ABAQUS User Element (UEL) for Coupled Multiphysics Moisture Diffusion in Epoxy

Overview

This FORTRAN subroutine implements a User Element (UEL) for ABAQUS finite element software to simulate **coupled multiphysics behavior** involving moisture diffusion and mechanical deformation in epoxy materials. The element captures the bidirectional coupling between **mechanical stress fields** and **moisture transport phenomena**, making it a true multiphysics simulation.

Multiphysics Problem Description

This implementation addresses a **coupled hydro-mechanical multiphysics system** where multiple physical phenomena interact simultaneously:

Primary Physics Domains

1. **Mechanical Physics (Structural Domain)**
   - Elastic deformation under applied loads
   - Stress-strain relationships
   - Equilibrium equations
2. **Diffusion Physics (Mass Transport Domain)**
   - Moisture concentration transport
   - Fick's diffusion laws
   - Concentration gradient-driven flow
3. **Coupling Physics (Interaction Domain)**
   - Stress-assisted diffusion
   - Hydrostatic stress influence on transport
   - Mechanical-chemical coupling

Multiphysics Coupling Mechanisms

1. Mechanical → Diffusion Coupling

```
∂c/∂t = ∇·(D∇c) - (D·Vh)/(R·T) ∇·(σh∇c)
         ↑_______↑   ↑__________________↑
      Pure diffusion   Stress-assisted term
```

Physical Interpretation:

- **Tensile stress** (σh > 0): Enhances diffusion by opening material microstructure
- **Compressive stress** (σh < 0): Reduces diffusion by closing pore networks
- **Stress gradients**: Create preferential diffusion pathways

2. **Diffusion → Mechanical Coupling** (Implicit)

While not explicitly implemented in this version, moisture typically affects:

- Material properties (E, ν variations with concentration)
- Hygroscopic swelling strains
- Degradation of mechanical properties

Governing Multiphysics Equations

Mechanical Equilibrium:

```
∇·σ = 0
σ = D:(ε - εswelling)  // εswelling would be concentration-dependent
```

Coupled Diffusion:

```
∂c/∂t = ∇·[D(∇c - (Vh/RT)σh∇c)]
```

Stress-Diffusion Coupling Parameter:

```
κ = Vh/(R·T) = 8000/(8314.5 × 300) ≈ 3.2 × 10⁻³ MPa⁻¹
```

Multiphysics Finite Element Formulation

Coupled DOF System

The element implements a **monolithic multiphysics approach** with:

- **80 total DOF per element**
- **Mechanical DOF (1-60)**: 3 displacement components × 20 nodes
- **Diffusion DOF (61-80)**: 1 concentration × 20 nodes

Multiphysics Matrix Structure

```
[K_mech    K_couple] [Δu] = [R_mech]
[K_couple^T K_diff ] [Δc]   [R_diff]
```

Where:

- **K_mech**: Standard mechanical stiffness
- **K_diff**: Diffusion stiffness with stress coupling
- **K_couple**: Cross-coupling terms (would include swelling effects)

Time Integration Strategy

Multiphysics time scales:

- **Mechanical**: Quasi-static (instantaneous equilibrium)
- **Diffusion**: Transient (governed by diffusion time constants)

Current Implementation: Backward Euler for diffusion, quasi-static for mechanics

Code Structure

Module Declaration

```fortran
module kvisual
    implicit none
    real*8 UserVar(70000,16,8)  ! State variables: [elements, variables, int_points]
    integer nelem               ! Total number of elements
    save
end module
```

Purpose: Stores state variables (stress, strain, etc.) for visualization and data transfer between subroutines.

Main UEL Subroutine

Input Parameters

```fortran
subroutine uel(rhs,amatrx,svars,energy,ndofel,nrhs,nsvars,
     1 props,nprops,coords,mcrd,nnode,u,du,v,a,jtype,time,dtime,
     2 kstep,kinc,jelem,params,ndload,jdltyp,adlmag,predef,npredf,
     3 lflags,mlvarx,ddlmag,mdload,pnewdt,jprops,njpro,period)
```

Parameter	Description
`rhs`	Right-hand side vector (residual forces)
`amatrx`	Element stiffness matrix
`svars`	State variables
`props`	Material properties array
`coords`	Nodal coordinates
`u`	Current nodal displacements and concentrations
`du`	Incremental displacements and concentrations
`dtime`	Time increment

Element Characteristics

```fortran
parameter(ndim=3,ntens=6,ninpt=8,nsvint=16)
```

- **ndim=3**: 3D analysis
- **ntens=6**: 6 stress/strain components (σxx, σyy, σzz, τxy, τyz, τxz)
- **ninpt=8**: 8 Gauss integration points
- **nsvint=16**: 16 state variables per integration point

DOF Structure

The element has **80 total DOF**:

- **DOF 1-60**: Mechanical (3 displacement DOF × 20 nodes)
- **DOF 61-80**: Concentration (1 concentration DOF × 20 nodes)

Material Properties Reading

```fortran
if(kstep.eq.1) then
    D=props(3) ! Diffusion coefficient for step 1
endif
if(kstep.eq.2) then
    D=props(4) ! Diffusion coefficient for step 2
endif
```

Material Properties Array (`props`):

- `props(1)`: Young's modulus (E)
- `props(2)`: Poisson's ratio (ν)
- `props(3)`: Diffusion coefficient for step 1
- `props(4)`: Diffusion coefficient for step 2

Physical Constants

```fortran
Vh=8000.d0     ! Molar volume of water [mm³/mol]
T=300.d0       ! Temperature [K]
R=8314.5d0     ! Gas constant [J/(mol·K)]
```

Node Coordinate System

20-Node Hexahedral Element

The element uses a 20-node quadratic hexahedral topology:

Corner nodes (1-8): Located at (±1, ±1, ±1) **Edge nodes (9-20)**: Located at mid-edges

```fortran
! Natural coordinates for 20-node element
coord320(1,1)=-1.d0; coord320(2,1)=-1.d0; coord320(3,1)=-1.d0  ! Node 1
coord320(1,2)=1.d0;  coord320(2,2)=-1.d0; coord320(3,2)=-1.d0  ! Node 2
! ... (pattern continues for all 20 nodes)
```

Hydrostatic Stress Calculation

```fortran
do inod=1,nnode
    g=dsqrt(3.d0)*coord320(1,inod)
    h=dsqrt(3.d0)*coord320(2,inod)
    l=dsqrt(3.d0)*coord320(3,inod)
    ! Calculate shape functions at nodal points
    dNS(1)=(1.d0-g)*(1.d0-h)*(1.d0-l)/8.d0
    ! ... (continue for all 8 corner nodes)

    ! Interpolate hydrostatic stress to nodes
    do i=1,ninpt
        isvinc=(i-1)*nsvint
        SHa(inod,1)=SHa(inod,1)+dNS(i)*svars(isvinc+10)
    end do
end do
```

Purpose: Extrapolates hydrostatic stress from integration points to nodes for use in diffusion calculations.

Integration Loop

Shape Function Evaluation

```fortran
do kintk=1,ninpt
    call kshapefcn(kintk,ninpt,nnode,ndim,dN,dNdz)
    call kjacobian(jelem,ndim,nnode,coords,dNdz,djac,dNdx,mcrd,COFACTOR)
    dvol=wght(kintk)*djac
```

For each integration point:

1. **kshapefcn**: Computes shape functions and derivatives in natural coordinates
2. **kjacobian**: Transforms derivatives to global coordinates and computes Jacobian
3. **dvol**: Differential volume element

B-Matrix Formation

```fortran
! Strain-displacement matrix for mechanics
do k=1,nnode
    b(1,1+ndim*(k-1))=dNdx(1,k)  ! ∂N/∂x for εxx
    b(2,2+ndim*(k-1))=dNdx(2,k)  ! ∂N/∂y for εyy
    b(3,3+ndim*(k-1))=dNdx(3,k)  ! ∂N/∂z for εzz
    b(4,1+ndim*(k-1))=dNdx(2,k)  ! ∂N/∂y for γxy
    b(4,2+ndim*(k-1))=dNdx(1,k)  ! ∂N/∂x for γxy
    ! ... (continue for shear strains)
end do

! Gradient matrix for diffusion
do inod=1,nnode
    bC(1,inod)=dNdx(1,inod)  ! ∂N/∂x
    bC(2,inod)=dNdx(2,inod)  ! ∂N/∂y
    bC(3,inod)=dNdx(3,inod)  ! ∂N/∂z
end do
```

Current Concentration Calculation

```fortran
cL=0.d0
do inod=1,nnode
    cL=cL+dN(1,inod)*u(3*nnode+inod)  ! Interpolate concentration
end do
```

Note: `u(3*nnode+inod)` accesses concentration DOF (DOF 61-80).

Material Response

```fortran
! Get strain increment
dstran=matmul(b,du(1:ndim*nnode,1))

! Retrieve state variables
call kstatevar(kintk,nsvint,svars,statevLocal,1)
stress=statevLocal(1:ntens)
stran(1:ntens)=statevLocal((ntens+1):(2*ntens))

! Update stress and get material tangent
call kumat(props,ddsdde,stress,dstran,ntens,statevLocal,ndim)
stran=stran+dstran

! Update state variables
statevLocal(1:ntens)=stress(1:ntens)
statevLocal((ntens+1):(2*ntens))=stran(1:ntens)
statevLocal(2*ntens+2)=(stress(1)+stress(2)+stress(3))/3.d0  ! Hydrostatic stress

call kstatevar(kintk,nsvint,svars,statevLocal,0)
```

Matrix Assembly

Mechanical Contribution (DOF 1-60)

```fortran
! Stiffness matrix: K_mech = ∫[B^T·D·B]dV
amatrx(1:60,1:60)=amatrx(1:60,1:60)+dvol*matmul(matmul(transpose(b),ddsdde),b)

! Residual: R_mech = -∫[B^T·σ]dV
rhs(1:60,1)=rhs(1:60,1)-dvol*(matmul(transpose(b),stress))
```

Diffusion Contribution (DOF 61-80)

```fortran
! Mass matrix (time derivative term)
xm=matmul(transpose(dN),dN)/D

! Diffusion matrix with stress coupling
BB=matmul(transpose(bC),bC)
xk=BB-Vh/(R*T)*matmul(BB,matmul(SHa,dN))

! Diffusion stiffness: K_diff = ∫[(N^T·N)/D/Δt + BB - (Vh/RT)·BB·σh·N]dV
amatrx(61:80,61:80)=amatrx(61:80,61:80)+dvol*(xm/dtime+xk)

! Diffusion residual: R_diff = -∫[K·c + M·Δc/Δt]dV
rhs(61:80,1)=rhs(61:80,1)-dvol*(matmul(xk,u(61:80))+matmul(xm,du(61:80,1))/dtime)
```

Key Terms:

- **xm/dtime**: Backward Euler time integration
- **BB**: Standard diffusion (Fick's law)
- **Vh/(R*T)*matmul(BB,matmul(SHa,dN))**: Stress-assisted diffusion term

Supporting Subroutines

kshapefcn: Shape Function Computation

```fortran
subroutine kshapefcn(kintk,ninpt,nnode,ndim,dN,dNdz)
```

Purpose: Computes 20-node quadratic hexahedral shape functions and their derivatives in natural coordinates (ξ,η,ζ).

Shape Functions:

- **Corner nodes (1-8)**: `Ni = 0.125*(1±ξ)*(1±η)*(1±ζ)*(-ξ-η-ζ-2)`
- **Edge nodes (9-20)**: `Ni = 0.25*(1-ξ²)*(1±η)*(1±ζ)` (and permutations)

kjacobian: Jacobian Computation

```fortran
subroutine kjacobian(jelem,ndim,nnode,coords,dNdz,djac,dNdx,mcrd,COFACTOR)
```

Purpose:

1. Computes Jacobian matrix: `J[i,j] = Σ(∂Nk/∂ξi · xk,j)`
2. Calculates Jacobian determinant
3. Transforms shape function derivatives to global coordinates: `∂N/∂x = J⁻¹ · ∂N/∂ξ`

kstatevar: State Variable Management

```fortran
subroutine kstatevar(npt,nsvint,statev,statev_ip,icopy)
```

Purpose: Transfers state variables between element-level and integration point-level arrays.

Parameters:

- `icopy=1`: Copy from element array to integration point array (before material call)
- `icopy=0`: Copy from integration point array to element array (after material call)

kumat: Material Model

```fortran
subroutine kumat(props,ddsdde,stress,dstran,ntens,statev,ndim)
```

Purpose: Implements linear elastic material model.

Constitutive Matrix (3D):

```
E = props(1)    ! Young's modulus
v = props(2)    ! Poisson's ratio

λ = E/((1-2v)(1+v))     ! Lame parameter
G = E/(2(1+v))          ! Shear modulus

D = [λ+2G   λ     λ     0   0   0  ]
    [λ     λ+2G   λ     0   0   0  ]
    [λ     λ     λ+2G   0   0   0  ]
    [0     0     0     G   0   0  ]
    [0     0     0     0   G   0  ]
    [0     0     0     0   0   G  ]
```

Stress Update: `σ = σ₀ + D·Δε`

umat: ABAQUS Material Interface

```fortran
subroutine umat(...)
```

Purpose: Provides interface for ABAQUS material routines. In this implementation, it simply retrieves state variables from the UEL for visualization purposes.

State Variables

The element tracks 16 state variables per integration point:

Index	Variable	Description
1-6	σxx, σyy, σzz, τxy, τyz, τxz	Stress components
7-12	εxx, εyy, εzz, γxy, γyz, γxz	Strain components
13	-	Unused
14	σh	Hydrostatic stress
15	cL	Local concentration
16	-	Unused

Usage Notes

Input File Setup

```
*USER ELEMENT, TYPE=U1, NODES=20, COORDINATES=3, VARIABLES=128
1,2,3,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80

*UEL PROPERTY, ELSET=EPOXY_ELEMENTS
E, nu, D1, D2
! Example: 3000.0, 0.35, 1.0E-3, 2.0E-3
```

DOF Assignment

- Nodes 1-20: DOF 1,2,3 (displacements) + DOF 11 (concentration)
- Element DOF mapping: [u1x,u1y,u1z,u2x,...,u20z,c1,c2,...,c20]

Physical Interpretation

Stress-Assisted Diffusion

The term `Vh/(R*T)*∇·(σh∇c)` represents the effect of hydrostatic stress on moisture transport:

- **Positive σh (tension)**: Enhances diffusion (stress opens material structure)
- **Negative σh (compression)**: Reduces diffusion (stress closes material structure)
- **Vh/(R*T)**: Stress-diffusion coupling parameter

Applications in Multiphysics Engineering

This multiphysics model is particularly relevant for:

Aerospace Applications

- **Composite structures**: Moisture absorption in carbon fiber/epoxy laminates
- **Environmental durability**: Hot/wet certification requirements
- **Stress concentration effects**: How mechanical loads accelerate moisture uptake

Marine Engineering

- **Hull structures**: Underwater pressure effects on moisture diffusion
- **Adhesive joints**: Combined mechanical loading and seawater exposure
- **Fatigue analysis**: Moisture-assisted crack propagation

Electronics Packaging

- **Chip encapsulation**: Moisture diffusion under thermal cycling stress
- **Reliability assessment**: Combined thermal, mechanical, and moisture effects
- **Delamination prediction**: Interface stress-moisture interactions

Infrastructure

- **Bridge components**: Moisture penetration under traffic loading
- **Protective coatings**: Mechanical damage accelerating moisture ingress
- **Concrete structures**: Chloride transport under structural loads

Multiphysics Simulation Advantages

Physical Realism

- Captures true **bidirectional coupling** between physics domains
- Accounts for **stress-concentration effects** on transport
- Predicts **coupled failure mechanisms**

Engineering Insight

- Identifies **critical stress-moisture combinations**
- Optimizes **loading sequences** to minimize moisture damage
- Guides **material selection** for multiphysics environments

Design Applications

- **Load path optimization**: Minimize stress-assisted diffusion
- **Geometry design**: Avoid stress concentrations in humid environments
- **Maintenance scheduling**: Predict moisture-induced degradation rates

Multiphysics Extensions

Potential Enhancements

1. **Thermal coupling**: Temperature-dependent properties
2. **Chemical reactions**: Moisture-induced chemical degradation
3. **Damage mechanics**: Crack-assisted diffusion pathways
4. **Hygroscopic swelling**: Concentration-induced mechanical strains

Advanced Multiphysics Models

```
∂c/∂t = ∇·[D(c,T,σ)∇c] + R_chemical(c,T)  // Thermo-chemo-mechanical
σ = D(c,T):(ε - εthermal - εhygro - εchemical)  // Full coupling
```

Limitations

1. **Linear elasticity**: No plasticity or damage
2. **Constant properties**: Temperature and molar volume are fixed
3. **Single-phase diffusion**: No vapor/liquid phase transitions
4. **No chemical reactions**: Pure physical diffusion

Project Structure and File Organization

Recommended Folder Structure

```
Abaqus_Diffusion_UEL/
├── Diffusion_3D.for          # Main UEL subroutine (18 KB)
├── VisualMesh.m               # MATLAB visualization script (2 KB)
├── Documentation.pdf          # Complete documentation (859 KB)
├── README.txt                 # Setup instructions
├── Examples/                  # Test cases and validation
│   ├── simple_cube/
│   ├── tension_test/
│   └── validation_cases/
├── Input_Files/               # ABAQUS input templates
│   ├── template.inp
│   └── material_properties/
└── Results/                   # Analysis outputs
    ├── ODB_files/
    └── Post_processing/
```

Core Files Description

Diffusion_3D.for (18 KB)

- Main FORTRAN subroutine containing the UEL implementation
- Contains all multiphysics coupling logic
- Includes supporting subroutines (kshapefcn, kjacobian, kumat, etc.)
- Ready for compilation with ABAQUS

VisualMesh.m (2 KB)

- MATLAB script for generating visualization elements
- Processes ABAQUS input files automatically
- Creates offset element numbering for post-processing
- Essential for result visualization

Documentation.pdf (859 KB)

- Comprehensive technical documentation
- Theory, implementation details, and usage guide
- Multiphysics formulation explanation
- Setup and execution instructions

Project Setup Instructions

1. Working Directory Setup

```bash
# Navigate to your project folder
cd C:\Users\betim\Documents\Documents\Abaqus_Diffusion_UEL
```

2. ABAQUS Environment Configuration

Ensure ABAQUS can find your UEL file:

```bash
# Set ABAQUS working directory
set ABAQUS_USER_SUBROUTINE=Diffusion_3D.for
```

3. MATLAB Path Configuration

For the visualization script:

```matlab
% Add project folder to MATLAB path
addpath('C:\Users\betim\Documents\Documents\Abaqus_Diffusion_UEL');
```

File Dependencies

Compilation Dependencies

```
Diffusion_3D.for → ABAQUS Standard → UEL Object
     ↓
Intel Fortran Compiler (required)
```

Visualization Dependencies

```
Input File (.inp) → VisualMesh.m → VisualMesh.inp → ABAQUS Viewer
```

Documentation Dependencies

```
Documentation.pdf ← Complete technical reference
README.txt ← Quick setup guide
```

ABAQUS Setup and Execution Guide

Step-by-Step Setup Procedure

Step 1: Material Properties Setup

In ABAQUS/CAE, define material properties:

```
General → Depvar: 11
General → User Material:
  - Mechanical Constants = 0
General → Density = 1
General → Thermal:
  - Specific Heat = 1
```

Note: The `Depvar: 11` allocates space for solution-dependent variables (concentration DOF).

Step 2: Output Requests

Define field output variables:

```
Field Output:
  - SDV (Solution Dependent Variables)
  - Displacement (U)
  - Reaction Forces (RF)
  - Nodal Temperature (NT)
```

Important: Temperature field is used to represent moisture concentration in post-processing.

Step 3: Step Definition

Configure analysis step for **multiphysics simulation**:

```
Step Type: Coupled Temperature-Displacement
Incrementation: Fixed
  (Ensures stable multiphysics coupling)
Other: "Ramp linearly over step"
  (For prescribed boundary conditions)
```

Multiphysics Considerations:

- **Coupled solution**: Simultaneous solution of both physics domains
- **Convergence criteria**: Must satisfy equilibrium in both mechanical and diffusion fields
- **Time stepping**: Controlled by diffusion time scales

Step 4: Amplitude Definition

Create amplitude for loading:

```
Amplitude Type: Tabular
Time    Amplitude
0       1
100000000   1
```

Purpose: Maintains constant amplitude throughout the analysis.

Step 5: Initial Moisture Conditions

If initial moisture content is required:

```
Predefined Field Type: Temperature
```

Note: Temperature field represents initial moisture concentration distribution.

Step 6: Mesh Generation

```
1. Create mesh using standard ABAQUS meshing tools
2. Use C3D20R elements (20-node quadratic brick with reduced integration)
3. Write input file
```

Step 7: Visualization Mesh Creation

Create a separate visualization mesh using the provided MATLAB script:

Method 1: Manual Creation

```
Create → Mesh → Copy mesh
Name: VisualMesh
```

Method 2: MATLAB Script (Recommended) Use the provided `VisualMesh.m` script to automatically generate visualization elements.

Step 8: Element Type Replacement

In the generated input file, replace:

```fortran
*Element, type=C3D20R
```

with:

```fortran
*User element, nodes=20, type=U1, properties=3, coordinates=3, var=128
1,2,3
1,11
*ELEMENT, TYPE=U1, ELSET=SOLID
```

Explanation:

- `nodes=20`: 20-node element
- `type=U1`: User element type identifier
- `properties=3`: Number of material properties
- `coordinates=3`: 3D coordinates
- `var=128`: Total state variables (16 × 8 integration points)
- `1,2,3`: Displacement DOF
- `1,11`: Additional DOF (11 represents concentration)

Step 9: UEL Property Definition

Add after connectivity list:

```fortran
*UEL PROPERTY, ELSET=SOLID
210000., 0.3, 0.0127
*Element, type=C3D20R, elset=Visualization
```

Properties Explanation:

- `210000.`: Young's modulus [MPa]
- `0.3`: Poisson's ratio
- `0.0127`: Diffusion coefficient [mm^2/s]

Step 10: Visualization Mesh Content

Right after Step 9, add:

```
Content of visual mesh
```

Purpose: Includes visualization elements for post-processing.

Step 11: Section Assignment Update

Replace:

```fortran
*Solid Section, elset=Set-1, material=Material-1
```

with:

```fortran
*Solid Section, elset=Visualization, material=Material-1
```

Execution Commands

From Project Directory

```bash
# Navigate to project folder
cd C:\Users\betim\Documents\Documents\Abaqus_Diffusion_UEL

# Debug Mode (recommended for development)
abaqus debug -standard -job YourJobName -user Diffusion_3D.for

# Production Run
abaqus -standard -job YourJobName -user Diffusion_3D.for

# With specific input file
abaqus -standard -job MyAnalysis -input MyModel.inp -user Diffusion_3D.for
```

File Management During Analysis

```bash
# Before running analysis
Input Files:     MyModel.inp, Diffusion_3D.for
Generated:       VisualMesh.inp (from MATLAB script)

# After analysis completion
Results:         MyModel.odb, MyModel.dat, MyModel.sta
Log Files:       MyModel.log, MyModel.msg
Temporary:       MyModel.lck, MyModel.sim
```

Project Workflow

Complete Analysis Workflow

1. **Model Preparation** (ABAQUS/CAE)

   ```
   Create geometry → Mesh → Material properties → Boundary conditions
   ```

2. **Input File Modification**

   ```
   Export .inp → Edit for UEL → Add UEL properties
   ```

3. **Visualization Mesh Generation** (MATLAB)

   ```matlab
   % Run in project directory
   VisualMesh  % Execute MATLAB script
   ```

4. **Analysis Execution** (Command Line)

   ```bash
   abaqus -standard -job Analysis -user Diffusion_3D.for
   ```

5. **Post-Processing** (ABAQUS/Viewer)

   ```
   Open .odb → View results → Extract data
   ```

Best Practices for Project Organization

Version Control

```
Abaqus_Diffusion_UEL/
├── v1.0/                    # Stable versions
├── development/             # Work in progress
└── archive/                 # Old versions
```

Analysis Organization

```
Results/
├── 2025_05_31_TensionTest/   # Date-based folders
├── 2025_06_01_ShearTest/
└── Validation/
    ├── Case1_SimpleCube/
    └── Case2_ComplexGeometry/
```

File Naming Convention

```
Analysis files:    ProjectName_LoadCase_Date.inp
Results:          ProjectName_LoadCase_Date.odb
Documentation:    ProjectName_Theory_vX.X.pdf
```

Complete Input File Template

fortran

```
*HEADING
Moisture Diffusion Analysis with UEL

*PREPRINT, ECHO=NO, MODEL=NO, HISTORY=NO

*NODE
! Node definitions...

*USER ELEMENT, NODES=20, TYPE=U1, PROPERTIES=3, COORDINATES=3, VAR=128
1,2,3
1,11

*ELEMENT, TYPE=U1, ELSET=SOLID
! UEL element connectivity...

*UEL PROPERTY, ELSET=SOLID
210000., 0.3, 0.0127

*ELEMENT, TYPE=C3D20R, ELSET=Visualization
! Visualization element connectivity...

*MATERIAL, NAME=Material-1
*DENSITY
1.0
*SPECIFIC HEAT
1.0
*USER MATERIAL, CONSTANTS=0

*SOLID SECTION, ELSET=Visualization, MATERIAL=Material-1

*INITIAL CONDITIONS, TYPE=TEMPERATURE
! Initial moisture distribution if needed...

*STEP, NAME=Diffusion_Step, NLGEOM=NO
*COUPLED TEMPERATURE-DISPLACEMENT, STEADY STATE
*CONTROLS, PARAMETERS=TIME INCREMENTATION
1, 10, , , ,
*CONTROLS, PARAMETERS=FIELD, FIELD=DISPLACEMENT
1e-3, 1e-3, 1e-6, 1e-6, 0.25, , ,
*CONTROLS, PARAMETERS=FIELD, FIELD=TEMPERATURE
1e-3, 1e-3, 1e-6, 1e-6, 0.25, , ,

*BOUNDARY
! Boundary conditions...

*CLOAD
! Mechanical loads...
```

```
*DFLUX
! Moisture flux boundary conditions...

*OUTPUT, FIELD, VARIABLE=PRESELECT
*OUTPUT, HISTORY, VARIABLE=PRESELECT

*END STEP
```

Multiphysics Convergence and Stability

Coupling Strength Assessment

The dimensionless coupling parameter:

```
Π = (Vh·σcharacteristic)/(R·T) = (8000 × σ)/(8314.5 × 300)
```

- **Π << 1**: Weak coupling (diffusion dominant)
- **Π ≈ 1**: Strong coupling (fully multiphysics)
- **Π >> 1**: Stress-dominated transport

Stability Considerations

1. **Time step limits**: Diffusion time scale constraints
2. **Spatial discretization**: Mesh refinement for stress gradients
3. **Coupling iterations**: Monolithic vs. staggered solution schemes

Validation Strategies

1. **Single-physics limits**: Verify pure diffusion (σ = 0)
2. **Analytical benchmarks**: Simple geometries with known solutions
3. **Experimental correlation**: Compare with moisture uptake tests under load

MATLAB Visualization Mesh Generator

The `VisualMesh.m` script automates the creation of visualization elements for post-processing. This is crucial because UEL elements cannot be directly visualized in ABAQUS/Viewer.

Script Purpose

- Reads ABAQUS input file containing UEL elements
- Creates duplicate C3D20R elements with offset numbering
- Enables standard ABAQUS visualization while maintaining UEL computation

Script Functionality

```matlab
%% Script developed to read and modify Abaqus input files %%

clear variables
clc

% Open the input file
fid = fopen('CT.inp','r');
```

Input: `CT.inp` - ABAQUS input file with UEL elements

Algorithm Steps

1. **File Reading**

```matlab
file=textscan(fid,'%s','Delimiter','\n');
flines=(file{1});
lif=length(flines);
```

2. **Element Section Detection**

```matlab
% Search for lines containing *ELEMENT
FFLINES=upper(flines);
Lin_element_cells=strfind(FFLINES,['*' upper('Element')]);
Lin_element_zeros=cellfun(@isempty,Lin_element_cells);
start_elements=find(Lin_element_zeros==0);
```

- Converts file to uppercase for case-insensitive search
- Finds all lines starting with `*ELEMENT`
- Identifies section boundaries

3. **Section Boundary Detection**

```matlab
Lin_order_cells=strfind(FFLINES,'*');
Lin_order_zeros=cellfun(@isempty,Lin_order_cells);
Lin_orders=find(Lin_order_zeros==0);

for i=1:length(start_elements)
    end_elements(i)=Lin_orders(find(Lin_orders==start_elements(i))+1);
```

- Finds all keyword lines (starting with `*`)
- Determines where each element section ends

4. **Element Data Extraction**

```matlab
for i=1:length(start_elements)
    % Check if this is a valid element section
    str_a=FFLINES{start_elements(i)}(~isspace(FFLINES{start_elements(i)}));
    flag=0;
    if length(str_a)==(length('Element')+1)
        flag=1;
    else
        if str_a(length('Element')+2)==','
            flag=1;
        end
    end
```

- Validates element keyword format
- Ensures proper parsing of element definitions

5. **Connectivity Reading**

```matlab
if flag==1
    for b=(start_elements(i)+1):(end_elements(i)-2)
        first=strread(FFLINES{b}, '%f', 'delimiter', ',');
        b=b+1;
        second=strread(FFLINES{b}, '%f', 'delimiter', ',');
        Element{i}(p,:)=vertcat(first,second);
        p=p+1;
    end
    Element{i}(2:2:end,:) = [];
end
```

- Reads element connectivity in pairs of lines
- Combines multi-line element definitions
- Removes duplicate entries

6. **Element Renumbering with Offset**

```matlab
LE=Element{1}; % Element numbering and connectivity
[NUME, ~] = size(LE);
Order=floor(log10(NUME));
offset=10*10^Order;

LEnew=zeros(NUME,21);
for i=1:NUME
    LEnew(i,:)= [(offset+i) LE(i,2) LE(i,3) LE(i,4) LE(i,5) LE(i,6) LE(i,7)...
         LE(i,8) LE(i,9) LE(i,10) LE(i,11) LE(i,12) LE(i,13) LE(i,14) LE(i,15) LE(i,16)...
         LE(i,17) LE(i,18) LE(i,19) LE(i,20) LE(i,21)];
end
```

Offset Calculation:

- For 100 elements: `offset = 10 × 10² = 1000`
- For 1000 elements: `offset = 10 × 10³ = 10000`
- Ensures visualization elements don't conflict with UEL numbering

Element Format:

- Column 1: New element number (original + offset)
- Columns 2-21: Node connectivity (unchanged)

7. **Output Generation**

```matlab
for i=1:length(Element)
    NameFile=strcat('VisualMesh.inp');
    dlmwrite(NameFile,LEnew,'precision', 8);
end

fclose all;
```

Usage Instructions

1. **Prepare Input File**

   ```
   - Save ABAQUS model as 'CT.inp'
   - Ensure it contains UEL element definitions
   ```

2. **Run MATLAB Script**

   ```
   matlab
   ```

   ```
   run('VisualMesh.m')
   ```

3. **Generated Output**

   ```
   File: VisualMesh.inp
   Format: [Element_ID, Node1, Node2, ..., Node20]
   ```

4. **Integration with Main Input File** Add to your ABAQUS input file:

   ```
   fortran
   ```

   ```
   *ELEMENT, TYPE=C3D20R, ELSET=Visualization
   ** Include visualization mesh content here
   *INCLUDE, INPUT=VisualMesh.inp
   ```

Example Output

Original UEL Elements:

```
1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
2, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40
```

Generated Visualization Elements:

```
1001, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
1002, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40
```

Advantages

1. **Automated Process**: No manual element copying required
2. **Consistent Numbering**: Systematic offset prevents conflicts
3. **Scalable**: Works for any number of elements
4. **Error Prevention**: Reduces manual input errors
5. **Maintainability**: Easy to modify for different element types

Script Modifications

For Different Element Types:

```matlab
% Modify for 8-node elements (C3D8R)
LEnew=zeros(NUME,9);  % 9 columns instead of 21
for i=1:NUME
    LEnew(i,:)= [(offset+i) LE(i,2:9)];  % Only first 8 nodes
end
```

For Custom Offset:

```matlab
offset = 50000;  % Fixed offset instead of calculated
```

For Multiple Element Sets:

```matlab
% Process multiple element sets
for set_id=1:length(Element)
    NameFile=sprintf('VisualMesh_Set%d.inp', set_id);
    dlmwrite(NameFile, LEnew{set_id}, 'precision', 8);
end
```

Troubleshooting

Common Issues:

1. **File Not Found**: Ensure `CT.inp` exists in MATLAB working directory
2. **Format Errors**: Check input file has proper ABAQUS format
3. **Memory Issues**: For large models, process elements in chunks
4. **Encoding Problems**: Ensure file uses proper text encoding

Debug Tips:

```matlab
% Add diagnostic output
fprintf('Found %d element sections\n', length(start_elements));
fprintf('Processing %d elements\n', NUME);
fprintf('Using offset: %d\n', offset);
```

Viewing Results

- **Displacement**: Standard U output
- **Stress**: Available through SDV1-SDV6
- **Concentration**: Available through NT (temperature field)
- **Hydrostatic Stress**: Available through SDV14

Data Extraction

```python
# Example Python script for result extraction
from abaqus import *
from abaqusConstants import *

# Open ODB file
odb = openOdb('Input_Interaction_UNTER.odb')

# Extract concentration (stored as temperature)
step = odb.steps['Diffusion_Step']
frame = step.frames[-1]
concentration = frame.fieldOutputs['NT']

# Extract stress components
stress_xx = frame.fieldOutputs['SDV1']
stress_yy = frame.fieldOutputs['SDV2']
# ... etc
```

Recommendations

1. **Parameterize constants**: Make Vh, T, R user-definable
2. **Add nonlinearity**: Implement concentration-dependent diffusion
3. **Include swelling**: Add hygroscopic strain effects
4. **Improve robustness**: Add convergence checks and error handling
5. **Validation**: Test against known analytical solutions for verification