1 USER-DEFINED SOIL MODELS

1.1 INTRODUCTION

PLAXIS has a facility for user-defined (UD) soil models. This facility allows users to implement a wide range of constitutive soil models (stress-strain-time relationship) in PLAXIS. Such models must be programmed in FORTRAN (or another programming language), then compiled as a Dynamic Link Library (DLL) and then added to the PLAXIS program directory.

In principle the user provides information about the current stresses and state variables and PLAXIS provides information about the previous ones and also the strain and time increments. In the material data base of the PLAXIS input program, the required model parameters can be entered in the material data sets.

$\sigma_{ij}^{t+\Delta t},\ \kappa^{t+\Delta t}$	current stresses and state variables
$\sigma_{ij}^{t},\ \kappa^{t}$	previous stresses and state variables
$\Delta\varepsilon_{ij},\ \Delta t$	strain and time increments

As an example, a UD subroutine based on the Drucker-Prager material model is provided in the user-defined soil models directory, which is included in the program installation package. In this section, a step-by-step description on how a user-defined soil model can be formed and utilised in PLAXIS is presented.

Hint: Please note that the PLAXIS organization cannot be held responsible for any malfunctioning or wrong results due to the implementation and/or use of user-defined soil models.

1.2 IMPLEMENTATION OF UD MODELS IN CALCULATIONS PROGRAM

The PLAXIS calculations program has been designed to allow for User-defined soil models. There are mainly four tasks (functionalities) to be performed in the calculations program:

- Initialisation of state variables
- Calculation of constitutive stresses (stresses computed from the material model at certain step)
- Creation of effective material stiffness matrix
- Creation of elastic material stiffness matrix

These main tasks (and other tasks) have to be defined by the user in a subroutine called 'User_Mod'. In this subroutine more than one user-defined soil model can be defined. If a UD soil model is used in an application, the calculation program calls the corresponding task from the subroutine User_Mod. To create a UD soil model, the User_Mod subroutine must have the following structure:

```
Subroutine User_Mod        (IDTask, iMod, IsUndr, iStep, iTer, Iel,Int, X,
                            Y, Z, Time0, dTime, Props, Sig0, Swp0, StVar0,
                            dEps, D, Bulk_W, Sig, Swp, StVar, ipl, nStat,
                            NonSym, iStrsDep, iTimeDep, iTang, iPrjDir,
                            iPrjLen, iAbort)
```

where:

`IDTask` = Identification of the task (1 = Initialise state variables; 2 = Calculate constitutive stresses; 3 = Create effective material stiffness matrix; 4 = Return the number of state variables; 5 = Return matrix attributes (NonSym, iStrsDep, iTimeDep, iTang); 6 = Create elastic material stiffness matrix)

`iMod` = User-defined soil model number (This option allows for more than one UD model, up to 10.)

`IsUndr` = Drained condition (IsUndr = 0) or undrained condition (IsUndr = 1). In the latter case, PLAXIS will add a large bulk stiffness for water.

`iStep` = Current calculation step number

`iter` = Current iteration number

`Iel` = Current element number

`Int` = Current local stress point number (1..3 for 6-noded elements, or 1..12 for 15-noded elements)

`X,Y,Z` = Global coordinates of current stress point

`Time0` = Time at the start of the current step

`dTime` = Time increment of current step

`Props` = Array(1..50) with User-defined model parameters for the current stress point

`Sig0` = Array(1..20) with previous (= at the start of the current step) effective stress components of the current stress point (σ'^0_{xx}, σ'^0_{yy}, σ'^0_{zz}, σ'^0_{xy}, σ'^0_{yz}, σ'^0_{zx}, p_{steady}, $\Sigma Mstage^0$, $\Sigma Mstage$, Sat, Sat^0, Suc, Suc^0, ΣMsf^0, ΣMsf, 0, 0, 0, 0, 0). In 2D calculations σ_{yz} and σ_{zx} should be zero.

`Swp0` = Previous excess pore pressure of the current stress point

`StVar0` = Array(1..nStat) with previous values of state variables of the current stress point

`dEps`	=	Array(1..12) with strain increments of the current stress point in the current step ($\Delta\epsilon_{xx}$, $\Delta\epsilon_{yy}$, $\Delta\epsilon_{zz}$, $\Delta\gamma_{xy}$, $\Delta\gamma_{yz}$, $\Delta\gamma_{zx}$, ϵ^0_{xx}, ϵ^0_{yy}, ϵ^0_{zz}, γ^0_{xy}, γ^0_{yz}, γ^0_{zx}). In 2D calculations $\Delta\gamma_{yz}$, $\Delta\gamma_{zx}$, γ^0_{yz} and γ^0_{zx} should be zero. In PLAXIS 2D this array may also contain non-local strains. Contact Plaxis for more details.
`D`	=	Effective material stiffness matrix of the current stress point (1..6, 1..6)
`Bulk_W`	=	Bulk modulus of water for the current stress point (for undrained calculations and consolidation)
`Sig`	=	Array (1..6) with resulting constitutive stresses of the current stress point (σ'_{xx}, σ'_{yy}, σ'_{zz}, σ'_{xy}, σ'_{yz}, σ'_{zx})
`Swp`	=	Resulting excess pore pressure of the current stress point
`StVar`	=	Array(1..nStat) with resulting values of state variables for the current stress point
`ipl`	=	Plasticity indicator: 0 = no plasticity, 1 = Mohr-Coulomb (failure) point; 2 = Tension cut-off point, 3 = Cap hardening point, 4 = Cap friction point, 5 = Friction hardening point.
`nStat`	=	Number of state variables (unlimited)
`NonSym`	=	Parameter indicating whether the material stiffness matrix is non-symmetric (NonSym = 1) or not (NonSym = 0) (required for matrix storage and solution).
`iStrsDep`	=	Parameter indicating whether the material stiffness matrix is stress-dependent (iStrsDep = 1) or not (iStrsDep = 0).
`iTimeDep`	=	Parameter indicating whether the material stiffness matrix is time-dependent (iTimeDep = 1) or not (iTimeDep = 0).
`iTang`	=	Parameter indicating whether the material stiffness matrix is a tangent stiffness matrix, to be used in a full Newton-Raphson iteration process (iTang = 1) or not (iTang = 0).
`iPrjDir`	=	Project directory (for debugging purposes)
`iPrjLen`	=	Length of project directory name (for debugging purposes)
`iAbort`	=	Parameter forcing the calculation to stop (iAbort = 1).

In the above, 'increment' means 'the total contribution within the current step' and not per iteration. 'Previous' means 'at the start of the current step', which is equal to the value at the end of the previous step.

In the terminology of the above parameters it is assumed that the standard type of parameters is used, i.e. parameters beginning with the characters A-H and O-Z are double (8-byte) floating point values and the remaining parameters are 4-byte integer values.

The parameters `IDTask` to `dEps` and `iPrjDir` and `iPrjLen` are input parameters; The values of these parameters are provided by PLAXIS and can be used within the subroutine. These input parameters should not be modified (except for `StVar0` in case `IDTask` = 1). The parameters `D` to `iTang` and `iAbort` are output parameters. The values of these parameters are to be determined by the user. In case `IDTask` = 1, `StVar0` becomes output parameter.

The user subroutine should contain program code for listing the tasks and output parameters (`IDTask` = 1 to 6). After the declaration of variables, the User_Mod subroutine must have the following structure (here specified in pseudo code):

```
Case IDTask of
   1  Begin
         { Initialise state variables StVar0 }
      End
   2  Begin
         { Calculate constitutive stresses Sig (and Swp) }
      End
   3  Begin
         { Create effective material stiffness matrix D }
      End
   4  Begin
         { Return the number of state variables nStat }
      End
   5  Begin
         { Return matrix attributes NonSym, iStrsDep,
         iTimeDep }
      End
   6  Begin
         { Create elastic material stiffness matrix De }
      End
End Case
```

If more than one UD model is considered, distinction should be made between different models, indicated by the UD model number `iMod`.

Initialise state variables (`IDTask = 1`)

State variables (also called the hardening parameters) are, for example, used in hardening models to indicate the current position of the yield loci. The update of state variables is considered in the calculation of constitutive stresses based on the previous value of the state variables and the new stress state. Hence, it is necessary to know about the initial value of the state variables, i.e. the value at the beginning of the calculation step. Within a continuous calculation phase, state variables are automatically transferred from one calculation step to another. The resulting value of the state variable in the previous step, StVar, is stored in the output files and automatically used as the initial value in the current step, StVar0. When starting a new calculation phase, the initial

value of the state variables is read from the output file of the previous calculation step and put in the StVar0 array. In this case it is not necessary to modify the StVar0 array.

However, if the previous calculation step does not contain information on the state variables (for example in the very first calculation step), the StVar0 array would contain zeros. For this case the initial value has to be calculated based on the actual conditions (actual stress state) at the beginning of the step. Consider, for example, the situation where the first state variable is the minimum mean effective stress, p' (considering that compression is negative). If the initial stresses have been generated using the K_0-procedure, then the initial effective stresses are non-zero, but the initial value of the state variable is zero, because the initialization of this user-defined variable is not considered in the K_0-procedure. In this case, part 1 of the user subroutine may look like:

```
1  Begin
      { Initialise state variables StVar0}
      p = (Sig0[1] + Sig0[2] + Sig0[3] ) / 3.0
      StVar0[1] = Min(StVar0[1] ,p)
   End
```

Calculate constitutive stresses (IDTask = 2)

This task constitutes the main part of the user subroutine in which the stress integration and correction are performed according to the user-defined soil model formulation. Let us consider a simple example using a linear elastic D-matrix as created under IDTask = 3.

In this case the stress components, Sig, can directly be calculated from the initial stresses, Sig0, the material `stiffness` matrix, D, and the strain increments, dEps: Sig[i]= Sig0[i] + $\sum$ (D[i, j]*dEps[j]). In this case, part 2 of the user subroutine may look like:

```
2  Begin
      { Calculate constitutive stresses Sig (and Swp) }
      For i=1 to 6 do
          Sig[i] = Sig0[i]
          For j=1 to 6 do
             Sig[i] = Sig[i] + D[i,j]*dEps[j]
          End for {j}
      End for {i}
   End
```

Create effective material stiffness matrix (IDTask = 3)

The material stiffness matrix, D, may be a matrix containing only the elastic components of the stress-strain relationship (as it is the case for the existing soil models in PLAXIS), or the full elastoplastic material stiffness matrix (tangent stiffness matrix). Let us consider the very simple example of Hooke's law of isotropic linear elasticity. There are only two model parameters involved: Young's modulus, E, and Poisson's ratio, ν. These parameters are stored, respectively, in position 1 and 2 of the model parameters array, Props(1..50). In this case, part 3 of the user subroutine may look like:

```
3  Begin
      { Create effective material stiffness matrix D}
      E = Props[1]
      v = Props[2]
```

```
    G = 0.5*E/(1.0+v)
    Fac = 2*G/(1.0-2*v)   { make sure that v < 0.5 !! }
    Term1 = Fac*(1-v)
    Term2 = Fac*v
    D[1,1] = Term1
    D[1,2] = Term2
    D[1,3] = Term2
    D[2,1] = Term2
    D[2,2] = Term1
    D[2,3] = Term2
    D[3,1] = Term2
    D[3,2] = Term2
    D[3,3] = Term1
    D[4,4] = G
    D[5,5] = G
    D[6,6] = G
  End
```

(By default, D will be initialized to zero, so the remaining terms are still zero; however, it is a good habit to explicitly define zero terms as well.)

If undrained behaviour is considered (IsUndr = 1), then a bulk stiffness for water (Bulk_W) must be specified at the end of part 3. After calling the user subroutine with IDTask = 3 and IsUndr = 1, PLAXIS will automatically add the stiffness of the water to the material `stiffness` matrix D such that: D[*i*=1..3, *j*=1..3] = D[*i*,*j*]+ Bulk_W. If Bulk_W is not specified, PLAXIS will give it a default value of 100*Avg(D[*i*=1..3, *j*=1..3]).

Return the number of state variables (`IDTask = 4`)

This part of the user subroutine returns the parameter nStat, i.e. the number of state variables. In the case of just a single state parameter, the user subroutine should look like:

```
4   Begin
      { Return the number of state variables nStat }
      nStat = 1
    End
```

Return matrix attributes (`IDTask = 5`)

The material stiffness matrix may be stress-dependent (such as in the Hardening Soil model) or time-dependent (such as in the Soft Soil Creep model). When using a tangent stiffness matrix, the matrix may even be non-symmetric, for example in the case of non-associated plasticity. The last part of the user subroutine is used to initialize the matrix attributes in order to update and store the global stiffness matrix properly during the calculation process. For the simple example of Hooke's law, as described earlier, the matrix is symmetric and neither stress- nor time-dependent. In this case the user subroutine may be written as:

```
5   Begin
      { Return matrix attributes NonSym, iStrsDep, }
      { iTimeDep, iTang }
      NonSym = 0
```

```
    iStrsDep = 0
    iTimeDep = 0
    iTang = 0
End
```

For NonSym = 0 only half of the global stiffness matrix is stored using a profile structure, whereas for Nonsym = 1 the full matrix profile is stored.

For iStrsDep = 1 the global stiffness matrix is created and decomposed at the beginning of each calculation step based on the actual stress state (modified Newton-Raphson procedure).

For iTimeDep = 1 the global stiffness matrix is created and decomposed every time when the time step changes.

For iTang = 1 the global stiffness matrix is created and decomposed at the beginning of each iteration based on the actual stress state (full Newton-Raphson procedure; to be used in combination with iStrsDep=1).

Create elastic material stiffness matrix (`IDTask = 6`)

The elastic material stiffness matrix, D^e, is the elastic part of the effective material stiffness matrix as described earlier.

In the case that the effective material stiffness matrix was taken to be the elastic stiffness matrix, this matrix may just be adopted here. However in the case that an elastoplastic or tangent matrix was used for the effective stiffness matrix, then the matrix to be created here should only contain the elastic components.

The reason that an elastic material stiffness matrix is required is because PLAXIS calculates the current relative global stiffness of the finite element model as a whole (CSP = Current Stiffness Parameter). The CSP parameter is defined as:

$$CSP = \frac{Total\ work}{Total\ elastic\ work}$$

The elastic material stiffness matrix is required to calculate the total elastic work in the definition of the CSP. The CSP equals unity if all the material is elastic whereas it gradually reduces to zero when failure is approached.

The CSP parameter is used in the calculation of the global error. The global error is defined as:

$$Global\ error = \frac{|unbalance\ force|}{|currently\ activated\ load| + CSP \cdot |previously\ activated\ load|}$$

The unbalance force is the difference between the external forces and the internal reactions. The currently activated load is the load that is being activated in the current calculation phase, whereas the previously activated load is the load that has been activated in previous calculation phases and that is still active in the current phase.

Using the above definition for the global error in combination with a fixed tolerated error results in an improved equilibrium situation when plasticity increases or failure is approached. The idea is that a small out-of-balance is not a problem when a situation is mostly elastic, but in order to accurately calculate failure state, safety factor or bearing capacity, a stricter equilibrium condition must be adopted.

Part 6 of the user subroutine looks very similar to part 3, except that only elastic components are considered here. It should be noted that the same variable D is used to store the elastic material stiffness matrix, whereas in Part 3 this variable is used to store the effective material stiffness matrix.

```
6  Begin
      { Create elastic material stiffness matrix D }
      D[1,1] =
      D[1,2] =
      D[1,3] =
      .....
      D[6,6] =
   End
```

Using predefined subroutines from the source code

In order to simplify the creation of user subroutines, a number of FORTRAN subroutines and functions for vector and matrix operations are available in the source code (to be included in the file with the user subroutine). The available subroutines may be called by User_Mod subroutine to shorten the code. An overview of the available subroutines is given in Appendix C.

Definition of user-interface functions

In addition to the user-defined model itself it is possible to define functions that will facilitate its use within the Plaxis user-interface. If available, Plaxis Input will retrieve information about the model and its parameters using the procedures described hereafter.

```
procedure GetModelCount(var C:longint) ;
```

`C` = number of models (return parameter)

This procedure retrieves the number of models that have been defined in the DLL. PLAXIS assumes that model IDs are successive starting at model ID = 1.

```
procedure GetModelName(var iModel : longint;
                       var Name : shortstring) ;
```

`iModel` = User-defined soil model number to retrieve the name for (input parameter)

`Name` = model name (return parameter)

This procedure retrieves the names of the models defined in the DLL.

```
procedure GetParamCount(var iModel : longint; var C :longint) ;
```

`iModel` = User-defined soil model number (input parameter)

`C` = number of parameters for the specified model (return parameter)

This procedure retrieves the number of parameters of a specific model.

```
procedure GetParamName(var iModel,iParam : longint;
                       var Name : shortstring);
```

`iModel` = User-defined soil modelnumber (input parameter)

`iParam` = Parameter number (input parameter)

`Name` = parameter name (return parameter)

This procedure retrieves the parameter name of a specific parameter.

```
Procedure GetParamUnit(var iModel,iParam : longint;
                           var Units : shortstring) ;
```

`iModel` = User-defined soil model number (input parameter)

`iParam` = Parameter number (input parameter)

`Units` = Parameter units (return parameter)

This procedure retrieves the parameter units of a specific parameter. Since the chosen units are dependent on the units of length, force and time chosen by the user the following characters should be used for defining parameter units:

'L' or 'l' for units of length 'F' or 'f' for units of force 'T' or 't' for units of time.

For model names, model parameter names and model parameter units special characters can be used for indicating subscript, superscript or symbol font (for instance for Greek characters).

`^` : From here characters will be superscript

`_` : From here characters will be subscript

`@` : From here characters will be in symbol font

`#` : Ends the current superscript or subscript.

Pairs of '^..#', '_...#' and '@...#' can be nested.

For example:

A UD model parameter uses the oedometer stiffness as parameter. The parameter name can be defined as 'E_oed#' and its units as 'F/L^2#'.

When defining a unit containing one of the letters 'l', 'f' or 't', like 'cal/mol', these letters will be replaced by the unit of length, the unit of force or the unit of time respectively. To avoid this, these letters should be preceded by a backslash. For example 'cal/mol' should be defined as 'ca\l/mo\l' to avoid getting 'cam/mom'.

The state variables to be displayed in the Output program can be defined.

```
procedure GetStateVarCount(var iModel : longint; var C :longint) ;
```

`iModel` = User-defined soil model number (input parameter)

`C` = number of state variables for the specified model (return parameter)

This procedure retrieves the number of state variables of a specific model.

```
procedure GetStateVarName(var iModel,iParam : longint;
                           var Name : shortstring);
```

```
iModel     =          Used-defined soil model number (input parameter)

iParam     =          Parameter number (input parameter)

Name       =          parameter name (return parameter)
```

This procedure retrieves the state parameter name of a specific parameter.

```
Procedure GetStateVArUnit(var iModel,iParam : longint;
                          var Units : shortstring) ;

iModel     =          User-defined soil model number (input parameter)

iParam     =          Parameter number (input parameter)

Units      =          Parameter units (return parameter)
```

This procedure retrieves the state parameter units of a specific parameter.

All procedures are defined in Pascal but equivalent procedures can be created, for instance in a Fortran programming language. Please make sure that the data format of the parameters in the subroutine headers is identical to those formulated before. For instance, the procedures mentioned above use a "shortstring" type; a "shortstring" is an array of 256 characters where the first character contains the actual length of the shortstring contents. Some programming languages only have null-terminated strings; in this case it may be necessary to use an array of 256 bytes representing the ASCII values of the characters to return names and units. An example of Fortran subroutines is included in the software package.

Compiling the user subroutine

The user subroutine User_Mod has to be compiled into a DLL file using an appropriate compiler. Note that the compiler must have the option for compiling DLL files. Below are examples for two different FORTRAN compilers. It is supposed that the user subroutine User_Mod is contained in the file USRMOD.FOR.

After creating the user subroutine User_Mod, a command must be included to export data to the DLL.

The following statement has to be inserted in the subroutine just after the declaration of variables:

- Using Lahey Fortran (LF95, ...): DLL_Export User_Mod
- Using Intel Visual Fortran: !DEC$ ATTRIBUTES DLLExport,StdCall,Reference :: User_Mod

In order to compile the USRMOD.FOR into a DLL file, the following command must be executed:

- Using Lahey Fortran 90: LF90 -win -dll USRMOD.FOR -lib LFUsrLib -ml bd
- Using Lahey Fortran 95: LF95 -win -dll USRMOD.FOR -lib LFUsrLib -ml bd
- Using Intel Visual Fortran: ifort /winapp USRMOD.FOR DFUsrLib.lib /dll
- Using GCC compiler: g55 USRMOD.FOR -o usermod.dll -shared -fcase -upper -fno-underscoring -mrtd

In all cases USRMOD.DLL file will be created. It can be renamed to 'any' .dll. This file should be placed in the **usdm** folder under the PLAXIS program directory, thereafter it can be used together with the existing PLAXIS calculations program (PLASW.EXE in PLAXIS 2D or PLASW3DF.EXE in PLAXIS 3D). Once the UD model is used, PLAXIS will execute the commands as listed in the USRMOD.DLL file.

In order to compile as 64-bit, you need both a 32-bit compiled 'USRMOD.DLL' and a 64-bit compiled ' USRMOD64.DLL' file in the **usdm** folder under the PLAXIS program directory. The last one only needs to contain the 'User_Mod' subroutine.

Debugging possibilities

When making computer programs, usually some time is spent to 'debug' earlier written source code. In order to be able to effectively debug the user subroutine, there should be a possibility for the user to write any kind of data to a file. Such a 'debug-file' is not automatically available and has to be created in the user subroutine.

After the debug-file is created, data can be written to this file from within the user subroutine. This can be done by using, for example, the availably written subroutines (Section A).

1.3 INPUT OF UD MODEL PARAMETERS VIA USER-INTERFACE

Input of the model parameters for user-defined soil models can be done using PLAXIS material data base. In fact, the procedure is very similar to the input of parameters for the existing PLAXIS models.

When creating a new material data set for soil and interfaces in the material data base, a window appears with three tabsheets: *General*, *Parameters*, *Interface*, Figure 1.1. A user-defined model can be selected from the *Material model* combo box in the *General* tabsheet.

After inputting general properties, the appropriate UD model can be chosen from the available models that have been found by PLAXIS Input.

The *Parameters* tabsheet shows two combo boxes; the top combo box lists all the DLLs that contain valid UD models and the next combo box shows the models defined in the selected DLL. Each UD model has its own set of model parameters, defined in the same DLL that contains the model definition.

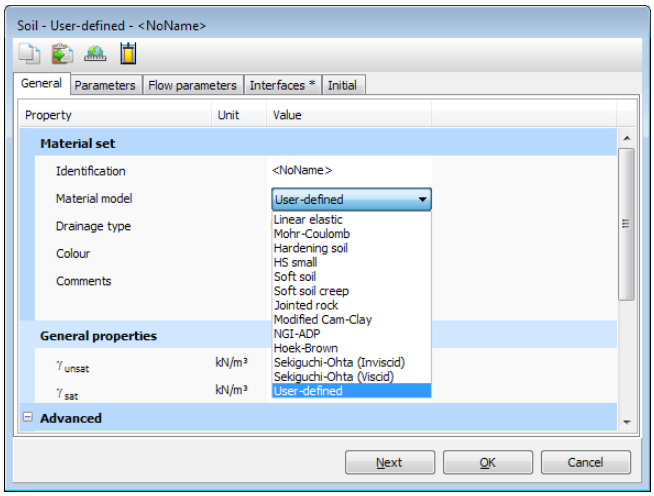
When an available model is chosen PLAXIS will automatically read its parameter names and units from the DLL and fill the parameter table below.

Interfaces

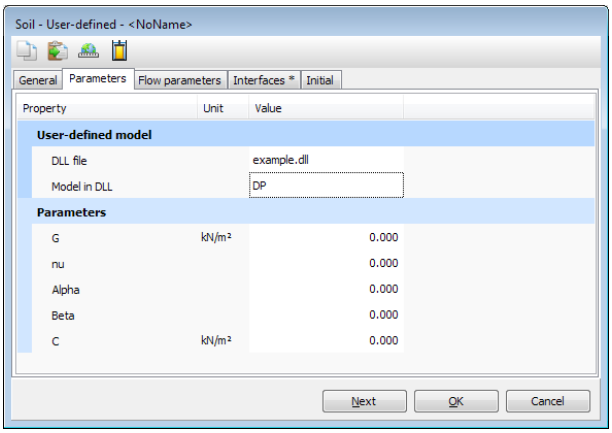
The *Interfaces* tabsheet, Figure 1.2, contains the material data for interfaces.

Normally, this tabsheet contains the R_{inter} parameter. For user-defined soil models the interface tabsheet is slightly different and contains the interface oedometer modulus, E_{oed}^{ref}, and the interface strength parameters c_{inter},φ_{inter} and ψ_{inter}. Hence, the interface shear strength is directly given in strength parameters instead of using a factor relating the interface shear strength to the soil shear strength, as it is the case in PLAXIS models.

After having entered values for all parameters, the data sets can be assigned to the



a. Selection of user-defined soil models



b. Input of parameters

Figure 1.1 *Selection* window

corresponding soil clusters, in a similar way as for the existing material models in PLAXIS. The user-defined parameters are transmitted to the calculation program and appear for the appropriate stress points as Props(1..50) in the User_Mod subroutine.

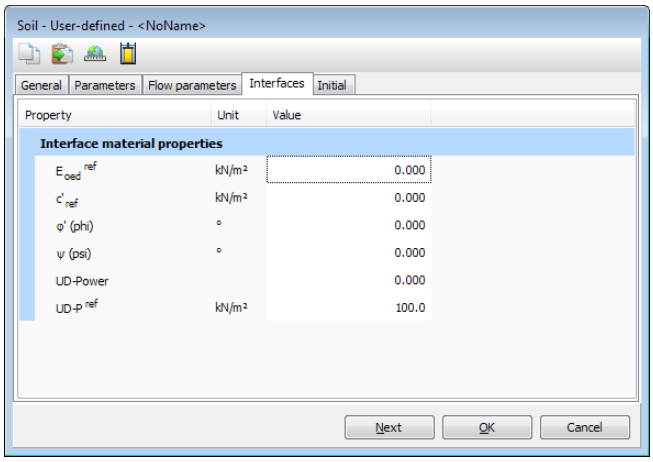


Figure 1.2 Interface tabsheet

APPENDIX A - FORTRAN SUBROUTINES FOR USER-DEFINED SOIL MODELS

In this appendix, a listing is given of the subroutines and functions which are provided by PLAXIS in libraries and source code in the User-defined soil models directory. These can be called by the User_Mod subroutine:

Subroutines

`MZeroR( R, K ):`

To initialize *K* terms of double array *R* to zero

`MZeroI( I, K ):`

To initialize *K* terms of integer array *I* to zero

`SetRVal( R, K, V ):`

To initialize *K* terms of double array *R* to *V*

`SetIVal( I, K, IV ):`

To initialize *K* terms of integer array *I* to *IV*

`CopyIVec( I1, I2, K ):`

To copy *K* values from integer array *I1* to *I2*

`CopyRVec( R1, R2, K ):`

To copy *K* values from double array *R1* to *R2*

`MulVec( V, F, n ):`

To multiply a vector *V* by a factor *F*, *n* values

`MatVec( xMat, im, Vec, n, VecR ):`

Matrix (xMat)-vector(Vec) operation.

First dimension of matrix is *im*; resulting vector is *VecR*

`AddVec( Vec1, Vec2, R1, R2, n, VecR ):`

To add n terms of two vectors; result in *VecR*

$$VecR_i = R1 \cdot Vec1_i + R2 \cdot Vec2_i$$

`MatMat( xMat1, id1, xMat2, id2, nR1, nC2, nC1, xMatR, idR):`

Matrix multiplication $xMatR_{ij} = xMat1_{ik} \cdot xMat2_{kj}$

id1, *id2*, *idR* : first dimension of matrices

nR1 number of rows in *xMat1* and resulting *xMatR*

nC2 number of column in *xMat2* and resulting *xMatR*

nC1 number of columns in *xMat2* =rows in *xMat2*

`MatMatSq( n, xMat1, xMat2, xMatR ):`

Matrix multiplication $xMatR_{ij} = xMat1_{ik} \cdot xMat2_{kj}$

Fully filled square matrices with dimensions *n*

`MatInvPiv( AOrig, B, n ):`

Matrix inversion of square matrices *AOrig* and *B* with dimensions *n*.

AOrig is NOT destroyed, *B* contains inverse matrix of *AOrig*.

Row-pivoting is used.

`WriVal( io, C, V ):`

To write a double value *V* to file unit *io* (when *io* $>$ 0)

The value is preceded by the character string *C.*

`WriIVl( io, C, I ):`

As *WriVal* but for integer value *I*

`WriVec( io, C, V, n ):`

As *WriVal* but for *n* values of double array *V*

`WriIVc( io, C, iV, n ):`

As *WriVal* but for *n* values of integer array *iV*

`WriMat( io, C, V, nd, nr, nc ):`

As *WriVal* but for double matrix *V*. *nd* is first dimension of *V*, *nr* and *nc* are the number of rows and columns to print respectively.

`PrnSig( iOpt, S, xN1, xN2, xN3, S1, S2, S3, P, Q ):`

To determine principal stresses and (for *iOpt*=1) principal directions.

iOpt = 0 to obtain principal stresses without directions

iOpt = 1 to obtain principal stresses and directions

S array containing 6 stress components (XX, YY, ZZ, XY, YZ, ZX)

xN1, *xN2*, *xN3* array containing 3 values of principal normalized directions only when *iOpt*=1.

S1, *S2*, *S3* sorted principal stresses ($S \leq S2 \leq S3$)

P isotropic stress (negative for compression)

Q deviatoric stress

`CarSig( S1, S2, S3, xN1, xN2, xN3, SNew ):`

To calculate Cartesian stresses from principal stresses and principal directions.

S1, *S2*, *S3* principal stresses

xN1, *xN2*, *xN3* arrays containing principal directions (from `PrnSig`)

SNew contains 6 stress components (XX, YY, ZZ, XY, YZ, ZX)

`CrossProd( xN1, xN2, xN3 ):`

Cross product of vectors *xN1* and *xN2*

`SetVecLen( xN, n, xL ):`

To multiply the *n* components of vector *xN* such that the length of *xN* becomes *xL* (for example to normalize vector *xN* to unit length).

Functions

`Logical Function LEqual( A, B, Eps ):`

Returns *TRUE* when two values *A* and *B* are almost equal, *FALSE* otherwise.

LEqual = $|A\text{-}B| < Eps * (|A| + |B| + Eps) / 2$

`Logical Function IsOArr( A, n ):`

Returns *TRUE* when all *n* values of real (double) array *A* are zero, *FALSE* otherwise

`Logical Function IsOIArr( IArr, n ):`

Returns *TRUE* when all *n* values of integer array *IArr* are zero, *FALSE* otherwise

`Double Precision Function DInProd( A, B, n ):`

Returns the dot product of two vectors with length *n*