A Domain Specific Language for the Finite Element Interpreter (DSL4FEI)
ECI284 Term Project

José Antonio Abell  [jaabell@ucdavis.edu]

March 13, 2012
Contents

1 Motivation

2 Objectives

3 Implementation

4 Demonstration

5 Concluding remarks
Motivation
Motivation

DSL’s are needed!

FEA analysis model specification (nodes, elements, material, constitutive parameters, etc.) is done with the aid of

- GUI: Graphical user interface (Plaxis, SAP2000, Abaqus, etc.)
- Scripting language
  - Adopt a programming language
    - SAP2000 → OAPI - Visual Basic (MS Excel, MS Access)
    - Abaqus → Python
    - OpenSEES → TCL
    - Comsol → Matlab LiveLink™
  - Create a DSL
    - FEAP
    - ANSYS
    - GTStrudl
    - Plaxis
Motivation

DSL’s are needed!

FEA analysis model specification (nodes, elements, material, constitutive parameters, etc.) is done with the aid of

- GUI: Graphical user interface (Plaxis, SAP2000, Abaqus, etc.)
- Scripting language
  - Adopt a programming language
    - SAP2000 → OAPI - Visual Basic (MS Excel, MS Access, etc.)
    - Abaqus → Python
    - OpenSEES → Tcl
    - Comsol → Matlab LiveLink™
  - Create a DSL
    - FEAP
    - ANSYS
    - GTStrudl
    - Plaxis
Motivation

DSL’s are needed!

FEA analysis model specification (nodes, elements, material, constitutive parameters, etc.) is done with the aid of

- GUI: Graphical user interface (Plaxis, SAP2000, Abaqus, etc.)
- Scripting language
  - Adopt a programming language
    - SAP2000 → OAPI - Visual Basic (MS Excel, MS Access)
    - Abaqus → Python
    - OpenSEES → Tcl
    - Comsol → Matlab LiveLink™
  - Create a DSL
    - FEAP
    - ANSYS
    - GTStrudl
    - Plaxis
Motivation

DSL’s are needed!

FEA analysis model specification (nodes, elements, material, constitutive parameters, etc.) is done with the aid of

- GUI: Graphical user interface (Plaxis, SAP2000, Abaqus, etc.)
- Scripting language
  - Adopt a programming language
    - SAP2000 → OAPI - Visual Basic (MS Excel, MS Access)
    - Abaqus → Python
    - OpenSEES → Tcl
    - Comsol → Matlab LiveLink™
  - Create a DSL
    - FEAP
    - ANSYS
    - GTStrudl
    - Plaxis
DSL’s are needed!

FEA analysis model specification (nodes, elements, material, constitutive parameters, etc.) is done with the aid of

- **GUI: Graphical user interface** (Plaxis, SAP2000, Abaqus, etc.)
- **Scripting language**
  - Adopt a programming language
    - SAP2000 → OAPI - Visual Basic (MS Excel, MS Access)
    - Abaqus → Python
    - OpenSEES → Tcl
    - Comsol → Matlab LiveLink™
  - Create a DSL
    - FEAP
    - ANSYS
    - GTStrudl
    - Plaxis
DSL’s are needed!

FEA analysis model specification (nodes, elements, material, constitutive parameters, etc.) is done with the aid of

- GUI: Graphical user interface (Plaxis, SAP2000, Abaqus, etc.)
- Scripting language
  - Adopt a programming language
    - SAP2000 → OAPI - Visual Basic (MS Excel, MS Access)
    - Abaqus → Python
    - OpenSEES → Tcl
    - Comsol → Matlab LiveLink™
  - Create a DSL
    - FEAP
    - ANSYS
    - GTStrudl
    - Plaxis
DSL’s are needed!

FEA analysis model specification (nodes, elements, material, constitutive parameters, etc.) is done with the aid of

- GUI: Graphical user interface (Plaxis, SAP2000, Abaqus, etc.)
- Scripting language
  - Adopt a programming language
    - SAP2000 → OAPI - Visual Basic (MS Excel, MS Access)
    - Abaqus → Python
    - OpenSEES → Tcl
    - Comsol → Matlab LiveLink™
  - Create a DSL
    - FEAP
    - ANSYS
    - GTStrudl
    - Plaxis
DSL’s are needed!

FEA analysis model specification (nodes, elements, material, constitutive parameters, etc.) is done with the aid of

- **GUI**: Graphical user interface (Plaxis, SAP2000, Abaqus, etc.)
- **Scripting language**
  - Adopt a programming language
    - SAP2000 → OAPI - Visual Basic (MS Excel, MS Access)
    - Abaqus → Python
    - OpenSEES → Tcl
    - Comsol → Matlab LiveLink™
  - Create a DSL
    - FEAP
    - ANSYS
    - GTStrudl
    - Plaxis
Motivation

DSL’s are needed!

FEA analysis model specification (nodes, elements, material, constitutive parameters, etc.) is done with the aid of

- GUI: Graphical user interface (Plaxis, SAP2000, Abaqus, etc.)
- Scripting language
  - Adopt a programming language
    - SAP2000 → OAPI - Visual Basic (MS Excel, MS Access)
    - Abaqus → Python
    - OpenSEES → Tcl
    - Comsol → Matlab LiveLink™
  - Create a DSL
    - FEAP
    - ANSYS
    - GTStrudl
    - Plaxis
Motivation

DSL’s are needed!

FEA analysis model specification (nodes, elements, material, constitutive parameters, etc.) is done with the aid of

- GUI: Graphical user interface (Plaxis, SAP2000, Abaqus, etc.)
- Scripting language
  - Adopt a programming language
    - SAP2000 $\rightarrow$ OAPI - Visual Basic (MS Excel, MS Access)
    - Abaqus $\rightarrow$ Python
    - OpenSEES $\rightarrow$ Tcl
    - Comsol $\rightarrow$ Matlab LiveLink™
  - Create a DSL
    - FEAP
    - ANSYS
    - GTStrudl
    - Plaxis
DSL’s are needed!

FEA analysis model specification (nodes, elements, material, constitutive parameters, etc.) is done with the aid of

- **GUI**: Graphical user interface (Plaxis, SAP2000, Abaqus, etc.)
- **Scripting language**
  - Adopt a programming language
    - SAP2000 → OAPI - Visual Basic (MS Excel, MS Access)
    - Abaqus → Python
    - OpenSEES → Tcl
    - Comsol → Matlab LiveLink™
  - Create a DSL
    - FEAP
    - ANSYS
    - GTStrudl
    - Plaxis
DSL’s are needed!

FEA analysis model specification (nodes, elements, material, constitutive parameters, etc.) is done with the aid of

- GUI: Graphical user interface (Plaxis, SAP2000, Abaqus, etc.)
- Scripting language
  - Adopt a programming language
    - SAP2000 → OAPI - Visual Basic (MS Excel, MS Access)
    - Abaqus → Python
    - OpenSEES → Tcl
    - Comsol → Matlab LiveLink™
  - Create a DSL
    - FEAP
    - ANSYS
    - GTStrudl
    - Plaxis
FEI - The Finite Element Interpreter

- FEA code maintained by the UCD CompGeoMech group.
- Branch from OpenSEES (MOSS).
- Removed TCL integration.
- Integrated new advanced constitutive model and elements for geotech modelling.
- Modelling now done in C++ via the API.
FEI - The Finite Element Interpreter

- FEA code maintained by the UCD CompGeoMech group.
- Branch from OpenSEES (MOSS).
  - Removed TCL integration.
  - Integrated new advanced constitutive model and elements for geotech modelling.
  - Modelling now done in C++ via the API.
FEI - The Finite Element Interpreter

- FEA code maintained by the UCD CompGeoMech group.
- Branch from OpenSEES (MOSS).
- Removed TCL integration.
- Integrated new advanced constitutive model and elements for geotech modelling.
- Modelling now done in C++ via the API.
Motivation

FEI - The Finite Element Interpreter

- FEA code maintained by the UCD CompGeoMech group.
- Branch from OpenSEES (MOSS).
- Removed TCL integration.
- Integrated new advanced constitutive model and elements for geotech modelling.
- Modelling now done in C++ via the API.
FEI - The Finite Element Interpreter

- FEA code maintained by the UCD CompGeoMech group.
- Branch from OpenSEES (MOSS).
- Removed TCL integration.
- Integrated new advanced constitutive model and elements for geotech modelling.
- Modelling now done in C++ via the API.
Input specification - The API

Example modelling with the api: Drained triaxial compression test.
Input specification - The API

Example modelling with the api.

```c
#include "../../DSL/CPPIncludes.h"

int main(int argc, char **argv)
{
    // ************************* Defining Material Models *****************************
    int MaterialNumber = 1;
    int Algorithm = 1; // 0: Explicit 1: Implicit

    double rho = 0.0;
    double e0 = 0.8;
    double M = 1.0;
    double lambda = 0.15;
    double kappa = 0.05;
    double PoissonRatio = 0.3;
    double Kc = 10000.0;
    double p0 = 25;

    add_material_NDMaterial_camclay(MaterialNumber, Algorithm, rho, e0, M, lambda, kappa, PoissonRatio, Kc, p0);

    // Defining Nodes
    add_node(1, 3, 1.0, 1.0, 1.0);
    add_node(2, 3, 0.0, 1.0, 1.0);
    add_node(3, 3, 0.0, 0.0, 1.0);
    add_node(4, 3, 1.0, 0.0, 1.0);
    add_node(5, 3, 1.0, 1.0, 0.0);
    add_node(6, 3, 0.0, 1.0, 0.0);
    add_node(7, 3, 0.0, 0.0, 0.0);
    add_node(8, 3, 1.0, 0.0, 0.0);
}
```
Input specification - The API

Example modelling with the api.

```c
// Defining Elements
add_element_brick_8node(1, 1, 2, 3, 4, 5, 6, 7, 8, 1, 1);

add_support_to_node(2, 1);
add_support_to_node(3, 1);
add_support_to_node(3, 2);
add_support_to_node(4, 2);
add_support_to_node(5, 3);
add_support_to_node(6, 1);
add_support_to_node(6, 3);
add_support_to_node(8, 2);
add_support_to_node(8, 3);
add_support_to_all_dofs_of_node(7);

// LOADS
add_path_series_load_pattern(1, 1.0, 1.0, "ts.txt");

// Confinement
add_load_to_node(1, 1, 1, -0.25, -0.25, -0.25);
add_load_to_node(1, 2, 2, 0.0, -0.25, -0.25);
add_load_to_node(1, 3, 3, 0.0, 0.0, -0.25);
add_load_to_node(1, 4, 4, -0.25, 0.0, -0.25);
add_load_to_node(1, 5, 5, -0.25, -0.25, 0.0);
add_load_to_node(1, 6, 6, 0.0, -0.25, 0.0);
add_load_to_node(1, 7, 8, -0.25, 0.0, 0.0);
```
Input specification - The API

Example modelling with the api.

```c
// Deviator
add_load_to_node(2, 8, 1, 0.0, 0.0, -0.25);
add_load_to_node(2, 9, 2, 0.0, 0.0, -0.25);
add_load_to_node(2, 10, 3, 0.0, 0.0, -0.25);
add_load_to_node(2, 11, 4, 0.0, 0.0, -0.25);

// OUTPUT
define_output_of_node_all_dofs_to_file(1, "Node1Displacement_UMF.out", "disp", 0.0, 0);
define_output_of_node_all_dofs_to_file(1, "Node1Force_UMF.out", "unbalance", 0.0, 0);
define_output_of_element_to_file(1, "elementstress_UMF.out", "stress", true, 0.0);
define_output_of_element_to_file(1, "elementstrain_UMF.out", "strain", true, 0.0);

define_constraint_handler_plain_for_analysis();
define_numberer_rcm_for_analysis();
define_integrator_loadcontrol_for_static_analysis(0.005);
define_convergence_test_normdisplacementincrement_for_analysis(1e-4, 2000, 1);
define_algorithm_newton_for_analysis();
define_solver_profilespd_for_analysis();

analyze_static_multistep (800);

exit(0);
```
Results

Loading
Pros / Cons

Pros,
- **Speed (big models)**
- Very complex analysis cases can be specified
- API approach makes it embeddable in any software.

Cons,
- Must compile source code and link with FEI libraries every time there is a change.
- C++ syntax can be cumbersome.
- Can be difficult to catch mistakes (ie. units, programming, etc.)
- Error reports can be cryptic.
- Difficult to gather information about the model in execution time.
Pros / Cons

Pros,

- Speed (big models)
- Very complex analysis cases can be specified
- API approach makes it embeddable in any software.

Cons,

- Must compile source code and link with FEI libraries every time there is a change.
- C++ syntax can be cumbersome.
- Can be difficult to catch mistakes (ie. units, programming, etc.)
- Error reports can be cryptic.
- Difficult to gather information about the model in execution time.
Pros / Cons

Pros,
- **Speed (big models)**
- **Very complex analysis cases can be specified**
- **API approach makes it embeddable in any software.**

Cons,
- Must compile source code and link with FEI libraries every time there is a change.
- **C++ syntax can be cumbersome.**
- Can be difficult to catch mistakes (ie. units, programming, etc.)
- **Error reports can be cryptic.**
- Difficult to gather information about the model in execution time.
Motivation

Pros / Cons

Pros,

- Speed (big models)
- Very complex analysis cases can be specified
- API approach makes it embeddable in any software.

Cons,

- Must compile source code and link with FEI libraries every time there is a change.
- C++ syntax can be cumbersome.
- Can be difficult to catch mistakes (ie. units, programming, etc.)
- Error reports can be cryptic.
- Difficult to gather information about the model in execution time.
Pros / Cons

Pros,
- Speed (big models)
- Very complex analysis cases can be specified
- API approach makes it embeddable in any software.

Cons,
- Must compile source code and link with FEI libraries every time there is a change.
- C++ syntax can be cumbersome.
- Can be difficult to catch mistakes (ie. units, programming, etc.)
- Error reports can be cryptic.
- Difficult to gather information about the model in execution time.
Pros / Cons

Pros,
- Speed (big models)
- Very complex analysis cases can be specified
- API approach makes it embeddable in any software.

Cons,
- Must compile source code and link with FEI libraries every time there is a change.
- C++ syntax can be cumbersome.
- Can be difficult to catch mistakes (i.e. units, programming, etc.)
  - Error reports can be cryptic.
  - Difficult to gather information about the model in execution time.
Pros / Cons

Pros,
- Speed (big models)
- Very complex analysis cases can be specified
- API approach makes it embeddable in any software.

Cons,
- Must compile source code and link with FEI libraries every time there is a change.
- C++ syntax can be cumbersome.
- Can be difficult to catch mistakes (ie. units, programming, etc.)
- Error reports can be cryptic.
- Difficult to gather information about the model in execution time.
Pros / Cons

Pros,

- Speed (big models)
- Very complex analysis cases can be specified
- API approach makes it embeddable in any software.

Cons,

- Must compile source code and link with FEI libraries every time there is a change.
- C++ syntax can be cumbersome.
- Can be difficult to catch mistakes (ie. units, programming, etc.)
- Error reports can be cryptic.
- Difficult to gather information about the model in execution time.
Objectives
Create a domain specific language (DSL) for FEI in the form of a scripting language.

Some requirements:

- Handle variables (double), flow control (if, while, etc.), "include" statements, user functions
- Handle basic math operations and functions
- Familiar syntax (C-like) and easy-to-remember command names
- Handle units and provide unit conversion and unit safety
- Simple to add commands / maintain
- Implement some form of command-line (shell) for interactive input
- Modular
Create a domain specific language (DSL) for FEI in the form of a scripting language.

Some requirements:

- Handle variables (double), flow control (if, while, etc.), “include” statements, user functions
- Handle basic math operations and functions
- Familiar syntax (C-like) and easy-to-remember command names
- Handle units and provide unit conversion and unit safety
- Simple to add commands / maintain
- Implement some form of command-line (shell) for interactive input
- Modular
Create a domain specific language (DSL) for FEI in the form of a scripting language.

Some requirements:
- Handle variables (double), flow control (if, while, etc.), “include” statements, user functions
- Handle basic math operations and functions
- Familiar syntax (C-like) and easy-to-remember command names
- Handle units and provide unit conversion and unit safety
- Simple to add commands / maintain
- Implement some form of command-line (shell) for interactive input
- Modular
Create a domain specific language (DSL) for FEI in the form of a scripting language.

Some requirements:

- Handle variables (double), flow control (if, while, etc.), “include” statements, user functions
- Handle basic math operations and functions
- Familiar syntax (C-like) and easy-to-remember command names
- Handle units and provide unit conversion and unit safety
- Simple to add commands / maintain
- Implement some form of command-line (shell) for interactive input
- Modular
Objectives

DSL4FEI

Create a domain specific language (DSL) for FEI in the form of a scripting language.

Some requirements:

- Handle variables (double), flow control (if, while, etc.), “include” statements, user functions
- Handle basic math operations and functions
- Familiar syntax (C-like) and easy-to-remember command names
- Handle units and provide unit conversion and unit safety
- Simple to add commands / maintain
- Implement some form of command-line (shell) for interactive input
- Modular
Objectives

Create a domain specific language (DSL) for FEI in the form of a scripting language.

Some requirements:

- Handle variables (double), flow control (if, while, etc.), “include” statements, user functions
- Handle basic math operations and functions
- Familiar syntax (C-like) and easy-to-remember command names
- Handle units and provide unit conversion and unit safety
- Simple to add commands / maintain
- Implement some form of command-line (shell) for interactive input
- Modular
Create a domain specific language (DSL) for FEI in the form of a scripting language.

Some requirements:

- Handle variables (double), flow control (if, while, etc.), “include” statements, user functions
- Handle basic math operations and functions
- Familiar syntax (C-like) and easy-to-remember command names
- Handle units and provide unit conversion and unit safety
- Simple to add commands / maintain
- Implement some form of command-line (shell) for interactive input
- Modular
Implementation
Parsers

- Use a *lexer* (Flex) to scan an input file and generate tokens.
- Stream of tokens feed a *parser* (Bison) which looks for patterns and executes code accordingly.
- The “looking for patterns” part is called *parsing* and is specified through the grammar specification of the language.
- The most common way to abstractly represent a grammar y to use the Extended Backus–Naur Form (EBNF).
- Actions associated with patterns are called *semantic actions*.
- Flex and Bison both automatically generate C or C++ code from the grammar + semantic action code.
Parsers

- Use a **lexer** (Flex) to scan an input file and generate tokens.
- Stream of tokens feed a **parser** (Bison) which looks for patterns and executes code accordingly.
- The “looking for patterns” part is called **parsing** and is specified through the grammar specification of the language.
- The most common way to abstractly represent a grammar $y$ to use the Extended Backus–Naur Form (EBNF).
- Actions associated with patterns are called **semantic actions**.
- Flex and Bison both automatically generate C or C++ code from the grammar + semantic action code.
Parsers

- Use a lexer (Flex) to scan an input file and generate tokens.
- Stream of tokens feed a parser (Bison) which looks for patterns and executes code accordingly.
- The “looking for patterns” part is called parsing and is specified through the grammar specification of the language.
- The most common way to abstractly represent a grammar y to use the Extended Backus–Naur Form (EBNF).
- Actions associated with patterns are called semantic actions.
- Flex and Bison both automatically generate C or C++ code from the grammar + semantic action code.
Parsers

- Use a lexer (Flex) to scan an input file and generate tokens.
- Stream of tokens feed a parser (Bison) which looks for patterns and executes code accordingly.
- The “looking for patterns” part is called parsing and is specified through the grammar specification of the language.
- The most common way to abstractly represent a grammar $y$ to use the Extended Backus–Naur Form (EBNF).
  - Actions associated with patterns are called semantic actions.
  - Flex and Bison both automatically generate C or C++ code from the grammar + semantic action code.
Parsers

- Use a *lexer* (Flex) to scan an input file and generate tokens.
- Stream of tokens feed a *parser* (Bison) which looks for patterns and executes code accordingly.
- The “looking for patterns” part is called *parsing* and is specified through the grammar specification of the language.
- The most common way to abstractly represent a grammar $y$ to use the Extended Backus–Naur Form (EBNF).
- Actions associated with patterns are called *semantic actions*.
- Flex and Bison both automatically generate C or C++ code from the grammar + semantic action code.
Parsers

- Use a *lexer* (Flex) to scan an input file and generate tokens.
- Stream of tokens feed a *parser* (Bison) which looks for patterns and executes code accordingly.
- The “looking for patterns” part is called *parsing* and is specified through the grammar specification of the language.
- The most common way to abstractly represent a grammar $y$ to use the Extended Backus–Naur Form (EBNF).
- Actions associated with patterns are called *semantic actions*.
- Flex and Bison both automatically generate C or C++ code from the grammar + semantic action code.
Parsing
Parsing

\[ x = y \times (2 + 3) \]
Parsing

\[ x = y \times (2 + 3) \]
Parsing

\[ x = y \times (2 + 3) \]

Token Stream (lexer)

- name
- assignment
- name
- binary operator
- start group
- number
- binary operator
- number
- end group
Parsing

\[ x = y \times (2 + 3) \]

**Token Stream (lexer)**
- name
- assignment
- name
- binary operator
- start group
- number
- binary operator
- number
- end group

**Parse Tree (parser)**

```
+  
/  
/   
2 3
```
Parsing

\[ x = y \times (2 + 3) \]

Token Stream (lexer)

- name
- assignment
- name
- binary operator
- start group
- number
- binary operator
- expression
- number
- end group

Parse Tree (parser)

- expression
- addition
- number
- number
Parsing

\[ x = y \times (2 + 3) \]

Token Stream (lexer):
- name
- assignment
- name
- binary operator
- start group
- number
- binary operator
- number
- end group

Parse Tree (parser):
- expression
  - \( \times \)
  - \( y \)
  - \( + \)
  - \( 2 \)
  - \( 3 \)
Parsing

\[ x = y \times (2 + 3) \]

Token Stream (lexer)
- name
- assignment
- name
- binary operator
- start group
- number
- binary operator
- number
- end group

Parse Tree (parser)
- expression
  - =
  - X
    - *
      - Y
        - +
          - 2
          - 3
Parsing

$x = y \times (2 + 3)$

Token Stream (lexer)
- name
- assignment
- name
- binary operator
- start group
- number
- binary operator
- number
- end group

Parse Tree (parser)

$\text{expression}$

$=$

$X$

$*$

$Y$

$+$

$2$

$3$
Demonstration
Input specification - with parser

Example modelling with the parser.

```
Algorithm         = 1;
rho               = 0.0 * kg / m^3;
e0                = 0.8;
M                 = 1.0;
lambda            = 0.15;
kappa             = 0.05;
PoissonRatio      = 0.3;
Kc                = 10000.0 * kPa;
p0                = 25 * kPa;

add_material_NDMaterial_camclay(MaterialNumber, Algorithm, rho, e0, M, lambda, kappa, ←
PoissonRatio, Kc, p0);

// Defining Nodes
add_node(1, 3, 1.0*m, 1.0*m, 1.0*m);
add_node(2, 3, 0.0*m, 1.0*m, 1.0*m);
add_node(3, 3, 0.0*m, 0.0*m, 1.0*m);
add_node(4, 3, 1.0*m, 0.0*m, 1.0*m);
add_node(5, 3, 1.0*m, 1.0*m, 0.0*m);
add_node(6, 3, 0.0*m, 1.0*m, 0.0*m);
add_node(7, 3, 0.0*m, 0.0*m, 0.0*m);
add_node(8, 3, 1.0*m, 0.0*m, 0.0*m);

// Defining Elements
add_element_brick_8node(1, 1, 2, 3, 4, 5, 6, 7, 8, 1, 1);
```
Input specification - with parser

Example modelling with the parser.

```cpp
add_support_to_node(2,1);
add_support_to_node(3,1);
add_support_to_node(3,2);
add_support_to_node(4,2);
add_support_to_node(5,3);
add_support_to_node(6,1);
add_support_to_node(6,3);
add_support_to_node(8,2);
add_support_to_node(8,3);
add_support_to_node(7);

// LOADS
add_path_series_load_pattern (1, 1.0 *s, 1.0, "ts.txt" );

// Confinement
add_load_to_node (1, 1, 1, -0.25*KN, -0.25*KN, -0.25*KN);
add_load_to_node (1, 1, 2, 0.0*KN, -0.25*KN, -0.25*KN);
add_load_to_node (1, 1, 3, 0.0*KN, 0.0*KN, -0.25*KN);
add_load_to_node (1, 1, 4, -0.25*KN, 0.0*KN, -0.25*KN);
add_load_to_node (1, 1, 5, -0.25*KN, -0.25*KN, 0.0*KN);
add_load_to_node (1, 1, 6, 0.0*KN, -0.25*KN, 0.0*KN);
add_load_to_node (1, 1, 8, -0.25*KN, 0.0*KN, 0.0*KN);
add_path_series_load_pattern (2, 1.0*s , 1.0, "ts2.txt" );

// Deviator
add_load_to_node (2, 8, 1, 0.0*KN, 0.0*KN, -0.25*KN);
add_load_to_node (2, 8, 2, 0.0*KN, 0.0*KN, -0.25*KN);
add_load_to_node (2, 8, 3, 0.0*KN, 0.0*KN, -0.25*KN);
add_load_to_node (2, 8, 4, 0.0*KN, 0.0*KN, -0.25*KN);
```
Input specification - with parser

Example modelling with the parser.

```plaintext
// OUTPUT
define_output_of_node_all_dofs_to_file(1, "Node1Displacement_UMF.out", "disp", 0.0, 0);
define_output_of_node_all_dofs_to_file(1, "Node1Force_UMF.out", "unbalance", 0.0, 0);
define_output_of_element_to_file(1, "elementstress_UMF.out", "stress", true, 0.0);
define_output_of_element_to_file(1, "elementstrain_UMF.out", "strain", true, 0.0);

define_constraint_handler_plain_for_analysis();
define_numberer_rcm_for_analysis();
define_integrator_loadcontrol_for_static_analysis(5*ms);
define_convergence_test_normdisplacementincrement_for_analysis(0.1*mm, 2000, 1);
define_algorithm_newton_for_analysis();
define_solver_profilespd_for_analysis();

analyze_static_multistep (800);
```
Concluding remarks
To do...

- Add full FEI api (modularized into toolboxes?)
- Support for vectors (tensors) of data to enhance input capabilities
- Manage I/O directly
- Control output units
- Meaningful error reporting
- Parallel processing commands
- Extensive testing and debugging (hopefully some of you will use it)
- Documentation
To do...

- Add full FEI api (modularized into toolboxes?)
- Support for vectors (tensors) of data to enhance input capabilities
  - Manage I/O directly
  - Control output units
  - Meaningful error reporting
  - Parallel processing commands
  - Extensive testing and debugging (hopefully some of you will use it)
- Documentation
To do...

- Add full FEI api (modularized into toolboxes?)
- Support for vectors (tensors) of data to enhance input capabilities
- Manage I/O directly
  - Control output units
  - Meaningful error reporting
  - Parallel processing commands
  - Extensive testing and debugging (hopefully some of you will use it)
- Documentation
To do...

- Add full FEI api (modularized into toolboxes?)
- Support for vectors (tensors) of data to enhance input capabilities
- Manage I/O directly
- Control output units
  - Meaningful error reporting
  - Parallel processing commands
  - Extensive testing and debugging (hopefully some of you will use it)
  - Documentation
Concluding remarks

To do...

- Add full FEI api (modularized into toolboxes?)
- Support for vectors (tensors) of data to enhance input capabilities
- Manage I/O directly
- Control output units
- Meaningful error reporting
  - Parallel processing commands
  - Extensive testing and debugging (hopefully some of you will use it)
- Documentation
To do...

- Add full FEI api (modularized into toolboxes?)
- Support for vectors (tensors) of data to enhance input capabilities
- Manage I/O directly
- Control output units
- Meaningful error reporting
- Parallel processing commands
- Extensive testing and debugging (hopefully some of you will use it)
- Documentation
To do...

- Add full FEI api (modularized into toolboxes?)
- Support for vectors (tensors) of data to enhance input capabilities
- Manage I/O directly
- Control output units
- Meaningful error reporting
- Parallel processing commands
- Extensive testing and debugging (hopefully some of you will use it)
- Documentation
To do...

- Add full FEI api (modularized into toolboxes?)
- Support for vectors (tensors) of data to enhance input capabilities
- Manage I/O directly
- Control output units
- Meaningful error reporting
- Parallel processing commands
- Extensive testing and debugging (hopefully some of you will use it)
- Documentation
Thank you...