Matlab – GenOpt Interface

Author:
Marco Bonvini
Ph.D. Student Politecnico di Milano
bonvini.m@gmail.com,
bonvini@elet.polimi.it
Visiting student at AIT – SBT unit
Marco.Bonvini.fl@ait.ac.at

User Guide

This guide shows how to use GenOpt for solving an optimization problem, in the case the function to be minimized is written in Matlab. The guide presents both the instruction for connecting GenOpt and Matlab on Linux and Windows OSs.

Folder structure

The folder that contains all the information and the examples that show how to interface GenOpt with Matlab are contained within the `MatlabGenOPT` folder. This folder contains two subfolders:

- `MatlabGenOPT_Linux`
- `MatlabGenOPT_Windows`

In the first one are listed a series of examples for Linux users, while the latter for Windows one. Both directories have the same structure. They contain a list of subfolders, each one representing an application example.

Example folders

Each example subfolder contains the following files:

- `command.txt`
- `MatlabLinux.cfg / MatlabWindows.cfg`
- `optLinuxMatlab.ini / optWindowsMatlab.ini`
- `InitialisationScriptTemplate.m`
- `run.sh / script.bat`
- `*.m`

The first file (`command.txt`) is the command file that contains information needed by GenOpt in order to define the algorithm to be employed and the name and number of parameters that vary during the optimization.
The second one (`MatlabLinux.cfg / MatlabWindows.cfg`) is the configuration file. Here are listed useful information that GenOpt needs in order to understand when the simulation fails and the command to be executed in order to execute the Matlab code.
The third one (`optLinuxMatlab.ini / optWindowsMatlab.ini`) is the initialization file, the one that has to be opened by GenOpt when to perform the optimization. This file contains information about the template file that will be used for creating the input file (needed by Matlab code), the name of the log file provided by Matlab (in order to check if error occurs during the computation) and the output file where the result (the value of the cost function computed) can be found. Other information needed by GenOpt are listed.
The fourth one (`InitialisationScriptTemplate.m`) is the template used by GenOpt for creating the initialization file needed by Matlab in order to perform its computations.
The fifth (`run.sh / script.bat`) is the script that is executed by GenOpt. This script contains all the necessary instructions for executing the Matlab code successfully.
The last one is a `*.m` file that contains the Matlab code that perform the computation of the cost function.

Detailed description

In this section an application example (the easier one) is deeply analyzed, in order to better understand the meaning of each part of the code needed for interfacing GenOpt with Matlab.

Function to be minimized

The function that GenOpt will try to minimize in this introductory example is a quadratic function

f(x) = a x^2 + b x + c

where a = 3, b = 10 and c = 1. GenOpt will try to find the x value for which f(x) reach its minimum value.

The Matlab code

GenOpt, when perform its procedures for finding the minimum of a given cost function, needs to call an unknown number of time the code that computes the cost function. GenOpt is able to communicate with any “piece of code” that is able to read input value from an input (textual) file, and provide results by a second (again textual) file.
In this case, in order to make Matlab able to communicate with GenOpt, it will read values from an input file and write the result in a second one. In addition, the Matlab

code have to write a log file for tracing if something goes wrong during the computation.

The code representing the input file, will be created according to this template

```
% InitialisationScriptTemplate.m
% initialization script for Matlab

% parameters of the function
% f(x) = a*x^2 + b*x + c
a = 3.0;
b = 10.0;
c = 1.0;

% Input value where the function is evaluated
x = %u00%;
```

The code listed above, contains a series of Matlab instructions. The first ones declare the parameters characterizing the cost function, while the latter declares the value in which the cost function will be evaluated. This value will be decided by GenOpt (it is possible to see that it is not a numerical value as the previous ones but it is a string. This string indicates the name of the variable used in the optimization algorithm. GenOpt will replace this string with a numerical value). The file created by GenOpt accordingly to this template is named `InitialisationScript.m`.

The Matlab code that computes the cost function follows

```
% QuadraticFunction.m
% first of all open the log file
% open the MatlabLog.txt file
try
     fid_log = fopen('MatlabLog.txt','w');
catch
     % error while opening the Log File
     quit();
end
```

The log file (named `MatlabLog.txt`) necessary for inspections is opened, if some problem arises during this task the program quit immediately.

```
% open the result file ./result.txt
% if already existing, overwrite it
try
     fid = fopen('result.txt','w');
     % result file open correctly
     fprintf(fid_log,'%s\n','Result file open correctly');
catch
     % Cannot open the file where write results
     fprintf(fid_log,'%s\n',
     'Matlab Error - Cannot open/create result file');
end
```

The `result.txt` file where the value of the cost function computed will be written is opened. If some problem, the log file take trace.

```
try
     try
          % initialize the values of the cost function and get
          % the input values
          InitialisationScript;
     catch
          % Cannot open the script for initialisation
          % probably it is not in the Matlab path
          fprintf(fid_log,'%s\n',
          'Matlab Error - Cannot open Initialisation script');
     end

     % compute the cost function
     y = a*x^2 + b*x + c;
catch
     % Cannot compute the cost function
     fprintf(fid_log,'%s\n',
     'Matlab Error - Cannot compute the cost function');
end
```

In this part of the code there are two `try-catch` structures nested. In the inner one, the instruction `InitialisationScript` recalls all the declarations contained within the `InitialisationScript.m` file. Thus, after executing this instruction all the parameters and the values where the cost function has to be evaluated are know.
If this instruction terminates successfully the cost function can be computed, otherwise if some error occurs, it is reported in the log file.

```
try
     % convert the number into a string and print it
     fprintf(fid,'%s\n','# MATLAB Results');
     fprintf(fid,'y = %.10f',y);

     % terminated correctly
     fprintf(fid_log,'%s\n','Matlab Terminated correctly');
catch
     % terminated with error
     fprintf(fid_log,'%s\n',
     'Matlab Error - Problem while writing the final result');
end

% close all files
fclose('all');

% close the Matlab terminal
quit();
```

After the computation of the cost function value is written in the file named `result.txt`. If some error occurs, again, the log file will be updated. At the end all the file descriptors are released and the program terminates. It is important the `quit()` command because without it at each iteration of the optimization algorithm Matlab occurrences remain open.

Executing the Matlab code

The Matlab code listed above, that evaluates the cost function at a given point x (selected by GenOpt) has to be executed. It is possible to call Matlab from a shell (`Bash` in Linux and `cmd` in Windows) and pass to it a series of parameters that indicate which file has to be executed and where to find all the necessary information.

```
# run.sh
#!/bin/sh
# script that executes the matlab function

# directory where matlab executable is located
matlab_home='/usr/share/matlab/bin/matlab'
matlab_options=' -nojvm -nodesktop -nosplash'

# path of the Matlab code that compute the cost function
add_path="path(path,
     '/home/marco/Desktop/MatlabGenOPT/QuadraticFunction')";
# path of the current directory
add_path2="path(path,'$PWD')";

# name of the Matlab code that compute the cost function
function_name='QuadraticFunction'

# execution
$matlab_home $matlab_options -r "$add_path ; $add_path2 ;
$function_name;"
```

The code listed above is the script that executes the Matlab code in Linux. First it is necessary to indicate the path of Matlab executable (`/usr/share/matlab/bin/matlab`). Before calling the Matlab executable a series of options have to be specified. These options are:

- `-nojvm` that indicates to do not load the Java Virtual Machine
- `-nodesktop` that indicates to do not load the desktop environment
- `-nosplash` that indicates to do not show the splash screen

The Matlab code is executed through this instruction:

```
$matlab_home $matlab_options -r
"$add_path ; $add_path2 ; $function_name;"
```

The first part (`$matlab_home`) indicates that Matlab executable has to be executed, the second part (`$matlab_options`) indicates the parameters. After these two parts there is `-r` and a sequence of instructions that have to be executed once Matlab is running. The sequence of instructions is `"$add_path ; $add_path2 ; $function_name;"`. The last one indicates that the script previously introduced (`QuadraticFunction.m`), that computes the cost function has to be executed. The first two instructions are needed because Matlab has to add the current working directory and the directory in which is located the Matlab code to its path.

In the following is reported the script that do the same operations in Windows

```
: script.bat
@echo off
: Important address - Matlab executable file location
set matlabLocation=C:\Programme\MATLAB\R2007b\bin\matlab

: Options
set matlabOptions=-nojvm -nosplash -nodesktop -wait

: get the current directory, in this one there will be the
: initialization File provided by GenOpt
set pwd=%cd%
: set the path for Matlab
set command1=path(path,'%pwd%')
: set the directory in which is contained the matlab file to
: be executed
set command2=path(path,'D:\Dokumente und Einstellungen\
MBonvini\Desktop\MatlabGenOPT_Windows\QuadraticFunction')


: the name of the file that will contain the Cost Function
: (written in Matlab code)
set fileName=QuadraticFunction

: execute this command
%matlabLocation% %matlabOptions% -r “%command1%;%command2%;
%fileName%;"
```

The only difference between the Linux version (except for the language syntax and paths) is the presence of the additional option `-wait`. This option is necessary because otherwise the command line do not wait for the termination of Matlab, and so problems may appears (e.g. the result and log files are not available yet).

The GenOpt files

At this point are shown the files needed for setting up the optimization procedure in GenOpt. The first one is `command.txt` (that is the same both for Linux and Windows)

```
Vary{
  Parameter{   Name = u00; Min = -10; Ini = -9; Max = 10;
               Step = 1; }
}
```

Here is reported the list of variables that are explored during the optimization. In this case just one variable namely `u00` (The same reference is into the template file).

```
OptimizationSettings{
  MaxIte = 1000;
  MaxEqualResults = 10;
  WriteStepNumber = false;
}

Algorithm{
  Main = DiscreteArmijoGradient;
  Alpha = 0.5;
  Beta  = 0.8;
  Gamma = 0.1;
  K0    = 10;
  KStar = 0;
  LMax  = 50;
  Kappa = 50;
  EpsilonM = 0.05;
  EpsilonX = 0.05;
}
```

Above is reported a list of parameters that describe the optimization algorithm.

In the following is listed the configuration file. This file contains a series of information about the number format representation and the string that may appear in the log file if any problem happens during the computation. The last part (the most important dependent from the OS) indicates how to execute the Matlab code. Here follows the `MatlabLinux.cfg` file

```
// Error messages of the simulation program.
// if in the log file this string is found the optimization
// stops
SimulationError
{
    ErrorMessage = "Matlab Error";
}
// Number format for writing the simulation input files.
IO
{
    NumberFormat = Double;
}
/*  Specifying how to start the simulation program.
    In "Command", only those words in %xx% are
    replaced (possibly with empty Strings).
*/
SimulationStart
{
  Command = "bash ./run.sh";

  WriteInputFileExtension = true;
}
```

The Linux version, `MatlabWindows.cfg`, differs just for the last part

```
// Error messages of the simulation program.
// if in th elog file this string is found the optimization
// stops
SimulationError
{
    ErrorMessage = "Matlab Error";
}
// Number format for writing the simulation input files.
IO
{
    NumberFormat = Double;
}
/*  Specifying how to start the simulation program.
    In "Command", only those words in %xx% are
    replaced (possibly with empty Strings).
*/
SimulationStart
{
  Command = "\%Simulation.Files.Template.Path1%\\script.bat";

  WriteInputFileExtension = true;
}
```

Last, there is the file that has to be passed to GenOpt when to perform the optimization. The file is named `optLinuxMatlab.ini` and its structure is the following

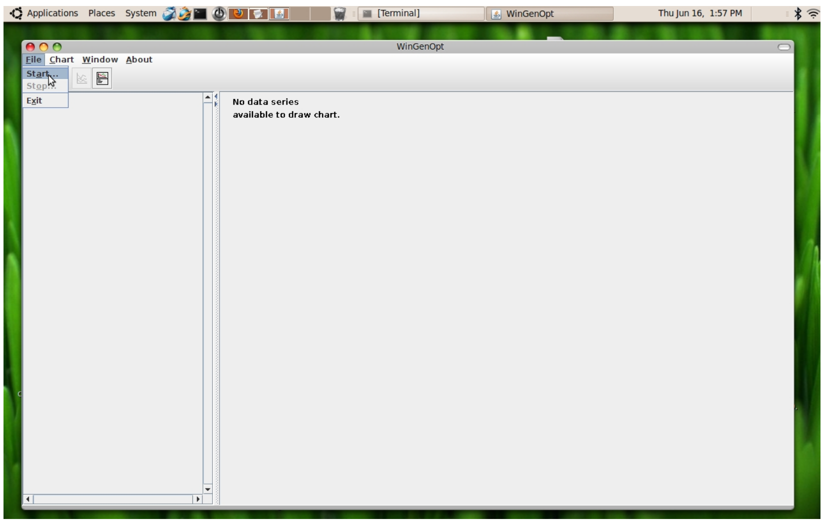
```
Simulation {
  Files {
    Template {
      File1 = InitialisationScriptTemplate.m;
    }
    Input {
      File1 = InitialisationScript.m;
    }
    Log {
      File1 = MatlabLog.txt;
    }
    Output {
      File1 = result.txt;
    }
    Configuration {
      File1 = "MatlabLinux.cfg";
    }
  }
  ObjectiveFunctionLocation{
      Delimiter1 = "y = ";
      Name1      = "y";
  }
} // end of section Simulation
Optimization {
  Files {
    Command {
      File1 = command.txt;
    }
  }
} // end of configuration file
```

The structure of the `.ini` file is simple. It is composed by two main parts; the first one concerns the simulation while the second one concerns the optimization. In the first part it is indicated the name of the template necessary for creating the input file. The name of the log file and the result one are specified too. The last file that is listed is the configuration file needed for executing the Matlab code (the unique part that depends on the OS). Before the optimization part is defined, it is specified where to read the output value in the result file (in this case after the string `“y = ”`). In the Optimisation part the command file described above is indicated.
The Windows version of this initialization file differs only for the configuration section

```
…
Configuration {
      File1 = "MatlabWindows.cfg";
}
…
```

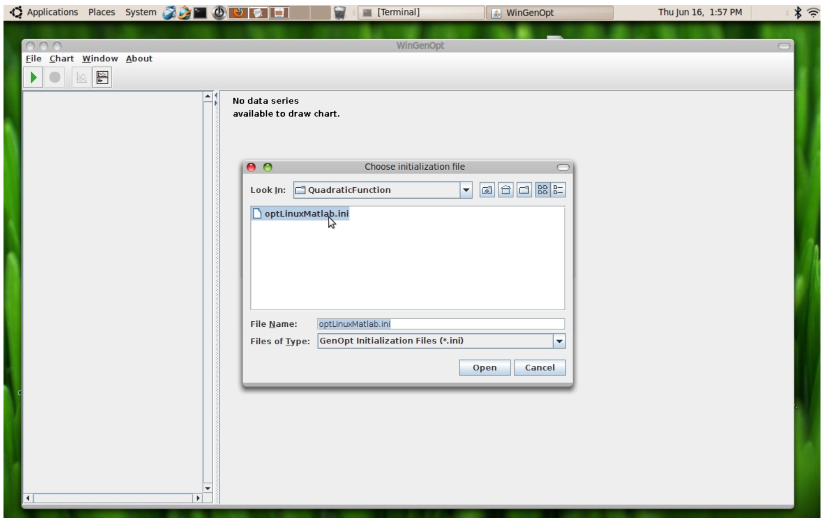
Starting the optimisation

The optimisation can be performed by opening the `*.ini` file with GenOpt. For doing this run GenOpt and select `File -> start`



No data series
available to draw chart.

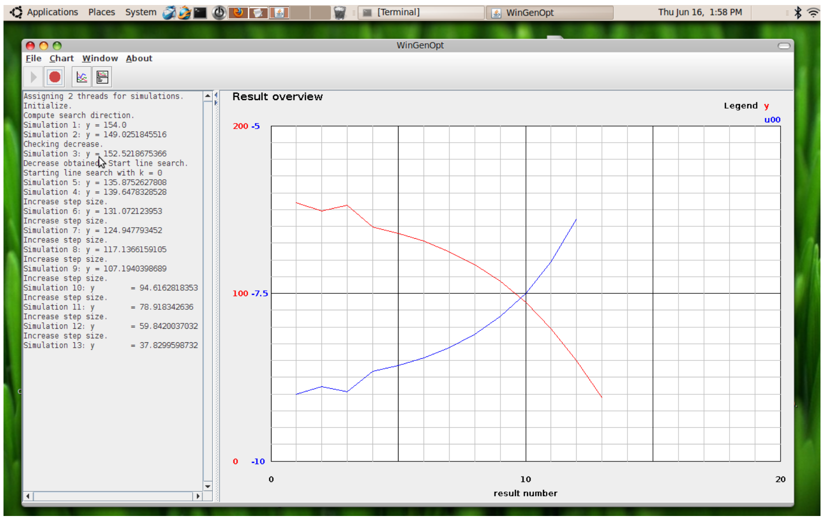
Then select the desired initialization file, in this case it is the `optLinuxMatlab.ini` located in the subfolder `QuadraticFunction`. Then click on open.



No data series
available to draw chart.

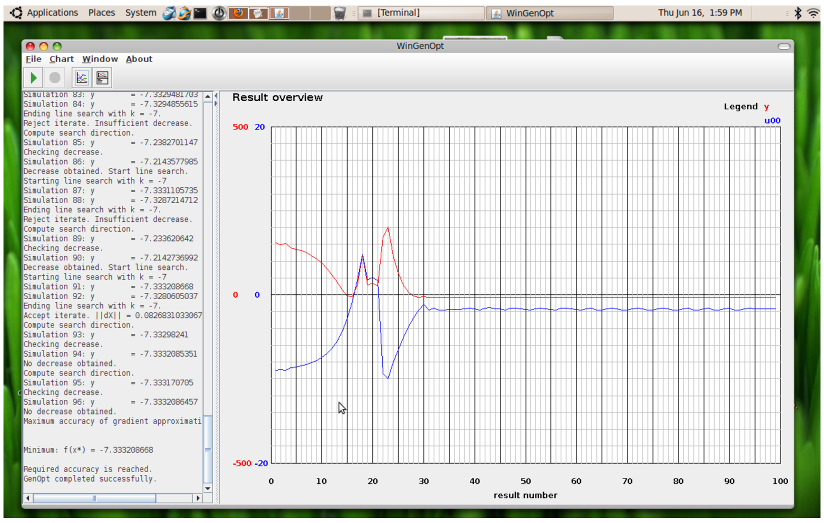
```
Look In:       QuadraticFunction
optLinuxMatlab.ini
File Name:     optLinuxMatlab.ini
Files of Type: GenOpt Initialization Files (*.ini)
```

After clicking on the open button, the optimisation procedure starts.



```
Assigning 2 threads for simulations.
Initialize.
Compute search direction.
Simulation 1: y = 154.0
Simulation 2: y = 149.0251845516
Checking decrease.
Simulation 3: y = 152.5218675366
Decrease obtained. Start line search.
Starting line search with k = 0
Simulation 5: y = 135.8752627808
Simulation 4: y = 139.6478328528
Increase step size.
Simulation 6: y = 131.072123953
Increase step size.
Simulation 7: y = 124.947793452
Increase step size.
Simulation 8: y = 117.1366159105
Increase step size.
Simulation 9: y = 107.1940398689
Increase step size.
Simulation 10: y       = 94.6162818353
Increase step size.
Simulation 11: y       = 78.918342636
Increase step size.
Simulation 12: y       = 59.8420037032
Increase step size.
Simulation 13: y       = 37.8299598732
```

Wait until its termination



```
Simulation 83: y       = -7.3329481703
Simulation 84: y       = -7.3294855615
Ending line search with k = -7.
Reject iterate. Insufficient decrease.
Compute search direction.
Simulation 85: y       = -7.2382701147
Checking decrease.
Simulation 86: y       = -7.2143577985
Decrease obtained. Start line search.
Starting line search with k = -7
Simulation 87: y       = -7.3331105735
Simulation 88: y       = -7.3287214712
Ending line search with k = -7.
Reject iterate. Insufficient decrease.
Compute search direction.
Simulation 89: y       = -7.233620642
Checking decrease.
Simulation 90: y       = -7.2142736992
Decrease obtained. Start line search.
Starting line search with k = -7
Simulation 91: y       = -7.333208668
Simulation 92: y       = -7.3280605037
Ending line search with k = -7.
Accept iterate. ||dX|| = 0.0826831033067
Compute search direction.
Simulation 93: y       = -7.33298241
Checking decrease.
Simulation 94: y       = -7.3332085351
No decrease obtained.
Compute search direction.
Simulation 95: y       = -7.333170705
Checking decrease.
Simulation 96: y       = -7.3332086457
No decrease obtained.
Maximum accuracy of gradient approximati


Minimum: f(x*) = -7.333208668

Required accuracy is reached.
GenOpt completed successfully.
```

The procedure terminates successfully, as can be seen in the left bar. The minimum value of the cost function is -7.33 and more information about the minimisation can be found in the directory `QuadraticFunction`.

How to create your own

In the directories `MatlabGenOPT_Linux` and `MatlabGenOPT_Windows` there are different examples. The one shown in the previous section was `QuadraticFunction`, which was an introductory example for showing how it works. It is possible to explore other directories and have a look to other examples in order to understand better how to perform an optimization with a cost function that has more than one variable (`MultivariableFunction`) or an application to control theory (`Control`). In this case the optimization procedure is employed for finding the "best" tuning of a PI controller that has to control a simple process, modeled with a second order system.

When creating your own application follow these instructions:

1. Create a subfolder and copy into it the files of a working example (e.g. the `QuadraticFunction` one)
2. First of all specify the new parameters and variables of the new cost function. For doing this edit the `InitialisationScriptTemplate.m` file accordingly to the needs of the new problem. Remember that for each variable that can be explored during the optimisation procedure a string identifier like `%u00%` is needed.
3. When all the parameters and variables are defined, modify the `*.m` file that compute the cost function. <u>Just replace the part that compute the old cost function with the new one</u>. Please be careful that all the variables and parameters referring to in the new cost function are defined (within the `InitialisationScript`). <u>Pay attention to the variable that has to be printed in the result file, and also to the string that will be printed beside it.</u>
4. Now the script has to be modified accordingly to the modifications. Be sure that the Matlab path is correct. The path of the directory containing the new cost function has to be updated as well as the name of the new cost function.
5. Modify the `command.txt` file. It has to contain all the variables mentioned in the `InitialisationScriptTemplate` and for each one indicate the minimum and maximum values as well as the starting one (the value from which the initialization procedure will start). Eventually modify the parameters of the optimization algorithm as well as the algorithm (for all this information please refer to the GenOpt User Guide).
6. Verify that the configuration file is the one needed by your OS.
7. Check the initialization file `*.ini` and all files mentioned within it (the template, the input, the log, the output and the command). Before starting the procedure assures that the output delimiter is the same that is printed by Matlab in the `result.txt` file.
8. Open GenOpt, load the `*.ini` file and start the optimization.

System requirements

Before starting be sure that on your system are present:

- A working version of GenOpt
  (Available at http://simulationresearch.lbl.gov/GO/)
- A working version of Matlab with its license file (the examples stored in the folders have been tested with Matlab 7.3.0.298 under Linux and with 7.5.0.342 under Windows)

N.B.

The Matlab control toolbox is needed in order to run the example contained in the `Control` folder.