UOpt Optimization - User manual

AI & IT UG (haftungsbeschränkt)

July 16, 2023

Contents

1 Introduction

The UOpt optimizer is a solver for unconstrained optimization (minimization) problems. It can make use of several method to compute search directions that can drive the objective function down given a certain starting point `x0`. These include the standard methods *Newton-Raphson*, *conjugate gradient* and *steepest descend*. Two further methods modify the *Newton-Raphson* method by applying regularization to enforce a positive definite Hessian.

To work the UOpt optimizer must be provided with the gradient of the objective function, and when Newton-type search directions are to be used, with the Hessian of the objective as well.

For large-scale optimizations, UOpt can compute a regularized Newton method search direction also with a so-called matrix free method, which means that Hessian-vector products instead of the full Hessian are used. The search direction is computed using an inner CG iteration to solve the linear system of Hessian and negative gradient that is required for the Newton method.

The regularization in the Newton methods is controlled by determining the eigenvalues of the Hessian and adjusted to cause just the minimal distortion possible to the original system.

When using the matrix-free method the largest and smallest eigenvalues of the system are guessed by default, but they can also optionally be determined exactly, either one of them or both.

The UOpt optimizer is a shared library written in C++ and can be called from other programs using the native C interface. Wrappers and modules are provided for the MATLAB, Python and R languages.

UOpt uses the OpenBLAS library to perform matrix arithmetic and for the matrix decompositions involved.

2 Basic usage

Uopt consists of one optimization routine that can be called from one of the langauge interfaces. Callback functions must be provided for the following computes tasks:

f The objective function

g The derivative

H The Hessian

Hv The Hessian-vector product

s The status callback

The status callback **s** is called once per iteration. It has a default value that prints console messages.

H must be provided when `sdirs` contains bit 0x10.

H or **Hv** must be provided when `sdirs` contains bit 0x8.

3 C/C++ interface

4 MATLAB interface

4.1 Installation

The UOpt MATLAB interface consists of a single MATLAB Extension (MEX) file written in C++. This file can be found in the directory `$UOPT_HOME/share/uopt/MATLAB`, along with some helper and example scripts.

The MEX file that must be compiled with the **mex** command in MATLAB. Assuming UOpt is installed to `/usr/local`, the required **mex** command is

```
mex -I/usr/local/include -L/usr/local/lib src/uopt.cc -luopt
```

The script file `build_uopt.m` which is distributed with UOpt contains this command, so you may also use

```
build_uopt
```

The compiled MEX function **uopt** must be added to the **path** so that MATLAB can find it. This can be checked with

```
where uopt
```

4.2 Usage

UOpt is available in MATLAB through the single function **uopt**. The function takes seven arguments:

x0 starting point for optimization, vector of N-values

f handle to the objective function, which must return a scalar

g handle to the gradient function, which must return two values, the objective, a scalar, and the gradient, a vector of N-values

H handle to the Hessian function, or an empty array. The Hessian function must return three values, the objective, a scalar, the gradient, a vector of N-values, and the Hessian, a N-by-N matrix

Hv handle to the Hessian-vector product function, or an empty array. The Hessian-vector product function is called with two arguments, the current point `x` and a N-by-P matrix `dx`. It must return three values, the objective, a scalar, the gradient, a vector of N-values, and the Hessian-vector product, a N-by-P matrix

s handle to the status function. It must return a scalar, where a non-zero value will stop the optimization. It is called with two arguments, the current point `x` and a struct `s` with status information

opts a struct with options for the optimizer

4.3 Examples

4.3.1 The two-dimensional Rosenbrock function

The two-dimensional Rosenbrock function is defined in the function **ffunc**

```
function r = ffunc(x)
  a = 1;
  b = 100;
  r = (a - x(1)).^2 + b * (x(2) - x(1).^2).^2;
```

For the gradient, we define the function **gfunc** and use the **admDiffRev** reverse mode driver from ADiMat (`https://www.adimat.de`).

```
function [r g] = gfunc(x)
  [g r] = admDiffRev(@ffunc, 1, x);
```

For the Hessian, we define the function **hfunc** and use the **admHessian** forward-over-reverse mode driver from ADiMat.

```
function [r g H] = hfunc(x)
  [H g r] = admHessRev(@ffunc, 1, x);
```

The output function **sfunc** prints the status fields to the console.

```
function res = sfunc(x, s)
  fns = fieldnames(s);
  for i=1:length(fns)
    fprintf('%s=%s ', fns{i}, s.(fns{i}));
  end
  fprintf('\n');
  res = 0;
```

Finally, we can set up the starting point and the options for UOpt and start the optimization:

```
N = 170;
x0 = rand(1, N) - 0.5

opts = struct('maxit', N*70, 'sdirs', 4);

xs = uopt(x0, @ffuncN, @gfuncN, @hfuncN, [], @sfunc, opts).'
```

Here, we pass the Hessian handle as the fourth argument and set the Hessian-vector handle to be empty. The option `sdirs` is set to 4 to enable the search direction SD2 which is the standard Newton method.

4.3.2 The N-dimensional Rosenbrock function with full Hessian

The N-dimensional Rosenbrock function is defined in the function **ffuncN**

```
function r = ffuncN(x)
  r = sum((1 - x(1:end-1)).^2 + 100 * (x(2:end) - x(1:end-1).^2).^2);
```

For the gradient, we define the function **gfuncN** and use the **admDiffRev** reverse mode driver from ADiMat (`https://www.adimat.de`).

```
function [r g] = gfuncN(x)
  [g r] = admDiffRev(@ffuncN, 1, x, admOptions('functionResults', 1));
```

For the Hessian, we define the function **hfuncN** and use the **admHessian** forward-over-reverse mode driver from ADiMat.

```
function [r g H] = hfuncN(x)
  [H g r] = admHessRev(@ffuncN, 1, x, admOptions('functionResults', 1));
```

The output function **sfunc** prints the status fields to the console.

```
function res = sfunc(x, s)
  fns = fieldnames(s);
  for i=1:length(fns)
    fprintf('%s=%s ', fns{i}, s.(fns{i}));
  end
  fprintf('\n');
  res = 0;
```

Finally, we can set up the starting point and the options for UOpt and start the optimization:

```
N = 170;
x0 = rand(1, N) - 0.5

opts = struct('maxit', N*70, 'sdirs', 4);

xs = uopt(x0, @ffuncN, @gfuncN, @hfuncN, [], @sfunc, opts).'
```

Here, we pass the Hessian handle as the fourth argument and set the Hessian-vector handle to be empty. The option `sdirs` is set to 4 to enable the search direction SD2 which is the standard Newton method.

4.3.3 The N-dimensional Rosenbrock function with Hessian-vector products

The N-dimensional Rosenbrock function is defined in the function **ffuncN**

```
function r = ffuncN(x)
  r = sum((1 - x(1:end-1)).^2 + 100 * (x(2:end) - x(1:end-1).^2).^2);
```

For the gradient, we define the function **gfuncN** and use the **admDiffRev** reverse mode driver from ADiMat (`https://www.adimat.de`).

```
function [r g] = gfuncN(x)
  [g r] = admDiffRev(@ffuncN, 1, x, admOptions('functionResults', 1));
```

For the Hessian-vector product, we define the function **hfuncN** and use the **admHessian** forward-over-reverse mode driver from ADiMat.

```
function [r g H] = hfuncN(x)
  [H g r] = admHessRev(@ffuncN, 1, x, admOptions('functionResults', 1));
```

The output function **sfunc** prints the status fields to the console.

```
function res = sfunc(x, s)
  fns = fieldnames(s);
  for i=1:length(fns)
    fprintf('%s=%s ', fns{i}, s.(fns{i}));
  end
  fprintf('\n');
  res = 0;
```

Finally, we can set up the starting point and the options for UOpt and start the optimization:

```
N = 170;
x0 = rand(1, N) - 0.5

opts = struct('maxit', N*70, 'sdirs', 8);

xs = uopt(x0, @ffuncN, @gfuncN, [], @hvfuncN, @sfunc, opts).'
```

Here, we set the Hessian handle to be empty and pass the Hessian-vector handle. The option `sdirs` is set to 8 to enable the search direction SD3 which is the only one to make use of Hessian-vector products.

5 Python interface

5.1 Installation

```
python3 -m pip install pysrc/dist/uopt-1.2.0.181-py3-none-any.whl
```

5.2 Usage

The python package is called `uopt` with main module `uopt`. The functions `uopt` and `usolve` are available in the package. Both are implemented in the module `uopt`, mapping the calls from the library to function calls using Numpy arrays to pass the data.

5.3 UOpt Example

As an example consider the 2D Rosenbrock function as defined in the module rosenbrock2d.py:

```
import numpy as np


a = 2;
b = 100;


def fhandle(x, J, userdata):

    # Rosenbrock function:
    # (a - x[[1]])^2 + b * (x[[2]] - x[[1]]^2)^2

    r = (a - x[0])**2 + b * (x[1] - x[0]**2)**2;

    J[0] = r
```

```
    return 0


def ghandle(x, J, g, userdata):

    r = (a - x[0])**2 + b * (x[1] - x[0]**2)**2;

    # first derivative of Rosenbrock function:
    ## (a - x1)^2 + b * (x2 - x1^2)^2
    ## [-2 * (a - x1) - b * 2 * (x2 - x1^2) * 2 * x1, b * 2 * (x2 - x1^2)]

    dr1 = -2 * (a - x[0]) - b * 2 * (x[1] - x[0]**2) * 2 * x[0];
    dr2 = b * 2 * (x[1] - x[0]**2);

    J[0] = r

    g[0] = dr1
    g[1] = dr2
    return 0


def Hhandle(x, J, g, H, userdata):

    r = (a - x[0])**2 + b * (x[1] - x[0]**2)**2;

    dr1 = -2 * (a - x[0]) - b * 2 * (x[1] - x[0]**2) * 2 * x[0];
    dr2 = b * 2 * (x[1] - x[0]**2);

    # second derivative of Rosenbrock function:
    ## (a - x1)^2 + b * (x2 - x1^2)^2
    ## [-2 * (a - x1) - b * 2 * (x2 - x1^2) * 2 * x1, b * 2 * (x2 - x1^2)]

    dr1_dr1 = 2 + b * 2 * 2 * x[0] * 2 * x[0] - b * 2 * (x[1] - x[0]**2) * 2;
    dr1_dr2 = - b * 2 * 2 * x[0];
    dr2_dr1 = -b * 2 * 2 * x[0];
    dr2_dr2 = b * 2;

    J[0] = r

    g[0] = dr1
    g[1] = dr2

    H[0][0] = dr1_dr1
    H[0][1] = dr1_dr2

    H[1][0] = dr2_dr1
    H[1][1] = dr2_dr2

    return 0


def Hvhandle(x, J, dx, g, Hv, userdata):

    n = np.size(x)
    ndx = dx.shape[1]

    r = (a - x[0])**2 + b * (x[1] - x[0]**2)**2;
```

```
    dr1 = -2 * (a - x[0]) - b * 2 * (x[1] - x[0]**2) * 2 * x[0];
    dr2 = b * 2 * (x[1] - x[0]**2);

    # second derivative of Rosenbrock function:
    ## (a - x1)^2 + b * (x2 - x1^2)^2
    ## [-2 * (a - x1) - b * 2 * (x2 - x1^2) * 2 * x1, b * 2 * (x2 - x1^2)]

    dr1_dr1 = 2 + b * 2 * 2 * x[0] * 2 * x[0] - b * 2 * (x[1] - x[0]**2) * 2;
    dr1_dr2 = - b * 2 * 2 * x[0];
    dr2_dr1 = -b * 2 * 2 * x[0];
    dr2_dr2 = b * 2;

    J[0] = r

    g[0] = dr1
    g[1] = dr2

    H = np.zeros((2,2))
    H[0][0] = dr1_dr1;
    H[0][1] = dr1_dr2;

    H[1][0] = dr2_dr1;
    H[1][1] = dr2_dr2;

    Hv[:] = np.matmul(H, dx)

    return 0


def ohandle(x, y, status, userdata):
    print(x, y)
    print(status)
    return 0
```

The Newton optimization method is invoked as follows:

```
import uopt
import numpy as np

from rosenbrock2d import *

n = 2

x0 = np.ndarray((n,))
x0[:] = [1.5,1.1]
xs = np.ndarray((n,))
ys = np.ndarray((1,))
opts = {'a': 1, 'b': 2, 'c': 3, 'wolcheck': 3}
udata = {'xxx': 123}

rc = uopt.uopt(x0, xs, ys, fhandle, ghandle, Hhandle, None, None, ohandle, opts, udata)

print('uopt status code:', rc)
print('uopt xs:', xs)
print('uopt ys:', ys)
print('expected:', np.array((a, a**2)))
```

```
exit(rc)
```

The matrix-free Newton method is invoked as follows:

```
import uopt
import numpy as np

from rosenbrock2d import *

n = 2

x0 = np.ndarray((n,))
x0[:] = [1.5,1.1]
xs = np.ndarray((n,))
ys = np.ndarray((1,))
opts = {'sdirs': 5, 'wolcheck': 3}
udata = {'xxx': 123}

rc = uopt.uopt(x0, xs, ys, fhandle, ghandle, None, Hvhandle, None, ohandle, opts, udata)

print('uopt status code:', rc)
print('uopt xs:', xs)
print('expected:', np.array((a, a**2)))

exit(rc)
```

5.4 USolve Example

As an example consider the N-dimensional Rosenbrock function as defined in the module rosenN.py:

```
import numpy as np

np.set_printoptions(linewidth=125)

def rosenNs(x):
    N = x.size
    return sum((1 - x[0:N-1])**2 + 100 * (x[1:N] - x[0:N-1]**2)**2)

def rosenN(x):
    N = x.size
    rterms1 = (1 - x[0:N-1])
    rterms2 = 10 * (x[1:N] - x[0:N-1]**2)
    r = np.hstack([rterms1, rterms2])
    return r

def rosenNObj(x, y, *args):
    y[:] = rosenN(x)

def rosenNJac(x, y, J, *args):
    rosenNObj(x, y)
    N = x.size
    yblk = N-1
    # rterms1 = (1 - x[0:N-1])
    J[0:yblk,:] = -np.eye(yblk, x.size)
    # rterms2 = 10 * (x[1:N] - x[0:N-1]**2)
    J[yblk:2*yblk,:] = np.diag(np.full(yblk,10), 1)[0:yblk,:]
```

```
    J[yblk:2*yblk,:] += np.diag(10 * -2 * x)[0:yblk,:]

def rosenNJacv(x, y, v, Jv, *args):
    rosenNObj(x, y)
    N = x.size
    yblk = N-1
    x = x.reshape(N)
    v = v.reshape(N)
    # rterms1 = (1 - x[0:N-1])
    Jv[0:yblk,0] = -1 * v[0:yblk]
    # rterms2 = 10 * (x[1:N] - x[0:N-1]**2)
    Jv[yblk:2*yblk,0] = 10 * (v[1:N] - 2 * x[0:yblk]*v[0:yblk])
    # J = np.ndarray((y.size, x.size), order='F')
    # rosenNJac(x, y, J)
    # Jv[:] = np.matmul(J, v)

def rosenNJacTv(x, y, v, Jv, *args):
    J = np.ndarray((y.size, x.size), order='F')
    rosenNJac(x, y, J)
    Jv[:] = np.matmul(v, J)

def rosenOutput(x, y, status, *args):
    maxLen = max(14, max([len(f)+1+len(status[f]) for f in status]))
    print("".join([("%%-%ds" % (maxLen+2,)) %
                    ("%s=%s" % (f, status[f])) for f in status]))
```

Note the main objective function `rosenN` returns a (N-1)-by-2 array. The usual result is obtained by computing the vector norm of these values. So we solve the Rosenbroock function by driving all the terms in rosenN() to zero with `uopt.usolve`. The normal Rosenbrock function is included in the listing as a reference as `rosenNs`.

The functions `rosenNJac`, `rosenNJacv`, and `rosenNJacTv` implement the derivatives of rosenN: the full Jacobian, Jacobian-vector product and Vector-Jacobian product. The details are not trivial in part, but it all follows from the rules of differentiation aplied to `rosenN`.

Note that the function results and the derivatives must be copied into the Numpy arrays provided to the callback functions, and they need not return a result. As with `uopt`, the output function can return a non-zero integer to stop the iteration.

The `usolve` optimization is then run as shown in the test module `test_usolve_rosenN.py`:

```
import uopt
import numpy as np
import random
import sys

from rosenN import *

def run():

    N = 20

    v0 = np.ndarray((N,))
    for i in range(N):
        v0[i] = random.random()

    J = rosenNs(v0)
    y = rosenN(v0)
    Js = np.linalg.norm(y)**2
```

```
    print(J, Js, J -Js)

    vs = v0.copy()
    ys = y.copy()
    opts = {}

    (rc, info) = uopt.usolve(v0, vs, ys, rosenNObj,
                             rosenNJac, rosenNJacv, None, rosenOutput,
                             opts, None)

    if N < 50:
        print(vs)
        print(ys)
    else:
        print(vs[0:20])
        print(ys[0:20])
    print('v error', np.linalg.norm(vs-1))
    print('J error', np.linalg.norm(ys)**2)

    sys.exit(rc)

if __name__ == "__main__":
    run()
```

The matrix-free optimization with `usolve` is invoked, by not providing the Jacobian-handle but instead the Vector-Jacobian-handle, as shown in the test module `test_usolve_rosenN_Jv.py`:

```
import uopt
import numpy as np
import random
import sys

from rosenN import *

def run():

    N = 20

    v0 = np.ndarray((N,))
    for i in range(N):
        v0[i] = random.random()

    J = rosenNs(v0)
    y = rosenN(v0)
    Js = np.linalg.norm(y)**2
    print(J, Js, J -Js)

    vs = v0.copy()
    ys = y.copy()
    opts = {'tolGradAbs': -1, 'tolGradRel': -1}

    (rc, info) = uopt.usolve(v0, vs, ys, rosenNObj,
                             None, rosenNJacv, rosenNJacTv, rosenOutput,
                             opts, None)

    print(vs)
    print(ys)
```

```
    sys.exit(rc)

if __name__ == "__main__":
    run()
```

6 R interface

The R interface is currently the most evolved in comparison to the other languages. On top of the plain procedural interface there is a utility layer that store and display the iteration history.

6.1 Installation

The R package is found in the folder `share/uopt/R/` of the UOpt distribution.

For installation on Windows, the RTools must be installed.

It is installed on the command line with

```
R CMD INSTALL uopt_1.2.0.181.tar.gz
```

or from within R with

```
install.packages('uopt_1.2.0.181.tar.gz', repos = NULL)
```

6.2 Usage

The R interface of UOPT has two levels: One is the procedural level via the function **uopt**. The other lets you create a UOpt object via **uoptCreate**, that can automatically record iteration history and can generate reports and visualizations.

6.3 Examples

6.3.1 The procedural interface

```
library(uopt)
library(adr)

f <- function(x) {
    sqrt(sum(x^2))
}

N <- 120

v0  <- array(rnorm(1:N)*1000, c(1, N))

f0 <- f(v0)

g <- function(x) {
    adrDiffRev(f, list(x))
}

H <- function(x) {
    adrHessian(f, list(x))
```

```
}

opts <- list(trace=0x0, sdirs=16)

vs <- uopt(v0, f, g, H, options = opts)
```

This produces the following output or similar:

```
J=11609.1  it=0        nv=120      t0=1589490000
serial=ZSCc7ZMl0zd5naZs/9TCLn version=UOPT-1.0.1
check-Hessian=9.00549e-10 check-gradient=5.99347e-08 ng=1        tg=0.00543398
Er=7.27509  Hc=1.11022e-16 J=11524.6  beta=1387.79  it=0        lh=100      m=SD4      st=32
tj=0.077569
Er=994.585  Hc=1.22125e-15 J=62.4021  beta=11462.2  it=1        lh=100      m=SD4      st=0
tj=0.130848
Er=0.975641 Hc=1.66533e-16 J=62.3412  beta=2.74939  it=2        lh=100      m=SD4      st=32
tj=0.199856
Er=990.833  Hc=3.47729e-24 J=0.5715   beta=61.7697  it=3        lh=100      m=SD4      st=0
tj=0.282021
Er=994.982  Hc=3.33067e-16 J=0.00286762 beta=0.574367 it=4        lh=100      m=SD4      st=0
tj=0.334755
Er=996.908  Hc=4.44089e-16 J=8.86533e-06 beta=0.00285876 it=5        lh=100      m=SD4      st=0
tj=0.41402
Er=996.833  Hc=2.22045e-16 J=2.80727e-08 beta=8.8934e-06 it=6        lh=100      m=SD4      st=0
tj=0.550907
Er=999      Hc=4.93038e-32 J=2.80592e-11 beta=2.80447e-08 it=7        lh=100      m=SD4
st=0        tj=0.613655
Er=998.899  Hc=7.77156e-16 J=3.08817e-14 beta=2.80283e-11 it=8        lh=100      m=SD4
st=0        tj=0.678236
Saved state to uopt-current.mat
J=3.08817e-14 code=27       exit=tolObjAbs reached it=8        status=0
```

The result object is a numerical vector, and carries attributes indicating the optimization result. The attribute `info` carries a list where element `exit` reports the exit status fields.

```
data.frame(attr(vs, 'info')$exit)
```

```
             J code               exit it status
1 3.08817e-14   27 tolObjAbs reached  8      0
```

Element `last` reports the status fields of the last iteration.

```
data.frame(attr(vs, 'info')$last)
```

```
       Er           Hc           J        beta it  lh   m st       tj
1 998.899 7.77156e-16 3.08817e-14 2.80283e-11  8 100 SD4  0 0.678236
```

6.3.2 The object interface

The main change in the object interface os that you create a UOpt object with **uoptCreate**. Then use the field `uopt` of that object instead of the global **uopt** function.

```
library(uopt)
library(adr)

myuopt <- uoptCreate()
uopt <- myuopt$uopt
```

For the next part, the usage is exactly equal to the procedural interface:

```
obj <- function(x) {
    N <- length(x)
    sum((1 - x[-N])^2 + 100 * (x[-1] - x[-N]^2)^2)
}

N <- 40
v0  <- array(rnorm(1:N)*10, c(1, N))
J0 <- obj(v0)

opts <- list(maxit=800, tolObjAbs=1e-7, sdirs=0x18,
             trace=0x3, saveplots=1)

vs <- uopt(v0,
           f = obj,
           g = function(x)
               adrDiffRev(obj, list(x)),
           H = function(x)
               adrHessRev(obj, list(x)),
           Hv = NULL,
           s = function(x, status) {
               uoptShowState(status)
             if ('st' %in% names(status)) {
              if (status$st != 0) {
                cat(sprintf('Non-0 status\n'))
              }
             }
               0
           },
           options = opts)
```

The UOpt object allows you access to the entire iteration history, so you may want to save it for later reference:

```
save(myuopt, file='myuopt.Rdat')
```

The report is a self contained HTML file generated by r2x and a simple report generator. When `show` is TRUE (default) the report is saved to a tempfile, the file name is returned and the report is shown in the viewer. Otherwise, the report document is returned.

```
r <- uoptReport(myuopt)

doc <- uoptReport(myuopt, show=FALSE)
```

The **uoptMovie** function is only available when `saveplots` is set non-zero.

```
m <- uoptMovie(myuopt)
```

The function result is again identical to the procedural interface.

```
data.frame(attr(vs, 'info')$last, attr(vs, 'info')$exit)
```

```
       Er          Hc           J       beta it       lc      lh   m st      tj
1 999.675 0.000282753 3.35251e-09 0.00421191 93 0.508819 100.028 SD3  0 9.72358
          J.1 code              exit it.1 status
1 3.35251e-09   27 tolObjAbs reached   93      0
```

6.4 License

The R interface to UOpt, including the report generator is licensed under the GPLv3 license.

7 The status call back function

The status call back is called in each iteration. It must return an integer. When the status function returns non-zero the iteration is stopped.

The status callback function is given the current state vector. The status callback function is also provided with a named map of values for the reference of the user.

The fields given to the status function may vary depending on the occasion:

init During startup

check After the derivative checks

iteration After each iteration

exit At exit

7.1 Status field reference

Fields in initial calls report the software version and initial parameters of the optimization task:

Name	Descripion
it	Initial interation number
version	Version info string
serial	Software serial number
nv	Number of variables
t0	Statrt timestamp
J	Initial function value
check-Hessian	Residual the Hessian check
check-gradient	Residual of the gradient check
ng	Norm of the test gradient
tg	Time of the test gradient

Fields in the iterations may be different depending on the modes being used by `sdirs`. They show the current state of the optimization and allow the progress monitoring.

Name	Descripion
Er	Enhancement in parts per thousand
Hc	Condition of the Hessian
J	Function value
beta	Combined step length
it	Iteration
lc	Regularization amount
lh	Regularization health indicator
m	The search direction
st	Status
tj	Timestamp relative to t0

Fields in the final call indicate the result and reason for stopping:

Name	Descripion
J	Function value
code	Numerical exit code
exit	Exit code in text
it	Iteration
st	Status

8 Parameter reference

Name	Default	Range	Description
trace	0		Bitfield for message verbosity
sdirs	24		Bitfield for preferred search directions
maxit	10000		Max. number of iterations
maxrt	3600*24 s		Max runtime
tolObjAbs	1e-12		Stop when Objective is smaller
tolObjRel	1e-12		Stop when quotient of current objective to initial objective is smaller
objLimit	0		Stop when Objective is smaller
tolGradAbs	1e-9		Stop when Gradient is smaller
tolStepAbs	1e-100		Stop when best step found is smaller
tolStepRel	eps		Stop when quotient of best step to current objective is smaller
tolEnhAbs	-1		Stop when current enhancement is smaller
tolEnhRel	-1		Stop when quotient of current enhancement to objective is smaller
checkgradient	0		Initial gradient check: -1 off, 0 auto, 1 on
checkHessian	0		Initial Hessian check: -1 off, 0 auto, 1 on
tolGradCheck	1e-4		Tolerance for checking user gradient against finite differences
tolHessCheck	1e-2		Tolerance for checking user Hessian against finite differences
cgmode	2	0 - 4	CG update: FR, PR, HS, or DY
enableSigIntHandler	0		Enable handler for signal SIGINT
enableSigUSR1Handler	1		Enable handler for signal SIGUSR1
enableSigUSR2Handler	1		Enable handler for signal SIGUSR2
enableFloatExcept	1		Enable floating point exception handler
enableStopFile	0		Enable stop file
enableResume	0		Enable load of initial value from resume file

9 Details

9.1 AI & IT UG

AI & IT UG (haftungsbeschränkt) – AI Software Solutions and Consulting

- Visit us on the web: `https://www.ai-and-it.de`
- or reach us by email: info@ai-and-it.de.

9.1.1 Address

AI & IT UG (haftungsbeschränkt)
Am Friedrich 7
52074 Aachen

9.1.2 Further information and contact data

Telefon +49 176 632 592 48
E-Mail info@ai-and-it.de

9.2 Copyright