ZSoil

for geotechnics & structures

ZSOIL + R

A MATLAB BASED MODULE LINKING ZSOIL AND UQLAB

Report 170823

(revised 03.11.2023)

J. Minini

GeoDev.

PO Box CH-1001 Lausanne
Switzerland
https://zsoil.com

Contents

Chapter 1

Introduction

This document contains the required documentation for using the ZS+R module. ZS+R provides a link between ZSOIL, MATLAB and UQLAB, allowing the user to :

I. Read the output data from a ZSOIL calculation in MATLAB.

II. Run ZSOIL calculations (sequential + parallel) from MATLAB based on a template file.

III. Perform probabilistic analyses using UQLAB.

IV. Print or display the results with the standard plotting functions in MATLAB.

In this context, ZSOIL abstractly becomes a function which can be seen as a back-box model, allowing the user to perform any kind of calculation directly in the MATLAB environment. The global execution sequence is described in the flowchart shown in the following figure.

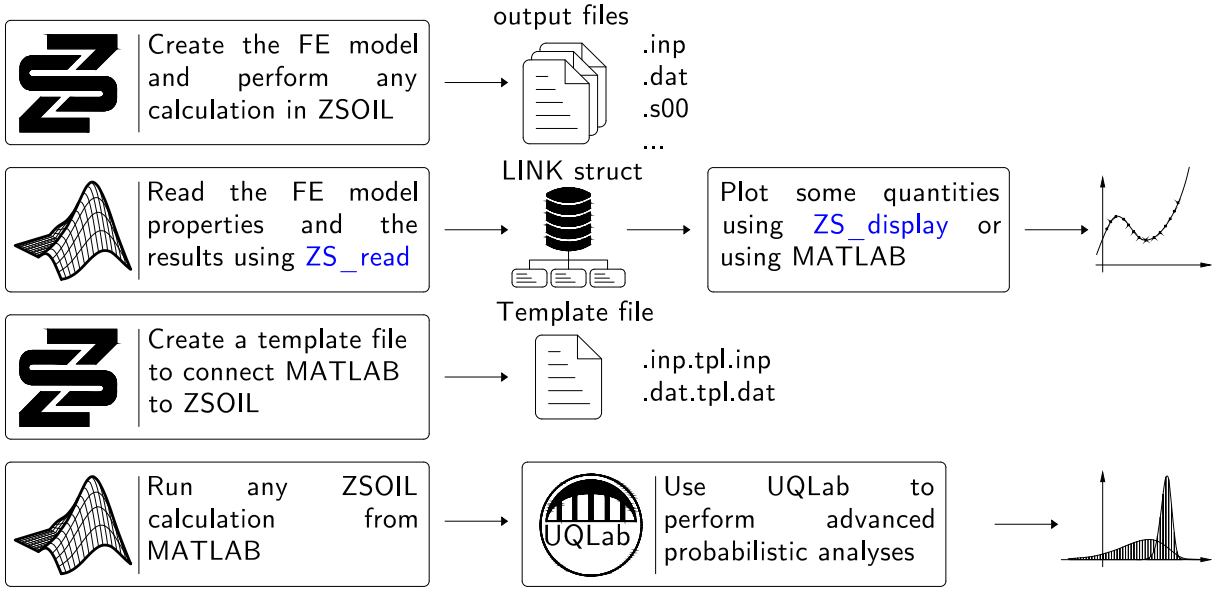


Figure 1.1: Flowchart of the ZS+R module procedure.

Chapter 2

Function Repository

2.1 ZS_createModel

The ZS_createModel function is a faster alternative to the original uq_createModel provided in UQLab. It simplifies the creation of a uq_Model class object of type 'UQLink' with a more direct ZSoil-oriented approach. Documentation related to the creation of a uq_Model class object of type 'UQLink' can be found in (Lataniotis et al. 2022, Moustapha et al. 2022) or by typing ZS_help model and ZS_help uqlink in the MATLAB console.

```
OUT = ZS_createModel (USER_OPTS)
```

Input arguments	Type of object
USER_OPTS	See Env. 2.1.1

Output arguments	Type of object
OUT	uq_Model class object

Environment 2.1.1 (USER_OPTS)

	Field name	Type of object	Description
•	.Name	String	Name of the model (often identical to the .inp file)
•	.Parser	See Env. 2.1.2	Parser informations
⊡	.Template	See Env. 2.1.3	Template informations
⊡	.ZSoil	See Env. 2.1.4	Options for the ZSoil software to be used
⊡	.ExecutionPath	String pwd (default)	Location where the ZSOIL calculations will be run
⊡	.Counter	Integer 1 (default)	Options defining how the numbering of the created ZSoil files will be carried out
⊡	.Archiving	String 'delete' (default) 'ignore' 'save'	Specifies how the files generated during execution will be handled Delete all generated files Do nothing Keep all files

Environment 2.1.2 (.Parser)

	Field name	Type of object	Description
•	.Name	String (default)	Name of the parser file (.m).
⊡	.Path	String	Location of the .m parser file. Note : if not provided, it will be assumed that the parser function already belongs to the current MATLAB path.

Environment 2.1.3 (.Template)

	Field name	Type of object	Description
⊡	.Name	String USER_OPTS.Name (default)	Name of the template file (.inp.tpl.inp or .dat.tpl.dat). If not provided, ZS_createModel will assume that a template file named 'USER_OPTS.Name' exists
⊡	.Path	String pwd (default)	Location of the template file

Note :ZS_createModel automatically detects the file extension of the template (.dat or .inp).

Environment 2.1.4 (.ZSoil)

	Field name	Type of object	Description
⊡	.Version	String Latest (default)	Version of ZSoil to be used (ex: 'v20.07'). If not provided, the latest version on the machine will be used
⊡	.SilentMode	String 'off' (default) 'on'	Option for launching ZSoil in silent mode (without the dialog window)

2.2 ZS_createParser

The ZS_createParser function will open a new script function already containing the standard parser syntax. The user should then fill the section delimited with '%%' to parse some results, by calling ZS_read for instance.

ZS_createParser does not take any arguments and does not return any.

```
ZS_createParser ()
```

2.3 ZS_evalModel

ZS_evalModel is the ZSoil-oriented version of the uq_evalModel function implemented in UQLab. It allows for the evaluation of the link model, in other words, it facilitates running ZSoil calculations sequentially from MATLAB. It's important to note that ZS_evalModel does not automatically save the input matrix X and the output OUT in a .mat file.

```
OUT = ZS_evalModel (MODEL , X)
```

Input arguments	Type of object
MODEL	uq_Model class object
X	Double array

Output arguments	Type of object
OUT	Depend on the parser

2.4 ZS_generateINP

In accordance with the MODEL and the input matrix X, ZS_generateINP will create the ZSoil input file (.inp or .dat) in a temporary folder as specified by the user options in USER_OPTS.ExecutionPath \temp_Execution. The display options DISP_OPTS are used to control the visibility of pop up boxes.

```
ZS_generateINP (MODEL , X , DISP_OPTS)
```

Input arguments	Type of object
Model	uq_Model class object
X	Double array
DISP_OPTS	Logical (default = false)

2.5 ZS_help

ZS_help will open the documentation related to NAME. If ZS_help is used without an argument, the ZS+R user manual will be opened. Type ZS_help ? in the MATLAB console to view the list of supported NAME arguments.

```
ZS_help (NAME)
```

Input arguments	Type of object
NAME	Character array (optional)

2.6 ZS_parallel_evalModel

ZS_parallel_evalModel is the parallel version of ZS_evalModel , enabling multiple model evaluations simultaneously. The number of parallel calculations depends on the number of available cores on the machine. To prevent instability and crashes, the number of jobs per core is limited to two. The default number of threads is 4. ZS_parallel_evalModel divides the input into multiple blocks and saves the results in a temporary file located according to the user options USER_OPTS.ExecutionPath (see Env. 2.1.1). When all blocks are evaluated, ZS_parallel_evalModel automatically saves the results in a .mat file at the same location.

OUT = ZS_evalModel (MODEL , X , NThreads)

Input arguments	Type of object
MODEL	uq_Model class object
X	Double array
NThreads	Integer (default = 4)

Output arguments	Type of object
OUT	Depend on the parser

Important note: When using ZS_evalModel or ZS_parallel_evalModel , the output argument can be any kind of MATLAB object (arrays, cells, structures). However, when using uq_evalModel , the output argument is restricted to a double array. This restriction is related to the requirement to return numerical values when using active learning methods. In this context, it is strongly recommended to use ZS_evalModel or ZS_parallel_evalModel only for generating experimental designs.

2.7 ZS_read

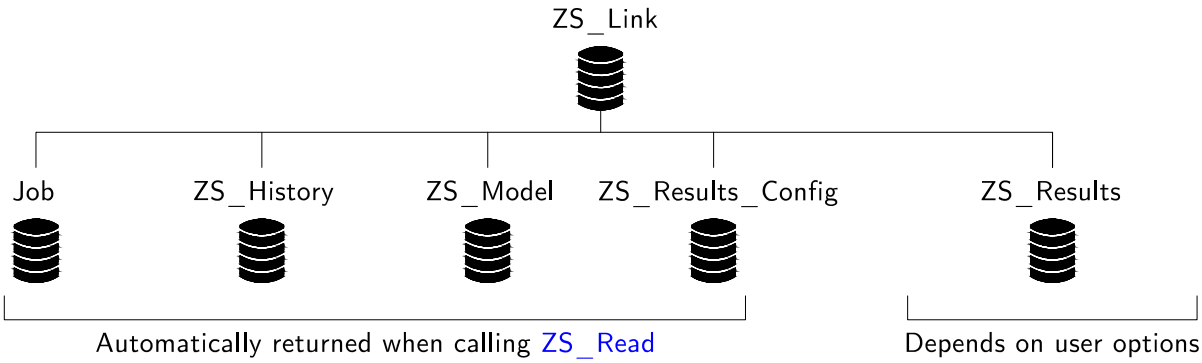


Figure 2.1: Properties of the ZS_Link class object.

Any ZSOIL calculation is composed of multiple objects saved under several output files, from the first user-defined node position until the displacement of this same node after any simulation. In the standard ZSOIL interface, reading these quantities is managed by the ZSOIL Pre- and Post-processor. In the ZS+R environment, reading the results is performed by a unique function ZS_read , which will return a ZS_Link class object. It contains 5 properties, for which one is dynamically added (ZS_Results) according to the user input. Below is a brief overview of the ZS_Link class objects. For a complete description of the class, please refer to chapter A.

- Job: contains the name (without file extension) and the path of the ZSOIL calculation.
- ZS_History: contains the summary history of execution, also readable when opening the .his file directly.
- ZS_Model: contains the FE model data (nodes, elements, etc).
- ZS_Results_Config: contains the results configuration setup (nodal results and element results).
- ZS_Results: contains the results taken from several output files and stored as matrices or cells.

After calling ZS_R in the MATLAB environment, the user is able to read the output files of any ZSOIL simulation using the function ZS_read . This section focuses on the syntax required to read the files. Following this syntax, including lower or upper case, is paramount. In the following environments, the symbol • means that the corresponding field is mandatory, whereas the symbol ⊡ denotes an optional field. If an optional field is not provided by the user, the default value is used. If a mandatory field is missing, ZS_read will return an error. Compared to the standard ZSOIL Pre- and Post-processor, the user will notice that certain results cannot be accessed. Therefore, only the quantities explicitly given below can be currently read by ZS_read .

OUT = ZS_read (USER_OPTS, DISP_OPTS)

Input arguments	Type of object
USER_OPTS	See Env. 2.7.1
DISP_OPTS	String 'off' (default) 'on'

Output arguments	Type of object
OUT	ZS_Link class object

USER_OPTS contain the user options defining which results have to be parsed and DISP_OPTS is an option related to the messages and warnings in the MATLAB command window, switching this option to 'off' will speed up the process.

Environment 2.7.1 (USER_OPTS)

	Field name	Type of object	Description
•	.Job	See Env. 2.7.2	Identifier of the ZSOIL calculation
⊡	.HIS	ZS_History	Data of the history of execution (.his)
⊡	.RCF	ZS_Model	Data of the result configuration file (.rcf)
⊡	.DAT	ZS_Results_Config	FE model data (.dat)
⊡	.INP	String 'off' (default) 'on'	Options for reading the .inp file (to be switched 'on' if ZS_write is then used)
⊡	.Results	See Env. 2.7.3	Option for parsing the results

Note: ZS_History , ZS_Model , and ZS_Results_Config class objects are automatically returned by ZS_read . However, in the case of multiple parsings, the user can input them in the options. In this case, ZS_read will not read the .his, .rcf, and .dat files, resulting in time savings (see Section 3.4.1).

Environment 2.7.2 (Job)

	Field name	Type of object	Description
•	.Name	String	Name of the ZSOIL file to be read
⊡	.Path	String pwd (default)	Path to the folder containing the ZSOIL files (default : current MATLAB path)

Note: If the path is not given and the ZSOIL files are not contained under the current MATLAB path, ZS_read will return an error.

Environment 2.7.3 (Results)

	Field name	Type of object	Description
⊡	.Nodal	See Env. 2.7.4	Nodal results (.s00)
⊡	.Continuum	See Env. 2.7.5	Continuum results (.s01)
⊡	.Shell	See Env. 2.7.6	Shell results (.s02)
⊡	.Truss	See Env. 2.7.7	Truss results (.s03)
⊡	.Beam	See Env. 2.7.8	Beam results (.s04)
⊡	.Contact	See Env. 2.7.9	Contact results (.s07)
⊡	.Membrane	See Env. 2.7.10	Membrane results (.s15)
⊡	.Pile	See Env. 2.7.11	Beam and contact results (.s04+.s07)
⊡	.Nail	See Env. 2.7.11	Beam and contact results (.s04+.s07)
⊡	.Eigenmode	See Env. 2.7.12	Eigenmode results (.eig)
⊡	.Reaction	See Env. 2.7.13	Nodal reaction results (.r00)

Note: If none of the above result group is specified, `ZS_Results` class object will be empty.

Note on the array syntax

`ZS_read` can parse multiple output quantities in a group at once. Typically, parsing multiple quantities of interest (QoI) can be performed at once using the following syntax: `.QoI = ["QoI_1" "QoI_2" ... "QoI_n"]` as well as in the case of numerical array: `.ID = [ID_1 ID_2 ... ID_n]`. As logical indexing is used, parsing multiple outputs does not take longer.

Environment 2.7.4 (Nodal)

	Field name	Type of object	Description
⊡	.QoI	String array 'All' (default) "ABS" "U_i" "R_i" "PORE_PRESSURES" "TOTAL_HEADS" "TEMPERATURE" "HUMIDITY'	 Absolute displacement Displacement in the i-direction Rotation in the i-direction Pore pressures Total heads Temperature Humidity
⊡	.ID	Integer array 'All' (default)	Node number (ID)
⊡	See Env. 2.7.14		History of execution options

Environment 2.7.5 (Continuum)

	Field name	Type of object	Description
⊡	.QoI	String array 'All' (default) "SIG_ij" "EPS_ij" "V_i" "G_i" "STRESS_LEVEL" "SAT_RATIO" "UNDR_PRESSURE" "BIOT_COEFF" "EFF_SATURATION"	 Effective stress in the ij-direction Strain in the ij-direction Fluid velocity in the i-direction $k_r \nabla H$ in the i-direction Stress level Saturation ratio Undrained pressure Biot coefficient Effective saturation
⊡	.ID	Integer array 'All' (default)	Continuum element number (ID)
⊡	See Env. 2.7.14		History of execution options

Environment 2.7.6 (Shell)

	Field name	Type of object	Description
⊡	.QoI	String array 'All' (default) "F_ij" "M_ij" "Q_i" "NLAYER" "THICKNESS"	 Membrane force in the ij-direction Moment in the ij-direction Shear force in the i-direction Number of layers within the shell element Thickness of the shell element
⊡	.ID	Integer array 'All' (default)	Shell element number (ID)
⊡	See Env. 2.7.14		History of execution options

Environment 2.7.7 (Truss)

	Field name	Type of object	Description
⊡	.QoI	String array 'All' (default) "SIG_XX" "EPS_XX" "N_X" "STRESS_LEVEL"	 Axial stress Axial strain Axial force Stress level
⊡	.ID	Integer array 'All' (default)	Truss element number (ID)
⊡	See Env. 2.7.14		History of execution options

Environment 2.7.8 (Beam)

	Field name	Type of object	Description
⊡	.QoI	String array 'All' (default) "N_X" "M_i" "Q_i"	 Axial force Bending moment in i-direction Shear force in i-direction
⊡	.ID	Integer array 'All' (default)	Beam element number (ID)
⊡	See Env. 2.7.14		History of execution options

Environment 2.7.9 (Contact)

	Field name	Type of object	Description
⊡	.QoI	String array 'All' (default) "SIG_ij" "T_SIG_i" "EPS_i" "STRESS_LEVEL"	 Effective stress in the ij-direction Total stress in the i-direction (for non-normal stresses : T_SIG = SIG) Gap in the i-direction Stress level
⊡	.ID	Integer array 'All' (default)	Contact element number (ID)
⊡	See Env. 2.7.14		History of execution options

Environment 2.7.10 (Membrane)

	Field name	Type of object	Description
⊡	.QoI	String array 'All' (default) "SIG_ij" "F_ij" "STRESS_LEVEL"	 Stress in the ij-direction Membrane force in the ij-direction Stress level
⊡	.ID	Integer array 'All' (default)	Membrane element number (ID)
⊡	See Env. 2.7.14		History of execution options

Environment 2.7.11 (Pile or Nail)

	Field name	Type of object	Description
⊡	.QoI	See Envs. 2.7.8 and 2.7.9	As piles and nails are interpreted as beam elements, both beam and contact results are available and are stored by pile/nail number.
⊡	.ID	Integer array 'All' (default)	Pile or nail number (ID)
⊡	See Env. 2.7.14		History of execution options

Environment 2.7.12 (Eigemodes)

	Field name	Type of object	Description
⊡	.QoI	String array 'All' (default) "ANG_FREQUENCY" "FREQUENCY" "EFF_MASS" "PART_FACTOR" "EIG_VECTOR"	 Angular frequency Frequency Modal effective mass Mode participation factor Eigenvector

Environment 2.7.13 (Reaction)

	Field name	Type of object	Description
⊡	.QoI	String array 'All' (default) "ABS" "R_i" "M_i" "FLUID_FLUX" "HEAT_FLUX" "HUMIDITY_FLUX"	 Translational absolute reaction Translational reaction in the i-direction Rotational reaction in the i-direction Fluid flux reaction Temperature reaction Humidity reaction
⊡	.ID	Integer array 'All' (default)	Node number (ID)
⊡	See Env. 2.7.14		History of execution options

Environment 2.7.14 (History of execution)

	Field name	Type of object	Description
⊡	.TimeStep	Double array 'All' (default)	Time steps number
⊡	.Driver	String array 'All' (default) "INITIAL_STATE" "STABILITY" "TIME_DEPENDENT" "ARC_LENGTH" "DYNAMICS" "PUSHOVER" "EIGENMODES"	Driver name
⊡	.DivergedSteps	Char 'delete' (default) 'keep'	Strategy for diverged steps Will be removed Will be kept

Note on the returned data

ZS_read will return 2D matrices [ ] in the case of nodal results or 2D cells { } otherwise, but in both cases, they have a size of N-by-M. The N rows contain the results according to the IDs given by the user, whereas the M columns index the corresponding time steps.

Consider the following example:

`.QoI = [ID_1 ID_2 ID_3]` and `.TimeSteps = [TS_1 : TS_10]`

The size of the returned cell or matrix will be `[3 10]`.

Element results are returned as a cell array due to the number of Gauss points depending on the element class. Thus, for one element and one time step, multiple results can be contained in a single cell, as shown in the following example:

Consider parsing beam results (let's say one QoI at time step TS $= 1$) from a model containing standard beam elements (displacement-based - BEL2 element with 1 GP) and one highly accurate beam element (flexibility-based - SBEL2 element with 5 GP). This will result in the following cell array:

$$\begin{array}{c} \\ \mathrm{BEL2}_{\mathrm{ID\_1}} \\ \mathrm{BEL2}_{\mathrm{ID\_2}} \\ \mathrm{SBEL2}_{\mathrm{ID\_3}} \\ \mathrm{BEL2}_{\mathrm{ID\_4}} \end{array} \left\{ \begin{array}{c} \mathrm{TS}=1 \\ \cdots \\ \cdots \\ \left[\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots\right] \\ \cdots \end{array} \right\}$$

This illustration holds to other element classes that contain multiple Gauss points.

The user can refer to Section 3.4 for a complete example of the use of ZS_read .

2.8 ZS_removeConstant

ZS_removeConstant is used to reduce an input matrix by eliminating constant variables. If the size of the initial input matrix X is $(N \times M)$, which includes N samples of M random variables and K constant variables, the size of the resulting reduced matrix OUT will be $(N \times M - K)$.

```
OUT = ZS_removeConstant (X)
```

Input arguments	Type of object
X	Double array

Output arguments	Type of object
OUT	Double array

2.9 ZS_removeDiverged

ZS_removeDiverged is used to reduce an input matrix and an output matrix by eliminating the diverged or unreturned calculations.

```
[NEW_X , NEW_Y] = ZS_removeConstant (X , Y)
```

Input arguments	Type of object
X	Double array
Y	Cell or double array

Output arguments	Type of object
NEW_X	Double array
NEW_Y	Cell or double array

2.10 ZS_R

The ZS_R function will launch the ZS+R module. It will simply add the \Interface folder to the current MATLAB path (see above), enabling the user to use ZS+R functions. Therefore, starting ZS+R is necessary to use ZS+R functions.

ZS_R does not take any arguments and does not return any.

```
ZS_R ()
```

2.11 ZS_R_install

The ZS_R_install function will install ZS+R. In brief, ZS_R_install will perform the following tasks:

I. It will add the ZS+R\core path to the MATLAB path, excluding the \Interface folder.

II. It will search the current machine's hard drives to find Z_Soil.exe files, returning the corresponding ZSoil versions if found.

III. It will search the current machine's hard drives to find uqlab.m files, returning the corresponding UQLab versions if found.

IV. It will launch the ZS+R module.

ZS_R_install does not take any arguments and does not return any.

```
ZS_R_install ()
```

2.12 ZS_R_uninstall

The ZS_R_uninstall function will uninstall ZS+R.

ZS_R_uninstall does not take any arguments and does not return any.

```
ZS_R_uninstall ()
```

Chapter 3

Tutorials

This chapter aims to provide some syntax examples for using ZS+R and some useful MATLAB commands. This is achieved by focusing on a specific example involving the numerical simulation of deep excavation in Berlin Sand. The corresponding file is HS-Brick-Exc-Berlin-Sand-2phase, available under the path my_folder/Tutorial. This example has been comprehensively covered in the following references: Benz (2007), Geotechnics & Schweiger (2004), Obrzud & Truty (2010).

3.1 Installing ZS+R

1) Unzip the zip archive in a folder, let's say my_folder.
2) Open MATLAB in administrator mode and choose my_folder/core as the working directory.
3) Enter `ZS_R_install` in the MATLAB console and press ENTER.

3.2 Starting ZS+R

Reference file : Start.m

The following syntax will start ZS+R :

```
ZS_R % <- This will start ZS+R
```

3.3 Parsing FE data with ZS_read

Reference file : Parsing_FE_Data.m

In order to parse the whole FE model data, consider the following syntax :

```
clear all
clc
ZS_R

% Define the file location
current_script_path = erase(mfilename("fullpath"),mfilename);
ZSoil_File_path = [current_script_path,'\ZSOIL'];
ZSoil_File_Name = 'HS-Brick-Exc-Berlin-Sand-2phase';

% Create the user options

% Job field
OPTS.Job.Path = ZSoil_File_path;
OPTS.Job.Name = ZSoil_File_Name;

% Get FE data
OUT = ZS_read(OPTS,'on');
FE_DATA = OUT.MODEL_dat;
clear OUT
```

Note: The command `erase(mfilename("fullpath"),mfilename)` will provide the location of the file currently being executed in MATLAB (in this example, Parsing_FE_Data.m).

Once the FE data is read, the user can access various model properties. Some examples are provided below.

Extracting node coordinates :

```
% Extract node coordinates
Node_XYZ = vertcat(FE_DATA.NODES.XYZ);
```

Extracting all master nodes IDs of the standard beam elements (BEL2) :

```
% Extract standard beam (BEL2) node IDs
temp = [FE_DATA.ELEMENTS.BEAMS.BEL2.NODES];
BEL2_Node_IDs = vertcat(temp.MASTER);
clear temp
```

Plot all load time functions :

```
% Plot all load time function
LF = FE_DATA.LOAD_TIME_FUNCTIONS;
N_LF = length(LF);
for i = 1:N_LF
    temp_ID = LF(i).ID;
    temp_data = LF(i).LTF;
    figure(i)
    plot(temp_data(:,1),temp_data(:,2))
    title(['Load function number ',num2str(temp_ID)])
    xlabel('Time (s)')
    ylabel('Value')
end
clear temp_data temp_ID i N_LF
```

3.4 Parsing results with ZS_read

3.4.1 Parsing nodal results

The standard procedure for parsing nodal results follows:

1) Start ZS+R (refer to 3.2).
2) Specify the .Job field (refer to Env. 2.7.2).
3) Specify any history of execution property (refer to **??**).
4) Specify any nodal result property (refer to **??**).

Case 1 - Reference file: Parsing_nodal_results_1.m

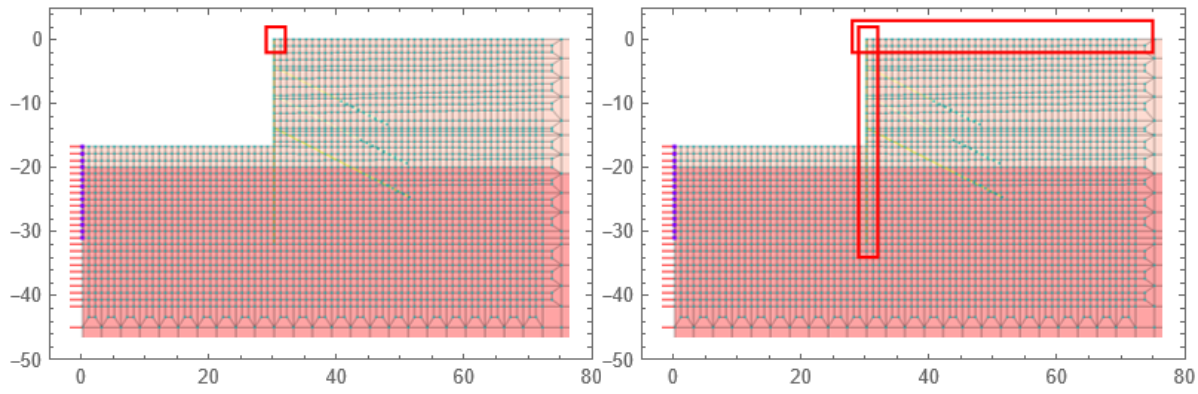
Suppose you want to parse only the horizontal and vertical displacement of the top node of the retaining wall (as shown in Fig. 3.1a) at time step T=9, excluding the stability steps. First, go to the ZSOIL Post-processor to select the corresponding node, and then follow this syntax:

```
% Parse one node at the top of the wall
OPTS.Results.Nodal.QoI = ["U_X","U_Y"];
OPTS.Results.Nodal.ID = 6;
OPTS.Results.Nodal.TimeStep = 9;
OPTS.Results.Nodal.Driver = "TIME_DEPENDENT";

% Get nodal results
OUT = ZS_read(OPTS,'on');
U_X_6 = OUT.RESULTS.NODAL_s00.DISPLACEMENTS.U_X; %(Matrix output)
U_Y_6 = OUT.RESULTS.NODAL_s00.DISPLACEMENTS.U_Y; %(Matrix output)
```

Note that the above syntax returns U_x and U_y as matrices. When multiple QoIs are involved, storing the variables as a structure array is more convenient, this logic will be used in this document :

```
% Store the results as a structure
RES.UX_6 = OUT.RESULTS.NODAL_s00.DISPLACEMENTS.U_X;
RES.UY_6 = OUT.RESULTS.NODAL_s00.DISPLACEMENTS.U_Y;
```



(a) Case 1 (b) Case 2

Figure 3.1: Selection of some quantities of interest of the HS-Brick-Exc-Berlin-Sand-2phase model.

Case 2 - Reference file : Parsing_nodal_results_2.m

Now, let's consider parsing the maximum absolute horizontal displacement of the retaining wall and the maximum absolute vertical displacement of the ground surface. To obtain two distinct result groups, ZS_read must be called multiple times. In this scenario, since the class objects ZS_History , ZS_Results_Config , and ZS_Model will not change, they can be provided as user options to avoid multiple readings of the .his, .rcf, and .dat files, thus saving time. Here's how it can be done:

```
% Pass HIS, RCF and DAT in the user options
OUT = ZS_read(OPTS,'on');
OPTS.HIS = OUT.HISTORY_his;
OPTS.RCF = OUT.RESULTS_CONFIG_rcf;
OPTS.DAT = OUT.MODEL_dat;
```

When parsing multiple nodes, you should define the .ID field using a 1-by-K array, and it must be sorted in ascending order. Sorting in ascending order is mandatory because the results are returned in ascending order based on the node/element ID. To parse the horizontal displacement of the wall for multiple nodes, follow this procedure:

```
% Parse the maximal horizontal displacement of the wall
OPTS.Results.Nodal.QoI = ["U_X"];
OPTS.Results.Nodal.ID = [3 6 14 982 1015 1048 1081 1114 1147 1180 1213 1246 1279 1312 1345 ...
                         1378 1411 1444 1477 1510 1543 1576 1609 1804 3278 3281 3284 3436 ...
                         3439 3442 3505 3508 3511];
OPTS.Results.Nodal.TimeStep = 9;
OPTS.Results.Nodal.Driver = "TIME_DEPENDENT";

OUT = ZS_read(OPTS);
RES.MAX_UX = max( abs(OUT.RESULTS.NODAL_s00.DISPLACEMENTS.U_X) );
```

Similarly, the maximal vertical displacement is parsed by recalling ZS_read . In this case, the user options must be redefined as follows :

```
% Parse the maximal vertical displacement of the surface ground
OPTS.Results.Nodal.QoI = ["U_Y"];
OPTS.Results.Nodal.ID = [8 11 2581 3604 3605 3606 3607 3608 3609 3610 3611 3612 3613 3614 ...
                         3615 3616 3617 3618 3619 3620 3621 3622 3623 3624 3625 3626 3627 ...
                         3628 3629 3630 3631 3632 3633 3634 3635 3636 3637 3638 3639 3640 ...
                         3641 3642 3643 3644];
OPTS.Results.Nodal.TimeStep = 9;
OPTS.Results.Nodal.Driver = "TIME_DEPENDENT";

OUT = ZS_read(OPTS);
RES.MAX_UY = max( abs(OUT.RESULTS.NODAL_s00.DISPLACEMENTS.U_Y) );
```

3.4.2 Parsing element results

The procedure to parse element results is exactly similar to parsing nodal results.

Case 1 - Reference file : Parsing_beam_results.m

Consider parsing the maximal and the minimal bending moment and shear force values within the beam elements constituting the retaining wall. This can be achieved using :

```
% Parse the min max value of bending moments and shear forces
OPTS.Results.Beam.ID = [3944 3945 3946 3947 3948 3949 3950 3951 3952 3953 3954 ...
                        3955 3956 3957 3958 3959 3960 3961 3962 3963 3964 3965 ...
                        3966 3967 3968 3969 3970 3971 3972 3973 3974 3975];
OPTS.Results.Beam.QoI = ["M_Z","Q_Y"];
OPTS.Results.Beam.TimeStep = 9;
OPTS.Results.Beam.Driver = "TIME_DEPENDENT";

OUT = ZS_read(OPTS,'on');

RES.Moment_Max = cell2mat( minmax(OUT.RESULTS.BEAM_s04.MOMENTS.M_Z') );
RES.Shear_Max = cell2mat( minmax(OUT.RESULTS.BEAM_s04.FORCES.Q_Y') );
```

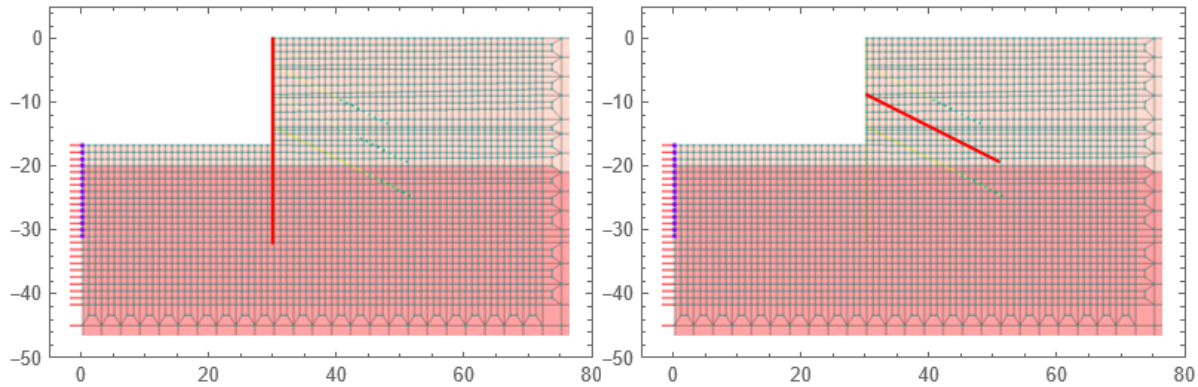
Case 2 - Reference file : Parsing_truss_results.m

As well as for the maximal axial force of the second anchor of the wall :

```
% Parse the min max value of bending moments and shear forces
OPTS.Results.Truss.ID = [3985 3986 3987 3988 3989 3990 3991 3992 3993];
OPTS.Results.Truss.QoI = "N_X";
OPTS.Results.Truss.TimeStep = 9;
OPTS.Results.Truss.Driver = "TIME_DEPENDENT";

OUT = ZS_read(OPTS,'on');

RES = max( cell2mat(OUT.RESULTS.TRUSS_s03.N_X) );
```



(a) Case 1 (b) Case 2

Figure 3.2: Selection of some quantities of interest of the HS-Brick-Exc-Berlin-Sand-2phase model.

3.5 Uncertainty quantification analysis

Following the general schemes proposed in Li et al. (2021), Minini et al. (2023), Rocquigny et al. (2008), Sudret (2007), this section aims to guide the user step by step through a geotechnical-oriented uncertainty quantification analysis using ZSoil and UQLab. The overall procedure is outlined in Fig. 3.3.

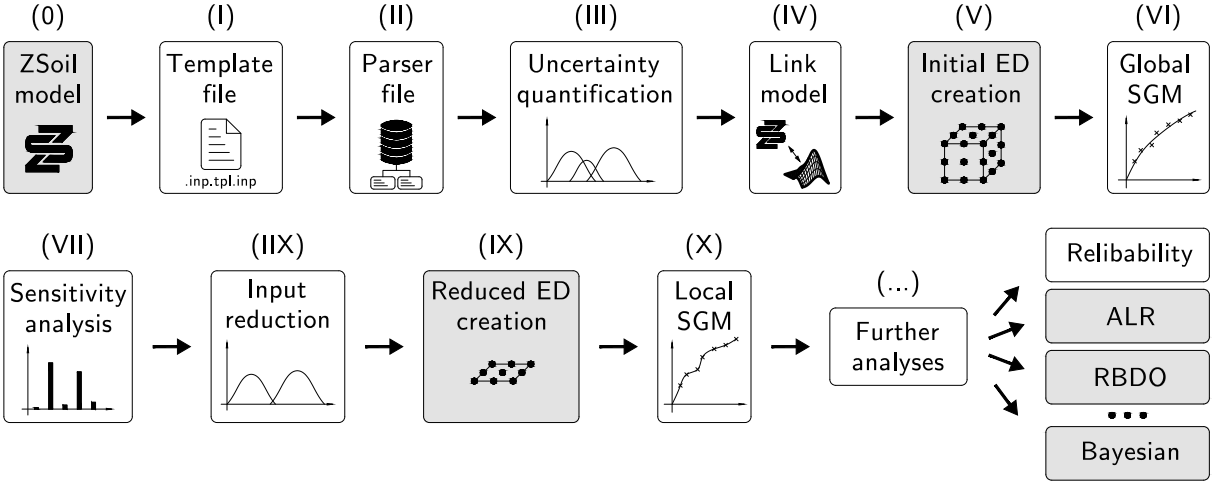


Figure 3.3: UQ analysis procedure scheme. Grey coloured boxes indicate a time consuming process.

3.5.1 (0) ZSoil model and UQ analysis

Reference file : HS-Brick-Exc-Berlin-Sand-2phase.inp

The Section 3.5 focuses on an UQ analysis on the following output quantities or quantity of interest : 1) the maximal absolute horizontal deflection of the retaining wall; 2) the maximal absolute settlement of the ground surface; 3) the maximal absolute bending moment of the wall; 4) the last converged safety factor.

3.5.2 (I) Making of the template file

Reference file : UQ \HS-Brick-Exc-Berlin-Sand-2phase_.inp.tpl.inp

The initial step in creating the template is to duplicate the original ZSoil input file (.inp). To prevent UQLab from making any unintended modifications to the original input file, the copy of the original file should be renamed with an additional extension '.tpl'. For example, if the original file is 'myZSoilModel.inp,' the template file should be renamed 'myZSoilModel.inp.tpl.inp.'

The fundamental concept behind the template file is to enable MATLAB to modify any third-party software input file. In this context, the template file must include the locations where variables need to be replaced. This can be accomplished in the following ways:

3.5.2.1 Manually

I. Open 'myZSoilModel.inp.tpl.inp' with a notepad editor software (for instance : Don (2023)).

II. Within the template file, find the location of the variables that must be replaced by MATLAB.

III. Edit the corresponding variables with the following format : '<X0001>' for variable 1, '<X0002>' for variable 2, ..., '<Xn>' for variable n.

3.5.2.2 Using ZS_write

3.5.3 (II) Making of the parser file

Reference file : UQ \Parser.m

The parser file is a program that accepts inputs and returns outputs. In MATLAB, this is achieved using a function written in a .m file. The goal of the parser is to collect results from ZSoil. Within the parser file, ZS_read can be called to read the ZSoil output files and return the results in a user-defined format (numerical array, structure array, etc.). For this example, let's call the parser file 'Parser.m'. In MATLAB, it is mandatory for the main function to have the same name as its corresponding file. Thus, this parser file begins with:

```
function RES = Parser(OutputFile)
```

When random variables are involved, the behaviour of the model can become unpredictable. If the requested quantity of interest does not exist because the model did not converge, ZS_read will generate an error, and the procedure will stop. This is why it is highly advisable to instruct the parser to return an empty array in all cases. This can be achieved by calling ZS_createParser or manually in the following way:

```
function RES = Parser(OutputFile)

try

% Do something

catch

RES = [];

end
```

As described in Section 3.4, parsing the quantities of interest is achieved by using ZS_read . Since ZS_read will be called multiple times within the execution of the parser function, it's recommendable to pass the class objects ZS_History , ZS_Results_Config , and ZS_Model as options. This helps to avoid unnecessary computational burdens and to save time (refer to Section Env. 2.7.1 and 3.4.1):

```
try

[file_path,file_name] = fileparts(regexprep(OutputFile,'\.[^.]*$',''));

OPTS.Job.Name = file_name;
OPTS.Job.Path = file_path;

% Store first the data, the his and the rcf solutions
OUT = ZS_read(OPTS);

DAT = OUT.MODEL_dat;
RCF = OUT.RESULTS_CONFIG_rcf;
HIS = OUT.HISTORY_his;

OPTS.DAT = DAT;
OPTS.RCF = RCF;
OPTS.HIS = HIS;
```

Maximal absolute horizontal deflection :

```
% Parse Ux_max
OPTS.Results.Nodal.QoI = "U_X";
OPTS.Results.Nodal.TimeStep = 9;
OPTS.Results.Nodal.Driver = "TIME_DEPENDENT";
OPTS.Results.Nodal.ID = [3 6 14 982 1015 1048 1081 1114 1147 1180 1213 1246 1279 1312 1345 ...
                         1378 1411 1444 1477 1510 1543 1576 1609 1804 3278 3281 3284 3436 ...
                         3439 3442 3505 3508 3511];
OUT = ZS_read(OPTS);
RES.Ux_max =  max(abs(OUT.RESULTS.NODAL_s00.DISPLACEMENTS.U_X));
clear OUT
```

Maximal absolute vertical settlement :

```
% Parse Uy_max
OPTS.Results.Nodal.QoI = "U_Y";
OPTS.Results.Nodal.TimeStep = 9;
OPTS.Results.Nodal.Driver = "TIME_DEPENDENT";
OPTS.Results.Nodal.ID = [8 11 2581 3604 3605 3606 3607 3608 3609 3610 3611 3612 3613 3614 ...
                         3615 3616 3617 3618 3619 3620 3621 3622 3623 3624 3625 3626 3627 ...
                         3628 3629 3630 3631 3632 3633 3634 3635 3636 3637 3638 3639 3640 ...
                         3641 3642 3643 3644];

OUT = ZS_read(OPTS);
RES.Uy_max =  max(abs(OUT.RESULTS.NODAL_s00.DISPLACEMENTS.U_Y));
clear OUT
```

Maximal absolute bending moment :

```
% Parse M_max
OPTS.Results.Beam.ID = [3944 3945 3946 3947 3948 3949 3950 3951 3952 3953 3954 ...
                        3955 3956 3957 3958 3959 3960 3961 3962 3963 3964 3965 ...
                        3966 3967 3968 3969 3970 3971 3972 3973 3974 3975];
OPTS.Results.Beam.QoI = "M_Z";
OPTS.Results.Beam.TimeStep = 9;
OPTS.Results.Beam.Driver = "TIME_DEPENDENT";

OUT = ZS_read(OPTS);
M_max = OUT.RESULTS.BEAM_s04.MOMENTS.M_Z;
M_max = cell2mat(M_max);
RES.M_max = max(abs(M_max));
clear OUT
```

Last converged safety factor :

```
% Parse the last safety factor
OPTS.Results.Stability.TimeStep = 9;
OPTS.Results.Stability.SF = 'last';

OUT = ZS_read(OPTS);
RES.SF = OUT.RESULTS.STABILITY_his.SF;
clear OUT
```

3.5.4 (III) Uncertainty quantification

Reference file : Create_Input.m

"Let's consider the friction angle, cohesion, Young's modulus for both soil layers, and also the Young's modulus of the retaining wall as random variables. The corresponding UQLab syntax reads:

```
current_script_path = erase(mfilename("fullpath"),mfilename);
save_path = strcat(current_script_path,"\UQ\","All_Inputs.mat");

% Create initial input ZSoil + full metamodels
InputOpts.Marginals(1).Name='Phi_S1';
InputOpts.Marginals(1).Type='Lognormal';
InputOpts.Marginals(1).Moments=[35,0.1*35];

InputOpts.Marginals(2).Name='C_S1';
InputOpts.Marginals(2).Type='Lognormal';
InputOpts.Marginals(2).Moments=[1,0.2*1];

InputOpts.Marginals(3).Name='E_S1';
InputOpts.Marginals(3).Type='Lognormal';
InputOpts.Marginals(3).Moments=[45,0.3*45];

InputOpts.Marginals(4).Name='Phi_S2';
InputOpts.Marginals(4).Type='Lognormal';
InputOpts.Marginals(4).Moments=[38,0.1*38];

InputOpts.Marginals(5).Name='C_S2';
InputOpts.Marginals(5).Type='Lognormal';
InputOpts.Marginals(5).Moments=[1,0.2*1];

InputOpts.Marginals(6).Name='E_S2';
InputOpts.Marginals(6).Type='Lognormal';
InputOpts.Marginals(6).Moments=[75,0.3*75];

InputOpts.Marginals(7).Name='E_Wall';
InputOpts.Marginals(7).Type='Lognormal';
InputOpts.Marginals(7).Moments=[30000,0.1*30000];

InputOpts.Copula.Type = 'Independent';

myInput = uq_createInput(InputOpts);
All_Inputs.ZSoil.Initial = myInput; % For Link model
All_Inputs.PCE = myInput;           % For PCE model
save(save_path,"All_Inputs")
clear InputOpts myInput
```

Please note that this input is available for both the ZSoil link model and the global surrogate model (PCE).

3.5.5 (IV) Link ZSoil with UQLab

Reference file : Create_Model.m

To create a link model, the `ZS_createModel` function can be used :

```
current_script_path = erase(mfilename("fullpath"),mfilename);
myPath = [current_script_path,'UQ'];
```

```
% Create a ZSoil link model
ModelOpts.Name='HS-Brick-Exc-Berlin-Sand-2phase';
ModelOpts.Parser.Name = 'Parser';
ModelOpts.Parser.Path = myPath;
ModelOpts.Template.Path = myPath;
ModelOpts.ExecutionPath = myPath;

All_Models.ZSoil = ZS_createModel(ModelOpts);

file_save = strcat(current_script_path,"\UQ\","All_Models.mat");
```

3.5.6 (V) Initial experimental design sampling

Reference file : Create_Sampling_Initial.m

To draw the initial sample points, the function ZS_parallel_evalModel can be used to generate model evaluations in parallel using four threads. It's important to note that the output will be automatically saved in a .mat file. In this case, the 'latin hypercube' ('lhs') sampling method is employed. For more information, please refer to ZS_help input.

```
current_script_path = erase(mfilename("fullpath"),mfilename);

input_file = strcat(current_script_path,"\UQ\All_Inputs.mat");
load(input_file); clear input_file

model_file = strcat(current_script_path,"\UQ\All_Models.mat");
load(model_file); clear model_file

% Draw some samples
myInput = All_Inputs.ZSoil.Initial;
X = uq_getSample(myInput,150,'lhs');

% Define the model and evaluate it
myModel = All_Models.ZSoil;
Y = ZS_parallel_evalModel(myModel,X,4);
```

3.5.7 (VI) Global surrogate model creation

Reference file : Create_Model.m

The global surrogate model is created using a polynomial chaos expansion (PCE) approximation. Below is a possible syntax, but you can refer to the PCE user manual by typing ZS_help pce in the MATLAB console to explore more options for PCE model creation. As there are four quantities of interest involved, the PCE creation is carried out using a for loop.

```
% Create multiple PCE
input_file = strcat(current_script_path,"\UQ\All_Inputs.mat");
load(input_file); clear input_file

ED_file = strcat( ...
    current_script_path,"\UQ\Backup_HS-Brick-Exc-Berlin-Sand-2phase_Initial_Sampling.mat");
load(ED_file); clear ED_file

QoI_Names = string(fieldnames(Y{1}));
```

```
converged =  ¬cellfun(@isempty,Y); % Select the converged results

for i = 1:length(QoI_Names)
    OPTS.Type = 'Metamodel';
    OPTS.MetaType = 'PCE';
    OPTS.Input = All_Inputs.PCE;
    OPTS.Degree = [1:10];
    OPTS.ExpDesign.X = X(converged,:);
    temp_Y = cell2mat(Y(converged));
    temp_Y = [temp_Y.(QoI_Names(i))]';
    OPTS.ExpDesign.Y = temp_Y;

    PCE_Model = uq_createModel(OPTS);
    All_Models.PCE.(QoI_Names(i)) = PCE_Model;
    clear OPTS PCE_Model temp_Y
end

save(file_save,"All_Models")
```

3.5.8 (VII) Sensitivity analysis

Reference file : Create_Sensitivity.m

As the random vector is independent in this case, sensitivity indices can be computed using a Sobol' sensitivity analysis. Below is one possible syntax to calculate the total and first Sobol' indices. You can also consult the UQLab manual by typing `ZS_help sensitivity` for more information. Once again, since there are four quantities of interest, a for loop can be used.

```
current_script_path = erase(mfilename("fullpath"),mfilename);

input_file = strcat(current_script_path,"\UQ\All_Inputs.mat");
load(input_file); clear input_file

model_file = strcat(current_script_path,"\UQ\All_Models.mat");
load(model_file); clear input_file

PCE_model_names = string(fieldnames(All_Models.PCE));
Total_indices = [];

for i = 1:length(PCE_model_names)
    OPTS.Name = PCE_model_names(i);
    OPTS.Type = 'Sensitivity';
    OPTS.Method = 'Sobol';
    OPTS.Input = All_Inputs.PCE;
    OPTS.Model = All_Models.PCE.(PCE_model_names(i));
    OPTS.Sobol.PCEBased = true;
    Sobol = uq_createAnalysis(OPTS);
    All_Sensitivity.(PCE_model_names(i)) = Sobol;
    Total_indices = [Total_indices,Sobol.Results.Total];
end

var = categorical({All_Inputs.ZSoil.Initial.Marginals.Name});
var = reordercats(var,{All_Inputs.ZSoil.Initial.Marginals.Name});
bar(var,Total_indices)
legend(PCE_model_names)
```

3.5.9 (IIX) Reducing the input

Reference file : Create_Input.m

Reducing the input is simply done by creating a new one with constant random variables as follows :

```
% Create reduced input ZSoil
InputOpts.Marginals(1).Name='Phi_S1';
InputOpts.Marginals(1).Type='Lognormal';
InputOpts.Marginals(1).Moments=[35,0.1*35];

InputOpts.Marginals(2).Name='C_S1';
InputOpts.Marginals(2).Type='Constant';
InputOpts.Marginals(2).Moments=1;

InputOpts.Marginals(3).Name='E_S1';
InputOpts.Marginals(3).Type='Lognormal';
InputOpts.Marginals(3).Moments=[45,0.3*45];

InputOpts.Marginals(4).Name='Phi_S2';
InputOpts.Marginals(4).Type='Lognormal';
InputOpts.Marginals(4).Moments=[38,0.1*38];

InputOpts.Marginals(5).Name='C_S2';
InputOpts.Marginals(5).Type='Constant';
InputOpts.Marginals(5).Moments=1;

InputOpts.Marginals(6).Name='E_S2';
InputOpts.Marginals(6).Type='Lognormal';
InputOpts.Marginals(6).Moments=[75,0.3*75];

InputOpts.Marginals(7).Name='E_Wall';
InputOpts.Marginals(7).Type='Constant';
InputOpts.Marginals(7).Moments=30000;

InputOpts.Copula.Type = 'Independent';

myInput = uq_createInput(InputOpts);
All_Inputs.ZSoil.Reduced = myInput;
save(save_path,"All_Inputs")
clear InputOpts myInput
```

When working with surrogate models, it is strongly recommended to remove the constant variable of the reduced input, the corresponding syntax reads

```
% Create reduced input metamodel
InputOpts.Marginals(1).Name='Phi_S1';
InputOpts.Marginals(1).Type='Lognormal';
InputOpts.Marginals(1).Moments=[35,0.1*35];

InputOpts.Marginals(2).Name='E_S1';
InputOpts.Marginals(2).Type='Lognormal';
InputOpts.Marginals(2).Moments=[45,0.3*45];

InputOpts.Marginals(3).Name='Phi_S2';
InputOpts.Marginals(3).Type='Lognormal';
InputOpts.Marginals(3).Moments=[38,0.1*38];

InputOpts.Marginals(4).Name='E_S2';
InputOpts.Marginals(4).Type='Lognormal';
```

```
InputOpts.Marginals(4).Moments=[75,0.3*75];

InputOpts.Copula.Type = 'Independent';

myInput = uq_createInput(InputOpts);
All_Inputs.PCK = myInput;
save(save_path,"All_Inputs")
clear InputOpts myInput
```

3.5.10 (IX) Reduced experimental design sampling

Reference file : Create_Sampling_Reduced.m

Generating the samples for the reduced input is identical to the point (V) :

```
current_script_path = erase(mfilename("fullpath"),mfilename);

input_file = strcat(current_script_path,"\UQ\All_Inputs.mat");
load(input_file); clear input_file

model_file = strcat(current_script_path,"\UQ\All_Models.mat");
load(model_file); clear model_file

% Draw some samples
myInput = All_Inputs.ZSoil.Reduced;
X = uq_getSample(myInput,100,'lhs');

% Define the model and evaluate it
myModel = All_Models.ZSoil;
Y = ZS_parallel_evalModel(myModel,X,4);
```

3.5.11 (X) Local surrogate model creation

Reference file : Create_Model.m

The local surrogate model is built using polynomial chaos Kriging (PCK), which combines a standard Kriging model with a PCE. For detailed information, please consult the corresponding UQLab user manual by typing `ZS_help pck`.

As explained in point (IIX), it is highly recommended to remove constant variables when working with surrogate models. Although the reduced input `All_Inputs.PCK` contains only 3 random variables, the X-sample points used for generating the reduced experimental design still contain 7 random variables, with 4 of them being constant. To remove these remaining 4 variables, the function `ZS_removeConstant` can be used to eliminate the constant variables from the input matrix.

```
% Create multiple PCK
input_file = strcat(current_script_path,"\UQ\All_Inputs.mat");
load(input_file); clear input_file

ED_file = strcat( ...
    current_script_path,"\UQ\Backup_HS-Brick-Exc-Berlin-Sand-2phase_Reduced_Sampling.mat");
load(ED_file); clear ED_file

X = ZS_removeConstant(X);
```

```
for i = 1:length(QoI_Names)
    OPTS.Type = 'Metamodel';
    OPTS.MetaType = 'PCK';
    OPTS.Input = All_Inputs.PCK;
    OPTS.Degree = [1:10];
    OPTS.ExpDesign.X = X;
    temp_Y = cell2mat(Y);
    temp_Y = [temp_Y.(QoI_Names(i))]';
    OPTS.ExpDesign.Y = temp_Y;

    PCK_Model = uq_createModel(OPTS);
    All_Models.PCK.(QoI_Names(i)) = PCK_Model;
    clear OPTS PCK_Model temp_Y
end

save(file_save,"All_Models")
```

3.5.12 (XI) Reliability analysis

Reference file : Create_Reliability.m

For the final step, a crude Monte-Carlo-based reliability analysis is suggested for the safety factor. It's worth noting that additional analyses are now specific to the case at hand. For various other uncertainty analyses, please refer to the other UQLab user manuals.

```
current_script_path = erase(mfilename("fullpath"),mfilename);
myPath = [current_script_path,'UQ'];

load(fullfile(myPath,"All_Inputs.mat"))
load(fullfile(myPath,"All_Models.mat"))

OPTS.Type = 'uq_reliability';
OPTS.Method = 'MCS';
OPTS.Input = All_Inputs.PCK;
OPTS.Model = All_Models.PCK.SF;
OPTS.LimitState.Threshold = 1;
OPTS.LimitState.CompOp = '<';
OPTS.Simulation.MaxSampleSize = 10^6; % For computational simplicity

MCS = uq_createAnalysis(OPTS);
```

Appendix A

ZS_Link_class

Bibliography

Benz, T. (2007), *Small-strain stiffness of soils and its numerical consequences*, number 55 *in* 'Mitteilung / Institut für Geotechnik Stuttgart', Inst. für Geotechnik, Stuttgart.

Don, H. (2023), 'Notepad++ - a free source code editor and Notepad replacement that supports several languages.'.
URL: *https://notepad-plus-plus.org/*

Geotechnics, B. I. & Schweiger, H. F. (2004), PART I: RESULTS OF BENCHMARKING PART II: REFERENCE SOLUTION AND PARAMETRIC STUDY.

Lataniotis, C., Marelli, S. & Sudret, B. (2022), UQLab user manual – The Model module, Technical report, Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland.

Li, Z., Gong, W., Li, T., Juang, C. H., Chen, J. & Wang, L. (2021), 'Probabilistic back analysis for improved reliability of geotechnical predictions considering parameters uncertainty, model bias, and observation error', *Tunnelling and Underground Space Technology* **115**, 104051.
URL: *https://linkinghub.elsevier.com/retrieve/pii/S088677982100242X*

Minini, J., Zhang, Y., Groslambert, M. & Commend, S. (2023), 'Finite element-based probabilistic framework including Bayesian inference for predicting displacements due to tunnel excavation', *Computers and Geotechnics* **162**, 105604.
URL: *https://linkinghub.elsevier.com/retrieve/pii/S0266352X23003610*

Moustapha, M., Marelli, S. & Sudret, B. (2022), UQLab user manual – The UQLink module, Technical report, Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland.

Obrzud, R. & Truty, A. (2010), The Hardening Soil Model: A Practical Guidebook, Technical Report Z_Soil.PC 100701, GeoDev SàRL.

Rocquigny, E. d., Devictor, N. & Tarantola, S., eds (2008), *Uncertainty in industrial practice: a guide to quantitative uncertainty management*, J. Wiley, Chichester, England ; Hoboken, NJ.

Sudret, B. (2007), 'Uncertainty propagation and sensitivity analysis in mechanical models–Contributions to structural reliability and stochastic spectral methods', *Habilitation a diriger des recherches, Université Blaise Pascal, Clermont-Ferrand, France* **147**, 53.