Matlab Optimization Package

Robin Deits

August 5, 2010

Contents

1 Introduction

This software package contains a set of MATLAB scripts and functions which can be used to perform optimizations on arbitrary functions. They are not meant to fully replace the MATLAB optimization methods, but rather to complement and improve them. The primary functions that the user will deal with will be explained in this documentation.

2 Setup

To install this software package, simply put the `.m` files included here in a folder somewhere on the hard drive and add that folder to the MATLAB search path. The files should all be in the same folder, and they should be the only files in that directory.

3 About Optimization

3.1 The Optimization Process

Optimization, as it is used in this software, consists of minimizing a single function of one or more input parameters. The function to be minimized is called the cost function. The optimizer itself is simply a program that generates inputs for the cost function, reads the output of the function given those inputs, and then generates the next set of inputs in an attempt to achieve the lowest cost function value. The optimizer needs no knowledge about the structure or workings of the cost function itself; instead, it treats the cost function as a black box that converts inputs to outputs.

In this way, a human worker guessing different input values to try could be seen as an optimizer, but there are a tremendous number of methods for automating and improving this process. This software package uses two such methods, the Nelder-Mead simplex method and the genetic algorithm (GA). The workings and advantages of each method can be found in most texts on optimization, so this document will only focus on their use here.

3.2 Local vs. Global Solutions

An important factor in the optimization process is the difference between local and global solutions. A local minimum is a point at which any infinitesimal adjustment of the input parameters results in a higher value of the cost function. A global minimum is the point with the absolute lowest cost function value out of all the possible points. In this way, we can see that the global minimum also satisfies the criteria for a local minimum.

This is important because many optimization methods have a tendency to find only a local solution, rather than the global one. This can be understood by imagining a hilly landscape in which one is trying to find the point with the lowest altitude. One could simply go downhill in whichever direction has the steepest slope, but this could easily leave one trapped in a valley which is lower than its surroundings but not the absolute lowest point. This software package uses the genetic algorithm to help overcome these issues, since its random mutation and recombination of parameters can sometimes allow it to escape local minima and find the global minimum.

4 File Description

4.1 Cost Function

4.1.1 simExample.m

This is a sample function to be optimized, and it is meant to be replaced by the user's own function in question. It is called by `costFunction.m`, so the call to `simExample` within that file must be replaced by a call to the user's desired function.

`simExample()` expects its argument to be a string of MATLAB variable assignment expressions, separated by commas. The reasoning and function behind this slightly odd choice of variable passing method is explained in the file's comments.

4.1.2 costFunction.m

This is the file which is called by the optimization routines. It calls the user's simulation program and saves the results of each test in `summary.txt` and `results.mat`. The function and purpose of these files can be found in the comments.

4.2 Running the Optimizer

4.2.1 optimizeGA.m

`optimizeGA.m` contains the actual call to the genetic algorithm. The variables to be optimized and their initial and limiting values should be set within the file itself, then the algorithm can be run to completion by setting the MATLAB current directory to an empty folder where the test results should be stored and typing

```
>> optimizeGA();
```

at the MATLAB prompt.

4.2.2 optimizeFMinSearch.m

This function is almost identical to optimizeGA, but it uses the Nelder-Mead Simplex algorithm implemented by `fminsearchbnd.m`, rather than the GA.

4.2.3 gridTest.m

Rather than running a traditional optimization routine, this program does sensitivity testing on two variables. The variables to be set up and their range and number of steps can be set within the file itself. `gridTest` also produces contour and 3D surface plots of its results.

4.3 Analyzing Results

4.3.1 reconstructSim.m

After the optimization has completed or been stopped, any test that it performed can be repeated by setting the MATLAB current directory to the location of the `results.mat` file and using `reconstructSim(testNumber)`. In addition, `reconstructSim('best')` will automatically repeat the test with the lowest cost.

4.3.2 plotOptimization.m

Like `reconstructSim`, this function should also be called with the MATLAB directory set to the location of `results.mat`. It generates a plot of test number versus lowest cost achieved so far, which is useful for viewing the progress of the optimizer.

5 Other Useful Files

Other useful files whose function and use are described in their comments are:

`makeContourPlot.m`
`oneVarTest.m`
`randomSearch.m`