Octeract Engine

User manual

June 12, 2020

Contents

User Manual
Octeract Engine v1.06.10

June 12, 2020

1 Introduction

Octeract Engine is a Deterministic Global Optimisation (DGO) solver which solves general nonlinear problems to guaranteed global optimality. The general mathematical description is encapsulated in problem P:

$$\begin{aligned} P: \min_{\substack{\boldsymbol{x}\in X\\ \boldsymbol{y}\in Y}} & f(\boldsymbol{x},\boldsymbol{y}) \\ \text{s.t. } & \boldsymbol{g}(\boldsymbol{x},\boldsymbol{y}) \leq \boldsymbol{0} \\ & \boldsymbol{h}(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{0} \\ & X = [\boldsymbol{x}^L, \boldsymbol{x}^U] \subset \mathbb{R}^n \\ & Y = \{0,1\} \end{aligned}$$

The solver can solve *any* nonlinear function to global optimality, as long as that function has an analytical form that does not include differential expressions or integrals. For instance,

$$\min_{x\in[-1,1]} \sqrt{x}, \quad \min_{x\in[-1,1]} \left| -\log\left(\frac{1}{\tan(x)^{0.85}}\right)\right|, \quad \min_{x\in[-1,1]} \sqrt{\max(x^{-1/2},1)},$$

are valid inputs, whereas:

$$\min_{x\in[-1,1]} \Gamma(x)$$

is not valid input because the expression for the Gamma function is an integral.

> **Note**
>
> Two important exceptions to the analytical expression rule are $\min(f(x),g(x))$ and $\max(f(x),g(x))$, which are fully supported.

1.1 Solver highlights

No starting points

Users do not need to define starting points when using Octeract Engine. The solver will locate feasible solutions automatically as part of its branch-and-bound search.

Guaranteed calculations

Octeract Engine uses a combination of **Interval Arithmetic** and **Arbitrary Precision Arithmetic** to guarantee the correctness of all internal calculations. The combination of these two paradigms enables the solver to handle extremely challenging numerics. This is very important in Deterministic Global Optimisation, as our algorithms are always at risk of incorrectly eliminating the global solution due to bad floating-point calculations. However, Octeract Engine relies on third-party solvers in order to solve NLP and LP subproblems, meaning that it is possible for the solver to return an incorrect result due to numerical error from third-party software. In the majority of cases, the solver compensates for such errors using conservative tolerances for external calculations, but it is important for users to be aware that the end result is still subject to third-party floating-point errors.

Symbolic engine

Octeract Engine comes equipped with a full-fledged symbolic engine designed for optimisation mathematics, which allows it to perform symbolic manipulation of user input to an unprecedented degree.

Octeract Reformulator

The Octeract symbolic engine for optimisation math is a standalone product, a.k.a. the **Octeract Reformulator**. The Reformulator is an extremely flexible tool which allows users to script complicated reformulations, create reformulation libraries and apply the same reformulations to other problems, or to share them with collaborators. Octeract Engine is equipped with a special version of the Reformulator, tuned for global optimisation problems.

Because of the Reformulator, users should never need to reformulate their problems manually to make them solvable by Octeract Engine - the solver handles bad math automatically.

Accessible parallelism

Octeract Engine can be run in parallel using **MPI** out of the box, for single machines and computing clusters alike. Check out Section 5 for a detailed overview.

2 Solving your first problem

2.1 Using the Command Line Interface

The solver can be invoked from Windows Powershell or a Linux terminal using the following syntax:

```
octeract-engine [problem_file]
```

The solver supports direct input in ASL (.nl), MPS (.mps), and LP (.lp) formats. It also has built-in support for the Pyomo and AMPL modeling environments. Example input files are included in the installation folder under `examples`. Let's begin by solving test problem `ex2_1_1.nl`. Invoke the solver using the following command:

```
octeract-engine ex2_1_1.nl
```

You should see the following ouput:

```
-------------------------------------------------------------------------------------------
Iteration            GAP               LLB          BUB        Pool      Time     Mem
-------------------------------------------------------------------------------------------
       11    5.000e-11 (  0.00%)   -1.700e+01   -1.700e+01        3      0.0s    91.0MB

Objective value at global solution: -1.700e+01
```

By default, the solver outputs the following information:

- Number of iterations (in this case 11).
- Absolute and relative optimality gap (here $5\text{x}10^{-11}$ and 0.00 % respectively).
- Least Lower Bound (LLB). This is the smallest lower bound throughout the branch-and-bound tree.
- Best Upper Bound (BUB). This is the best local optimum that has been located so far.
- Pool. This is the number of nodes in the branching pool.
- Time. This is the real time (*not* CPU time) spent in branch-and-bound (in seconds).
- Memory. The amount of RAM that the solver is currently taking up.

By default, the solver will automatically write an `.octsol` solution file in the system's `LOCAL_TEMP`. This directory can be set explicitly using the `-s` flag:

```
octeract-engine ex2_1_1.nl -s MY_SOLUTION_PATH
```

Detailed solution information can be accessed by inspecting the solution file which produces the following output:

```
=======================================
*************** Solution ***************
=======================================
- Local Engine : ipopt
- Local Eng. Status : 0
- Feasible : Yes
- Problem type : UB
- Found in node : 20
- Objective value : -16.9999999999499778
- Model name : ex2_1_1
- Problem structure : QP
- DGO exit status : Solved_To_Global_Optimality
- Solving time (s) : 0.078000
- Nu cores : 1
- nu vars : 5
  * nu nlvars : 5
  * nu linear vars : 0
  * nu binary vars : 0
- nu cons : 1
  * nu linear cons : 1
  * nu fixed cons : 0
  * nu quadratic cons : 0
  * nu general nl cons : 0
- Solution vector :
x1 : 0.9999999999998276
x2 : 0.9999999999998215
x3 : 0.0000000000002222
x4 : 0.9999999999998114
x5 : 0.0000000000002105
- Lagrange multipliers :
con1 : 0.0000000000099999
x1 : -57.9999999999529479
x2 : -56.0000000000025153
x3 : 45.0000000009757386
x4 : -53.0000000000258993
x5 : 47.5000000009467271
=======================================
```

That's it! Congratulations, you just solved your first problem using Octeract Engine!

2.2 Using Python

Check the Python API documentation for simple examples to get started.

2.3 Using C++

Check the C++ API documentation for simple examples to get started.

3 Solver flags

3.1 Input flags

The only mandatory argument for the `octeract_solver` executable is the model file. The executable also accepts the following optional arguments:

- `--help` : Show help.
- `-o, --option` : Location of an `octeract.opt` options file.
- `-s, --solution` : Directory where solution files should be saved.
- `-t, --timeout` : Solver timeout in seconds. A solution file is written in the solution directory automatically on timeout.
- `-n, --core` : Number of MPI processes to be created (equivalent to number of CPU cores to use if n$\leq$total_cores).
- `--hostfile` : Location of the MPI hostfile (if running in a cluster).

3.2 Exit flags

The solver can print the following exit status messages to the .octsol file upon termination:

- `Global_Solution_Status_Undefined`
- `Solved_To_Global_Optimality`
- `Proved_Global_Infeasibility`
- `Found_Solution_For_CS_Problem` (CS stands for constraint satisfaction)
- `Timeout_With_Feasible_Solution`
- `Timeout_Without_Feasible_Solution`
- `Interrupt_With_Feasible_Solution`
- `Interrupt_Without_Feasible_Solution`
- `Catastrophic_Error`
- `Parsing_Error`

4 Solver Options

The solver accepts options through an options file named `octeract.opt`. By default, the solver will attempt to locate an options file in the current directory, but this can be overridden using the `-o` flag. All options are case-sensitive and are set using the following syntax:

```
OPTION_NAME = option_value
```

For instance, `USE_OBBT = true` will set that option to true. An example custom options file would look like the following:

```
# MY OPTIONS FILE
CONVERGENCE_TOLERANCE = 1.e-1
INFINITY = 1.e6
MAX_SOLVER_TIME = 60
USE_OBBT = false
```

4.1 Convergence settings

`MAX_SOLVER_ITERATIONS`

Type : int
Default Value: 2.e9
Range: 1 - 2.e9

Sets the maximum number of solver iterations **in serial mode** (this option is ignored in parallel mode).

`MAX_SOLVER_TIME`

Type : int
Default Value: 2.e9
Range: 1 - 2.e9

Sets a timeout for the solver in seconds (elapsed real time).

`CONVERGENCE_TOLERANCE`

Type : double
Default Value: 1.e-3
Range: 0 - 1.e308

Sets a tolerance ε to determine whether the algorithm has converged to global optimality. The solver will terminate if either of the following conditions is satisfied:

- $BUB - LLB \leq \varepsilon$ (absolute optimality gap)
- $(BUB - LLB) \leq \varepsilon |LLB|$ **and** $(BUB - LLB) <= \varepsilon |BUB|$ (relative optimality gap)

4.2 Domain reduction

USE_OBBT

Type : bool
Default Value: true
Available options: true, false

Enables or disables optimality-based bounds tightening (OBBT). Although OBBT can provide amazing variable bounds, it is very expensive and scales badly with the number of variables. Recommended for small to mid-size problems, especially if a commercial LP solver is available. If running in MPI mode with significant computing power, OBBT becomes viable for large problems as well.

MAX_OBBT_ITERATIONS

Type : int
Default Value: 1
Range: 0 - 2.e9

Maximum number of OBBT passes.

MAX_OBBT_DEPTH

Type : int
Default Value: 2.e9
Range: 0 - 2.e9

Sets the maximum depth in the branch-and-bound tree to apply OBBT.

4.3 General solver settings

OUTPUT_FREQUENCY

Type : int
Default Value: 2.e9
Range: 1 - 2.e9

The solver will print output every `OUTPUT_FREQUENCY` iterations.

OUTPUT_TIME_FREQUENCY

Type : int
Default Value: 1
Range: 1 - 2.e9

The solver will print output every `OUTPUT_TIME_FREQUENCY` seconds.

PRESOLVE

Type : bool
Default Value: true
Available options: true, false

Enable or disable the presolver.

UB_FREQUENCY

Type : int
Default Value: 4
Range: 1 - 2.e9

Sets how often the solver should solve local optimization problems to find upper bounds. Small values for this option can be helpful in problems where finding feasible upper bounds is challenging. For instance, UB_FREQUENCY = 1 will instruct the solver to calculate an upper bound on every node it processes, thus increasing the probability that a feasible solution is going to be located.

PRINT_SOLUTION

Type : bool
Default Value: false
Available options: true, false

Sets whether to print the solution object on the screen or not, upon termination.

INFINITY

Type : double
Default Value: 1.e8
Range: 0 - 1.e308

Sets the default bound for unbounded variables. Global optimization requires all variables to be bounded to begin the solving process. If the user does not provide bounds for a variable, its bounds will be automatically set to ±INFINITY. The default value is quite conservative - reducing this number can greatly improve performance.

4.4 Third-party solvers

NLP_SOLVER

Type : string
Default value: IPOPT
Available options: IPOPT

Sets the solver which will be used to solve non-linear local optimization problems.

LP_SOLVER

Type : string
Default: OSICLP
Available options: IPOPT, OSICLP, OSICBC

Sets the solver which will be used to solve linear problems.

MILP_SOLVER

Type : string
Default Value: OSICBC
Available options: OSICBC

Sets the solver which will be used to solve mixed-integer linear problems.

4.5 Tolerances

BOUND_VIOLATION_TOLERANCE

Type : double
Default Value: 1.e-8
Range: 0 - 1.e308

Bound violation tolerance is used to confirm whether solutions returned by third-party solvers are indeed feasible within that tolerance. Relaxing this tolerance can help in problems where finding a feasible solution is difficult, or where the global solution is eliminated because of slight constraint violations.

INTEGRALITY_VIOLATION_TOLERANCE

Type : double
Default Value: 1.e-3
Range: 0 - 1.e308

Integrality violation tolerance is the acceptable violation before the solver determines that a variable is no longer an integer. Relaxing this tolerance can prevent integer solutions from being fathomed because of small constraint violations.

CONSTRAINT_VIOLATION_TOLERANCE

Type : double
Default Value: 1.e-5
Range: 0 - 1.e308

This tolerance determines the acceptable constraint violation when applying domain reduction techniques. Relaxing this tolerance can help in problems where finding a feasible solution is difficult, or where the global solution is eliminated because of slight constraint violations.

NUMBER_COMPARISON_TOLERANCE

Type : double
Default Value: 1.e-6
Range: 0 - 1.e308

This tolerance is used to determine whether two floating point numbers are the same. Its use in the solver is context sensitive, as we always use outward rounding to ensure that solutions are not incorrectly eliminated. Making this tolerance smaller can accelerate convergence, as the solver becomes more aggressive in domain reduction, but increases the risk of eliminating otherwise acceptable solutions. Relaxing this tolerance slows down convergence, as the solver adopts a more conservative approach.

LLB_TOLERANCE

Type : double
Default Value: 1.e-3
Range: 0 - 1.e308

This tolerance is used to compare solutions of upper bounding problems and lower bounding problems. These two numbers are typically results of two different local optimization problems, potentially using two different third-party solvers, and are thus prone to high numerical error. Tightening this tolerance can accelerate convergence, but it can increase the risk of eliminating the global solution because of accumulation of error.

BIG_COEFF_TOLERANCE

Type : double
Default Value: 1.e9
Range: 0 - 1.e308

Sets the maximum acceptable absolute value of a coefficient in the lower bounding problem. If a coefficient is greater than this number, the solver will use other, less effective methods instead of solving the lower bounding LP. This option rarely needs to be changed because the coefficients are naturally improved through branching. However, in rare cases where the problem is numerically well-behaved but has a large gap that is not being closed, increasing this tolerance will force the solver to solve these ill-posed lower bounding problems.

4.6 Variable Branching

VARIABLE_SELECTION_HEURISTIC

Type : string
Default Value: OS1
Available options: OS1, OS2, OS3, OS4

Sets which heuristic should be used to select branching variables in serial mode. The default is OS1, which is a lightweight strategy with good performance across the board. OS2-4 are increasingly more expensive strategies (with OS4 being the most expensive one), which can yield great results in small to mid-size problems.

5 Running in parallel mode using MPI

The Message Passing Interface (MPI) is a framework for distributed computing, originally designed for use in supercomputers. Octeract Engine is distributed with MPI, and can be used in parallel mode out of the box.

5.1 Running on a single machine

The syntax to invoke MPI on a single machine is the following:

```
octeract-engine -n [number_of_processes] [problem_file]
```

The solver will now generate and run `n` processes in parallel. It is highly recommended to use at most as many processes as there are physical cores in your system.

5.2 Running on a computer cluster

Octeract Engine should run on any Linux cluster out of the box. The syntax to invoke MPI on a distributed architecture is very similar to single machine MPI mode:

```
octeract-engine -n [nu_processes] --hostfile [hostfile] [problem_file]
```

The `hostfile` is required by MPI, as it contains the IP addresses of all the machines that the solver can connect to. A sample `hostfile` could look like this:

```
10.200.300.01 : 32
10.200.300.02 : 8
10.200.300.45 : 2
10.200.300.32 : 12
```

This file contains two columns delimited by a colon. The IP addresses of the available machines are listed in the first column. In the second column, the user can optionally declare the maximum number of cores that can be used in each machine. If the column is ommited, then MPI will use all cores by default. In this example, the first machine is allowed to utilise up to 32 cores.

> **Note**
>
> Octeract Engine will spawn as many processes as the user requests on startup. If that number is smaller than all the processors specified in the hostfile, some machines will not be used at all.