Linköping Institute of Technology
Department of Mathematics/Division of Optimization
Kaj Holmberg

2018–09–10

Lab Information

1 NILEOPT: Nonlinear Optimization

1.1 Introduction

NILEOPT is a program for visual optimization of nonlinear optimization problems (NLP). The main aim is to provide insights into peculiarities of nonlinear problems and methods. NILEOPT contains two parts, one for problems of any dimension, with constraints, without graphics, and a graphical part for functions of two variables, without constraints. The program is able to read problem data and make changes in data, and solve the problem using Cobyla (from SciPy). The graphical part of NILEOPT allows for visualization of two-dimensional functions (as 3D surface and level curves). It is also an interactive application for a number of unconstrained nonlinear optimization methods (see section 1.4). The program is implemented in Tcl/Tk by Kaj Holmberg, and run as a tclkit.

To run the program, open a terminal window, and write `/course/TAOP88/dine 3`.

NILEOPT minimizes an objective function, subject to constraints, if required. All constraints should be written explicitly. One can use constraints of the form $=$, $\leq$ or $\geq$. (NILEOPT turns the $\geq$-constraint into $\leq$-constraint by multiplying it with -1.) NILEOPT can deal with polynomials (without parentheses), like $f(x) = 3x_1x_2^2x_3^2 + 7x_1^4x_3 - 13x_1x_2 + 3x_3$, but not $(x_1 - x_2)^2$, $sin(x)$ or $exp(x)$.

1.2 Menu

NILEOPT starts with the main window which contains the problem part, the graphic part and the menus which will be described below. The first status bar contains several buttons and displays the name of the problem (the name of the data file) and the size of the problem. The second displays the function's name. The third status bar displays the iteration point, objective function value and current method.

File "Alt"-f
Here one can read input data for a problem, save data in a current file or in a new one (in an internal format). One can also read the model file and save it. One can print out model and graphics. It is also possible to list and erase existing problem files.

Optimization "Alt"-o
Here one can define a starting point for the optimization method. One can edit the problem, and solve it using Cobyla. (This can be done using the buttons `Edit` and `Cobyla` in the status bar.) Editing opens a new window, which contains among others the button `Check`, which is used for controlling the result before saving changes in the editor window. Please do this before you save, because parts of the model which are not recognized will disappear when the changes are saved.

Visualization "Alt"-v
Here one can choose to look at the model, as well as gradient or Hessian for objective function and constraints (explicitly and in the given starting point) in the problem window. For graphical visualization, one can choose resolution, view options, and decide whether any constraints should be displayed. One can also indicate infeasible domain (approximately).

Graphical optimization "Alt"-g
Here one can choose function and method, as well as the step size in the method. (Most of this can be done using the buttons in the status bar.) One can also change the maximum number of iteration (default is 1000).

Help "Alt"-h
Here one can get a short help text in a separate window, in Swedish or English. One can also get a list of the available shortcuts. There is a possibility to change layout such that the window with the level curves is larger and the 3D graphic is smaller.

There are buttons in the first status bar, which are the shortcuts for the frequently executed commands.
• EDIT: Edit a current problem.
• STEP: Make a step by the method. Trajectory and a new iteration point are displayed in the level curves graph.
• RUN/STOP: Run the method without stops at each iteration. (Alternatively: Stop the run.)
• START POINT: Introduce a starting point (could be a current point, if there is one).
• CLEAR PLOT: Remove all marked points in the level curves graph.
• FUNCTION: Choose a function from the list.
• METHOD: Choose a method, i.e. how the search direction is computed. (See the list of methods below.)
• COBYLA: Solve the current problem (using Cobyla).

Observe that one has to select a method before pressing START POINT. When pressing COBYLA, you will be asked to provide a starting point, if it hasn't been provided yet. Otherwise, a provided starting point is used. The other methods are started by pressing START POINT. One can also assign a starting point by clicking on the level curves graph.

There are buttons for moving and zooming the graphic window to the right of the model window.

1.3 Data format

NILEOPT provides a possibility to change data interactively, and also to create a new problem. NILEOPT saves and solves a problem using an internal format which can not be edited manually. Such files have extension ".nlp".

NILEOPT can also read model files (with extension ".mod") in a more convenient format. This format is used to create new problems, and when editing them. The format is a simplified version of the one used in GMPL/AMPL. The simplification is that only polynomials can be used, but not trigonometrical functions, logarithms and exponential functions. Moreover, it can not read parenthesis, index sets, sums or vectors.

The format of the data file is the following. Each row ends with semicolon (;). First each variable is declared with

```
var x1;
```

Then the objective function is declared with some name:

```
minimize Obj:  2*x1^2 - 2*x1*x2 + 3*x2;
```

First `minimize` (or `min`), then the name of the objective function (here `Obj`), followed by a colon (:). Then follows the function expression itself. It may contain any number of terms with plus (+) or (-) in between. Each term consists of a coefficient and variables with the multiplication sign (*) in between. Each variable can have an integer positive exponent, indicated with the sign ^. (Sometimes one may have to press space to get this sign.)

After that the constraints follow. Each constraint is introduced with the text `subject to` (or `s.t.` or `st`). Then the name of the constraint follows (e.g. `con1`), followed by a colon (:). Then follows a function expression according to the same rules as for the objective function, then an equality sign (=) or inequality signs (`<=` or `>=`). Finally, the right hand side coefficient is given.

```
s.t.  con1:  x1^2 + x2^2 <= 1;
s.t.  con2:  x1 + x1*x2 <= 1;
st con3:  x1+4*x2<=2;
```

1.4 Graphical optimization

The graphical optimization part in NILEOPT is made for two-dimensional problems. (If the problem has more than two variables, no graphics is displayed.) It contains a window with a 3D figure of the function surface and a window with level curves.

The main objective is to perform optimization step-wise and display each iteration point in the level curves graph. Then one can compare different methods, and also the effect of using different starting points.

This part of NILEOPT can read more general function expressions, but if they do not fulfill the requirements for input data described above (polynomial without parenthesis), it is not possible to do editing. There is a large number of test functions. They are divided into 3 categories:

1. Polynomials according to the rules. They can be saved in an internal format, edited, and gradient and Hessian can be computed symbolically (exactly). They are marked with * in the menu.

2. More general functions, for which one can specify explicitly gradient and Hessian in the data file. These functions can not be edited or saved in an internal format, but one can use methods that require gradient and Hessian. They are marked with + in the menu.

3. More complicated functions, for which gradient and Hessian can not be computed or provided. They are marked with - in the menu. These function are given in a form of a subroutine (in Tcl) which returns the function value. This subroutine can be arbitrary complicated. NILEOPT has a possibility to numerically compute an approximate gradient. Therefore gradient based methods can be used. There is however no Hessian approximation available.

Each time a function is selected, the function expression (if it is available) is displayed in the problem window. If a function is of category 1, it can be edited further in the editor window. In the function list, function 0 is the function currently loaded. This way one may create a new function and use it graphically.

Methods:

Cobyla: Constrained Optimization by Linear Approximation, provided by SciPy.
Steepest descent: The steepest descent method.
Newton's method: Newton's method, with explicit, exact gradient and Hessian.
Levenberg (convexified Newton): Levenberg's convexification of Newton's method, with explicit, exact gradient and Hessian.
Levenberg-Marquardt (convexified Newton): Levenberg-Marquardt's convexification of Newton's method, with explicit, exact gradient and Hessian.
Nelder-Mead's metod: Nelder-Mead's method (with a shrinking simplex). Does not require gradient.
Conjugate gradient (Fletcher-Reeves): Modified steepest descent method, by Fletcher-Reeves.
Conjugate gradient (Polak-Ribiere): Modified steepest descent method, by Polak-Ribiere.
Conjugate gradient (Hestenes-Stiefel): Modified steepest descent method, by Hestenes-Stiefel.
Quasi Newton (DFP): Newton's method with Hessian replaced by an approximation, by Davidon-Fletcher-Powell. Does not require Hessian.
Quasi Newton (Broyden): Newton's method with Hessian replaced by an approximation, by Broyden. Does not require Hessian.
Quasi Newton (SR1): Newton's method with Hessian replaced by an approximation, by "Symmetric Rank 1". Does not require Hessian.
Quasi Newton (BFGS): Newton's method with Hessian replaced by an approximation, by Broyden-Fletcher-Goldfarb-Shanno. Does not require Hessian.

NILEOPT terminates when the norm of the gradient is very small, when the iteration step is very small or when the maximum number of iterations has been done. (The maximum number of iterations can be changed.)

There is no constraint $t \geq 0$ for the line search, so a negative length step can be obtained. (However, this typically does not matter.) If there are several local minima along the search direction, we do not know if the closest or the best one is chosen.