Plaxis Scripting Tutorial
Advanced Course on Computational Geotechnics

Ivan Depina
SINTEF Infrastructure
Rock and Geotechnical Engineering

September 23, 2016

1 Introduction

The following tutorial on Plaxis scripting is based on the Plaxis 2D Tutorial 02: Submerged construction of an excavation. The information on the problem and the corresponding .p2dxlog file are available on http://kb.plaxis.nl/tutorials/2d-tutorial-02-submerged-construction-excavation. Additionally, the information on the problem can be examined in the Plaxis 2D Tutorial Manual 2015.
This tutorial is organized to provide an introduction to the use of Plaxis commands in Section 2. An introduction to the use of the Plaxis Sensitivity Analysis is provided in Section 3. The tutorial concludes with an inverse analysis problem in Section 4 that examines several practical aspects related to the use of Plaxis commands in Python.

2 Plaxis Input and Output commands

2.1 Use of the Command runner

- Download the following files related to the PLAXIS 2D 2016 - Tutorial Manual Lesson 2 from http://kb.plaxis.nl/tutorials/2d-tutorial-02-submerged-construction-excavation:
  - PLAXIS2D2016-Tutorial-Lesson02.pdf
  - PLAXIS2D2016-Tutorial-Lesson02.p2dxlog

  The .pdf file provides a reference for the problem, while the .p2dxlog file is a command log file that contains all the necessary commands to regenerate the model in Plaxis.

- Start the Plaxis 2D Input application and select *Start a new project* on the *Quick select* window.

- Press *OK* on the *Project properties* window
- Open the *Commands runner* by selecting from the menu *Expert* → *Run commands...* or *Ctrl+F9*
- From the menu of the *Commands runner* window select *File* → *Open...* or *Ctrl+O*
- Locate the *PLAXIS2D2016-Tutorial-Lesson02.p2dxlog* file and select *Open*
- From the menu of the *Commands runner* window select *Run* → *Run everything*
- Save the project by selecting *File* → *Save project* or *Ctrl+S* from the main menu

2.2 Plaxis Input commands

- Later in the tutorial we will examine ground settlements at the surface level, 10 m from the excavation. In order to ensure that Plaxis assigns a mesh node at the considered location, a point needs to be added at coordinates x=40.0 m, y=20.0 m.
  The point will be added by the use of the Plaxis Input commands. The *Command reference* can be accessed by selecting *Help* → *Command reference*.
- Before a point can be added to the model, it is necessary to select the *Structures* mode in the Plaxis Input. Examine the command *gotostructures (gtt)* in *Help* → *Command reference* → *Input commands reference* → *gotostructures*.
  Type the following command in the *Command line*:

```
gotostructures
```

- Use the command *point (pt)* to add a point at x=40.0 m and y=20.0 m. Examine the *point (pt)* command in the *Input commands reference* and select one of the alternatives. For example, the following command can be typed in the *Command line*:

```
point(40 20)
```

- Note that a point object *Point_12* was created as a result of the previous command. The properties of a point object can be examined in *Help* → *Command reference* → *Input objects reference* → *Point*.
- Use the command *info* to display all available commands and attributes of the object *Point_12*.

```
info Point_12
```

- Use the command *echo* to display the details of the object *Point_12*.

```
echo Point_12
```

- Use the command *echo* to display the x coordinate of the object *Point_12*.

```
echo Point_12.x
```

- The properties of the line load next to the excavation will be modified in the following section. The line load will be renamed to *VerticalLoad*, extended for 1.0 m in the left direction and the vertical component of the line load will be updated to -4.17 kN/m/m.
  First, display a list of all the line loads with the command *__dumplineloads (__dll)*:

```
__dll
```

  From the output of the *__dumplineloads (__dll)* command it can be observed that there is a single line load, *LineLoad_1*, associated with the line, *Line_5*.

- Use the commands *info*, *echo*, *rename*, and *set* to modify the *LineLoad_1*. For example, the changes can be implemented with the following commands:

```
info LineLoad_1
rename LineLoad_1 "VerticalLoad"
echo VerticalLoad
echo Line_5
echo Point_10
echo Point_11
set Point_10.x 42
set VerticalLoad.qy_start -4.17
```

- In the following section, the mesh of the finite element model will be generated and the calculations will be performed. In order to mesh the model use the *gotomesh (gtm)* command to open the *Mesh* mode. The mesh is generated with the *mesh* command.

```
gotomesh
mesh
```

- In order to calculate the model, the *Staged construction* mode needs to be opened with the *gotostages (gts)* command.

```
gotostages
```

- Prior to the calculation it is necessary to select points for curves. The points are selected by opening the Plaxis Output with the command *selectmeshpoints (smp)*.

```
selectmeshpoints
```

Note that the *selectmeshpoints (smp)* command opens the Plaxis Output application. In order to select points for curves it is necessary to use the Plaxis Output commands that can be accessed by selecting *Help* → *Command reference*.

- Points for curves are selected with the *addcurvepoint (acp)* command that can be examined in *Help* → *Command reference* → *Output commands reference*. When selecting points for curves special attention should be given to different clusters generated during the model construction.
  In this tutorial, two points are to be selected; Node A at x=50.0 m and y=10.0 m, and Node B at x=40.0 and y=20.0 m. Node A is found in the cluster *Plate_1_3* corresponding to the diaphragm wall, while the Node B is found in the cluster *Soil_1_3*.
  The following commands can be typed in the *Command line*:

```
addcurvepoint "Node" Plate_1_3 (50 10)
addcurvepoint "Node" Soil_1_3 (40 20)
```

  Close the Plaxis Output by selecting the *Update* button.

- Calculate and save the project with the commands *calculate (ca)* and *save (sv)*.

```
ca
sv
```

- Open the results for the last calculation phase (i.e., *Phase_5*) with the command *view (vr)*.

```
view Phase_5
```

2.3 Plaxis Output commands

- Investigate the object corresponding to the last calculation phase with the *info* and *echo* commands and access the ΣMsf in the Plaxis Output *Command line*.

```
info Phase_5
echo Phase_5
echo Phase_5.Info.SumMsf
```

- Use the function *getsingleresult (gsres)* to obtain the following results:
  - Horizontal displacements at Node A after *Phase_4*
  - Vertical displacements at Node B after *Phase_5*
  - Base heave at the center of the excavation after *Phase_5*
  - Vertical effective stress at depth of 2.0 m below Node B after *InitialPhase*

```
getsingleresult Phase_4 Soil.Ux Node_A
getsingleresult Phase_5 Soil.Uy Node_B
getsingleresult Phase_5 Soil.Uy (65 0)
getsingleresult InitialPhase Soil.SigyyE (40 18)
```

3 Sensitivity analysis

A *Sensitivity Analysis* is performed to evaluate the effects of parameter variation on the model response. In this tutorial, the effect of variation in the soil stiffness parameters and the bending stiffness of the diaphragm wall on the ground settlements at the Node B in *Phase_5* will be examined, as illustrated in Figure 1.

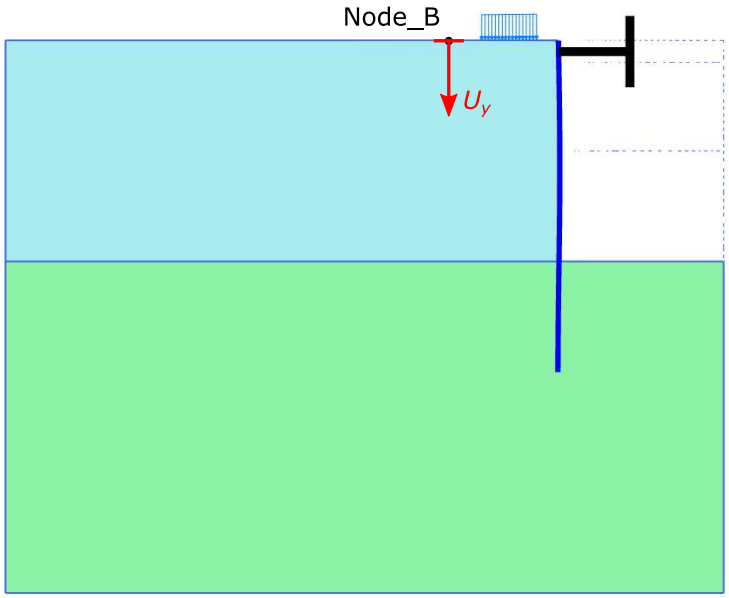

Figure 1: Ground settlements at Node B, denoted as U_y, after *Third_excavation* phase (i.e., *Phase_5*).

- Activate the *Sensitivity Analysis and Parameter Variation* module by selecting *Expert* $\rightarrow$ *Sensitivity and parameter variation*.
- Activate the *Remote scripting server*. Select *Start server* and *Close* the window.
- From the parameters of the Soft soil model for the upper clay layer select κ^* and λ^* by dragging them to the panel on the right.
  λ^* is related to the oedometer modulus E_{oed}^{ref}: $E_{oed}^{\text{ref}} = \frac{p^{\text{ref}}}{\lambda^*}$, where $p^{\text{ref}} = 100$ kPa.

κ^* is related to the unloading/reloading modulus E_{ur}: $\frac{E_{ur}}{3(1-2\nu_{ur})} = \frac{p'+c\cot\varphi}{\kappa^*}$, where p' is the mean effective stress, c is the cohesion, φ is the friction angle, ν_{ur} is the Poisson's ratio for unloading/reloading.

- Select the parameters E_{50}^{ref}, E_{oed}^{ref}, and E_{ur}^{ref} from the parameters of the Hardening soil model for the sand layer.
- Select the bending stiffness, EI, of the diaphragm wall.
- A variation of $\pm 10\%$ is considered for the soil parameters, while a variation of $\pm 5\%$ is considered for the bending stiffness of the diaphragm wall. The *Min* and *Max* values should correspond to the values in Figure 2.
- Select the *Sensitivity analysis* tab and add a criterion by selecting the *Add criterion* icon on the panel on the right. The properties of the criterion are defined as follows:

Phase	Third_excavation [Phase_5]
Criterion	Displacement
Point	B(40.00;20.00;0.00)
Value type	Uy

- Perform a *Sensitivity Analysis* by selecting the *Run analysis* icon in the upper left corner. The results of the *Sensitivity Analysis* can be examined in Figure 2.
- Should the strength parameters be included in the *Sensitivity Analysis*?

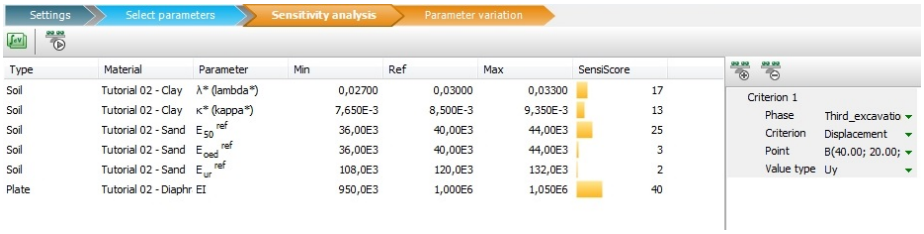

Figure 2: Results of the Sensitivity Analysis.

4 Inverse analysis with Plaxis commands in Python

Consider that a measurement of the horizontal displacement of the diaphragm wall at Node A of $u'_{xA} = 0.024$ m in the positive x direction is available after the *Second_excavation* phase (i.e., *Phase_4*), as illustrated in Figure 3. Perform an inverse analysis using Plaxis commands in Python to obtain the most likely combination of the κ^* and λ^* parameters of the clay layer that corresponds to the measurement. In this case, it is assumed that a prediction of the horizontal displacements of the

diaphragm wall with the Plaxis model at Node A is mostly influenced by the stiffness parameters of the clay layer. For example, this can be further examined by conducting a *Sensitivity Analysis*.

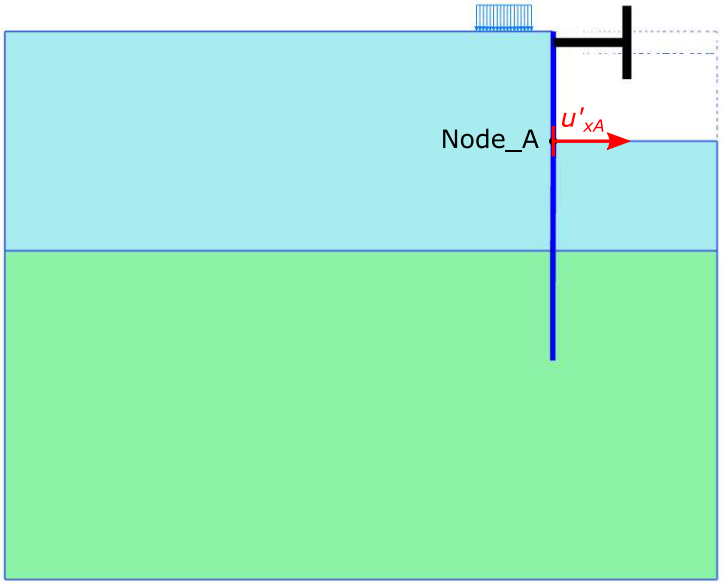

Figure 3: Horizontal displacement of Node A, u'_{xA}, after *Second_excavation* phase (i.e., *Phase_4*).

- The inverse problem can be formulated as an optimization problem, such that the difference between the measured horizontal displacement of the diaphragm wall at Node A, u'_{xA}, and the displacement predicted by the Plaxis model at the corresponding node, $u_{xA}(\lambda^*, \kappa^*)$ is minimized. The optimization problem is defined as follows:

$$\{\lambda^*_{\min}, \kappa^*_{\min}\} = \arg\min \epsilon(\lambda^*, \kappa^*)$$

$$= \arg\min |u'_{xA} - u_{xA}(\lambda^*, \kappa^*)| \tag{1a}$$

subject to

$$0.8 \cdot \lambda^*_{\text{ref}} \leq \lambda^* \leq 1.2 \cdot \lambda^*_{\text{ref}} \tag{1b}$$

$$0.8 \cdot \kappa^*_{\text{ref}} \leq \kappa^* \leq 1.2 \cdot \kappa^*_{\text{ref}} \tag{1c}$$

where $\{\lambda^*_{\min}, \kappa^*_{\min}\}$ is the solution of the inverse problem, $\epsilon(\lambda^*, \kappa^*)$ is the objective function defined as an absolute value of the difference between u'_{xA} and $u_{xA}(\lambda^*, \kappa^*)$, $\lambda^*_{\text{ref}} = 0.03$ and $\kappa^*_{\text{ref}} = 8.5{\cdot}10^{-3}$ are the reference values used to specify bounds of the search space. The bounds are selected as $\pm 20\%$ of

the reference values to represent uncertainties in the estimates of the reference values. The selection of bounds is case-specific and should be estimated from the available information, while accounting for different sources of uncertainty (e.g., natural variability of soil properties, interpretation uncertainty).

- The solution to the inverse problem can be found by implementing an optimization algorithm. The Python module *scipy.optimize* provides a wide range of algorithms that can be implemented to obtain the solution of the inverse problem. More information can be found on:
http://docs.scipy.org/doc/scipy/reference/optimize.html.

- In this tutorial, the solution to the inverse problem in Eq. 1 will be simplified by discretizing the search domain, defined by Eqs. 1b and 1c, into 10×10 equally spaced points and implementing the exhaustive search optimization algorithm. An estimate of the solution, $\left\{\hat{\lambda}^*_{\min}, \hat{\kappa}^*_{\min}\right\}$, will correspond to the values of λ^* and κ^* in the discretized space with the lowest value of the objective function.

- Note that although the implementation of the exhaustive search algorithm is relatively simple, the algorithm is usually associated with relatively high computational demands. For this reason, more advanced and computationally efficient optimization algorithms should be considered in practical projects. For example, one can examine the algorithms implemented in the *scipy.optimize* module.

- An example of a Python code that can be applied to solve the inverse problem follows. Examine the code and try to understand the overall structure of the code. Examine the syntax of the Plaxis commands in Python.

```
1 # ----------------------------------------------------------
2 # Short description and instructions for use.
3 # ----------------------------------------------------------
4
5 # Python code for the inverse problem is based on the
    PLAXIS2D2016-Tutorial-Lesson02 and defined by an
    observation of horizontal displacement of 0.024 m in
    the positive x direction at Node_A (i.e., at location
    (50,10)), after the second excavation phase.
6
7 # Note that prior to the analysis, the Plaxis project
    based on the code in PLAXIS2D2016-Tutorial-Lesson02.
    p2dxlog needs to be opened and calculated with Node_A
    defined on the coordinate (50,10). The remote
    scripting server in the Plaxis Input needs to be
    started on the port 10000. Additionally, the Plaxis
    Output should be started WITHOUT AN ACTIVE PROJECT and
     the remote scripting server started on the port
    10001.
8
```

```
# ------------------------------------------------------------
# Set up Plaxis scripting
# ------------------------------------------------------------
# Specify the local ports for the Plaxis Input and Plaxis Output
localhostport_input = 10000
localhostport_output = 10001

# Specify the boilerplate code
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 *

# Connect to the Plaxis Input and Plaxis Output servers
s_i, g_i = new_server('localhost', localhostport_input )
s_o, g_o = new_server('localhost', localhostport_output )

# ------------------------------------------------------------
# Import additional modules
# ------------------------------------------------------------
# NumPy is the fundamental package for scientific computing with Python
import numpy as np
# Matplotlib is a 2D plotting library
import matplotlib.pyplot as plt

# ------------------------------------------------------------
# Define the objective function
# ------------------------------------------------------------
def objfun(x):
  # Note that x is a row vector with the first value, x[0], corresponding to lambda*, and the second value, x[1], corresponding to kappa*

  # Specify the horizontal measurement at Node A
  uxAMeas=0.024 # m

  # Update the parameters in the Plaxis model
  # lambdaModified parameter
  g_i.set(g_i.Tutorial02Clay.lambdaModified,x[0])
  # kappaModified parameter
  g_i.set(g_i.Tutorial02Clay.kappaModified,x[1])

  # Move to stages
  g_i.gotostages()

```

```
    # Set the phases to calculate
    for phase in g_i.Phases: # Note that g_i.Phases is a
      list of phases
      # Set phases to calculate
      g_i.set(phase.ShouldCalculate,True)

    # Calculate the project
    g_i.calculate()

    # Save the project
    g_i.save()

    # Open Phase_4
    g_i.view(g_i.Phase_4)

    # Obtain the predicted value of the horizontal
      displacement at Node A from the Plaxis Output for the
      specified values of lambda* and kappa*
    uxACalc=np.float(g_o.getsingleresult(g_o.Phase_4,g_o.
      Soil.Ux,g_o.Node_A))

    # Close the project in the Plaxis Output window
    s_o.close()

    # Value of the objective function
    objFunVal=np.abs(uxAMeas-uxACalc)

    # Return the objective function value
    return objFunVal

# ----------------------------------------------------------
# Optimization algorithm
# ----------------------------------------------------------
# Reference values
ref=[0.03,8.5e-3]

# Define bounds
bounds=((0.8*ref[0],1.2*ref[0]),(0.8*ref[1],1.2*ref[1]))

# Discretize the domain
# Number of discretization points
nLambda=10
nKappa=10

# Row vectors for the discretization of the domain
lambdaLin=np.linspace(bounds[0][0],bounds[0][1],nLambda)
kappaLin=np.linspace(bounds[1][0],bounds[1][1],nKappa)

# Create a mesh of points
lam,kappa=np.meshgrid(lambdaLin,kappaLin)
```

```
99
100 # Allocate an array to store the objective function
        values
101 objVal=np.zeros(lam.shape)
102
103 # Calculate the values of the objective function at the
        discretized points
104 for i in xrange(nKappa):
105   for j in xrange(nLambda):
106     objVal[i,j]=objfun([lam[i,j],kappa[i,j]])
107
108 # Locate the minimum with the numpy.argmin function
109 objMin=np.amin(objVal)
110 # Find the indices of the objVal array that correspond to
        the minimum with the numpy.where function
111 indMin=np.where(objVal==objMin)
112
113 # lambda* value corresponding to the minimum
114 lamMin=lam[indMin[0],indMin[1]]
115
116 # kappa* value corresponding to the minimum
117 kappaMin=kappa[indMin[0],indMin[1]]
118
119 # ------------------------------------------------------------
120 # Print the results and the log values of the objective
        function
121 # ------------------------------------------------------------
122 # Print the results
123 print('The minimum value of the objective function is %e,
        with the corresponding\
124 values of lambda*=%e and kappa*=%e.' % (objMin, lamMin,
        kappaMin))
125
126 # Plot the log values of the objective function and the
        minimizer
127 # Create a figure
128 fig=plt.figure()
129 ax = fig.add_subplot(111)
130
131 # Plot the contour plot of the log10 values of the
        objective function
132 CS = plt.contour(lam, kappa, np.log10(objVal),9)
133 plt.clabel(CS, inline=1, fontsize=10)
134
135 # Plot the minimum denoted with a star marker
136 SC=plt.scatter(lamMin,kappaMin,s=100,marker='*')
137
138 # Set the figure title
139 plt.title('Log10 values of the objective function')
140 # Set the axes titles
```

```
plt.xlabel('lambda*')
plt.ylabel('kappa*')

# Show the plot
plt.show()
```

- After evaluating the Python code, an estimate of the solution of the inverse problem $\left\{\hat{\lambda}^*, \hat{\kappa}^*\right\} = \left\{0.028, 7.178 \cdot 10^{-3}\right\}$ is obtained with the value of the objective function of $\epsilon(\hat{\lambda}^*, \hat{\kappa}^*) = 5.687 \cdot 10^{-5}$. Log values of the objective function, $\log_{10} \epsilon$, are presented in Figure 4, with a star marker ($\star$) denoting the location of the inverse problem solution estimate.

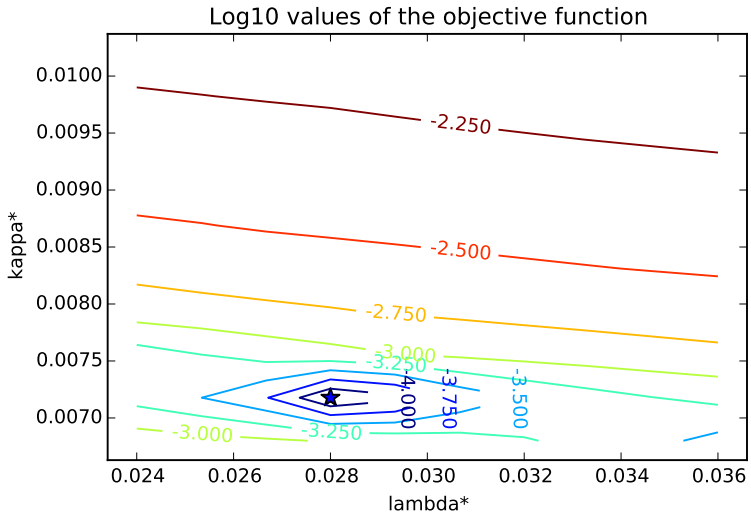

Figure 4: Log values of the objective function, $\log_{10} \epsilon$, with a star marker ($\star$) denoting the location of the inverse problem solution estimate.

- Additionally, one can examine the effect of the information provided by the measurement on the prediction of the diaphragm wall displacements and ground settlements in the last excavation phase.