Octeract Engine

User manual

January 17, 2020

Page intentionally left blank.

Contents

1 Introduction **3**
- 1.1 Solver highlights . . . 4
  - 1.1.1 No starting points . . . 4
  - 1.1.2 Guaranteed calculations . . . 4
  - 1.1.3 Symbolic engine . . . 4
  - 1.1.4 Accessible parallelism . . . 4
  - 1.1.5 Cross platform . . . 5

2 Installation of Octeract Engine for Linux and macOS **5**
- 2.1 Prerequisites . . . 5
- 2.2 Installation using cross-platform Python script . . . 5
  - 2.2.1 Installation . . . 5
  - 2.2.2 Starting and stopping the Octeract Engine Container . . . 6
  - 2.2.3 Uninstalling the Octeract Engine . . . 6

3 Installation of Octeract Engine for Windows 10 **6**
- 3.1 Prerequisites . . . 6
- 3.2 Installation using the Graphical User Interface (GUI) . . . 6
  - 3.2.1 Installation . . . 6
  - 3.2.2 Starting and stopping the Octeract Engine Container . . . 7
  - 3.2.3 Uninstalling the Octeract Engine . . . 7

4 Solving your first problem **7**
- 4.1 Using the Linux terminal or Windows Powershell . . . 7
- 4.2 Using Python and the Octeract Shell . . . 9

5 Solver flags **12**
- 5.1 Input flags . . . 12
- 5.2 Exit flags . . . 13

6 Solver Options **13**
- 6.1 Third-party solvers . . . 13
- 6.2 Convergence settings . . . 14
- 6.3 Heuristics . . . 14
  - 6.3.1 Branching . . . 14
  - 6.3.2 Multistart . . . 15
  - 6.3.3 Bounds tightening . . . 15
- 6.4 Tolerances . . . 16
- 6.5 General solver settings . . . 17

7 Running in parallel (MPI) mode **18**
- 7.1 Running on a single machine . . . 18
- 7.2 Running on a cluster . . . 18

User Manual
Octeract Engine Beta v1.01.17

January 17, 2020

!Developer release notes for Octeract Engine Beta

The focus for this release has been on support for directly reading and writing LP and MPS formats for linear problems; improving the performance and robustness of our presolve algorithms; the introduction of several new primal heuristics for MINLPs; intelligent variable branching heuristics; the implementation of novel cutting plane techniques for mixed-integer problems; and specialised reformulation procedures for problems with quadratic structure. User feedback is extremely important to us, so we would like to encourage all users to share their experience and to report any issue encountered at **www.octeract.com/bug-reports/**. Enjoy optimising!

The Octeract Dev team.

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 engine 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.

!Dev Beta note

Octeract Engine can solve discontinuous problems without reformulations or tweaks. The current version can resolve discontinuities for two, and sometimes three levels of functional compositions, i.e., $f(g(h(x)))$. Support for infinitely nested discontinuities will be added in the future.

1.1 Solver highlights

1.1.1 No starting points

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

1.1.2 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 engine 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, the engine relies on third-party solvers in order to solve NLP and LP subproblems, meaning that it is possible for the engine to return an incorrect result due to numerical error from third-party software. In the majority of cases, the engine 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.

1.1.3 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 engine handles bad math automatically.

1.1.4 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 7 for a detailed overview.

1.1.5 Cross platform

Octeract Engine is deployed as a Docker container, which makes it highly portable. The engine has been tested on many different versions of Ubuntu, CentOs, macOS, Windows 7, and Windows 10.

2 Installation of Octeract Engine for Linux and macOS

Getting started guide for Octeract Engine Beta.

2.1 Prerequisites

1. Internet connectivity is required throughout the procedure.
2. Download and install **Docker** for your OS:
   - Mac: **https://docs.docker.com/docker-for-mac/install/**
   - Linux: **https://docs.docker.com/install/linux/docker-ce/ubuntu/**
3. Download and install Python 3 for your OS from **https://www.python.org/downloads/**
4. The rest of this section assumes familiarity with shell environments.

2.2 Installation using cross-platform Python script

2.2.1 Installation

In this section instructions are provided for the installation of Octeract Engine using the cross-platform Python script.

1. Download the `octeract-installer.zip` and `octeract-engine.tgz` files from the Downloads page **https://octeract.com/downloads/**
2. Extract (unzip) the contents of the `octeract-installer.zip` file to a selected location in your PC.
3. In a terminal window, navigate to the folder "installer" and run

   ```
   python3 octeract_installer.py
   ```

   from the location where the `octeract_installer.py` has been saved and follow the on-screen instructions.

   Note

   During installation, if you are unsure about the tmp and/or mount directory you are advised to use the defaults.

4. You are advised to append your system `PATH` so that the octeract engine can be invoked throughout your system.

   Note

   The installation reports the installation directory upon completion which will be referred to as `SAVE_LOCATION` throughout this section.

2.2.2 Starting and stopping the Octeract Engine Container

To start/stop the Octeract Engine container you can run:

```
octeract_launcher.py
```

if you chose to append your system `PATH` or

```
python3 SAVE_LOCATION/octeract-engine/octeract_launcher.py
```

if you chose not to append your system `PATH`.

2.2.3 Uninstalling the Octeract Engine

You can simply uninstall the Octeract Engine by running the command:

```
python3 octeract_installer.py --uninstall
```

3 Installation of Octeract Engine for Windows 10

Getting started guide for Octeract Engine Beta cross-platform edition

3.1 Prerequisites

1. Internet connectivity is required throughout the procedure.
2. Only Windows 10 **Pro** and **Enterprise** editions are currently supported.
3. Download and install **Docker** for Windows from **https://docs.docker.com/docker-for-windows/install/**
4. Download and install Python 3 for your OS from **https://www.python.org/downloads/**
5. Download Octeract DGO Engine Docker archive from **https://www.octeract.com/downloads/**
6. The rest of this section assumes some familiarity with Powershell.

3.2 Installation using the Graphical User Interface (GUI)

3.2.1 Installation

In this section the user is guided through the installation of the Octeract Engine Beta cross-platform edition using the Graphical User Interface (GUI).

1. Download the `octeract-installer.zip` and `octeract-engine.tgz` files from the Downloads page **https://octeract.com/downloads/**
2. Extract (unzip) the contents of the `octeract-installer.zip` file to a selected location in your PC.
3. Open the folder "installer" and double click `octeract_installer.py` to launch the installer wizard.
4. Read and accept the license agreement by clicking "I Agree".
5. If Docker is already installed and running check the "I have installed and started Docker", otherwise please see section 3.1 and start Docker before you proceed to the next step.
6. Click Browse and locate the file "octeract-engine.tgz" that was downloaded on your PC. Click Next.

7. Click Browse and select the installation folder of your choice. Click Next.
8. (Optional) Check the "Add to `PATH`" box before clicking Next if you want to automatically update your `PATH` variables to include the installation location.
9. Click Browse and choose the mount directory that the Octeract Engine Docker will have access to. To choose the default location, just click "Select Folder" without explicitly selecting a folder. Click Next.
10. Click Browse and choose the temporary directory that the Octeract Engine Docker will have access to. To choose the default location, just click "Select Folder" without explicitly selecting a folder. Click Next.

    Note

    If you are unsure about the temporary and/or mount directory you are advised to use `C:\\Users\YOUR_USERNAME\AppData\Local\Temp` and `C:\\Users\YOUR_USERNAME` respectively.

11. Wait for the installation to complete and click Next.
12. Octeract Engine has been successfully installed. Click Done and close the installation wizard

3.2.2 Starting and stopping the Octeract Engine Container

To start/stop the Octeract Engine container you can follow the steps:

1. Navigate to the location where the folder "octeract-engine" was installed.
2. Double click the `octeract_launcher.py` file. This will launch a PowerShell session.
3. Choose and type the appropriate number and hit Enter. Docker may request your user password to run.

3.2.3 Uninstalling the Octeract Engine

You can simply uninstall the Octeract Engine by running the command in a PowerShell terminal:

```
py octeract_installer.py --uninstall
```

4 Solving your first problem

4.1 Using the Linux terminal or Windows Powershell

The engine can be invoked from a Linux terminal or Windows Powershell (replace `python3` with `py` in the below examples) using the following syntax:

```
python3 SAVE_LOCATION/octeract-engine/octeract-engine.py [problem_file]
```

The engine supports input in ASL (.nl), Pyomo (.py), and AMPL (.mod) formats. Example input files are included in the installation folder under `SAVE_LOCATION/app/octeract/examples`.

Let's begin by solving test problem `ex2_1_1.nl`.

Invoke the engine using the following command:

```
python3 SAVE_LOCATION/octeract-engine/octeract-engine.py \
    SAVE_LOCATION/octeract-engine/app/octeract/examples/data/ex2_1_1.nl
```

Advanced

If you wish to invoke `octeract-engine.py` from any directory in your system, first confirm it is in the system `PATH`, then run:

```
octeract-engine.py /app/octeract/examples/data/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
Solution file written to /tmp/ex2_1_1.octsol
```

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:

```
python3 SAVE_LOCATION/octeract-engine/octeract-engine.py \
    SAVE_LOCATION/octeract-engine/app/octeract/examples/data/ex2_1_1.nl \
    -s MY_SOLUTION_PATH
```

Detailed solution information can be accessed by inspecting the solution file:

```
cat /tmp/ex2_1_1.octsol
```

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!

4.2 Using Python and the Octeract Shell

Octeract Shell is a Python interface for user friendly interaction with Octeract products. The shell can be invoked from a Linux terminal using the following syntax:

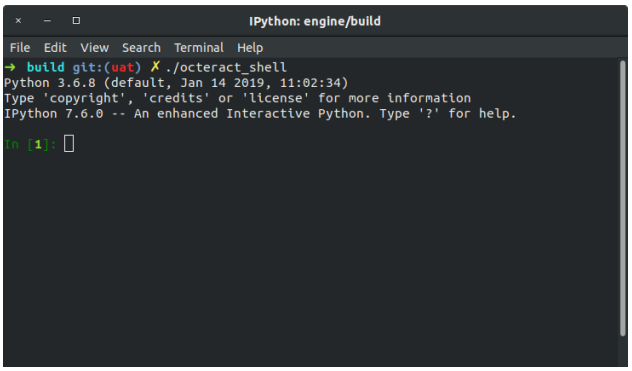
```
SAVE_LOCATION/octeract-engine/octeract_shell.py
```

Advanced

Confirm that octeract_shell is in the system `$PATH` (`$ which octeract_shell` should not be null), and simply run:

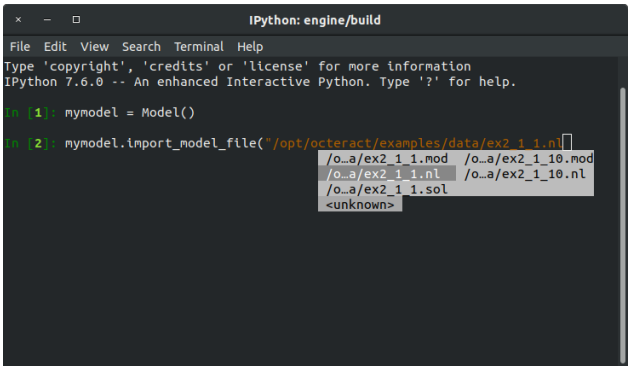
```
octeract_shell.py
```

You should now see this in your terminal:



Let's begin by solving test problem `ex2_1_1.nl`. We need to create a new Model object in the shell, and then import our file into that model object. This is done using the following two commands:

```
mymodel = Model()
mymodel.import_model_file('the/path/to/my/nl_or_mod_or_pyomo_file')
```



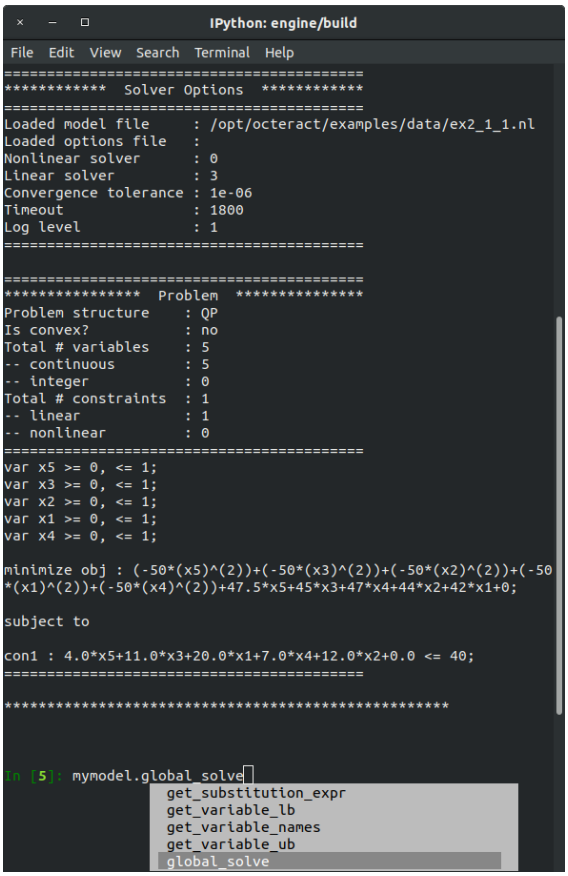
The engine supports input in ASL (.nl), Pyomo (.py), and AMPL (.mod) formats. Example input files are included in the installation folder under `SAVE_LOCATION/octeract-engine/app/octeract/examples`.

Advanced

All commands run in Octeract Shell can be run directly in a Python3.6+ script by importing libocteract:

```
from libocteract import *
```

Now that the problem is loaded into the shell, we can inspect it by running `print(mymodel)`, and solve it using `mymodel.global_solve()`:



By default, the solver will generate a unique name for imported model and write an `.octsol` solution file in the user's `LOCAL_TEMP`.

Advanced

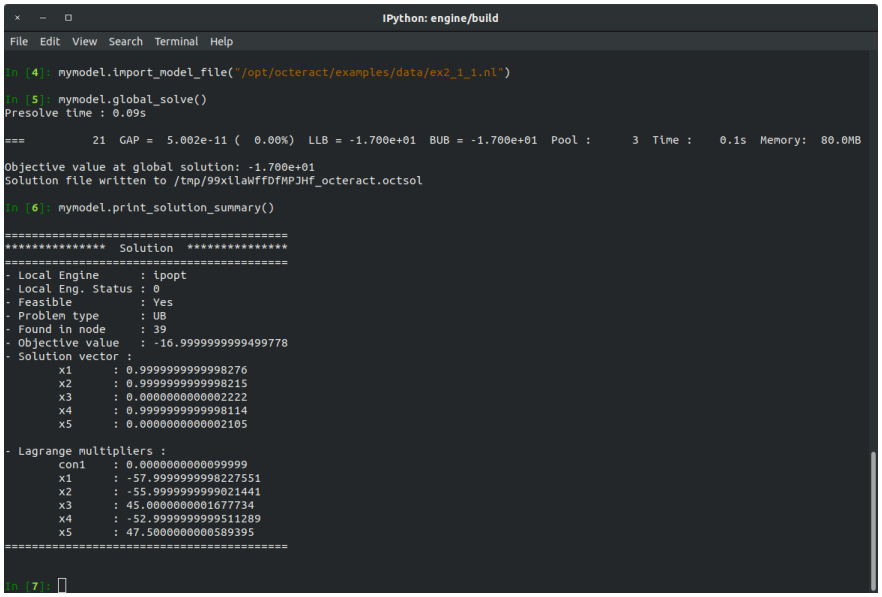
When invoking `global_solve()`, the user can optionally specify how many CPU cores to use in MPI mode. For example:

```
mymodel.global_solve(16)
```

will use 16 cores to solve the problem.

```
mymodel.print_solution_summary()
```

Running these two commands should produce the following ouput:



By default, the solver outputs the following information:

- Number of iterations (in this case 21).
- 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.

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

5 Solver flags

5.1 Input flags

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

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

5.2 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`
- `Catastrophic_Error`
- `Parsing_Error`

6 Solver Options

The engine 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_MULTISTART = true` will set that option to true. In the sections that follow, options are listed in the following format:

```
OPTION_NAME(type) = <Default value>
```

According to this format, `USE_MULTISTART` would be: `USE_MULTISTART(bool) = false`

6.1 Third-party solvers

Octeract Engine currently has interfaces for 5 third-party solvers. They can be set by setting the corresponding option to the solver code (e.g., LP_SOLVER = OSICLP will use OSICLP).

Table 1 Available options for third-party solvers.

Option name/Solver	IPOPT	OSICLP	NLOPT	OSICBC	BONMIN
NLP_SOLVER	yes	-	yes	-	yes
LP_SOLVER	yes	yes	yes	yes	yes
MILP_SOLVER	-	-	-	yes	yes
MINLP_SOLVER	yes	-	-	-	yes

NLP_SOLVER(string) = IPOPT

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

LP_SOLVER(string) = OSICLP

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

MILP_SOLVER(string) = OSICBC

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

MINLP_SOLVER(string) = IPOPT

Default: IPOPT.
Sets the solver which will be used to solve mixed-integer non-linear local optimization problems. The default is an Octeract local MINLP solver which uses IPOPT to solve subproblems.

6.2 Convergence settings

MAX_SOLVER_ITERATIONS(int) = 2000000000

Default: 2000000000.
This option is ignored in MPI mode.
Sets the maximum number of solver iterations **in serial mode**. The concept of iteration is not applicable in parallel mode. Therefore, this option is ignored when using MPI.

MAX_SOLVER_TIME(int) = +inf

Default: +inf seconds.
Sets a timeout for the solver. Note that because the solver runs in parallel we don't keep track of CPU time - all time-related information is in real time.

CONVERGENCE_TOLERANCE(double) = 1.e-3

Default: 0.001.
Sets a tolerance ε to determine whether the algorithm has converged to global optimality. The engine will terminate if either of the following conditions is satisfied:

- $BUB - LLB <= \varepsilon$ (absolute tolerance within ε)
- $(BUB - LLB) <= \varepsilon |LLB|$ **and** $(BUB - LLB) <= \varepsilon |BUB|$ (optimality gap % within ε)

6.3 Heuristics

6.3.1 Branching

PARALLEL_VAR_SELECTION_HEURISTIC(string) = OS1

Default: OS1. Available options: OS1, OS2, OS3, OS4
This option is ignored in serial mode.
Sets which heuristic should be used to select branching variables in MPI mode. The default is Octeract 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.

VARIABLE_SELECTION_HEURISTIC(string) = OS1

Default: OS1. Available options: OS1, OS2, OS3, OS4
This option is ignored in MPI mode.
Sets which heuristic should be used to select branching variables in serial mode. The default is Octeract 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.

MAX_NODES_PER_PARALLEL_BRANCH(int) = 2

Default: 2.
Sets how many nodes will be generated per parallel branching iterations. This option does not need to be changed unless you are solving very easy problems with many MPI cores. In this edge case, the solver will process nodes much faster than they can be created. If this happens, increasing this number can help keep the parallel workers saturated.

NU_NODES_PER_BRANCH(int) = 2

Default: 2.
Sets how many nodes to split a branching node into. 2 is bisection, 3 is trisection, etc.

6.3.2 Multistart

USE_MULTISTART(bool) = false

Default: false.
Enables or disables the multistart heuristic for finding feasible upper bounding solutions using multiple threads. Note that this option is only safe to use with `NLP_SOLVER = NLOPT`, as `IPOPT` is not thread safe.

NUM_MULTISTART_WORKERS(int) = 32

Default: 32.
Sets the number of threads for the multistart heuristic.

NUM_NODES_PER_MULTISTART_WORKER(int) = 10

Default: 10.
Sets the number of starting points to be investigated per worker (CPU thread).

6.3.3 Bounds tightening

USE_OBBT(bool) = true

Default: true.
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(int) = 1

Default: 1.
Maximum number of OBBT passes.

MAX_OBBT_DEPTH(int) = 4

Default: 4.
Sets the maximum depth in the branch-and-bound tree to apply OBBT.

6.4 Tolerances

BOUND_VIOLATION_TOLERANCE(double) = 1.e-8

Default: 1.e-8.
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(double) = 1.e-6

Default: 1.e-6.
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(double) = 1.e-5

Default: 1.e-5.
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(double) = 1.e-6

Default: 1.e-6.
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(double) = 1.e-3

Default: 1.e-3.
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_GAP_TOLERANCE(double) = 1.e12

Default: 1.e12.
Set the minimum optimality gap before the engine starts invoking a linear solver to solve the lower bounding problems. This is necessary because linear solvers are sometime not reliable for very large numbers. If the gap is greater than this tolerance, the engine will not solve the lower bounding LPs and use other, less effective methods instead. If your problem is numerically well-behaved but has a large gap that is not being closed, increasing this tolerance will force the engine to solve the lower bounding problems.

BIG_COEFF_TOLERANCE(double) = 1.e12

Default: 1.e12.
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 engine to solve these ill-posed lower bounding problems.

6.5 General solver settings

OUTPUT_FREQUENCY(int) = 2000000000

Default: 2000000000.
The solver will print output every `OUTPUT_FREQUENCY` iterations.

OUTPUT_TIME_FREQUENCY(int) = 1

Default:1 second. Minimum value is 1.
The solver will print output every `OUTPUT_TIME_FREQUENCY` seconds.

PRESOLVE(bool) = true

Default: true.
Enable or disable the presolver. Useful for debugging.

NUMBER_OF_FLOAT_BITS(int) = 128

Default: 128. Valid options: 64, 128, 256, 512.
Sets the number of bits that will be used to represent floating point numbers when performing Arbitrary Precision calculations. Increasing this value can help in edge cases of badly behaved numerics. Minimum value is 64.

UB_FREQUENCY(int) = 5

Default: 10. Range: 1 - 2000000000.
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(bool) = false

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

INFINITY(double) = 1.e8

Default: 1.e8.
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.

7 Running in parallel (MPI) mode

The Message Passing Interface (MPI) is a framework for distributed computing, originally designed for use in supercomputers. Octeract Engine is distributed with **MPICH 3.3**, and can be used in MPI mode out of the box.

7.1 Running on a single machine

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

```
python3 octeract-engine.py -n [number_of_processes] \
    [optimization/model/location]
```

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

Note on Hyperthreading

If your CPU supports Hyperthreading, the system will report logical processors in addition to the physical ones. However, these two types of cores are not equivalent when it comes to multiprocessing performance. If more processes than the number of physical cores are spawned, the use of logical processors can actually lead to a performance drop as it increases the probability of CPU cache misses.

If you are using the Python API or Octeract Shell, then MPI can be invoked by:

```
mymodel.global_solve(number_of_processes)
```

which will spawn the specified `number_of_processes` to solve the optimization problem.

7.2 Running on a cluster

!Dev Beta note

HPC clusters can have subtle differences which are hard for us to test without access to the individual clusters. Octeract is committed to building products that are easy to use and install, so if you are having trouble getting the engine to run on a cluster, your **feedback** is invaluable to us.

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:

```
python3 octeract-engine.py -n [nu_processes] -c [hostfile] \
    [optimization/model/location]
```

The `hostfile` is required by MPI, as it contains the IP addresses of all the machines that the engine can connect to. A sample `hostfile` for MPICH 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

The hostfile contains information about the cluster network. It is important to note that 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.