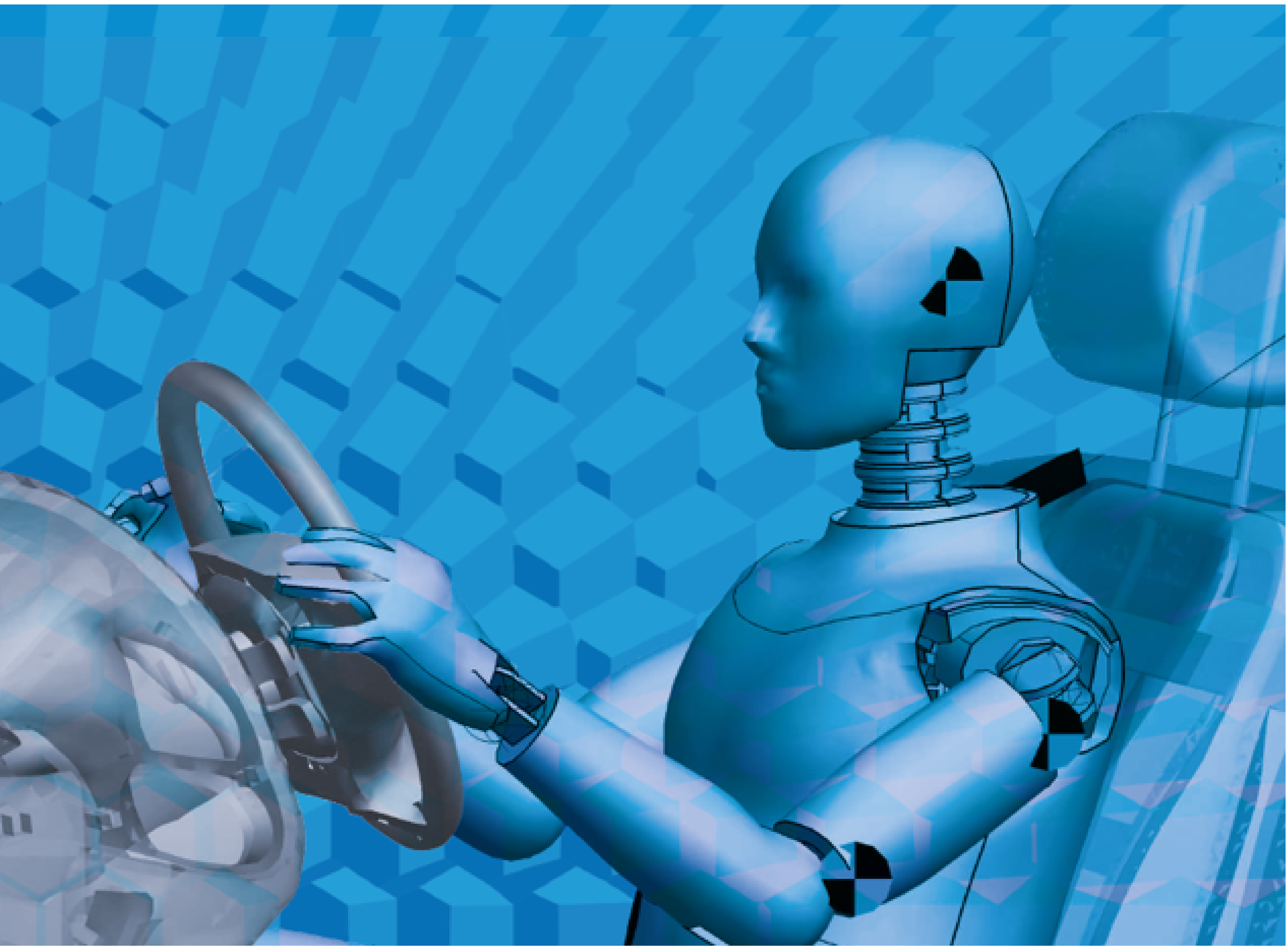
PRIMER D3PLOT T/HIS REPORTER

Python API Manual

from Oasys Ltd



Version 21.1

For help and support from Oasys Ltd please contact:

UK

Tel: +44 121 213 3399
Email: dyna.support@arup.com

China

Tel: +86 21 3118 8875
Email: china.support@arup.com

India

Tel: +91 40 69019723 / 98
Email: india.support@arup.com

USA West

Tel: +1 415 940 0959
Email: us.support@arup.com

Web: www.arup.com/dyna

or contact your local Oasys Ltd distributor.

Python API 21.1

Table of Contents

1. Title

Python API reference manual

Click on any of the programs below to access their Python API documentation:

PRIMER

D3PLOT

T/HIS

REPORTER

2. Preamble

Python scripts can control the Oasys LS-DYNA Environment software programs via the Python API.

The Python API has some advantages over the JavaScript one:

- The Python scripts run **outside** the programs (PRIMER, D3PLOT, etc), not **inside**, as JS scripts do. Therefore from **the same** Python script it is possible to speak to PRIMER, D3PLOT, T/HIS and REPORTER.
- The same script can also speak to any other software that has a Python API, for example LS-DYNA, Microsoft Excel, etc.
- In Python there are lots of available modules that can be easily installed and imported into the script.

Since Python scripts run outside the programs, not inside as JavaScripts do, Python will be slower than JavaScript. The difference is that JavaScript is running **inside** our software so has direct access to all the data. Python is running **outside** our software so has to communicate with the external process (using gRPC). Each Python call requires a "round trip" to PRIMER to do so, increasing the time required for such operations.

Furthermore, using Graphics and Widgets is not possible from an external process like Python. Graphics and Widgets in JavaScript are running in the user interface which is running inside the software in an event loop. It is not possible to do that from Python, see PRIMER, D3PLOT, T/HIS and REPORTER pages for more details of which classes cannot be used from Python.

Please consider this carefully when deciding whether to use Python or JavaScript and when writing Python routines.

3. Python modules

For each program there is a corresponding Python module that needs to be installed:

- Oasys.PRIMER module
- Oasys.D3PLOT module
- Oasys.THIS module
- Oasys.REPORTER module

Each of the Python modules above uses gRPC to communicate with the executable of the program using the Oasys.gRPC module, but this one doesn't need to be installed (it's automatically installed when installing any of the other modules).

To install the Python modules: execute the following commands in a Windows comand prompt, or in a Linux terminal:

- pip install Oasys.PRIMER
- pip install Oasys.D3PLOT
- pip install Oasys.THIS
- pip install Oasys.REPORTER

To update the Python modules when new versions are released: execute the following commands:

- pip install Oasys.PRIMER --upgrade
- pip install Oasys.D3PLOT --upgrade
- pip install Oasys.THIS --upgrade
- pip install Oasys.REPORTER --upgrade

4. gRPC connection

gRPC is the framework that enables the communication between Python and the Oasys LS-DYNA programs.

If a Python script starts an instance of an Oasys LS-DYNA program, for example PRIMER, Python will connect to the program automatically.

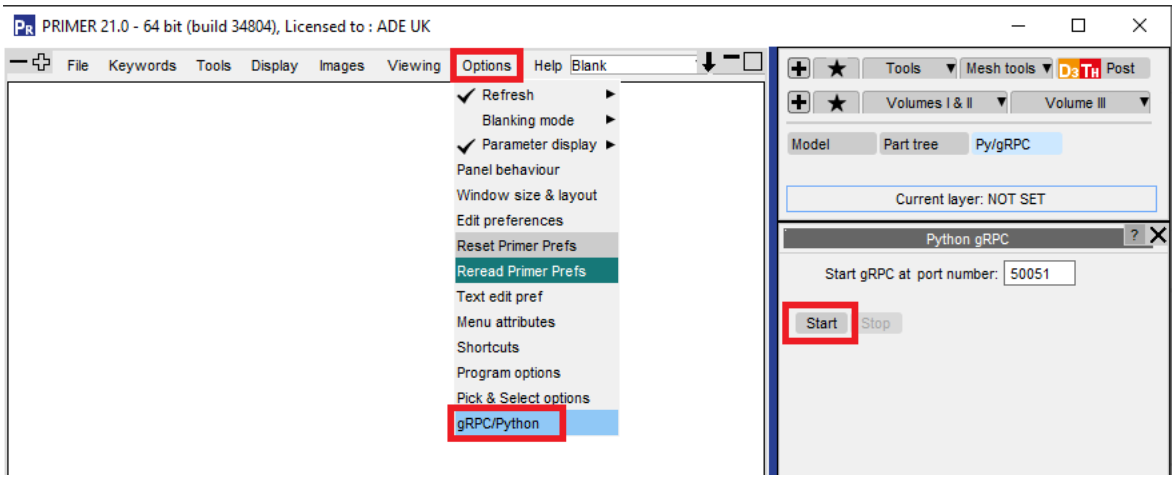
But for a Python script to connect to an already running instance of a program, the program must be in a special mode telling it to listen on a port for gRPC messages. There are 2 possible methods to do so:

- Use the -grpc command line argument when starting the program, e.g:

Bash (Unix Shell)

```
C:\\path_to_your_executable\\primer21_x64.exe' -grpc=50051
```

- Start the program in the usual way, and then under **Options > gRPC/Python** click on "**Start**":

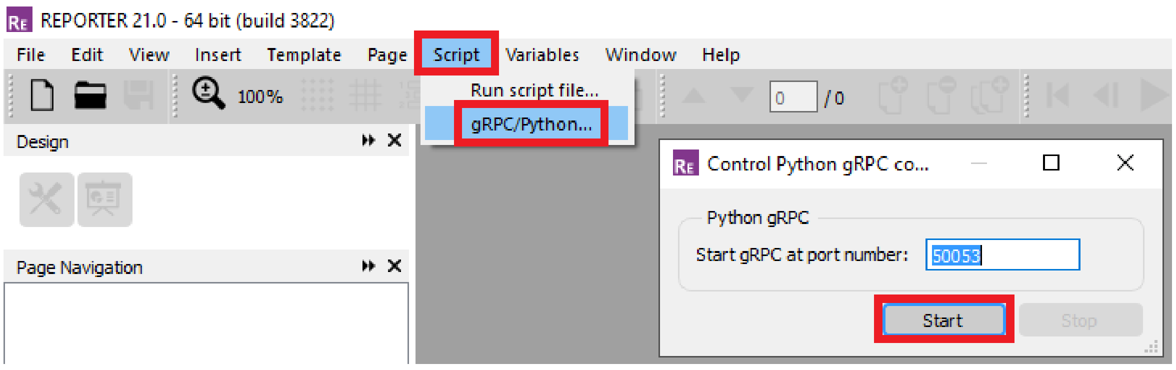

PRIMER 21.0 - 64 bit (build 34804), Licensed to : ADE UK

Start gRPC at port number: 50051

Note: 50051 is the default gRPC port for PRIMER, but a different port can be used.

In an analogous mode gRPC can be started for **D3PLOT** and **T/HIS** (the default port numbers are 50054 and 50052 respectively).

For **REPORTER** the start gRPC panel is under **Script > gRPC/Python** (the default port number is 50053):



Start gRPC at port number: 50053

5. PRIMER

Getting started

As Python is running outside PRIMER, the first thing a script needs to do is to either start an instance of PRIMER, or to connect to an already running instance of PRIMER. At the end of the script you should then either disconnect again or terminate the PRIMER instance.

A skeleton Python script to start PRIMER (Installed at C:\Oasys 21\primer21_x64.exe) and then terminate it is:

Python

```
import Oasys.PRIMER

connection = Oasys.PRIMER.start(abspath="C:\\Oasys
21\\primer21_x64.exe")

...

Oasys.PRIMER.terminate(connection)
```

By default PRIMER will use port 50051 to communicate with Python and will allocate 25Mb of memory for running scripts. These can be changed by adding port and memory arguments to the start function. e.g:

Python

```
connection = Oasys.PRIMER.start(abspath="C:\\Oasys
21\\primer21_x64.exe", port=1234, memory=100)
```

PRIMER can also be started in batch mode so that the main graphics window is not shown by using a batch argument:

Python

```
connection = Oasys.PRIMER.start(abspath="C:\\Oasys
21\\primer21_x64.exe", batch=True)
```

To connect to an instance of PRIMER that is already running, **PRIMER must currently have been started in a special mode telling it to listen on a port for gRPC messages**. See further information **here**.

A skeleton script to connect to PRIMER and disconnect again would then be:

```Python
import Oasys.PRIMER

connection = Oasys.PRIMER.connect(port=50051)

...

Oasys.PRIMER.disconnect(connection)
```

or if you want to terminate the instance of PRIMER use `terminate` instead of `disconnect`.

Python API

The JS API has been available for several years, is stable and works well, so the Python API has been designed to have the same classes, methods and properties as the JS API.

However, the following classes are not available:

- PopupWindow (GUIs not available from Python)
- Widget (GUIs not available from Python)
- WidgetItem (GUIs not available from Python)
- Check (Custom checks not available from Python)
- File (use Python i/o instead)

- Graphics (Graphics drawing not available from Python)
- Ssh (use Python modules instead)
- XlsxWorkbook (use Python modules instead)
- XlsxWorksheet (use Python modules instead)
- XMLParser (use Python modules instead)
- Zip (use Python modules instead)

and the following methods are not available:

- ForEach
- Error
- Warning

If an argument in the JS API is an object then the equivalent in Python will be a dict, and if an array in JS, the equivalent in Python will be a list.

Simple example

The following example shows how to make a model containing some nodes and shells using the JS API and the Python API :

JavaScript

```javascript
var m = new
Model();

Message("Making
nodes");

for (y=0; y<11;
y++)
{
    for (x=0; x<11;
x++)
        var n = new
Node(m, 1+x+(y*11),
x*10, y*10, 0);
}
```

Python

```python
import Oasys.PRIMER

connection =
Oasys.PRIMER.start(abspath="C:\\oasys
21\\primer21_x64.exe")

m = Oasys.PRIMER.Model()

Oasys.PRIMER.Message("Making nodes")

for y in range(0, 11):
    for x in range(0, 11):
        Oasys.PRIMER.Node(m, 1+x+(y*11),
x*10, y*10, 0)
```

`Message("Making shells");`
 	
`for (i=1; i<=10; i++)`
`{`
`    for (j=1; j<=10; j++)`
`        var s = new Shell(m, i+(j*10), i, ((i-1)*11)+j+0, ((i-1)*11)+j+1, ((i-0)*11)+j+1, ((i-0)*11)+j+0);`
`}`
 	
`m.UpdateGraphics();`
`View.Show(View.XY);`
`View.Ac();`
	`Oasys.PRIMER.Message("Making shells")`
	
	`for i in range(1, 11):`
	`    for j in range(1, 11):`
	`        Oasys.PRIMER.Shell(m, i+(j*10), i, ((i-1)*11)+j+0, ((i-1)*11)+j+1, ((i-0)*11)+j+1, ((i-0)*11)+j+0)`
	
	`m.UpdateGraphics()`
	`Oasys.PRIMER.View.Show(Oasys.PRIMER.View.XY)`
	`Oasys.PRIMER.View.Ac()`
	
	`Oasys.PRIMER.disconnect(connection)`

All the classes and methods or functions used in the Python script above can be found in the PRIMER Python API documentation, in the following pages:

- Connection methods (to start an instance of PRIMER and to disconnect Python from it at the end)
- Model class (to create a model, and to update graphics)
- Functions (to print a message)
- Node class (to create nodes)
- Shell class (to create shells)
- View class (to change the view and autoscale)

6. D3PLOT

Getting started

As Python is running outside D3PLOT, the first thing a script needs to do is to either start an instance of D3PLOT, or to connect to an already running instance of D3PLOT. At the end of the script you should then either disconnect again or terminate the D3PLOTinstance.

A skeleton Python script to start D3PLOT (Installed at C:\Oasys 21\d3plot21_x64.exe) and then terminate it is:

Python

```
import Oasys.D3PLOT

connection = Oasys.D3PLOT.start(abspath="C:\\Oasys
21\\d3plot21_x64.exe")

...

Oasys.D3PLOT.terminate(connection)
```

By default D3PLOT will use port 50054 to communicate with Python and will allocate 25Mb of memory for running scripts. These can be changed by adding port and memory arguments to the start function. e.g:

Python

```
connection = Oasys.D3PLOT.start(abspath="C:\\Oasys
21\\d3plot21_x64.exe", port=1234, memory=100)
```

D3PLOT can also be started in batch mode so that the main graphics window is not shown by using a batch argument:

Python

```
connection = Oasys.D3PLOT.start(abspath="C:\\Oasys
21\\d3plot21_x64.exe", batch=True)
```

To connect to an instance of D3PLOT that is already running, **D3PLOT must currently have been started in a special mode telling it to listen on a port for gRPC messages**. See further information **here**.

A skeleton script to connect to D3PLOT and disconnect again would then be:

Python

```python
import Oasys.D3PLOT

connection = Oasys.D3PLOT.connect(port=50054)

...

Oasys.D3PLOT.disconnect(connection)
```

or if you want to terminate the instance of D3PLOT use `terminate` instead of `disconnect`.

Python API

The JS API has been available for several years, is stable and works well, so the Python API has been designed to have the same classes, methods and properties as the JS API.

However, the following classes are not available:

- PopupWindow (GUIs not available from Python)
- Widget (GUIs not available from Python)
- WidgetItem (GUIs not available from Python)
- File (use Python i/o instead)
- Ssh (use Python modules instead)

- XlsxWorkbook (use Python modules instead)
- XlsxWorksheet (use Python modules instead)
- XMLParser (use Python modules instead)
- Zip (use Python modules instead)

If an argument in the JS API is an object then the equivalent in Python will be a dict, and if an array in JS, the equivalent in Python will be a list.

Simple example

The following example shows how to return the first model loaded in D3PLOT, set its state to 10, obtain the node with ID=1, and obtain the X displacement of that node using the JS API and the Python API :

JavaScript

```javascript
var m = Model.First();

m.state = 10;

var n = Node.GetFromID(m,
1);

var x_disp =
n.GetData(Component.DX);

Message("X displacement of
node 1: " + x_disp);
```

Python

```python
import Oasys.D3PLOT

#D3PLOT must be in listening mode
(Options>gRPC/Python>Start)
connection =
Oasys.D3PLOT.connect(port=50054)

m = Oasys.D3PLOT.Model.First()

m.state = 10

n = Oasys.D3PLOT.Node.GetFromID(m,
1)

x_disp =
n.GetData(Oasys.D3PLOT.Component.DX)

print("X displacement of node 1: " +
str(x_disp))

Oasys.D3PLOT.disconnect(connection)
```

All the classes and methods or functions used in the Python script above can be found in the D3PLOT Python API documentation, in the following pages:

- Connection methods (to connect to an instance of D3PLOT and to disconnect Python from it at the end)
- Model class (to get a model, and to set its state)
- Node class (to obtain a node from an ID and to GetData)

7. T/HIS

Getting started

As Python is running outside T/HIS, the first thing a script needs to do is to either start an instance of T/HIS, or to connect to an already running instance of T/HIS. At the end of the script you should then either disconnect again or terminate the T/HIS instance.

A skeleton Python script to start T/HIS (Installed at C:\Oasys 21\this21_x64.exe) and then terminate it is:

Python

```python
import Oasys.THIS

connection = Oasys.THIS.start(abspath="C:\\Oasys
21\\this21_x64.exe")

...

Oasys.THIS.terminate(connection)
```

By default T/HIS will use port 50052 to communicate with Python and will allocate 25Mb of memory for running scripts. These can be changed by adding port and memory arguments to the start function. e.g:

Python

```python
connection = Oasys.THIS.start(abspath="C:\\Oasys
21\\this21_x64.exe", port=1234, memory=100)
```

T/HIS can also be started in batch mode so that the main graphics window is not shown by using a batch argument:

Python

```
connection = Oasys.THIS.start(abspath="C:\\Oasys
21\\this21_x64.exe", batch=True)
```

To connect to an instance of T/HIS that is already running, **T/HIS must currently have been started in a special mode telling it to listen on a port for gRPC messages**. See further information **here**.

A skeleton script to connect to T/HIS and disconnect again would then be:

Python

```
import Oasys.THIS

connection = Oasys.THIS.connect(port=50052)

...

Oasys.THIS.disconnect(connection)
```

or if you want to terminate the instance of T/HIS use `terminate` instead of `disconnect`.

Python API

The JS API has been available for several years, is stable and works well, so the Python API has been designed to have the same classes, methods and properties as the JS API.

However, the following classes are not available:

- PopupWindow (GUIs not available from Python)
- Widget (GUIs not available from Python)
- WidgetItem (GUIs not available from Python)
- File (use Python i/o instead)
- Ssh (use Python modules instead)

- XlsxWorkbook (use Python modules instead)
- XlsxWorksheet (use Python modules instead)
- XMLParser (use Python modules instead)
- Zip (use Python modules instead)

If an argument in the JS API is an object then the equivalent in Python will be a dict, and if an array in JS, the equivalent in Python will be a list.

Simple example

The following example shows how to plot the X displacement for all the time history nodes in a model using the JS API and the Python API :

JavaScript

```
var m =
Model.GetFromID(1);

var f = AllocateFlag();

m.FlagAll(f, Entity.NODE);

if
(m.QueryDataPresent(Compone
nt.DX, Entity.NODE))
{
   var curve_array=
m.GetDataFlagged(f,
Component.DX);
}

Plot();
```

Python

```
import Oasys.THIS

#T/HIS must be in listening mode
(Options>gRPC/Python>Start)
connection =
Oasys.THIS.connect(port=50052)
m =  Oasys.THIS.Model.GetFromID(1)

f = Oasys.THIS.AllocateFlag()

m.FlagAll(f, Oasys.THIS.Entity.NODE)

if
m.QueryDataPresent(Oasys.THIS.Compon
ent.DX, Oasys.THIS.Entity.NODE):
   curve_list = m.GetDataFlagged(f,
Oasys.THIS.Component.DX);

Oasys.THIS.Plot()

Oasys.THIS.disconnect(connection)
```

All the classes and methods or functions used in the Python script above can be found in the T/HIS Python API documentation, in the following pages:

- Connection methods (to start an instance of T/HIS, and to disconnect Python from it at the end)
- Model class (to get the model that has an ID=1, to QueryDataPresent, and to GetDataFlagged)
- Functions (to allocate a flag and to Plot/update graphs)

8. REPORTER

Getting started

As Python is running outside REPORTER, the first thing a script needs to do is to either start an instance of REPORTER, or to connect to an already running instance of REPORTER. At the end of the script you should then either disconnect again or terminate the REPORTERinstance.

A skeleton Python script to start REPORTER (Installed at C:\Oasys 21\reporter21_x64.exe) and then terminate it is:

Python

```
import Oasys.REPORTER

connection = Oasys.REPORTER.start(abspath="C:\\Oasys
21\\reporter21_x64.exe")

...

Oasys.REPORTER.terminate(connection)
```

By default REPORTER will use port 50053 to communicate with Python and will allocate 25Mb of memory for running scripts. These can be changed by adding port and memory arguments to the start function. e.g:

Python

```
connection = Oasys.REPORTER.start(abspath="C:\\Oasys
21\\reporter21_x64.exe", port=1234, memory=100)
```

REPORTER can also be started in batch mode so that the main graphics window is not shown by using a batch argument:

Python

```
connection = Oasys.REPORTER.start(abspath="C:\\Oasys
21\\reporter21_x64.exe", batch=True)
```

To connect to an instance of REPORTER that is already running, **REPORTER must currently have been started in a special mode telling it to listen on a port for gRPC messages**. See further information **here**.

A skeleton script to connect to REPORTERand disconnect again would then be:

```python
import Oasys.REPORTER

connection = Oasys.REPORTER.connect(port=50053)

...

Oasys.REPORTER.disconnect(connection)
```

or if you want to terminate the instance of REPORTER use `terminate` instead of `disconnect`.

Python API

The JS API has been available for several years, is stable and works well, so the Python API has been designed to have the same classes, methods and properties as the JS API.

However, the following classes are not available:

- Ssh (use Python modules instead)
- XlsxWorkbook (use Python modules instead)
- XlsxWorksheet (use Python modules instead)
- XMLParser (use Python modules instead)
- Zip (use Python modules instead)

If an argument in the JS API is an object then the equivalent in Python will be a dict, and if an array in JS, the equivalent in Python will be a list.

Simple example

The following example shows how to create a template and to add a page with some text using the JS API and the Python API :

JavaScript

```javascript
// Save this in a file named "exampleJavaScript.js" and run it via command line:
// C:\Oasys 21>reporter21_x64.exe -script=exampleJavaScript.js -batch

var aTemplate = new Template();

Debug("Created template with name " + aTemplate.filename);

var aPage = aTemplate.GetPage(0);

LogPrint("Creating a text item!");

var aItem = new Item(aPage, Item.TEXT, "Title", 100, 100, 270, 290);
aItem.text = "First page";
aItem.fontSize = 48;
aItem.textColour = Colour.Red();
aItem.justify = Reporter.JUSTIFY_CENTRE;

Debug("Finishing script: leaving the template open!");
```

Python

```python
import Oasys.REPORTER

connection = Oasys.REPORTER.start(abspath="C:\\Oasys 21\\reporter21_x64.exe")

aTemplate = Oasys.REPORTER.Template()

Oasys.REPORTER.Debug("Created template with name " + aTemplate.filename)

aPage = aTemplate.GetPage(0)

Oasys.REPORTER.LogPrint("Creating a text item!")

aItem = Oasys.REPORTER.Item(aPage, Oasys.REPORTER.Item.TEXT, "Title", 100, 100, 270, 290)
aItem.text = "First page"
aItem.fontSize = 48
aItem.textColour = Oasys.REPORTER.Colour.Red()
aItem.justify = Oasys.REPORTER.Reporter.JUSTIFY_CENTRE

Oasys.REPORTER.Debug("Closing the connection and leaving the template open!")

Oasys.REPORTER.disconnect(connection)
```

All the classes and methods or functions used in the Python script above can be found in the REPORTER Python API documentation, in the following pages:

- Connection methods (to start and instance of REPORTER and to disconnect Python from it at the end)
- Template class (to create a template)
- REPORTER functions (to print messages)
- Item class (to add items to a page of the template)

9. Examples

9.1. Create geometry with Gmsh

9.1.1. Overview

Introduction

This example is intended to illustrate the benefits of using the PRIMER Python API in coordination with open source Python libraries. In this particular example the Gmsh Python library (https://gmsh.info/): Gmsh is a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities.

Using Gmsh we are able to use its built in functions to define geometry and mesh the geometry. Once the geometry is meshed we can reformat the geometry data and use it to recreate the geometry in PRIMER.

Example

This example is a reproduction of the "t1.py" example produced by Gmsh can be found here: (https://gitlab.onelab.info/gmsh/gmsh/-/blob/gmsh_4_12_2/tutorials/python/t1.py)

The example is slightly modified to give the user the ability to specify the width and length of the rectangle with input variables, and the option to declare what type of meshing algorithm they would like to use.

Running the code

Inputs

- l - length of the rectangle (float)
- w - width of the rectangle (float)
- lc - size of the mesh (float)

- alg - type of meshing algorithm (int) 1: MeshAdapt, 2: Automatic, 3: Initial mesh only, 5: Delaunay, 6: Frontal-Delaunay, 8: Frontal-Delaunay for Quads, 9: Packing of Parallelograms, 11: Quasi-structured Quad
- saveFolder - name of folder to save model (str)
- keyName - name of PRIMER key file (str)
- showGUI - show gmsh GUI (window must be exited to continue running of program) (boolean)
- dynaGen - create PRIMER model (boolean)

Executing

To execute the program one must first create the **.env** file following the **.env-template**. For this example only the **PRIMERCON** variable needs to be filled out, this variable should be the path to the PRIMER executable, this will vary from user to user but in Windows is typically in "C:\Users\xxxx.xxxx\AppData\Roaming\Ove Arup\v21.0_x64\primer21_x64.exe"

Once the .env file has been configured the program is ready to be run.
To run execute '**gmesh_geom_gen.py**' through terminal or through IDLE like VSCode.

9.1.2. Instructions

- Download the following files and save them in your working directory:
    - gmesh_geom_gen.py
    - dynaCon.py
    - .env (*Make sure the downloaded file has a dot at the start. If necessary, edit the file name to add it*)
- Edit the **.env** file with the path to your PRIMER executable (only the PRIMERCON variable is used in this example).
- If you don't have it already, install the following Python modules:
    - Oasys.PRIMER: **pip install Oasys.PRIMER**
    - gmsh: **pip install gmsh**
    - python-dotenv: **pip install python-dotenv**
- Run the script **gmesh_geom_gen.py**

9.2. Open 3D - Import geometry to PRIMER

9.2.1. Overview

Introduction

This example serves to illustrate how geometry can be imported from external sources and imported into PRIMER in a usable format. Here the open source library Open3D is used to import various meshes, these meshes are then put into PRIMER as shell elements using the PRIMER Python API.

Running the code

Inputs

- model - Name of the model and the geometry to be imported, options are: **armadillo**, **monkey**, **avocado**, **sword**, and **bunny**. ALL options are from Open3D.
- folder - folder where model will be saved in outputs folder (will be created if does not exist)
- saveFolder - name of folder to save model (str)

Executing

To execute the program one must first create the **.env** file following the **.env-template**. For this example only the PRIMERCON variable needs to be filled out, this variable should be the path to the PRIMER executable, this will vary from user to user but in Windows is typically in "C:\Users\xxxx.xxxx\AppData\Roaming\Ove Arup\v21.0_x64\primer21_x64.exe"

Once the .env file has been configured the program is ready to be run. To run execute '**geom2Primer.py**' through terminal or through IDLE like VSCode.

9.2.2. Instructions

- Download the following files and save them in your working directory:
  - geom2Primer.py
  - dynaCon.py
  - .env (*Make sure the downloaded file has a dot at the start. If necessary, edit the file name to add it)*
- Edit the **.env** file with the path to your PRIMER executable (only the PRIMERCON variable is used in this example).
- If you don't have it already, install the following Python modules:
  - Oasys.PRIMER: **pip install Oasys.PRIMER**
  - open3d: **pip install open3d**
  - python-dotenv: **pip install python-dotenv**
  - numpy: **pip install numpy**
- Run the script **geom2Primer.py**

9.3. 3D Mesh from 2D STEP file

9.3.1. Overview

Introduction

This example is intended to illustrate the benefits of using the PRIMER Python API in coordination with open source Python libraries. In this particular example the gmsh Python library (https://gmsh.info/): Gmsh is a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities.

Using Gmsh we are able to import a 2D surface as an IGES or STEP file. This surface can then be extruded and meshed to produce a model. With the current configuration the script will create both triangula and rectangular prisms. Gmsh can easily create tetrahydrons or other element shapes, but the conversion script from Gmsh to PRIMER is only configured for triangular and rectangular prisms.

Example

This example is similar to the following Gmsh examples:

't4.py'
https://gitlab.onelab.info/gmsh/gmsh/blob/gmsh_4_12_2/tutorials/python/t3.py
't20.py'
https://gitlab.onelab.info/gmsh/gmsh/blob/gmsh_4_12_2/tutorials/python/t20.py#L24

The example is slightly modified to give the user the ability to specify the file containing a 2D surface to be extruded, direction of extrusion, magnitude of the extrusion, number of vertical elements, and the size of 2D mesh. The user may also modify the algorithms used to work, but this may require additional steps to ensure the model can be exported into PRIMER.

Running the code

Inputs

- fileName: name of the input file to be meshed and extruded. This file should be placed in the same directory as the meshSTEP.py and dynaCon.py files (string)
- fileExt: the file extension to be used, options are '.step'/'.stp', or '.igs' (string). NOTE: the file will be imported with the assumed unit of meters.
- h: magnitude, in meters, of the extrusion (int/float)
- extrdDir: direction, in meters, of the extrusion options are 'x', 'y', or 'z'. Must be orthogonal to 2D Surf (string)
- meshSize: 2D element mesh size to be extruded, in meters (int/float)
- numExtrdElems: number of elements in the extrusion direction (int)

Executing

To execute the program one must first create the **.env** file following the **.env-template**. For this example only the **PRIMERCON** variable needs to be filled out, this variable should be the path to the PRIMER executable, this will vary from user to user but is typically in "C:\Users\xxxx.xxxx\AppData\Roaming\Ove Arup\v21.0_x64\primer21_x64.exe"

After the .env file is configured and the required libraries downloaded, one can execute '**mesh2DGeom.py**' via terminal or through an IDLE like VSCode.

9.3.2. Instructions

- Download the following files and save them in your working directory:
  - 2dmesh.step
  - mesh2DGeom.py
  - dynaCon.py
  - .env (*Make sure the downloaded file has a dot at the start. If necessary, edit the file name to add it)*
- Edit the **.env** file with the path to your PRIMER executable (only the PRIMERCON variable is used in this example).
- If you don't have it already, install the following Python modules:
  - Oasys.PRIMER: **pip install Oasys.PRIMER**
  - python-dotenv: **pip install python-dotenv**
- Run the script **mesh2DGeom.py**

9.4. Beam optimization

9.4.1. Introduction

Introduction and background

LS-Dyna Von Mises Topology Optimization

The following is a description and overview of an implementation of a common topology optimization using LS-Dyna and the Python API.

The general idea of this optimization is to take a solid beam subject to boundary conditions and loading. The goal is to minimize the material needed to resist the load. This is done by analyzing the model, and removing elements that have a Von Mises stress multiplied by a calculated density factor under the user defined threshold. This is shown by Equation 1.

Equation 1:

$$\frac{\sigma_i}{\sigma_{max}} * d \leq t$$

Where:

σ_i – element Von Mises Stress

σ_{max} – max element Von Mises Stress

d – density filter factor

t – current threshold

Figure 1 illustrates the density filter. This filter is a ratio ranging from 0-1 of active (red) elements to all the elements whose centroids are within the circle defined by radius *r*.

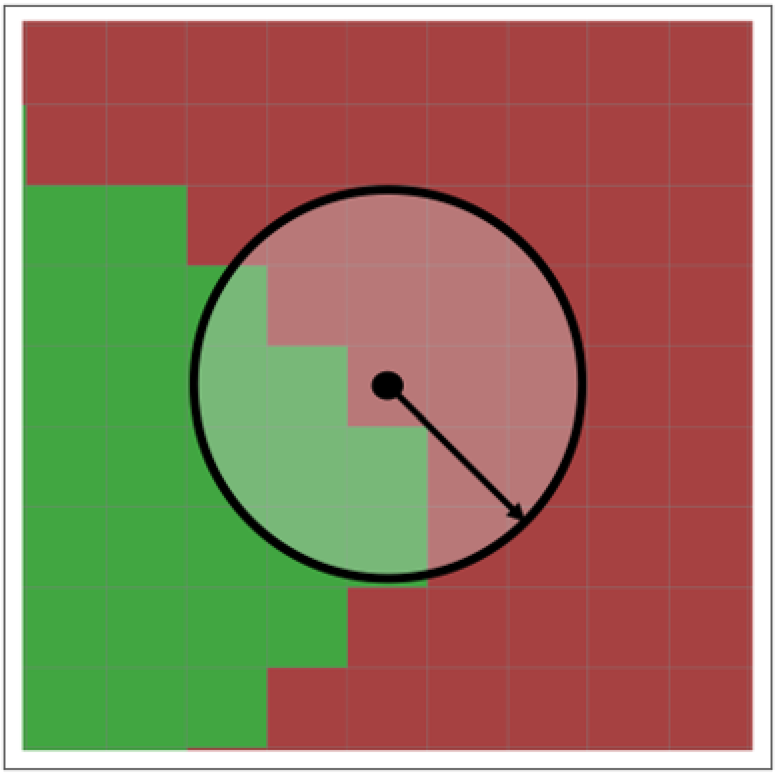
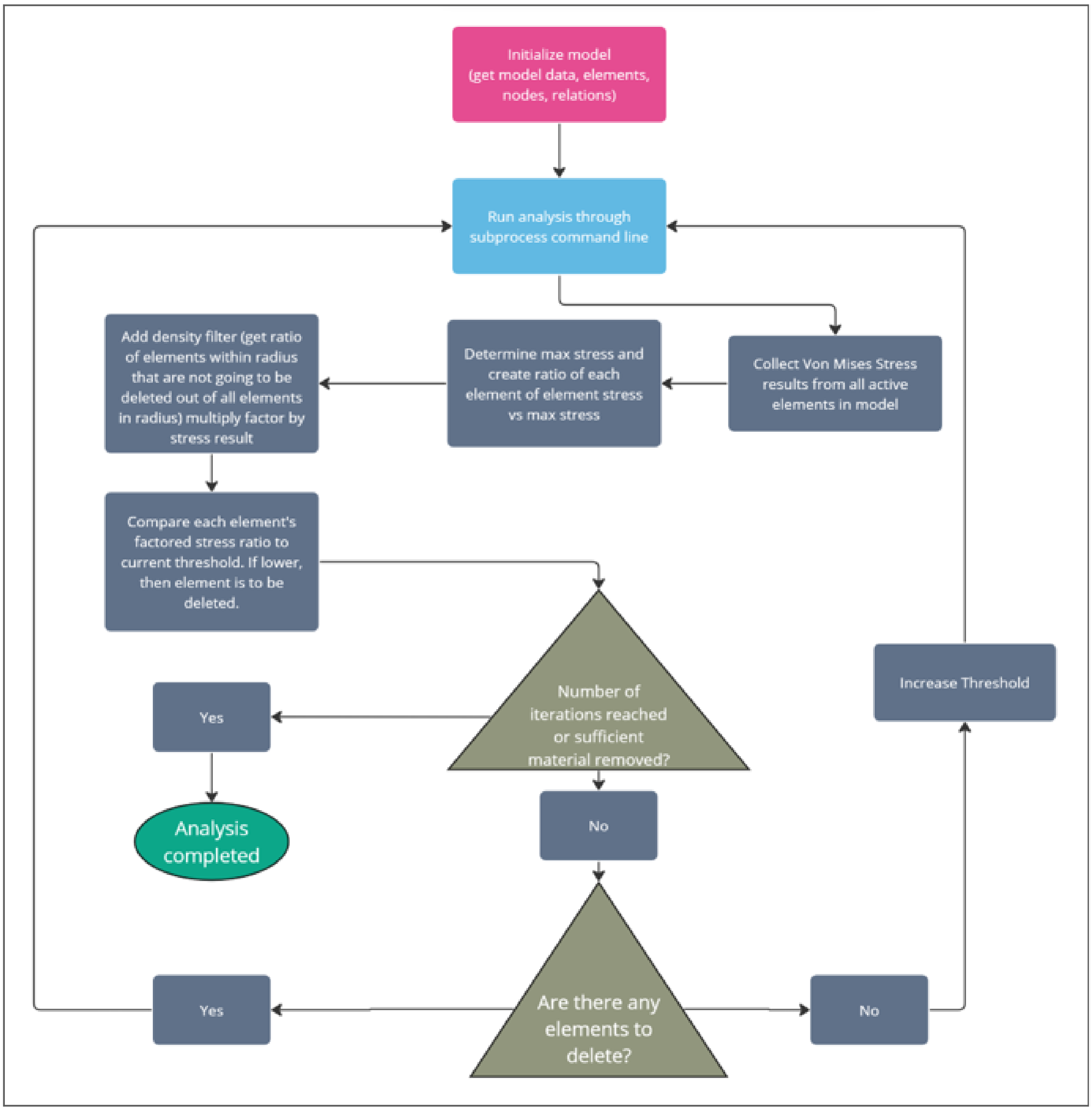


Figure 1: Density filter example

Once the elements have been removed the model is run again, when there are no elements that are beneath the threshold, the threshold is increased by a user defined value and the process is repeated. The flow diagram is shown below in Figure 2.



Initialize model (get model data, elements, nodes, relations)
Run analysis through subprocess command line
Collect Von Mises Stress results from all active elements in model
Determine max stress and create ratio of each element of element stress vs max stress
Add density filter (get ratio of elements within radius that are not going to be deleted out of all elements in radius) multiply factor by stress result
Compare each element's factored stress ratio to current threshold. If lower, then element is to be deleted.
Number of iterations reached or sufficient material removed?
Yes
Analysis completed
No
Are there any elements to delete?
Yes
No
Increase Threshold

Figure 2: Flow chart

An example of the implemented algorithm is shown below in Figure 3, where the beam is supported at two points on the left of the beam at the top and the bottom. A point load is applied at mid height on the far right of the beam. In this example the final solution at iteration 39 has 50% less material than the initial beam. The initial model can be seen labeled as ‘0’:

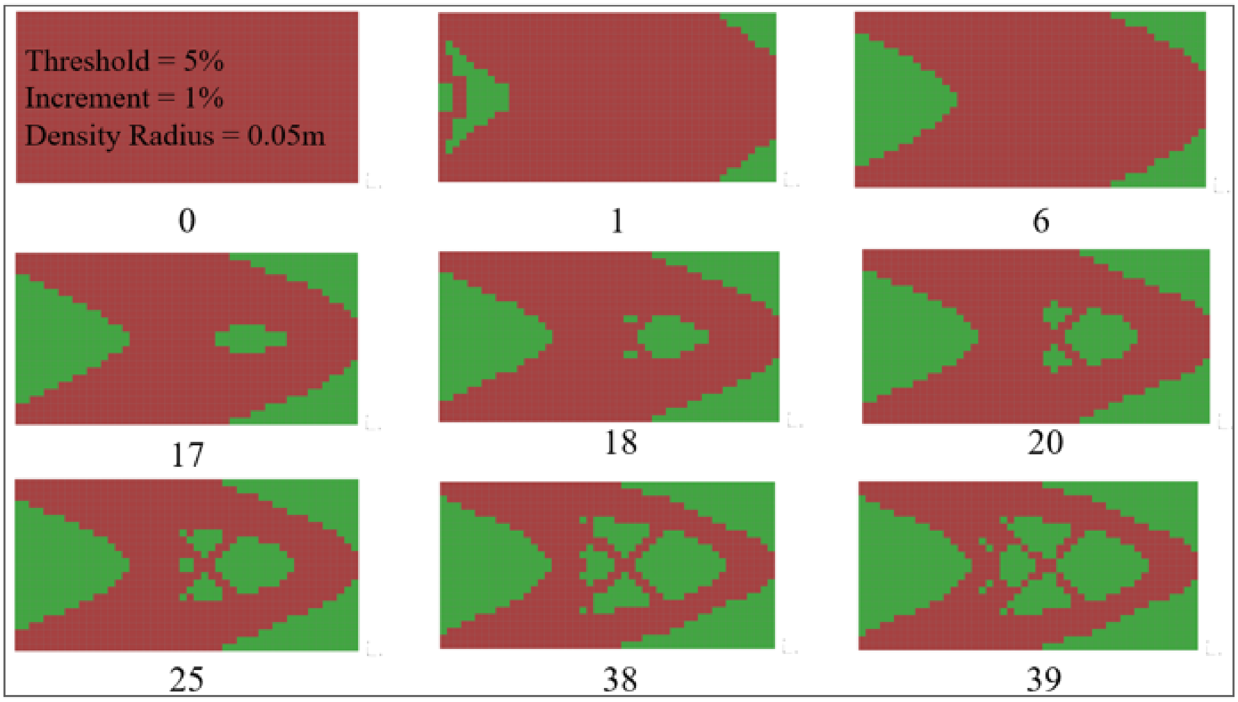


Figure 3: Cantilever optimization example

9.4.2. Overview

About the code

Folder structure

In the working directory the user needs to create a folder called "**initialM**". This folder will contain the variations of input or starting models. Each model resides in a numbered folder, as when it is run it will create additional files that we would like to restirct to that input folder. Once the "initalM" folder is created the user can place their starting model (inside a numbered folder) into it. Once the analysis starts the code should automatically create the 'Outputs' area, inside of which each iteration will be stored in numbered folders with the starting index as the intial model number.

ie Input model: >initialM>01>beam.key || Output: >outputs>02>beam.key

Inputs

The current conficuration assumes that the input model is a rectangular prism comprised of solid elements. The code has the following input parameters:

- numInt - Number of iterations (int)
- key - Name of model .key (string)
- timeHis - name of model T/His .thf (same as .key) (str)
- L - number of elements in Length direction (int)
- H - number of elements in height direction (int)
- r - density filter radius (float)
- thresh - initial threshold -> thresh < e_vms/max(vonMises Stress) (float)
- threshInc - increment that the threshold is to be increased by (float)

- cnt - Number of input model folder located in the 'initalM' folder (int)

Env Variables

The environment variables shown in the template yield the locations of the executables for the PRIMER, T/HIS, and the LS-DYNA solver.

This example was run with the following executables:

- DYNARUNEXE = "... ls-dyna_smp_d_R13.1_138-g8429c8a10f_winx64_ifort190.exe" https://www.oasys-software.com/dyna/downloads/ls-dyna-executables/ (V13, SMP, Double)
- PRIMERCON = "... primer21_x64.exe"
- THISCON = "... this21_x64.exe"

Running

To start the optimization one should first ensure that they are able to run the dyna executable through the command line. Examples of dyna execuatables and strings may be found in the manual and the executables used at the writing of this are listed in the Env Variables section above. Depending on the users setup and preferences regarding LS-DYNA the user may want to review the "dynaRun.py" file and configure the 'run' variable as needed.

Once a user can run a model through the command line, the script can be run. In a terminal window navigate to the working directory and execute **mainOPT.py**.

9.4.3. Instructions

- Download the following files and save them if your working directory:
  - beam.py
  - beamOPT.py
  - dynaCon.py
  - dynaRun.py
  - getResults.py
  - mainOPT.py
  - optModel.py
  - results.py
  - .env (*Make sure the downloaded file has a dot at the start. If necessary, edit the file name to add it)*
- In that folder create a subfolder called "initialM"
- In "initialM" create another subfolder called "450"
- In "450" put the following file: beam.key
- Download and install LS-DYNA R13.1 SMP Double precision from the Oasys website
- If you don't have it already, install the following Python modules:
  - Oasys.PRIMER: **pip install Oasys.PRIMER**
  - Oasys.THIS: **pip install Oasys.THIS**
  - python-dotenv: **pip install python-dotenv**
- Edit the file .env with the following 3 environment variables:
  - DYNARUNEXE = "*path to the LS-DYNA executable you have installed in the step above*"
  - PRIMERCON = "*path to your PRIMER executable*"
  - THISCON = "*path to your T/HIS executable*"
- Run the script **mainOPT.py**