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

MonolithFE^2 – A monolithic FE^2 implementation for Abaqus
Version 3.0

Nils Lange, Geralf Hütter, Björn Kiefer

April 12, 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, but other UEL implementations can be simply compiled into 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 micro-scale can be done in Abaqus/CAE with aid of a Python plug-in, which creates an Abaqus like keyword style inputfile, which could also be created directly following the rules given in this documentary. 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 element wise 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
- general linear constraints at micro-scale, periodic boundary conditions can be set up using the provided Abaqus Plugin
- 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
  - reordering the equations to minimize the wavefront only once at the beginning of a simulation
- 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, hyper ROM training)
- 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 elements and their corresponding element multiplication factors
- 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 $\downarrow$ macro $\rightarrow$	plane stress	plane strain	axisymmetric	3D
plane strain	(✓) $\Sigma_{33} = 0$	(✓) $\Sigma_{33} = 0$	X	X
plane stress	✓	✓	X	X
3D	X	✓	✓	✓

- For further developments: In principle suited for generalized continua FE^2 simulations $\rightarrow$ requires a suitable UEL on the micro- and/or macroscale, generation of the necessary constraints and a "generalized UMAT" on the macroscale which interprets the in- and outputs of the MonolithFE^2 kernel differently than in the moment for the mechanical case

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.

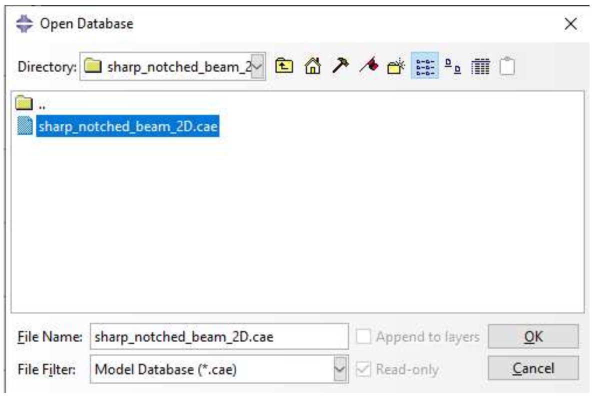
[1]In contrast to Abaqus, the fully integrated elements use the same quadrature rule for all terms, i.e., no selective reduced integration.

[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`


Open Database

Directory: sharp_notched_beam_2D

..

sharp_notched_beam_2D.cae

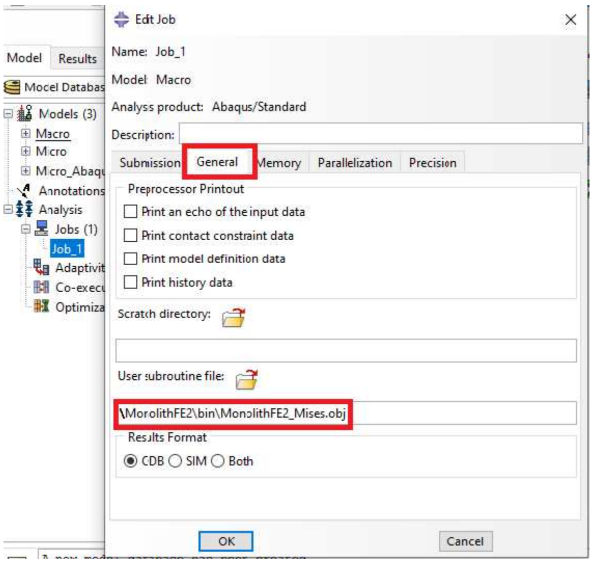
File Name: sharp_notched_beam_2D.cae

File Filter: Model Database (*.cae)

☐ Append to layers

☑ Read-only

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).


Edit Job

Name: Job_1

Model: Macro

Analysis product: Abaqus/Standard

Description:

Submission | General | Memory | Parallelization | Precision

Preprocessor Printout

☐ Print an echo of the input data

☐ Print contact constraint data

☐ Print model definition data

☐ Print history data

Scratch directory:

User subroutine file:

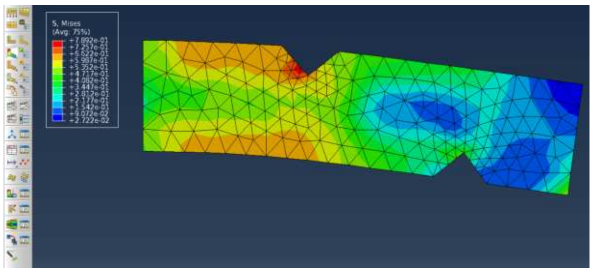
\MonolithFE2\bin\MonolithFE2_Mises.obj

Results Format

● CDB ○ SIM ○ Both

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).

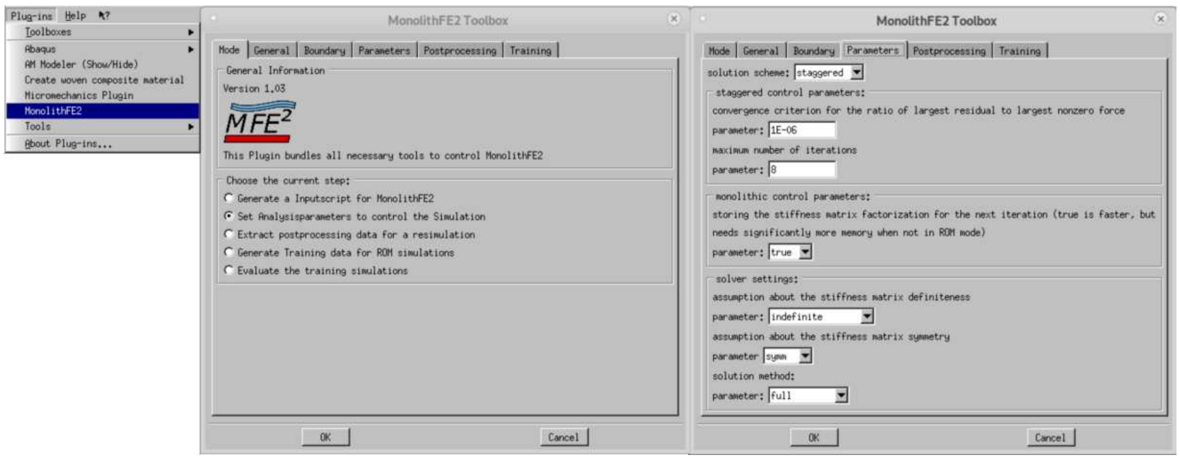
1. Generate micro-scale model(s) `*.FE#` (inputfile with mesh, material definition, boundary conditions etc.) as described in section 4.1, or take existing file from `examples/` (section 3) whereby `#` is the RVE-Number (a label, in one macro simulation the label must be named from 1,2 ... n-1,n when 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.



General Information
Version 1.03
This Plugin bundles all necessary tools to control MonolithFE2

Choose the current step:
( ) Generate a Inputscript for MonolithFE2
[✓ Set Analysisparameters to control the Simulation]
( ) Extract postprocessing data for a resimulation
( ) Generate Training data for ROM simulations
( ) Evaluate the training simulations

solution scheme: staggered

staggered control parameters:
convergence criterion for the ratio of largest residual to largest nonzero force
parameter: 1E-06
maximum number of iterations
parameter: 8

monolithic control parameters:
storing the stiffness matrix factorization for the next iteration (true is faster, but needs significantly more memory when not in ROM mode)
parameter: true

solver settings:
assumption about the stiffness matrix definiteness
parameter: indefinite
assumption about the stiffness matrix symmetry
parameter symm
solution method:
parameter: full

4. Run FE^2 simulation by command

   ```
   abaqus job=XXX user=PATHTODIR/bin/MonolithFE2_Mises.o cpus=%n_cpus
   ```

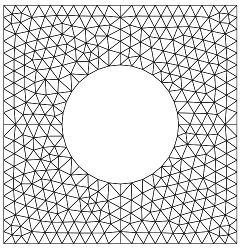
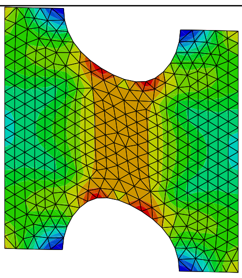
   under Linux or

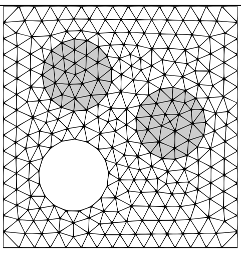
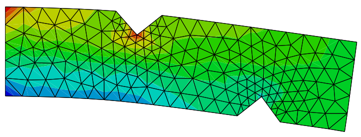
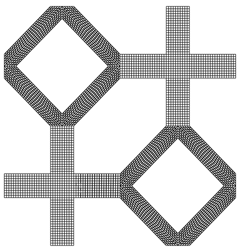
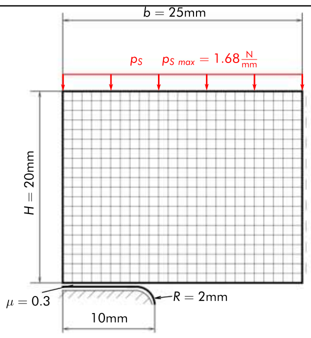
   ```
   abaqus job=XXX user=PATHTODIR/bin/MonolithFE2_Mises.obj cpus=%n_cpus
   ```

   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.

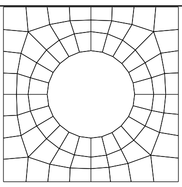
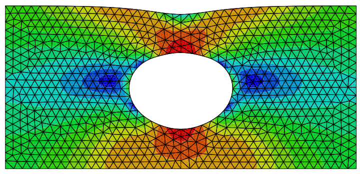
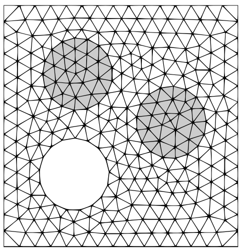
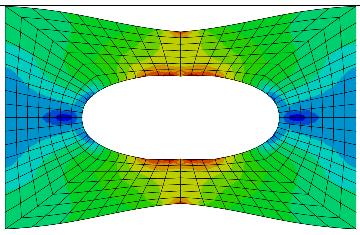
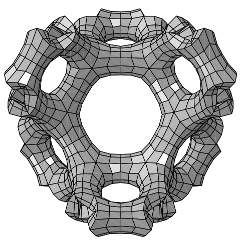
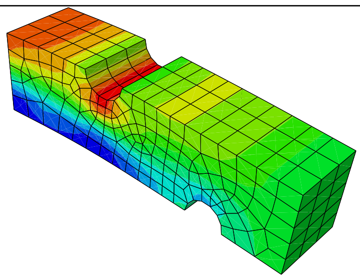
3 Examples

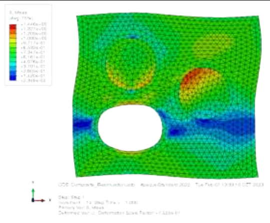
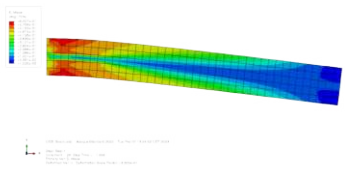
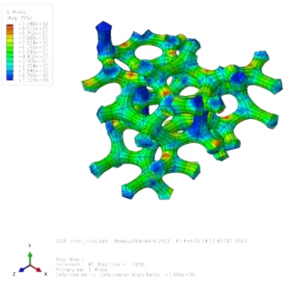
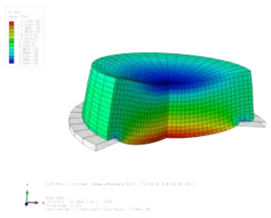
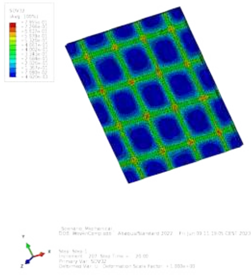
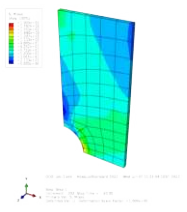
The following examples are shipped:

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]		 $b = 25$mm p_S $p_{S\,max} = 1.68\,\frac{N}{mm}$ $H = 20$mm $\mu = 0.3$ $R = 2$mm 10mm

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.

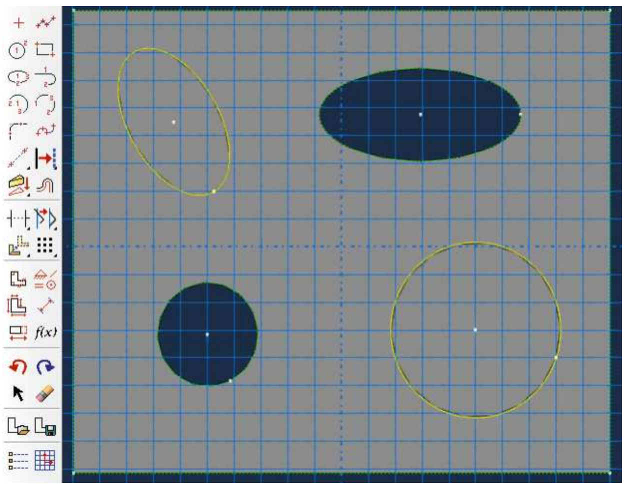
4 Creating Microscale Models

General requirements:

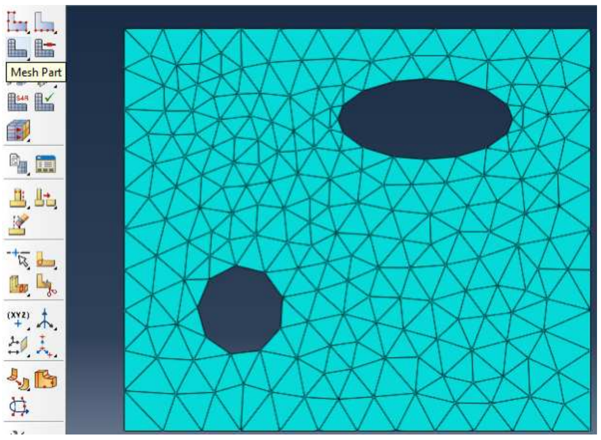
- using elements (not restricted to a certain element type, multiple element types per part allowed) that are part of the shipped UEL interface
- congruent meshes at homologous parts of the boundary

4.1 Within Abaqus/CAE using Plug-In

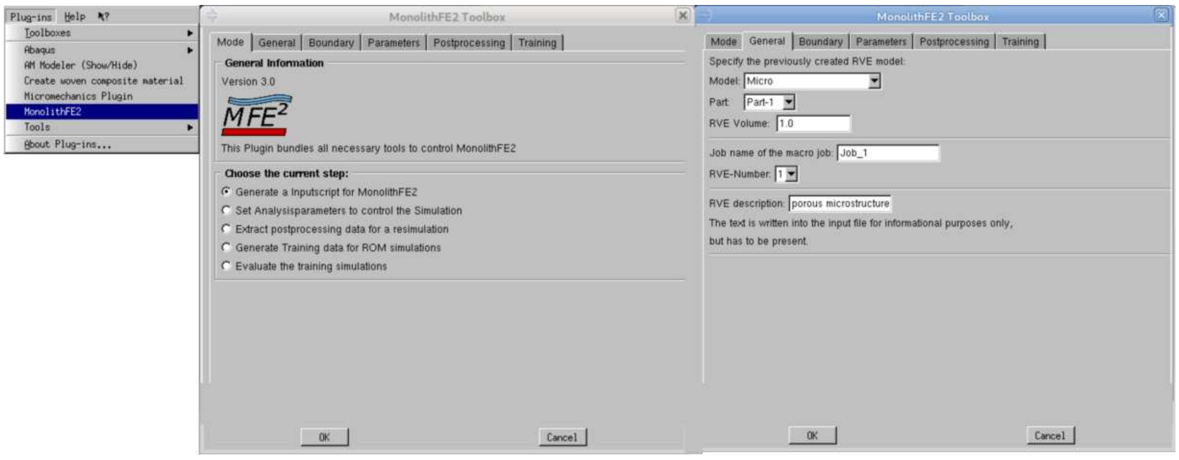
- Create a 2D Shell or 3D solid part, that can be periodically continued. It can include any kind of pores and may be partitioned, to assign material sections.



- Create one or more material definitions which sets the parameters of an user Material including the Depvar definition, 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.
- Mesh the Part (not its Instance!) with elements available in the UEL library. 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. RVE Volume corresponds to V in $\langle \bullet \rangle = \frac{1}{V} \int_{\mathcal{V}} \bullet \, \mathrm{d}\, V$. A short description has to be added.



General Information

Version 3.0

This Plugin bundles all necessary tools to control MonolithFE2

Choose the current step:

- [✓ Generate a Inputscript for MonolithFE2]
- Set Analysisparameters to control the Simulation
- Extract postprocessing data for a resimulation
- Generate Training data for ROM simulations
- Evaluate the training simulations

Specify the previously created RVE model:

Model: Micro

Part: Part-1

RVE Volume: 1.0

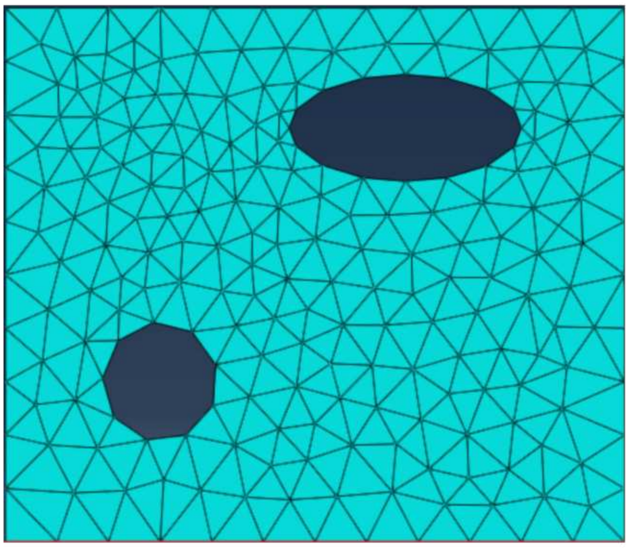
Job name of the macro job: Job_1

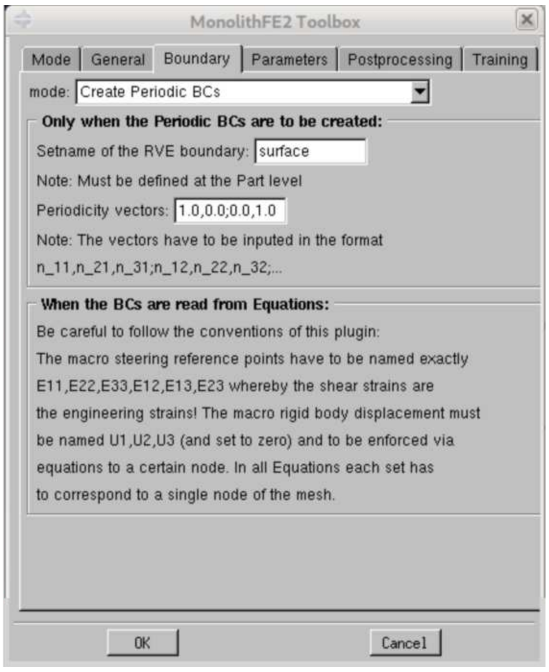
RVE-Number: 1

RVE description: porous microstructure

The text is written into the input file for informational purposes only, but has to be present.

- In the tab „Boundary“ select “Create Periodic BCs” (use the other option only, if you already set up the constraints manually in CAE using the naming conventions as specified in the plugin). Create a set in the Part including the geometry of the boundary (outer edges in 2D, outer surfaces in 3D). Specify the name of this set and specify the periodicity vectors, e.g. 1.0, 0.0; 0.0, 1.0 in 2D for a rectangular RVE.





mode: Create Periodic BCs

Only when the Periodic BCs are to be created:

Setname of the RVE boundary: surface

Note: Must be defined at the Part level

Periodicity vectors: 1.0,0.0;0.0,1.0

Note: The vectors have to be inputed in the format n_11,n_21,n_31;n_12,n_22,n_32;...

When the BCs are read from Equations:

Be careful to follow the conventions of this plugin:
The macro steering reference points have to be named exactly E11,E22,E33,E12,E13,E23 whereby the shear strains are the engineering strains! The macro rigid body displacement must be named U1,U2,U3 (and set to zero) and to be enforced via equations to a certain node. In all Equations each set has to correspond to a single node of the mesh.

- 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 models with existing equations in CAE

It is also possible to set up the constraints "manually", or e.g. by other available tools like EasyPBC, Micromechanics Plugin etc. but then some conventions must be met. Each term of an equation must have a set that contains only a single node, or a reference point. The macroscopic reference points (or nodes) have to be in sets called exactly "U1", "U2", "U3" for the rigid body motion and "E11", "E22", "E33", "E12", "E13", "E23" for the strains (small def.) resp. right stretch (finite def.).

4.3 Set up the model by directly creating an inputfile for MonolithFE^2

The provided Abaqus Plugin is not the only possibility to create an inputfile. It can be created manually and has a syntax which is similar to Abaqus and can therefore be code-highlighted e.g. in Geany, but the commands are not exactly the same! The following rules have to be respected

- Keywords begin with a single star `*`
- Comments begin with two stars `**` and should incorporate at least two separate words, e.g. `**A test`
- Empty lines are not allowed
- Upper and lower case letter make no difference
- The order of the appearing Keywords with content does not matter, but the Elements have to be defined before the Element to Node Assignment is specified
- necessary Keywords:
  - `*Part, "Partname"` where `"Partname"` is just some user defined name, that will be printed in to the Message File
  - `*RVE_Volume, V=`V where V corresponds to the Volume of the RVE in $\langle \bullet \rangle = \frac{1}{V}\int_{\mathcal{V}} \bullet \, \mathrm{d}V$.
  - `*Coupling, Ndof=`n where n is the maximal number of DOF that a node can have. In the following n lines enter n-times `0` or `1` separated by "," to specify if the respective dofs are coupled in the system of equations or not.
  - `*User_Elements, N=`l where l is the number of user element (UEL) definitions that will follow with all of the following Keywords:
    - `*Element`m`, TYPE=...` where m is a element label from 1 to l and `TYPE` is the name of the UEL, whereby the UEL is called with `JTYPE` set to this name which has to be an integer number. This has to be the first Keyword in the Element definition!
    - `*NNODE, N=`o, where o is the number of nodes.
    - `*Element_dof, N=`k where k is the number of DOFs of the element. In the following k lines in each line specify the element node (number from 1 to o) and the node dof (number from 1 to n).
    - `*NSVARS, N=`p where p is the total number of state variables per Element
    - `*PROPS, N=`q where q is the number of real properties that follow in the next line separated by ",".

    - `*JPROPS, N=`r where r is the number of integer properties that follow in the next line separated by ",".
    - `*n_additional_hyper_outputs, N=`s where s is a number of variables that are output by the UEL in `SVARS` needed for the hyper integration (if it is not needed just set it to 0).
    - `*End_of_Element` has to be the last keyword in the Element definition.
  - `*End_of_User_Elements` has to be the last keyword that is wrapped around of the element definitions.
  - `*Node, dimens=`d where d is the dimension (2D=2,3D=3) and in the following lines specify the coordinates of all nodes using d numbers per line separated by "," (the nodes will get the label in the order they were specified by counting from 1)
  - `*Element_to_Node, N=`l where l is the number of element to node assignments that must match the number of Element definitions! In the following lines specify the element to node assignments for all l elements with
    - `*Element_to_Node_Assignments`m`, N=`t with m being a label from 1 to l corresponding to the element definition from above and t is the number of actual elements of that type. Now t lines follow with o entries separated by "," specifying the actual node of the mesh to local node of a specific element.
    - `*End_of_Element_to_Node_Assignments` ends the element to node assignment for the element type
  - `*End_of_Element_to_Node` ends all element to node assignments.
  - `*Reaction_Force_dof, N=`u specifies a number of u of (macroscopic) dofs that are not connected to elements, but inserted into the main program by the array `macro_measures` (in the order in which they are specified in this inputfile). These dofs must appear in the equations and will get a reaction response that is output by the main program through `macro_response` and the condensed tangent w.r.t. these dofs will be output through `DDSDDE`. Dummy nodes have to be set in `*Node` which are then specified in u lines by specifying: node number (dummy), node dof
  - `*Additional_dof, N=`v specifies a number of v of (macroscopic) dofs (to be inserted in `macro_measures` after the reaction force dofs). These dofs also must appear in the equations, but its response will not be computed, even if it might be nonzero.
  - `*Equations, N=`w where w is the number of linear constraints to be specified by:
    - `*EQUATION`x`, N=`y where x is a number from 1 to w and y is the number of terms in the equation. In the following line 3 times y entries have to be specified and separated by ",". The equations have to be put in zero form and in each triplet, the first number is the node number, the second number is the node dof and the third number is the multiplication factor. The first dof in each equation will be removed from the global system of equations and shall not appear again in another equation!
  - `*End_of_Equations` ends specifying the equations.
  - `*End_of_File` ends the inputfile, any information below will be ignored

- optional Keywords:

- `*ROM_Modes, N=`a is necessary only in ROM and hyper ROM simulations, then a is the number of active dofs of the system. In the following lines the modes are to be specified where each node contains a real numbers.
- `*Active_Elements, N=`b is necessary only in hyper ROM simulations, then b is the number of elements which are called. In the following line b element labels must be specified, for the elements that are active, where the elements are counted consecutively starting with the first element type then the second and so on. In the next line the element multiplication factors corresponding to the active element labels have to be specified.

Example for a homogeneous 2D RVE with 4 linear Elements:

```
**MonolithFE2 Version 3.0
**Do not modify this inputfile, otherwise it may not work properly!
*Part, "homogeneous"
*RVE_Volume, V=1.0
*Coupling, Ndof=2
1, 1
1, 1
*User_Elements, N=1
*Element1, TYPE=2008
*Element_dof, N=8
1, 1
1, 2
2, 1
2, 2
3, 1
3, 2
4, 1
4, 2
*NNODE, N=4
*NSVARS, N=6
*PROPS, N=6
100.0, 0.3, 1.0, 0.0, 3.0, 1.0
*JPROPS, N=0
*n_additional_hyper_outputs, N=4
*End_of_Element
*End_of_User_Elements
*Node, dimens=2
-0.5, -0.5
0.0, -0.5
0.5, -0.5
-0.5, 0.0
0.0, 0.0
0.5, 0.0
-0.5, 0.5
0.0, 0.5
0.5, 0.5
```

```
0.0, 0.0
0.0, 0.0
0.0, 0.0
*Element_to_Node, N=1
*Element_to_Node_Assignments1, N=4
1, 2, 5, 4
2, 3, 6, 5
4, 5, 8, 7
5, 6, 9, 8
*End_of_Element_to_Node_Assignments
*End_of_Element_to_Node
**Reaction forces for E11 E22 E12
*Reaction_Force_dof, N=3
11, 1
11, 2
12, 1
**macro displacements U1 U2 (always set to zero -> rigid body motion)
*Additional_dof, N=2
10, 1
10, 2
*Equations, N=12
*EQUATION1, N=2
3, 1, -1.0, 11, 1, 1.0
*EQUATION2, N=2
3, 2, -1.0, 12, 1, 0.5
*EQUATION3, N=2
7, 1, -1.0, 12, 1, 0.5
*EQUATION4, N=2
7, 2, -1.0, 11, 2, 1.0
*EQUATION5, N=3
9, 1, -1.0, 11, 1, 1.0, 12, 1, 0.5
*EQUATION6, N=3
9, 2, -1.0, 12, 1, 0.5, 11, 2, 1.0
*EQUATION7, N=3
8, 1, -1.0, 2, 1, 1.0, 12, 1, 0.5
*EQUATION8, N=3
8, 2, -1.0, 2, 2, 1.0, 11, 2, 1.0
*EQUATION9, N=3
6, 1, -1.0, 4, 1, 1.0, 11, 1, 1.0
*EQUATION10, N=3
6, 2, -1.0, 4, 2, 1.0, 12, 1, 0.5
*EQUATION11, N=2
1, 1, -1.0, 10, 1, 1.0
*EQUATION12, N=2
1, 2, -1.0, 10, 2, 1.0
*End_of_Equations
*End_of_File
```

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.

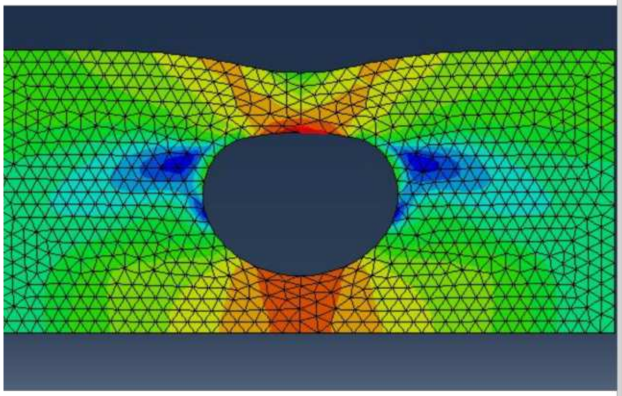
0. Before starting the FE^2 simulation select SDV as field output in the macro model. There the macro stretch tensor (resp. strain) will be saved. Without having the SDVs as field output no postprocessing can be done in large displacement analysis.

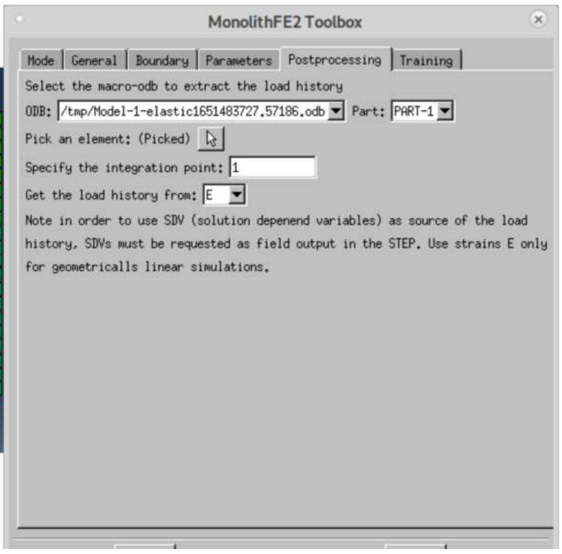
1. Open the created micro model. If the box „generate constraints in CAE“ in the „Generate FE2 microscale mesh“ Plugin as described in chapter 4.1 wasn't ticked in the first call, redo the process, now with ticked box. This creates constraints between nodes on opposite boundary's, boundary conditions, 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.





Select the macro-odb to extract the load history

ODB: /tmp/Model-1-elastic1651483727.57186.odb Part: PART-1

Pick an element: (Picked)

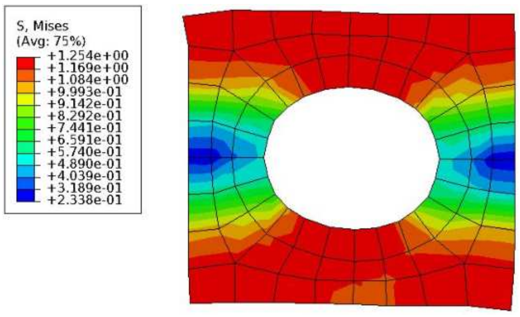
Specify the integration point: 1

Get the load history from: E

Note in order to use SDV (solution depenend variables) as source of the load history, SDVs must be requested as field output in the STEP. Use strains E only for geometricalls linear simulations.

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

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

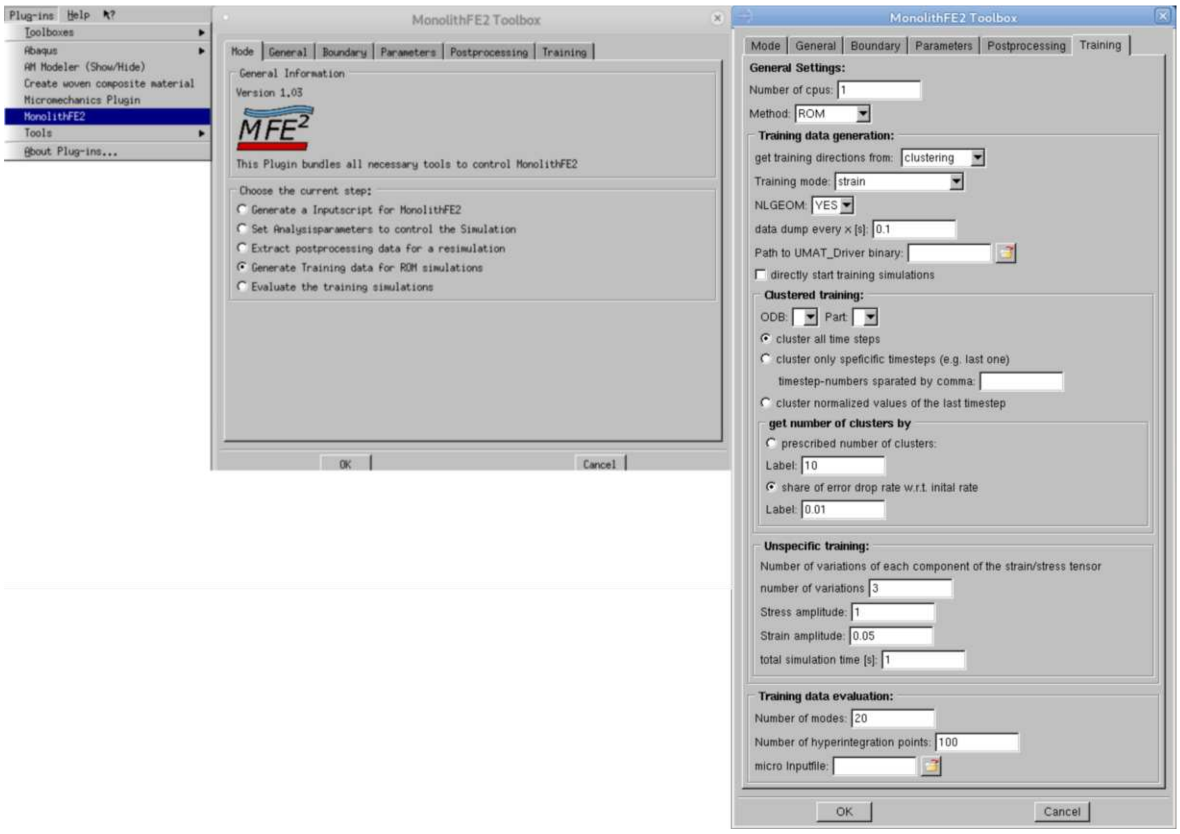


7. Repeat steps 4 to 6 for all macro integration points to be postprocessed.

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 weighted elements. 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.



General Information
Version 1.03
This Plugin bundles all necessary tools to control MonolithFE2

Choose the current step:
- Generate a Inputscript for MonolithFE2
- Set Analysisparameters to control the Simulation
- Extract postprocessing data for a resimulation
- [✓ Generate Training data for ROM simulations]
- Evaluate the training simulations

General Settings:
Number of cpus: 1
Method: ROM

Training data generation:
get training directions from: clustering
Training mode: strain
NLGEOM: YES
data dump every x [s]: 0.1
Path to UMAT_Driver binary:
☐ directly start training simulations

Clustered training:
ODB: Part:
- [✓ cluster all time steps]
- cluster only speficific timesteps (e.g. last one)
  timestep-numbers sparated by comma:
- cluster normalized values of the last timestep

get number of clusters by
- prescribed number of clusters:
  Label: 10
- [✓ share of error drop rate w.r.t. inital rate]
  Label: 0.01

Unspecific training:
Number of variations of each component of the strain/stress tensor
number of variations 3
Stress amplitude: 1
Strain amplitude: 0.05
total simulation time [s]: 1

Training data evaluation:
Number of modes: 20
Number of hyperintegration points: 100
micro Inputfile:

- 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}

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

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

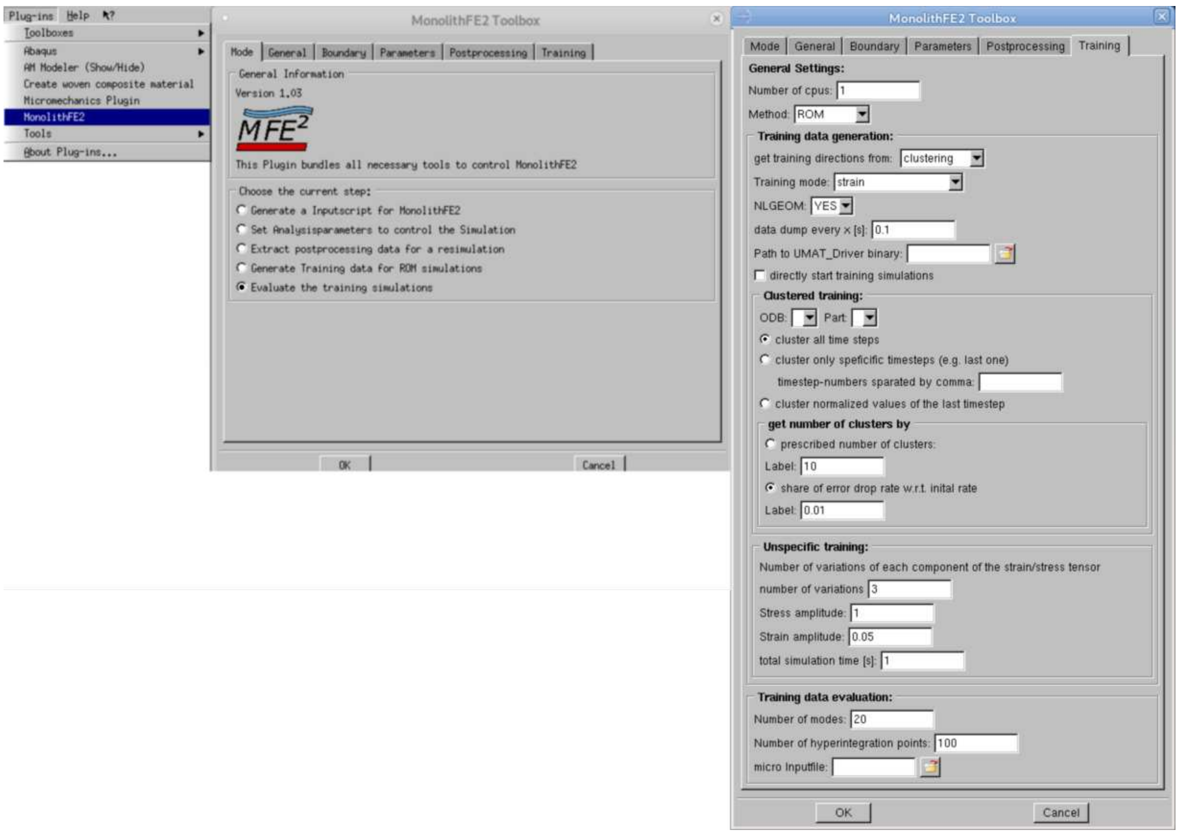
  Thereby the compiled program `UMAT_Driver.o(bj)` must be in the same folder. Note that the `Jobname.FE#` file must have the RVE label 1 (`#=1`)
- The simulations can be parallelized by specifying the number of cpus.
- The simulation outputs the displacement values in files with the name training-data-u-*.txt and the (generalized) "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 hyper elements and their corresponding element multiplication factors.
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“.



General Information
Version 1.03
This Plugin bundles all necessary tools to control MonolithFE2

Choose the current step:
- ( ) Generate a Inputscript for MonolithFE2
- ( ) Set Analysisparameters to control the Simulation
- ( ) Extract postprocessing data for a resimulation
- ( ) Generate Training data for ROM simulations
- [✓ Evaluate the training simulations]

General Settings:
Number of cpus: 1
Method: ROM

Training data generation:
get training directions from: clustering
Training mode: strain
NLGEOM: YES
data dump every x [s]: 0.1
Path to UMAT_Driver binary:
- [ ] directly start training simulations

Clustered training:
ODB: Part:
- [✓ cluster all time steps]
- ( ) cluster only speficific timesteps (e.g. last one)
  timestep-numbers sparated by comma:
- ( ) cluster normalized values of the last timestep

get number of clusters by
- ( ) prescribed number of clusters:
  Label: 10
- [✓ share of error drop rate w.r.t. inital rate]
  Label: 0.01

Unspecific training:
Number of variations of each component of the strain/stress tensor
number of variations 3
Stress amplitude: 1
Strain amplitude: 0.05
total simulation time [s]: 1

Training data evaluation:
Number of modes: 20
Number of hyperintegration points: 100
micro Inputfile:

  - 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 element IDs and multiplication factors 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'`
    - `n_modes=` from 1...n_{modes}
    - `NELEM=` from 1...n_{elements}
    - `ncpus=` from 1...n_{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.

Tab. 3: Options for compilation

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

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

- UELlib [3]
- 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 **??**. When `MATERIAL` is omitted the standard value is `Mises`.

[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.

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. The following table explains selected routines of the MonolithFE^{2}kernel.

filename	**routine**	**description**

`include.f`	-	Includes all files that are needed for MonolithFE^2. It is in turn included by the macro UMAT. If in a new implementation a different macro UMAT and UEL will be implemented: This include file stays the same, the UEL sourcefile must be named exactly "UEL.f" and the new macro UMAT must include this file at the beginning.
`UMATmacro_ mechanical.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
`UMATmacro_ mechanical.f`	get_abaqus_stress_stiffness	return STRESS and DDSDDE in the format Abaqus expects it.
`UMATmacro_ mechanical.f`	trans_stress_stiffness	The (symmetric) Biot stress is transformed to Cauchy stress (also the material tangent: nominal tangent $\rightarrow$ DDSDDE).
`UMATmacro_ mechanical.f`	polar_decomposition	The macro deformation gradient is split in the rotation and right stretch tensor
`MonolithFE2.f`	main_program_staggered	actual main program for the staggered algorithm, which calls all needed routines (assemble, solve...) in the right order
`MonolithFE2.f`	main_program_monolithic	main program for monolithic algorithm
`MonolithFE2.f`	convergence_check	checks if convergence is reached (currently infinity norm)
`MonolithFE2.f`	enforce_ constraint	Enforce the constraint using the Equations defined in the Inputfile.
`MonolithFE2.f`	update_nodal_ sol	Update the nodal solution, by adding the computing increment at the correct place of the solution vector and enforcing the constraint.
`MonolithFE2.f`	assemble	Call the element routine for all active elements and sort the right hand side and the tangent matrix at the correct place and manage the state variables.
`MonolithFE2.f`	assemble_full_derivative_ matrix_ and_RHS	In the HF simulation case, insert the tangent matrix into the sparse system matrix and the non sparse matrices corresponding to macro reactions, sort in the RHS.
`MonolithFE2.f`	assemble_ROM_derivative_ matrix_ and_RHS	In the hyper ROM simulation case, insert the tangent matrix into the non sparse system matrix and the non sparse matrices corresponding to macro reactions, sort in the RHS.
`MonolithFE2.f`	ROM_projection	Do the Galerkin projection with the ROM modes in the not hyper integrated case for the RHS and the Tangent.
`MonolithFE2.f`	static_condensation	Compute the macroscopic response and corresponding tangent through static condensation.
`type_solver.f`	type_solver: initialize, factor, solve, finish	Empty interface, that declares an object with methods, to either call the PARDISO solver for sparse system matrices or the LAPACK solver for dense systems.
`type_solver.f`	Solver_PARDISO:initialize, factor, solve, finish, get_permutation_matrix, interpret_error	Module that wraps around the PARDISO solver
`type_solver.f`	Solver_LAPACK:initialize, factor, solve, finish	Module that wraps around the LAPACK d(ge/sy)tr(s/f) solver
`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_SVARS_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 allocate_solver_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_meshparameters.f`	type_meshparameters: 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.). The read_data routine reads the data from disk and sets up the system of equations, the get_pointer routine accesses the data, by setting the pointers and the deallocate routine frees the allocated memory
`type_analysisparameters.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.
`type_systemmatrices.f`	allocation deallocation	In this module a FORTRAN type is declared, that encapsulates the matrices of derivatives, rhs's and solution increments.

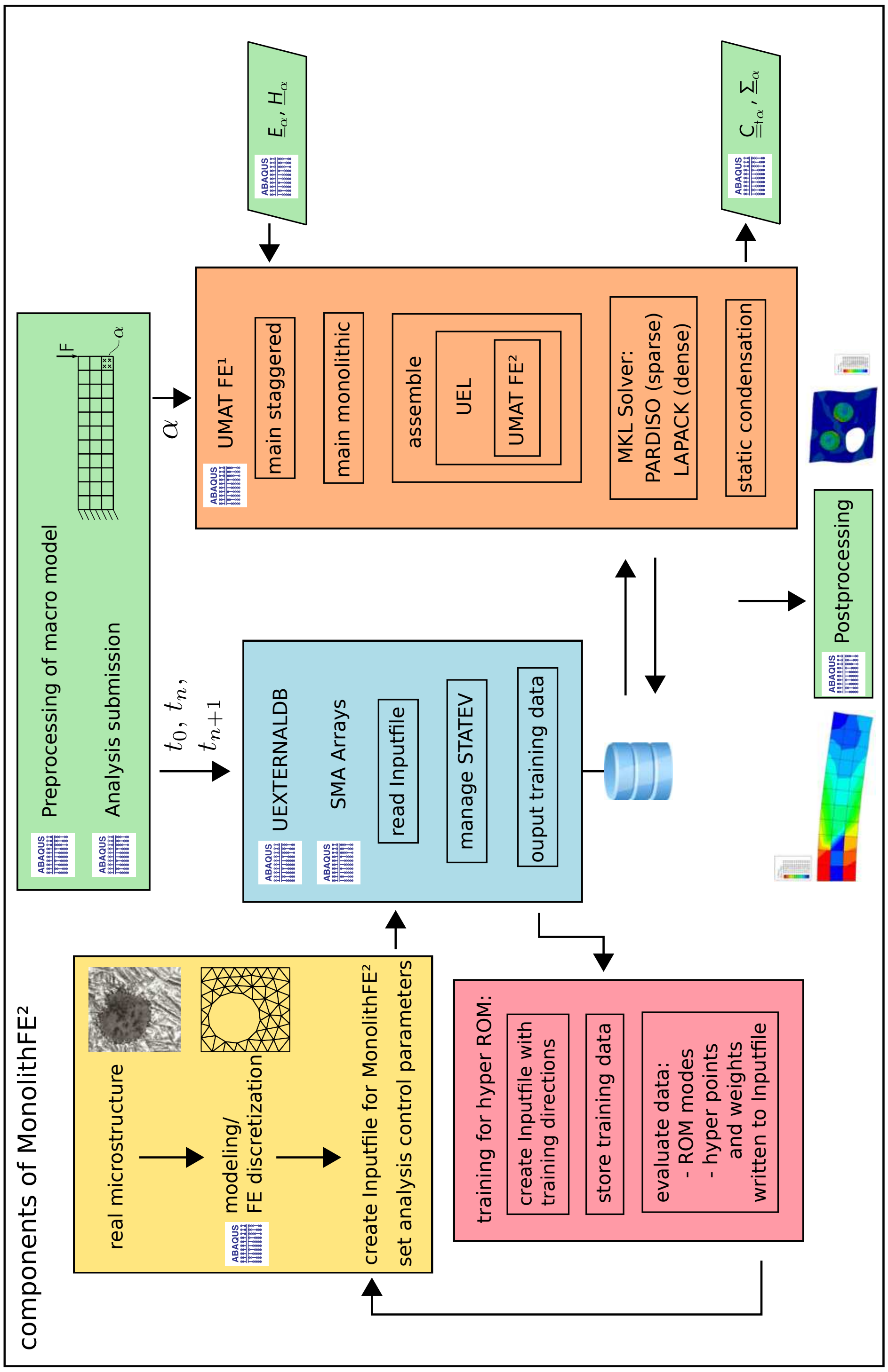


Fig. 1: Structure of MonolithFE^2

8 Limitations, Problems and Future Developments

8.1 Limitations

- The provided UELs are only suited for mechanical behavior (not thermal etc.).

- The provided UMAT at the macroscale (`UMATmacro_mechanical.f`) is only sutied for mechanical behavior. If “multiphysical” problems have to be solved a respective UMAT has to be programmed in analogy, that interprets the in- and outputs differently.
- No real visualization of the Microscale FE problems. In a future work this could be done with ParaView if necessary.
- 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)

8.2 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").

8.3 Software environment

- The file `ABA_PARAM.INC` is not found by Abaqus by default under Linux, but put "artificially" to folder `src/`. Its content 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.

8.4 Development for generalized continua

If other cases than First Order mechanical Homogenization have to be simulated, follow these implementation steps:

- Write a micro element using the UEL interface as defined by Abaqus (if the mechanical cauchy continuum is not sufficient) and call the source code to be included (may itself include other files) exactly “UEL.f” and make it accessible while compiling the program.
- Write an RVE Inputfile for MonolithFE^2, that defines the created UEL element and suitable constraints correctly.
- Write an “generalized UMAT” at the macro scale in analogy to the existing First Order mechanical one, but interpret the `macro_measure` input to MonolithFE^{2}and resulting reaction force`macro_response` and tangent `DDSDDE` according to the problem solved.
- Write a macro UEL, that calls the macro UMAT at its integration points.

9 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
2024-04-05	v3.0, major version change, new Inputfile without information on the structure of the system of equations, more flexibility on the elements (more than one element type may be defined), general Equations, general number of dof per node of an element, element based hyper integration, UEL interface called without non-Abaqus-Standard arguments, System of equations is set up at the start of the simulation.

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