Plaxis scripting

Ivan Depina

SINTEF Infrastructure
Rock and Geotechnical Engineering

Advanced Course on Computational Geotechnics in Trondheim, 2016

Outline

Plaxis commands

- The most common approach for interacting with the Plaxis 2D or 3D software is based on the graphical user interface (GUI).
- The Plaxis GUI provides a set of menus, toolbars, panels, and icons to construct a finite element model, modify it and examine results.

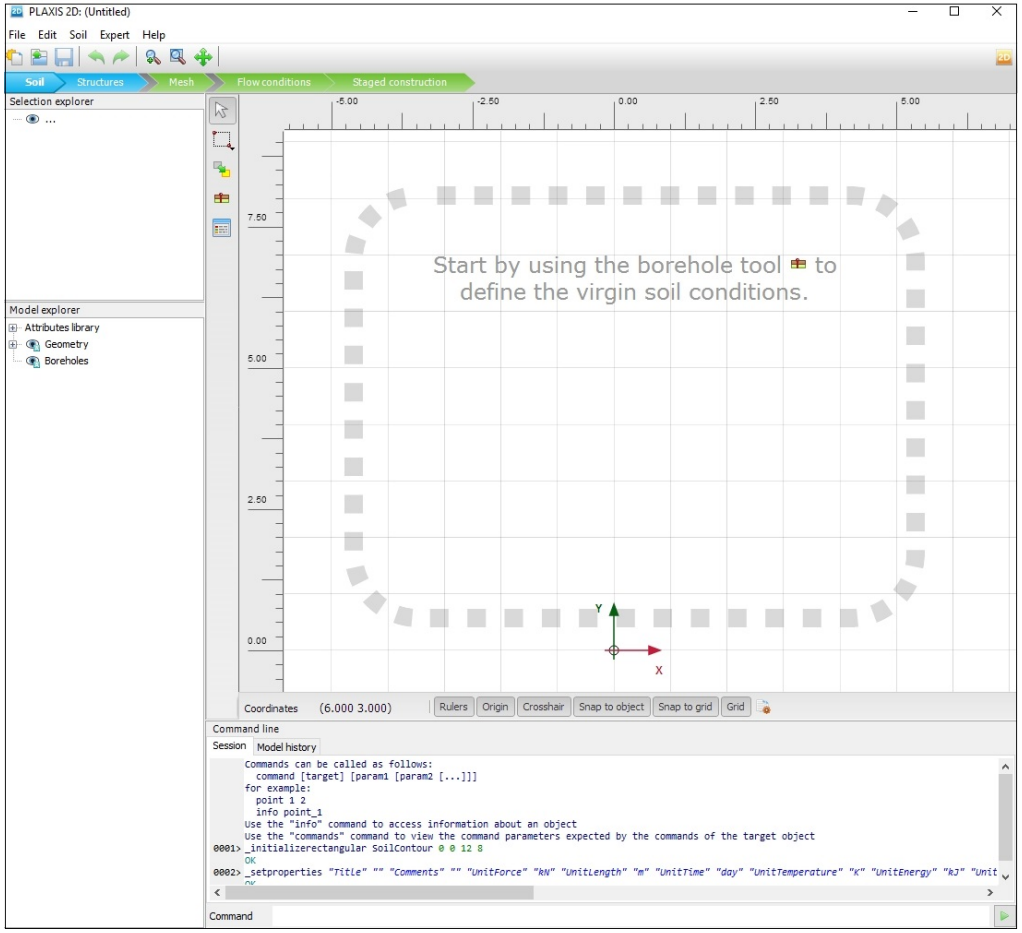


Figure: Plaxis GUI.

Plaxis commands

- Alternative approach to constructing and interacting with a finite element model can be achieved with the use of Plaxis commands.
- Commands corresponding to different GUI actions can be examined in the *Command line*.

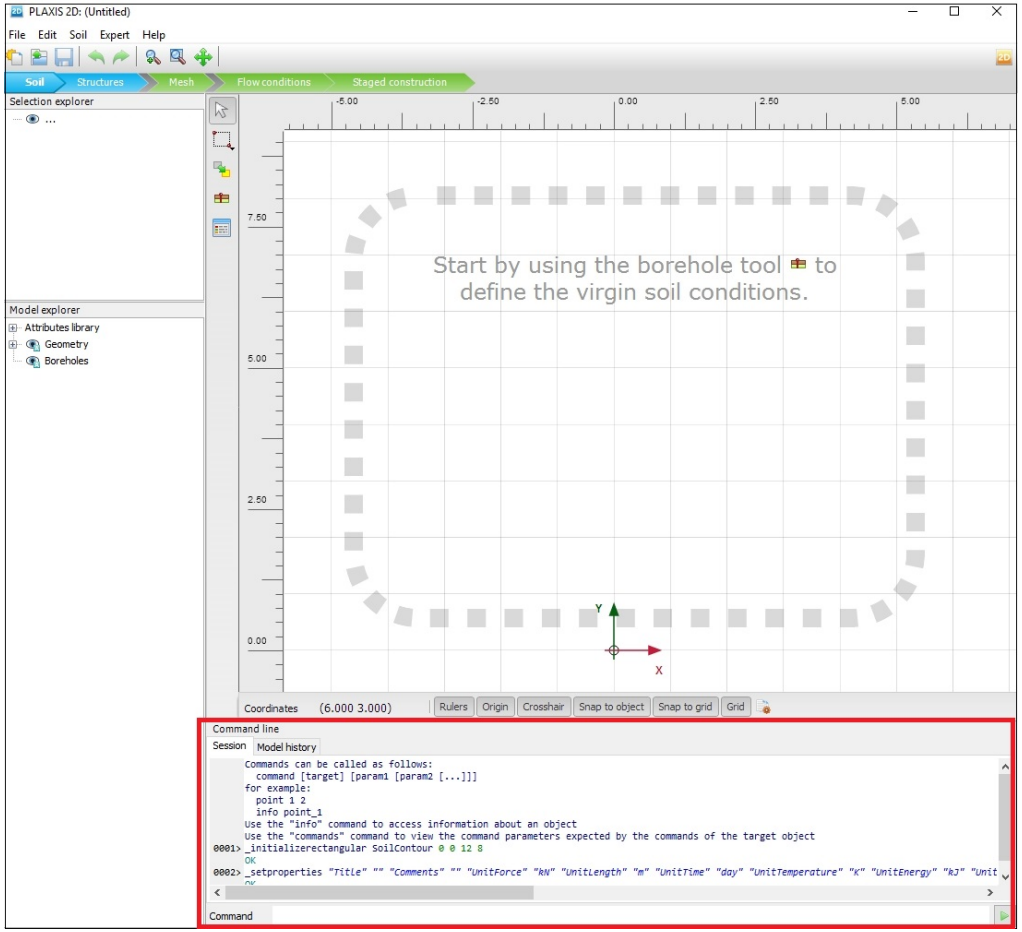


Figure: Plaxis *Command line*.

Plaxis commands

- The main advantage of using Plaxis commands is that they enable one to automate the workflow of an iterative design process.
- Additionally, quick changes can be made to a FE model using pre-defined design parameters.

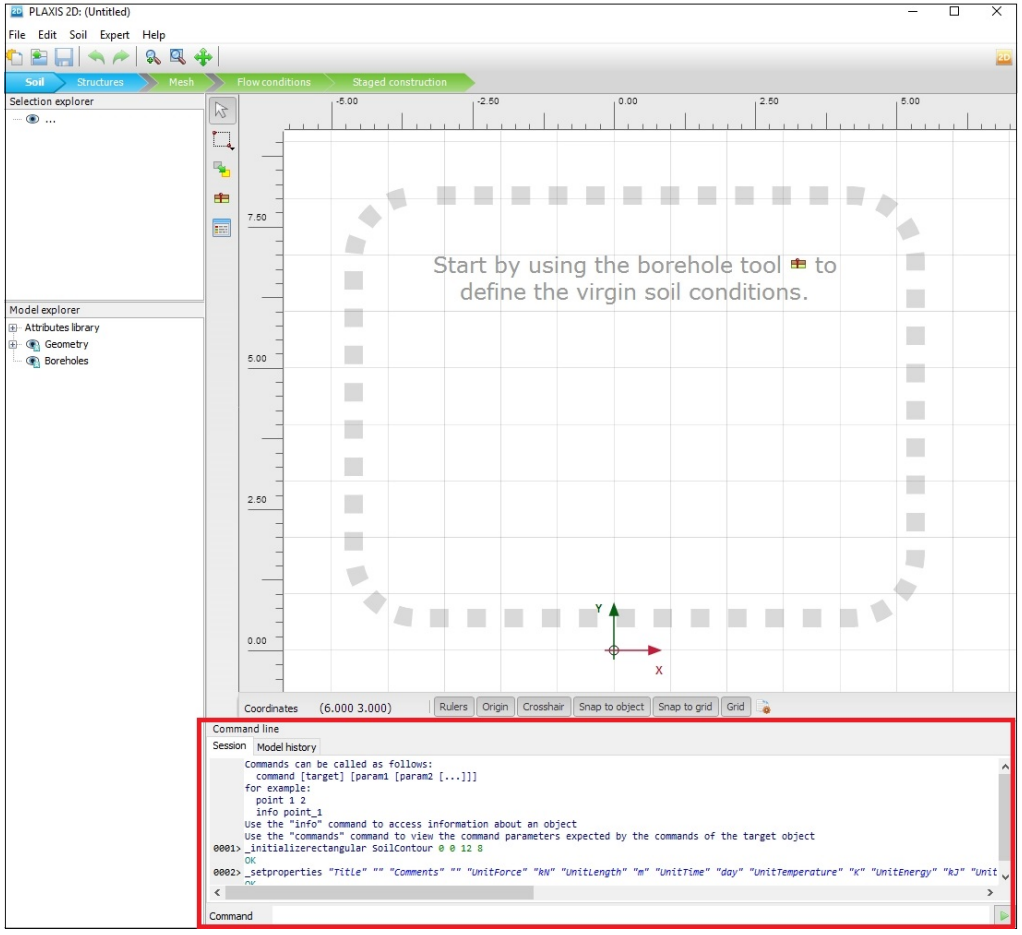


Figure: Plaxis *Command line*.

Plaxis commands

- The goal of this presentation is provide an insight into different options of interacting with an FE model using the Plaxis commands.
- Note that the majority of these options require the VIP license.

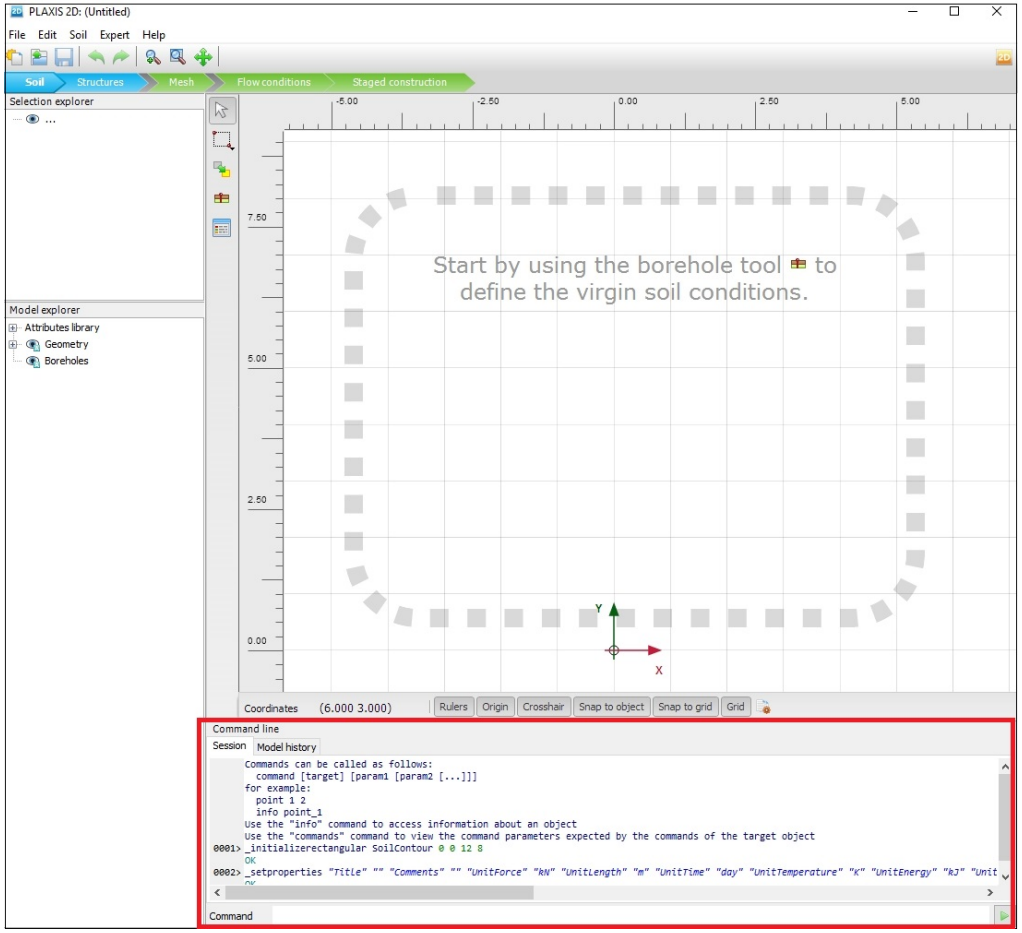


Figure: Plaxis *Command line*.

Commands and objects

- Information on the input commands and objects can be found in *Help* $\rightarrow$ *Command reference*
- The commands reference gives an overview of different commands and their parameters, and provides useful examples.
- The input reference provides an overview of the objects that can be created and accessed in Plaxis, including lists of their properties and their meaning.

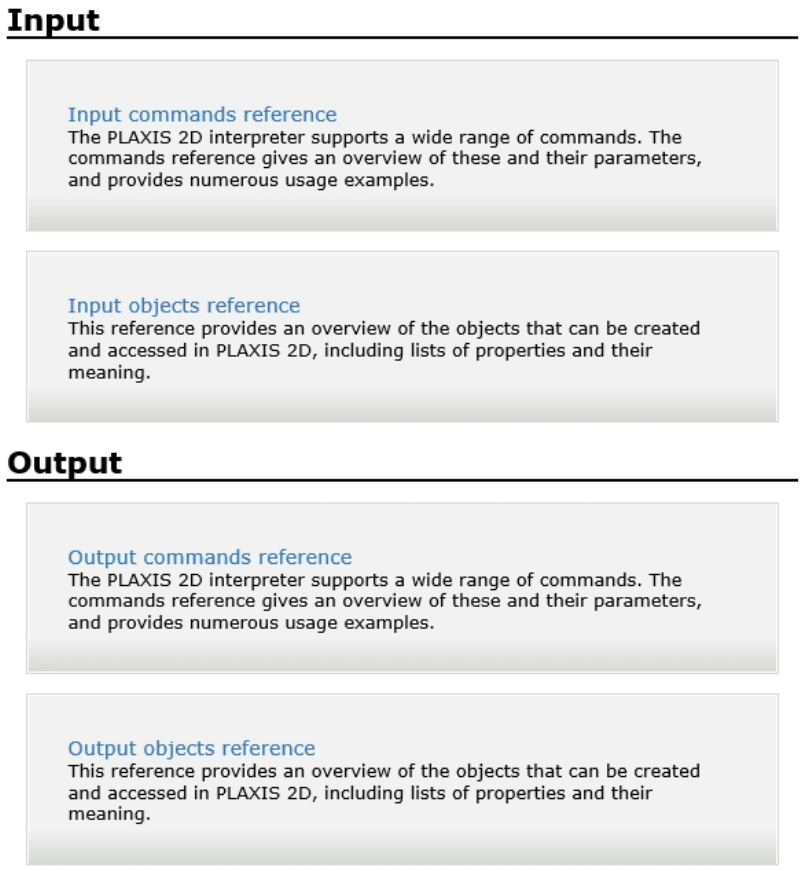


Figure: Plaxis commands and objects reference.

Outline

Input commands example: *pointload (pld)*

- Input commands are specified in the Plaxis input window.
- *pointload* adds point load features to points.
- The reference manual provides six alternatives for the use of the command *pointload*:
  1. Add point load features to one or more existing points in the geometry.
     Example: *pointload Point_2 Point_3*
     Adds point load features to *Point_2* and *Point_3*.
  2. Create a new point and add a point load feature to it.
     Example: *pointload (5 6)*
     Creates a new point at (5, 6) and adds a point load feature to it.
  3. Create several new points and add point load features to them.
     Example: *pointload (5 6) (8 9)*
     Creates two new points at (5, 6) and (8, 9) and adds point load features to them.

Input commands example: *pointload (pld)*

- The reference manual provides six alternatives for the use of the command *pointload*:
  4. Add point load features to one or more existing points in the geometry and directly set their properties.
     Example: *pointload (Point_2 Point_3) "Fx" 3 "Fy" 7*
     Adds point load features to *Point_2* and *Point_3* sets the *Fx* to 3 and the *Fy* to 7 for both point loads.
  5. Create a new point, add a point load feature to it and directly set its properties.
     Example: *pointload (5 6) "Fx" 3 "Fy" 7*
     Creates a new point at (5, 6), adds a point load feature to it and sets the *Fx* to 3 and the *Fy* to 7.
  6. Create several new points, add point load features to them and directly set their properties.
     Example: *pointload (5 6) (8 9) "Fx" 3 "Fy" 7*
     Creates two new points at (5, 6) and (8, 9), adds point load features to them and sets the *Fx* to 3 and the *Fy* to 7 for both point loads.

Input objects example: *PointLoad*

- *PointLoad* specifies the point load feature of a point.

Property	Long name	Unit	Description
Name	Name		Name of the point load
Comments	Comments		Comments on the point load
Fx	Fx	F	x-component of the load
Fy	Fy	F	y-component of the load
F	|F|	F	Absolute value of the load
BendingMoment	BendingMoment	F·L	Value of the Bending Moment

info, *commands*, and *echo* commands

- Information on all available commands and attributes for an existing input object can be obtained with the command *info*.
- *commands (cms)* displays available commands and their signatures.
- *echo* displays the details of the project or a specified object.

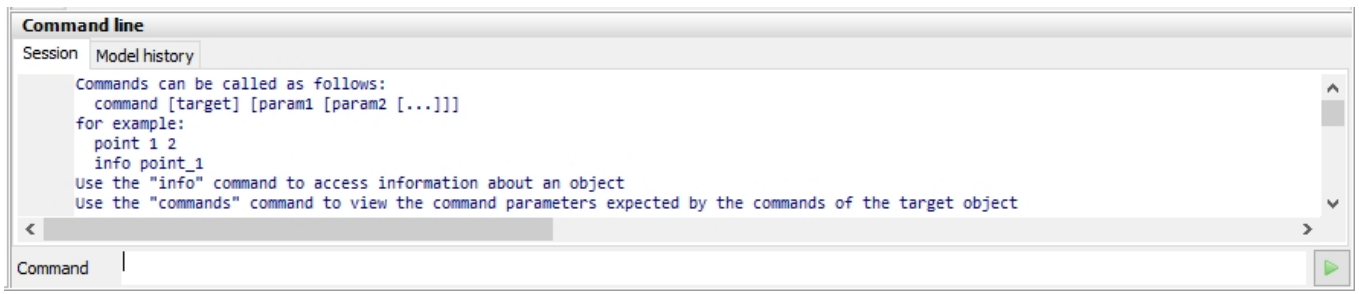


Figure: *Command line* on startup.

Example for the use of *info*, *commands*, and *echo* commands

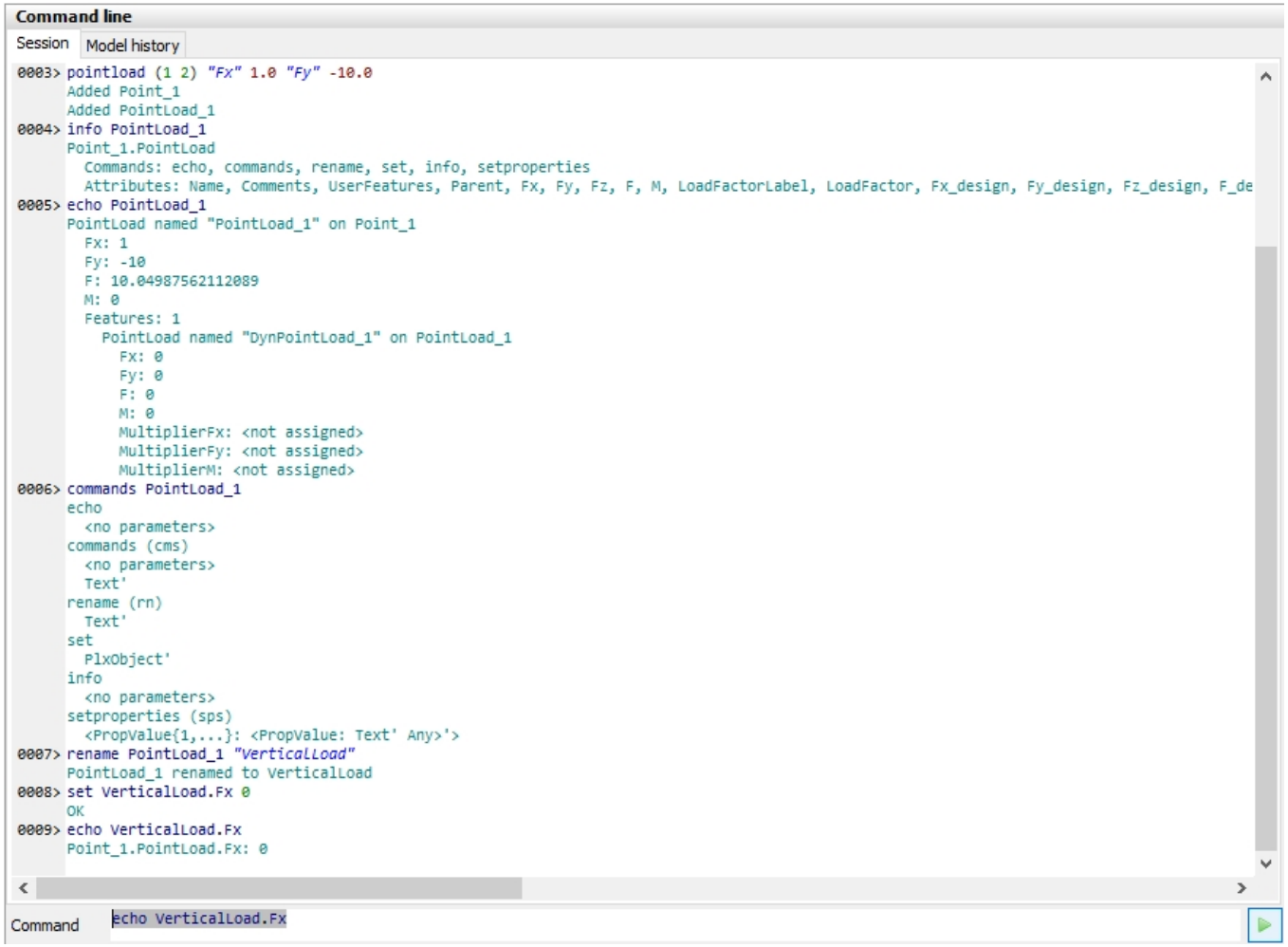


Figure: Example for the use of *info*, *commands*, and *echo* commands.

Outline

Output commands example: *getsingleresult (gsres)*

- Output commands are specified in the Plaxis output window.
- *getsingleresult (gsres)* gives the result at the specified coordinate or a specific node/stress point with or without result smoothing.
- The following signature needs to be used with the command *getsingleresult (gsres)*:

getsingleresult	Phase reference Phase reference	Result type	Point			
getsingleresult	Phase reference Phase reference	Result type	Point	Boolean		
getsingleresult	Phase reference Phase reference	Result type	x	y		
getsingleresult	Phase reference Phase reference	Result type	x	y	Boolean	
getsingleresult	Phase reference Phase reference	Result type	x	y	Boolean	Directional vector x

Figure: Signature for the use of the *getsingleresult (gsres)* command.

Output commands example: *getsingleresult (gsres)*

- The following five alternatives for the use of the command *getsingleresult (gsres)* can be considered:
  1. Gives the result at a specific node or stress point.
     Example: *getsingleresult Phase_1 Soil.Suction 18*
     Gives the result for suction a specific soil node 18 for *Phase_1*.
  2. Gives the result at a specific node or stress point with or without result smoothing.
     Example: *getsingleresult Phase_1 Soil.Suction 18 True*
     Gives the result for suction a specific soil node 18 for *Phase_1* with result smoothing.
  3. Gives the result at a specified coordinate
     Example: *getsingleresult Phase_1 Soil.Suction 0 1* Gives the result for suction a specific coordinate (0 1) for *Phase_1*.

Output commands example: *getsingleresult (gsres)*

- The following five alternatives for the use of the command *getsingleresult (gsres)* can be considered:
  4. Gives the result at a specified coordinate with or without result smoothing.
     Example: *getsingleresult Phase_1 Soil.Suction 3 5 False*
     Gives the result for suction a specific coordinate (3 5) for *Phase_1* without result smoothing.
  5. Gives the result at a specified coordinate with or without result smoothing along a preferred direction.
     Example: *getsingleresult Phase_1 Soil.Suction (3 5) True (1 0)*
     Gives the result for suction a specific point (3 5) for *Phase_1* with result smoothing in the direction of (1 0).

Output objects example: *Soil*

- *Soil* provides different result types for soil .

Property	Long name	Unit	Description
X	X	m	x coordinate
Y	Y	m	y coordinate
Ux	ux	m	Total displacement in x direction
Uy	uy	m	Total displacement in y direction
Utot	|u|	m	Total displacement
dUx	Δux	m	Incremental displacement in x direction
dUy	Δuy	m	Incremental displacement in y direction
dUtot	$\|\Delta u\|$	m	Total incremental displacement
PUx	Pux	m	Summed displacement per phase in x direction
⋮	⋮	⋮	⋮

Output commands example

- Results from the Plaxis 2D 2016 Tutorial 01: Settlement of a circular footing on sand, Case A: Rigid footing.

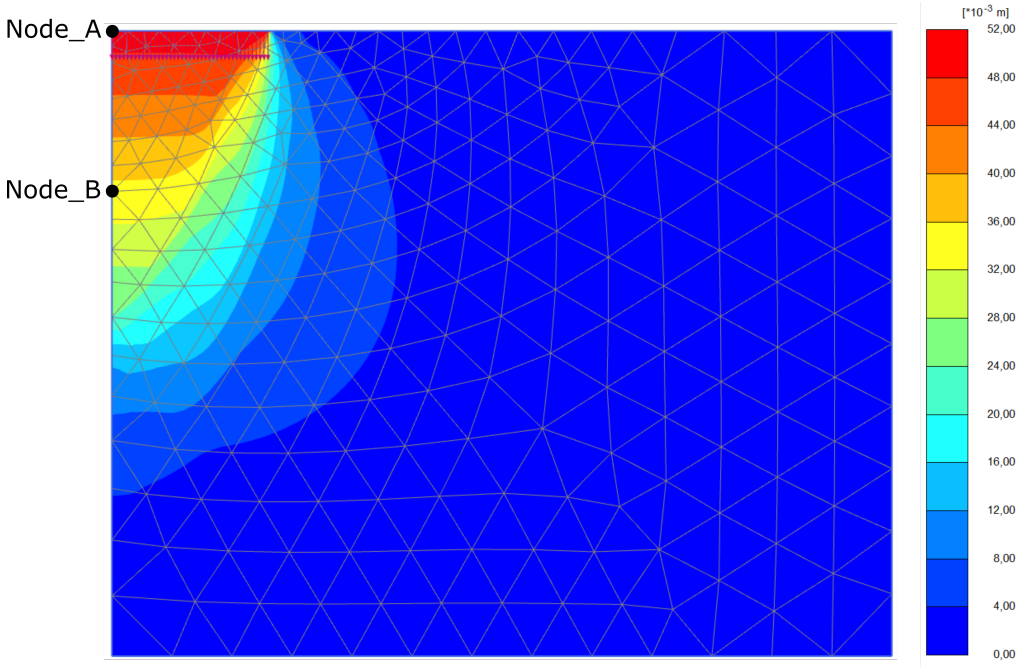
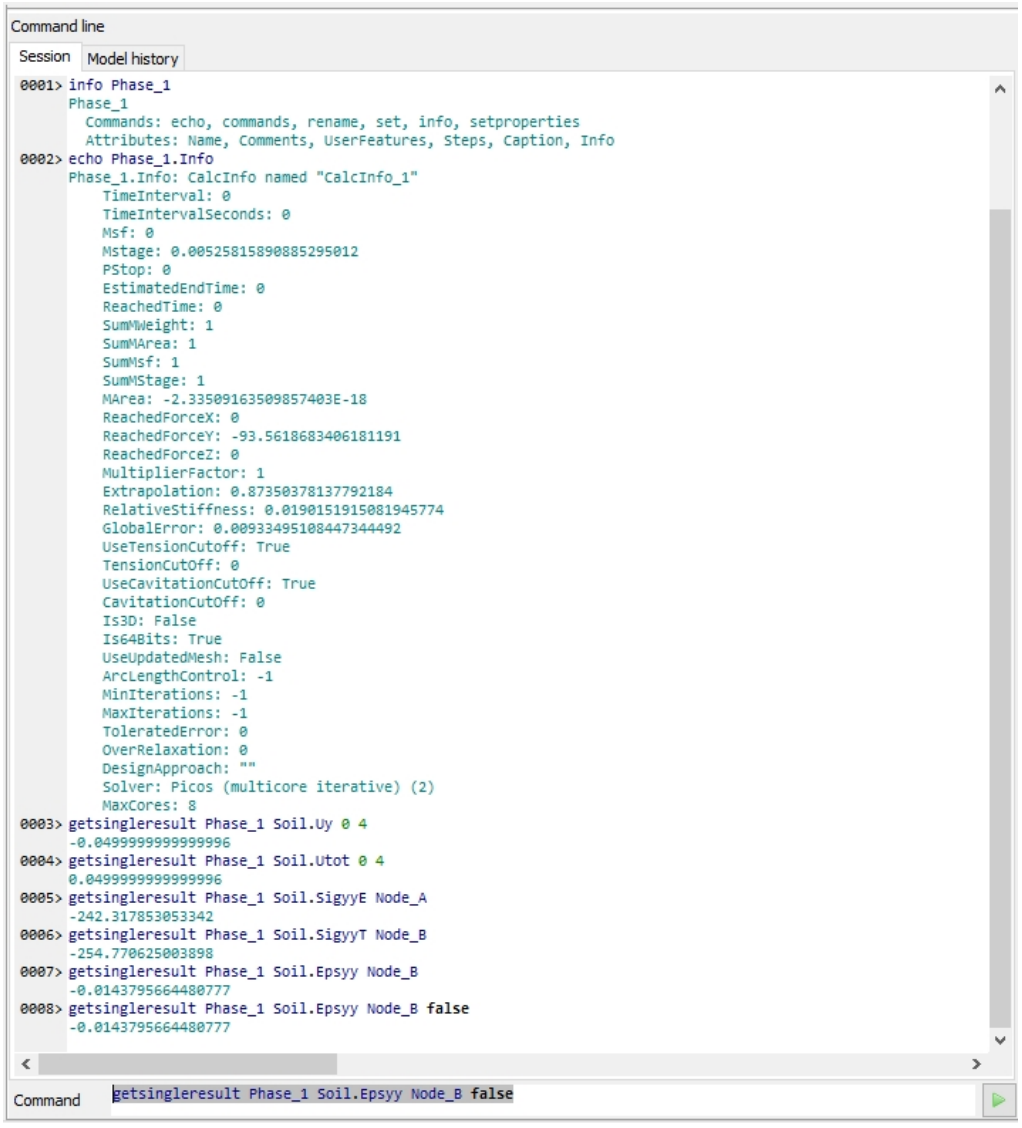


Figure: Total displacement after *Phase_1*

Output commands example



Examine commands

- Commands employed to generate the model can be investigated in *Expert* → *Examine commands...*
- Commands are classified with respect to the mode and qualification.

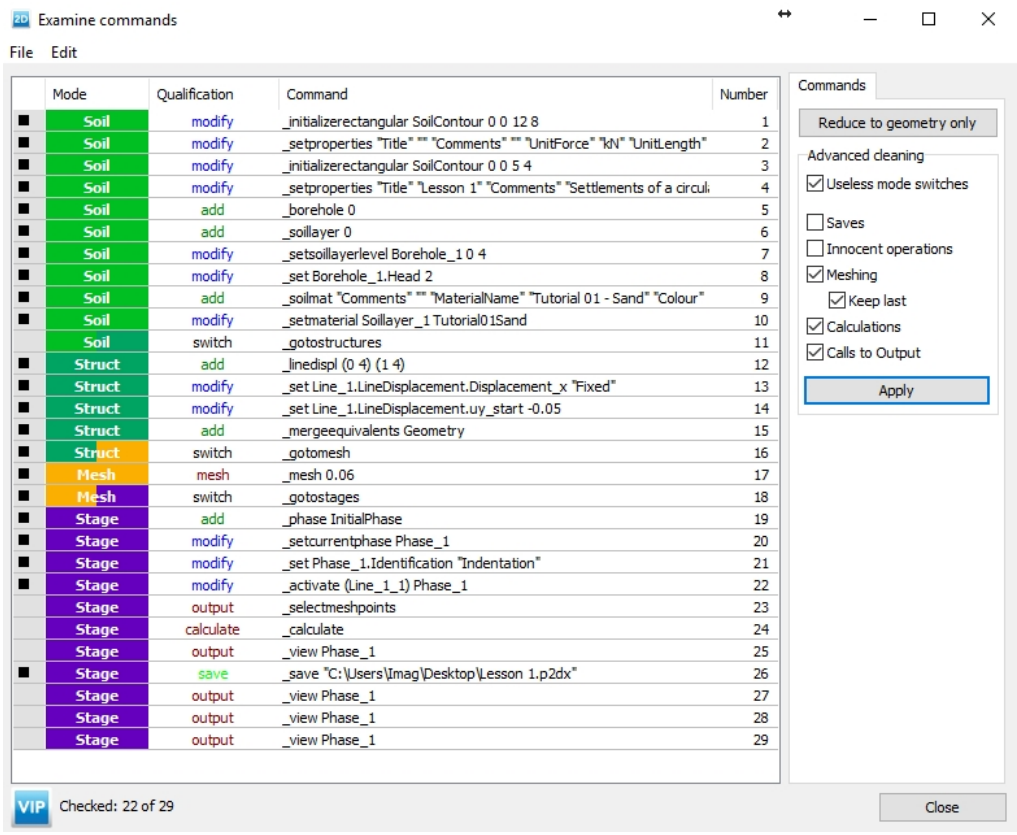


Figure: *Examine commands*.

.p2xlog files

- Commands can be saved in a .p2xlog file by selecting *File* → *Save checked...*
- .p2xlog file can be opened on a compatible Plaxis software to regenerate the model.
- This allows one to share and modify Plaxis models relatively simply.

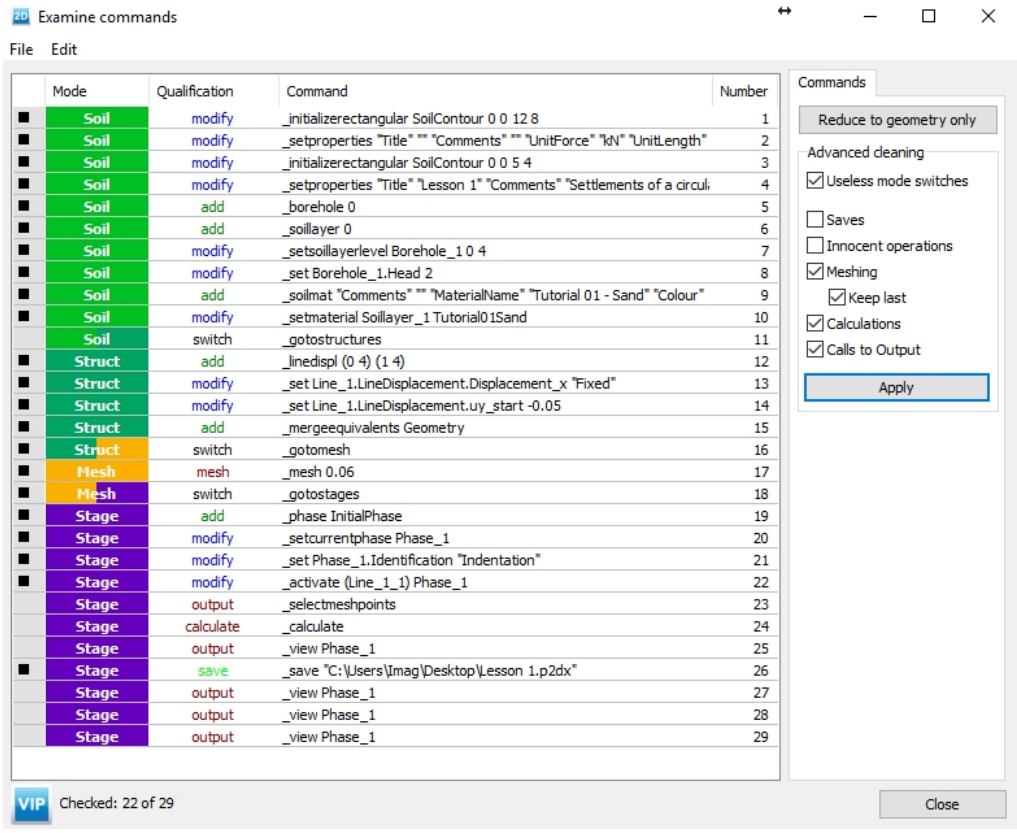


Figure: *Examine commands*.

.p2xlog files

- Prior to saving the commands it is recommended to exclude redundant commands.
- This can be performed manually or by unchecking them on the panel on the right.
- Exclusion of meshing, calculations, and calls to output can reduce the time necessary to regenerate the model.

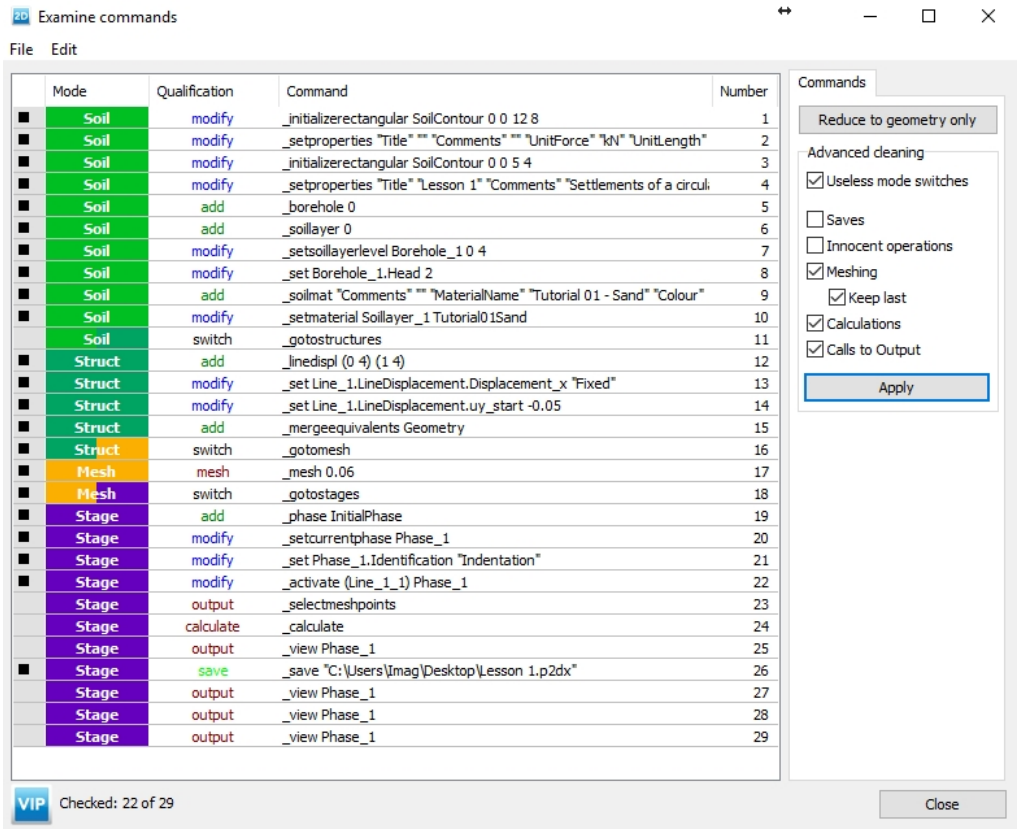


Figure: *Examine commands*.

Command runner

- .p2xlog files can be executed by selecting *Expert* → *Commands runner*
- Several options for the execution of commands are available in *Run*.

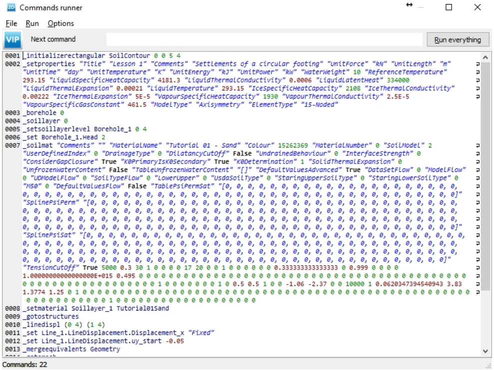


Figure: *Command runner.*

Executing commands with a batch file

- Commands can be executed with a batch file (.bat) as shown in: http://kb.plaxis.nl/videos/practical-use-commands-runner-using-batch-file-run-command-file
- This allows one to automatize the regeneration and calculation of Plaxis models.
- In the case of Plaxis 2D, the batch file should contain the following commands:

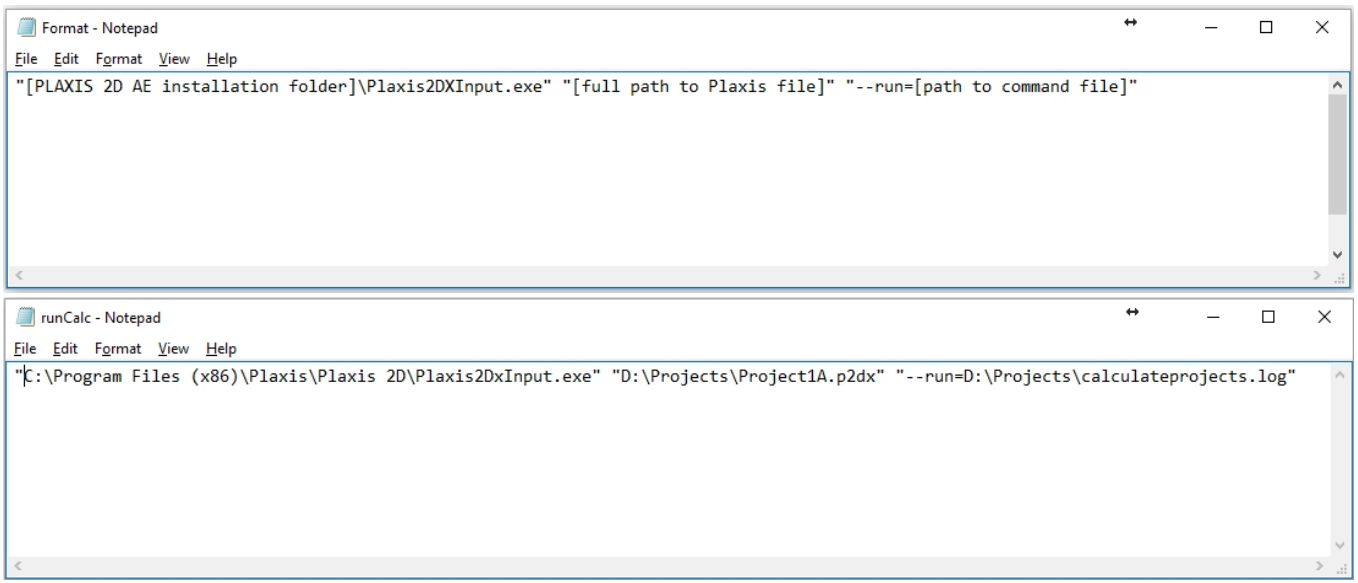


Figure: The format of a batch file and an example.

Introduction

- The *Sensitivity Analysis* and *Parameter Variation* modules provide a toolbox to evaluate the influence of model parameters on calculation results.
- The modules are useful in situation where estimates of the model parameters are associated with uncertainties.
- The effects of uncertainties in the estimates of the model parameters on the model results is evaluated based on the sensitivity score.
- The sensitivity score is obtained by performing a sensitivity analysis where each of the uncertain parameters is individually varied to determine their effect on the model results.
- A parameter variation can be performed after the sensitivity analysis to examine the range of results that can be expected as a results of the variation in the estimates of the model parameters.

Sensitivity Analysis

- The goal of a sensitivity analysis is to evaluate the effect of the variation of a model parameter x within the lower and upper bounds, x_L and x_R, respectively, on the model response, $f(x)$.
- The effect of the variability in x is evaluated with the sensitivity ratio, η_{SR}:

$$\eta_{SR} = \frac{\left[\frac{f(x_R)-f(x_L)}{f(x)}\right] \cdot 100\%}{\left[\frac{x_R-x_L}{x}\right] \cdot 100\%}$$

where $f(x)$ is the reference value of the model response, $f(x_L)$ and $f(x_R)$ are the values of the model response after changing the values of the input parameter.

Sensitivity Analysis

- Given N uncertain model parameters, the sensitivity analysis requires $2N + 1$ calculations.
- A more robust approach to evaluate the sensitivity of the variation is an extension of the sensitivity ratio, known as the sensitivity score, η_{SS}, which is obtained by multiplying η_{SR} with a normalized measure of the variability in the input variable.

$$\eta_{SS} = \eta_{SR} \frac{\max x - \min x}{x}$$

- In the case of a probabilistic analysis, the normalized measure of the variability can be related to the coefficient of variation.

Sensitivity Analysis

- The sensitivity analysis can be generalized by considering the effects of variation in a set of model parameters $\mathbf{x} = [x_1, ..., x_N]^T$ on the set of model responses $A, B, ..., Z$.
- The sensitivity scores for each of the model parameters can be examined in the so-called sensitivity matrix:

Input variable	η_{SS}					
	A	B	$\dots$	Z	$\sum$	α
x_1	$\eta_{SS,A1}$	$\eta_{SS,B1}$	$\dots$	$\eta_{SS,Z1}$	$\sum \eta_{SS,1}$	$\alpha(x_1)$
x_2	$\eta_{SS,A2}$	$\eta_{SS,B2}$	$\dots$	$\eta_{SS,Z2}$	$\sum \eta_{SS,2}$	$\alpha(x_2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_N	$\eta_{SS,AN}$	$\eta_{SS,BN}$	$\dots$	$\eta_{SS,ZN}$	$\sum \eta_{SS,N}$	$\alpha(x_N)$

- The total sensitivity scores of each of the variable, $\eta_{SS,i}$, can be obtained by summing the sensitivity scores for each of the variables.

Sensitivity Analysis

- The total sensitivity scores of each of the parameters, $\eta_{SS,i}$, can be obtained by summing the sensitivity scores for the corresponding parameter.
- The relative sensitivity of each of the parameters is calculated as follows:

$$\alpha(x_i) = \frac{\sum \eta_{SS,i}}{\sum_{i=1}^{N} \sum \eta_{SS,i}}$$

- Note that the results of the sensitivity analysis depend on the selection of the parameter ranges.

Sensitivity Analysis example

- Plaxis 2D 2016 Tutorial 02: Submerged construction of an excavation.
- The goal is to perform a sensitivity analysis of the stiffness parameters in the soil and the diaphragm wall on the horizontal displacement at Node_A in the last excavation phase.

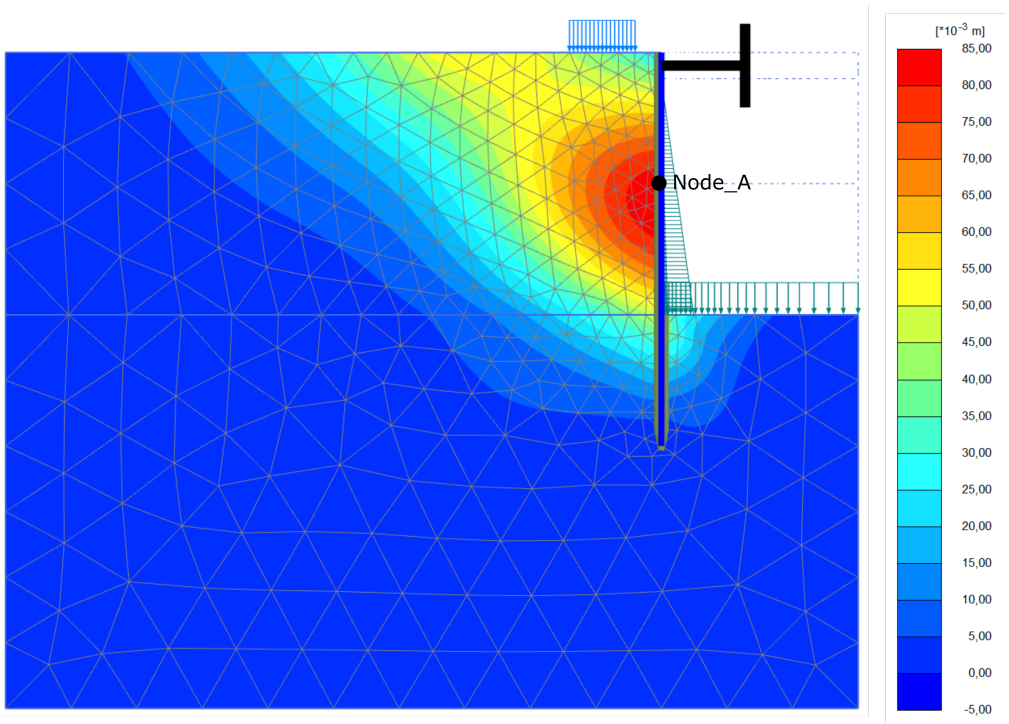


Figure: Horizontal displacements, Tutorial 02.

Sensitivity Analysis example

- The soil parameter are varied within the range of $\pm 10\%$ of the reference value.
- The bending stiffness is varied within $\pm 5\%$ of the reference value.

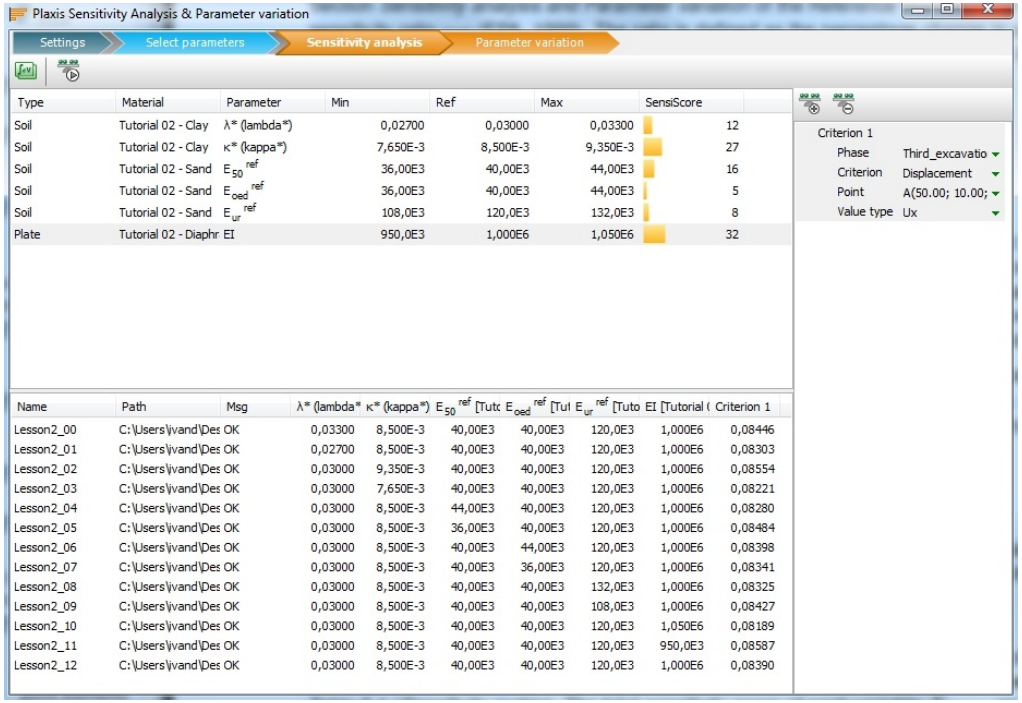


Figure: Results of the sensitivity analysis.

Plaxis scripting in Python

- Plaxis provides a Python wrapper to support Plaxis Input and Output commands.
- In order to use the Python wrapper the following is required:
  - A Plaxis VIP licence
  - A recent version of Python. The latest version of Python can be downloaded from: https://www.python.org/downloads/
  - Basic programming skills in Python. An introduction to programming in Python can be found in various sources, for example: http://www.diveintopython3.net/
  - The firewall should not block the PLAXIS application from accessing the internet, nor must it block other applications (in particular the python.exe executable) from communicating with the remote scripting server inside the PLAXIS application.

Plaxis scripting in Python

- In order to use the Python wrapper the Plaxis remote scripting server needs to be activated.
- The server is activated by: *Expert* $\rightarrow$ *Configure remote scripting server...*

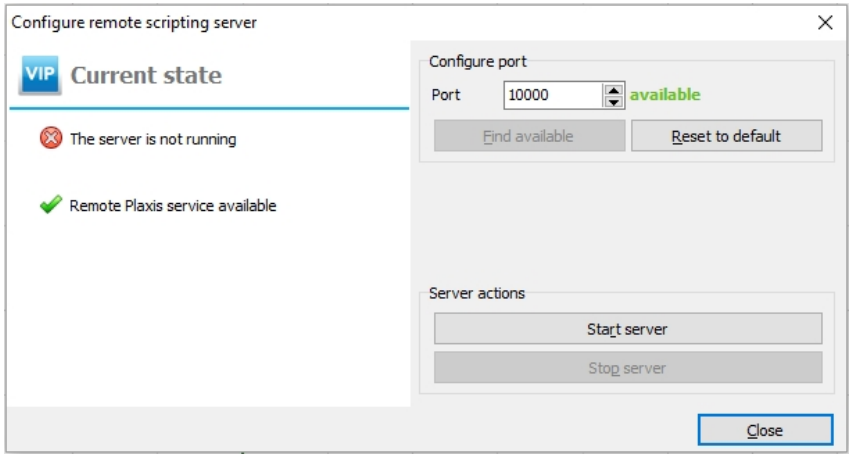


Figure: Configure remote scripting server window.

Plaxis scripting in Python

In order for the remote scripting server to run the following steps need to be completed:

1. A valid PLAXIS VIP licence.
2. A working HTTP connection to the Plaxis Remote scripting Authorization site (i.e. a working internet connection).
3. The local connection port needs to be configured and the server activated.

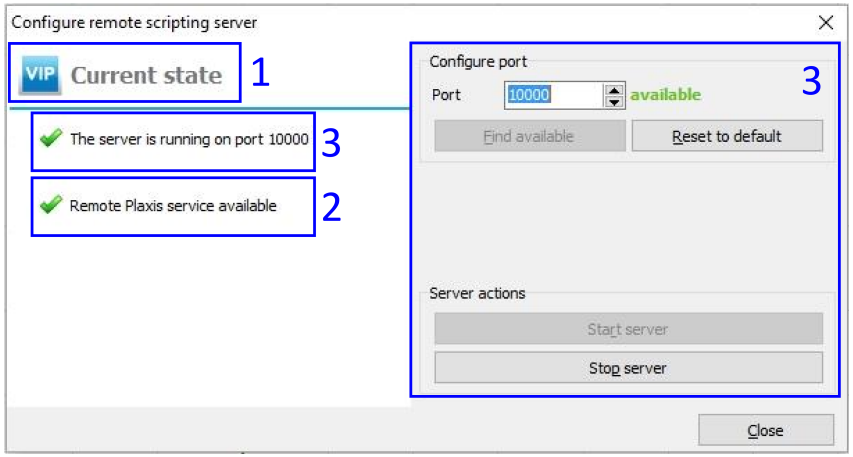


Figure: Remote scripting server setup.

Boilerplate code

- In order for a Python script to be able to communicate with Plaxis, the Plaxis scripting wrapper needs to be imported as a module.
- For a locally running Plaxis application the following boilerplate Python code needs to added prior to the start of a Python script.

Boilerplate code:

```
1 localhostport = 10000
2 plaxis_path = r'C:\Program Files (x86)\Plaxis\PLAXIS 2D'
3
4 import imp
5 found_module = imp.find_module('plxscripting', [plaxis_path])
6 plxscripting = imp.load_module('plxscripting', *found_module)
7 from plxscripting.easy import *
8 s_i, g_i = new_server('localhost', localhostport)
```

Boilerplate code

Boilerplate code:

```
1 localhostport = 10000
2 plaxis_path = r'C:\Program Files (x86)\Plaxis\PLAXIS 2D'
3
4 import imp
5 found_module = imp.find_module('plxscripting', [plaxis_path])
6 plxscripting = imp.load_module('plxscripting', *found_module)
7 from plxscripting.easy import *
8 s_i, g_i = new_server('localhost', localhostport)
```

- The `localhostport` variable should take the value set in the *Configure remote scripting server* window.
- `plaxis_path` is a string that specifies the location of the Plaxis installation.
- From this location the `plxscripting` module is loaded that allows the communication with the Plaxis application.

Boilerplate code

Boilerplate code:

```
1 localhostport = 10000
2 plaxis_path = r'C:\Program Files (x86)\Plaxis\PLAXIS 2D'
3
4 import imp
5 found_module = imp.find_module('plxscripting', [plaxis_path])
6 plxscripting = imp.load_module('plxscripting', *found_module)
7 from plxscripting.easy import *
8 s_i, g_i = new_server('localhost', localhostport)
```

- `s_i` is a variable bound to an object representing the Plaxis Input Application (the Remote scripting server).
- `g_i` is a variable bound to the global object of the current open Plaxis model in Input. This should be used when you want to make changes to the model (e.g. adding a borehole or a point, generating the mesh and start the calculation).

Plaxis Input and Output

- The connections to the Input and Output Plaxis applications should be setup separately.

Boilerplate code:

```
1 localhostport_input = 10000
2 localhostport_output = 10001
3 s_i, g_i = new_server('localhost', localhostport_input )
4 s_o, g_o = new_server('localhost', localhostport_output )
```

Plaxis commands in Python

The use of Plaxis commands in Python is based on the following rules:

- To call a command, prefix the command with the global object reference (in the boilerplate referred to as g_i) and add the parameters in brackets.
- To refer to a Plaxis object, prefix the Plaxis object name also with this global object reference.

Plaxis command	Python equivalent
`borehole 5`	`g_i.borehole(5)`
`set Point_1.x 9.3`	`g_i.Point_1.x = 9.3`

Example of the Plaxis Input commands in Python

```
Python 3.3.5 (v3.3.5:62cf4e77f785, Mar  9 2014, 10:35:05) [MSC v.1600 64 bit (AMD64)] on win32
>>> localhostport = 10000
>>> plaxis_path = r'C:\Program Files (x86)\Plaxis\PLAXIS 2D'
>>> import imp
>>> found_module = imp.find_module('plxscripting', [plaxis_path])
>>> plxscripting = imp.load_module('plxscripting', *found_module)
>>> from plxscripting.easy import *
>>> s_i, g_i = new_server('localhost', localhostport)
>>> # Note that the Plaxis Input is opened and the server is configured on the port 10000
>>> s_i.new()
'OK'
>>> g_i.pointload(1,2,"Fx",1.0,"Fy",-10.0)
[<Point {ACF920C3-F692-4A90-9A97-028AE528FD25}>, <PointLoad {09FD3037-3E9B-4775-975D-D4729EF4E099}>]
>>> g_i.info(g_i.PointLoad_1)
'Point_1.PointLoad\r\n  Commands: echo, commands, rename, set, info, setproperties\r\n  Attributes:
      Name, Comments, UserFeatures, Parent, Fx, Fy, Fz, F, M, LoadFactorLabel, LoadFactor, Fx_design,
       Fy_design, Fz_design, F_design, IsDynamicComponent, M_design, MultiplierFx, MultiplierFy,
      MultiplierFz, MultiplierM'
>>> g_i.echo(g_i.PointLoad_1)
'PointLoad named "PointLoad_1" on Point_1\r\n  Fx: 1\r\n  Fy: -10\r\n  F: 10.049875621120890\r\n  M: 0\
      r\n  Features: 1\r\n    PointLoad named "DynPointLoad_1" on PointLoad_1\r\n      Fx: 0\r\n
       Fy: 0\r\n      F: 0\r\n      M: 0\r\n      MultiplierFx: <not assigned>\r\n      MultiplierFy:
       <not assigned>\r\n      MultiplierM: <not assigned>'
>>> g_i.commands(g_i.PointLoad_1)
"echo\r\n  <no parameters>\r\ncommands (cms)\r\n  <no parameters>\r\n  Text'\r\nrename (rn)\r\n  Text
      '\r\nset\r\n  PlxObject'\r\ninfo\r\n  <no parameters>\r\nsetproperties (sps)\r\n  <PropValue
      {1,...}: <PropValue: Text' Any>'>\r\n"
>>> g_i.rename(g_i.PointLoad_1,"VerticalLoad")
'PointLoad_1 renamed to VerticalLoad'
>>> g_i.set(g_i.VerticalLoad.Fx,0)
'OK'
>>> g_i.echo(g_i.VerticalLoad.Fx)
'Point_1.PointLoad.Fx: 0'
>>> s_i.close()
'OK'
```

Example of the Plaxis Output commands in Python

```
Python 3.3.5 (v3.3.5:62cf4e77f785, Mar  9 2014, 10:35:05) [MSC v.1600 64 bit (AMD64)] on win32
>>> localhostport = 10001
>>> plaxis_path = r'C:\Program Files (x86)\Plaxis\PLAXIS 2D'
>>> import imp
>>> found_module = imp.find_module('plxscripting', [plaxis_path])
>>> plxscripting = imp.load_module('plxscripting', *found_module)
>>> from plxscripting.easy import *
>>> s_o, g_o = new_server('localhost', localhostport)
>>> # Note that the Tutorial01 is opened in the Plaxis Output and the server is configured on the port
      10001
>>> g_o.info(g_o.Phase_1)
'Phase_1\r\n  Commands: echo, commands, rename, set, info, setproperties\r\n  Attributes: Name,
     Comments, UserFeatures, Steps, Caption, Info'
>>> g_o.echo(g_o.Phase_1.Info)
'Phase_1.Info: CalcInfo named "CalcInfo_1"\r\n    TimeInterval: 0\r\n    TimeIntervalSeconds: 0\r\n
      Msf: 0\r\n    Mstage: 0.005258158908852950 12\r\n    PStop: 0\r\n    EstimatedEndTime: 0\r\n
      ReachedTime: 0\r\n    SumMWeight: 1\r\n    SumMArea: 1\r\n    SumMsf: 1\r\n    SumMStage: 1\r\
     n    MArea: -2.33509163509857403E-18\r\n    ReachedForceX: 0\r\n    ReachedForceY:
     -93.5618683406181191\r\n    ReachedForceZ: 0\r\n    MultiplierFactor: 1\r\n    Extrapolation:
     0.87350378137792184\r\n    RelativeStiffness: 0.0190151915081945774\r\n    GlobalError:
     0.009334951084473444 92\r\n    UseTensionCutoff: True\r\n    TensionCutOff: 0\r\n
     UseCavitationCutOff: True\r\n    CavitationCutOff: 0\r\n    Is3D: False\r\n    Is64Bits: True\r
     \n    UseUpdatedMesh: False\r\n    ArcLengthControl: -1\r\n    MinIterations: -1\r\n
     MaxIterations: -1\r\n    ToleratedError: 0\r\n    OverRelaxation: 0\r\n    DesignApproach: ""\r
     \n    Solver: Picos (multicore iterative) (2)\r\n    MaxCores: 8'
>>> g_o.getsingleresult(g_o.Phase_1,g_o.Soil.Uy,0,4)
'-0.0499999999999996'
>>> g_o.getsingleresult(g_o.Phase_1,g_o.Soil.Utot,0,4)
'0.0499999999999996'
>>> g_o.getsingleresult(g_o.Phase_1,g_o.Soil.SigyyE,g_o.Node_A)
'-242.317853053342'
>>> g_o.getsingleresult(g_o.Phase_1,g_o.Soil.SigyyT,g_o.Node_B)
'-254.770625003898'
>>> g_o.getsingleresult(g_o.Phase_1,g_o.Soil.Epsyy,g_o.Node_B,False)
'-0.0143795664480777'
```