optimization problems with integer or binary variables. Even though Knitro has been designed for solving large-scale general problems, it is efficient for solving all of the following classes of optimization problems:

- unconstrained,
- bound constrained,
- equality constrained,
- systems of nonlinear equations,

- least squares problems,
- linear programming problems (LPs),
- quadratic programming problems (QPs),
- general (inequality) constrained problems,
- (convex) mixed integer 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,
- Both interior-point (barrier) and active-set optimizers,
- Both feasible and infeasible versions,
- Both iterative and direct approaches for computing steps,

The problems solved by KNITRO have the form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) && (1.1a) \\ \text{subject to} \quad & c^L \leq c(x) \leq c^U && (1.1b) \\ & b^L \leq x \leq b^U, && (1.1c) \end{aligned}$$

where the variables $x$ can be continuous, binary, or integer. This allows many forms of constraints, including bounds on the variables. KNITRO requires that the functions $f(x)$ and $c(x)$ be smooth functions.

KNITRO implements both state-of-the-art interior-point and active-set methods for solving nonlinear optimization problems. In the interior method (also known as a barrier method), the nonlinear programming problem is replaced by a series of barrier sub-problems controlled by a barrier parameter $\mu$. The algorithm uses trust regions and a merit function to promote convergence. The algorithm performs one or more minimization steps on each barrier problem, then decreases the barrier parameter, and repeats the process until the original problem (1.1) has been solved to the desired accuracy.

KNITRO provides two procedures for computing the steps within the interior point approach. In the version known as `Interior/CG` each step is computed using a projected conjugate gradient iteration. This approach differs from most interior methods proposed in the literature in that it does not compute each step by solving a linear system involving the KKT (or primal-dual) matrix. Instead, it factors a projection matrix, and uses the conjugate gradient method, to approximately minimize a quadratic model of the barrier problem.

The second procedure for computing the steps, which we call `Interior/Direct`, always attempts to compute a new iterate by solving the primal-dual KKT matrix using direct linear algebra. In the case when this step cannot be guaranteed to be of good quality, or if negative curvature is detected, then the new iterate is computed by the `Interior/CG` procedure.

KNITRO also implements an active-set sequential linear-quadratic programming (SLQP) algorithm which we call `Active`. This method is similar in nature to a sequential quadratic programming method but uses linear programming sub-problems to estimate the active-set at each iteration. This active-set code may be preferable when a good initial point can be provided, for example, when solving a sequence of related problems.

For problems with discrete variables, 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.

*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.*

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 [8]. The `Active` algorithm is described in [3] and the global convergence theory for this algorithm is in [2]. An important component of KNITRO is the HSL routine `MA27` [6] which is used to solve the linear systems arising at every iteration of the algorithm. In addition, the `Active` 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`.

## 2 Usage

Basic details of solver usage, including how to choose KNITRO as the solver and how to use a solver-specific option file, are part of Chapter 0 "Basic Solver Usage".

As an NLP solver, KNITRO can also be used to solve linear programs (LP), and both convex and nonconvex quadratic programs (QCP).

## 3 GAMS Options

The following GAMS options are used by the GAMS/KNITRO link:

**Option ResLim = x;**

Sets the time limit in seconds. If this limit is exceeded the solver will terminate and pass on the current solution to GAMS.

**Option SysOut = On;**

This option sends additional KNITRO messages to the GAMS listing file. It is useful in case of a solver failure or to get algorithmic details.

***ModelName*.optCA = x;**

Absolute gap stop criterion for a discrete problem. The KNITRO option mip_integral_gap_abs takes its default from this value.

***ModelName*.optCR = x;**

Relative gap stop criterion for a discrete problem. The KNITRO option mip_integral_gap_rel takes its default from this value.

## 4 Summary of KNITRO Options

The KNITRO options file knitro.opt allows the user to easily set options controlling KNITRO's behavior. Options are set by specifying a keyword and a corresponding value on a line in the knitro.opt file. Lines that begin with a # character are treated as comments and blank lines are ignored. For example, to set the maximum allowable number of iterations to 500, one could use the following options file:

```
# KNITRO-GAMS Options file
maxit        500
```

### 4.1 General Options

| | |
|---|---|
| algorithm | controls which NLP algorithm to use |
| delta | initial trust region radius scaling factor |
| feastol | controls the relative feasibility tolerance |
| feastolabs | controls the absolute feasibility tolerance |
| gradopt | controls how to compute gradients |
| hessopt | controls how to compute Hessians |
| honorbnds | controls satisfaction of variable bounds |
| linsolver | controls which linear system solver to use |
| maxcgit | controls CG iteration limit |
| maxcrossit | controls crossover iteration limit |
| maxit | controls iteration limit |

| | |
|---|---|
| maxtime_cpu | CPU time limit |
| maxtime_real | real or wall-clock time limit |
| objrange | controls unboundedness check limit |
| opttol | controls the relative optimality tolerance |
| opttolabs | controls the absolute optimality tolerance |
| outlev | controls the output level |
| pivot | controls the initial pivot threshold |
| scale | controls scaling of model |
| soc | controls second order correction steps |
| xtol | controls termination based on stepsize |

## 4.2 Barrier Options

| | |
|---|---|
| bar_feasible | control for entering feasible mode |
| bar_feasmodetol | feasible mode tolerance |
| bar_initmu | control initial barrier parameter value |
| bar_initpt | control initial point strategy |
| bar_murule | control barrier parameter update strategy |

## 4.3 MINLP Options

**NOTE**: the KNITRO library uses the `mip_` prefix for options and calls that are specific to problems with binary or integer variable, including mixed-integer nonlinear problems. This is in constrast to the GAMS convention, where `MIP` denotes a mixed-integer linear program and `MINLP` denotes a mixed-integer nonlinear program. We use the KNITRO convention to avoid changing the option names.

| | |
|---|---|
| mip_branchrule | to use for MIP B&B |
| mip_gub_branch | toggles branching on generalized upper bounds |
| mip_heuristic | used in searching for an initial integer feasible point |
| mip_heuristic_maxit | iteration limit for heuristic |
| mip_implications | toggles addition of derived constraints |
| mip_integer_tol | integrality tolerance for discrete vars |
| mip_integral_gap_abs | absolute gap stop tolerance |
| mip_integral_gap_rel | relative gap stop tolerance |
| mip_lpalg | LP subsolver to use |
| mip_maxnodes | limit number of nodes explored |
| mip_maxsolves | limit subproblem solves allowed |
| mip_maxtime_cpu | cumulative CPU time limit |
| mip_maxtime_real | cumulative real or wall-clock time limit |
| mip_method | controls method to use |
| mip_outinterval | controls output frequency |
| mip_outlevel | controls output level |
| mip_rootalg | controls algorithm used for root node solve |
| mip_rounding | controls which rounding rule to apply |
| mip_selectrule | controls node selection rule |
| mip_strong_candlim | candidate limit for strong branching |
| mip_strong_level | controls tree levels for strong branching |
| mip_strong_maxit | iteration limit for strong branching |
| mip_terminate | controls termination test |

## 4.4 Multi-Start Options

| | |
|---|---|
| ms_enable | toggles the multi-start method |
| ms_maxbndrange | Maximum range to vary unbounded $x$ when generating start points |
| ms_maxsolves | Specifies the maximum number of start points to try |
| ms_maxtime_cpu | cumulative CPU time limit |
| ms_maxtime_real | cumulative real or wall-clock time limit |
| ms_startptrange | Maximum range to vary all $x$ when generating start points |
| ms_terminate | controls termination test |

## 5 Detailed Descriptions of Knitro Options

**algorithm *(integer)***

Controls which NLP algorithm to use.

0 Knitro will automatically try to choose the best algorithm based on the problem characteristics

1 Knitro will use the Interior/Direct algorithm

2 Knitro will use the Interior/CG algorithm

3 Knitro will use the Active Set algorithm

*(default = 0)*

**bar_feasible *(integer)***

Indicates whether or not to use the feasible version of Knitro. **NOTE**: This option can be used only with the Interior/CG and Interior/Direct algorithms, i.e. when `algorithm=2` or `3`. See section 9.2 for more details.

0 No special emphasis on feasibility.

1 Iterates must satisfy inequality constraints once they become sufficiently feasible.

2 Special emphasis is placed on getting feasible before trying to optimize.

3 Implement both options 1 and 2 above.

Options 1 and 3 above activate the feasible version of Knitro. Given an initial point which *sufficiently* satisfies all *inequality* constraints as defined by,

$$cl + tol \le c(x) \le cu - tol \tag{5.2}$$

(for $cl \neq cu$), the feasible version of Knitro ensures that all subsequent solution estimates strictly satisfy the *inequality* constraints. However, the iterates may not be feasible with respect to the *equality* constraints. The tolerance $tol > 0$ in (5.2) for determining when the feasible mode is active is determined by the double precision parameter `bar_feasmodetol` described below. This tolerance (i.e. `bar_feasmodetol`) must be strictly positive. That is, in order to enter feasible mode, the point given to Knitro must be strictly feasible with respect to the inequality constraints.

If the initial point is infeasible (or not sufficiently feasible according to (5.2)) with respect to the *inequality* constraints, then Knitro will run the infeasible version until a point is obtained which sufficiently satisfies all the *inequality* constraints. At this point it will switch to feasible mode.

*(default = 0)*

**bar_feasmodetol *(double)***

Specifies the tolerance in (5.2) by which the iterate must be feasible with respect to the inequality constraints before the feasible mode becomes active. This option is only relevant when *feasible*=1.

*(default = 1e-4)*

**bar_initmu** ***(double)***

Specifies the initial value for the barrier parameter $\mu$.

*(default = 1e-1)*

**bar_initpt** ***(integer)***

Indicates whether an initial point strategy is used.

0 Knitro will automatically choose the initial point strategy

1 Shift the initial point to improve barrier algorithm performance

3 Do not alter the initial point supplied by the user

*(default = 0)*

**bar_murule** ***(integer)***

Controls the barrier parameter update strategy.

0 Knitro will automatically choose the rule for updating the barrier parameter

1 Knitro will monotonically decrease the barrier parameter

2 Knitro uses an adaptive rule based on the complementarity gap to determine the value of the barrier parameter at every iteration

3 Knitro uses a probing (affine-scaling) step to dynamically determine the barrier parameter value at each iteration

4 Knitro uses a Mehrotra predictor-corrector type rule to determine the barrier parameter with safeguards on the corrector step

5 Knitro uses a Mehrotra predictor-corrector type rule to determine the barrier parameter without safeguards on the corrector step

6 Knitro minimizes a quality function at each iteration to determine the barrier parameter

**NOTE**: Only strategies 0-2 are available for the Interior/CG algorithm. All strategies are available for the Interior/Direct algorithm. Strategies 4 and 5 are typically recommended for linear programs or convex quadratic programs.

*(default = 0)*

**delta** ***(double)***

Specifies the initial trust region radius scaling factor used to determine the initial trust region size.

*(default = 1)*

**feastol** ***(double)***

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.

*1.0e-6*

**feastolabs** ***(double)***

Specifies the final absolute stopping tolerance for the feasibility error. Smaller values of `feastolabs` result in a higher degree of accuracy in the solution with respect to feasibility.

*0.0*

**gradopt** ***(integer)***

Specifies how to compute the gradients of the objective and constraint functions.

1 exact gradients computed by GAMS

2 gradients computed by forward finite differences

3 gradients computed by central finite differences

*(default = 1)*

**hessopt** ***(integer)***

Specifies how to compute the (approximate) Hessian of the Lagrangian.

1 exact Hessians computed by GAMS

2 KNITRO will compute a (dense) quasi-Newton BFGS Hessian

3 KNITRO will compute a (dense) quasi-Newton SR1 Hessian

4 KNITRO will compute Hessian-vector products using finite-differences

5 exact Hessian-vector products computed by GAMS

6 KNITRO will compute a limited-memory quasi-Newton BFGS Hessian

**NOTE**: In nearly all cases it is strongly recommended to use the exact Hessian option (option 1) or the exact Hessian-vector product option (option 5).

If exact Hessians (or exact Hessian-vector products) are not efficient to compute but exact gradients are provided and are not too expensive to compute, option 4 above is typically recommended. The finite-difference Hessian-vector option is comparable in terms of robustness to the exact Hessian option (*assuming exact gradients are provided*) and typically not too much slower in terms of time if gradient evaluations are not the dominant cost.

In the event that the exact Hessian (or Hessian-vector products) are too expensive to compute, multiple quasi-Newton options which internally approximate the Hessian matrix using first derivative information are provided. Options 2 and 3 are only recommended for small problems ($n < 1000$) since they require working with a dense Hessian approximation. Option 6 should be used in the large-scale case.

**NOTE**: Options `hessopt=4` and `hessopt=5` are not available when `algorithm=1`. See section 9.1 for more detail on second derivative options.

*(default = 1)*

**honorbnds** ***(integer)***

Indicates whether or not to enforce satisfaction of the simple bounds (1.1*c*) throughout the optimization (see section 9.3).

0 KNITRO does not enforce that the bounds on the variables are satisfied at intermediate iterates.

1 KNITRO enforces that the initial point and all subsequent solution estimates satisfy the bounds on the variables (1.1*c*).

2 KNITRO enforces that the initial point satisfies the bounds on the variables (1.1*c*).

*(default = 0)*

**linsolver** ***(integer)***

Indicates which linear solver to use to solve linear systems arising in KNITRO algorithms.

0 **auto**: let KNITRO automatically choose the linear solver.

1 **internal**: not currently used; reserved for future use. Same as auto for now.

2 **hybrid**: use a hybrid approach where the solver chosen depends on the particular linear system which needs to be solved.

3 **QR**: use a dense QR method. This approach uses LAPACK QR routines. Since it uses a dense method, it is only efficient for small problems. It may often be the most efficient method for small problems with dense Jacobians or Hessian matrices.

4 **MA27**: use the HSL MA27 sparse symmetric indefinite solver.

5 **MA57**: use the HSL MA57 sparse symmetric indefinite solver.

*(default = 0)*

**maxcgit** ***(integer)***

Specifies the maximum allowable number of inner conjugate gradient (CG) iterations per KNITRO minor iteration.

0 KNITRO automatically determines an upper bound on the number of allowable CG iterations based on the problem size.

$n$ At most $n$ CG iterations may be performed during one KNITRO minor iteration, where $n > 0$.

*(default = 0)*

**maxcrossit** ***(integer)***

Specifies the maximum number of crossover iterations before termination. If the value is positive, then KNITRO will crossover from the barrier to the Active Set algorithm near the solution. The Active Set algorithm will then perform at most $n$ 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.

*(default = 0)*

**maxit** ***(integer)***

Specifies the maximum number of iterations before termination.

0 KNITRO automatically determines a value based on the problem size. Currently KNITRO 7.0 sets this value to 10000 for LPs/NLPs and 3000 for MIPs/MINLPs.

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

*(default = 0)*

**maxtime_cpu** ***(double)***

Specifies the CPU time limit, in seconds.

*(default = 1e8)*

**maxtime_real** ***(double)***

Specifies the real or wall-clock time limit, in seconds.

*(default = 1e8)*

**mip_branchrule** ***(integer)***

Branching rule to use for MIP B&B.

0 automatic

1 use most fractional (most infeasible) branching

2 use pseudo-cost branching

3 use strong branching

*(default = 0)*

**mip_gub_branch** ***(boolean)***

Toggles branching on generalized upper bounds.

*(default = false)*

**mip_heuristic (*integer*)**

Heuristic to use in searching for an initial integer feasible point.

0 automatic

1 none

2 feasibility pump

3 heuristic based on MPEC formulation

*(default = 0)*

**mip_heuristic_maxit (*integer*)**

Specifies the maximum number of iterations to allow for MIP heuristic, if one is enabled.

*(default = 100)*

**mip_implications (*boolean*)**

Toggles addition of constraints derived from logical implications.

*(default = true)*

**mip_integer_tol (*double*)**

Specifies the integrality tolerance for discrete variables.

*(default = 1e-8)*

**mip_integral_gap_abs (*double*)**

The absolute integrality gap stop tolerance. If not set by the user, the GAMS optCA value is used.

*(default = GAMS optCA)*

**mip_integral_gap_rel (*double*)**

The relative integrality gap stop tolerance. If not set by the user, the GAMS optCR value is used.

*(default = GAMS optCR)*

**mip_lpalg (*integer*)**

Specifies which algorithm to use for any LP subproblem solves that may occur in the B&B procedure. LP subproblems may arise if the problem has no nonlinear parts or if using `mip_method=2`.

0 KNITRO will automatically try to choose the best algorithm based on the problem characteristics

1 use the Interior/Direct algorithm

2 use the Interior/CG algorithm

3 use the Active Set (simplex) algorithm

*(default = 0)*

**mip_maxnodes (*integer*)**

Specifies the maximum number of nodes explored (0 means no limit).

*(default = 100000)*

**mip_maxsolves (*integer*)**

Specifies the maximum number of subproblem solves allowed (0 means no limit).

*(default = 200000)*

**mip_maxtime_cpu (*double*)**

Specifies the cumulative CPU time limit, in seconds.

*(default = 1e8)*

**mip_maxtime_real (*double*)**

Specifies the cumulative real or wall-clock time limit, in seconds.

*(default = 1e8)*

**mip_method (*integer*)**

Specifies which method to use.

| | |
|---|---|
| 0 | automatic |
| 1 | use the standard B&B method |
| 2 | use the hybrid Quesada-Grossman method (for convex, nonlinear problems only) |

*(default = 0)*

**mip_outinterval (*integer*)**

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 $n$'th node |

*(default = 10)*

**mip_outlevel (*integer*)**

Specifies how much MIP information to print.

| | |
|---|---|
| 0 | do not print any MIP node information |
| 1 | print one line of output for every node |

*(default = 1)*

**mip_rootalg (*integer*)**

Specifies which algorithm to use for the root node solve.

| | |
|---|---|
| 0 | Knitro will automatically try to choose the best algorithm based on the problem characteristics |
| 1 | Knitro will use the Interior/Direct algorithm |
| 2 | Knitro will use the Interior/CG algorithm |
| 3 | Knitro will use the Active Set algorithm |

*(default = 0)*

**mip_rounding (*integer*)**

Specifies the rounding rule to apply.

| | |
|---|---|
| 0 | automatic |
| 1 | do not round if a node is infeasible |
| 2 | round using a fast heuristic only |
| 3 | round and solve a subproblem if likely to succeed |
| 4 | always round and solve a subproblem |

*(default = 0)*

**mip_selectrule (*integer*)**

Specifies the select rule for choosing the next node in the tree.

0 automatic
1 search the tree using a depth first procedure
2 select the node with the best relaxation bound
3 use depth first unless pruned, then best bound

*(default = 0)*

**mip_strong_candlim (*integer)***

Specifies the maximum number of candidates to explore for strong branching.

*(default = 10)*

**mip_strong_level (*integer)***

Specifies the maximum number of tree levels on which to perform strong branching.

*(default = 10)*

**mip_strong_maxit (*integer)***

Specifies the maximum number of iterations to allow for strong branching.

*(default = 1000)*

**mip_terminate (*integer)***

Specifies conditions for terminating the MIP algorithm.

0 terminate at optimum
1 terminate at first integer feasible point

*(default = 0)*

**ms_enable (*boolean)***

Toggles multi-start method.

*(default = false)*

**ms_maxbndrange (*double)***

Maximum range to vary unbounded $x$ when generating start points.

*(default = 1e3)*

**ms_maxsolves (*integer)***

Specifies the maximum number of start points to try during multi-start.

0 Knitro sets the number based on problem size
$n$ try exactly $n > 0$ start points

*(default = 200000)*

**ms_maxtime_cpu (*double)***

Specifies the cumulative CPU time limit, in seconds.

*(default = 1e8)*

**ms_maxtime_real (*double)***

Specifies the cumulative real or wall-clock time limit, in seconds.

*(default = 1e8)*

**ms_startptrange (*double)***

Maximum range to vary all $x$ when generating start points.

*(default = 1e20)*

**ms_terminate (*integer*)**

Specifies conditions for terminating the multi-start algorithm.

0 terminate after msmaxsolves

1 terminate at first local optimum (if before msmaxsolves)

2 terminate at first feasible solution (if before msmaxsolves)

*(default = 0)*

**objrange (*double*)**

Specifies the extreme limits of the objective function for purposes of determining unboundedness. If the magnitude of the objective function is greater than `objrange` and the iterate is feasible, then the problem is determined to be unbounded and Knitro proceeds no further.

*(default = 1e20)*

**opttol (*double*)**

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.

*(default = 1e-6)*

**opttolabs (*double*)**

Specifies the final absolute stopping tolerance for the KKT (optimality) error. Smaller values of `opttolabs` result in a higher degree of accuracy in the solution with respect to optimality.

*(default = 0.0)*

**outlev (*integer*)**

controls the level of output.

0 printing of all output is suppressed

1 print only summary information

2 print basic information every 10 iterations

3 print basic information at each iteration

4 print basic information and the function count at each iteration

5 print all of the above, and the values of the solution vector `x`

6 print all of the above, and the values of the constraints `c` and the Lagrange multipliers `lambda`

*(default = 2)*

**pivot (*double*)**

Specifies the initial pivot threshold used in the factorization routine. The value should be in the range $[0 \ldots 0.5]$ 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 = 1e-8)*

**scale (*integer*)**

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 scaling is performed.

1 The objective function and constraints may be scaled.

*(default = 1)*

**soc (*integer*)**

Specifies whether or not to try second order corrections (SOC). A second order correction may be beneficial for proglems with highly nonlinear constraints.

0 No second order correction steps are attempted

1 Second order correction steps may be attempted on some iterations.

2 Second order correction steps are always attempted if the original step is rejected and there are nonlinear constraints.

*(default = 1)*

**xtol (*double*)**

Specifies when to terminate the optimization based on stepsize. The optimization will terminate when the relative change in the solution estimate is less than `xtol`. If using an interior-point algorithm and the barrier parameter is still large, KNITRO will first try decreasing the barrier parameter before terminating.

*(default = 1e-15)*

## 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:

$$\begin{aligned}
\nabla_x \mathcal{L}(x,\lambda) = \nabla f(x) + \sum_{i=1\dots m} \lambda_i^c \nabla c_i(x) + \sum_{j=1\dots n} \lambda_j^b \quad &= \quad 0 && (6.3)\\
\lambda_i^c \min[(c_i(x) - c_i^L), (c_i^U - c_i(x))] \quad &= \quad 0, \quad i = 1\dots m && (6.4)\\
\lambda_j^b \min[(x_j - b_j^L), (b_j^U - x_j)] \quad &= \quad 0, \quad j = 1\dots n && (6.5)\\
c_i^L \le c_i(x) \quad &\le \quad c_i^U, \quad i = 1\dots m && (6.6)\\
b_j^L \le x_j \quad &\le \quad b_j^U, \quad j = 1\dots n && (6.7)\\
\lambda_i^c \quad &\ge \quad 0, \quad i \in \mathcal{I},\ c_i^L = -\infty,\ c_i^U \text{finite} && (6.8)\\
\lambda_i^c \quad &\le \quad 0, \quad i \in \mathcal{I},\ c_i^U = \infty,\ c_i^L \text{finite} && (6.9)\\
\lambda_j^b \quad &\ge \quad 0, \quad j \in \mathcal{B},\ b_j^L = -\infty,\ b_j^U \text{finite} && (6.10)\\
\lambda_j^b \quad &\le \quad 0, \quad j \in \mathcal{B},\ b_j^U = \infty,\ b_j^L \text{finite} && (6.11)
\end{aligned}$$

where $\mathcal{I}$ and $\mathcal{B}$ represent the sets of indices corresponding to the general inequality constraints and non-fixed variable bound constraints respectively, $\lambda_i^c$ is the Lagrange multiplier corresponding to constraint $c_i(x)$, and $\lambda_j^b$ is the Lagrange multiplier corresponding to the simple bounds on $x_j$. 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 the corresponding constraint or variable bounds, as indicated in (6.8) - (6.11).

In KNITRO we define the feasibility error (`FeasErr`) at a point $x^k$ to be the maximum violation of the constraints (6.7), (6.8), i.e.,

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

while the optimality error (`OptErr`) is defined as the maximum violation of the first three conditions (6.3) - (6.5):

$$\text{OptErr} = \max_{i=1\dots m,\ j=1\dots n} (\|\nabla_x \mathcal{L}(x^k,\lambda^k)\|_\infty, \lambda_i^c \min[(c_i(x) - c_i^L), (c_i^U - c_i(x))], \lambda_j^b \min[(x_j - b_j^L), (b_j^U - x_j)]). \tag{6.13}$$

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 In order to take into account problem scaling in the termination test, the following scaling factors are defined

$$\tau_1 = \max(1, (c_i^L - c_i(x^0)), (c_i(x^0) - c_i^U), (b_j^L - x_j^0), , (x_j^0 - b_j^U))), \tag{6.14}$$

$$\tau_2 = \max(1, \|\nabla f(x^k)\|_\infty), \tag{6.15}$$

where $x^0$ represents the initial point.

For unconstrained problems, the scaling (6.15) 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

$$\tau_2 = \max(1, \min(|f(x^k)|, \|\nabla f(x^0)\|_\infty)), \tag{6.16}$$

in place of (6.15).

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

$$\text{FeasErr} \le \max(\tau_1 * \texttt{feastol}, \texttt{feastolabs}) \tag{6.17}$$

$$\text{OptErr} \le \max(\tau_2 * \texttt{opttol}, \texttt{opttolabs}) \tag{6.18}$$

where `feastol`, `opttol`, `feastolabs` and `opttolabs` are user-defined options (see section 2).

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 purely scaled stopping test (which is the recommended and default option) by setting `feastolabs` and `opttolabs` equal to `0.0e0`. Likewise, an absolute stopping test can be enforced by setting `feastol` and `opttol` equal to `0.0e0`.

#### Unbounded problems

Since by default KNITRO uses a relative/scaled stopping test it is possible for the optimality conditions to be satisfied for an unbounded problem. For example, if $\tau_2 \rightarrow \infty$ while the optimality error (6.13) stays bounded, condition (6.18) 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 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

$$f(x_I) - f(x_R) \le \texttt{mip\_integral\_gap\_abs} \tag{6.19}$$

or

$$f(x_I) - f(x_R) \le \texttt{mip\_integral\_gap\_rel} \cdot \max(1, |f(x_I)|), \tag{6.20}$$

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).

Note that the default values for `mip_integral_gap_abs` and `mip_integral_gap_rel` are taken from the GAMS options optCA and optCR, but an explicit setting of `mip_integral_gap_abs` or `mip_integral_gap_rel` will override those.

## 7 KNITRO Output

If `outlev=0` then all printing of output is suppressed. The description below assumes the default output level (`outlev=2`) except where indicated:

### Nondefault Options:

This output lists all user options (see section 2) which are different from their default values. If nothing is listed in this section then all user options are set to their default values.

### Problem Characteristics:

The output begins with a description of the problem characteristics.

### Iteration Information - Continuous Problems:

An iteration, in the context of KNITRO, is defined as a step which generates a new solution estimate (i.e., a successful step). The columns of the iteration log are as follows:

`Iter` Iteration number.

`fCount` The cumulative number of function evaluations, only included if (`outlev>3`)

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

`FeasErr` Gives a measure of the feasibility violation at the current iterate.

`OptErr` Gives a measure of the violation of the Karush-Kuhn-Tucker (KKT) (first-order) optimality conditions (not including feasibility) at the current iterate.

`||Step||` The 2-norm length of the step (i.e., the distance between the new iterate and the previous iterate).

`CG its` The number of Projected Conjugate Gradient (CG) iterations required to compute the step.

If `outlev=2`, information is printed every 10 major iterations. If `outlev=3` information is printed at each major iteration. If `outlev>4` addition information is included in the log.

### Iteration Information - Discrete Problems:

By default, the GAMS/KNITRO link prints a log line at every 10'th node. This frequency can be changed via the mip_outinterval option. To turn off the node log completely, set the mip_outlevel option to 0. The columns of the iteration log for discrete models are as follows:

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

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

`Iinf` The current number of active nodes left in the branch and bound tree.

`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).

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

**Termination Message**: At the end of the run a termination message is printed indicating whether or not the optimal solution was found and if not, why the solver terminated. Below is a list of some possible termination messages.

`EXIT: 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.

`EXIT: Iteration limit reached.`

The iteration limit was reached before being able to satisfy the required stopping criteria.

```
EXIT: Convergence to an infeasible point.
      Problem appears to be locally infeasible.
```

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. If this occurs for a variety of initial points, it is likely the problem is infeasible.

`EXIT: Problem appears to be unbounded.`

The objective function appears to be decreasing without bound, while satisfying the constraints.

`EXIT: Current point cannot be improved.`

No more progress can be made. If the current point is feasible it is likely it may be optimal, however the stopping tests cannot be satisfied (perhaps because of degeneracy, ill-conditioning or bad scaling).

```
EXIT: Current point cannot be improved.  Point appears to be
      optimal, but desired accuracy 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.

`EXIT: Time limit reached.`

The time limit was reached before being able to satisfy the required stopping criteria.

`EXIT: 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.

`EXIT: Not enough memory available to solve problem.`

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

**Final Statistics**:

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

**Solution Vector/Constraints**:

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

## 8 Algorithm Options

### 8.1 Automatic

By default, KNITRO will automatically try to choose the best optimizer for the given problem based on the problem characteristics.

### 8.2 Interior/Direct

If the Hessian of the Lagrangian is ill-conditioned or the problem does not have a large-dense Hessian, it may be advisable to compute a step by directly factoring the KKT (primal-dual) matrix rather than using an iterative approach to solve this system. KNITRO offers the Interior/Direct optimizer which allows the algorithm to take direct steps by setting `algorithm=1`. This option will try to take a direct step at each iteration and will only fall back on the iterative step if the direct step is suspected to be of poor quality, or if negative curvature is detected.

Using the Interior/Direct optimizer may result in substantial improvements over Interior/CG when the problem is ill-conditioned (as evidenced by Interior/CG taking a large number of Conjugate Gradient iterations). We encourage the user to try both options as it is difficult to predict in advance which one will be more effective on a given problem. In each case, also experiment with the bar_murule option, as it is difficult to predict which update rule will work best.

**NOTE:** Since the Interior/Direct algorithm in KNITRO requires the explicit storage of a Hessian matrix, this version can only be used with Hessian options, `hessopt=1, 2, 3` or `6`. It may not be used with Hessian options, `hessopt=4` or `5`, which only provide Hessian-vector products. Both the Interior/Direct and Interior/CG methods can be used with the bar_feasible option.

### 8.3 Interior/CG

Since KNITRO was designed with the idea of solving large problems, the Interior/CG optimizer in KNITRO offers an iterative Conjugate Gradient approach to compute the step at each iteration. This approach has proven to be efficient in most cases and allows KNITRO to handle problems with large, dense Hessians, since it does not require factorization of the Hessian matrix. The Interior/CG algorithm can be chosen by setting `algorithm=2`. It can use any of the Hessian options as well as the bar_feasible option.

### 8.4 Active Set

KNITRO includes an active-set Sequential Linear-Quadratic Programing (SLQP) optimizer. This optimizer is particular advantageous 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 the preferred algorithm for detecting infeasible problems quickly. The Active Set algorithm can be chosen by setting `algorithm=3`. It can use any of the Hessian options.

# 9 Other KNITRO special features

This section describes in more detail some of the most important features of KNITRO and 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 version of KNITRO assumes that exact second derivatives of the objective function and constraint functions can be computed. If this is possible and the cost of computing the second derivatives is not overly expensive, it is highly recommended to use exact second derivatives. However, KNITRO also 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 many 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 can be chosen by setting options value `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 can be chosen by setting options value `hessopt=3`.

*Finite-difference Hessian-vector product option*
If the problem is large and gradient evaluations are not the dominate 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 can be chosen by setting options value `hessopt=4`. This option is generally only recommended if the exact gradients are provided.

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

*Exact Hessian-vector products*
In some cases the problem which the user wishes to solve may have a large, dense Hessian which makes it impractical to store or work with the Hessian directly. In this case KNITRO provides an option which does not require that the Hessian itself be computed and stored, but rather only Hessian times vector products are stored. The performance of this option should be nearly identical to the exact Hessian option but requires much less storage. This option can be chosen by setting options value `hessopt=5`.

**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 only stores a limited number of gradient vectors used to approximate the Hessian. In general it requires more iterations to converge than the dense quasi-Newton BFGS approach but will be much more efficient on large-scale problems. This option can be chosen by setting options value `hessopt=6`.

## 9.2 Feasible version

KNITRO offers the user the option of forcing intermediate iterates to stay feasible with respect to the *inequality* constraints (it does not enforce feasibility with respect to the *equality* constraints however). Given an initial point which is *sufficiently* feasible with respect to all inequality constraints and selecting `bar_feasible = 1`, forces all the iterates to strictly satisfy the inequality constraints throughout the solution process. For the feasible mode to become active the iterate $x$ must satisfy

$$cl + tol \leq c(x) \leq cu - tol \tag{9.21}$$

for *all* inequality constraints (i.e., for $cl \neq cu$). The tolerance $tol > 0$ by which an iterate must be strictly feasible for entering the feasible mode is determined by the parameter `bar_feasmodetol` which is `1.0e-4` by default. If the initial point does not satisfy (9.21) then the default infeasible version of KNITRO will run until it obtains a point which is sufficiently feasible with respect to all the inequality constraints. At this point it will switch to the feasible version of KNITRO and all subsequent iterates will be forced to satisfy the inequality constraints.

For a detailed description of the feasible version of KNITRO see [5].

**NOTE:** This option may only be used when `algorithm=2`.

## 9.3 Honor Bounds

By default KNITRO does not enforce that the simple bounds on the variables (1.1$c$) are satisfied throughout the optimization process. Rather, satisfaction of these bounds is only enforced at the solution. In some applications, however, the user may want to enforce that the initial point and all intermediate iterates satisfy the bounds $bl \leq x \leq bu$. This can be enforced by setting `honorbnds=1`.

## 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 option. If this option 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. By default, KNITRO will then print a message indicating that it was unable to improve the solution within the iterations allowed. 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 Solving Systems of Nonlinear Equations

KNITRO is quite 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., constraints with $cl = cu$), and specify the objective function (1.1$a$) as zero (i.e., $f(x) = 0$).

## 9.6 Solving 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

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

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

$$J(x) = \left[ \frac{\partial r_j}{\partial x_i} \right]_{\substack{j = 1,2,\ldots,q \\ i = 1,2,\ldots,n}} .$$

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=1`, and will be very similar to the classical Levenberg-Marquardt method when `algorithm=2`. For a discussion of these methods see, for example, [7].

# KNITRO **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. Technical Report OTC 2002/5, Optimization Technology Center, Northwestern University, Evanston, IL, USA, 2002.

[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] Harwell Subroutine Library. *A catalogue of subroutines (HSL 2002).* AEA Technology, Harwell, Oxfordshire, England, 2002.

[7] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer Series in Operations Research. Springer, 1999.

[8] 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. Technical Report 2003-6, Optimization Technology Center, Northwestern University, Evanston, IL, USA, June 2003. To appear in Mathematical Programming, Series A.