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

Faculty of Mechanical, Process and Energy Engineering
Institute for Mechanics and Fluid Dynamics (IMFD)
Chair Applied Mechanics – Solid Mechanics

UELlib
A library for user-defined elements in Abaqus

Geralf Hütter, Stephan Roth, Rostyslav Skrypnyk

April 19, 2021

Contact:
Geralf Hütter
TU Bergakademie Freiberg
Institute of Mechanics and Fluid Dynamics
Lampadiusstr. 4
09596 Freiberg, Germany
Geralf.Huetter@imfd.tu-freiberg.de

Contents

1 General

The commercial finite element code Abaqus/standard provides the UEL interface to implement user-defined elements. Implementing different problems as UEL often requires the same ingredients like particular shape functions, quadrature rules and mathematical utility routines. The present library UELlib provides a library with modular structure for the implementation of elements via the UEL interface. UELlib distributed under a CC BY-NC-SA 4.0 license. UELlib has been used by the authors in [1, 2, 3, 4]. Please refer to any of these publications when using UELlib.

Requirements to the program:

1. It shall be possible to create elements with different different dimensionality (2D/3D), material routines, shape functions, and different quadrature rules without modifying the source code of the actual UEL routine but by changing only the loaded modules.
2. It shall be possible to use elements with different shape functions, quadrature rule etc. in a single simulation.
3. The library shall be easily serviceable, i.e. it shall contain as few dublicate information and subroutines as possible.
4. The modular structure shall not influence the runtime speed if possible. For this purpose the libraries are procedurally programmed and it is envisaged that as much information on dimensions of arrays are accessible at compile time in order to allow vectorization.
5. However, in order to achieve the desired flexibility the library implements static polymorphism features and modules in FORTRAN 90/95.

The source code has been developed and extensively tested using versions 6.10EF1 and 6.12-3 of ABAQUS with version 11.0 of the Intel Fortran compiler. Later, it was used succesfully with Abaqus versions 6.14, 2018 and 2020 and Intel Fortran compiler 2017.

2 Installation

2.1 Files

In folder `lib/`:

- `ABQinterface.f90`: provides data types of Abaqus variables and explicit interfaces to Abaqus utility routines
- `Math.f90`: mathematical utility routines e.g. for inversion of matrices
- `UEL_lib.f90`: shape functions, quadrature rules and B matrices

2.2 Example elements

In folder `examples/`:

- `elastic`: standard small displacement element, material routines for isotropic elastic 3D and 2D plane stress and plain strain analysis by Rostyslav Skrypnyk
- `empty_element_framework`: framework for building new elements

2.3 Usage

UELLIB can either be used by including its source files (as listed in section 2.1) or by linking the compiled UELLIB to the desired library. Both scenarios are demonstrated with the example elements for which respective `Makefiles` are provided to build `libstandardU` in subfolders `bin/` of the example elements. They are implemented as make targets buildlibsrc and buildlibbin, respectively. In the latter case the files of UELLIB have to be compiled first. A further `Makefile` ist provided in the root directory for this task to build the object files in folder `bin/`.

If UELLIB shall be used for building a new element, copy one of the content of one folders of the example elements to another folder and start editing it. The variable UELlibpath in the copied `Makefile` has to be set to the directory where UELLIB is located.

3 Implementation

3.1 Structure

3.1.1 Interface to Abaqus

The module ABQINTERFACE in `ABQinterface.f90`provides an explicit interface to some utility provided by Abaqus. Another important point is the precision of integer and float variables in the UEL interface. Abaqus offers only an implicit type declaration over the file `ABA_PARAM.INC`. However, implicit type declaration is prone to errors and thus outdated. For this reason private variables are defined implicitly in ABQINTERFACE. Their "kinds" are picked and provided as AbqRK and AbqIK for explicit type declarations in all following modules and routines.

3.1.2 Shape Functions

The shape functions are provided over the modules ShapeFunc* in `UEL_lib.f90`. Each of this modules contains

- ShapeFunc(chi) which returns the vector (of size NNODES) of values of the shape functions at a given position chi in the unit domain. chi is a column vector of suitable dimension NDIM. According to the concept of the libraries, the values NNODES and NDIM are provided as paremeters and can be used for defining fixed-size arrays.
- ShapeFuncDeriv(chi) returns the values of the derivative of the shape functions w.r.t. the unit coordinates chi as array of dimension **DIMENSION**(NNODES,NDIM).

3.1.3 Quadrature Rules

The modules Integr*_* contain the quadrature rules. Their names like Integr2D_Square9GP indicate that they refer to a plane square unit domain with nine Gauss points. Each of this modules provides:

- the number of Gauss points NGP
- the dimension of unit domain NDIMGP
- a column vector GPWeight with weights of the Gauss points
- a matrix GPPos(NGP,NDIMGP) containing the positions of the points in the unit domain

All these values are defined as parameters and are thus available at compile time.

3.1.4 B-matrices

The modules BMatrices* contain functions to derive the B-matrices for different szenarios

- BMatTensSym for symmetric tensors in Voigt notation like e.g. the strain tensor
- BMatTensUnsym for unsymmetric tensors in Voigt notation like e.g. the deformation

by default the B-matrices are computed from the inverse of Jacobian of the isoparametric mapping and from the derivative matrix of the shape functions w.r.t. the unit coordinates. Alternative versions of the function with suffix _SFG are available indicating that the B-matrices are computed from the already computed gradient of the shape functions w.r.t. the physical coordinates. This option may save computational time in multi-field problems if the same shape functions are used for the different fields.

3.2 Applications

3.2.1 Elements with Different Shape Function, Quadrature Rules etc.

If only a single element shall be developed the aforementioned modules can just be loaded in the UEL routine. However, one main goal of the present library is to allow to develop elements with different shape function, quadrature rules, dimensions etc. with a single source code, i.e. without the necessity of maintaining different almost identical source codes for similar elements. If they shall be used in a single simulation (or compiled into a single universal shared library) this can be achieved via modules as demonstrated in the examples:

```
MODULE CXU2D4PlaneStrain
  USE ShapeFunc2D_Square_Lin
  USE BMatrices2DPlaneStrain
  USE Integr2D_Square4GP
  USE UMAT1
  IMPLICIT NONE
  PUBLIC :: UEL
  CONTAINS
    INCLUDE "UelCXU.f90"
END MODULE
```

Then the file UelCXU.f90 contains the actual UEL that makes use of the generally named routines and variables defined in the modules as described in section 3.1 (like ShapeFunc(chi), NNODES etc.). Several of such modules with different combinations of shape function, quadrature rules, dimensions etc. can be created. This means that the actual element routine from UelCXU.f90 is compiled individually for each combination allowing the respective optimzations since information like the size of the arrays are available at compile time. Finally, the modules with different element routines are managed by the "global" UEL routine provided to Abaqus. This routine loads the modules and calls the respective module element routine depending on the JTYPE parameter. The latter contains the number of the user element defined in the Abaqus input file, e.g. a definition *USER ELEMENT, TYPE=U2001, ... would correspond to JTYPE=2001.

```
SUBROUTINE UEL(...Parameters according to Abaqus UEL interface...)
  USE CXU2D4PlaneStrain, ONLY: UEL1 => UEL
  USE CXU3D20, ONLY: UEL2 => UEL

!  Type declarations ...
```

```
  SELECT CASE( JTYPE )
    CASE (2001)
      CALL UEL1(Parameters)
    CASE (2002)
      CALL UEL2(Parameters)
  END SELECT
END SUBROUTINE
```

Note that the local renaming operator => is used here to distinguish the originally equally named module element routines. It may be remarked here that the interface of the module element routines has not inevitably to coincide with the Abaqus UEL interface. For instance module element routines may use assumed-shape arrays or may drop some irrelevant parameters.

3.2.2 Multi-field problems

Using the possibilities of Fortran 90/95 also multi-field problems may be implemented easily. For this purpose the shape function modules are loaded several times and the needed subroutines are renamed correspondingly. For instance for a coupled temperature-displacement element with quadratic shape functions for the displacements and linear ones for the temperature the frame could look like this:

```
MODULE CXUT2D8M1
  USE  ShapeFunc2D_Square_Quad,&
    ONLY: ShapeFuncDispl => ShapeFunc, &
    ShapeFunDerivcDispl => ShapeFuncDeriv, &
    NNODESDISPL=>NNODES, NDIM
  USE  BMatrices2DPlaneStrain,&
    ONLY: BmatStrain => BMatTensSym, NDI, NSHR, NTENS
  USE  ShapeFunc2D_Square_Lin,&
    ONLY: ShapeFuncTemp => ShapeFunc, &
    ShapeFunDerivcTemp => ShapeFuncDeriv, NNODESTEMP=>NNODES
  USE  BMatricesScalar,  ONLY: BmatTemp => BMatScal
  IMPLICIT NONE
  PUBLIC::UEL
  CONTAINS
    INCLUDE "UelCXUT.f90"
END MODULE
```

Now the actual element routine in `UelCXUT.f90` can use the loaded routines like ShapeFuncDispl and ShapeFuncTemp which may be although identical if the same modules are loaded for temperature and displacement.

4 Perspective and Problems

4.1 Known problems

- For an unknown reason the UEL interface of Abaqus reserves arrays of different sizes for the generalized vectors of displacements and internal forces, namely U(NDOFEL) and **DIMENSION** RHS(MLVARX,NRHS) with NDOFEL *not* being equal to MLVARX (actually, the latter is larger). If the implementation is done in Fortran 77 with all vector and matrix operations implemented via DO loops one does not need to care about this matter.

However, if use of vector and matrix capabilities of Fortran 90/95 shall be made allowing a vectorization for an improved performance one has to take care. We recommend to introduce an additional array of suitable **DIMENSION** RHS_TEMP(NDOFEL,NRHS) and finally copy the respective entries to the original array RHS as it is done in the provided example.

- The provided subroutines should support complete parallelization since they do not use critical features like COMMON blocks, module variables or variables with SAVE attribute. However, at least under the tested environments the compiler options used by default by Abaqus do not permit parallelization. The compiler options are stored in the file `"ABQDIR/site/abaqus_v6.env"`. It has to be copied to the job directory. Within this file, the line `compile_fortran="... -auto ..."` has to be changed to `compile_fortran="... -openmp ..."`. With this modification the computations can be run completely in parallel, i.e. for computing the element matrices and solving the final system of equations.
- Abaqus versions 2016 and later employ the compiler option –fpp by default. However, the Intel Fortran preprocessor fpp does not recognize the compiler directive *!DEC$ FREEFORM* which switches to free-form Fortran. In order to overcome this problem, either the switch –fpp has to be removed in the line `compile_fortran="..."` of the Abaqus environment `abaqus_v6.env` or the compiler option –free has to be set therein explicitly.
- The B-matrices for unsymmetric tensors are not verified yet.

4.2 Future developments:

- second derivatives of shape functions as necessary e.g. for Euler-Bernoulli beam elements or Kirchhoff plate elements
- B-matrices for higher order tensors
- utilities for surface loads for implementing DLOAD routine with UELlib
- performance improvements
  - usage of libraries like Intel MKL or LAPACK for mathematical utility routines
  - hard coded derivatives of shape functions with respect to unit coordinates to improve the performance
  - inline compilation of smaller utility routines
- testing and verification with other compilers and under MS Windows

5 Version History

Date	SVN/GIT version	editor	remarks
2014/05/06	1	GH	framework with interfaces of routines only

Date	SVN/GIT version	editor	remarks
2014/05/06	3	GH	first working version with example element by R. Skrypnyk
2014/05/07	5	GH	added: Gauss point numbering as in Abaqus, separated B-matrices based on shape function gradient; modifications: changed naming of some modules and routines
2014/05/08	6	GH	elimination of minor bugs
2014/06/24	8	GH	added: shape functions and integration scheme for linear triangle element + example
2014/10/27	9	GH	added: shape functions and integration scheme for tetrahedra by RS
2015/04/22	10	GH	added: quadratic shape functions for triangle element incl. integration schemes and shape functions and integration schemes for line elements by S. Roth
2015/10/23	18	GH	added: `Makefiles` for compiling UELLIB and the provided example elements (requiring some restructuring of the folders of the latter)
2015/10/26	19	GH	changed: Use of Abaqus utility routine STDB_ABQERR() for reporting errors in submitted material parameters for example element instead of **WRITE**()
2015/11/06	24	GH	`Makefile` now builds single object file `UEL_lib-std.o`, which contains all modules, in folder `/bin`
2015/12/04	27	GH	added some direct integration rules for quadrilateron (5,7,8 point) and hexahedron (14 points)
2017/02/17	31	GH	removed: bug when calling inverse() without optional argument det; added: generic interface inverse which can be called also as function
2017/10/23	33	SR	B-matrices for axi- and spheri-symmetric elements
2018/02/07	35	GH	added: "SFG" variant of B-matrices for axi-symmetric elements; few minor modifications and optimisations
2018/03/06	04dd530	GH	added: interface to PreFactor to be accessible as function or subroutine, the latter providing the derivative w. r. t. radius for large displacement analyses
2018/06/27		GH	added to documentation: hint regarding compiler options for Abaqus 2016 and later

References

[1] S. ROTH, G. HÜTTER, M. KUNA: Simulation of fatigue crack growth with a cyclic cohesive zone model, International Journal of Fracture, 188 (2014), 23-45.

[2] G. HÜTTER, R. SKRYPNYK: Micromorphic Homogenisation of a Porous Medium: Application to Size Effects and Quasi-Brittle Damage, Proceedings of Applied Mathematics and Mechanics 16 (2016), 347-348.

[3] G. HÜTTER: Micromorphic homogenisation and its application to a model of ductile damage, Proceedings of Applied Mathematics and Mechanics 17 (2017), 599-600.

[4] N. Lange, G. Hütter, B. Kiefer: An efficient monolithic solution scheme for FE^2 problems, arxiv.org/2101.01802.