COBYQA User Guide

Release 1.0.dev0

Tom M. Ragonneau

October 15, 2021

CONTENTS

Release 1.0.dev0

Date October 15, 2021

This reference manual details functions, modules, and objects included in COBYQA. For a complete description of the software, see the general documentation.

CHAPTER

ONE

OPTIMIZATION FRAMEWORK

minimize(fun, x0[, args, xl, xu, Aub, bub, ...])	Minimize a real-valued function.
OptimizeResult	Structure for the result of an optimization algorithm.
optimize.TrustRegion(fun, x0[, args, xl, ...])	Framework atomization of the derivative-free trust-region SQP method.
optimize.Models(fun, x0, xl, xu, Aub, bub, ...)	Model a nonlinear optimization problem.
optimize.Quadratic(bmat, zmat, idz, fval)	Representation of a quadratic multivariate function.

1.1 cobyqa.minimize

`cobyqa.`**`minimize`**(*fun*, *x0*, *args=()*, *xl=None*, *xu=None*, *Aub=None*, *bub=None*, *Aeq=None*, *beq=None*, *cub=None*, *ceq=None*, *options=None*, ***kwargs*)

Minimize a real-valued function.

The minimization can be subject to bound, linear inequality, linear equality, nonlinear inequality, and nonlinear equality constraints using a derivative-free trust-region SQP method. Although the solver may tackle infeasible points (including the initial guess), the bounds constraints (if any) are always respected.

Parameters

fun [callable] Objective function to be minimized.

```
fun(x, *args) -> float
```

where `x` is an array with shape (n,) and *args* is a tuple of parameters to forward to the objective function.

x0 [array_like, shape (n,)] Initial guess.

args [tuple, optional] Parameters to forward to the objective, the nonlinear inequality constraint, and the nonlinear equality constraint functions.

xl [array_like, shape (n,), optional] Lower-bound constraints on the decision variables. Use `-numpy.inf` to disable the bounds on some variables.

xu [array_like, shape (n,), optional] Upper-bound constraints on the decision variables. Use `numpy.inf` to disable the bounds on some variables.

Aub [array_like, shape (mlub, n), optional] Jacobian matrix of the linear inequality constraints. Each row of *Aub* stores the gradient of a linear inequality constraint.

bub [array_like, shape (mlub,), optional] Right-hand side vector of the linear inequality constraints `Aub @ x <= bub`, where `x` has the same size than *x0*.

Aeq [array_like, shape (mleq, n), optional] Jacobian matrix of the linear equality constraints. Each row of *Aeq* stores the gradient of a linear equality constraint.

beq [array_like, shape (mleq,), optional] Right-hand side vector of the linear equality constraints *Aeq @ x = beq*, where `x` has the same size than *x0*.

cub [callable] Nonlinear inequality constraint function `ceq(x, *args) <= 0`.

```
cub(x, *args) -> array_like, shape (mnlub,)
```

where `x` is an array with shape (n,) and *args* is a tuple of parameters to forward to the constraint function.

ceq [callable] Nonlinear equality constraint function `ceq(x, *args) = 0`.

```
ceq(x, *args) -> array_like, shape (mnleq,)
```

where `x` is an array with shape (n,) and *args* is a tuple of parameters to forward to the constraint function.

options [dict, optional] Options to forward to the solver. Accepted options are:

rhobeg [float, optional] Initial trust-region radius (the default is 1).

rhoend [float, optional] Final trust-region radius (the default is 1e-6).

npt [int, optional] Number of interpolation points for the objective and constraint models (the default is `2 * n + 1`).

maxfev [int, optional] Upper bound on the number of objective and constraint function evaluations (the default is `500 * n`).

target [float, optional] Target value on the objective function (the default is `-numpy.inf`). If the solver encounters a feasible point at which the objective function evaluations is below the target value, then the computations are stopped.

disp [bool, optional] Whether to print pieces of information on the execution of the solver (the default is False).

debug [bool, optional] Whether to make debugging tests during the execution, which is not recommended in production (the default is False).

Returns

OptimizeResult Result of the optimization solver. Important attributes are: `x` the solution point, `success` a flag indicating whether the optimization terminated successfully, and `message` a description of the termination status of the optimization. See *`OptimizeResult`* for a description of other attributes.

Other Parameters

bdtol [float, optional] Tolerance for comparisons on the bound constraints (the default is `10 * eps * n * max(1, max(abs(xl)), max(abs(xu)))`, where the values of `xl` and `xu` evolve to include the shift of the origin).

lctol [float, optional] Tolerance for comparisons on the linear constraints (the default is `10 * eps * n * max(1, max(abs(bub)))`, where the values of `bub` evolve to include the shift of the origin).

lstol [float, optional] Tolerance on the approximate KKT conditions for the calculations of the least-squares Lagrange multipliers (the default is `10 * eps * max(n, m) * max(1, max(abs(g)))`, where `g` is the gradient of the current model of the objective function).

1.2 cobyqa.OptimizeResult

class cobyqa.**OptimizeResult**

Structure for the result of an optimization algorithm.

Attributes

x [numpy.ndarray, shape (n,)] Solution point provided by the optimization solver.

success [bool] Flag indicating whether the optimization solver terminated successfully.

status [int] Termination status of the optimization solver.

message [str] Description of the termination status of the optimization solver.

fun [float] Value of the objective function at the solution point provided by the optimization solver.

jac [numpy.ndarray, shape (n,)] Approximation of the gradient of the objective function at the solution point provided by the optimization solver, based on undetermined interpolation.

nfev [int] Number of objective and constraint function evaluations.

nit [int] Number of iterations performed by the optimization solver.

maxcv [float] Maximum constraint violation at the solution point provided by the optimization solver. It is set only if the problem is not declared unconstrained by the optimization solver.

Methods

`clear`()	
`copy`()	
`fromkeys`(iterable[, value])	Create a new dictionary with keys from iterable and values set to value.
`get`(key[, default])	Return the value for key if key is in the dictionary, else default.
`items`()	
`keys`()	
`pop`(key[, default])	If key is not found, default is returned if given, otherwise KeyError is raised
`popitem`(/)	Remove and return a (key, value) pair as a 2-tuple.
`setdefault`(key[, default])	Insert key with a value of default if key is not in the dictionary.
`update`([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
`values`()	

1.2.1 cobyqa.OptimizeResult.clear

`OptimizeResult.clear()` → None. Remove all items from D.

1.2.2 cobyqa.OptimizeResult.copy

`OptimizeResult.copy()` → a shallow copy of D

1.2.3 cobyqa.OptimizeResult.fromkeys

`OptimizeResult.fromkeys(iterable, value=None, /)`

Create a new dictionary with keys from iterable and values set to value.

1.2.4 cobyqa.OptimizeResult.get

`OptimizeResult.get(key, default=None, /)`

Return the value for key if key is in the dictionary, else default.

1.2.5 cobyqa.OptimizeResult.items

`OptimizeResult.items()` → a set-like object providing a view on D's items

1.2.6 cobyqa.OptimizeResult.keys

`OptimizeResult.keys()` → a set-like object providing a view on D's keys

1.2.7 cobyqa.OptimizeResult.pop

`OptimizeResult.pop(key, default=<unrepresentable>, /)`

If key is not found, default is returned if given, otherwise KeyError is raised

1.2.8 cobyqa.OptimizeResult.popitem

`OptimizeResult.popitem(/)`

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

1.2.9 cobyqa.OptimizeResult.setdefault

`OptimizeResult.setdefault(key, default=None, /)`

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

1.2.10 cobyqa.OptimizeResult.update

`OptimizeResult.update([E], **F)` → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

1.2.11 cobyqa.OptimizeResult.values

`OptimizeResult.values()` → an object providing a view on D's values

1.3 cobyqa.optimize.TrustRegion

`class cobyqa.optimize.TrustRegion(fun, x0, args=(), xl=None, xu=None, Aub=None, bub=None, Aeq=None, beq=None, cub=None, ceq=None, options=None, **kwargs)`

Framework atomization of the derivative-free trust-region SQP method.

Attributes

aeq Jacobian matrix of the linear equality constraints.

aub Jacobian matrix of the linear inequality constraints.

beq Right-hand side vector of the linear equality constraints.

bub Right-hand side vector of the linear inequality constraints.

copteq Evaluation of the nonlinear equality constraint function of the nonlinear optimization problem at `xopt`.

coptub Evaluation of the nonlinear inequality constraint function of the nonlinear optimization problem at `xopt`.

cvaleq Evaluations of the nonlinear equality constraint function of the nonlinear optimization problem at the interpolation points.

cvalub Evaluations of the nonlinear inequality constraint function of the nonlinear optimization problem at the interpolation points.

fopt Evaluation of the objective function of the nonlinear optimization problem at `xopt`.

fval Evaluations of the objective function of the nonlinear optimization problem at the interpolation points.

is_model_step Flag indicating whether the current step is a model step.

knew Index of the interpolation point to be removed from the interpolation set.

kopt Index of the best interpolation point so far, corresponding to the point around which the Taylor expansions of the quadratic models are defined.

lmleq Lagrange multipliers associated with the linear equality constraints.

lmlub Lagrange multipliers associated with the linear inequality constraints.

lmnleq Lagrange multipliers associated with the quadratic models of the nonlinear equality constraints.

lmnlub Lagrange multipliers associated with the quadratic models of the nonlinear inequality constraints.

maxcv Constraint violation evaluated on the nonlinear optimization problem at t‘xopt‘.

mleq Number of the linear equality constraints.

mlub Number of the linear inequality constraints.

mnleq Number of the nonlinear equality constraints.

mnlub Number of the nonlinear inequality constraints.

options Options forwarded to the solver.

peneq Penalty coefficient associated with the equality constraints.

penub Penalty coefficient associated with the inequality constraints.

rval Residuals associated with the constraints of the nonlinear optimization problem at the interpolation points.

type Type of the nonlinear optimization problem.

xbase Shift of the origin in the calculations.

xl Lower-bound constraints on the decision variables.

xopt Best interpolation point so far, corresponding to the point around which the Taylor expansion of the quadratic models are defined.

xpt Displacements of the interpolation points from the origin.

xu Upper-bound constraints on the decision variables.

Methods

`__call__`(x, fx, cubx, ceqx[, model])	Evaluate the merit function.
`ceq`(x)	Evaluate the nonlinear equality constraint function of the nonlinear optimization problem.
`check_models`([stack_level])	Check the interpolation conditions.
`check_options`(n[, stack_level])	Ensure that the options are consistent, and modify them if necessary.
`cub`(x)	Evaluate the nonlinear inequality constraint function of the nonlinear optimization problem.
`fun`(x)	Evaluate the objective function of the nonlinear optimization problem.
`get_best_point`()	Get the index of the optimal interpolation point.
`less_merit`(mval1, rval1, mval2, rval2)	Indicates whether a point is better than another.
`model_ceq`(x, i)	Evaluate an equality constraint function of the model.

continues on next page

Table 3 – continued from previous page

`model_ceq_alt`(x, i)	Evaluate an alternative equality constraint function of the model.
`model_ceq_alt_curv`(x, i)	Evaluate the curvature of an alternative equality constraint function of the model.
`model_ceq_alt_grad`(x, i)	Evaluate the gradient of an alternative equality constraint function of the model.
`model_ceq_alt_hess`(i)	Evaluate the Hessian matrix of an alternative equality constraint function of the model.
`model_ceq_alt_hessp`(x, i)	Evaluate the product of the Hessian matrix of an alternative equality constraint function of the model with any vector.
`model_ceq_curv`(x, i)	Evaluate the curvature of an equality constraint function of the model.
`model_ceq_grad`(x, i)	Evaluate the gradient of an equality constraint function of the model.
`model_ceq_hess`(i)	Evaluate the Hessian matrix of an equality constraint function of the model.
`model_ceq_hessp`(x, i)	Evaluate the product of the Hessian matrix of an equality constraint function of the model with any vector.
`model_cub`(x, i)	Evaluate an inequality constraint function of the model.
`model_cub_alt`(x, i)	Evaluate an alternative inequality constraint function of the model.
`model_cub_alt_curv`(x, i)	Evaluate the curvature of an alternative inequality constraint function of the model.
`model_cub_alt_grad`(x, i)	Evaluate the gradient of an alternative inequality constraint function of the model.
`model_cub_alt_hess`(i)	Evaluate the Hessian matrix of an alternative inequality constraint function of the model.
`model_cub_alt_hessp`(x, i)	Evaluate the product of the Hessian matrix of an alternative inequality constraint function of the model with any vector.
`model_cub_curv`(x, i)	Evaluate the curvature of an inequality constraint function of the model.
`model_cub_grad`(x, i)	Evaluate the gradient of an inequality constraint function of the model.
`model_cub_hess`(i)	Evaluate the Hessian matrix of an inequality constraint function of the model.
`model_cub_hessp`(x, i)	Evaluate the product of the Hessian matrix of an inequality constraint function of the model with any vector.
`model_lag`(x)	Evaluate the Lagrangian function of the model.
`model_lag_alt`(x)	Evaluate the alternative Lagrangian function of the model.
`model_lag_alt_curv`(x)	Evaluate the curvature of the alternative Lagrangian function of the model.
`model_lag_alt_grad`(x)	Evaluate the gradient of the alternative Lagrangian function of the model.
`model_lag_alt_hess`()	Evaluate the Hessian matrix of the alternative Lagrangian function of the model.

continues on next page

Table 3 – continued from previous page

model_lag_alt_hessp(x)	Evaluate the product of the Hessian matrix of the alternative Lagrangian function of the model with any vector.
model_lag_curv(x)	Evaluate the curvature of the Lagrangian function of the model.
model_lag_grad(x)	Evaluate the gradient of Lagrangian function of the model.
model_lag_hess()	Evaluate the Hessian matrix of the Lagrangian function of the model.
model_lag_hessp(x)	Evaluate the product of the Hessian matrix of the Lagrangian function of the model with any vector.
model_obj(x)	Evaluate the objective function of the model.
model_obj_alt(x)	Evaluate the alternative objective function of the model.
model_obj_alt_curv(x)	Evaluate the curvature of the alternative objective function of the model.
model_obj_alt_grad(x)	Evaluate the gradient of the alternative objective function of the model.
model_obj_alt_hess()	Evaluate the Hessian matrix of the alternative objective function of the model.
model_obj_alt_hessp(x)	Evaluate the product of the Hessian matrix of the alternative objective function of the model with any vector.
model_obj_curv(x)	Evaluate the curvature of the objective function of the model.
model_obj_grad(x)	Evaluate the gradient of the objective function of the model.
model_obj_hess()	Evaluate the Hessian matrix of the objective function of the model.
model_obj_hessp(x)	Evaluate the product of the Hessian matrix of the objective function of the model with any vector.
model_step(delta, **kwargs)	Estimate a model-improvement step from `xopt`.
prepare_model_step(delta)	Set the next iteration to a model-step if necessary.
prepare_trust_region_step()	Set the next iteration to a trust-region step.
reduce_penalty_coefficients()	Reduce the penalty coefficients if possible, to prevent overflows.
reset_models()	Reset the models.
set_default_options(n)	Set the default options for the solvers.
shift_origin(delta)	Shift the origin of the calculations if necessary.
trust_region_step(delta, **kwargs)	Evaluate a Byrd-Omojokun-like trust-region step from `xopt`.
update(step, **kwargs)	Include a new point in the interpolation set.
update_multipliers(**kwargs)	Set the least-squares Lagrange multipliers.
update_penalty_coefficients(step, fx, cubx, ceqx)	Increase the penalty coefficients.

1.3.1 cobyqa.optimize.TrustRegion.__call__

`TrustRegion.__call__`(*x*, *fx*, *cubx*, *ceqx*, *model=False*)

Evaluate the merit function.

The merit function is an augmented Lagrangian.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the merit function is to be evaluated.

fx [float] Value of the objective function at *x*.

cubx [numpy.ndarray, shape (mnlub,)] Value of the nonlinear inequality constraint function at *x*.

ceqx [numpy.ndarray, shape (mnleq,)] Value of the nonlinear equality constraint function at *x*.

model [bool, optional] Whether to also evaluate the merit function on the different models (the default is False).

Returns

float or (float, float) Value of the merit function at *x*, evaluated on the nonlinear optimization problem. If `model = True`, the merit function evaluated on the different models is also returned.

1.3.2 cobyqa.optimize.TrustRegion.ceq

`TrustRegion.ceq`(*x*)

Evaluate the nonlinear equality constraint function of the nonlinear optimization problem.

Parameters

x [array_like, shape (n,)] Point at which the constraint function is to be evaluated.

Returns

numpy.ndarray, shape (mnleq,) Value of the nonlinear equality constraint function of the nonlinear optimization problem at *x*.

1.3.3 cobyqa.optimize.TrustRegion.check_models

`TrustRegion.check_models`(*stack_level=2*)

Check the interpolation conditions.

The method checks whether the evaluations of the quadratic models at the interpolation points match their expected values.

Parameters

stack_level [int, optional] Stack level of the warning (the default is 2).

Warns

RuntimeWarning The evaluations of a quadratic function do not satisfy the interpolation conditions up to a certain tolerance.

1.3.4 cobyqa.optimize.TrustRegion.check_options

`TrustRegion.`**`check_options`**`(`*n*, *stack_level=2*`)`

Ensure that the options are consistent, and modify them if necessary.

Parameters

n [int] Number of decision variables.

stack_level [int, optional] Stack level of the warning (the default is 2).

Warns

RuntimeWarning The options are inconsistent and modified.

1.3.5 cobyqa.optimize.TrustRegion.cub

`TrustRegion.`**`cub`**`(`*x*`)`

Evaluate the nonlinear inequality constraint function of the nonlinear optimization problem.

Parameters

x [array_like, shape (n,)] Point at which the constraint function is to be evaluated.

Returns

numpy.ndarray, shape (mnlub,) Value of the nonlinear inequality constraint function of the nonlinear optimization problem at *x*.

1.3.6 cobyqa.optimize.TrustRegion.fun

`TrustRegion.`**`fun`**`(`*x*`)`

Evaluate the objective function of the nonlinear optimization problem.

Parameters

x [array_like, shape (n,)] Point at which the objective function is to be evaluated.

Returns

float Value of the objective function of the nonlinear optimization problem at *x*.

1.3.7 cobyqa.optimize.TrustRegion.get_best_point

`TrustRegion.`**`get_best_point`**`()`

Get the index of the optimal interpolation point.

Returns

int Index of the optimal interpolation point.

1.3.8 cobyqa.optimize.TrustRegion.less_merit

`TrustRegion.less_merit`(*mval1*, *rval1*, *mval2*, *rval2*)

Indicates whether a point is better than another.

Parameters

mval1 [float] Merit value associated with the first point.

rval1 [float] Residual value associated with the first point.

mval2 [float] Merit value associated with the second point.

rval2 [float] Residual value associated with the second point.

Returns

bool A flag indicating whether the first point is better than the other.

1.3.9 cobyqa.optimize.TrustRegion.model_ceq

`TrustRegion.model_ceq`(*x*, *i*)

Evaluate an equality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

i [int] Index of the equality constraint to be considered.

Returns

float Value of the i-th equality constraint function of the model at x.

1.3.10 cobyqa.optimize.TrustRegion.model_ceq_alt

`TrustRegion.model_ceq_alt`(*x*, *i*)

Evaluate an alternative equality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

i [int] Index of the equality constraint to be considered.

Returns

float Value of the i-th alternative equality constraint function of the model at x.

1.3.11 cobyqa.optimize.TrustRegion.model_ceq_alt_curv

`TrustRegion.model_ceq_alt_curv`(*x*, *i*)

Evaluate the curvature of an alternative equality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

i [int] Index of the equality constraint to be considered.

Returns

float Curvature of the i-th alternative equality constraint function of the model at x.

1.3.12 cobyqa.optimize.TrustRegion.model_ceq_alt_grad

`TrustRegion.`**`model_ceq_alt_grad`**`(`x, i`)`

Evaluate the gradient of an alternative equality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

i [int] Index of the equality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Gradient of the i-th alternative equality constraint function of the model at x.

1.3.13 cobyqa.optimize.TrustRegion.model_ceq_alt_hess

`TrustRegion.`**`model_ceq_alt_hess`**`(`i`)`

Evaluate the Hessian matrix of an alternative equality constraint function of the model.

Parameters

i [int] Index of the equality constraint to be considered.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the i-th alternative equality constraint function of the model.

1.3.14 cobyqa.optimize.TrustRegion.model_ceq_alt_hessp

`TrustRegion.`**`model_ceq_alt_hessp`**`(`x, i`)`

Evaluate the product of the Hessian matrix of an alternative equality constraint function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

i [int] Index of the equality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the i-th alternative equality constraint function of the model with the vector x.

1.3.15 cobyqa.optimize.TrustRegion.model_ceq_curv

`TrustRegion.model_ceq_curv(x, i)`

Evaluate the curvature of an equality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

i [int] Index of the equality constraint to be considered.

Returns

float Curvature of the *i*-th equality constraint function of the model at *x*.

1.3.16 cobyqa.optimize.TrustRegion.model_ceq_grad

`TrustRegion.model_ceq_grad(x, i)`

Evaluate the gradient of an equality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

i [int] Index of the equality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Gradient of the *i*-th equality constraint function of the model at *x*.

1.3.17 cobyqa.optimize.TrustRegion.model_ceq_hess

`TrustRegion.model_ceq_hess(i)`

Evaluate the Hessian matrix of an equality constraint function of the model.

Parameters

i [int] Index of the equality constraint to be considered.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the *i*-th equality constraint function of the model.

1.3.18 cobyqa.optimize.TrustRegion.model_ceq_hessp

`TrustRegion.model_ceq_hessp(x, i)`

Evaluate the product of the Hessian matrix of an equality constraint function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

i [int] Index of the equality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the i-th equality constraint function of the model with the vector x.

1.3.19 cobyqa.optimize.TrustRegion.model_cub

`TrustRegion.`**`model_cub`**`(`*x*, *i*`)`

Evaluate an inequality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

i [int] Index of the inequality constraint to be considered.

Returns

float Value of the i-th inequality constraint function of the model at x.

1.3.20 cobyqa.optimize.TrustRegion.model_cub_alt

`TrustRegion.`**`model_cub_alt`**`(`*x*, *i*`)`

Evaluate an alternative inequality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

i [int] Index of the inequality constraint to be considered.

Returns

float Value of the i-th alternative inequality constraint function of the model at x.

1.3.21 cobyqa.optimize.TrustRegion.model_cub_alt_curv

`TrustRegion.`**`model_cub_alt_curv`**`(`*x*, *i*`)`

Evaluate the curvature of an alternative inequality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

i [int] Index of the inequality constraint to be considered.

Returns

float Curvature of the i-th alternative inequality constraint function of the model at x.

1.3.22 cobyqa.optimize.TrustRegion.model_cub_alt_grad

TrustRegion.**model_cub_alt_grad**(*x*, *i*)

Evaluate the gradient of an alternative inequality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

i [int] Index of the inequality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Gradient of the *i*-th alternative inequality constraint function of the model at *x*.

1.3.23 cobyqa.optimize.TrustRegion.model_cub_alt_hess

TrustRegion.**model_cub_alt_hess**(*i*)

Evaluate the Hessian matrix of an alternative inequality constraint function of the model.

Parameters

i [int] Index of the inequality constraint to be considered.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the *i*-th alternative inequality constraint function of the model.

1.3.24 cobyqa.optimize.TrustRegion.model_cub_alt_hessp

TrustRegion.**model_cub_alt_hessp**(*x*, *i*)

Evaluate the product of the Hessian matrix of an alternative inequality constraint function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

i [int] Index of the inequality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the *i*-th alternative inequality constraint function of the model with the vector *x*.

1.3.25 cobyqa.optimize.TrustRegion.model_cub_curv

`TrustRegion.model_cub_curv`(*x*, *i*)

Evaluate the curvature of an inequality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

i [int] Index of the inequality constraint to be considered.

Returns

float Curvature of the *i*-th inequality constraint function of the model at *x*.

1.3.26 cobyqa.optimize.TrustRegion.model_cub_grad

`TrustRegion.model_cub_grad`(*x*, *i*)

Evaluate the gradient of an inequality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

i [int] Index of the inequality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Gradient of the *i*-th inequality constraint function of the model at *x*.

1.3.27 cobyqa.optimize.TrustRegion.model_cub_hess

`TrustRegion.model_cub_hess`(*i*)

Evaluate the Hessian matrix of an inequality constraint function of the model.

Parameters

i [int] Index of the inequality constraint to be considered.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the *i*-th inequality constraint function of the model.

1.3.28 cobyqa.optimize.TrustRegion.model_cub_hessp

`TrustRegion.model_cub_hessp`(*x*, *i*)

Evaluate the product of the Hessian matrix of an inequality constraint function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

i [int] Index of the inequality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the i-th inequality constraint function of the model with the vector x.

1.3.29 cobyqa.optimize.TrustRegion.model_lag

`TrustRegion.`**`model_lag`**`(x)`

Evaluate the Lagrangian function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

Returns

float Value of the Lagrangian function of the model at x.

1.3.30 cobyqa.optimize.TrustRegion.model_lag_alt

`TrustRegion.`**`model_lag_alt`**`(x)`

Evaluate the alternative Lagrangian function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

Returns

float Value of the alternative Lagrangian function of the model at x.

1.3.31 cobyqa.optimize.TrustRegion.model_lag_alt_curv

`TrustRegion.`**`model_lag_alt_curv`**`(x)`

Evaluate the curvature of the alternative Lagrangian function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

Returns

float Curvature of the alternative Lagrangian function of the model at x.

1.3.32 cobyqa.optimize.TrustRegion.model_lag_alt_grad

`TrustRegion.`**`model_lag_alt_grad`**`(x)`

Evaluate the gradient of the alternative Lagrangian function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

Returns

numpy.ndarray, shape (n,) Gradient of the alternative Lagrangian function of the model at *x*.

1.3.33 cobyqa.optimize.TrustRegion.model_lag_alt_hess

`TrustRegion.`**`model_lag_alt_hess`**`()`

Evaluate the Hessian matrix of the alternative Lagrangian function of the model.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the alternative Lagrangian function of the model.

1.3.34 cobyqa.optimize.TrustRegion.model_lag_alt_hessp

`TrustRegion.`**`model_lag_alt_hessp`**`(x)`

Evaluate the product of the Hessian matrix of the alternative Lagrangian function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the alternative Lagrangian function of the model with the vector *x*.

1.3.35 cobyqa.optimize.TrustRegion.model_lag_curv

`TrustRegion.`**`model_lag_curv`**`(x)`

Evaluate the curvature of the Lagrangian function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

Returns

float Curvature of the Lagrangian function of the model at *x*.

1.3.36 cobyqa.optimize.TrustRegion.model_lag_grad

`TrustRegion.`**`model_lag_grad`**`(x)`

Evaluate the gradient of Lagrangian function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

Returns

numpy.ndarray, shape (n,) Gradient of the Lagrangian function of the model at *x*.

1.3.37 cobyqa.optimize.TrustRegion.model_lag_hess

`TrustRegion.model_lag_hess()`

Evaluate the Hessian matrix of the Lagrangian function of the model.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the Lagrangian function of the model.

1.3.38 cobyqa.optimize.TrustRegion.model_lag_hessp

`TrustRegion.model_lag_hessp(x)`

Evaluate the product of the Hessian matrix of the Lagrangian function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the Lagrangian function of the model with the vector *x*.

1.3.39 cobyqa.optimize.TrustRegion.model_obj

`TrustRegion.model_obj(x)`

Evaluate the objective function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

Returns

float Value of the objective function of the model at *x*.

1.3.40 cobyqa.optimize.TrustRegion.model_obj_alt

`TrustRegion.model_obj_alt(x)`

Evaluate the alternative objective function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

Returns

float Value of the alternative objective function of the model at *x*.

1.3.41 cobyqa.optimize.TrustRegion.model_obj_alt_curv

TrustRegion.**model_obj_alt_curv**(*x*)

Evaluate the curvature of the alternative objective function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

Returns

float Curvature of the alternative objective function of the model at *x*.

1.3.42 cobyqa.optimize.TrustRegion.model_obj_alt_grad

TrustRegion.**model_obj_alt_grad**(*x*)

Evaluate the gradient of the alternative objective function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

Returns

numpy.ndarray, shape (n,) Gradient of the alternative objective function of the model at *x*.

1.3.43 cobyqa.optimize.TrustRegion.model_obj_alt_hess

TrustRegion.**model_obj_alt_hess**()

Evaluate the Hessian matrix of the alternative objective function of the model.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the alternative objective function of the model.

1.3.44 cobyqa.optimize.TrustRegion.model_obj_alt_hessp

TrustRegion.**model_obj_alt_hessp**(*x*)

Evaluate the product of the Hessian matrix of the alternative objective function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the alternative objective function of the model with the vector *x*.

1.3.45 cobyqa.optimize.TrustRegion.model_obj_curv

`TrustRegion.model_obj_curv(x)`

Evaluate the curvature of the objective function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

Returns

float Curvature of the objective function of the model at *x*.

1.3.46 cobyqa.optimize.TrustRegion.model_obj_grad

`TrustRegion.model_obj_grad(x)`

Evaluate the gradient of the objective function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

Returns

numpy.ndarray, shape (n,) Gradient of the objective function of the model at *x*.

1.3.47 cobyqa.optimize.TrustRegion.model_obj_hess

`TrustRegion.model_obj_hess()`

Evaluate the Hessian matrix of the objective function of the model.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the objective function of the model.

1.3.48 cobyqa.optimize.TrustRegion.model_obj_hessp

`TrustRegion.model_obj_hessp(x)`

Evaluate the product of the Hessian matrix of the objective function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the objective function of the model with the vector *x*.

1.3.49 cobyqa.optimize.TrustRegion.model_step

`TrustRegion.model_step(delta, **kwargs)`

Estimate a model-improvement step from `xopt`.

Parameters

delta [float] Trust-region radius.

Returns

numpy.ndarray, shape (n,) Model-improvement step from `xopt`.

Other Parameters

bdtol [float, optional] Tolerance for comparisons on the bound constraints (the default is `10 * eps * n * max(1, max(abs(xl)), max(abs(xu)))`, where the values of `xl` and `xu` evolve to include the shift of the origin).

Notes

Two alternative steps are computed.

1. The first alternative step is selected on the lines that join `xopt` to the other interpolation points that maximize a lower bound on the denominator of the updating formula.
2. The second alternative is a constrained Cauchy step.

Among the two alternative steps, the method selects the one that leads to the greatest denominator in Equation (2.12) of [1].

References

[1]

1.3.50 cobyqa.optimize.TrustRegion.prepare_model_step

`TrustRegion.prepare_model_step(delta)`

Set the next iteration to a model-step if necessary.

The method checks whether the furthest interpolation point from `xopt` is more than the provided trust-region radius to set a model-step. If such a point does not exist, the next iteration is a trust-region step.

Parameters

delta [float] Trust-region radius.

1.3.51 cobyqa.optimize.TrustRegion.prepare_trust_region_step

`TrustRegion.prepare_trust_region_step()`

Set the next iteration to a trust-region step.

1.3.52 cobyqa.optimize.TrustRegion.reduce_penalty_coefficients

`TrustRegion.reduce_penalty_coefficients()`

Reduce the penalty coefficients if possible, to prevent overflows.

Notes

The thresholds at which the penalty coefficients are set are empirical and based on Equation (13) of [1].

References

[1]

1.3.53 cobyqa.optimize.TrustRegion.reset_models

`TrustRegion.reset_models()`

Reset the models.

The standard models of the objective function, the nonlinear inequality constraint function, and the nonlinear equality constraint function are set to the ones whose Hessian matrices are least in Frobenius norm.

1.3.54 cobyqa.optimize.TrustRegion.set_default_options

`TrustRegion.set_default_options(n)`

Set the default options for the solvers.

Parameters

n [int] Number of decision variables.

1.3.55 cobyqa.optimize.TrustRegion.shift_origin

`TrustRegion.shift_origin(delta)`

Shift the origin of the calculations if necessary.

Although the shift of the origin in the calculations does not change anything from a theoretical point of view, it is designed to tackle numerical difficulties caused by ill-conditioned problems. If the method is triggered, the origin is shifted to the best point so far.

Parameters

delta [float] Trust-region radius.

1.3.56 cobyqa.optimize.TrustRegion.trust_region_step

`TrustRegion.`**`trust_region_step`**(*delta*, ***kwargs*)

Evaluate a Byrd-Omojokun-like trust-region step from `xopt`.

Parameters

delta [float] Trust-region radius.

Returns

numpy.ndarray, shape (n,) Trust-region step from `xopt`.

Other Parameters

bdtol [float, optional] Tolerance for comparisons on the bound constraints (the default is `10 * eps * n * max(1, max(abs(xl)), max(abs(xu)))`, where the values of `xl` and `xu` evolve to include the shift of the origin).

lctol [float, optional] Tolerance for comparisons on the linear constraints (the default is `10 * eps * n * max(1, max(abs(bub)))`, where the values of `bub` evolve to include the shift of the origin).

Notes

The trust-region constraint of the tangential subproblem is not centered if the normal step is nonzero. To cope with this difficulty, we use the result presented in Equation (15.4.3) of [1].

References

[1]

1.3.57 cobyqa.optimize.TrustRegion.update

`TrustRegion.`**`update`**(*step*, ***kwargs*)

Include a new point in the interpolation set.

When the new point is included in the interpolation set, the models of the nonlinear optimization problems are updated.

Parameters

step [numpy.ndarray, shape (n,)] Step from `xopt` of the new point to include in the interpolation set.

Returns

mopt [float] Merit value of the new interpolation point.

ratio [float] Trust-region ratio associated with the new interpolation point.

Other Parameters

lstol [float, optional] Tolerance on the approximate KKT conditions for the calculations of the least-squares Lagrange multipliers (the default is `10 * eps * max(n, m) * max(1, max(abs(g)))`, where `g` is the gradient of the current model of the objective function).

Raises

RestartRequiredException The iteration must be restarted because the index of the optimal point among the interpolation set has changed.

1.3.58 cobyqa.optimize.TrustRegion.update_multipliers

`TrustRegion.update_multipliers(**kwargs)`

Set the least-squares Lagrange multipliers.

Other Parameters

lstol [float, optional] Tolerance on the approximate KKT conditions for the calculations of the least-squares Lagrange multipliers (the default is `10 * eps * max(n, m) * max(1, max(abs(g)))`, where `g` is the gradient of the current model of the objective function).

1.3.59 cobyqa.optimize.TrustRegion.update_penalty_coefficients

`TrustRegion.update_penalty_coefficients(step, fx, cubx, ceqx)`

Increase the penalty coefficients.

The penalty coefficients are increased to make the trust-region ratio meaningful. The increasing process of the penalty coefficients may be prematurely stop if the index of the best point so far changes.

Parameters

step [numpy.ndarray, shape (n,)] Trial step from `xopt`.

fx [float] Value of the objective function at the trial point.

cubx [numpy.ndarray, shape (mnlub,)] Value of the nonlinear inequality constraint function at the trial point.

ceqx [numpy.ndarray, shape (mnleq,)] Value of the nonlinear equality constraint function at the trial point.

Returns

mx [float] Value of the merit function at the trial point, evaluated on the nonlinear optimization problem.

mmx [float] Value of the merit function at the trial point, evaluated on the different models.

mopt [float] Value of the merit function at `xopt`, evaluated on the nonlinear optimization problem.

1.4 cobyqa.optimize.Models

class `cobyqa.optimize.Models(fun, x0, xl, xu, Aub, bub, Aeq, beq, cub, ceq, options)`

Model a nonlinear optimization problem.

The nonlinear optimization problem is modeled using quadratic functions obtained by underdetermined interpolation. The interpolation points may be infeasible with respect to the linear and nonlinear constraints, but they always satisfy the bound constraints.

Notes

Given the interpolation set, the freedom bequeathed by the interpolation conditions is taken up by minimizing the updates of the Hessian matrices of the objective and nonlinear constraint functions in Frobenius norm [1].

References

[1]

Attributes

aeq Jacobian matrix of the linear equality constraints.

aub Jacobian matrix of the linear inequality constraints.

beq Right-hand side vector of the linear equality constraints.

bmat Last `n` columns of the inverse KKT matrix of interpolation.

bub Right-hand side vector of the linear inequality constraints.

copteq Evaluation of the nonlinear equality constraint function of the nonlinear optimization problem at `xopt`.

coptub Evaluation of the nonlinear inequality constraint function of the nonlinear optimization problem at `xopt`.

cvaleq Evaluations of the nonlinear equality constraint function of the nonlinear optimization problem at the interpolation points.

cvalub Evaluations of the nonlinear inequality constraint function of the nonlinear optimization problem at the interpolation points.

fopt Evaluation of the objective function of the nonlinear optimization problem at `xopt`.

fval Evaluations of the objective function of the nonlinear optimization problem at the interpolation points.

idz Number of nonpositive eigenvalues of the leading `npt` submatrix of the inverse KKT matrix of interpolation.

kopt Index of the interpolation point around which the Taylor expansions of the quadratic models are defined.

mleq Number of the linear equality constraints.

mlub Number of the linear inequality constraints.

mnleq Number of the nonlinear equality constraints.

mnlub Number of the nonlinear inequality constraints.

ropt Residual associated with the constraints of the nonlinear optimization problem at `xopt`.

rval Residuals associated with the constraints of the nonlinear optimization problem at the interpolation points.

type Type of the nonlinear optimization problem.

xl Lower-bound constraints on the decision variables.

xopt Interpolation point around which the Taylor expansion of the quadratic models are defined.

xpt Displacements of the interpolation points from the origin.

xu Upper-bound constraints on the decision variables.

zmat Rank factorization matrix of the leading `npt` submatrix of the inverse KKT matrix of interpolation.

Methods

ceq(x, i)	Evaluate an equality constraint function of the model.
ceq_alt(x, i)	Evaluate an alternative equality constraint function of the model.
ceq_alt_curv(x, i)	Evaluate the curvature of an alternative equality constraint function of the model.
ceq_alt_grad(x, i)	Evaluate the gradient of an alternative equality constraint function of the model.
ceq_alt_hess(i)	Evaluate the Hessian matrix of an alternative equality constraint function of the model.
ceq_alt_hessp(x, i)	Evaluate the product of the Hessian matrix of an alternative equality constraint function of the model with any vector.
ceq_curv(x, i)	Evaluate the curvature of an equality constraint function of the model.
ceq_grad(x, i)	Evaluate the gradient of an equality constraint function of the model.
ceq_hess(i)	Evaluate the Hessian matrix of an equality constraint function of the model.
ceq_hessp(x, i)	Evaluate the product of the Hessian matrix of an equality constraint function of the model with any vector.
check_models([stack_level])	Check the interpolation conditions.
cub(x, i)	Evaluate an inequality constraint function of the model.
cub_alt(x, i)	Evaluate an alternative inequality constraint function of the model.
cub_alt_curv(x, i)	Evaluate the curvature of an alternative inequality constraint function of the model.
cub_alt_grad(x, i)	Evaluate the gradient of an alternative inequality constraint function of the model.
cub_alt_hess(i)	Evaluate the Hessian matrix of an alternative inequality constraint function of the model.
cub_alt_hessp(x, i)	Evaluate the product of the Hessian matrix of an alternative inequality constraint function of the model with any vector.
cub_curv(x, i)	Evaluate the curvature of an inequality constraint function of the model.
cub_grad(x, i)	Evaluate the gradient of an inequality constraint function of the model.
cub_hess(i)	Evaluate the Hessian matrix of an inequality constraint function of the model.
cub_hessp(x, i)	Evaluate the product of the Hessian matrix of an inequality constraint function of the model with any vector.
improve_geometry(klag, delta, **kwargs)	Estimate a step from `xopt` that aims at improving the geometry of the interpolation set.

continues on next page

Table 4 – continued from previous page

`lag`(x, lmlub, lmleq, lmnlub, lmnleq)	Evaluate the Lagrangian function of the model.
`lag_alt`(x, lmlub, lmleq, lmnlub, lmnleq)	Evaluate the alternative Lagrangian function of the model.
`lag_alt_curv`(x, lmnlub, lmnleq)	Evaluate the curvature of the alternative Lagrangian function of the model.
`lag_alt_grad`(x, lmlub, lmleq, lmnlub, lmnleq)	Evaluate the gradient of the alternative Lagrangian function of the model.
`lag_alt_hess`(lmnlub, lmnleq)	Evaluate the Hessian matrix of the alternative Lagrangian function of the model.
`lag_alt_hessp`(x, lmnlub, lmnleq)	Evaluate the product of the Hessian matrix of the alternative Lagrangian function of the model with any vector.
`lag_curv`(x, lmnlub, lmnleq)	Evaluate the curvature of the Lagrangian function of the model.
`lag_grad`(x, lmlub, lmleq, lmnlub, lmnleq)	Evaluate the gradient of Lagrangian function of the model.
`lag_hess`(lmnlub, lmnleq)	Evaluate the Hessian matrix of the Lagrangian function of the model.
`lag_hessp`(x, lmnlub, lmnleq)	Evaluate the product of the Hessian matrix of the Lagrangian function of the model with any vector.
`new_model`(val)	Generate a model obtained by underdetermined interpolation.
`obj`(x)	Evaluate the objective function of the model.
`obj_alt`(x)	Evaluate the alternative objective function of the model.
`obj_alt_curv`(x)	Evaluate the curvature of the alternative objective function of the model.
`obj_alt_grad`(x)	Evaluate the gradient of the alternative objective function of the model.
`obj_alt_hess`()	Evaluate the Hessian matrix of the alternative objective function of the model.
`obj_alt_hessp`(x)	Evaluate the product of the Hessian matrix of the alternative objective function of the model with any vector.
`obj_curv`(x)	Evaluate the curvature of the objective function of the model.
`obj_grad`(x)	Evaluate the gradient of the objective function of the model.
`obj_hess`()	Evaluate the Hessian matrix of the objective function of the model.
`obj_hessp`(x)	Evaluate the product of the Hessian matrix of the objective function of the model with any vector.
`reset_models`()	Reset the models.
`resid`(x[, cubx, ceqx])	Evaluate the residual associated with the constraints of the nonlinear optimization problem.
`shift_constraints`(x)	Shift the bound and linear constraints.
`shift_origin`()	Update the models when the origin of the calculations is modified.
`update`(step, fx, cubx, ceqx[, knew])	Update the models of the nonlinear optimization problem when a point of the interpolation set is modified.

1.4.1 cobyqa.optimize.Models.ceq

Models.**ceq**(*x*, *i*)

Evaluate an equality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

i [int] Index of the equality constraint to be considered.

Returns

float Value of the *i*-th equality constraint function of the model at *x*.

1.4.2 cobyqa.optimize.Models.ceq_alt

Models.**ceq_alt**(*x*, *i*)

Evaluate an alternative equality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

i [int] Index of the equality constraint to be considered.

Returns

float Value of the *i*-th alternative equality constraint function of the model at *x*.

1.4.3 cobyqa.optimize.Models.ceq_alt_curv

Models.**ceq_alt_curv**(*x*, *i*)

Evaluate the curvature of an alternative equality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

i [int] Index of the equality constraint to be considered.

Returns

float Curvature of the *i*-th alternative equality constraint function of the model at *x*.

1.4.4 cobyqa.optimize.Models.ceq_alt_grad

Models.**ceq_alt_grad**(*x*, *i*)

Evaluate the gradient of an alternative equality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

i [int] Index of the equality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Gradient of the *i*-th alternative equality constraint function of the model at *x*.

1.4.5 cobyqa.optimize.Models.ceq_alt_hess

`Models.`**`ceq_alt_hess`**(*i*)

Evaluate the Hessian matrix of an alternative equality constraint function of the model.

Parameters

i [int] Index of the equality constraint to be considered.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the *i*-th alternative equality constraint function of the model.

1.4.6 cobyqa.optimize.Models.ceq_alt_hessp

`Models.`**`ceq_alt_hessp`**(*x*, *i*)

Evaluate the product of the Hessian matrix of an alternative equality constraint function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

i [int] Index of the equality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the *i*-th alternative equality constraint function of the model with the vector *x*.

1.4.7 cobyqa.optimize.Models.ceq_curv

`Models.`**`ceq_curv`**(*x*, *i*)

Evaluate the curvature of an equality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

i [int] Index of the equality constraint to be considered.

Returns

float Curvature of the *i*-th equality constraint function of the model at *x*.

1.4.8 cobyqa.optimize.Models.ceq_grad

`Models.`**`ceq_grad`**`(`*x*, *i*`)`

Evaluate the gradient of an equality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

i [int] Index of the equality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Gradient of the *i*-th equality constraint function of the model at *x*.

1.4.9 cobyqa.optimize.Models.ceq_hess

`Models.`**`ceq_hess`**`(`*i*`)`

Evaluate the Hessian matrix of an equality constraint function of the model.

Parameters

i [int] Index of the equality constraint to be considered.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the *i*-th equality constraint function of the model.

1.4.10 cobyqa.optimize.Models.ceq_hessp

`Models.`**`ceq_hessp`**`(`*x*, *i*`)`

Evaluate the product of the Hessian matrix of an equality constraint function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

i [int] Index of the equality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the *i*-th equality constraint function of the model with the vector *x*.

1.4.11 cobyqa.optimize.Models.check_models

`Models.`**`check_models`**`(`*stack_level=2*`)`

Check the interpolation conditions.

The method checks whether the evaluations of the quadratic models at the interpolation points match their expected values.

Parameters

stack_level [int, optional] Stack level of the warning (the default is 2).

Warns

RuntimeWarning The evaluations of a quadratic function do not satisfy the interpolation conditions up to a certain tolerance.

1.4.12 cobyqa.optimize.Models.cub

`Models.cub(x, i)`

Evaluate an inequality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

i [int] Index of the inequality constraint to be considered.

Returns

float Value of the i-th inequality constraint function of the model at x.

1.4.13 cobyqa.optimize.Models.cub_alt

`Models.cub_alt(x, i)`

Evaluate an alternative inequality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

i [int] Index of the inequality constraint to be considered.

Returns

float Value of the i-th alternative inequality constraint function of the model at x.

1.4.14 cobyqa.optimize.Models.cub_alt_curv

`Models.cub_alt_curv(x, i)`

Evaluate the curvature of an alternative inequality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

i [int] Index of the inequality constraint to be considered.

Returns

float Curvature of the i-th alternative inequality constraint function of the model at x.

1.4.15 cobyqa.optimize.Models.cub_alt_grad

`Models.cub_alt_grad(x, i)`

Evaluate the gradient of an alternative inequality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

i [int] Index of the inequality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Gradient of the *i*-th alternative inequality constraint function of the model at *x*.

1.4.16 cobyqa.optimize.Models.cub_alt_hess

`Models.cub_alt_hess(i)`

Evaluate the Hessian matrix of an alternative inequality constraint function of the model.

Parameters

i [int] Index of the inequality constraint to be considered.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the *i*-th alternative inequality constraint function of the model.

1.4.17 cobyqa.optimize.Models.cub_alt_hessp

`Models.cub_alt_hessp(x, i)`

Evaluate the product of the Hessian matrix of an alternative inequality constraint function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

i [int] Index of the inequality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the *i*-th alternative inequality constraint function of the model with the vector *x*.

1.4.18 cobyqa.optimize.Models.cub_curv

`Models.cub_curv(x, i)`

Evaluate the curvature of an inequality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

i [int] Index of the inequality constraint to be considered.

Returns

float Curvature of the *i*-th inequality constraint function of the model at *x*.

1.4.19 cobyqa.optimize.Models.cub_grad

`Models.cub_grad(x, i)`

Evaluate the gradient of an inequality constraint function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

i [int] Index of the inequality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Gradient of the *i*-th inequality constraint function of the model at *x*.

1.4.20 cobyqa.optimize.Models.cub_hess

`Models.cub_hess(i)`

Evaluate the Hessian matrix of an inequality constraint function of the model.

Parameters

i [int] Index of the inequality constraint to be considered.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the *i*-th inequality constraint function of the model.

1.4.21 cobyqa.optimize.Models.cub_hessp

`Models.cub_hessp(x, i)`

Evaluate the product of the Hessian matrix of an inequality constraint function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

i [int] Index of the inequality constraint to be considered.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the *i*-th inequality constraint function of the model with the vector *x*.

1.4.22 cobyqa.optimize.Models.improve_geometry

`Models.`**`improve_geometry`**(*klag*, *delta*, ***kwargs*)

Estimate a step from `xopt` that aims at improving the geometry of the interpolation set.

Two alternative steps are computed.

1. The first alternative step is selected on the lines that join `xopt` to the other interpolation points that maximize a lower bound on the denominator of the updating formula.
2. The second alternative is a constrained Cauchy step.

Among the two alternative steps, the method selects the one that leads to the greatest denominator of the updating formula.

Parameters

klag [int] Index of the interpolation point that is to be replaced.

delta [float] Upper bound on the length of the step.

Returns

numpy.ndarray, shape (n,) Step from `xopt` that aims at improving the geometry of the interpolation set.

Other Parameters

bdtol [float, optional] Tolerance for comparisons on the bound constraints (the default is `10 * eps * n * max(1, max(abs(xl)), max(abs(xu)))`, where the values of `xl` and `xu` evolve to include the shift of the origin).

1.4.23 cobyqa.optimize.Models.lag

`Models.`**`lag`**(*x*, *lmlub*, *lmleq*, *lmnlub*, *lmnleq*)

Evaluate the Lagrangian function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

lmlub [numpy.ndarray, shape (mlub,)] Lagrange multipliers associated with the linear inequality constraints.

lmleq [numpy.ndarray, shape (mleq,)] Lagrange multipliers associated with the linear equality constraints.

lmnlub [numpy.ndarray, shape (mnlub,)] Lagrange multipliers associated with the quadratic models of the nonlinear inequality constraints.

lmnleq [numpy.ndarray, shape (mnleq,)] Lagrange multipliers associated with the quadratic models of the nonlinear equality constraints.

Returns

float Value of the Lagrangian function of the model at *x*.

1.4.24 cobyqa.optimize.Models.lag_alt

`Models.`**`lag_alt`**(*x*, *lmlub*, *lmleq*, *lmnlub*, *lmnleq*)

Evaluate the alternative Lagrangian function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

lmlub [numpy.ndarray, shape (mlub,)] Lagrange multipliers associated with the linear inequality constraints.

lmleq [numpy.ndarray, shape (mleq,)] Lagrange multipliers associated with the linear equality constraints.

lmnlub [numpy.ndarray, shape (mnlub,)] Lagrange multipliers associated with the quadratic models of the nonlinear inequality constraints.

lmnleq [numpy.ndarray, shape (mnleq,)] Lagrange multipliers associated with the quadratic models of the nonlinear equality constraints.

Returns

float Value of the alternative Lagrangian function of the model at *x*.

1.4.25 cobyqa.optimize.Models.lag_alt_curv

`Models.`**`lag_alt_curv`**(*x*, *lmnlub*, *lmnleq*)

Evaluate the curvature of the alternative Lagrangian function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

lmnlub [numpy.ndarray, shape (mnlub,)] Lagrange multipliers associated with the quadratic models of the nonlinear inequality constraints.

lmnleq [numpy.ndarray, shape (mnleq,)] Lagrange multipliers associated with the quadratic models of the nonlinear equality constraints.

Returns

float Curvature of the alternative Lagrangian function of the model at *x*.

1.4.26 cobyqa.optimize.Models.lag_alt_grad

`Models.`**`lag_alt_grad`**(*x*, *lmlub*, *lmleq*, *lmnlub*, *lmnleq*)

Evaluate the gradient of the alternative Lagrangian function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

lmlub [numpy.ndarray, shape (mlub,)] Lagrange multipliers associated with the linear inequality constraints.

lmleq [numpy.ndarray, shape (mleq,)] Lagrange multipliers associated with the linear equality constraints.

lmnlub [numpy.ndarray, shape (mnlub,)] Lagrange multipliers associated with the quadratic models of the nonlinear inequality constraints.

lmnleq [numpy.ndarray, shape (mnleq,)] Lagrange multipliers associated with the quadratic models of the nonlinear equality constraints.

Returns

numpy.ndarray, shape (n,) Gradient of the alternative Lagrangian function of the model at *x*.

1.4.27 cobyqa.optimize.Models.lag_alt_hess

`Models.`**`lag_alt_hess`**(*lmnlub*, *lmnleq*)

Evaluate the Hessian matrix of the alternative Lagrangian function of the model.

Parameters

lmnlub [numpy.ndarray, shape (mnlub,)] Lagrange multipliers associated with the quadratic models of the nonlinear inequality constraints.

lmnleq [numpy.ndarray, shape (mnleq,)] Lagrange multipliers associated with the quadratic models of the nonlinear equality constraints.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the alternative Lagrangian function of the model.

1.4.28 cobyqa.optimize.Models.lag_alt_hessp

`Models.`**`lag_alt_hessp`**(*x*, *lmnlub*, *lmnleq*)

Evaluate the product of the Hessian matrix of the alternative Lagrangian function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

lmnlub [numpy.ndarray, shape (mnlub,)] Lagrange multipliers associated with the quadratic models of the nonlinear inequality constraints.

lmnleq [numpy.ndarray, shape (mnleq,)] Lagrange multipliers associated with the quadratic models of the nonlinear equality constraints.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the alternative Lagrangian function of the model with the vector *x*.

1.4.29 cobyqa.optimize.Models.lag_curv

`Models.lag_curv(x, lmnlub, lmnleq)`

Evaluate the curvature of the Lagrangian function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

lmnlub [numpy.ndarray, shape (mnlub,)] Lagrange multipliers associated with the quadratic models of the nonlinear inequality constraints.

lmnleq [numpy.ndarray, shape (mnleq,)] Lagrange multipliers associated with the quadratic models of the nonlinear equality constraints.

Returns

float Curvature of the Lagrangian function of the model at *x*.

1.4.30 cobyqa.optimize.Models.lag_grad

`Models.lag_grad(x, lmlub, lmleq, lmnlub, lmnleq)`

Evaluate the gradient of Lagrangian function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

lmlub [numpy.ndarray, shape (mlub,)] Lagrange multipliers associated with the linear inequality constraints.

lmleq [numpy.ndarray, shape (mleq,)] Lagrange multipliers associated with the linear equality constraints.

lmnlub [numpy.ndarray, shape (mnlub,)] Lagrange multipliers associated with the quadratic models of the nonlinear inequality constraints.

lmnleq [numpy.ndarray, shape (mnleq,)] Lagrange multipliers associated with the quadratic models of the nonlinear equality constraints.

Returns

numpy.ndarray, shape (n,) Gradient of the Lagrangian function of the model at *x*.

1.4.31 cobyqa.optimize.Models.lag_hess

`Models.lag_hess(lmnlub, lmnleq)`

Evaluate the Hessian matrix of the Lagrangian function of the model.

Parameters

lmnlub [numpy.ndarray, shape (mnlub,)] Lagrange multipliers associated with the quadratic models of the nonlinear inequality constraints.

lmnleq [numpy.ndarray, shape (mnleq,)] Lagrange multipliers associated with the quadratic models of the nonlinear equality constraints.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the Lagrangian function of the model.

1.4.32 cobyqa.optimize.Models.lag_hessp

`Models.`**`lag_hessp`**`(`*x*, *lmnlub*, *lmnleq*`)`

Evaluate the product of the Hessian matrix of the Lagrangian function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

lmnlub [numpy.ndarray, shape (mnlub,)] Lagrange multipliers associated with the quadratic models of the nonlinear inequality constraints.

lmnleq [numpy.ndarray, shape (mnleq,)] Lagrange multipliers associated with the quadratic models of the nonlinear equality constraints.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the Lagrangian function of the model with the vector *x*.

1.4.33 cobyqa.optimize.Models.new_model

`Models.`**`new_model`**`(`*val*`)`

Generate a model obtained by underdetermined interpolation.

The freedom bequeathed by the interpolation conditions defined by *val* is taken up by minimizing the Hessian matrix of the quadratic function in Frobenius norm.

Parameters

val [int or numpy.ndarray, shape (npt,)] Evaluations associated with the interpolation points. An integer value represents the `npt`-dimensional vector whose components are all zero, except the *val*-th one whose value is one. Hence, passing an integer value construct the *val*-th Lagrange polynomial associated with the interpolation points.

Returns

Quadratic The quadratic model that satisfy the interpolation conditions defined by *val*, whose Hessian matrix is least in Frobenius norm.

1.4.34 cobyqa.optimize.Models.obj

`Models.`**`obj`**`(`*x*`)`

Evaluate the objective function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

Returns

float Value of the objective function of the model at *x*.

1.4.35 cobyqa.optimize.Models.obj_alt

`Models.obj_alt(x)`

Evaluate the alternative objective function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

Returns

float Value of the alternative objective function of the model at *x*.

1.4.36 cobyqa.optimize.Models.obj_alt_curv

`Models.obj_alt_curv(x)`

Evaluate the curvature of the alternative objective function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

Returns

float Curvature of the alternative objective function of the model at *x*.

1.4.37 cobyqa.optimize.Models.obj_alt_grad

`Models.obj_alt_grad(x)`

Evaluate the gradient of the alternative objective function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

Returns

numpy.ndarray, shape (n,) Gradient of the alternative objective function of the model at *x*.

1.4.38 cobyqa.optimize.Models.obj_alt_hess

`Models.obj_alt_hess()`

Evaluate the Hessian matrix of the alternative objective function of the model.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the alternative objective function of the model.

1.4.39 cobyqa.optimize.Models.obj_alt_hessp

`Models.obj_alt_hessp(x)`

Evaluate the product of the Hessian matrix of the alternative objective function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the alternative objective function of the model with the vector *x*.

1.4.40 cobyqa.optimize.Models.obj_curv

`Models.obj_curv(x)`

Evaluate the curvature of the objective function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

Returns

float Curvature of the objective function of the model at *x*.

1.4.41 cobyqa.optimize.Models.obj_grad

`Models.obj_grad(x)`

Evaluate the gradient of the objective function of the model.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

Returns

numpy.ndarray, shape (n,) Gradient of the objective function of the model at *x*.

1.4.42 cobyqa.optimize.Models.obj_hess

`Models.obj_hess()`

Evaluate the Hessian matrix of the objective function of the model.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the objective function of the model.

1.4.43 cobyqa.optimize.Models.obj_hessp

`Models.obj_hessp(x)`

Evaluate the product of the Hessian matrix of the objective function of the model with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the objective function of the model with the vector *x*.

1.4.44 cobyqa.optimize.Models.reset_models

`Models.reset_models()`

Reset the models.

The standard models of the objective function, the nonlinear inequality constraint function, and the nonlinear equality constraint function are set to the ones whose Hessian matrices are least in Frobenius norm.

1.4.45 cobyqa.optimize.Models.resid

`Models.resid(x, cubx=None, ceqx=None)`

Evaluate the residual associated with the constraints of the nonlinear optimization problem.

Parameters

x [int or numpy.ndarray, shape (n,)] Point at which the residual is to be evaluated. An integer value represents the *x*-th interpolation point.

cubx [numpy.ndarray, shape (mnlub,), optional] Value of the nonlinear inequality constraint function at *x*. It is required only if *x* is not an integer, and is not considered if *x* represents an interpolation point.

ceqx [numpy.ndarray, shape (mnleq,), optional] Value of the nonlinear equality constraint function at *x*. It is required only if *x* is not an integer, and is not considered if *x* represents an interpolation point.

Returns

float Residual associated with the constraints of the nonlinear optimization problem at *x*.

1.4.46 cobyqa.optimize.Models.shift_constraints

`Models.shift_constraints(x)`

Shift the bound and linear constraints.

Parameters

x [numpy.ndarray, shape (n,)] Coordinates of the shift to be performed.

1.4.47 cobyqa.optimize.Models.shift_origin

`Models.shift_origin()`

Update the models when the origin of the calculations is modified.

Notes

Given `xbase` the previous origin of the calculations, it is assumed that the origin is shifted by `xopt`.

1.4.48 cobyqa.optimize.Models.update

`Models.update(step, fx, cubx, ceqx, knew=None)`

Update the models of the nonlinear optimization problem when a point of the interpolation set is modified.

Parameters

step [numpy.ndarray, shape (n,)] Displacement from `xopt` of the point to replace an interpolation point.

fx [float] Value of the objective function at the trial point.

cubx [numpy.ndarray, shape (mnlub,)] Value of the nonlinear inequality constraint function at the trial point.

ceqx [numpy.ndarray, shape (mnleq,)] Value of the nonlinear equality constraint function at the trial point.

knew [int, optional] Index of the interpolation point to be removed. It is automatically chosen if it is not provided.

Returns

int Index of the interpolation point that has been replaced.

Raises

ZeroDivisionError The denominator of the updating formula is zero.

Notes

When the index *knew* of the interpolation point to be removed is not provided, it is chosen by the method to maximize the product absolute value of the denominator in Equation (2.12) of [1] with the quartic power of the distance between the point and `xopt`.

References

[1]

1.5 cobyqa.optimize.Quadratic

class cobyqa.optimize.**Quadratic**(*bmat*, *zmat*, *idz*, *fval*)

Representation of a quadratic multivariate function.

Notes

To improve the computational efficiency of the updates of the quadratic functions, the Hessian matrix of a quadratic function is stored as an explicit and an implicit part, which define the model relatively to the coordinates of the interpolation points [1]. Initially, the explicit part of the Hessian matrix is zero and so, is not explicitly stored.

References

[1]

Attributes

gq Stored gradient of the model.

hq Stored explicit part of the Hessian matrix of the model.

pq Stored implicit part of the Hessian matrix of the model.

Methods

__call__(x, xpt, kopt)	Evaluate the quadratic function.
check_model(xpt, fval, kopt[, stack_level])	Check the interpolation conditions.
curv(x, xpt)	Evaluate the curvature of the quadratic function.
grad(x, xpt, kopt)	Evaluate the gradient of the quadratic function.
hess(xpt)	Evaluate the Hessian matrix of the quadratic function.
hessp(x, xpt)	Evaluate the product of the Hessian matrix of the quadratic function with any vector.
shift_expansion_point(step, xpt)	Shift the point around which the quadratic function is defined.
shift_interpolation_points(xpt, kopt)	Update the components of the quadratic function when the origin from which the interpolation points are defined is to be displaced.
update(xpt, kopt, xold, bmat, zmat, idz, ...)	Update the model when a point of the interpolation set is modified.

1.5.1 cobyqa.optimize.Quadratic.__call__

Quadratic.**__call__**(*x*, *xpt*, *kopt*)

Evaluate the quadratic function.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the quadratic function is to be evaluated.

xpt [numpy.ndarray, shape (npt, n)] Interpolation points that define the quadratic function. Each row of *xpt* stores the coordinates of an interpolation point.

kopt [int] Index of the interpolation point around which the quadratic function is defined. The constant term of the quadratic function is not maintained, and zero is returned at `xpt[kopt, :]`.

Returns

float Value of the quadratic function at *x*.

1.5.2 cobyqa.optimize.Quadratic.check_model

`Quadratic.`**`check_model`**(*xpt*, *fval*, *kopt*, *stack_level=2*)

Check the interpolation conditions.

The method checks whether the evaluations of the quadratic function at the interpolation points match their expected values.

Parameters

xpt [numpy.ndarray, shape (npt, n)] Interpolation points that define the quadratic function. Each row of *xpt* stores the coordinates of an interpolation point.

fval [numpy.ndarray, shape (npt,)] Evaluations associated with the interpolation points.

kopt [int] Index of the interpolation point around which the quadratic function is defined. The constant term of the quadratic function is not maintained, and zero is returned at `xpt[kopt, :]`.

stack_level [int, optional] Stack level of the warning (the default is 2).

Warns

RuntimeWarning The evaluations of the quadratic function do not satisfy the interpolation conditions up to a certain tolerance.

1.5.3 cobyqa.optimize.Quadratic.curv

`Quadratic.`**`curv`**(*x*, *xpt*)

Evaluate the curvature of the quadratic function.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the curvature of the quadratic function is to be evaluated.

xpt [numpy.ndarray, shape (npt, n)] Interpolation points that define the quadratic function. Each row of *xpt* stores the coordinates of an interpolation point.

Returns

float Curvature of the quadratic function at *x*.

Notes

Although the value can be recovered using `hessp`, the evaluation of this method improves the computational efficiency.

1.5.4 cobyqa.optimize.Quadratic.grad

`Quadratic.grad(x, xpt, kopt)`

Evaluate the gradient of the quadratic function.

Parameters

x [numpy.ndarray, shape (n,)] Point at which the gradient of the quadratic function is to be evaluated.

xpt [numpy.ndarray, shape (npt, n)] Interpolation points that define the quadratic function. Each row of *xpt* stores the coordinates of an interpolation point.

kopt [int] Index of the interpolation point around which the quadratic function is defined. The constant term of the quadratic function is not maintained, and zero is returned at `xpt[kopt, :]`.

Returns

numpy.ndarray, shape (n,) Value of the gradient of the quadratic function at *x*.

1.5.5 cobyqa.optimize.Quadratic.hess

`Quadratic.hess(xpt)`

Evaluate the Hessian matrix of the quadratic function.

Parameters

xpt [numpy.ndarray, shape (npt, n)] Interpolation points that define the quadratic function. Each row of *xpt* stores the coordinates of an interpolation point.

Returns

numpy.ndarray, shape (n, n) Hessian matrix of the quadratic function.

Notes

The Hessian matrix of the model is not explicitly stored and its computation requires a matrix multiplication. If only products of the Hessian matrix of the model with any vector are required, consider using instead `hessp`.

1.5.6 cobyqa.optimize.Quadratic.hessp

`Quadratic.hessp(x, xpt)`

Evaluate the product of the Hessian matrix of the quadratic function with any vector.

Parameters

x [numpy.ndarray, shape (n,)] Vector to be left-multiplied by the Hessian matrix of the quadratic function.

xpt [numpy.ndarray, shape (npt, n)] Interpolation points that define the quadratic function. Each row of *xpt* stores the coordinates of an interpolation point.

Returns

numpy.ndarray, shape (n,) Value of the product of the Hessian matrix of the quadratic function with the vector *x*.

1.5.7 cobyqa.optimize.Quadratic.shift_expansion_point

`Quadratic.shift_expansion_point(step, xpt)`

Shift the point around which the quadratic function is defined.

This method must be called when the index around which the quadratic function is defined is modified, or when the point in *xpt* around which the quadratic function is defined is modified.

Parameters

step [numpy.ndarray, shape (n,)] Displacement from the current point `xopt` around which the quadratic function is defined. After calling this method, the value of the quadratic function at `xopt + step` is zero, since the constant term of the function is not maintained.

xpt [numpy.ndarray, shape (npt, n)] Interpolation points that define the quadratic function. Each row of *xpt* stores the coordinates of an interpolation point.

1.5.8 cobyqa.optimize.Quadratic.shift_interpolation_points

`Quadratic.shift_interpolation_points(xpt, kopt)`

Update the components of the quadratic function when the origin from which the interpolation points are defined is to be displaced.

Parameters

xpt [numpy.ndarray, shape (npt, n)] Interpolation points that define the quadratic function. Each row of *xpt* stores the coordinates of an interpolation point.

kopt [int] Index of the interpolation point around which the quadratic function is defined. The constant term of the quadratic function is not maintained, and zero is returned at `xpt[kopt, :]`.

Notes

Given `xbase` the previous origin of the calculations, it is assumed that the origin is shifted to `xbase + xpt[kopt, :]`.

1.5.9 cobyqa.optimize.Quadratic.update

`Quadratic.update`(*xpt*, *kopt*, *xold*, *bmat*, *zmat*, *idz*, *knew*, *diff*)

Update the model when a point of the interpolation set is modified.

Parameters

xpt [numpy.ndarray, shape (npt, n)] Interpolation points that define the quadratic function. Each row of *xpt* stores the coordinates of an interpolation point.

kopt [int] Index of the interpolation point around which the quadratic function is defined. The constant term of the quadratic function is not maintained, and zero is returned at `xpt[kopt, :]`.

xold [numpy.ndarray, shape (n,)] Previous point around which the quadratic function was defined.

bmat [numpy.ndarray, shape (npt + n, n)] Last `n` columns of the inverse KKT matrix of interpolation.

zmat [numpy.ndarray, shape (npt, npt - n - 1)] Rank factorization matrix of the leading `npt` submatrix of the inverse KKT matrix of interpolation.

idz [int] Number of nonpositive eigenvalues of the leading `npt` submatrix of the inverse KKT matrix of interpolation. Although its theoretical value is always 0, it is designed to tackle numerical difficulties caused by ill-conditioned problems.

knew [int] Index of the interpolation point that is modified.

diff [float] Difference between the evaluation of the previous model and the expected value at `xpt[kopt, :]`.

CHAPTER TWO

LINEAR ALGEBRA

linalg.bvcs(xpt, kopt, gq, curv, args, xl, ...)	Evaluate Cauchy step on the absolute value of a Lagrange polynomial, subject to bound constraints on its coordinates and its length.
linalg.bvlag(xpt, kopt, klag, gq, xl, xu, ...)	Estimate a point that maximizes a lower bound on the denominator of the updating formula, subject to bound constraints on its coordinates and its length.
linalg.bvtcg(xopt, gq, hessp, args, xl, xu, ...)	Minimize approximately a quadratic function subject to bound and trust-region constraints using a truncated conjugate gradient.
linalg.cpqp(xopt, Aub, bub, Aeq, beq, xl, ...)	Minimize approximately a convex piecewise quadratic function subject to bound and trust-region constraints using a truncated conjugate gradient.
linalg.givens(M, cval, sval, i, j, axis[, ...])	Perform a Givens rotation.
linalg.lctcg(xopt, gq, hessp, args, Aub, ...)	Minimize approximately a quadratic function subject to bound, linear, and trust-region constraints using a truncated conjugate gradient.
linalg.nnls(A, b[, k, maxiter])	Compute the least-squares solution of `A @ x = b` subject to the nonnegativity constraints `x[:k] >= 0`.
linalg.qr(a[, overwrite_a, pivoting, ...])	Compute the QR factorization `a = Q @ R` where `Q` is an orthogonal matrix and `R` is an upper triangular matrix.

2.1 cobyqa.linalg.bvcs

`cobyqa.linalg.`**`bvcs`**(*xpt*, *kopt*, *gq*, *curv*, *args*, *xl*, *xu*, *delta*, ***kwargs*)

Evaluate Cauchy step on the absolute value of a Lagrange polynomial, subject to bound constraints on its coordinates and its length.

Parameters

xpt [numpy.ndarray, shape (npt, n)] Set of points. Each row of *xpt* stores the coordinates of a point.

kopt [int] Index of the point from which the Cauchy step is evaluated.

gq [array_like, shape (n,)] Gradient of the Lagrange polynomial of the points in *xpt* (not necessarily the *kopt*-th one) at `xpt[kopt, :]`.

curv [callable] Function providing the curvature of the Lagrange polynomial.

```
curv(x, *args) -> float
```

where `x` is an array with shape (n,) and `args` is the tuple of fixed parameters needed to specify the function.

args [tuple] Parameters to forward to the curvature function.

xl [array_like, shape (n,)] Lower-bound constraints on the decision variables. Use `-numpy.inf` to disable the bounds on some variables.

xu [array_like, shape (n,)] Upper-bound constraints on the decision variables. Use `numpy.inf` to disable the bounds on some variables.

delta [float] Upper bound on the length of the Cauchy step.

Returns

step [numpy.ndarray, shape (n,)] Cauchy step.

cauchy [float] Square of the Lagrange polynomial evaluation at the Cauchy point.

Other Parameters

bdtol [float, optional] Tolerance for comparisons on the bound constraints (the default is `10 * eps * n * max(1, max(abs(xl)), max(abs(xu)))`.

Raises

AssertionError The vector `xpt[kopt, :]` is not feasible.

Notes

The method is adapted from the ALTMOV algorithm [1], and the vector `xpt[kopt, :]` must be feasible.

References

[1]

2.2 cobyqa.linalg.bvlag

`cobyqa.linalg.`**`bvlag`**(*xpt*, *kopt*, *klag*, *gq*, *xl*, *xu*, *delta*, *alpha*, ***kwargs*)

Estimate a point that maximizes a lower bound on the denominator of the updating formula, subject to bound constraints on its coordinates and its length.

Parameters

xpt [numpy.ndarray, shape (npt, n)] Set of points. Each row of *xpt* stores the coordinates of a point.

kopt [int] Index of a point in *xpt*. The estimated point will lie on a line joining `xpt[kopt, :]` to another point in *xpt*.

klag [int] Index of the point in *xpt*.

gq [array_like, shape (n,)] Gradient of the *klag*-th Lagrange polynomial at `xpt[kopt, :]`.

xl [array_like, shape (n,)] Lower-bound constraints on the decision variables. Use `-numpy.inf` to disable the bounds on some variables.

xu [array_like, shape (n,)] Upper-bound constraints on the decision variables. Use `numpy.inf` to disable the bounds on some variables.

delta [float] Upper bound on the length of the step.

alpha [float] Real parameter.

Returns

step [numpy.ndarray, shape (n,)] Step from `xpt[kopt, :]` towards the estimated point.

Other Parameters

bdtol [float, optional] Tolerance for comparisons on the bound constraints (the default is `10 * eps * n * max(1, max(abs(xl)), max(abs(xu)))`.

Raises

AssertionError The vector `xpt[kopt, :]` is not feasible.

Notes

The denominator of the updating formula is given in Equation (3.9) of [1], and the parameter *alpha* is the referred in Equation (4.12) of [2].

References

[1], [2]

2.3 cobyqa.linalg.bvtcg

`cobyqa.linalg.`**`bvtcg`**(*xopt*, *gq*, *hessp*, *args*, *xl*, *xu*, *delta*, ***kwargs*)

Minimize approximately a quadratic function subject to bound and trust-region constraints using a truncated conjugate gradient.

Parameters

xopt [numpy.ndarray, shape (n,)] Point around which the Taylor expansions of the quadratic function is defined.

gq [array_like, shape (n,)] Gradient of the quadratic function at *xopt*.

hessp [callable] Function providing the product of the Hessian matrix of the quadratic function with any vector.

```
hessp(x, *args) -> array_like, shape(n,)
```

where `x` is an array with shape (n,) and *args* is a tuple of parameters to forward to the objective function. It is assumed that the Hessian matrix implicitly defined by *hessp* is symmetric, but not necessarily positive semidefinite.

args [tuple] Parameters to forward to the Hessian product function.

xl [array_like, shape (n,)] Lower-bound constraints on the decision variables. Use `-numpy.inf` to disable the bounds on some variables.

xu [array_like, shape (n,)] Upper-bound constraints on the decision variables. Use `numpy.inf` to disable the bounds on some variables.

delta [float] Upper bound on the length of the step from *xopt*.

Returns

step [numpy.ndarray, shape (n,)] Step from *xopt* towards the estimated point.

Other Parameters

bdtol [float, optional] Tolerance for comparisons on the bound constraints (the default is `10 * eps * n * max(1, max(abs(xl)), max(abs(xu)))`.

Raises

AssertionError The vector *xopt* is not feasible.

Notes

The method is adapted from the TRSBOX algorithm [1].

References

[1]

2.4 cobyqa.linalg.cpqp

`cobyqa.linalg.`**`cpqp`**(*xopt*, *Aub*, *bub*, *Aeq*, *beq*, *xl*, *xu*, *delta*, ***kwargs*)

Minimize approximately a convex piecewise quadratic function subject to bound and trust-region constraints using a truncated conjugate gradient.

The method minimizes the function

$$\frac{1}{2}(\|[\mathtt{Aub} \times x - \mathtt{bub}]_+\|_2^2 + \|\mathtt{Aeq} \times x - \mathtt{beq}\|_2^2),$$

where $[\cdot]_+$ denotes the componentwise positive part operator.

Parameters

xopt [numpy.ndarray, shape (n,)] Center of the trust-region constraint.

Aub [array_like, shape (mlub, n)] Matrix *Aub* as shown above.

bub [array_like, shape (mlub,)] Vector *bub* as shown above.

Aeq [array_like, shape (mleq, n)] Matrix *Aeq* as shown above.

beq [array_like, shape (meq,)] Vector *beq* as shown above.

xl [array_like, shape (n,)] Lower-bound constraints on the decision variables. Use `-numpy.inf` to disable the bounds on some variables.

xu [array_like, shape (n,)] Upper-bound constraints on the decision variables. Use `numpy.inf` to disable the bounds on some variables.

delta [float] Upper bound on the length of the step from *xopt*.

Returns

step [numpy.ndarray, shape (n,)] Step from *xopt* towards the estimated point.

Other Parameters

bdtol [float, optional] Tolerance for comparisons on the bound constraints (the default is `10 * eps * n * max(1, max(abs(xl)), max(abs(xu)))`.

Raises

AssertionError The vector *xopt* is not feasible.

Notes

The method is adapted from the TRSTEP algorithm [1]. To cope with the convex piecewise quadratic objective function, the method minimizes

$$\frac{1}{2}(\|\mathtt{Aeq} \times x - \mathtt{beq}\|_2^2 + \|y\|_2^2)$$

subject to the original constraints, where the slack variable y is lower bounded by zero and $\mathtt{Aub} \times x - \mathtt{bub}$.

References

[1]

2.5 cobyqa.linalg.givens

`cobyqa.linalg.givens`(*M*, *cval*, *sval*, *i*, *j*, *axis*, *slicing=None*)

Perform a Givens rotation.

Parameters

M [numpy.ndarray] Matrix on which the Givens rotation is performed in-place.

cval [float] Multiple of the cosine value of the angle of rotation.

sval [float] Multiple of the sine value of the angle of rotation.

i [int] First index of the Givens rotation procedure.

j [int] Second index of the Givens rotation procedure.

axis [int] Axis over which to select values. If *M* is a matrix with two dimensions, the calculations will be applied to the rows by setting `axis = 0` and to the columns by setting `axis = 1`.

slicing [slice, optional] Part of the data at which the Givens rotation should be applied. Default applies it to to all the components.

Returns

hval [float] Length of the two-dimensional vector of components *cval* and *sval*, given by $\sqrt{\mathtt{cval}^2 + \mathtt{sval}^2}$.

2.6 cobyqa.linalg.lctcg

`cobyqa.linalg.lctcg`(*xopt*, *gq*, *hessp*, *args*, *Aub*, *bub*, *Aeq*, *beq*, *xl*, *xu*, *delta*, ***kwargs*)

Minimize approximately a quadratic function subject to bound, linear, and trust-region constraints using a truncated conjugate gradient.

Parameters

xopt [numpy.ndarray, shape (n,)] Point around which the Taylor expansions of the quadratic function is defined.

gq [array_like, shape (n,)] Gradient of the quadratic function at *xopt*.

hessp [callable] Function providing the product of the Hessian matrix of the quadratic function with any vector.

```
hessp(x, *args) -> array_like, shape(n,)
```

where `x` is an array with shape (n,) and *args* is a tuple of parameters to forward to the objective function. It is assumed that the Hessian matrix implicitly defined by *hessp* is symmetric, but not necessarily positive semidefinite.

args [tuple] Parameters to forward to the Hessian product function.

Aub [array_like, shape (mlub, n), optional] Jacobian matrix of the linear inequality constraints. Each row of *Aub* stores the gradient of a linear inequality constraint.

bub [array_like, shape (mlub,), optional] Right-hand side vector of the linear inequality constraints `Aub @ x <= bub`, where `x` has the same size than *xopt*.

Aeq [array_like, shape (mleq, n), optional] Jacobian matrix of the linear equality constraints. Each row of *Aeq* stores the gradient of a linear equality constraint.

beq [array_like, shape (mleq,), optional] Right-hand side vector of the linear equality constraints *Aeq @ x = beq*, where `x` has the same size than *xopt*.

xl [array_like, shape (n,)] Lower-bound constraints on the decision variables. Use `-numpy.inf` to disable the bounds on some variables.

xu [array_like, shape (n,)] Upper-bound constraints on the decision variables. Use `numpy.inf` to disable the bounds on some variables.

delta [float] Upper bound on the length of the step from *xopt*.

Returns

step [numpy.ndarray, shape (n,)] Step from *xopt* towards the estimated point.

Other Parameters

bdtol [float, optional] Tolerance for comparisons on the bound constraints (the default is `10 * eps * n * max(1, max(abs(xl)), max(abs(xu)))`.

lctol [float, optional] Tolerance for comparisons on the linear constraints (the default is `10 * eps * n * max(1, max(abs(bub))))`.

Raises

AssertionError The vector *xopt* is not feasible.

Notes

The method is adapted from the TRSTEP algorithm [1]. It is an active-set variation of the truncated conjugate gradient method, which maintains the QR factorization of the matrix whose columns are the gradients of the active constraints. The linear equality constraints are then handled by considering them are always active.

References

[1]

2.7 cobyqa.linalg.nnls

`cobyqa.linalg.`**`nnls`**(*A*, *b*, *k=None*, *maxiter=None*, ***kwargs*)

Compute the least-squares solution of `A @ x = b` subject to the nonnegativity constraints `x[:k] >= 0`.

Parameters

A [array_like, shape (m, n)] Matrix *A* as shown above.

b [array_like, shape (m,)] Right-hand side vector *b* as shown above.

k [int, optional] Number of nonnegativity constraints. The first *k* components of the solution vector are nonnegative (the default is `A.shape[1]`).

maxiter [int, optional] Maximum number of inner iterations (the default is `3 * A.shape[1]`).

Returns

x [numpy.ndarray, shape (n,)] Solution vector `x` as shown above.

rnorm [float] Residual at the solution.

Other Parameters

lstol [float, optional] Tolerance on the approximate KKT conditions for the calculations of the least-squares Lagrange multipliers (the default is `10 * eps * max(n, m) * max(1, max(abs(b)))`).

Notes

The method is adapted from the NNLS algorithm [1].

References

[1]

2.8 cobyqa.linalg.qr

`cobyqa.linalg.`**`qr`**(*a*, *overwrite_a=False*, *pivoting=False*, *check_finite=True*)

Compute the QR factorization `a = Q @ R` where `Q` is an orthogonal matrix and `R` is an upper triangular matrix.

Parameters

a [array_like, shape (m, n)] Matrix to be factorized.

overwrite_a [bool, optional] Whether to overwrite the data in *a* with the matrix `R` (may improve the performance by limiting the memory cost).

pivoting [bool, optional] Whether the factorization should include column pivoting, in which case a permutation vector `P` is returned such that `A[:, P] = Q @ R`.

check_finite [bool, optional] Whether to check that the input matrix contains only finite numbers.

Returns

Q [numpy.ndarray, shape (m, m)] Above-mentioned orthogonal matrix `Q`.

R [numpy.ndarray, shape (m, n)] Above-mentioned upper triangular matrix `R`.

P [numpy.ndarray, shape (n,)] Indices of the permutations. Not returned if `pivoting=False`.

Raises

AssertionError The matrix *a* is not two-dimensional.

CHAPTER

THREE

TEST SUPPORT

`tests.assert_array_less_equal`(x, y[, ...])	Raise an AssertionError if two objects are not less-or-equal-ordered.
`tests.assert_dtype_equal`(actual, desired)	Compare the data type of two arrays.

3.1 cobyqa.tests.assert_array_less_equal

cobyqa.tests.**assert_array_less_equal**(*x*, *y*, *err_msg=''*, *verbose=True*)

Raise an AssertionError if two objects are not less-or-equal-ordered.

Parameters

x [array_like] Smaller object to check.

y [array_like] Larger object to compare.

err_msg [str, optional] Error message to be printed in case of failure.

verbose [bool, optional] Whether the conflicting values are appended to the error message (default is True).

Raises

AssertionError The two arrays are not less-or-equal-ordered.

3.2 cobyqa.tests.assert_dtype_equal

cobyqa.tests.**assert_dtype_equal**(*actual*, *desired*)

Compare the data type of two arrays.

Parameters

actual [array_like or type] Array obtained.

desired [array_like or type] Array desired.

Raises

AssertionError The two arrays do not share the same data type.

CHAPTER

FOUR

REPORTING BUGS

To report a bug, request a new feature, or make contributions (e.g., code patches), please open a new issue on GitHub: https://github.com/ragonneau/cobyqa/issues.

CHAPTER

FIVE

COBYQA LICENSE

BIBLIOGRAPHY

[1] M. J. D. Powell. “On updating the inverse of a KKT matrix.” In: Numerical Linear Algebra and Optimization. Ed. by Y. Yuan. Beijing, CN: Science Press, 2004, pp. 56–78.

[1] M. J. D. Powell. “A direct search optimization method that models the objective and constraint functions by linear interpolation.” In: Advances in Optimization and Numerical Analysis. Ed. by S. Gomez and J. P. Hennart. Dordrecht, NL: Springer, 1994, pp. 51–67.

[1] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Trust-Region Methods. MPS-SIAM Ser. Optim. Philadelphia, PA, US: SIAM, 2009.

[1] M. J. D. Powell. “Least Frobenius norm updating of quadratic models that satisfy interpolation conditions.” In: Math. Program. 100 (2004), pp. 183–215.

[1] M. J. D. Powell. “On updating the inverse of a KKT matrix.” In: Numerical Linear Algebra and Optimization. Ed. by Y. Yuan. Beijing, CN: Science Press, 2004, pp. 56–78.

[1] M. J. D. Powell. “The NEWUOA software for unconstrained optimization without derivatives.” In: Large-Scale Nonlinear Optimization. Ed. by G. Di Pillo and M. Roma. New York, NY, US: Springer, 2006, pp. 255–297.

[1] M. J. D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Tech. rep. DAMTP 2009/NA06. Cambridge, UK: Department of Applied Mathematics and Theoretical Physics, University of Cambridge, 2009.

[1] M. J. D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Tech. rep. DAMTP 2009/NA06. Cambridge, UK: Department of Applied Mathematics and Theoretical Physics, University of Cambridge, 2009.

[2] M. J. D. Powell. “The NEWUOA software for unconstrained optimization without derivatives.” In: Large-Scale Nonlinear Optimization. Ed. by G. Di Pillo and M. Roma. New York, NY, US: Springer, 2006, pp. 255-–297.

[1] M. J. D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Tech. rep. DAMTP 2009/NA06. Cambridge, UK: Department of Applied Mathematics and Theoretical Physics, University of Cambridge, 2009.

[1] M. J. D. Powell. “On fast trust region methods for quadratic models with linear constraints.” In: Math. Program. Comput. 7 (2015), pp. 237-–267.

[1] M. J. D. Powell. “On fast trust region methods for quadratic models with linear constraints.” In: Math. Program. Comput. 7 (2015), pp. 237–267.

[1] C. L. Lawson and R. J. Hanson. Solving Least Squares Problems. Classics Appl. Math. Philadelphia, PA, US: SIAM, 1974.

INDEX

Non-alphabetical

A

B

C

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V