Analytical Optimizer Manual v1.0

Moji Ghadimi

January 2021

Contents

1 Introduction

Analytical Optimizer 1.0 is a python code that finds the analytical (symbolic) maximum value of a quadratic function of any number of variables boxed in linear constraints. It is an extension to linear programming and is not optimized to be used for linear objective functions.

Another important feature is that it supports having a free constant in the constraints (not more than one for this version) and returns different maximum values for different intervals of values of the constant. The code supports parallel processing using MPI.

2 Extension to other Nonlinear Functions

In principle this program can be used for some nonlinear functions other than quadratic functions. There are two main limiting parts that determines if the program gives an answer for those functions or not.

The first easier hurdle is the ability to take the derivative of the main function with respect to all the variables. Sympy Diff() is a very powerful function that is capable of doing almost any differentiation that can be done by hand.

The second more limiting factor on nonlinearity is the ability of nonlinsolve() function of Sympy to solve the system of equations of the derivatives of the function with respect to all the variables. Obivously for quadratic functions of any number of variables this can be done easily since the derivatives are linear functions of variables.

3 Steps breakdown

The whole procedure can be divided into these broad steps:

- In the first step the code generates an array of all possible combinations of constraints (including not having any constraints).
- In the next step, for each combination of constraints, "intersect()" function solves system of the equations of that combination with solve() function of SymPy package (solution can be underdetermined). Then the solution is plugged into the main function to find the intersection of the main function with that combination of constraints.
- In the next step "optimum()" function calculates partial derivatives of each intersection function in the list with respect to all the variables and sets those to zero to find optimum points and their values (can be a function of the free unknown constant). This is done with diff() and nonlinsolve() functions of SymPy package. If the solution is underdetermind, the program discards that because this means that the solution is a ridge and it manifests itself as a point at some boundary.
- Next the function "feasible()" checks that those optimums are feasible (compatible with constraints). With no constants this is a yes or no question but with a constant the answer might be feasible for some values of the constant and not feasible for some other values. This step returns a feasible interval for the constant if we have a constant.
- In the final step the function max compares all the optimums in their feasible regions to find the maximum for different intervals of the constant (if there is no constant it just returns the single optimum value).

4 Installation

To use the code Python 3 needs to be installed along with these packages:

- sympy==1.5
- tabulate==0.8.6

- antlr4-python3-runtime==4.7.2
- mpi4py==3.0.3 (only if you have MPI setup and want to run in parallel)

To run the code simply download all the files and folders from https://github.com/Moji131/Analytical-Optimizer and run main.py.

5 Tutorial

5.1 Installation for beginners

One way of installing Python 3 and using the code is explained here:

- download files and folders from https://github.com/Moji131/Analytical-Optimizer and place them in a convenient place and name it "Analytical Optimizer".
- Download and install latest version of Python 3 from here and install with default settings.
- Download "Community" version of PyCharm editor here and install with default settings.
- Open PyCharm and create a new project using this tutorial but use theses settings in the project window:
  - For "location" go to the location of "Analytical Optimizer" that you saved the codes.
  - Under "Project interpreter" select "Existing interpreter at the bottom instead of "New environment using Virtualenv".
- Use this tutorial to add the required packages listed above or use the command below with the requirements.txt file provided with the code.

```
pip install -r /path/to/requirements.txt
```

- After installing packages, in the project window on the left double click on "main.py" to open it.
- For the first run you need to right click on the "main.py" tab at the top and click ▷ Run 'main'. For the next runs you can just use the play icon ▷ at the top.

5.2 Basic usage guide

5.2.1 Running

The main file to run is "main.py".

5.2.2 Input

To define your own input function and constraints you need to open "input.py" and edit following inputs:

- Function: Latex script of your input function.
- constraints: An array that contains all the constraints in Latex format. Right-hand-side of inequalities must always be zero.
- Constants: An array that contains list of your constants (empty array if there is no constant).
- Constants_constraints: An array that contains constraints on constants (empty array if there is no constant).

An example input for a two dimensional quadratic function with a ridge:

```
### simple 2D quadratic function with a ridge

####### Input Function and Constraints ####################
############################################################

# main function. A string in latex format.
Function = '2-(x_1+y_1)^2'

# constraints on independent variables.
# Add a constraint to the list using Constraints.append() command.
# The constraint needs to be a string in latex format.
# Right hand side of inequality must always be zero.
Constraints = []
Constraints.append('x_1 \geq 0' )
Constraints.append('x_1 - 1 \leq 0')
Constraints.append('y_1 \geq 0')
Constraints.append( 'y_1 - 1 \leq 0')
Constraints.append('x_1 + y_1 - s \geq 0' )

# list of constants. The constant name can only be o, s or m.
# Only one constant is supported for this version.
Constants = [s]

# constraints on constants
Constants_Constraints = []
Constants_Constraints.append('s \geq 0')
Constants_Constraints.append('s - 2 \leq 0')
```

There are also other more advanced options like saving output of each step or printing result of intermidiate steps can be set in the file run_options.py.

5.2.3 Output

After running is finished, two tables will be saved to files in the folder "output":

- The main table, 0_max_out_1.txt, shows the main result by listing maximum values of the function and corresponding input values for different

intervals of the value of the constant (prints one point and one maximum value if there is no constant).

- The second table, 0_max_out_2.txt, shows the combination of constraint from which each maximum is found.

6 Run in parallel

The program has the ability to be run in parallel. To do this, MPI must be installed. To run in parallel go to the commandline and navigate to the code folder (no need if you are using commandline in your editor) and then you need to run the command below:

```
mpirun.openmpi -n 4 python3 main.py
```

-n determines the number processors to be used. On a single computer with a cpu with 4 cores "-n 4" is the fastest.