Package 'KnitroR'

July 20, 2017

Type Package

Title R Interface for the nonlinear optimization solver Artelys Knitro

Version 1.2.0

Date 2016-09-11

Author Jean-Hubert Hours <jean-hubert.hours@artelys.com>

Maintainer Artelys S.A. <support-knitro@artelys.com>

URL https://www.artelys.com/en/optimization-tools/knitro

Description Provides an R interface to the nonlinear optimization solver Artelys Knitro.
This interface passes user-defined R functions on to Knitro C interface.
To use this package you need to own a valid license of Artelys Knitro.

R topics documented:

`knitro` *Nonlinear optimization*

Description

The `R` function `knitro` is part of the `KnitroR` package, which is an `R` interface to the nonlinear solver Knitro (Nonlinear Interior-point Trust Region Optimization) developped and distributed by Artelys SA.

The `R` function `knitro` is designed to compute local solutions of general nonlinear programs of the form:

$$\begin{array}{lccccc} \min_x & f(x) & & & & \\ \text{s.t.} & c_L & \leq & g(x) & \leq & c_U \\ & x_L & \leq 1 & x & \leq & x_U \end{array}$$

given x a vector of variables in $\mathbb{R}^n$, via the algorithms available in Knitro.

The function $f : \mathbb{R}^n \to \mathbb{R}$ is the objective.
The function $g : \mathbb{R}^n \to \mathbb{R}^m$ corresponds to the constraints.
The vectors c_L and c_U in $\mathbb{R}^m$ are bounds on the constraints function.

Note that equality constraints can be enforced by setting appropriate components of c_L and c_U to the same value. The vectors x_L and x_U are bounds in $\mathbb{R}^n$.

The API also allows to specify complementarity constraints in order to use the MPEC algorithm from Knitro. See arguments `numCompConstraints, ccIdxList1, ccIdxList2`.

All `Knitro` options are supported, as well as additional options specific to the `R` interface.
List of the main supported options:

- `environ` for the user to specify an `R` environment.
- `outlev` for changing the output level of Knitro, see Knitro documentation.
- `ms_enable` for setting the multi-start option.
- `objtype` for changing the type of the objective (linear, quadratic or general), see Knitro documentation.
- `objgoal` for changing the type of the objective (minimize or maximize), see Knitro documentation.
- `contype` for changing the type of constraints, see Knitro documentation.
- `hessopt` for setting exact hessian (user-defined), quasi-Newton hessian or exact hessian, see Knitro documentation.
- `gradopt` for setting exact gradient (1, user-defined), forward finite-difference (2) or central finite-difference (3), see Knitro documentation for details.
- `derivcheck` for setting the derivative checker of Knitro.

The user can also set all Knitro options via a file with extension `.opt` to be specified in the argument `optionsFile`.

Usage

```
knitro(nvar = numeric(0),
   ncon = numeric(0),
   nnzJ = numeric(0),
   nnzH = numeric(0),
   x0 = numeric(0),
   objective,
   gradient = NULL,
   constraints = NULL,
   jacobian = NULL,
   jacIndexCons = numeric(0),
   jacIndexVars = numeric(0),
   jacBitMap = numeric(0),
   hessianLag = NULL,
   hessIndexRows = numeric(0),
   hessIndexCols = numeric(0),
   hessBitMap = numeric(0),
```

```
  xL = numeric(0),
  xU = numeric(0),
  cL = numeric(0),
  cU = numeric(0),
  numCompConstraints = numeric(0),
  ccIdxList1 = NULL,
  ccIdxList2 = NULL,
  xScaleFactors = numeric(0),
  xScaleCenters = numeric(0),
  cScaleFactors = numeric(0),
  ccScaleFactors = numeric(0),
  objScaleFactor = numeric(0),
  honorBnds = numeric(0),
  constraintsTypes = numeric(0), # For retro-compatibility with 10.2.1
  constraintTypes = numeric(0),
  options = list(),
  optionsFile = NULL)
```

Arguments

nvar	Number of variables, or dimension of x.
ncon	Number of constraints, or dimension of g.
nnzJ	Number of nonzero elements in the jacobian of the nonlinear constraints function g.
nnzH	Number of nonzero elements in the hessian of the lagrangian.
x0	Initial guess on primal variable.
objective	Objective function given as an R function.
gradient	Gradient of objective given as an R function.
constraints	Nonlinear inequality constraints given as an R function.
jacobian	Jacobian of inequality constraints given as an R function returning vector of nonzero elements (c(...)).
jacIndexCons	Constraints indices of nonzero elements given as an R vector (c(...)).
jacIndexVars	Variables indices of nonzero elements given as an R vector (c(...)).
jacBitMap	Sparsity pattern of jacobian given as bitmap in an R vector (c(...)) in row major : 1 if element is nonzero, 0 otherwise.
hessianLag	Hessian of Lagrangian given as an R function returning an R vector of nonzero elements (c(...)).
hessIndexRows	Row indices of nonzero elements of hessian of Lagrangian given as an R vector (c(...)).
hessIndexCols	Columns indices of nonzero elements of hessian of Lagrangian given as an R vector (c(...)).
hessBitMap	Sparsity pattern of hessian of Lagrangian as bitmap in an R vector (c(...)) in row major : 1 if element is nonzero, 0 otherwise.
xL	Lower bounds on variables given as an R vector (c(...)).

xU	Upper bounds on variables given as an R vector (c(...)).
cL	Lower bounds on nonlinear constraints function given as an R vector (c(...)).
cU	Upper bounds on nonlinear constraints function given as an R vector (c(...)).
numCompConstraints	Number of complementarity constraints among nonlinear constraints.
ccIdxList1	Indices of first variables in complementarity constraints.
ccIdxList2	Indices of second variables in complementarity constraints.
xScaleFactors	Vector of scaling factors to be applied on the fitting parameter.
xScaleCenters	Vector of scale centers by which the fitting parameter needs to be shifted when performing variables scaling.
cScaleFactors	Vector of scaling factors to be applied to the nonlinear constraints functional g.
ccScaleFactors	Vector of scaling factors to be applied to to the complementarity constraints functional.
objScaleFactor	Scaling factor by which the objective function is to be multiplied.
honorBnds	Vector of indices of variables for which bounds satisfaction needs to be enforced.
constraintsTypes	Vector of constraint types : general (0), linear (1), quadratic (2). Used for retro-compatibility purpose.
constraintTypes	Vector of constraint types : general (0), linear (1), quadratic (2).
options	List of options given as an R list.
optionsFile	Option file ([file].opt) listing knitro options.

Examples

```
##### Simple optimization of a quadratic function #####
knitro(objective=function(x) x[1]*x[2], xL=c(-5,-5), xU=c(10,10))


##### Optimizing Rosenbrock function #####

# Objective function
eval_f <- function(x) {
        return( 100 * (x[2] - x[1] * x[1])^2 + (1 - x[1])^2 )
}

# Initial guess
x0 <- c( -1.2, 1 )

# Optimizing with finite differences
sol <- knitro(x0 = x0, objective = eval_f)

# Providing exact gradient
eval_grad_f <- function(x) {
    grad_f <-rep(0, length(x))
```

```
    grad_f[1] <- 2*x[1]-2+400*x[1]^3-400*x[1]*x[2]
        grad_f[2] <- 200*(x[2]-x[1]^2)

        return( grad_f )
}

# Optimizing with exact gradient gradient
sol <- knitro(x0 = x0, objective = eval_f, gradient = eval_grad_f)
```

---

knitrolsq *Vectorial nonlinear least squares optimization*

Description

The R function knitrolsq is part of the KnitroR package, which is an R interface to the nonlinear solver Knitro (Nonlinear Interior-point Trust Region Optimization) developped and distributed by Artelys SA.

The function knitrolsq is designed to interface nonlinear least-squares problems with the nonlinear solver Knitro. These problems are of the form:

$$\begin{array}{lccc} \min_{par} & \frac{1}{2} & \|F(X, par) - Y\|_2^2 & \\ \text{s.t.} & par_L & \leq \quad par \quad \leq & par_U \end{array}$$

where par is a fitting parameter, X and Y are two vectors of sample points, F is a nonlinear model to be fitted and par_L and par_U are bounds on the fitting parameter par.

This API is very close to the knitro API.

The user can also set all basic Knitro options via a file with extension .opt to be specified in the variable optionsFile in the R function knitro.

Usage

```
knitrolsq(dimp = numeric(0),
          par0 = numeric(0),
          dataFrameX,
          dataFrameY,
          residual,
          jacobian = NULL,
          parL = numeric(0),
          parU = numeric(0),
          xScaleFactors = numeric(0),
          xScaleCenters = numeric(0),
          objScaleFactor = numeric(0),
          jacIndexRows = numeric(0),
          jacIndexCols = numeric(0),
          options = list(),
          optionsFile = NULL)
```

Arguments

dimp	Dimension of the fitting parameter.
par0	Initial guess for the fitting parameter.
dataFrameX	Data frame containing the x samples.
dataFrameY	Data frame containing the y samples.
residual	A function implementing the nonlinear model to be fitted, in vectorial form.
jacobian	Jacobian of nonlinear vectorial model, provided by user.
parL	Lower bound on fitting parameter.
parU	Upper bound on fitting parameter.
xScaleFactors	Vector of scaling factors to be applied on the fitting parameter.
xScaleCenters	Vector of scale centers by which the fitting parameter needs to be shifted when performing variables scaling.
objScaleFactor	Scaling factor by which the objective function is to be multiplied when performing objective scaling.
jacIndexRows	Row (or residual) indices of nonzero elements in jacobian of residuals function.
jacIndexCols	Column (or sample) indices of nonzero elements in jacobian of residuals function.
options	List of options.
optionsFile	Option file ([file].opt) listing knitro options.

Examples

```
# Residuals function to be used in knitrolsq, which returns a vector of residuals
# computed as the difference between the nonlinear fitting model and the target value.
nlres <- function(x, dfX, dfY) {
    dMatX <- data.matrix(dfX)
    dVecX <- as.vector(t(dMatX))

    dMatY <- data.matrix(dfY)
    dVecY <- as.vector(t(dMatY))

    return( c( x[1] * dVecX[1]^(x[2]) - dVecY[1],
               x[1] * dVecX[2]^(x[2]) - dVecY[2],
               x[1] * dVecX[3]^(x[2]) - dVecY[3],
               x[1] * dVecX[4]^(x[2]) - dVecY[4],
               x[1] * dVecX[5]^(x[2]) - dVecY[5],
               x[1] * dVecX[6]^(x[2]) - dVecY[6] ) )
}

# x samples as a data frame
x <- c(1.309, 1.471, 1.49, 1.565, 1.611, 1.68)
dfx <- data.frame(x)
# y samples, or target values as a data frame
y <- c(2.138, 3.421, 3.597, 4.34, 4.882, 5.66)
dfy <- data.frame(y)
```

```
# Call knitrolsq from the initial guess p1=1, p2=1
ktrSolLsq <- knitrolsq(dimp = 2,
                       par0 = c(1, 1),
                       dataFrameX = dfx,
                       dataFrameY = dfy,
                       residual = nlres)


##### Using exact derivatives #####

# Jacobian of the residuals function R(p) = F(X, p) -Y
nljac <- function(x, dfX, dfY) {
    dMatX <- data.matrix(dfX)
    dVecX <- as.vector(t(dMatX))

    dMatY <- data.matrix(dfY)
    dVecY <- as.vector(t(dMatY))

    return( c( dVecX[1]^(x[2]), x[1] * log(dVecX[1]) * dVecX[1]^(x[2]),
               dVecX[2]^(x[2]), x[1] * log(dVecX[2]) * dVecX[2]^(x[2]),
               dVecX[3]^(x[2]), x[1] * log(dVecX[3]) * dVecX[3]^(x[2]),
               dVecX[4]^(x[2]), x[1] * log(dVecX[4]) * dVecX[4]^(x[2]),
               dVecX[5]^(x[2]), x[1] * log(dVecX[5]) * dVecX[5]^(x[2]),
               dVecX[6]^(x[2]), x[1] * log(dVecX[6]) * dVecX[6]^(x[2]) ) )
}

# Call knitrolsq using exact (dense) jacobian
ktrSolLsq <- knitrolsq(dimp = 2,
                       par0 = c(1, 1),
                       dataFrameX = dfx,
                       dataFrameY = dfy,
                       residual = nlres,
                       jacobian = nljac)
```

---

knitromip *Mixed-integer nonlinear optimization*

Description

The R function knitromip is part of the KnitroR package, which is an R interface to the nonlinear solver Knitro (Nonlinear Interior-point Trust Region Optimization) developped and distributed by Artelys SA.

The function knitromip is designed to solve mixed-integer nonlinear programs, that is NLPs in which part of the variables are integers.

The API of this function is very close to the API of the function knitro. The only difference is the 3 arguments xType, cFnType and objfnType. Some of the available options (specific to MIPs) are:

- xPriorities: branching priorities for integer variables. Array xPriorities has length equal to the number of binary variables.

- `mipBranchRule`: specifies branching rule to use in branch and bound algorithm. Possible values are:
  - 0 (auto) Knitro chooses the branching rule
  - 1 (most_frac) Use fractional branching
  - 2 (pseudcost) Pseudo-cost branching
  - 3 (strong) Strong branching
- `mipHeuristic`: specifies MIP heuristic search.
  - 0 (auto) Knitro chooses the heuristic
  - 1 (none) No heuristic search
  - 2 (feaspump) Apply feasibility pump heuristic
  - 3 (mpec) Apply heuristic based on MPEC formulation
- `mipMethod`: specifies MIP method.
  - 0 (auto) Knitro chooses the method
  - 1 (BB) Use the standard Branch and Bound method
  - 2 (HQG) Use the hybrid Quesada-Grossman method (for convex, nonlinear problems only)
  - 3 (MISQP) Use mixed-integer SQP method (allows for non-relaxable integer variables)
- `mipLPalg`: 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 = HQG. (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.

The user can specify Knitro options and `R` environment in the options argument. See `knitro` function documentation for more details and examples about available options.

Knitro options can also be set via a file with extension `.opt` to be specified in the argument `optionsFile`.

Usage

```
knitromip(nvar = numeric(0),
          ncon = numeric(0),
  nnzJ = numeric(0),
  nnzH = numeric(0),
          x0 = numeric(0),
          objective,
          gradient = NULL,
          constraints = NULL,
          jacobian = NULL,
```

```
        jacIndexCons = numeric(0),
        jacIndexVars = numeric(0),
        jacBitMap = numeric(0),
  hessianLag = NULL,
        hessIndexRows = numeric(0),
        hessIndexCols = numeric(0),
  hessBitMap = numeric(0),
        xL = numeric(0),
        xU = numeric(0),
        cL = numeric(0),
        cU = numeric(0),
        xType = numeric(0),
        cFnType = numeric(0),
        objfntype = numeric(0),
        xScaleFactors = numeric(0),
        xScaleCenters = numeric(0),
        cScaleFactors = numeric(0),
        ccScaleFactors = numeric(0),
        objScaleFactor = numeric(0),
        xIndex = numeric(0),
        xStrategy = numeric(0),
  constraintsTypes = numeric(0), # For retro-compatibility with 10.2.1
  constraintTypes = numeric(0),
        options = list(),
        optionsFile = NULL)
```

Arguments

nvar	Number of variables, or dimension of x.
ncon	Number of constraints, or dimension of g.
nnzJ	Number of nonzero elements in the jacobian of the nonlinear constraints function g.
nnzH	Number of nonzero elements in the hessian of the lagrangian.
x0	Initial guess on primal variable.
objective	Objective function given as an R function.
gradient	Gradient of objective given as an R function.
constraints	Nonlinear inequality constraints given as an R function.
jacobian	Jacobian of inequality constraints given as an R function returning vector of nonzero elements (c(...)).
jacIndexCons	Constraints indices of nonzero elements given as an R vector (c(...)).
jacIndexVars	Variables indices of nonzero elements given as an R vector (c(...)).
jacBitMap	Sparsity pattern of jacobian given as bitmap in an R vector (c(...)) in row major : 1 if element is nonzero, 0 otherwise.
hessianLag	Hessian of Lagrangian given as an R function returning an R vector of nonzero elements (c(...)).

hessIndexRows	Row indices of nonzero elements of hessian of Lagrangian given as an R vector (c(...)).
hessIndexCols	Columns indices of nonzero elements of hessian of Lagrangian given as an R vector (c(...)).
hessBitMap	Sparsity pattern of hessian of Lagrangian as bitmap in an R vector (c(...)) in row major : 1 if element is nonzero, 0 otherwise.
xL	Lower bounds on variables given as an R vector (c(...)).
xU	Upper bounds on variables given as an R vector (c(...)).
cL	Lower bounds on nonlinear constraints function given as an R vector (c(...)).
cU	Upper bounds on nonlinear constraints function given as an R vector (c(...)).
xScaleFactors	Vector of scaling factors to be applied on the fitting parameter.
xScaleCenters	Vector of scale centers by which the fitting parameter needs to be shifted when performing variables scaling.
cScaleFactors	Vector of scaling factors to be applied to the nonlinear constraints functional g.
ccScaleFactors	Vector of scaling factors to be applied to to the complementarity constraints functional.
objScaleFactor	Scaling factor by which the objective function is to be multiplied.
xType	Defines the variable type and must be of the same length as x0, it it is used. Continuous variables are denoted by 0, integer variables by 1 and binary variables by 2.
cFnType	Defines the type of the constraints and must be of the same length as the number of inequality constraints. Convex, nonconvex and uncertain constraints are denoted by 0, 1 or 2 respectively.
objfntype	Defines the type of the objective function. Convex, nonconvex and uncertain constraints is denoted by 0, 1 or 2 respectively.
xIndex	Vector of indices of integer variables for which a specialized strategy is to be applied.
xStrategy	Vector of integers corresponding to strategies to be applied on each integer variable (KTR_MIP_INTVAR_STRATEGY_NONE, KTR_MIP_INTVAR_STRATEGY_RELAX or KTR_MIP_INTVAR_STRATEGY_MPEC).
constraintsTypes	Vector of constraint types : general (0), linear (1), quadratic (2). Used for retrocompatibility purpose.
constraintTypes	Vector of constraint types : general (0), linear (1), quadratic (2).
options	List of options.
optionsFile	Option file ([file].opt) listing knitro options.

Examples

```
# Objective callback
eval_f <- function(x){
   return ( 5*x[4]+6*x[5]+8*x[6]+10*x[1]-7*x[3]-18*log(x[2]+1)-19.2*log(x[1]-x[2]+1)+10 )
```

```
}

# Objective gradient callback
eval_grad_f <- function(x){
    return ( c(10.0-(19.2/(x[1]-x[2]+1.0)),
                (-18.0/(x[2]+1.0))+(19.2/(x[1]-x[2]+1.0)),
                -7.0,
                5.0,
                6.0,
                8.0) )
}

# Constraints callback
eval_g <- function(x){
    return ( c(0.8*log(x[2]+1.0)+0.96*log(x[1]-x[2]+1.0)-0.8*x[3],
                log(x[2]+1.0)+1.2*log(x[1]-x[2]+1.0)-x[3]-2.0*x[6],
                x[2]-x[1],
                x[2]-2.0*x[4],
                x[1]-x[2]-2.0*x[5],
                x[4]+x[5]) )
}

# Constraints jacobian callback
eval_jac_g <- function(x){
    tmp1 <- x[1]-x[2]+1.0
    tmp2 <- x[2]+1.0
    return ( c(0.96/tmp1, (-0.96/tmp1)+(0.8/tmp2), -0.8,
                1.2/tmp1, (-1.2/tmp1)+(1.0/tmp2),-1.0, -2.0,
                -1.0, 1.0,
                1.0, -2.0,
                1.0, -1.0, -2.0,
                1.0, 1.0) )
}

# Sparsity pattern of jacobian
jIndC <- c(rep(0,3), rep(1,4), rep(2,2), rep(3,2), rep(4,3), rep(5,2))
jIndV <- c(0,1,2,0,1,2,5,0,1,1,3,0,1,4,3,4)

# Bounds on variables
xL <- c( rep(0,3), rep(0, 3) )
xU <- c( 2, 2, 1, rep(2, 3) )

# Bounds on constraints
cL <- c( 0, -2, rep(-1e20, 4) )
cU <- c( rep(1e20, 2), rep(0,3), 1 )

# Knitro's MINLP parameters
xType <- c(rep(0,3), rep(1,3))
xPriorities <- c(rep(0,6))
cFnType <- c(rep(2,2), rep(1,4))
objfntype <- 2

# Set knitro options
```

```
knitro_opts <- list("derivcheck" = 1,
                    "hessopt" = 2)

# Call Knitro MINLP solver
mip_sol <- knitromip(objective = eval_f,
                     gradient = eval_grad_f,
                     constraints = eval_g,
                     jacobian = eval_jac_g,
                     jacIndexCons = jIndC, jacIndexVars = jIndV,
                     xL = xL, xU = xU,
                     cL = cL, cU = cU,
                     xType = xType,
                     cFnType = cFnType,
                     objfntype = objfntype,
                     options = knitro_opts)

# Solve using MISQP
knitro_opts <- list("hessopt" = 2, "mipMethod" = 3)

mip_sol <- knitromip(objective = eval_f,
                     gradient = eval_grad_f,
                     constraints = eval_g,
                     jacobian = eval_jac_g,
                     jacIndexCons = jIndC, jacIndexVars = jIndV,
                     xL = xL, xU = xU,
                     cL = cL, cU = cU,
                     xType = xType,
                     cFnType = cFnType,
                     objfntype = objfntype,
                     options = knitro_opts)
```

Index