Optimiser.py: a Python code for global optimisation of constrained objective functions using the stochastic Simulated Annealing Algorithm

The python code in this repository contains the Simulated Annealing Algorithm as designed by Goffe *et al.* (J. Econometrics 60(1994):65-99),

http://econpapers.repec.org/software/wpawuwppr/9406001.htm

1 Introduction

The simulated annealing algorithm (SAA) is a stochastic optimisation method designed to find the global minimum of a constrained function. A comprehensive (but not extensive) documentation of the SAA may be found in Chapter 6 of the 'Integrated Multi-Fluid Flows and Thermodynamic Models' report.

2 Quick-Run

The module can perform global optimisation using the SAA in two classes of cases:

a) 'Benchmarks' (there is a list of test-cases in the "Benchmarks.in" file);

b) 'Problem' (i.e., case problem that you want to find the global minimum/maximum).

The current code is run by either,

python Optimiser.py SAA Benchmarks < N >

or

python Optimiser.py SAA Problem < Name >.

The former command line performs the SAA method in a set of benchmark test-cases (listed in the 'Benchmarks.in' file and included in the 'Tests' directory), where < N > is either the test-case number (currently 1 to 5) or 'All' (in which all 5 test-cases are performed at once). The later command line performs the SAA method in the chosen problem, where < Name > stands for the problem test-case name.

3 Code Design

The algorithm is divided into 8 code files:

'Optimiser.py' is the "main" code that contains the preamble of the algorithm and reads and interpret the command line for the code. As it stands, there are 3 distinct configurations:

a) Run the SAA in a function (e.g., 'test1'):

python Optimiser.py SAA Problem test1

b) Run one benchmark test-case. Currently there are 5 test-cases listed in the 'Benchmarks.in' file and coded in the 'BenchmarkTests.py' file:

python Optimiser.py SAA Benchmarks 1

This will find the global minimum of the Judge function (Test_1).

c) Run all benchmark test-cases at once:

python Optimiser.py SAA Benchmarks All

This will find the global minimum of all functions contained in the 'Benchmarks.in' file and coded in the 'BenchmarkTests.py' file.

The 'Optimiser.py' will read the input task and will either perform a global optimisation in a prescribed function given by the operator, or seek for the global minimum of 5 standard optimisation test-cases.

'RandomGenerator.py' contains a simple random number generator (between 0.0 and 1.0) with Gaussian distribution.

'SA_IO.py' contains functions that will read the standard input files as,

< Function >.sa

This '*.sa' file **must** contain all cooling schedule parameters, including:

a) dimensionality of the problem;

b) type of problem (i.e., maximisation or minimisation);

c) temperature parameter;

d) upper and lower bounds of the problem's solution-coordinate;

e) reducing factor of the temperature parameter;

f) maximum number of cycles;

g) maximum number of iterations before the temperature parameter is reduced;

h) main diagonal of the stepping matrix;

i) parameter for controlling the stepping matrix;

j) maximum number of evalutions of the objective function;

k) minimum acceptable discrepancy between solution-coordinates and functions;

l) Initial guess of the solution-coordinate.

Examples of cooling schedules, as '*.sa' files, may be found in the 'Tests' directory (which contains all benchmark test-cases).

'SAA_Tools.py' contains the global variables used throughout the code.

'SA_Print.py' contains functions that output solutions into a file named as either (see function 'Output' in the 'SA_IO.py' file),

Benchmarks_SAA_TestCase_ $<$ N $>$.out (e.g., Benchmarks_SAA_TestCase_3.out),

or

Problem_SAA_ $<$ Problem_Name $>$.out (e.g., Problem_SAA_ $<$ Problem_Flash-C1C2.out),

where $<$ N $>$ is either the number of the benchmark test-case (defined in the 'Benchmarks.in' file) or 'All' (for running the algorithm in all test-cases). $<$ Problem_Name $>$ stands for the problem case you wish to minimise.

'SpecialFunctions.py' contains a few functions necessary for the SAA to be smoothly performed. In particular, the 'Envelope_Constraints' was designed to ensure that an initial feasibility analysis (for Thermodynamic problems) is undertaken, however it should be further integrated the thermodynamic module to ensure it work properly. Ideally, the feasibility test should ensure:

a) Solution-coordinates are bounded, i.e.,

$$\mathrm{LB}_i < \mathrm{X}_i < \mathrm{UB}_i,$$

where LB and UB are lower- and upper-bounds defined in the cooling schedule.

b) Summation of $(N_c - 1)$ elements of the solution-coordinate that relates to thermodynamic composition (i.e., mole/mass fractions) are also bounded, i.e.,

$$0 < \sum_{i=1}^{N_c-1} X_i^{(j)} < 1,$$

where j is the phase (i.e., = L or V in a VLE problem).

c) Compositions of the second phase of the solution-coordinate that relates to thermodynamic composition (i.e., mole/mass fractions) are also bounded, i.e.,

$$X_i^{(V)} = \frac{Z_i - LX_i^{(L)}}{1-L}$$

or

$$X_i^{(L)} = \frac{Z_i - (1-L)X_i^{(V)}}{L},$$

subjected to,

$$0 < X_i^{(j)} < 1, \quad \text{where } i = 1, \cdots, N_c \text{ and } j = L, V.$$

'ObjectiveFunction.py' should link the SAA with the problem that needs to be optimised.

'SimulateAnnealing.py' is the main optimisation function – 'SimulatedAnnealing' that,

a) Reads the cooling schedule of the benchmark test-cases ('*.sa' contained in the 'Tests' directory) or the problem;

b) Undertakes the algorithm (contained in the 'ASA_Loops' function);

c) Print out the diagnostic of the test-case/problem.