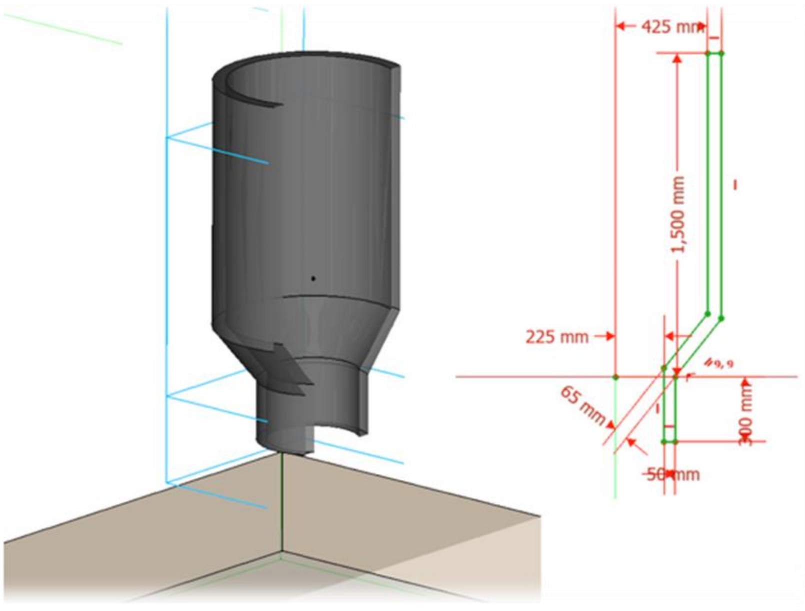
GeoXPM

*Particle-based continuum solutions for extreme geoengineering, geomechanics & geophysics applications*



# GeoXPM Tutorial Manual

***Ha H. Bui***

***Tien V. Nguyen***

***Giang D. Nguyen***

## Introduction

GeoXPM is a new generation of computational software package that was originally developed by Prof Ha H. Bui and his research team at Monash University in Australia based on the Smoothed Particle Hydrodynamics (SPH) method. It has been developed specifically to address the ever-evolving demands of solving intricate geotechnical engineering, geomechanics and geophysics challenges, particularly those involving fracturing, large deformation and flow behaviours of geomaterials. One standout feature of GeoXPM is its interactive graphical user-interface, which empowers users to effortlessly generate and modify complex computational models without encountering any obstacles. The calculation procedure within GeoXPM is both robust and fully automated, eliminating the need for extensive training or specialised knowledge. With GeoXPM, academics and professionals alike can unlock the full potential of engineering applications, pushing boundaries and delivering remarkable results that exceed expectations.

The tutorial explores a wide range of challenging and practical scenarios that encompass the intricate aspects of large deformation and post-failure behaviour in geomaterials. The main aim is to equip new users with a comprehensive understanding of GeoXPM. However, it is crucial to recognise that this tutorial should not be solely relied upon as the basis for practical projects. Instead, it serves as an invaluable resource to familiarise users with the functionalities and capabilities of the software.

Users are expected to possess a fundamental understanding of the SPH method and constitutive modelling of geomaterials. For those who are new to the software, it is highly recommended that they adhere to the default setting parameters and follow the step-by-step procedure meticulously to ensure accurate results. It is important to note that variations in hardware and software configurations may yield slight differences in the obtained results.

This Tutorial Manual does not provide the theoretical background information on the SPH method or elaborate on the specific details of the constitutive models utilised within the software. Therefore, users are highly advised to refer to the provided references for comprehensive information regarding the SPH method and constitutive models implemented in GeoXPM. These references offer in-depth insights into the underlying principles and methodologies employed, allowing users to gain a thorough understanding of the theoretical aspects involved.

# Tutorial:

# User-defined constitutive model

GeoXPM offers users the flexibility to incorporate their own constitutive models through a user-defined feature. To implement this, users need to program the model in FORTRAN and compile it as a Dynamic Link Library (DLL) before adding the DLL to a project folder.

In general, for the user-defined constitutive model, information from the previous step will be provided, based on which users will calculate and update the information for the current step. Below details on the last step will be provided:

- Stresss (in Voigt notation).
- Strains (in Voigt notation).
- Strain rate (in Voigt notation).
- Spin rate (in Voigt notation).
- Time increment.
- State variables.
- Type of calculation.

The template for the implementation of the user-defined constitutive model is provided with the software package. In general, users will have to implement 2 main tasks corresponding to 2 different subroutines:

i. Initialising model parameters. (“udm_params” subroutine)
ii. Calculating and updating stresses, strains and state parameters. (“udc” subroutine)

Once the codes are ready, the intel Fortran compiler will be called to create a DLL file which will be copied into the related project folder. Users can download and install the intel Fortran compiler included in the intel oneAPI HPC Toolkit following this link (oneAPI Base Toolkit is not required).

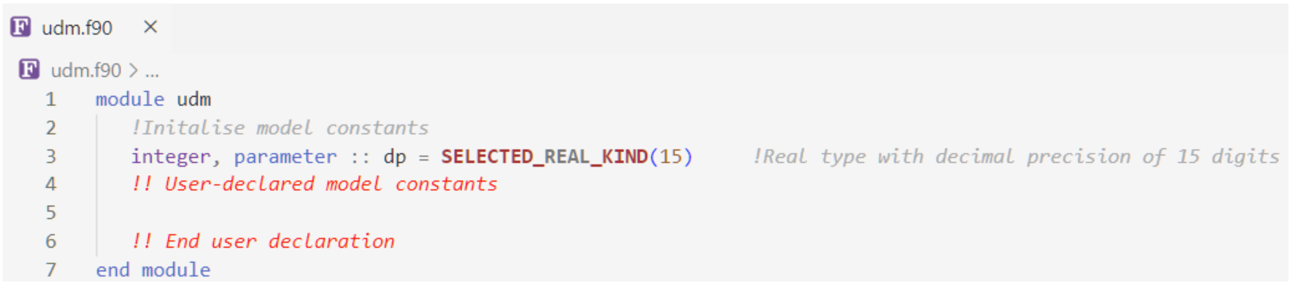
**Note:** Other programming languages can also be used to create the dynamic library as long as the Interlanguage Communication standard is met.

## ➢ Step 1. Modifying template files

### i. Initialising model parameters.:

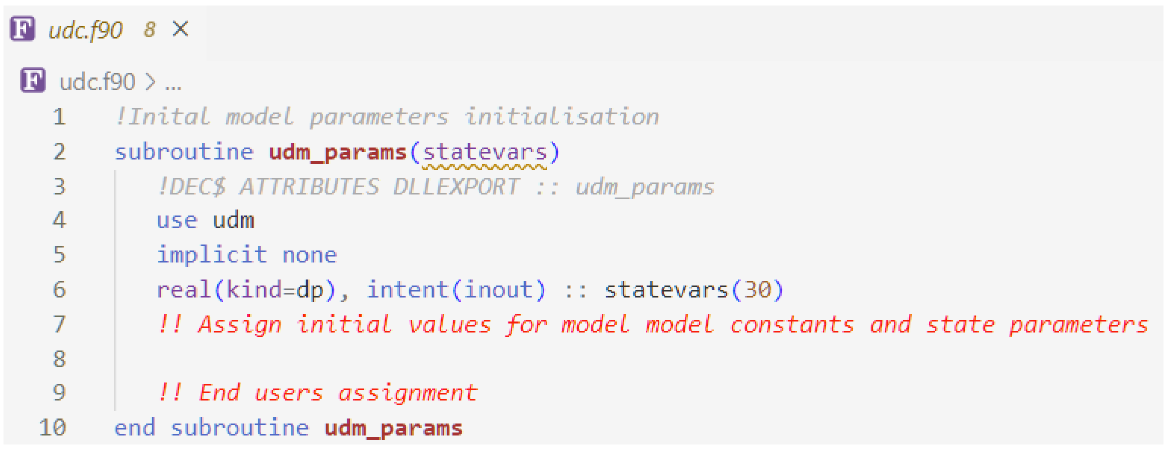
In this task, users will have to do 3 smaller steps:

a. Declare model constants (for example: Young modulus, Poisson ratio…). This is done by editing the “udm” module in the fortran “udm.f90” file.



```fortran
module udm
   !Initalise model constants
   integer, parameter :: dp = SELECTED_REAL_KIND(15)      !Real type with decimal precision of 15 digits
   !! User-declared model constants

   !! End user declaration
end module
```

b. Initialise values for model constants and state parameters (if needed) in subroutine “udm_params” in the fortran “udc.f90” file.



udc.f90 8 ×

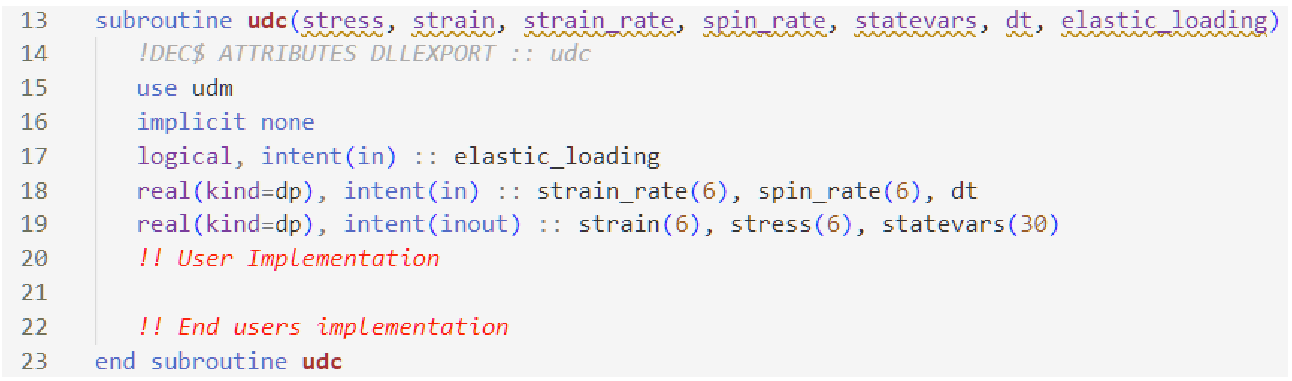
udc.f90 > ...

```fortran
!Inital model parameters initialisation
subroutine udm_params(statevars)
    !DEC$ ATTRIBUTES DLLEXPORT :: udm_params
    use udm
    implicit none
    real(kind=dp), intent(inout) :: statevars(30)
    !! Assign initial values for model model constants and state parameters

    !! End users assignment
end subroutine udm_params
```

**Note**: The user declaration part lies between the 2 commented red lines. The rest of the code needs to be kept unchanged.

## ii. Calculating and updating stresses, strains and state parameters:

In this task, users must implement their constitutive model in the subroutine “udc” in the fortran “udc.f90” file. The structure of the subroutine must have the following format:



```fortran
subroutine udc(stress, strain, strain_rate, spin_rate, statevars, dt, elastic_loading)
    !DEC$ ATTRIBUTES DLLEXPORT :: udc
    use udm
    implicit none
    logical, intent(in) :: elastic_loading
    real(kind=dp), intent(in) :: strain_rate(6), spin_rate(6), dt
    real(kind=dp), intent(inout) :: strain(6), stress(6), statevars(30)
    !! User Implementation

    !! End users implementation
end subroutine udc
```

Where

- “stress” is a Voigt vector of size 6: $\boldsymbol{\sigma} = (\sigma_x, \sigma_y, \sigma_z, \sigma_{xy}, \sigma_{xz}, \sigma_{yz})$
- “strain” is a Voigt vector of size 6: $\boldsymbol{\varepsilon} = (\varepsilon_x, \varepsilon_y, \varepsilon_z, \varepsilon_{xy}, \varepsilon_{xz}, \varepsilon_{yz})$
- “strain_rate” is a Voigt vector of size 6: $\dot{\boldsymbol{\varepsilon}} = \left(\dot{\varepsilon}_x, \dot{\varepsilon}_y, \dot{\varepsilon}_z, \dot{\varepsilon}_{xy}, \dot{\varepsilon}_{xz}, \dot{\varepsilon}_{yz}\right)$
- “spin_rate” is a Voigt vector of size 6: $\dot{\boldsymbol{\omega}} = \left(\dot{\omega}_x, \dot{\omega}_y, \dot{\omega}_z, \dot{\omega}_{xy}, \dot{\omega}_{xz}, \dot{\omega}_{yz}\right)$. Note that, unlike stress, strain and strain rate tensors, which are symmetric, the spin rate tensor is an antisymmetric tensor $\dot{\boldsymbol{\omega}} = \begin{pmatrix} \dot{\omega}_x & \dot{\omega}_{xy} & \dot{\omega}_{xz} \\ -\dot{\omega}_{xy} & \dot{\omega}_y & \dot{\omega}_{yz} \\ -\dot{\omega}_{xz} & -\dot{\omega}_{yz} & \dot{\omega}_z \end{pmatrix}$. Here, we establish a convention that the upper triangular of the spin rate tensor will be used to convert the spin rate tensor to its Voigt notation.
- “statevars”: a vector of size 30 that can be used to store state variables per user needs (For example: plastic strains, void ratio…)
- “dt” is the timestep increment that will be used in conjunction with the strain and spin rates to calculate the objective stress increment. (Large deformation applications with rigid body rotation need an objective stress rate)

- “elastic_loading” is a boolean variable to distinguish between the purely elastic and elastoplastic stages. When “elastic_loading” is True, then the correct constitutive responses return from the “udc” subroutine has to be a purely elastic one.

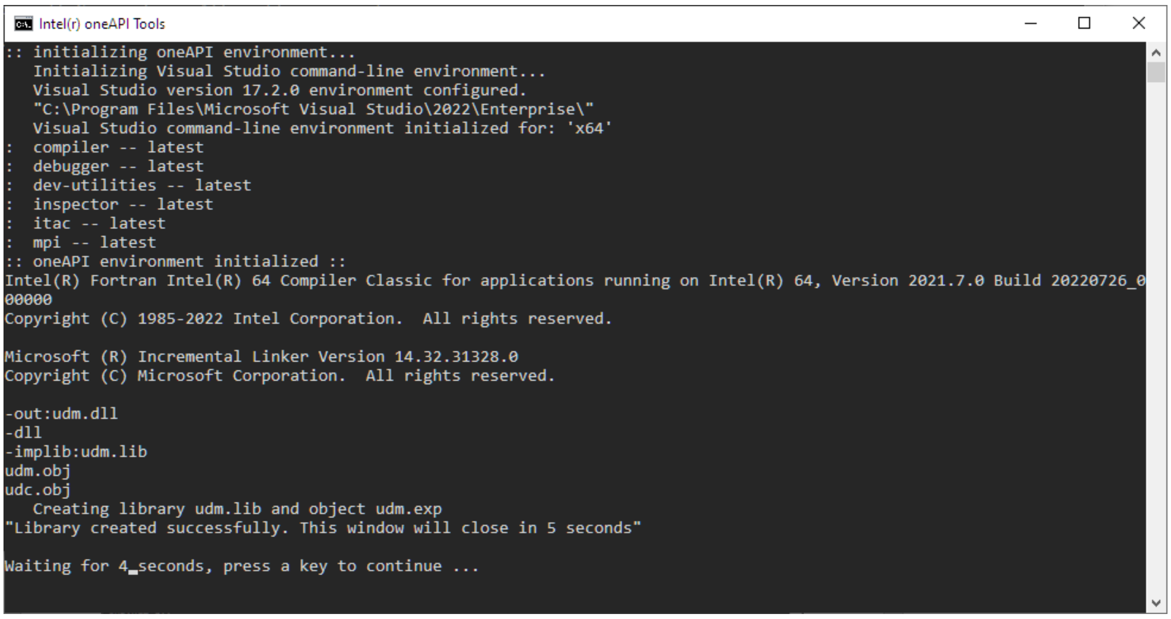
The data types for all the above variables except for “elastic_loading” must use real with 15 decimal digits precision (kind=dp).

**Note**: The user declaration part lies between the 2 commented red lines, the rest of the code needs to be kept unchanged.

## ➢ Step 2. Creating a dynamic-link library (DLL) file

To create the library, simply run the file “CreateLibrary.bat” (require installing intel oneAPI HPC toolkit). This file will activate the Fortran compiler environment to compile the “udm.dll” library file. Note that the three files “CreateLibrary.bat”, “udm.f90”, and “udc.f90” have to be in the same folder.

```
@echo off
@REM Remove previous library
rm udm.dll
@REM Activate compiler environment
call "C:\Program Files (x86)\Intel\oneAPI\setvars.bat"
@REM Compile the dynamic link library
ifort /dll udm.f90 udc.f90
@REM Output to user
IF %ERRORLEVEL% EQU 0 (
echo "Library created successfully. This window will close in 5 seconds"
) else (
echo "Library created failed"
)
@REM Remove intermediate files
rm *.obj *.exp *.mod *.lib
timeout -t 5
```

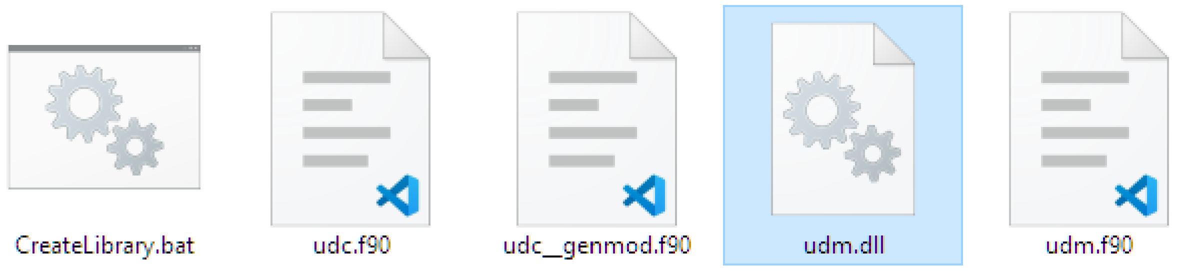


```
:: initializing oneAPI environment...
   Initializing Visual Studio command-line environment...
   Visual Studio version 17.2.0 environment configured.
   "C:\Program Files\Microsoft Visual Studio\2022\Enterprise\"
   Visual Studio command-line environment initialized for: 'x64'
:  compiler -- latest
:  debugger -- latest
:  dev-utilities -- latest
:  inspector -- latest
:  itac -- latest
:  mpi -- latest
:: oneAPI environment initialized ::
Intel(R) Fortran Intel(R) 64 Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0 Build 20220726_0
00000
Copyright (C) 1985-2022 Intel Corporation.  All rights reserved.

Microsoft (R) Incremental Linker Version 14.32.31328.0
Copyright (C) Microsoft Corporation.  All rights reserved.

-out:udm.dll
-dll
-implib:udm.lib
udm.obj
udc.obj
   Creating library udm.lib and object udm.exp
"Library created successfully. This window will close in 5 seconds"

Waiting for 4 seconds, press a key to continue ...
```

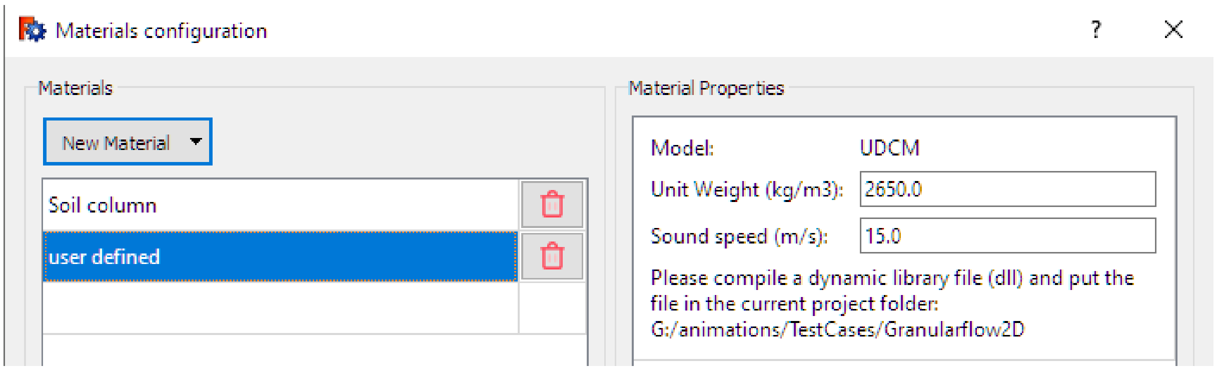
**Note:** the name of the output file “udm.dll” must not be changed.

## ➢ Step 3. Link with current project

1. To use the newly created model in a project, simply copy the output "udm.dll" library file into the project folder.



2. In GeoXPM. When creating a material model, choose "User-defined constitutive model" and then input the required parameters into the Material properties dialogue



Model: UDCM
Unit Weight (kg/m3): 2650.0
Sound speed (m/s): 15.0
Please compile a dynamic library file (dll) and put the file in the current project folder: G:/animations/TestCases/Granularflow2D

**Hint:** Timestep for SPH for granular material depends heavily on the sound speed. For an elastoplastic model, we normally calculate this sound speed as the square root of the Young modulus over the material's density ($c = \sqrt{E/\rho}$).

## References:

1. Bui H.H, Nguyen V.T & Nguyen G.D (2023). GeoXPM: Particle-based continuum solutions for extreme geoengineering, geomechanics & geophysics applications, Tutorial Manual. Link
2. Bui H.H, Nguyen G.D (2021). Smoothed particle hydrodynamics (SPH) and its applications in geomechanics: From solid fracture to granular behaviour and multiphase flows in porous media, *Computers and Geotechnics* 138, 104315. Link
3. Bui H.H, Fukagawa R, Sako K & Ohno S (2008). Lagrangian mesh-free particles method (SPH) for large deformation and failure flows of geomaterial using elastic-plastic soil constitutive model, International Journal for Numerical and Analytical Methods in Geomechanics, Vol.32(12), pp.1537-1570. Link