TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG
The University of Resources. Since 1765.

MonolithFE² – A monolithic FE² implementation for Abaqus Version 2.0

Nils Lange, Geralf Hütter, Björn Kiefer

February 16, 2024

Contents

1 Description

MonolithFE^{2}is an open-source program for Abaqus, distributed under a CC BY-NC-SA 4.0 license. Details on the implemented theory can be found in [1]. Please refer to this publication if you use MonolithFE^2.

The main idea behind the present FE^2 implementation is to use Abaqus Standard for solving the macro FE problem and a self-written light-weight code for solving the micro problems. The micro-macro data exchange is performed through the Abaqus UMAT interface. MonolithFE^2 employs the UEL interface of Abaqus at the micro-scale. An UEL is shipped with MonolithFE^2. This UEL comprise a certain number of established element types and employs the UMAT interface for the material law at the micro-scale. Thus, previously developed UELs and UMATs can be employed directly at the micro-scale and be tested in Abaqus directly independent of MonolithFE^2. By default, an elastic-plastic MISES material routine UMAT in rate formulation is employed at the microscale. The preprocessing for the microscale is done in Abaqus/CAE with aid of a Python plug-in. The postprocessing of selected microscopic FE problems can be done by a re-simulation in Abaqus with the actual deformation history of a macroscopic integration point.
To gain further speedup, Reduced Order Modeling (ROM) with hyper integration using an improved ECM [8] algorithm can be used. To use the hyper reduced ROM method a tool to simulate and evaluate training data is provided (the necessary driver Routine UMAT_Driver is provided as a separate project). This method can be used together with the staggered or monolithic approach.
All beginnings are difficult. Even though the program and all of its components were implemented carefully and user friendly, it is still a code from academia. A short familiarization with the program might be needed, to understand how all parts are joined together. We strongly adviced to start with section 2.1 "First Success", to get your first FE^2 model running in Abaqus and to discover that working with MonolithFE^2 is not complicated.

1.1 Features

- monolithic and staggered algorithm
- periodic boundary conditions at micro-scale
- small deformation and large deformation theory
- UMAT interface to Abaqus at macro-scale
- Intel MKL PARDISO solver on the microscale (full simulation)
  - for symmetric and unsymmetric (but structurally symmetric) matrices
  - parallelizable: Intel MKL PARDISO solver can be run in parallel for each macroscopic integration point (limited to a shared memory architecture and not necessarily favorable, since the FE^2-method parallizes very well)
- INTEL MKL LAPACK solver on the microscale (ROM simulation)
- modular concept: UEL, UMAT and UHARD interfaces at micro-scale for easy extensibility
- implemented element types (via UELlib [3]):

  - plane stress (nested Newton algorith after Dodds, 1987), plane strain and 3D
  - different element types (quadrilateral, triangular, tetrahedral, hexagonal)
  - linear and quadratic shape functions with full or reduced integration[1]
  - (rate-independent) elastic-plastic MISES material in rate formulation as micro UMAT with tabular yield curve (as UHARD) as standard material routine, other simple hyperelastic routines provided
- Python plugin for preprocessing and postprocessing of micro-scale models in Abaqus/-CAE (meshing, material assignment, convergence parameters etc.)
- ROM projection and hyper integration (ROM also without hyper integration possible)
- Training data creation and evaluation through Python plugin to get the ROM modes an hyper integration points and their corresponding weights
- 3D Models at the microscale with plane strain/axisymmetric elements at macro-scale
- parallelizability: Abaqus can be run in parallel (also over multiple computer nodes)
- binaries for Linux contained
- combination of element formulations:

micro ↓ macro →	plane stress	plane strain	axisymmetric	3D
plane strain	(✓) $\Sigma_{33} = 0$	✓	X	X
plane stress	✓	✓	X	X
3D	X	✓	✓	✓

1.2 Requirements

- Abaqus/Standard version 2017 or later
- Intel Fortran version 2017 or later (moved to Intel OneAPI HPC Toolkit) with correctly set environment variables
- Intel MKL version 2017 or later (in Intel OneAPI Base Toolkit) with correctly set environment variables
- only under Windows[2]: Microsoft Visual Studio Community or Enterprise (for the linker)

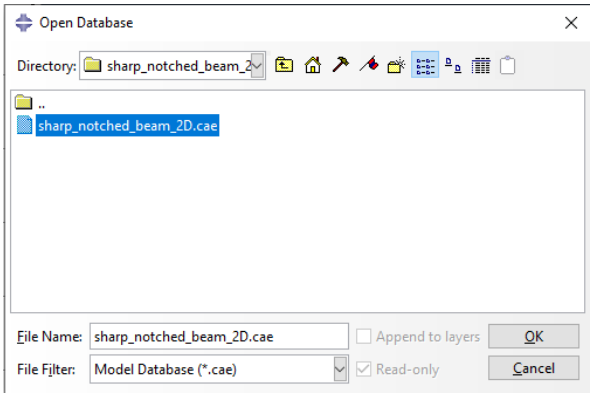
The program has been developed and tested extensively with Abaqus 2022 and Intel Fortran 2021 under Debian Linux 10 as well as under Windows 10/11 with Intel OneAPI 2021 and Microsoft Visual Studio Community 2019.

[1] In contrast to Abaqus, the fully integrated elements use the same quadrature rule for all terms, i.e., no selective reduced integration.
[2] see Intel Forum for detailed instructions on installation

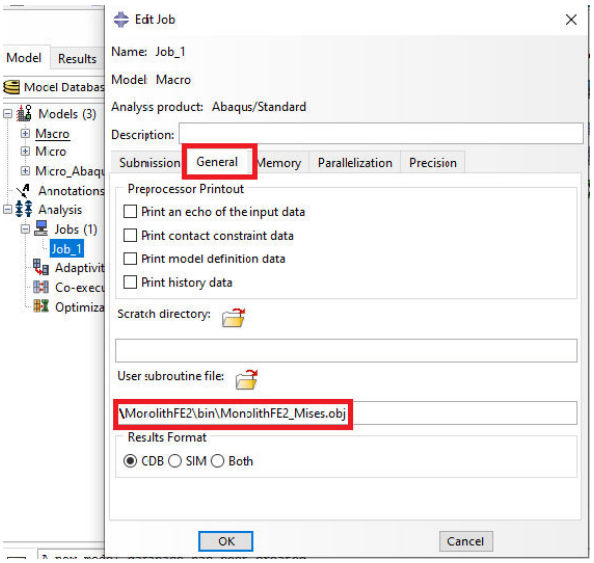
2 Quick Start Guide

2.1 Running Your First FE^2 Simulation

1. In Abaqus/CAE, open `sharp_notched_beam_2D.cae` from folder `examples/sharp_notched_beam_2D`

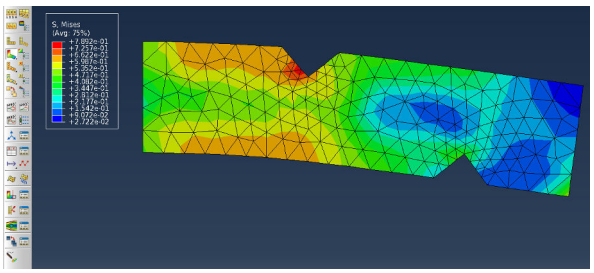


2. Set the "'User Subroutine File"' of `Job_1` to `YOURPATH\MonolithFE2\bin\MonolithFE2_Mises.obj` (Windows) or `YOURPATH/MonolithFE2/bin/MonolithFE2_Mises.o` (Linux).



3. Wait until the simulation is finished.

4. You are done. Open `Job_1.odb` and look through your macroscopic results. Congratulations, you have finished succesfully your first FE^2-simulation.



- If you want to visualize the microscale results, you need to install the CAE-plugin (cf. in section 7.2) and use proceed as described in section 5 using the micro-model named `Micro_Abaqus_Material`, or continue with the next sections to find out, how to set up your own examples and how to adapt MonolithFE^2 to your own needs..

If you want to use the command line instead of Abaqus/CAE, open it

1. Change to

   ```
   cd MonolithFE2/examples/sharp_notched_beam_2D
   ```

2. Run the simulation

   ```
   abqXXXX interactive job=Job_1 user=../../bin/MonolithFE2_Mises.obj cpus=4
   ```

   (wherein `XXXX` referes to your version number, e.g. `abq2023`)

3. Wait until it is finished and visualize the results as described above.

2.2 General proceeding

0. Install the Python plugins by copying the folder `abaqus_plugins` to either the current directory, home directory or Abaqus installation directory. In the next start of Abaqus/CAE these plugins are available.
   - only MS Windows: Copy the files `abaqus_6.env` and `win86_64.env` from the subfolder `src` to one of the working/home/installation directory to have the linker options correctly set (as described in detail in section 7.1).

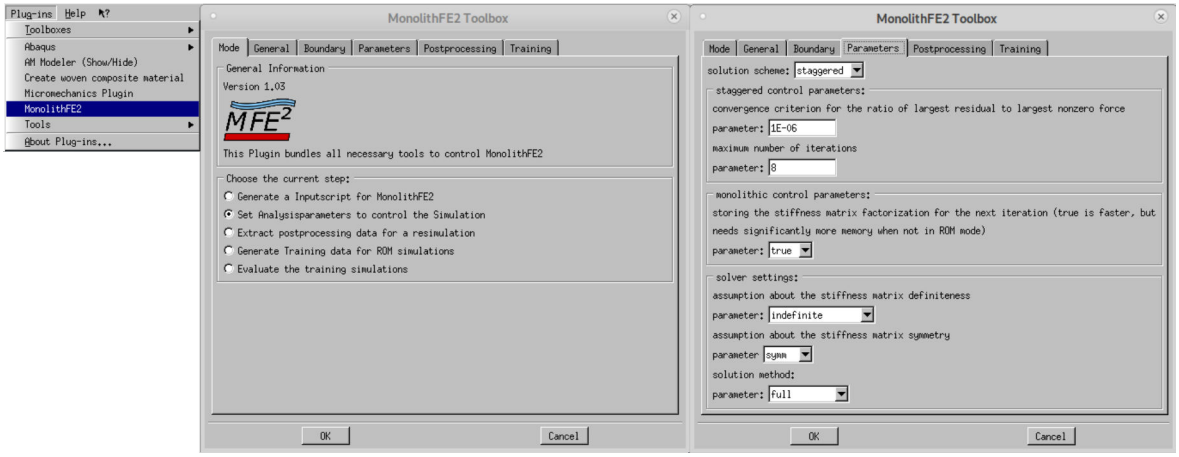
1. Generate micro-scale model(s) `*.FE#` (inputfile with mesh, material definition, boundary conditions etc.) as described in section 4.1, or take existing file from `examples/` (section 3) whereby `#` is the RVE-Number (a label, in one macro simulation the label must be named from 1,2 ... n-1,n when more then 1 RVE definition is used, otherwise always set the label to 1) and * is the job name of the macro problem (→ e.g. “beam_model.FE1”).

2. Generate a macro-scale model for Abaqus, e.g. with Abaqus/CAE, as usual and assign a user-defined material to the FE^2 regions. Therein, the only constant refers to the label of the RVE as described in step 1. The user material can also be specified directly in the `.inp` file:

   ```
   *User Material, constants=1
    %RVElabel
    *Depvar
    6
   ```

   Note: Specifying `*Depvar` isn't mandatory since the program only saves the deformation history (macroscopic strain resp. left stretch) in the solution dependent variables (SDV) for postprocessing, since in large deformation simulations the displacement gradient isn't accessible in the Abaqus Viewer. When the macro stretch is needed for postprocessing it's necessary to request SDVs in the "Field Output Request".

3. Analysis parameters can be set by the Abaqus plugin "MonolithFE2". This plugin creates a configuration file `FE2_Analysisparameters.cfg`, to be placed in the directory from which the Abaqus job is started. If no file is supplied, the default parameters will be used: monolithic algorithm without storing the factorized stiffness matrix, symmetric stiffnessmatrix.



4. Run FE^2 simulation by command

   `abaqus job=XXX user=PATHTODIR/bin/MonolithFE2_Mises.o cpus=%`n_{cpus}

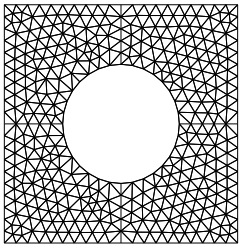
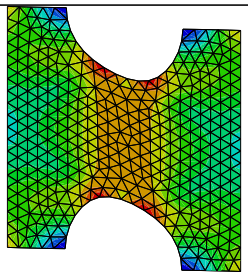
   under Linux or

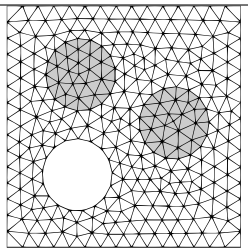
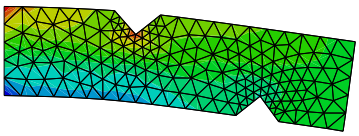
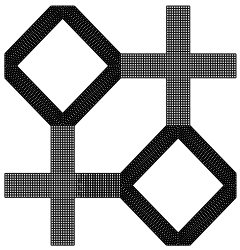
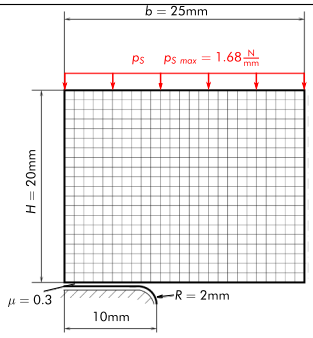
   `abaqus job=XXX user=PATHTODIR/bin/MonolithFE2_Mises.obj cpus=%`n_{cpus}

   under Windows or create Job in Abaqus/CAE and choose `MonolithFE2_Mises.o` or `MonolithFE2_Mises.obj`, respectively, as User subroutine file (tested with Abaqus versions 2017, 2018, 2020 and 2022). If a different material routine than the standard delivered Mises routine is used, first compile MonolithFE2 with it as described in section 7.1.

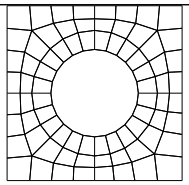
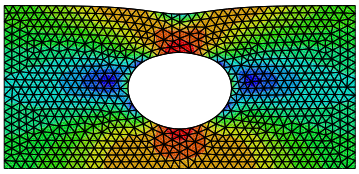
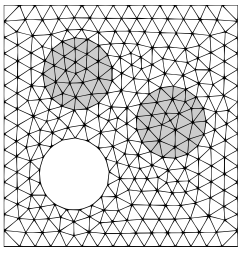
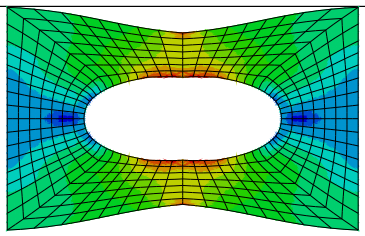
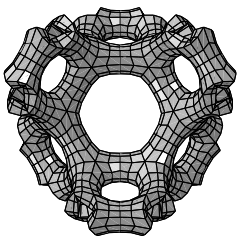
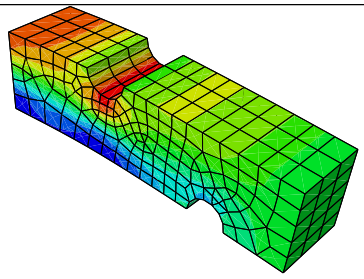
3 Examples

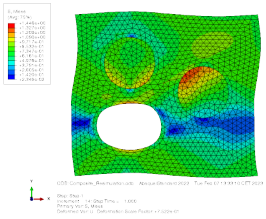
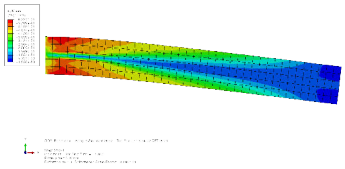
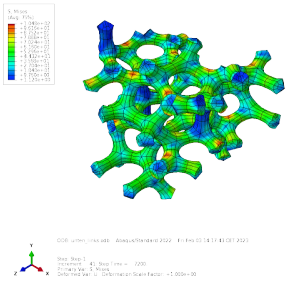
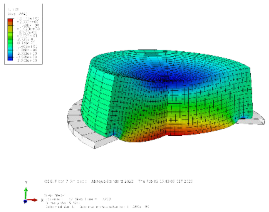
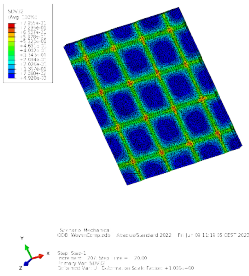
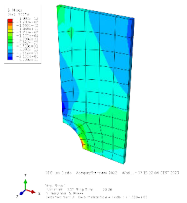
The following examples are shipped:

Name and reference	**micro**	**macro**
homogeneous [4]	homogenous microstructure, 4 rectangular elements	pure shear, pure bending
notched_plate_shear [1]		

sharp_notched_ beam_2D [1, 4]	 Miehe Koch Composite	
2D_foam_filter [5]		

The following examples can be made available on request:

Name and reference	**micro**	**macro**
bimaterial [4]	simple microstructure, 4 rectangular elements, two sets with different constitutive parameters,	pure shear, pure bending
plate_with_ hole_tension [4]		
quater_plate_ hole_tension	 from second example	 quarter model at large deformations, Miehe Koch composite
3D_beam_3D_ foam_power_law _hardening [1, 4]		

HyperROM/ Composite [2]		
HyperROM/ Filter (needs UMAT for Creeping) [2]		
HyperROM/ WovenComposite (needs UMAT for Polymer and Fiber Bundles) [2]		

Selected examples can be run automatically (in Linux) by the command

```
make verify
```

Further parameters like `NCPUS=...` or `ABACALL=...` can be passed, cf. section 7.1.

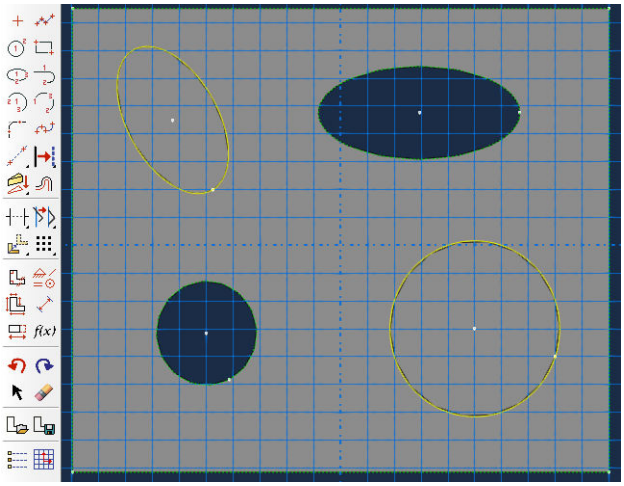
4 Creating Microscale Models

General requirements:

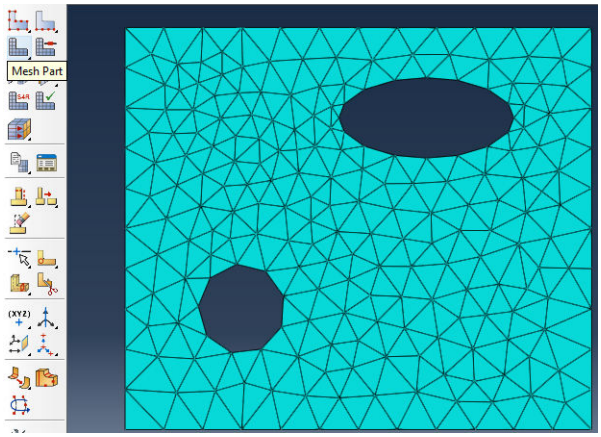
- same element type in complete micro-scale model
- congruent meshes at homologous parts of the boundary

4.1 Within Abaqus/CAE using Plug-In for 2D models

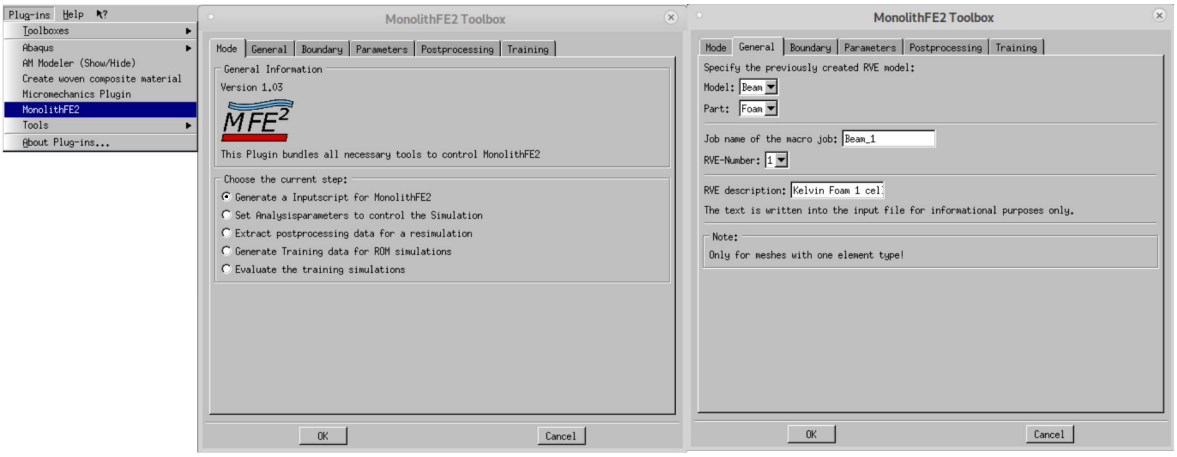
- Create a 2D Planar Part with rectangular outer shape. It can include any kind of pores and may be partioned.



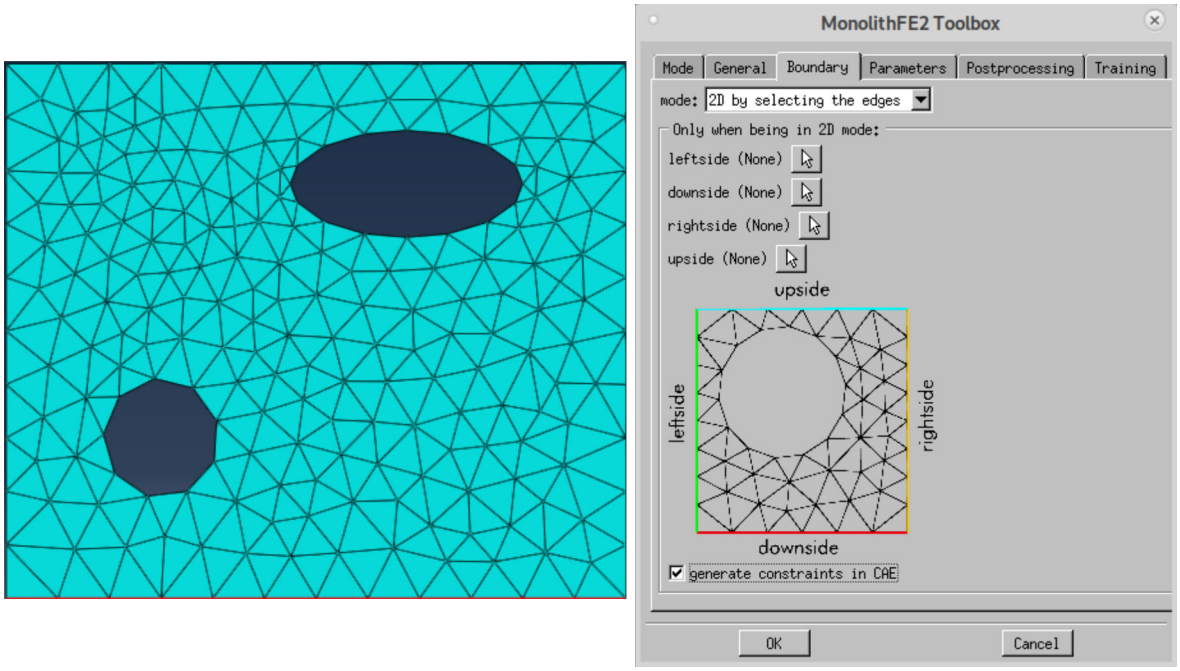
- Create one or more material definitions which sets the parameters of an user Material, because only user material (UMAT) routines can be used with MonoltihFE2! Put the Material into sections and assign them to all regions of the mesh.
- Put the part into an assembly and mesh the Part (not its Instance!) with either quadrilateral or triangular elements (Only meshes with exactly one element type are allowed!, can be linear/quadratic, full/reduced integration). Ensure that the nodes at opposite boundary faces are congruent.



- Open the Plugin „MonolithFE2“. Specify the created Model and Part. „Job-Name of Macro job“ is the Job name of the macro problem to be solved. A short description may be added.



- In the tab „Boundary“ Tab first click on the mouse courser button and then select the respective nodes of the edges of the created RVE (use rectangular box selection by holding the left mouse button pressed, to not miss a node!). Confirm each time with „Done“.



- If some micro problems are to be postprocessed after the FE^2 simulation, tick the box „generate constraints in CAE“.

- In one FE^2 Analysis, different RVE's can be used. When creating more then one RVE input file for a macro FE Analysis, make sure to choose different RVE numbers in the General Settings of the plugin. Clicking "OK" generates a file `*.FE#` in the current working directory, where `#` is the RVE-Number and `*` is the macro job name.

4.2 For 3D models with existing equations for periodic boundary conditions

4.2.1 Micromechanics Plugin

Following steps are necessary for the usage of the "Micromechanics Plugin" [6]:

1. Modelling the microstructure and generating a mesh. Note that the "Micromechanics Plugin" requires the Part to be in the Assembly and the Regions to have a Material assigned via a Section. Alternatively a model can be generated using the "Micromechanics Plugin" (FE-RVE →Library). Ensure that the nodes at opposite boundary faces are congruent. Note that this is not necessarily true for all examples of the "Micromechanics Plugin"!

2. Create a Job and give it exactly(!) the same name of the model created before.

3. Create the periodic boundary conditions in the "Micromechanics Plugin" under FE-RVE →Loading by choosing the correct model and job and confirm with "OK".

4. Create the Abaqus-Inputscript by right-clicking on the Job and then on "Write Input".

5. Now start the "Generate-FE^2-Inputfile" Plugin and fill out the "General" dialogue.

6. In the "Boundary" dialogue choose "Micromechanics Plugin", optionally click generate constraints in CAE for a resimulation of the RVE in the postprocessing, then confirm with "OK".

4.2.2 FoamGUI

When the periodic boundary conditions are created for 3D foams using the "FoamGUI" [7] code, then they are included in the Abaqus-Inputscript as:

```
*Equation
...
1
master-label|1|-1|slave-label|1|1|reference-node|1|-1
1
master-label|2|-1|slave-label|2|1|reference-node|2|-1
...
```

Following steps are necessary to create a Inputfile for MonolithFEsqr:

1. Import the Inputfile by right-clicking on Models and choosing "Import...". The equations are now imported as constraints and have the names 'Eqn-1'-'Eqn-%n'

2. Now open the "Generate-FE^2-Inputfile" Plugin and fill out the "General" dialogue.

3. In the "Boudary" dialogue choose "FoamGUI", optionally click generate constraints in CAE for a resimulation of the RVE in the postprocessing, then confirm with "OK".

5 Postprocessing

The postprocessing for the macro problem can be done as usual in Abaqus/Viewer. For the micro problems no conventional postprocessing is available at the moment. Instead the load history of selected macroscopic integration points can be extracted and then resimulated

directly in Abaqus. It is not sufficient to start Abaqus Viewer, opening Abaqus CAE is mandatory.

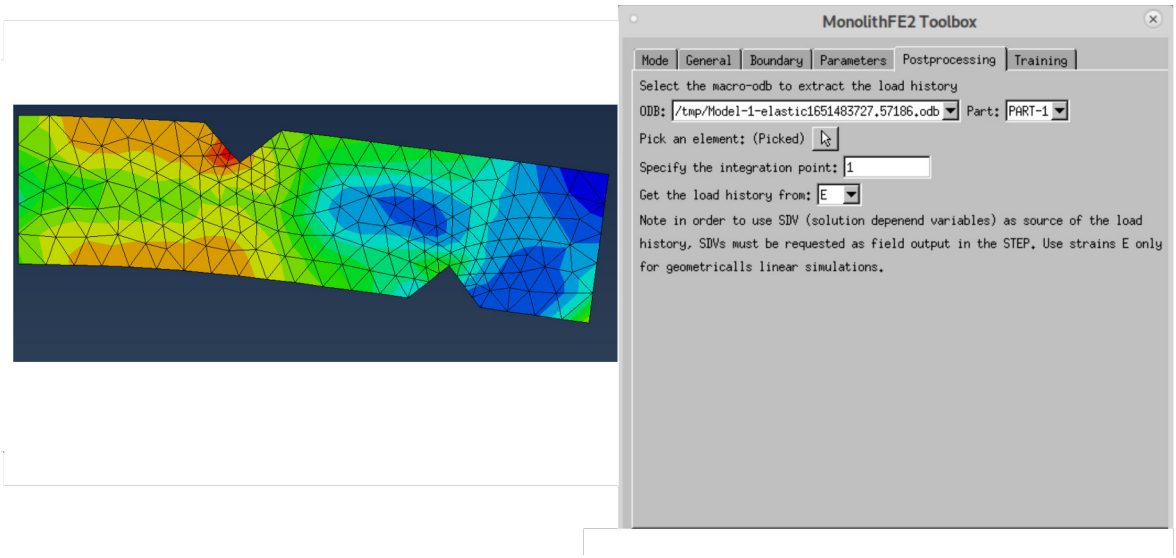
0. Before starting the FE^2 simulation select SDV as field output in the macro model. There the macro stretch tensor (resp. strain) will be saved. Without having the SDVs as field output no postprocessing can be done in large displacement analysis.

1. Open the created micro model. If the box „generate constraints in CAE“ in the „Generate FE2 microscale mesh“ Plugin as described in chapter 4.1 wasn't ticked in the first call, redo the process, now with ticked box. This creates constraints between nodes on opposite boundary's, boundary conditions, amplitudes and a step. This step can be skipped for all example problems!

2. Modify the steps settings as needed.

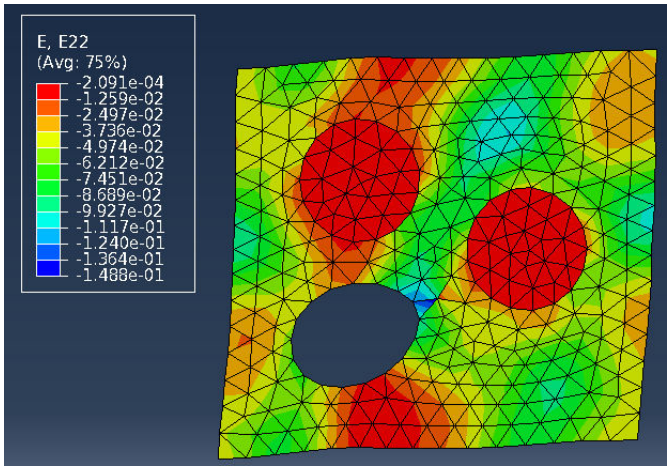
3. Now load the odb-file resulting from the FE^2 simulation.

4. Open the Plugin "MonolithFE2". In the General dialogue select the micro model. The therein mentioned micro model is the model from point 1 and should still be open. Clicking on "OK" creates amplitudes for the strain resp. displacement gradient in the micro CAE model.



5. Now create a job for the micro model and submit it.

6. Do the postprocessing as usually with the Abaqus/Viewer.

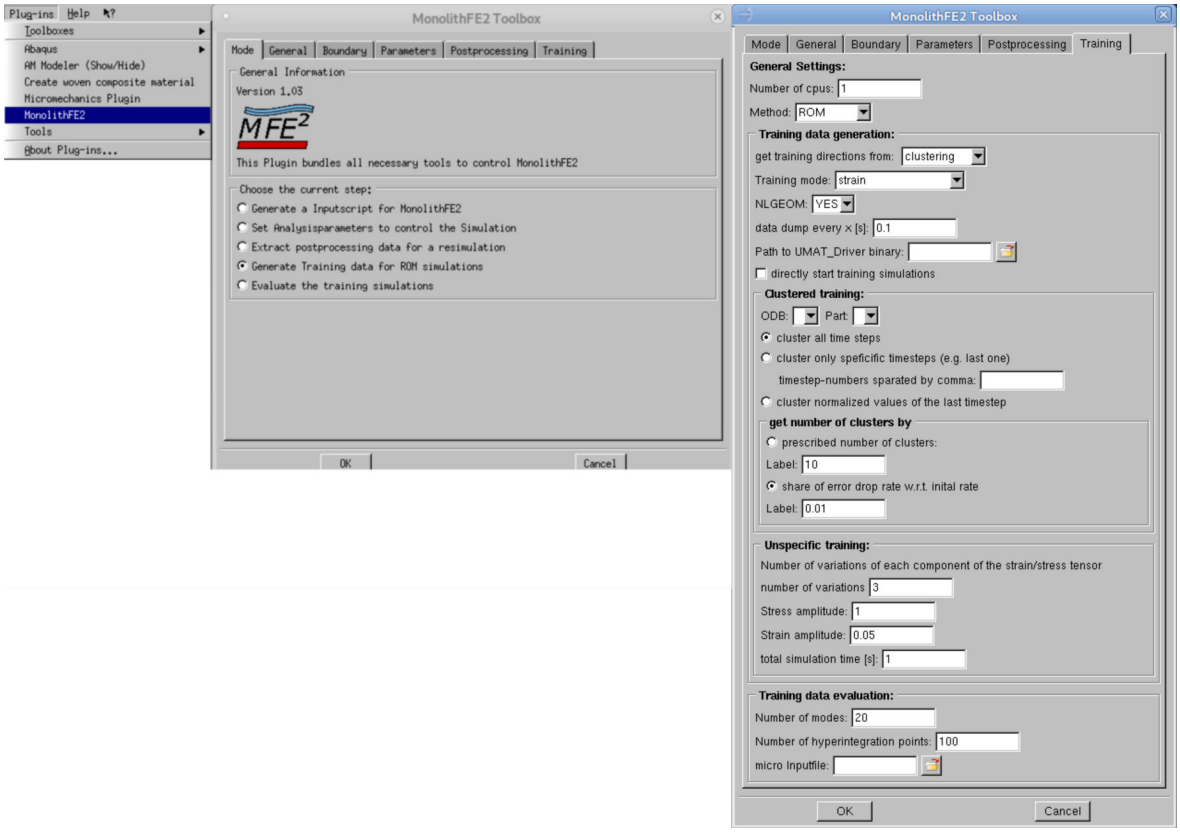


7. Repeat steps 4 to 6 for all macro integration points to be postprocessed.

6 Hyper ROM Method

The basic idea of the ROM method is, that the vector of nodal displacements can be approximated by a linear combination of a low number of modes. The integration can then also be reduced by a lower number of integration points and their corresponding weights. Necessary steps in MonolithFE2:

1. Generate training data (snapshots of nodal displacements and force vector at integration points).
    - Open then ”MonolithFE2“ Plugin, select the micro model and macro job name in the General dialogue and fill out the upper dialogue of ”Training“. This creates an UMAT_Driver inputfile defining the training directions with the name of the macro job chosen. It also creates a configuration file ”Analysisparameters.cfg“ which sets the necessary parameters.



- Thereby number of variations means the how often every entry of the macroscopic strain (resp. stress) tensor is varied.
- The output of training data can be requested at certain time steps, whereby the plugin assumes equidistant dump steps (which is no requirement). The time steps could be set differently in the file "Analysisparameters.cfg" if needed.

  `*Data_dump, N=`n_{steps}

  $t_1, t_2, t_3 \ \ldots\ , t_{n_{\text{steps}}}$
- Run the simulation by ticking the box "directly start training simulations". Otherwise start UMAT_Driver

  `./UMAT_Driver cpus=`n_{cpus} `job=Jobname`

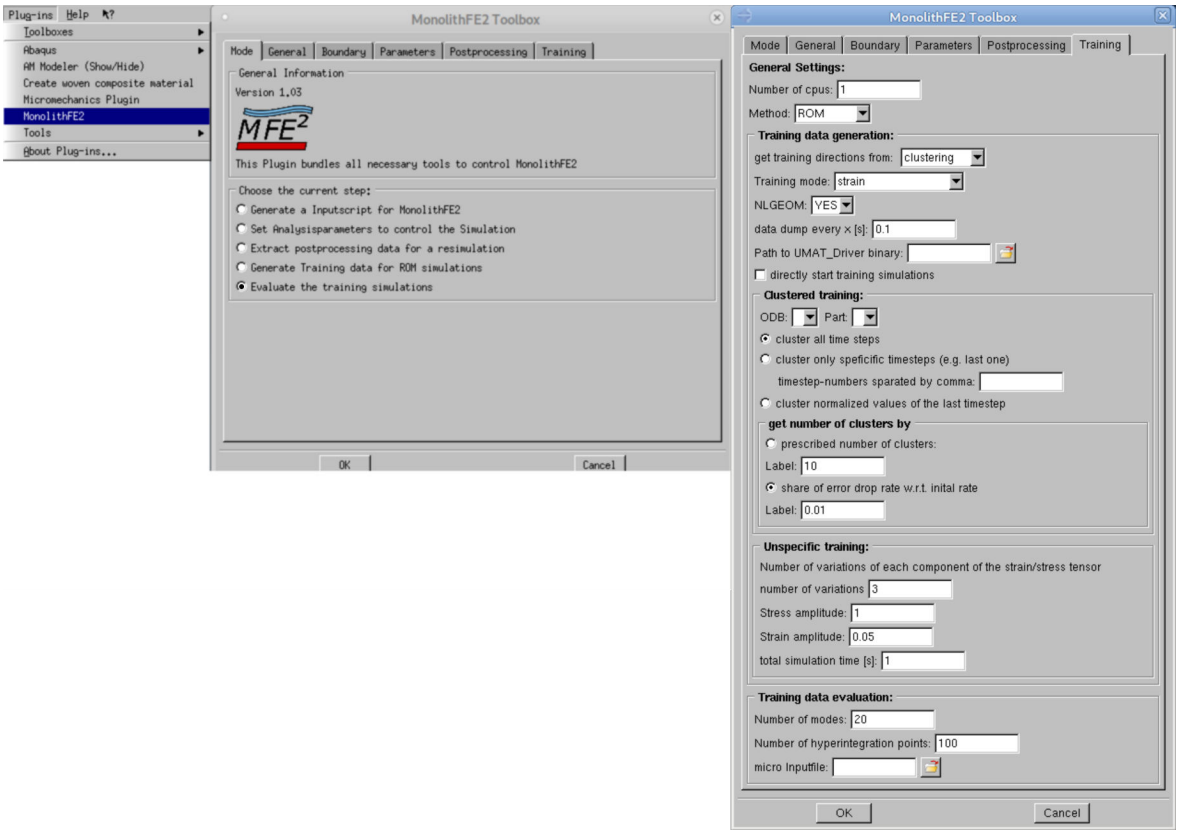
  Thereby the compiled program `UMAT_Driver.o(bj)` must be in the same folder. Note that the `Jobname.FE#` file must have the RVE label 1 (`#=1`)
- The simulations can be parallelized by specifying the number of cpus.
- The simulation outputs the displacement values in files with the name training-data-u-*.txt and the force values in training-data-f-*.txt whereby * is an ID number.
- Clustered training and unspecific training are possible
  - In unspecific training, the entries of the macroscopic stress/strain tensor will be varied
  - For the clustered training, at first a simulation with some replacement material must be performed. Then the odb has to be opened in Abaqus and the clustering parameters have to be set. Then by using the data of the odb the training directions will be created.

2. Evaluate data to get the displacement modes.
3. Evaluate the data to get the integration points and their weights.
4. Write the results to the input file of the trained RVE.
   - All steps can be done using the Python Plugin graphically or running the data evaluation script from a command line using the `Hyperintegration.py` script with the therein specified input parameters.
   - As a pre step the FORTRAN parts of the code must be compiled into a shared object file. Therefore run

     `make` (Linux) or

     `compile` (Windows)

     in the Plugin folder.
   - In Abaqus CAE:
     - Open the "MonolithFE2" Plugin, write the macro job name in the General dialogue and fill out the under dialogue of "Training".



     - Note that the micro Inputfile which has to be chosen (and into which the resulting data is written) has to be in the folder containing the training data
     - The data evaluation can be parallelized, whereby the reading of files and the singular value decomposition run both in parallel.

  - As a result the modes and integration point numbers and weights are written into the inputfile `#.FE1`.
- Note that the SVD of the various data is stored to text files named `LSV_...txt` (LSV=Left Singular Vectors). Note that when the data Evaluation is restarted, the data is read from these files.
- When using the script directly from the command line:

```
python pluginpath/Hyperreduction.py path_to_inputfile=...  ...  ncpus=...
```

  - allowed values:
    - `path_to_inputfile` absolute Path to inputfile
    - `Method='ROM' or 'hyperROM'`
    - `store_SVD= 1 or 0`
    - `from_stored_SVD= 1 or 0`
    - `n_modes=` from $1...n_{\text{modes}}$
    - `NGP=` from $1...n_{\text{GP}}$
    - `ncpus=` from $1...n_{\text{cpus}}$

5. Do actual simulations by setting a reduction mode in the "Analysisparameters.cfg" control file.
   - Either by using the Python Plugin or by setting the following option:

```
*Solving_Process
```

     - `full`: no reduction
     - `reduced`: only ROM projection, but full integration
     - `hyperreduced`: ROM projection and hyper integration

7 Source Code and Compilation

7.1 Compilation

The code is compiled under Linux by command

```
make [OPTION=value]
```

or under Windows[3] by

```
[SET OPTION=value]
compile
```

The possible options are listed in Table 3. These commands create the object files `MonolithFE2_XXX.o` resp. `MonolithFE2_XXX.obj` (where `XXX` corresponds to the chosen material routine name), which can be used as described in section 2.

Required additional packages (by default in same folder as folder of MonolithFE^2):

- UELlib [3]

[3] If the environment variables are not set correctly in the default shell or in "Abaqus Command" shell, the command should be run in "Intel oneAPI command prompt ... for Visual Studio" from start button of Windows.

Tab. 3: Options for compilation

Option	default value	remark
ABACALL	abaqus	command for calling Abaqus (e.g. abq2020)
MATERIAL	Mises	material routine for microscale

- UEL-large-deformation
- UMAT_Driver (only needed for generating Training data for ROM Method)
  - The UMAT_Driver program has to be compiled as described in the documentation of the respective project and then copied to the corresponding training folder, when used as driver routine for generating training data.

The location of these packages is given in the `Makefile` and `compile.bat` and can be set by `make UELlibDIR=...` etc or modifying `compile.bat`. Alternatively, the files can be made accessible either directly in folder `src/` (linked or copied) or the respective directories are added to the include path with option `-I` of line `compile_fortran` in the environment file (`lnx86_64.env`/`win86_64.env` loaded from `abaqus_v6.env`). With the option `MATERIAL=directory` a user defined material behavior can be specified. The `directory` coincides with the name of the folder containing the necessary source files to be placed in the `materialroutines` folder of uel-large-deformation. For further details look into chapter 8. When `MATERIAL` is omitted the standard value is `Mises`.

Further requirements:

- Linux: Compiler options in the environment file `lnx86_64.env` loaded from `abaqus_v6.env` (as set in present package) in line `compile_fortran`
  - `-mkl=cluster`: include MKL
  - `-heap-arrays`: Create large arrays in heap instead of stack to avoid stack overflow
  - `-nostandard-realloc-lhs`: Turned out to be necessary to avoid certain runtime errors.
  - The Fortran preprocessor `FPP` must be switched off by deleting the option `-fpp`. The reason is that the present code is written in free-form Fortran whereas other included files may be written in fixed form. The present implementation employs the compiler directives `!DEC$ FREEFORM` and `!DEC$ NOFREEFORM` to switch between both settings. However, this compiler directive is incompatible with `FPP`.
- Windows: Changes in the environment file `win86_64.env` loaded from `abaqus_v6.env` (and as shipped with present package)
  - additional compiler options to line `compile_fortran`: `\heap-arrays`, `\nostandard-realloc-lhs`, `\names:lowercase` (problem of Abaqus2020 with OneAPI), `\Qmkl`, `\I"%MKLROOT%\include"`
  - additional linker option to `link_sl`: `mkl_rt.lib` in order to include MKL library in linking
- The program adapts free-form versions of the required interface definitions from `SMAAspUser*.hdr`. It may be necessary to check whether these definitions have been changed in other version of Abaqus.
- The UEL has to provide an additional subroutine `GET_n_STATEV_elem`, which returns the number of state variables which have to be reserved by MonolithFE^{2}for each element (`NSVARS` in the head of UEL), cf. attached file `UEL.f`.

7.2 Plugin for Abaqus/CAE

All necessary FORTRAN routines in the `abaqus_plugin` folder have to be precompiled in this folder by:

```
make
```

in Linux or

```
compile
```

in Windows. This creates a shared object file needed for the Abaqus Python Plugin MonolithFE2.

Subsequently, the folder `abaqus_plugins` has to be copied either to the current directory, home directory or Abaqus installation directory. In the next start of Abaqus/CAE the Plugin is available.

7.3 Source Code

The structure of the components of MonolithFE^2 is illustrated in Fig. 1. It may be added that the file `*.FE*`, created by the Python plug-in, represents the node-to-element connectivity in form of lists and the connection between the entries of the stiffness matrices of the micro-UELs and the global stiffness matrix in Compressed Sparse Row (CSR) for the solver. The following table explains selected Fortran routines.

filename	routine	description
UMATmacro.f	UMAT	actual macro-UMAT interface to be called from Abaqus at each macro-GP; includes all needed source files; gets all pointers to the needed data; calls the main program (staggered/-monolithic) to get macro STRESS and macro DDSDDE; extrapolates micro displacements in staggered case
FE2MODULE.f	main_program_staggered	actual main program for the staggered algorithm, which calls all needed routines (assemble, solve...) in the right order
FE2MODULE.f	main_program_monolithic	main program for monolithic algorithm
FE2MODULE.f	enforce_constraint	enforces constraints for the periodic boundary conditions
FE2MODULE.f	update_displacements	when the increment of the displacements in the Newton-Rhapson algorithm was computed, this routine updates the displacements
FE2MODULE.f	assemble	this routine calls the micro element routine for each micro element and assembles the stiffness matrices, the residual and the macro STRESS
FE2MODULE.f	static_condensation	this routine computes the macro stiffness matrix DDSDDE and macro stress STRESS
Solver.f	initialize	Create memory for solving sparse systems of equations with the MKL PARDISO solver and get a pointer to that memory
Solver.f	factor	factor the matrix
Solver.f	solve	solve system for given right hand sides
Solver.f	finish	free solver storage
Solver.f	get_permutation_matrix	get a permutation vector for a given matrix structure
Solver.f	interpret_error	interpret a possible PARDISO error
SMAAspUser Arrays Fortran.f	SMA*Array*CreateFortan* SMA*Array*AccessFortan* SMA*Array*DeleteFortan*	Abaqus provides routines called "allocatable arrays" to create threadsafe data and access them via pointers. These routines output cray pointers, which aren't FORTRAN Standard and can't be used in user derived types. Therefore these routines act as a interface to the Abaqus allocatable arrays and output FORTRAN Standard conform pointers.
manage_ data.f	UEXTERNALDB	Abaqus interface UEXTERNALDB is called from Abaqus at the start/end of a analysis and start/end of a timestep. It's used to manage the mesh and state variable data at the correct point of the analysis
manage_ data.f	readdata	at the analysis begin the micro mesh & analysis parameters are read and stored

manage_ data.f	manage_STATEV_data	at the end of a time step the state variables of all macro GPs are written to: t→t-1, t+1→t; at the start of a time step the data is initialized; at the end of the analysis the allocated disk space is freed
type_macro_ GP_DATA.f	allocate_data get_pointer deallocate_data	in this module a user derived FORTRAN type is declared which encapsulates all the macro GP state date (micro displacements, state variables of the micro GPs, ...)
type_meshp arameters.f	read_data get_pointer deallocate_data	In this module a user derived FORTRAN type is declared which encapsulates all the data for defining the microscopic mesh (coordinates, element to node connection, etc.) plus the program-flow/convergence parameters. The create* routine reads the data from disk, the get_pointer* routine accesses the data and the deallocate* routine frees the allocated memory
type_analy sispara meters.f	read_data get_pointer deallocate_data	In this module a user derived FORTRAN type is declared which encapsulates all the data for defining analysis parameters, for example if the monolithic or staggered algorithm shall be used, if symmetric matrix storage shall be used etc.
utility_ transform_ stress_ stiffness.f	get_abaqus_stress_stiffness	return STRESS and DDSDDE in the format Abaqus expects it.
utility_ transform_ stress_ stiffness.f	trans_stress_stiffness	The (symmetric) Biot stress is transformed to Cauchy stress (also the material tangent: nominal tangent → DDSDDE).
utility_ transform_ stress_ stiffness.f	polar_decomposition	The macro deformation gradient is split in the rotation and right stretch tensor

8 Defining a different Material behavior

Defining the material behavior is similar to Abaqus. A material routine has to be written using the UMAT interface as declared in the Abaqus Manual [9]. The UMAT has to be named precisely "UMAT.f". The routine has to be written for 3D, plane strain or axisymmetric deformation states. The plane stress case is handled internally. The file UMAT.f can also contain various `SUBROUTINES` or `FUNTIONS` but no `MODULES`. Preferred is FORTRAN 90 free form code. When the UMAT is written in FORTRAN 77 fixed form style compiler directives have to be included before and after the subroutine as shown below:

```
!DEC$ NOFREEFORM

SUBROUTINE UMAT(...)

...user coding to define UMAT...

...all variables are defined just as in the Abaqus/Standard UMAT interface
and have the same type and dimension...

END SUBROUTINE UMAT

!DEC$ FREEFORM
```

The number of state variables required by the UMAT has to be specified in the Routine GetNSTATV. Name this routine precisely "GetNSTATV.f".

```
FUNCTION GetNSTATV(NTENS,NDI,PROPS)

INTEGER(KIND=AbqIK)::  GetNSTATV
INTEGER(KIND=AbqIK), INTENT(IN)::  NTENS,NDI
REAL(KIND=AbqRK),INTENT(IN)::  PROPS(:)

...user coding to define GetNSTATV. If NTENS==3 (plane stress) add +1 to that
value since the plane stress algorithm needs one internal state variable too...

END FUNCTION
```

Write a routine to check if the material parameters are reasonable and name it precisely "CheckMaterialParameters.f". Use the STDB_ABQERR function to stop the simulation and report errors to the inputfile [9]. If you do not want to check the parameters just return using the FORTRAN statement `RETURN`.

```
SUBROUTINE CheckMaterialParameters(PROPS)

REAL(KIND=AbqRK),INTENT(IN)::  PROPS(:)
...
```

```
!  Check the properties

if ( ...  )  then
CALL STDB_ABQERR(-3, "CheckMaterialParameters reports:  ...  ", 0, 0.0, " ")
else if ( ...  )  then
...
end if

END SUBROUTINE CheckMaterialParameters
```

It's also possible to use modules, therefore include all of them into a file and name it precisely "UMATUtilityModules.f". If modules are not needed, just leave the file blank.

Put all 4 files into a directory with a reasonable name inside of the `materialroutines` folder of uel-large-deformation. When the program is compiled as described in section 7.1 specify `MATERIAL=` with the name of the just created directory.

9 Limitations, Problems and Future Developments

9.1 Limitations

- single element type per micro mesh definition
- single micro-material routine (UMAT) per simulation (`PROPS` may be used to switch between different encapsulated material routines); MonolithFE^2 has been successfully tested with the UMAT_creep material routine (requires to make the UMAT accessible for compilation, as shown in chapter 8)
- Python plug-in „MonolithFE2“ supports 3D micromeshes only if constraints already generated in Abaqus ”equations“ (Micromechanics Plugin or FoamGUI)
- purely mechanical problems (no "multiphysics")
- distributed-memory computations ("multi-node") may not work with the monolithic-stored stiffness matrix factorization algorithm in some cases (first tests yield no problem yet)

9.2 Known Issues

- sometimes convergence problems, specifically:
  - large deformation analysis
  - plane stress (especially in large deformation analysis)
- At the moment Abaqus Plugins only work if English System language is installed and set (tested under Linux, include in the .bashrc "export LANG=en_US.utf8" and "export LANGUAGE=en_US").

9.3 Software environment

- The file `ABA_PARAM.INC` is not found by Abaqus by default under Linux, but put "artificially" to folder `src/`. Its contained must be checked under different OS and computer architectures.
- Integer kind specifications in `SMAAspUserArraysFortran.f` are fixed (adopted from `SMAAspUserSubroutines.hdr` of Abaqus) and may be specific to OS and computer architecture as well.

10 Version history

date	description
2020-07-29	final version of diploma thesis of N. Lange [4]
2020-12-17	Material in rate formulation; large deformations; all elements of UELlib added; in Plugin "Generate FE^2 Inputfile" plane strain/plane stress/3D stress selectable; general applicable plane stress algorithm in `UXMATMises.f` added; stiffnessmatrix factorization indefinite/positiv definite selectable
2021-02-03	switched from MKL DSS solver to MKL PARDISO solver due to an existing memory leak when using the DSS solver, added a postprocessing for micro problems
2021-04-20	version 1.0 published
2021-05-06	v1.01 with binaries for Windows
2021-07-20	v1.02, Inputfile adopts *Keyword style of Abaqus, number of node dof's named generally to be compatible with generalized continua, material routines can now be integrated more easily without changing the UEL.f file
2021-07-23	v1.02a, bug removed; added: automatic verification run auf examples within Makefile
2022-05-05	v2.0, major version change, all plugins combined to a single one; micro material definition now in the *.FE# inputfile; hyper ROM reduction implemented, in a training step data is collected, in a evaluation step the ROM modes and integration point weights are calculated, written to the *.FE# inputfile, then it can be used in actual simulations; in finite deformation now a polar decomposition is done and only the stretch is impressed onto the RVE; now no more limitation to 5 RVE definitions in a simulation
2023-07-20	Memory Leak of MKL fixed, now the solver memory ca be released in the full mode

References

[1] N. Lange, G. Hütter, B. Kiefer: An efficient monolithic solution scheme for FE^2 problems, Computer Methods in Applied Mechanics and Engineering 382 (2021), 113886.

[2] N. Lange, G. Hütter, B. Kiefer: A monolithic hyper ROM FE^2 method with clustered training at finite deformations , Computer Methods in Applied Mechanics and Engineering 418 (2024), 116522.

[3] G. Hütter, S. Roth, R. Skrypnyk: UELlib – A library for user-defined elements in Abaqus, Technical Report, TU Bergakademie Freiberg, Institute of Mechanics and Fluid Dynamics.

[4] N. Lange: Implementation of a monolithic FE^2 program (in German), diploma thesis, TU Bergakademie Freiberg, 2020.

[5] G. Hütter, C. Settgast, N. Lange, M. Abendroth, B. Kiefer: A hybrid approach for the multi-scale simulation of irreversible material behavior incorporating neural networks, Proc. Appl. Math. Mech. 20 (2020), e202000248.

[6] R. McLendon: Micromechanics Plugin for Abaqus, `https://www.linkedin.com/pulse/micromechanics-plugin-abaqus-ross-mclendon`, 2017.

[7] M. Abendroth, E. Werzner, C. Settgast, S. Ray: An Approach Toward Numerical Investigation of the Mechanical Behavior of Ceramic Foams during Metal Melt Filtration Processes, Adv. Eng. Mater. 19 (2017), 1700080. DOI:10.1002/adem.201700080

[8] J. Hernandez, M. Caicedo-Silva, A. Ferrer Ferre: Dimensional hyper-reduction of nonlinear finite element models via empirical cubature, Computer Methods in Applied Mechanics and Engineering 313 (2016), DOI:10.1016/j.cma.2016.10.022

[9] ABAQUS/Standard User's Manual, Version 6.9, Michael Smith, Dassault Systèmes Simulia Corp, 2009

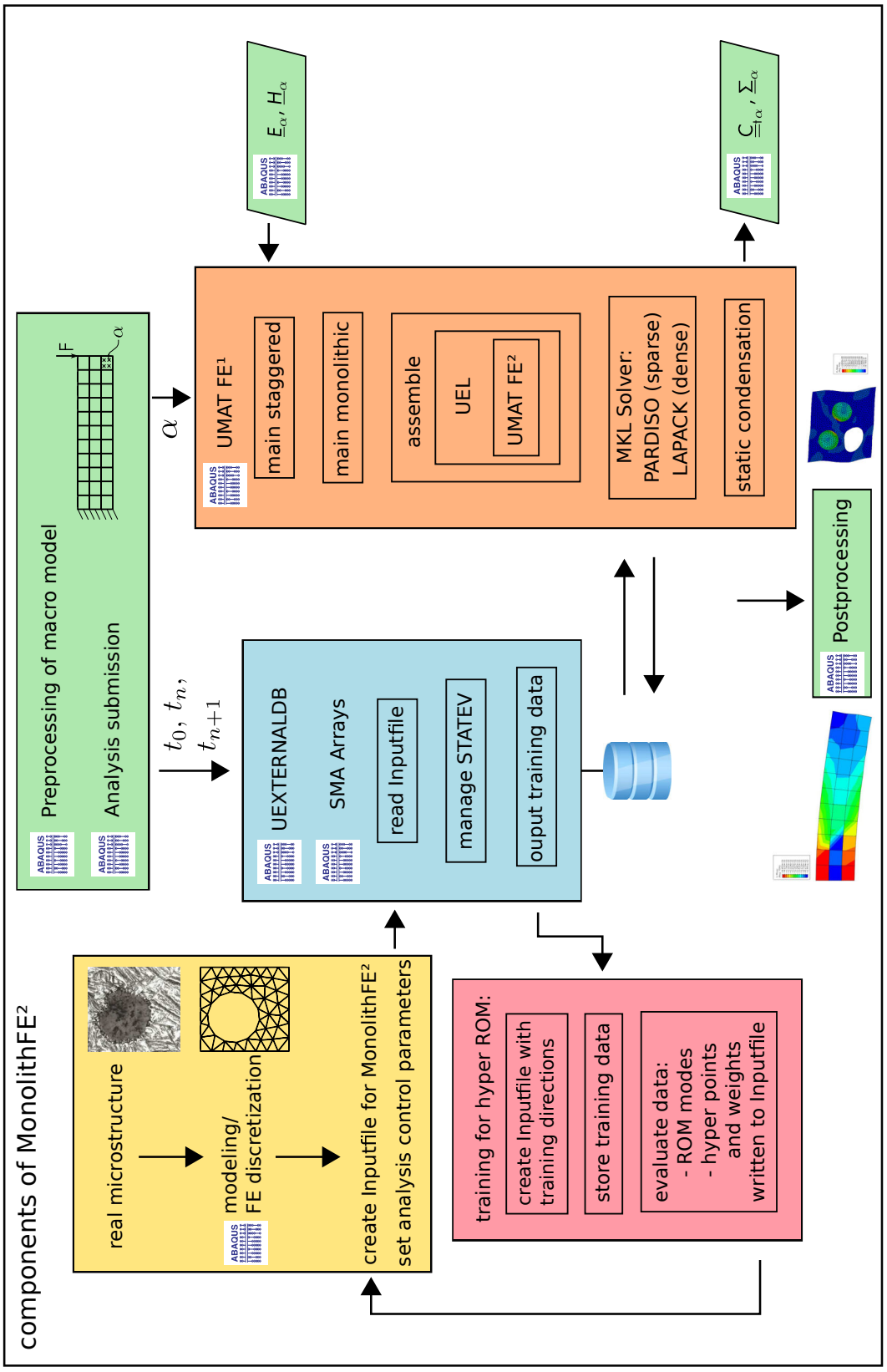


Fig. 1: Structure of MonolithFE^2