Uppsala universitet
1TD184 Optimisation

Assignment 1 – basic option
October 27, 2023

Assignment 1: Nonlinear least squares problems and the Levenberg–Marquardt algorithm

In this assignment you will practice implementation and gain experience in parameter estimation with the Levenberg–Marquardt algorithm. The assignment is rather easy to carry out in its most basic form, but the idea is that you should develop your code as far as possible. Depending on how much you develop your code, you may have practical use for it also in the future. Co-operation with another group solving a non-linear least squares problem (eg. the reaction kinetics assignment) is encouraged.

Background

There exist many advanced optimisation packages whose performance is difficult to match with home-made implementations. However, many of these packages are proprietary (and sometimes quite expensive) or restricted to certain programming languages. Parameter estimation is a very common type of problem encountered in science, engineering, statistics etc, and for this class of problems even a home-made implementation can be practically useful.

Tasks

1. Write a Matlab implementation of the Levenberg–Marquardt algorithm for the solution of an arbitrary nonlinear least squares problem. The header of your function should look like

   ```
   function [x,resnorm,residual] = levmarq(func,x0)
   ```

   Here, the output data is the solution vector `x`, the residual norm `resnorm` and the residual vector `residual`. The input data should be a string variable `func` with the name of the function defining your least squares problem and an initial solution `x0`. If you find it useful, you can add a third argument `options` that passes various optimisation parameters (such as termination conditions, maximum number of iteration etc.) to your function. See for instance the options structure created by `optimset` in Matlab for an example.

   If the user supplies the gradient of the residual vector, this should of course be used. The function computing the residual should then have a header like `function [r,gradr]=residualfunc(x)`. It should also be possible to

call your Levenberg–Marquardt function without supplying the gradient. In this case your implementation must calculate an approximation of the gradient using a numerical approximation such as finite differencing.

The choice of the regularisation parameter μ in the Levenberg–Marquardt algorithm needs some particular attention. At an initial stage, you can use a fixed value, but in order for the function to be reliable for a variety of problems, some kind of updating strategy will be necessary. There is no general rule for how this is done in the best way, but some more or less heuristic method can be used. For an idea of how you can reason, study the the sketch of the Trust-Region algorithm in chapter 11.6 of the textbook for *inspiration*. You can also think of what makes the Gauss–Newton method fail and let μ be related to the size of the residual. Experiment!

2. Test your function on the model

$$y(t) = x_1 \mathrm{e}^{x_2 t}$$

with the following data:

t_i	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
y_i	7.2	3.0	1.5	0.85	0.48	0.25	0.20	0.15

using a set of different initial solutions for x_1 and x_2. Compare to results found using the function `lsqnonlin` in the Optimization toolbox. Use

```
options = optimset()
```

in order to trace the progress of the execution.

3. If you find another group that has solved either the black-box optimisation or the chemical reaction assignment, give them your code and let them use it for their optimisation. If you have followed the instructions regarding the function header, and supplied your function with a sufficient help text, they ought to be able to use your function without difficulties.

Acknowledgements: Original assignment idea created by Cheng Gong and Daniel Noreland