Version 1.3

User's Manual

Last Updated: March 3, 2011

Contributors

- Steve Maas (steve.maas@utah.edu)
- Dave Rawlins (rawlins@sci.utah.edu)
- Dr. Jeffrey Weiss (jeff.weiss@utah.edu)
- Dr. Gerard Ateshian (ateshian@columbia.edu)

Contact address
Musculoskeletal Research Laboratories, University of Utah
72 S. Central Campus Drive, Room 2646
Salt Lake City, Utah

Website
MRL: http://mrl.sci.utah.edu
FEBio: http://mrl.sci.utah.edu/software/febio

Forum
http://mrl.sci.utah.edu/forums/

Development of the FEBio project is supported in part by a grant from the U.S. National Institutes of Health.

Table of Contents

Chapter 1 - Introduction

1.1. Overview of FEBio

FEBio is a nonlinear finite element solver that is specifically designed for biomechanical applications. It offers modeling scenarios, constitutive models and boundary conditions that are relevant to many research areas in biomechanics. This chapter briefly describes the available features of FEBio. All features can be used together seamlessly, giving the user a powerful tool for solving 3D problems in computational biomechanics. The software is open-source, and pre-compiled executables for Windows, OS-X and Linux platforms are available for download at http://mrl.sci.utah.edu/software

FEBio supports two analysis types, namely *quasi-static* and *dynamic*. In a *quasi-static* analysis, the (quasi-) static response of the system is sought and the effects of inertia are ignored. In the presence of biphasic materials, a coupled solid-fluid problem is solved. In a dynamic analysis, the inertial effects are included in the governing equations to calculate the time dependent response of the system.

Many nonlinear constitutive models are available, allowing the user to model the often complicated biological tissue behavior. Several isotropic constitutive models are supported such as Neo-Hookean, Mooney-Rivlin, Ogden, Arruda-Boyce and Veronda-Westmann. All these models have a nonlinear stress-strain response and are objective for large deformations. In addition to the isotropic models there are several transversely isotropic constitutive models available. These models exhibit anisotropic behavior in a single preferred direction and are useful for representing biological tissues such as tendons, muscles and other tissues that contain fibers. FEBio also contains a *rigid body* constitutive model. This model can be used to represent materials or structures whose deformation is negligible compared to that of other materials in the overall model. Several constitutive models are available for representing the solid phase of biphasic materials, which are materials that contain both a solid phase and a fluid phase.

FEBio supports a wide range of boundary conditions to model interactions between materials that are relevant to problems in biomechanics. These include prescribed displacements, nodal forces and pressure forces. Deformable models can be connected to rigid bodies. With this feature, the user can model prescribed rotations and torques for rigid segments, thereby allowing the coupling of rigid body mechanics with deformable continuum mechanics. FEBio provides the ability to represent frictionless and frictional contact between rigid and/or deformable materials using sliding interfaces. A sliding surface is defined between two surfaces that are allowed to separate and slide across each other but are not allowed to penetrate. Variations of the sliding interface, such as tied interfaces and rigid walls, are available as well. As of version 1.2 it is also possible to model the fluid flow across two contacting poroelastic materials. Finally, the user may specify a body force to model the effects of, for instance, gravity or base acceleration.

FEBio is a nonlinear implicit FE solver and does not have mesh generation capabilities. Therefore the input files, which are described in detail in this document, need to be generated by preprocessing software. The preferred preprocessor for FEBio is called *PreView*. PreView can convert some other formats to the FEBio input specification. For instance, NIKE3D [1] and

Abaqus input files can be imported to PreView and can be exported from PreView as a FEBio input file. See the PreView User's Manual for more information

1.2. About this document

This document is part of a set of three manuals that accompany FEBio: the User's Manual, describing how to use FEBio (this manual), a Developer's Manual for users who wish to modify or add features to the code, and a Theory Manual, which describes the theory behind the FEBio algorithms.

This document discusses how to use FEBio and describes the input file format in detail. Chapter 2 describes how to run FEBio and explains the various command line options. It also discusses the different files that are required and created by FEBio. Chapter 3 describes the format of the FEBio input file. An XML-based format is used, organizing the data in a convenient hierarchical structure. Chapter 4 discusses the restart capabilities of FEBio. The restart feature allows the user to interrupt a run and continue it at a later time, optionally making changes to the problem data. Chapter 5 describes the multi-step analysis feature, which allows the user to split up the entire analysis into several steps.

For historical reasons, FEBio also supports a fixed input format, which is detailed in Appendix A. This format lists the problem data in a predefined sequential list and is almost identical to the input specification of NIKE3D [1]. Therefore, NIKE3D input files can be used with FEBio, with some limitations as discussed in Appendix A. Users are strongly encouraged to use the XML-based input format as described in Chapter 2.

Although this document describes some of the theoretical aspects of FEBio, a complete theoretical development can be found in the *FEBio Theory Manual*. Developers who are interested in modifying or extending the FEBio code will find the *FEBio Developer's Manual* very useful.

Chapter 2 - Running FEBio

FEBio runs on several different computing platforms including Windows XP, Mac OSX and most versions of Linux. The command line input and output options are described in this chapter.

2.1. Running FEBio on Windows

There are several ways to run FEBio on Windows. The easiest way is by simply selecting the FEBio program from the Programs menu or by double-clicking the FEBio icon in the installation folder. However, this runs FEBio with the installation folder as the working folder and unless the FEBio input files are in this folder you will need to know the relative or absolute path to your input files. A more practical approach is to run FEBio from a command prompt. Before you can do this, you need to know two things: how to open a command prompt and how to add the FEBio installation folder to your PATH environment variable so that you can run FEBio from any folder on your system. The process is slightly different depending on whether you are using Windows XP or Windows 7, so we'll look at the two Windows versions separately.

2.1.1. Windows XP

First, we'll add the FEBio installation folder to the PATH variable. Open the *Control Panel* from the *Start* menu. Switch to *Classic View* and double-click the *System* icon. In the dialog box that appears, select the *Advanced* tab and click the button named *Environment variables*. Find the *path* variable and click the *Edit* button. Add the end of the PATH's value (don't delete the current value) type a semi-colon and then the absolute path to the FEBio installation folder (e.g. C:/Program Files/FEBio/). Then click the *OK*-button on all open dialog boxes.
To open a command prompt, click the *Run* menu item on the *Start* menu. In the dialog box that appears type *cmd* and press the *OK*-button. A command prompt window appears. You can now use the *cd* (change directory) command to navigate to the folder that contains the FEBio input files. To run FEBio, simply type *febio* (with or without additional arguments) and press *Enter*.

2.1.2. Windows 7

Let's first modify the *PATH* environment variable. Open the *Start* menu and type *system* in the search field. From the search results, select the *System* option under *Control Panel*. The *System* window will appear. Find the *Change Settings* option (on the lower, right side) and click it. The System Properties dialog box appears. Activate the *Advanced* tab and click the *Environment Variables* button. Find the *path* variable and click the *Edit* button. Add the end of the PATH's value (don't delete the current value) type a semi-colon and then the absolute path to the FEBio installation folder (e.g. C:/Program Files/FEBio/). Then click the *OK*-button on all open dialog boxes.
Next, open a command prompt as follows. Click the *Start* menu and type *cmd* in the search field. From the search results, select the *cmd* option under Programs. A command prompt window appears. You can now use the *cd* (change directory) command to navigate to the folder that

contains the FEBio input files. To run FEBio, simply type *febio* (with or without additional arguments) and press *Enter*.

2.2. Running FEBio on Linux or MAC

Running FEBio on Linux or Mac is as easy as opening op a shell window and typing FEBio on the command line. However, you may need to do define an alias to the folder that contains the FEBio executable if you want to run FEBio from any folder on your system. Since this depends on your shell, you need to consult your Linux documentation on how to do this. E.g. if you are using c-shell, you can define an alias as follows,

```
alias febio '/path/to/febio/executable/'
```

If you don't want to define this alias every time you open a shell window, you can place it in your shell start up file (e.g. *.cshrc* for c-shell).

2.3. The Command Line

FEBio is started from a shell window (also known as the *command prompt* in Windows). The command line is the same for all platforms:

```
febio [-o1 [name1] | -o2 [name2] | …]
```

Where `-o1`, `-o2` are options and *`name1, name2,`* … are filenames. The different options (of which most are optional) are given by the following list:

- `-i` : name of input file in free or fixed format
- `-r` : restart file name
- `-g` : debug flag (does not require a file name)
- `-p`: plot file name
- `-a`: dump file name
- `-o`: log file name
- `-c`: data check only
- `-nosplash:` don't show the welcome screen
- `-cnf:` configuration filename
- `-noconfig:` don't use the configuration file

A more detailed description of these options follows.

-i

The `-i` option is used to specify the name of the input file. Note that the format of the input file is determined by the file extension. A ".feb" or ".xml" implies that the input file is formatted in the free format as described in Chapter 3. A ".n" extension or no extension indicates that the input file is formatted in the older fixed format as described in appendix A.

Example: > `febio -i input.feb`

-r
The `-r` option allows you to restart a previous analysis. The filename that must follow this option is an FEBio *restart input file* or a *dump file*. The restart input file and dump file are described in more detail in Chapter Chapter 4 - . The `-i` and `-r` options are mutually exclusive; only one of them may appear on the command line.

Example: > `febio -r file.feb`

-g
The `–g` option runs FEBio in *debug mode*. See section 2.8.2 for more information on running FEBio in debug mode.

Example: > `febio -i input.feb –g`

-p
The `–p` option allows the user to specify the name of the *plot file*. The plot file is a binary file that contains the main results of the analysis. FEBio usually provides a default name for this file; however, the user can override the default name using this option. See section 2.7 for more details on the output files for FEBio.

Example: > `febio -i input.feb –p out.plt`

-a
It is possible to restart a previous run using the restart capability in FEBio. This is useful when a run terminates unexpectedly. If that happens, the user can restart the analysis from the last converged timestep. Before this feature can be used, the user must request the creation of a *dump file*. This file will store all the information that FEBio will need to restart the analysis. FEBio will usually provide a default name for the dump file, but the `–a` command line option allows the user to override the default name for the dump file. See section 2.8.3 and chapter 4 for more details on how to use the restart feature.

Example: > `febio -i input.feb –a out.dmp`

-o
The `–o` option allows the user to set the name of the *log file*. The log file will contain a record of the screen output that was generated during a run. FEBio usually provides a default name for this file (see section 2.7), but the user can override it with this command line option.

Example: > `febio -i input.feb –o out.log`

-c

When the `-c` option is specified on the command line, FEBio will only read the input file and check it for possible errors. When the check is complete, FEBio will terminate. See section 2.8.4 for more details on this option.

Example: > `febio -i input.feb –c`

-nosplash
When the –nosplash command is entered on the command line, FEBio will not print the welcome message to the screen. This is useful when calling FEBio from another application and when the user wishes to suppress any screen output from FEBio. Other options for suppressing output can be set in control section of the FEBio input file (see section 3.2.2).

Example: > `febio -i input.feb –nosplash`

-cnf
-noconfig
As of version 1.2, FEBio uses a *configuration file* to store platform specific settings. Usually FEBio assumes that the location for this configuration file is the same as the executable. However, the user can specify a different location using the `-cnf` command line option. If the user does not have a configuration file or does not wish to use one, this can be specified using the `-noconfig` option. More details on the configuration file can be found in section 2.5 and Appendix B.

Example: > `febio -i input.feb –cnf C:\path\to\febio.xml.`

2.4. The FEBio prompt

If you start FEBio without any command arguments, the FEBio prompt will appear (after the welcome message). It will look something like this:

```
febio>
```

You can now enter one of the following commands:

- **help**: prints an overview of available commands with a brief description of each command.
- **quit**: exit FEBio.
- **run**: run an FEBio input file. This command takes the same options as you can enter on the command line. For example, to run a file named *test.feb* from the FEBio prompt, type the following:

  ```
  run –i test.feb
  ```

- **version**: print version information.

You can also bring up the FEBio prompt during a run by pressing ctrl+c. See section 2.8.1 for more details.

2.5. The configuration file

As of version 1.2, FEBio uses a *configuration file* to store platform-specific settings, such as the default linear solver. The configuration file uses an xml format to store data and is detailed in Appendix B. For backward compatibility, it is still possible to run FEBio without the configuration file. In that case, the default settings prior to version 1.2 are used.

Example: > `febio –i myfile.feb -noconfig`

The configuration file needs to be stored in the same location as the executable. Alternatively, the location of the file can also be specified on the command line using the –cnf option.

Example: `febio -i myfile.feb –cnf /home/my/folder/FEBio/febio.cnf`

2.6. Using multiple processors

Version 1.2 of FEBio implements the MKL version of the PARDISO linear solver, which can run on multiple processors. Set the environment variable OMP_NUM_THREADS to the number of threads to be used. For example, on Linux using the Bash shell, execute:

Example: > `export OMP_NUM_THREADS=4`

Or at a Windows command prompt:

Example: `> set OMP_NUM_THREADS=4`

2.7. FEBio output

After running FEBio, two or three files are created: the *log file*, the *plot file* and optionally the *dump file*. The log file is a text file that contains the same output (and usually more) that was written to the screen. The *plot file* contains the results of the analysis. Since this is a binary file, the results must be analyzed using post processing software such as *PostView*. In some cases, the user may wish to request the creation of a *dump file*. This file contains temporary results of the run. If an analysis terminates unexpectedly or with an error, this file can be used to restart the analysis from the last converged time step. See Chapter 4 for more details. The names of these files can be specified with the command options `-p` (plot file), `-a` (dump file), `-o` (log file). If one or more of the file names following these flags are omitted, then the omitted file name(s) will be given a default name. If a standard XML format input file was used, the default file names are derived from the input file name. For example, if the input file name is *input.feb* the logfile will have the name *input.log*, the plot file is called *input.plt* and the dump file is called *input.dmp*. If the input file used the fixed format, the following default names are used: *f3plot* for the plot file, *f3log.txt* for the log file and *f3dmp* for the dump file.

2.8. Advanced Options

2.8.1. Interrupting a run*

The user can pause the run by pressing *ctrl+c*. This will bring up the FEBio prompt, and the user can enter a command. The following commands are available.

- *cont*: continues the run. FEBio will continue the analysis where it left off.
- *conv*: force the current time step to converge. This is useful for example when a time step is having difficulty satisfying too tight of convergence criteria. The user can then manually force the convergence of the time step. However, if the convergence difficulties are due to instabilities, forcing a time step to converge could cause the solution to become unstable or even incorrect. Also be aware that even if the solution recovers on later timesteps, the manually converged step might be incorrect.
- *debug [on|off]*: entering *debug* will toggle debug mode. Adding *on* (*off*) will turn the debug mode on (resp. off). In debug mode, FEBio will store additional information to the log and plot file that could be useful in debugging the run. It is important to note that since FEBio will store all non-converged states to the plot file, this file may become very large in a short number of time steps. See section 2.8.2 for more details on debugging.
- *dtmin*: set the minimum time step size. This command overrides the minimum time step size that was specified in the input file.
- *fail*: stop the current iteration and retry. If the current time step is not converging and if the auto-time stepper is enabled, the fail command will stop the current time step and retry it with a smaller time step size. If the auto-time stepper is not enabled, the fail command will simply exit the application.
- *help*: list the available commands with a short explanation. Prints the information provided in this section (2.4.1) of the manual.
- *plot*: dump current state to plot database and continue. This command is useful when you want to store the non-converged state at the current iteration. Note that this command only stores the state at the current iteration. If you turn on debug mode, all the iterations are stored to the plot file.
- *print*: print values of variables:
  - *nnz*: number of non-zeroes in stiffness matrix
  - *neq*: number of equations
  - *time*: the current time step
- *quit*: exit the application
- *restart*: toggles restart flag. When the restart flag is set, FEBio will create a dump file at the end of each converged time step. This dump file can then later be used to restart the analysis from the last converged time step. See Chapter 4 for more details on FEBio's restart feature.
- *time*: print elapsed time and an estimation of the remaining time.
- *version*: print version information

Note that it may take a while before the FEBio prompt is displayed after the user requests a *ctrl+c* interruption. This may be because the program may be in the middle of a call to the linear solver or another time-consuming part of the analysis procedure.

* This feature may not work properly on all systems, although it will always work on Windows systems.

2.8.2. Debugging a run

As stated in Section 2.3, FEBio can be run in debug-mode by specifying the `-g` option on the command line. When running in debug mode, FEBio performs additional checks and prints out more information to the screen and to the plot file. It will also store all non-converged geometry states to the plot file. These non-converged states can be very useful for determining the cause of non-convergence or slow convergence. Because of this additional work, the problem may run slightly slower. Note that debug mode can be turned on/off while running an analysis by first interrupting the run with *ctrl+c* and then using the *debug* command to toggle the debug mode on or off. It is important to note that since FEBio will store all non-converged states to the plot file, this file may become very large in a short number of time steps. An alternative approach is to use the *plot* command to write out select non-converged states.

2.8.3. Restarting a run

When the creation of a restart file is requested, the analysis can be restarted from the last converged timestep. This is useful when the run terminated unexpectedly or when the user wishes to modify some parameters during the analysis. To request a restart file, simply set the appropriate option in the control section of the input file. This will generate a *dump* file which then can be used to restart the analysis. See Chapter 4 for more details.

To restart an analysis, use the *–r* command line option. This option requires a filename as a parameter, and this name can be either the name of a dump file or the name of a restart input file. The latter case is a text file that allows the user to redefine some parameters when restarting the run. The format of this file is described in Chapter 4.

2.8.4. Input file checking

The *–c* option allows the user to stop FEBio after the initial data checking is done. This way, potential input errors can be spotted without running the actual problem.

Example: `febio –i input.feb –c`

Chapter 3 - Free Format Input

This chapter describes the XML-based input format used by FEBio. Since this format follows standard XML conventions, the files can be viewed with any file viewer that supports XML files. Since the free format input file is a text file, it can be edited with any text editor.

An XML file is composed of a hierarchical list of *elements*. The first element is called the *root element*. Elements can have multiple *child elements*. All elements are enclosed by two *tags:* a tag defining the element and an *end tag*. A simple example of an XML file might look like this.

```
<root>
      <child>
            <subchild> ... </subchild>
      </child>
</root>
```

The *value* of an element is enclosed between the name and the end tag.

```
<element> here is the value </element>
```

Note that the XML format is case-sensitive.

XML elements can also have *attributes* in name/value pairs. The attribute value must always be quoted.

```
<element attr="value">...</element>
```

If an XML element has no value, an abbreviated syntax can be used. The following two lines are identical.

```
<element [attribute list] ></element>
```

or

```
<element [attribute list]/>
```

Comments can be added as follows.

```
<!-- This is a comment -->
```

The first line in the document – the XML declaration – defines the XML version and the character encoding used in the document. An example can be,

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

3.1. Free format overview

The free format organizes the FEBio input data into hierarchical XML elements. The root element is called *febio_spec*. This root element also defines the format version number. This

document describes version 1.1 of the FEBio specification[†]. The root element will therefore be defined as follows.

```
<febio_spec version="1.1">
	<!-- contents of file -->
</febio_spec>
```

The different sections introduced in this chapter are child elements of this root element. The following sections are currently defined:

- *Control*: specifies control and solver parameters.
- *Material*: Specifies the materials used in the problem and the material parameters.
- *Geometry*: Defines the geometry of the problem, such as nodal coordinates and element connectivity.
- *Initial*: Defines initial conditions for dynamic problems, such as initial velocities.
- *Boundary*: Defines the boundary conditions that are applied on the geometry.
- *Globals*: Defines the global variables in the scene, e.g. body force.
- *LoadData*: Defines the load curve data.
- *Ouput*: Defines additional data that is to be stored.
- *Step*: Defines different problem steps, where in each problem boundary and initial conditions can be defined.

[†] FEBio continues to read older formats but they are to be considered obsolete.

3.2. Control Section

The control section is defined by the *Control* element. This section defines all parameters that are used to control the evolution of the solution as well as parameters for the nonlinear solution procedure. These parameters are defined as child elements of the *Control* element. The parameters are grouped into two groups: mandatory parameters and optional parameters.

3.2.1. Mandatory Parameters

The following parameters must be specified in every input file.

Parameter	Description
`time_steps`	Total number of time steps that will be taken. (= *ntime*)(1)
`step_size`	The initial time step size. (= *dt*) (1)

Comments:

1. The total running time of the analysis is determined by *ntime* * *dt*. Note that when the auto-time stepper is enabled (see below), the actual number of time steps and time step size may be different than specified in the input file. However, the total running time will always be determined by *ntime* * *dt*.

3.2.2. Optional Parameters

The following parameters are optional. If not specified they are assigned the default values, which are found in the last column.

Parameter	Description	Default
`title`	Title of problem	(none)
`dtol`	Convergence tolerance on displacements (1)	0.001
`etol`	Convergence tolerance on energy (1)	0.01
`rtol`	Convergence tolerance on residual (1)	0 (disabled)
`lstol`	Convergence tolerance on line search (2)	0.9
`time_stepper`	Enable the auto time stepper (3)	(off)
`max_refs`	Max number of stiffness reformations (4)	15
`max_ups`	Max number of BFGS stiffness updates (4)	10
`optimize_bw`	Optimize bandwidth of stiffness matrix (5)	0
`restart`	Generate restart flag (6)	0
`plot_level`	Sets the level of state dumps to the plot file (7)	PLOT_MAJOR_ITRS
`cmax`	Set the max condition number for the stiffness matrix (8)	1e+5
`analysis`	Sets the analysis type (9)	static
`print_level`	Sets the amount of output that is generated on screen (10)	PRINT_MINOR_ITRS
`linear_solver`	Set the linear solver (11)	skyline
`min_residual`	Sets minimal value for residual tolerance (12)	1e-20

Comments:

1. FEBio determines convergence of a time step based on three convergence criteria: displacement, residual and energy (that is residual multiplied by displacement). Each of these criteria requires a tolerance value that will determine convergence when the relative change will drop below this value. For example, a displacement tolerance of ε means that the ratio of the displacement increment (i.e. the solution of the linearized FE equations, $\Delta\mathbf{U} = -\mathbf{K}_k^{-1}\mathbf{R}_k$) norm at the current iteration *k+1* to the norm of the total displacement ($\mathbf{U}_{k+1} = \mathbf{U}_k + \Delta\mathbf{U}$) must be less than ε:

$$\frac{\|\Delta\mathbf{U}\|}{\|\mathbf{U}_{k+1}\|} < \varepsilon$$

   For the residual and energy norms, it is the ratio of the current residual norm (resp. energy norm) to the initial one that needs to be less than the specified convergence tolerance.

   To disable a specific convergence criteria, set the corresponding tolerance to zero. For example, by default, the residual tolerance is zero, so that this convergence criteria is not used.

2. The *lstol* parameter controls the scaling of the vector direction obtained from the line search. A line search method is used to improve the convergence of the nonlinear (quasi-) Newton solution algorithm. After each BFGS or Newton iteration, this algorithm searches in the direction of the displacement increment for a solution that has less energy (that is, residual multiplied with the displacement increment) than the previous iteration. In many problems this will automatically be the case. However, in some problems that are very nonlinear (e.g. contact), the line search can improve convergence significantly. The line search can be disabled by setting the *lstol* parameter to zero, although this is not recommended.

3. If the *time_stepper* parameter is defined it will enable the auto time-stepper, which will adjust the new time step size based on convergence information from the previous time step. The following sub-elements may also be defined, although all are optional. Note that these are sub-elements of the *time_stepper* element and not of the *Control* element.

Parameter	Description	Default
`dtmin`	Minimum time step size	dt/3
`dtmax`	Maximum time step size	dt*3
`max_retries`	Maximum nr. of retries allowed per time step	
`opt_iter`	Optimal number of iterations	11

   The *dtmin* and *dtmax* values are used to constrain the range of possible time step values. The *opt_iter* defines the estimated optimal number of quasi-Newton iterations. If the

actual number of iterations is less than or equal to this value the time step size is increased, otherwise it is decreased.

When a time step fails (e.g. due to a negative jacobian), FEBio will retry the time step with a smaller time step size. The *max_retries* parameter determines the maximum number of times a timestep may be retried before FEBio error terminates. The new time step size is determined by the ratio of the previous time step size and the *max_retries* parameter. For example, if the last time step size is 0.1 and *max_retries* is set to 5, then the time step size is adjusted by 0.02: The first retry will have a step size of 0.08; the next will be 0.06, and so on.

4. The *max_ups* and *max_refs* parameters control the BFGS method that FEBio uses to solve the nonlinear FE equations. In this method the global stiffness matrix is only calculated at the beginning of each time step. For each iteration, a matrix update is then done. The maximum number of such updates is set with *max_ups*. When FEBio reaches this number, it reforms the global stiffness matrix (that is, it recalculates it) and factorizes it, essentially taking a "full Newton" iteration. Then FEBio continues with BFGS iterations. The *max_refs* parameter is used to set the maximum of such reformations FEBio can do, before it fails the timestep. In that case, FEBio will either terminate or, if the auto-time stepper is enabled, retry with a smaller time step size.

   Note that when max_ups is set to 0, FEBio will use the Full-Newton method instead of the BFGS method. In other words, the stiffness matrix is reformed for every iteration. In this case it is recommended to increase the number of *max_refs* (to e.g. 50), since the default value might cause FEBio to terminate prematurely when convergence is slow.

5. The *optimize_bw* parameter enables bandwidth minimization for the global stiffness matrix. This can drastically decrease the memory requirements and running times when using the skyline solver. It is highly recommended when using the skyline solver.

   ```
   <optimize_bw>1</optimize_bw>
   ```

   When using a different linear solver (e.g., pardiso or SuperLU), the bandwidth optimization can still be performed if so desired. However, for these solvers there will be little or no effect since these solvers are not as sensitive to the bandwidth as the skyline solver.

6. The *restart* parameter can be used to generate a restart dump file. To activate it, specify a non-zero value. A filename may be specified as an option. If the filename is omitted, a default name will be provided. Note that this will only generate the binary dump file that is needed to restart the analysis. To override certain parameters before restarting, create a restart input file. FEBio does not generate this file automatically so the user needs to create that file manually. See chapter 4 on the format of the restart input file.

   ```
   <restart file="out.dmp">1</restart>
   ```

7. The *plot_level* allows the user to control exactly when the solution is to be saved to the plot file. The following values are allowed:

Value	Description
PLOT_NEVER	Don't save the solution
PLOT_MAJOR_ITRS	Save the solution after each converged timestep
PLOT_MINOR_ITRS	Save the solution for every quasi-Newton iteration
PLOT_MUST_POINTS	Only save the solution at the must points

8. When the condition number of the stiffness matrix becomes too large, inversions of the stiffness matrix become inaccurate. This will negatively affect the convergence of the quasi-Newton or Newton solution algorithm. FEBio monitors the condition number of the BFGS stiffness update and when it exceeds *cmax* it reforms the stiffness matrix.

```
<cmax>1e5</cmax>
```

9. The *analysis* element sets the analysis type. Currently, FEBio defines two analysis types: (quasi-)static and dynamic. In a quasi-static analysis, inertial effects are ignored and an equilibrium solution is sought. Note that in this analysis mode it is still possible to simulate time dependant effects such as viscoelasticity. In a dynamic analysis the inertial effects are included.

Value	Description
static	(quasi-) static analysis
dynamic	dynamic analysis.

10. The *print_level* allows the user to control exactly how much output is written to the screen. The following values are allowed.

Value	Description
PRINT_NEVER	Don't generate any output
PRINT_PROGRESS	Only print a progress bar
PRINT_MAJOR_ITRS	Only print the converged solution
PRINT_MINOR_ITRS	Print convergence information during equilibrium iterations
PRINT_MINOR_ITRS_EXP	Print additional convergence info during equilib. iterations

11. The linear solver is by default set in the configuration file. Since this allows FEBio to run the same file on different platforms (which may support different linear solvers), this is the recommended way to set the linear solver. However, if the need arises, the user can also override the default linear solver, by explicitly specifying the *linear_solver* parameter in the FEBio input file. The following linear solvers are implemented although they might not all be supported on all platforms. The only solver that is guaranteed to work on all platforms is the skyline solver.

Name	Description	Sym	ASym	MT
skyline	The default skyline solver	●		
pardiso	The preferred solver for most platforms	●	●	●
superlu	The single-threaded SuperLU solver	●	●	
superlu_mt	The multi-threader SuperLU solver	●	●	●
wsmp	The WSMP solver	●	●	●
lusolver	Full-matrix solver	●		

12. If no force is acting on the model, then convergence might be problematic due to numerical noise in the calculations. For example, this can happen in a displacement driven contact problem where one of the contacting bodies is moved before initial contact is made. When this happens, the residual norm will be very small. When it drops below the tolerance set by *min_residual* FEBio will assume that there is no force acting on the system and will converge the time step.

3.3. Material Section

The material section is defined by the *Material* element. This section defines all the materials and material parameters that are used in the model. A material is defined by the *material* child element. This element has two attributes: *id,* which specifies a number that is used to reference the material, and *type*, which specifies the type of the material. The *material* element can also have a third optional attribute called *name*, which can be used to identify the material by a text description. A material definition might look like this:

```
<material id="1" type="isotropic elastic">
```

Or, if the optional *name* attribute is present,

```
<material id="2" type="rigid body" name="femur">
```

The material parameters that have to be entered depend on the material type. The following material types are available.

- *isotropic elastic*
- *neo-Hookean*
- *Mooney-Rivlin*
- *Arruda-Boyce*
- *Ogden*
- *Veronda-Westmann*
- *Holmes-Mow*
- *trans iso Mooney-Rivlin*
- *trans iso Veronda-Westmann*
- *TC nonlinear orthotropic*
- *Ellipsoidal fiber distribution Mooney-Rivlin*
- *muscle material*
- *tendon material*
- *Fung orthotropic*
- *Linear orthotropic*
- *viscoelastic*
- *poroelastic*
- *poroelastic Holmes Mow*
- *poroelastic isotropic Spectral Permeability*
- *poroelastic trans iso Spectral Permeability*
- *poroelastic orthotropic Spectral Permeability*
- *Elipsoidal fiber distribution w/Donnan equilibrium swelling*
- *rigid body*

The following sections describe the material parameters for each of the available constitutive models, along with a short description of each material. A more detailed theoretical description of the constitutive models can be found in the *FEBio Theory Manual*.

3.3.1. Incompressible materials

For materials which are considered to be incompressible, the incompressibility constraint is by default enforced using a penalty method that uses the bulk modulus as the penalty factor. However, the constraint can also be enforced more accurately using an augmented Lagrangian method. To use the augmented Lagrangian method for enforcement of the incompressibility constraint to a user-defined tolerance, the user must define two additional material parameters.

Parameter	Description	Default
`<laugon>`	Turn augmented Lagrangian on for this material or off (1)	0 (off)
`<atol>`	Augmentation tolerance (2)	0.01

Comments:

1. A value of *1* (one) turns augmentation on, where a value of 0 (zero) turns it off.

2. The augmentation tolerance determines the convergence condition that is used for the augmented Lagrangian method: convergence is reached when the relative ratio of the lagrange multiplier norm of the previous augmentation $\|\lambda_k\|$ to the current one $\|\lambda_{k+1}\|$ is less than the specified value:

$$\left| \frac{\|\lambda_{k+1}\| - \|\lambda_k\|}{\|\lambda_{k+1}\|} \right| < \varepsilon .$$

   Thus a value of 0.01 implies that the change in norm between the previous augmentation loop and the current loop is less than 1%.

The augmented Lagrangian method for incompressibility enforcement is available for all materials that are based on an uncoupled hyperelastic strain energy function, namely:

- *Mooney-Rivlin*
- *Veronda-Westmann*
- *trans iso Mooney-Rivlin*
- *trans iso Veronda-Westmann*
- *TC nonlinear orthotropic*
- *muscle material*
- *tendon material*
- *Ogden*
- *Arruda-Boyce*

Example:

```
<material id="1" type="Mooney-Rivlin">
      <c1>5</c1>
      <c2>0.4</c2>
```

```
      <k>10000</k>
      <laugon>1</laugon>        ← turns on augmented Lagrangian iterations
      <atol>0.05</atol>         ← sets the augmentation tolerance
</material>
```

3.3.2. Specifying fiber orientation

Some of the anisotropic materials are modeled as an isotropic matrix in which fibers are embedded. For these materials, the user must specify several parameters related to the fibers, including an initial fiber orientation. This orientation can be specified in several ways. FEBio gives the option to automatically generate the fiber orientation, based on some user-specified parameters. However, the user can override this feature and specify the fiber directions for each element manually in the *ElementData* section. See section 3.4.3 for more details on how to do this.

Transversely Isotropic materials

For transversely isotropic materials this is specified with the *fiber* element. This element takes one attribute, namely *type*, which specifies the type of the fiber generator. The possible values are specified in the following table.

***type* Value**	**Description**
`local`	Use local element numbering (1)
`spherical`	Define the fiber orientation as a constant vector (2)
`vector`	specifies a spherical fiber distribution (3)
`user`	fibers are defined by users in the *ElementData* section. (4)

Note that if the *type* attribute is omitted the fiber distribution will follow the local element nodes 1 and 2. This would be the same as setting the fiber attribute to *local* and setting the value to "1,2".

Comments:

1. In this case, the fiber direction is determined by local element node numbering. The value is interpreted as the local node numbers of the nodes that define the direction of the fiber. The following example defines a local fiber axis by local element nodes 1 and 2. This option is very useful when the local fiber direction can be interpreted as following one of the mesh edges.

   ```
   <fiber type="local">1,2</fiber>
   ```

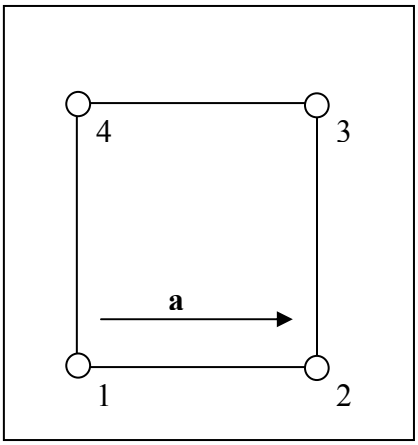

Figure 3-1. *local* fiber direction option.

2. The fiber orientation is determined by a point in space and the global location of each element integration point. The value is the location of the point. The following example defines a spherical fiber distribution centered at [0,0,1].

```
<fiber type="spherical">0,0,1</fiber>
```

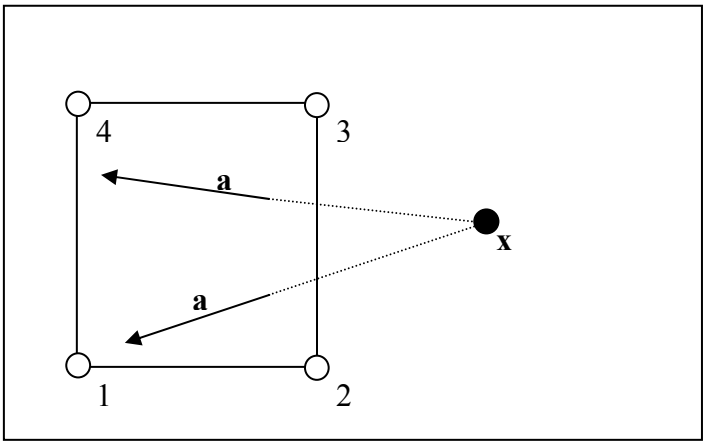

Figure 3-2. spherical fiber direction option.

3. The fiber orientation is specified by a vector. The value is the direction of the fiber. The following defines all element fiber directions in the direction of the vector [1,0,0].

```
<fiber type="vector">1,0,0</fiber>
```

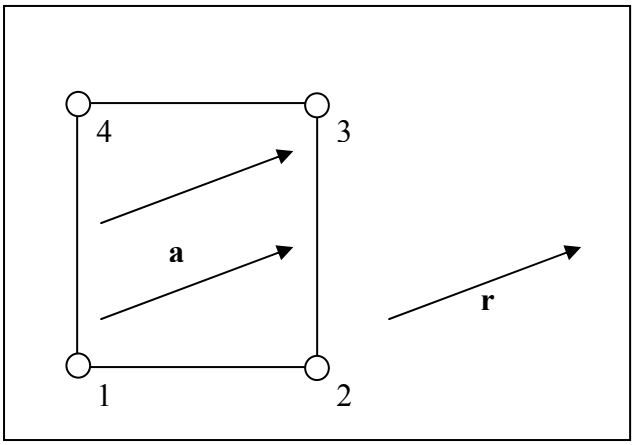

Figure 3-3. *vector* fiber direction option.

4. *user*: the fiber distribution is specified in the *ElementData* section. In this section you can define a different fiber direction for each element separately. See section 3.4 for more details. In this case the value is ignored. This option is useful when experimental data for fiber direction are available.

```
<fiber type="user"></fiber>
```

Orthotropic materials

For orthotropic materials, the user needs to specify two fiber directions **a** and **d**. From these FEBio will generate an orthonormal set of fiber vectors as follows,

$$\mathbf{e}_1 = \frac{\mathbf{a}}{\|\mathbf{a}\|}, \mathbf{e}_2 = \mathbf{e}_3 \times \mathbf{e}_1, \mathbf{e}_3 = \frac{\mathbf{a} \times \mathbf{d}}{\|\mathbf{a} \times \mathbf{d}\|}$$

The vectors **a** and **d** are defined using the *mat_axis* element. This element takes a *type* attribute which can take on the following values.

Value	Description
`local`	Use local element numbering (1)
`vector`	Specify the vectors **a** and **d** directly. (2)

Comments:

1. When specifying *local* as the material axis type, the value is interpreted as a list of three local element node numbers. When specifying zero for all three, the default (1,2,4) is used.

```
<mat_axis type="local">0,0,0</mat_axis>
```

2. When using the *vector* type, you need to define the two generator vectors *a* and *d*. These are specified as child elements of the *mat_axis* element.

```
<mat_axis type="vector">
```

```
      <a>1,0,0</a>
      <d>0,1,0</d>
<mat_axis>
```

3.3.3. Isotropic Elastic

The material type for isotropic elasticity is *isotropic elastic*[‡]. The following material parameters must be defined:

<E>	Young's modulus
<v>	Poisson's ratio

This material is an implementation of a hyperelastic constitutive material that reduces to the classical linear elastic material for small strains, but is objective for large deformations and rotations. The hyperelastic strain-energy function is given by:

$$W = \frac{1}{2}\lambda\left(\operatorname{tr}\mathbf{E}\right)^2 + \mu\mathbf{E}:\mathbf{E}\,.$$

Here, **E** is the Euler-Lagrange strain tensor and λ and μ are the Lamé parameters, which are related to the more familiar Young's modulus E and Poisson's ratio ν as follows:

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \qquad \mu = \frac{E}{2(1+\nu)}\,.$$

It is often convenient to express the material properties using the bulk modulus K and shear modulus G. To convert to Young's modulus and Poisson's ratio, use the following formulas:

$$E = \frac{9KG}{3K+G}, \quad \nu = \frac{3K-2G}{6K+2G}\,.$$

Remark: Note that although this material is objective, it is not advised to use this model for large strains since the behavior may be unphysical. For example, it can be shown that for a uni-axial tension the stress grows unbounded and the volume tends to zero for finite strains. Also for large strains, the Young's modulus and Poisson's ratio input values have little relationship to the actual material parameters. Therefore it is advisable to use this material only for small strains and/or large rotations. For large strains, it is to best to use some of the other available nonlinear material models described in this chapter.

Example:

```
<material id="1" type="isotropic elastic">
      <E>1000.0</E>
      <v>0.45</v>
</material>
```

[‡] This material replaces the now-obsolete *linear elastic* and *St.Venant-Kirchhoff* materials. These materials are still available for backward compatibility although it is recommended to use the *isotropic elastic* material instead.

3.3.4. Neo-Hookean

The material type for a Neo-Hookean material is *neo-Hookean*. The following parameters must be defined:

<E>	Young's modulus
<v>	Poisson's ratio

This model describes a compressible Neo-Hookean material [2]. It has a non-linear stress-strain behavior, but reduces to the classical linear elasticity model for small strains *and* small rotations. It is derived from the following hyperelastic strain-energy function:

$$W = \frac{\mu}{2}\left(I_1 - 3\right) - \mu \ln J + \frac{\lambda}{2}\left(\ln J\right)^2 .$$

Here, I_1 and I_2 are the first and second invariants of the right Cauchy-Green deformation tensor **C** and J is the determinant of the deformation gradient tensor.

This material model uses a standard displacement-based element formulation, so care must be taken when modeling materials with nearly-incompressible material behavior to avoid element locking. For the latter case it is recommended to use the *Mooney-Rivlin* material.

Example:

```
<material id="1" type="neo-Hookean">
      <E>1000.0</E>
      <v>0.45</v>
</material>
```

3.3.5. Mooney-Rivlin

The material type for incompressible Mooney-Rivlin materials is *Mooney-Rivlin*. The following material parameters must be defined:

<c1>	Coefficient of first invariant term
<c2>	Coefficient of second invariant term
<k>	Bulk modulus

This material model is a hyperelastic Mooney-Rivlin type with uncoupled deviatoric and volumetric behavior. The strain-energy function is given by:

$$W = C_1\left(\tilde{I}_1 - 3\right) + C_2\left(\tilde{I}_2 - 3\right) + \frac{1}{2}K\left(\ln J\right)^2.$$

C_1 and C_2 are the Mooney-Rivlin material coefficients. The variables $\tilde{I}_1$ and $\tilde{I}_2$ are the first and second invariants of the deviatoric right Cauchy-Green deformation tensor $\tilde{\mathbf{C}}$. The coefficient K is a bulk modulus-like penalty parameter and J is the determinant of the deformation gradient tensor. When $C_2 = 0$, this model reduces to an uncoupled version of the neo-Hookean constitutive model.

This material model uses a three-field element formulation, interpolating displacements as linear field variables and pressure and volume ratio as piecewise constant on each element [3].

This material model is useful for modeling certain types of isotropic biological tissues. For example, the isotropic collagen matrix can be described quite well with this material model.

Example:

```
<material id="2" type="Mooney-Rivlin">
      <c1>10.0</c1>
      <c2>20.0</c2>
      <k>1000</k>
</material>
```

3.3.6. Ogden

This material describes an incompressible hyperelastic Ogden material [3]. The following material parameters must be defined:

<c[n]>	Coefficient of nth term, where n can range from 1 to 6
<m[n]>	Exponent of nth term, where n can range from 1 to 6
<k>	Bulk modulus

The uncoupled hyperelastic strain energy function for this material is given in terms of the eigenvalues of the deformation tensor:

$$W\left(\tilde{\lambda}_1,\tilde{\lambda}_2,\tilde{\lambda}_3\right)=\sum_{i=1}^{N}\frac{c_i}{m_i^2}\left(\tilde{\lambda}_1^{m_i}+\tilde{\lambda}_2^{m_i}+\tilde{\lambda}_3^{m_i}-3\right)+U\left(J\right).$$

Here, $\tilde{\lambda}_i^2$ are the eigenvalues of the deviatoric right Cauchy deformation tensor, c_i and m_i are material coefficients and N ranges from 1 to 6. Note that you only have to include the material parameters for the terms you intend to use.

Example:

```
<material id="1" type="Ogden">
      <m1>2.4</m1>
      <c1>1</c1>
      <k>100</k>
</material>
```

3.3.7. Ogden Unconstrained

This material describes a compressible (unconstrained) hyperelastic Ogden material [3]. The following material parameters must be defined:

<c[n]>	Coefficient of nth term, where n can range from 1 to 6
<m[n]>	Exponent of nth term, where n can range from 1 to 6
<cp>	Bulk-like modulus

The hyperelastic strain energy function for this material is given in terms of the eigenvalues of the right or left stretch tensor:

$$W\left(\lambda_1,\lambda_2,\lambda_3\right)=\frac{1}{2}c_p\left(J-1\right)^2+\sum_{i=1}^{N}\frac{c_i}{m_i^2}\left(\lambda_1^{m_i}+\lambda_2^{m_i}+\lambda_3^{m_i}-3-m_i\ln J\right).$$

Here, λ_i^2 are the eigenvalues of the right or left Cauchy deformation tensor, c_p, c_i and m_i are material coefficients and N ranges from 1 to 6. Note that you only have to include the material parameters for the terms you intend to use.

Example:

```
<material id="1" type="Ogden unconstrained">
      <m1>2.4</m1>
      <c1>1</c1>
      <cp>2</cp>
</material>
```

3.3.8. Arruda-Boyce

This material describes an incompressible Arruda-Boyce model [4]. The following material parameters are required:

mu	initial modulus
N	number of links in chain
k	Bulk modulus

The uncoupled strain energy function for the Arruda-Boyce material is given by:

$$W = \mu \sum_{i=1}^{5} \frac{C_i}{N^{i-1}} \left(\tilde{I}_1^i - 3^i \right) + U(J),$$

where, $C_1 = \frac{1}{2}, C_2 = \frac{1}{20}, C_3 = \frac{11}{1050}, C_4 = \frac{19}{7000}, C_5 = \frac{519}{673750}$ and I_1 the first invariant of the right Cauchy-Green tensor.

This material model was proposed by Arruda and Boyce [4] and is based on an eight-chain representation of the macromolecular network structure of polymer chains. The strain energy form represents a truncated Taylor series of the inverse Langevin function, which arises in the statistical treatment of non-Gaussian chains. The parameter N is related to the locking stretch λ_L, the stretch at which the chains reach their full extended state, by $\lambda_L = \sqrt{N}$.

Example:

```
<material id="1" type="Arruda-Boyce">
      <mu>0.09</mu>
      <N>26.5</N>
      <k>100</k>
</material>
```

3.3.9. Veronda-Westmann

The material type for incompressible Veronda-Westmann materials is *Veronda-Westmann* [5]. The following material parameters must be defined:

<c1>	First VW coefficient
<c2>	Second VW coefficient
<k>	Bulk modulus

This model is similar to the Mooney-Rivlin model in that it also uses an uncoupled deviatoric dilatational strain energy:

$$W = C_1\left[e^{\left(C_2\left(\tilde{I}_1-3\right)\right)}-1\right]-\frac{C_1C_2}{2}\left(\tilde{I}_2-3\right)+U\left(J\right).$$

The dilatational term is identical to the one used in the Mooney-Rivlin model. This model can be used to describe certain types of biological materials that display exponential stiffening with increasing strain. It has been used to describe the response of skin tissue [5].

Example:

```
<material id="2" type="Veronda-Westmann">
      <c1>1000.0</c1>
      <c2>2000.0</c2>
      <k>1000</k>
</material>
```

3.3.10. Holmes-Mow

The material type for the Holmes-Mow material [6] is *Holmes-Mow*. This isotropic hyperelastic material has been used to represent the solid matrix of articular cartilage [6, 7] and intervertebral disc [8]. The following material parameters must be defined:

<E>	Young's modulus
<v>	Poisson's ratio
<beta>	Exponential stiffening coefficient

The coupled hyperelastic strain-energy function for this material is given by [6]:

$$W\left(I_1, I_2, J\right) = \frac{1}{2}c\left(e^Q - 1\right),$$

where I_1 and I_2 are the first and second invariants of the right Cauchy-Green tensor and J is the jacobian of the deformation gradient. Furthermore,

$$Q = \frac{\beta}{\lambda + 2\mu}\left[\left(2\mu - \lambda\right)\left(I_1 - 3\right) + \lambda\left(I_2 - 3\right) - \left(\lambda + 2\mu\right)\ln J^2\right]$$

$$c = \frac{\lambda + 2\mu}{2\beta},$$

and λ and μ are the Lame parameters which are related to the more familiar Young's modulus and Poisson's ratio in the usual manner,

$$\lambda = \frac{E}{\left(1+\nu\right)\left(1-2\nu\right)}$$

$$\mu = \frac{E}{2\left(1+\nu\right)}$$

Example:

```
<material id="3" type="Holmes-Mow">
      <E>1</E>
      <v>0.35</v>
      <beta>0.25</beta>
</material>
```

3.3.11. Transversely Isotropic Mooney-Rivlin

The material type for transversely isotropic Mooney-Rivlin materials is "*trans iso Mooney-Rivlin"*. The following material parameters must be defined:

<c1>	Mooney-Rivlin coefficient 1
<c2>	Mooney-Rivlin coefficient 2
<c3>	Exponential stress coefficient
<c4>	Fiber uncrimping coefficient
<c5>	Modulus of straightened fibers
<k>	Bulk modulus
<lam_max>	Fiber stretch for straightened fibers
<fiber>	Fiber distribution option

This constitutive model can be used to represent a material that has a single preferred fiber direction and was developed for application to biological soft tissues [9-11]. It can be used to model tissues such as tendons, ligaments and muscle. The elastic response of the tissue is assumed to arise from the resistance of the fiber family and an isotropic matrix. It is assumed that the uncoupled strain energy function can be written as follows:

$$W = F_1\left(\tilde{I}_1, \tilde{I}_2\right) + F_2\left(\tilde{\lambda}\right) + \frac{K}{2}\left[\ln\left(J\right)\right]^2 .$$

Here $\tilde{I}_1$ and $\tilde{I}_2$ are the first and second invariants of the deviatoric version of the right Cauchy Green deformation tensor $\tilde{\boldsymbol{C}}$ and $\tilde{\lambda}$ is the deviatoric part of the stretch along the fiber direction ($\tilde{\lambda}^2 = \boldsymbol{a}_0 \cdot \tilde{\boldsymbol{C}} \cdot \boldsymbol{a}_0$, where $\boldsymbol{a}_0$ is the initial fiber direction), and $J = \det\left(\boldsymbol{F}\right)$ is the Jacobian of the deformation (volume ratio). The function F_1 represents the material response of the isotropic ground substance matrix and is the same as the Mooney-Rivlin form specified above, while F_2 represents the contribution from the fiber family. The strain energy of the fiber family is as follows:

$$\begin{aligned}
\tilde{\lambda}\frac{\partial F_2}{\partial \tilde{\lambda}} &= 0, \quad \tilde{\lambda} \le 1 \\
\tilde{\lambda}\frac{\partial F_2}{\partial \tilde{\lambda}} &= C_3\left(e^{C_4\left(\tilde{\lambda}-1\right)} - 1\right), \quad 1 < \tilde{\lambda} < \lambda_m \\
\tilde{\lambda}\frac{\partial F_2}{\partial \tilde{\lambda}} &= C_5 + C_6\tilde{\lambda}, \quad \tilde{\lambda} \ge \lambda_m
\end{aligned} .$$

Here, C_1 and C_2 are the Mooney-Rivlin material coefficients, *lam_max* (λ_m) is the stretch at which the fibers are straightened, C_3 scales the exponential stresses, C_4 is the rate of uncrimping of the fibers, and C_5 is the modulus of the straightened fibers. C_6 is determined from the requirement that the stress is continuous at λ_m.

This material model uses a three-field element formulation, interpolating displacements as linear field variables and pressure and volume ratio as piecewise constant on each element [3].

The fiber orientation can be specified as explained in section 3.3.2**.** Active stress along the fiber direction can be simulated using an active contraction model. To use this feature you need to define the *active_contraction* parameter. This parameter takes an optional attribute, *lc*, which defines the loadcurve. There are also several options.

<ca0>	Intracellular calcium concentration
<beta>	tension-sarcomere length relation constant
<l0>	Unloaded sarcomere length
<refl>	No tension sarcomere length

Example:
This example defines a transversely isotropic material with a Mooney-Rivlin basis. It defines a homogeneous fiber direction and uses the active fiber contraction feature.

```
<material id="3" type="trans iso Mooney-Rivlin">
      <c1>13.85</c1>
      <c2>0.0</c2>
      <c3>2.07</c3>
      <c4>61.44</c4>
      <c5>640.7</c5>
      <k>100.0</k>
      <lam_max>1.03</lam_max>
      <fiber type="vector">1,0,0</fiber>
      <active_contraction lc="1">
            <ca0>4.35</ca0>
            <beta>4.75</beta>
            <l0>1.58</l0>
            <refl>2.04</refl>
      </active_contraction>
</material>
```

3.3.12. Transversely Isotropic Veronda-Westmann

The material type for transversely isotropic Veronda-Westmann materials is "*trans iso Veronda-Westmann"*. The following material parameters must be defined:

<c1>	Veronda-Westmann coefficient 1
<c2>	Veronda-Westmann coefficient 2
<c3>	Exponential stress coefficient
<c4>	Fiber uncrimping coefficient
<c5>	Modulus of straightened fibers
<k>	Bulk modulus
<lam_max>	Fiber stretch for straightened fibers
<fiber>	Fiber distribution option.

This uncoupled hyperelastic material differs from the Transversely Isotropic Mooney-Rivlin model in that it uses the Veronda-Westmann model for the isotropic matrix. The interpretation of the material parameters, except C_1 and C_2 is identical to this material model.

The fiber distribution option is explained in section 3.3.2. An active contraction model can also be defined for this material. See the transversely isotropic Mooney-Rivlin model for more details.

Example:
This example defines a transversely isotropic material model with a Veronda-Westmann basis. The fiber direction is implicitly implied as *local*.

```
<material id="3" type="trans iso Veronda-Westmann">
      <c1>13.85</c1>
      <c2>0.0</c2>
      <c3>2.07</c3>
      <c4>61.44</c4>
      <c5>640.7</c5>
      <lam_max>1.03</lam_max>
</material>
```

3.3.13. Fung Orthotropic

The material type for orthotropic Fung elasticity [12, 13] is "*Fung orthotropic*". The following material parameters must be defined:

<E1>	Young's modulus in the x-direction
<E2>	Young's modulus in the y-direction
<E3>	Young's modulus in the z-direction
<G12>	Shear modulus in the xy-plane
<G23>	Shear modulus in the yz-plane
<G31>	Shear modulus in the xz-plane
<v12>	Poisson's ratio between x- and y-direction
<v23>	Poisson's ratio between y- and z-direction
<v31>	Poisson's ratio between z- and x-direction
<c>	c coefficient
<K>	bulk modulus

The coupled hyperelastic strain energy function is given by [14],

$$W = \frac{1}{2}c\left(e^{Q}-1\right), \tag{0.1}$$

where,

$$Q = c^{-1}\sum_{a=1}^{3}\left[2\mu_a \mathbf{A}_a^0 : \mathbf{E}^2 + \sum_{b=1}^{3}\lambda_{ab}\left(\mathbf{A}_a^0 : \mathbf{E}\right)\left(\mathbf{A}_b^0 : \mathbf{E}\right)\right].$$

Here, $\mathbf{E}$ is the Green-Lagrange strain tensor and $\mathbf{A}_a^0 = \mathbf{a}_a^0 \otimes \mathbf{a}_a^0$ where $\mathbf{a}_a^0$ defines the initial direction of material axis *a*. See section 3.3.2 on how to define the material axis for orthotropic materials. The orthotropic Lame parameters relate to the Young's modulus and Poisson's ratios as follows:

$$\begin{bmatrix} \lambda_{11}+2\mu_1 & \lambda_{12} & \lambda_{13} \\ \lambda_{12} & \lambda_{22}+2\mu_2 & \lambda_{23} \\ \lambda_{13} & \lambda_{23} & \lambda_{33}+2\mu_3 \end{bmatrix} = \begin{bmatrix} 1/E_1 & -\nu_{12}/E_1 & -\nu_{31}/E_3 \\ -\nu_{12}/E_1 & 1/E_2 & -\nu_{23}/E_2 \\ -\nu_{31}/E_3 & -\nu_{23}/E_2 & 1/E_3 \end{bmatrix}^{-1}$$

and

$$\begin{aligned} \mu_1 &= G_{12}+G_{31}-G_{23} \\ \mu_2 &= G_{12}-G_{31}+G_{23} \\ \mu_3 &= -G_{12}+G_{31}+G_{23} \end{aligned},$$

Example:

```
<material id="3" type="Fung orthotropic">
      <E1>1</E1>
      <E2>2</E2>
      <E3>3</E3>
      <v12>0</v12>
      <v23>0</v23>
      <v31>0</v31>
      <G12>1</G12>
```

```
      <G23>1</G23>
      <G31>1</G31>
      <c>1</c>
      <K>0</K>
</material>
```

3.3.14. Linear Orthotropic

The material type for linear orthotropic elasticity is "*linear orthotropic*". The following material parameters must be defined:

<E1>	Young's modulus in the x-direction
<E2>	Young's modulus in the y-direction
<E3>	Young's modulus in the z-direction
<G12>	Shear modulus in the xy-plane
<G23>	Shear modulus in the yz-plane
<G31>	Shear modulus in the xz-plane
<v12>	Poisson's ratio between x- and y-direction
<v23>	Poisson's ratio between y- and z-direction
<v31>	Poisson's ratio between z- and x-direction

Example:

```
<material id="3" type="linear orthotropic">
      <E1>1</E1>
      <E2>2</E2>
      <E3>3</E3>
      <v12>0</v12>
      <v23>0</v23>
      <v31>0</v31>
      <G12>1</G12>
      <G23>1</G23>
      <G31>1</G31>
</material>
```

3.3.15. Constant Permeability Poroelasticity

The material type for constant permeability poroelastic materials is "*poroelastic*". The following material parameters must be defined:

`<perm>`	Hydraulic permeability k
`<solid_id>`	Material ID of the solid phase

This isotropic material model uses the biphasic theory for describing the time-dependent material behavior of materials that consist of both a solid and fluid phase [15, 16]. The user is referred to the *FEBio Theory Manual* for a description of the biphasic theory.

The hydraulic permeability relates the flux of the fluid relative to the solid, $\mathbf{w}$, to the interstitial fluid pressure gradient, ∇p, according to

$$\mathbf{w} = -\mathbf{k}\nabla p$$

where $\mathbf{k}$ is the hydraulic permeability tensor. (This relation does not account for the contribution of external body forces on the fluid.) When the permeability is isotropic,

$$\mathbf{k} = k\,\mathbf{I}$$

For this material model, k is constant. Therefore this model should be used only when strains are small.

The solid_id parameter is the material number that is used to describe the solid matrix. In other words, to use this material model you need to define at least *two* materials: a material to describe the elasticity of the solid matrix, and a poroelastic material to describe its permeability. The solid matrix may be described by any of the available material models except *rigid body* and *poroelastic*.

Example:
This example defines a poroelastic material that uses a neo-Hookean model for the solid matrix.

```
<material id="1" type="neo-Hookean">
      <E>1.0</E>
      <v>0.4</v>
</material>
<material id="2" type="poroelastic">
      <perm>0.001</perm>
      <solid_id>1</solid_id>
</material>
```

In this example, poroelastic elements must be associated with material id="2".

3.3.16. Poroelastic Holmes-Mow

The material type for Holmes-Mow poroelasticity is *"poroelastic Holmes-Mow"*. The following material parameters need to be defined:

<perm>	isotropic hydraulic permeability k_0 in the reference state
<phi0>	solid volume fraction φ_0 in the reference state ($0 \le \varphi_0 \le 1$)
<M>	exponential strain-dependence coefficient M ($M \ge 0$)
<alpha>	power-law exponent α ($\alpha \ge 0$)
<solid_id>	material ID of the solid phase

This isotropic material is similar to the poroelastic material described above, except that it uses a strain-dependent permeability tensor [6]:

$$\mathbf{k} = k(J)\mathbf{I},$$

where,

$$k(J) = k_0 \left(\frac{J-\varphi_0}{1-\varphi_0} \right)^{\alpha} e^{\frac{1}{2}M\left(J^2-1\right)},$$

and J is the jacobian of the deformation, i.e. $J = \det \mathbf{F}$ where $\mathbf{F}$ is the deformation gradient.

Example:
This example defines a poroelastic material that uses a neo-Hookean model for the solid phase.

```
<material id="1" type="neo-Hookean">
      <E>1.0</E>
      <v>0</v>
</material>
<material id="2" type="poroelastic Holmes-Mow">
      <perm>0.001</perm>
      <phi0>0.2</phi0>
      <M>1.5</M>
      <alpha>2</alpha>
      <solid_id>1</solid_id>
</material>
```

3.3.17. Referentially Isotropic Permeability

The material type for a poroelastic material with strain-dependent permeability which is isotropic in the reference configuration is *"ref iso perm"*. The following material parameters need to be defined:

<perm0>	hydraulic permeability k_{0r}
<perm1>	hydraulic permeability k_{1r}
<perm2>	hydraulic permeability k_{2r}
<phi0>	solid volume fraction φ_r^s in the reference state ($0 \le \varphi_r^s \le 1$)
<M>	exponential strain-dependence coefficient M ($M \ge 0$)
<alpha>	power-law exponent α ($\alpha \ge 0$)
<solid_id>	material ID of the solid phase

This material uses a strain-dependent permeability tensor that accommodates strain-induced anisotropy:

$$\mathbf{k} = \left(k_{0r}\mathbf{I} + \frac{k_{1r}}{J^2}\mathbf{b} + \frac{k_{2r}}{J^4}\mathbf{b}^2 \right) \left(\frac{J - \varphi_r^s}{1 - \varphi_r^s} \right) e^{M\left(J^2-1\right)/2},$$

where J is the jacobian of the deformation, i.e. $J = \det \mathbf{F}$ where $\mathbf{F}$ is the deformation gradient, and $\mathbf{b} = \mathbf{F} \cdot \mathbf{F}^T$ is the left Cauchy-Green tensor. Note that the permeability in the reference state ($\mathbf{F} = \mathbf{I}$) is isotropic and given by $\mathbf{k} = \left(k_{0r} + k_{1r} + k_{2r} \right) \mathbf{I}$.

Example:
This example defines a poroelastic material that uses a neo-Hookean model for the solid phase.

```
<material id="1" type="neo-Hookean">
      <E>1.0</E>
      <v>0</v>
</material>
<material id="2" type="ref iso perm">
      <perm0>0.001</perm0>
      <perm1>0.005</perm1>
      <perm2>0.002</perm2>
      <phi0>0.2</phi0>
      <M>1.5</M>
      <alpha>2</alpha>
      <solid_id>1</solid_id>
</material>
```

3.3.18. Referentially Transversely Isotropic Permeability

The material type for a poroelastic material with strain-dependent permeability which is transversely isotropic in the reference configuration is *"ref trans iso perm"*. The following material parameters need to be defined:

<perm0>	isotropic hydraulic permeability k_{0r}
<perm1A>	axial hydraulic permeability k_{1r}^{A}
<perm2A>	axial hydraulic permeability k_{2r}^{A}
<perm1T>	transverse hydraulic permeability k_{1r}^{T}
<perm2T>	transverse hydraulic permeability k_{2r}^{T}
<phi0>	solid volume fraction φ_r^s in the reference state ($0 \le \varphi_r^s \le 1$)
<M0>	isotropic exponential strain-dependence coefficient M_0 ($M_0 \ge 0$)
<MA>	axial exponential strain-dependence coefficient M_A ($M_A \ge 0$)
<MT>	transverse exponential strain-dependence coefficient M_T ($M_T \ge 0$)
<alpha0>	isotropic power-law exponent α_0 ($\alpha_0 \ge 0$)
<alphaA>	axial power-law exponent α_A ($\alpha_A \ge 0$)
<alphaT>	transverse power-law exponent α_T ($\alpha_T \ge 0$)
<solid_id>	material ID of the solid phase

This material uses a strain-dependent permeability tensor that accommodates strain-induced anisotropy:

$$\begin{aligned}\mathbf{k} = {} & k_{0r}\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_0} e^{M_0\left(J^2-1\right)/2}\mathbf{I} \\ & +\left(\frac{k_{1r}^T}{J^2}\left(\mathbf{b}-\mathbf{m}\right)+\frac{k_{2r}^T}{2J^4}\left[2\mathbf{b}^2-\left(\mathbf{m}\cdot\mathbf{b}+\mathbf{b}\cdot\mathbf{m}\right)\right]\right)\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_T} e^{M_T\left(J^2-1\right)/2}, \\ & +\left(\frac{1}{J^2}k_{1r}^A\mathbf{m}+\frac{1}{2J^4}k_{2r}^A\left(\mathbf{m}\cdot\mathbf{b}+\mathbf{b}\cdot\mathbf{m}\right)\right)\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_A} e^{M_A\left(J^2-1\right)/2}\end{aligned}$$

where J is the jacobian of the deformation, i.e. $J = \det\mathbf{F}$ where $\mathbf{F}$ is the deformation gradient, and $\mathbf{b} = \mathbf{F}\cdot\mathbf{F}^T$ is the left Cauchy-Green tensor. $\mathbf{m}$ is a second order tensor representing the spatial structural tensor describing the axial direction, given by

$$\mathbf{m} = \mathbf{F}\cdot\left(\mathbf{V}\otimes\mathbf{V}\right)\cdot\mathbf{F}^T,$$

where $\mathbf{V}$ is a unit vector along the axial direction (defined as described in Section 3.3.2). Note that the permeability in the reference state ($\mathbf{F} = \mathbf{I}$) is given by $\mathbf{k} = \left(k_{0r} + k_{1r}^T + k_{2r}^T\right)\mathbf{I} + \left(k_{1r}^A - k_{1r}^T + k_{2r}^A - k_{2r}^T\right)\left(\mathbf{V}\otimes\mathbf{V}\right)$.

Example:

This example assumes that a constitutive model for the solid elasticity is given in material id=”1”.

```
<material id=”2” type=”ref trans iso perm”>
      <perm0>0.002</perm0>
      <perm1A>0.01</perm1A>
      <perm2A>0.01</perm2A>
      <perm1T>0.001</perm1T>
      <perm2T>0.05</perm2T>
      <phi0>0.2</phi0>
      <M0>1.0</M0>
      <MA>0.5</MA>
      <MT>1.5</MT>
      <alpha0>1.0</alpha0>
      <alphaA>0.5</alphaA>
      <alphaT>2.0</alphaT>
      <solid_id>1</solid_id>
</material>
```

3.3.19. Referentially Orthotropic Permeability

The material type for a poroelastic material with strain-dependent permeability which is orthotropic in the reference configuration is *"ref ortho perm"*. The following material parameters need to be defined:

<perm0>	isotropic hydraulic permeability k_{0r}
<perm1>	hydraulic permeabilities k_{1r}^a along orthogonal directions ($a=1,2,3$)
<perm2>	hydraulic permeabilities k_{2r}^a along orthogonal directions ($a=1,2,3$)
<phi0>	solid volume fraction φ_0^s in the reference state ($0 \le \varphi_0^s \le 1$)
<M0>	isotropic exponential strain-dependence coefficient M_0 ($M_0 \ge 0$)
<M>	orthotropic exponential strain-dependence coefficient M_a ($a=1,2,3$, $M_a \ge 0$)
<alpha0>	isotropic power-law exponent α_0 ($\alpha_0 \ge 0$)
<alpha>	power-law exponent α_a ($a=1,2,3$, $\alpha_a \ge 0$)
<solid_id>	material ID of the solid phase

This material uses a strain-dependent permeability tensor that accommodates strain-induced anisotropy:

$$\mathbf{k} = k_0\mathbf{I} + \sum_{a=1}^{3} k_1^a \mathbf{m}_a + k_2^a\left(\mathbf{m}_a\cdot\mathbf{b} + \mathbf{b}\cdot\mathbf{m}_a\right),$$

where,

$$k_0 = k_{0r}\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_0} e^{M_0\left(J^2-1\right)/2}$$

$$k_1^a = \frac{k_{1r}^a}{J^2}\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_a} e^{M_a\left(J^2-1\right)/2}, \quad a=1,2,3 \ ,$$

$$k_2^a = \frac{k_{2r}^a}{2J^4}\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_a} e^{M_a\left(J^2-1\right)/2}, \quad a=1,2,3$$

J is the jacobian of the deformation, i.e. $J=\det\mathbf{F}$ where $\mathbf{F}$ is the deformation gradient. $\mathbf{m}_a$ are second order tensor representing the spatial structural tensors describing the orthogonal planes of symmetry, given by

$$\mathbf{m}_a = \mathbf{F}\cdot\left(\mathbf{V}_a\otimes\mathbf{V}_a\right)\cdot\mathbf{F}^T, \quad a=1-3,$$

where $\mathbf{V}_a$ are orthonormal vectors normal to the planes of symmetry (defined as described in Section 3.3.2). Note that the permeability in the reference state ($\mathbf{F}=\mathbf{I}$) is given by $\mathbf{k} = k_{0r}\mathbf{I} + \sum_{a=1}^{3}\left(k_{1r}^a + k_{2r}^a\right)\mathbf{V}_a\otimes\mathbf{V}_a$.

Example:
This example defines a poroelastic material that uses a neo-Hookean model for the solid phase.

```
<material id=”1” type=”neo-Hookean”>
      <E>1.0</E>
      <v>0</v>
</material>
<material id=”2” type=”ref ortho perm”>
      <perm0>0.001</perm0>
      <perm1>0.01, 0.02, 0.03</perm1>
      <perm2>0.001, 0.002, 0.003</perm2>
      <phi0>0.2</phi0>
      <M0>0.5</M0>
      <M>1.5, 2.0, 2.5</M>
      <alpha0>1.5</alpha0>
      <alpha>2, 2.5, 3</alpha>
      <solid_id>1</solid_id>
</material>
```

3.3.20. Viscoelastic

The material type for viscoelastic materials is "*viscoelastic*". The following parameters need to be defined:

<t1>-<t6>	relaxation times
<g1>-<g6>	viscoelastic coefficients
<solid_id>	material ID for the elastic component

For a viscoelastic material, the second Piola Kirchhoff stress can be written as follows [9]:

$$\mathbf{S}(t) = \int_{-\infty}^{t} G(t-s)\frac{d\mathbf{S}^e}{ds}ds\,,$$

where $\mathbf{S}^e$ is the elastic stress and G is the relaxation function. It is assumed that the relaxation function is given by the following discrete relaxation spectrum:

$$G(t) = 1 + \sum_{i=1}^{N} \gamma_i \exp(-t/\tau_i),$$

Note that the user does not have to enter all τ_i and γ_i coefficients. Instead, only the values that are used need to be entered. So, if N is 2, only τ_1, τ_2, γ_1 and γ_2 have to be entered.

The *solid_id* parameter is the material ID of the elastic material. The elastic material is defined as a separate material in the FEBio input file. This allows the user to use any of the other hyperelastic materials as the elastic part of the viscoelastic material.

Example:

```
<material id="1" name="Material 1" type="neo-Hookean">
      <E>1</E>
      <v>0.0</v>
</material>
<material id="2" name="Material 2" type="viscoelastic">
      <solid_id>1</solid_id>
      <t1>0.01</t1>
      <g1>0.95</g1>
</material>
```

3.3.21. Rigid Body

A rigid body can be defined using the rigid material model. The material type for a rigid body is "*rigid body*". The following parameters are defined:

<density>	Density of rigid body
<center_of_mass>	Position of the center of mass
<E>	Young's modulus
<v>	Poisson's ratio

If the *center_of_mass* parameter is omitted, FEBio will calculate the center of mass automatically. In this case, a density *must* be specified. If the *center_of_mass* is defined the *density* parameter may be omitted. Omitting both will generate an error message.

The Young's modulus *E* and Poisson ratio *v* currently have no effect on the results. The only place where they are used is in the calculation of a material stiffness for some auto-penalty contact formulation. If you are using contact it is advised to enter sensible values. Otherwise these parameters may be omitted.

The degrees of freedom of a rigid body are initially unconstrained[§]. This implies that a rigid body is free to move in all three directions and rotate about any of the coordinate axes. To constrain a rigid body degree of freedom you need the *Constraints* section. See section 3.6.1 for more information.

Example:

```
<material id="1" type="rigid body">
      <density>1.0</density>
      <center_of_mass>0,0,0</center_of_mass>
</material>
```

[§] This is different from previous versions of FEBio where rigid bodies were initially fully constrained.

3.3.22. Tension-Compression Nonlinear Orthotropic

The material type for the tension-compression nonlinear orthotropic material is "*TC nonlinear orthotropic*". The following material parameters are defined:

<c1>	First Mooney-Rivlin material parameter
<c2>	Second Mooney-Rivlin material parameter
<k>	bulk modulus
<beta>	the β parameter (see below)
<ksi>	the ξ parameter (see below)
<mat_axis>	defines the material axes

This material is based on the following uncoupled hyperelastic strain energy function [17]:

$$W\left(\mathbf{C},\lambda_1,\lambda_2,\lambda_3\right)=W_{iso}\left(\tilde{\mathbf{C}}\right)+\sum_{i=1}^{3}W_i^{TC}\left(\tilde{\lambda}_i\right)+U\left(J\right).$$

The isotropic strain energy W_{iso} and the dilatational energy U are the same as for the Mooney-Rivlin material. The tension-compression term is defined as follows:

$$W_i^{TC}\left(\tilde{\lambda}_i\right)=\begin{cases}\xi_i\left(\tilde{\lambda}_i-1\right)^{\beta_i}, & \tilde{\lambda}_i>1\\ 0, & \tilde{\lambda}_i\le 1\end{cases}\qquad \xi_i\ge 0\quad\text{(no sum on i)}.$$

The $\tilde{\lambda}_i$ parameters are the deviatoric fiber stretches of the local material fibers,

$$\tilde{\lambda}_i=\left(\mathbf{a}_i^0\cdot\tilde{\mathbf{C}}\cdot\mathbf{a}_i^0\right)^{1/2}$$

The local material fibers are defined (in the reference frame) as an orthonormal set of vectors $\mathbf{a}_i^0$. See section 3.3.2 for more information. As with all uncoupled materials, this material uses the three-field element formulation.

A complete example for this material follows.

```
<material id="7" name="cartilage" type="TC nonlinear orthotropic>
      <c1>1.0</c1>
      <c2>0.0</c2>
      <k>100</k>
      <beta>4.3,4.3,4.3</beta>
      <ksi>4525, 4525, 4525</ksi>
      <mat_axis type="local">0,0,0</mat_axis>
</material>
```

3.3.23. Ellipsoidal Fiber Distribution Mooney-Rivlin

The material type for a Mooney-Rivlin material with an ellipsoidal continuous fiber distribution is "*EFD Mooney-Rivlin*". The following material parameters need to be defined:

<c1>	Mooney-Rivlin parameter c1
<c2>	Mooney-Rivlin parameter c2
<k>	bulk modulus
<beta>	parameters $(\beta_1, \beta_2, \beta_3)$
<ksi>	parameters (ξ_1, ξ_2, ξ_3)

The Cauchy stress for this material is given by,

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_{MR} + \boldsymbol{\sigma}_f .$$

Here, $\boldsymbol{\sigma}_{MR}$ is the stress from the Mooney-Rivlin basis, and $\boldsymbol{\sigma}_f$ is the stress contribution from the fibers [18][19, 20]:

$$\boldsymbol{\sigma}_f = \int_0^{2\pi}\int_0^{\pi} H\left(I_n - 1\right)\boldsymbol{\sigma}_n\left(\mathbf{n}\right)\sin\varphi d\varphi d\theta .$$

Here, $\mathbf{n}$ is the unit vector along the fiber direction which in spherical angles is directed along (θ, φ), $I_n = \lambda_n^2 = \mathbf{n}\cdot\mathbf{C}\mathbf{n}$ is related to the fiber stretch λ_n and $H(.)$ is the unit step function that enforces the tension-only contribution.

The fiber stress is determined from a fiber strain energy function in the usual manner,

$$\boldsymbol{\sigma}_n = \frac{2}{J}\mathbf{F}\cdot\frac{\partial\Psi_f}{\partial\mathbf{C}}\mathbf{F}^T = \frac{2I_n}{J}\frac{\partial\Psi}{\partial I_n}\mathbf{n}\otimes\mathbf{n} ,$$

where in this case,

$$\Psi\left(\mathbf{n}, I_n\right) = \xi\left(\mathbf{n}\right)\left(I_n - 1\right)^{\beta(\mathbf{n})} .$$

Finally, the materials parameters β and ξ are determined from:

$$\xi\left(\mathbf{n}\right) = \left(\frac{\cos^2\theta\sin^2\varphi}{\xi_1^2} + \frac{\sin^2\theta\sin^2\varphi}{\xi_2^2} + \frac{\cos^2\varphi}{\xi_3^2}\right)^{-1/2}$$

$$\beta\left(\mathbf{n}\right) = \left(\frac{\cos^2\theta\sin^2\varphi}{\beta_1^2} + \frac{\sin^2\theta\sin^2\varphi}{\beta_2^2} + \frac{\cos^2\varphi}{\beta_3^2}\right)^{-1/2} .$$

The orientation of the material axis can be defined as explained in detail in section 3.3.2.

Example:

```
<material id="1" type="EFD Mooney-Rivlin">
      <c1>1</c1>
      <c2>0</c2>
      <beta>4.5,4.5,4.5</beta>
      <ksi>1,1,1</ksi>
      <k>20000</k>
      <mat_axis type="local"></mat_axis>
</material>
```

3.3.24. Ellipsoidal Fiber Distribution Neo-Hookean

The material type for a Neo-Hookean material with an ellipsoidal continuous fiber distribution is "*EFD neo-Hookean*". The following material parameters need to be defined:

`<E>`	Young's modulus
`<v>`	Poisson's ratio
`<beta>`	parameters $(\beta_1, \beta_2, \beta_3)$
`<ksi>`	parameters (ξ_1, ξ_2, ξ_3)

The Cauchy stress for this material is given by,

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_{NH} + \boldsymbol{\sigma}_f .$$

Here, $\boldsymbol{\sigma}_{NH}$ is the stress from the Neo-Hookean basis, and $\boldsymbol{\sigma}_f$ is the stress contribution from the fibers, as described in Section 3.3.23

Example:

```
<material id="1" type="EFD neo-Hookean">
      <E>1</E>
      <v>0.3</v>
      <beta>4.5,4.5,4.5</beta>
      <ksi>1,1,1</ksi>
      <mat_axis type="local"></mat_axis>
</material>
```

3.3.25. Ellipsoidal Fiber Distribution Veronda-Westmann

The material type for a Veronda-Westmann material with an ellipsoidal continuous fiber distribution is "*EFD Veronda-Westmann*". The following material parameters need to be defined:

`<c1>`	First VW coefficient
`<c2>`	Second VW coefficient
`<k>`	Bulk modulus
`<beta>`	parameters $(\beta_1, \beta_2, \beta_3)$
`<ksi>`	parameters (ξ_1, ξ_2, ξ_3)

The Cauchy stress for this material is given by,

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_{VW} + \boldsymbol{\sigma}_f \ .$$

Here, $\boldsymbol{\sigma}_{VW}$ is the stress from the Neo-Hookean basis, and $\boldsymbol{\sigma}_f$ is the stress contribution from the fibers, as described in Section 3.3.23

Example:

```
<material id="1" type="EFD Veronda-Westmann">
      <c1>1</c1>
      <c2>0.5</c2>
      <k>1000</k>
      <beta>4.5,4.5,4.5</beta>
      <ksi>1,1,1</ksi>
      <mat_axis type="local"></mat_axis>
</material>
```

3.3.26. Ellipsoidal Fiber Distribution with Donnan Equilibrium Swelling

The material type for a swelling pressure combined with an ellipsoidal continuous fiber distribution is "*EFD Donnan equilibrium*". The swelling pressure is described by the equations for ideal Donnan equilibrium, assuming that the material is porous, with a charged solid matrix, and the external bathing environment consists of a salt solution of monovalent counter-ions. The following material parameters need to be defined:

`<phiw0>`	gel porosity (fluid volume fraction) in reference (strain-free) configuration, φ_0^w
`<cF0>`	fixed-charge density in reference (strain-free) configuration, c_0^F
`<bosm>`	external bath osmolarity, $\overline{c}^*$
`<R>`	universal gas constant, R
`<T>`	absolute temperature, T
`<beta>`	parameters $(\beta_1, \beta_2, \beta_3)$
`<ksi>`	parameters (ξ_1, ξ_2, ξ_3)

The Cauchy stress for this material is given by,

$$\mathbf{T} = \mathbf{T}_{DE} + \mathbf{T}_f \,.$$

$\mathbf{T}_f$ is the stress contribution from the fibers, as described in Section 3.3.23. $\mathbf{T}_{DE}$ is the stress from the Donnan equilibrium response [19]:

$$\mathbf{T}_{DE} = -\pi \mathbf{I}$$

where π is the osmotic pressure, given by

$$\pi = RT\left(\sqrt{\left(c^F\right)^2 + \left(\overline{c}^*\right)^2} - \overline{c}^*\right)$$

and c^F is the fixed-charge density in the current configuration, related to the reference configuration via

$$c^F = \frac{\varphi_0^w}{J - 1 + \varphi_0^w} c_0^F$$

where $J = \det \mathbf{F}$ is the relative volume. π is <u>not</u> currently saved as a separate field in the output file. Only the total stress $\mathbf{T}$ is saved.

Note that c_0^F may be negative or positive; the gel porosity is unitless and must be in the range $0 < \varphi_0^w < 1$. A self-consistent set of units must be used for this model. For example:

	(m, N, s, mol, K)	(mm, N, s, nmol, K)
R	8.314 J/mol·K	8.314×10^{-6} mJ/nmol·K
T	K	K
c_0^F	Eq/m^3 = mEq/L	nEq/mm^3 = mEq/L
$\overline{c}^*$	mol/m^3 = mM	nmol/mm^3 = mM
ξ_i	Pa	MPa

π	Pa	MPa

Though this material is porous, this is not a full-fledged poroelastic material as described in Section 3.3.15 for example. The behavior described by this material is strictly valid only after the transient response of interstitial fluid and ion fluxes has subsided (thus Donnan *equilibrium*). Nevertheless, this material may be used to represent the solid matrix of a poroelastic material, under the assumption that ion fluxes in the interstitial fluid equilibrate faster than the solvent flux. In that case, the fluid pressure saved to the output file only describes the part of the pressure over and above the osmotic component π.

Donnan osmotic swelling reduces to zero when either $c_0^F = 0$ or $\overline{c}^* \to \infty$. Therefore, entering any other values for c_0^F and $\overline{c}^*$ at the initial time point of an analysis produces an instantaneous, non-zero swelling pressure. Depending on the magnitude of this pressure relative to the fiber stiffness, the nonlinear analysis may not converge due to this sudden swelling. Therefore, it is recommended to prescribe a load curve for either `<cF0>` or `<bosm>`, to ease into the initial swelling prior to the application of other loading conditions.

Example (using units of mm, N, s, nmol, K):

```
<material id="1" type="EFD Donnan equilibrium">
      <phiw0>0.8</ phiw0>
      <cF0 lc=”1”>1</cF0>
      <bosm>300</bosm>
      <R>8.314e-6</R>
      <T>310</T>
      <beta>3,3,3</beta>
      <ksi>0.01,0.01,0.01</ksi>
      <mat_axis type="local"></mat_axis>
</material>
<LoadData>
      <loadcurve id="1">
            <loadpoint>0,0</loadpoint>
            <loadpoint>1,150</loadpoint>
      </loadcurve>
</LoadData>
```

3.3.27. Muscle Material

This material model implements the constitutive model developed by Silvia S. Blemker [21]. The material type for the muscle material is *muscle material*. The model is designed to simulate the passive and active material behavior of skeletal muscle. It defines the following parameters:

<g1>	along fiber shear modulus
<g2>	cross fiber shear modulus
<p1>	exponential stress coefficients
<p2>	fiber uncrimping factor
<Lofl>	optimal fiber length
<smax>	maximum isometric stress
<lambda>	fiber stretch for straightened fibers
<k>	bulk modulus
<active_contraction>	activation level

The main difference between this material formulation compared to other transversely hyperelastic materials is that it is formulated using a set of new invariants, originally due to Criscione [22], instead of the usual five invariants proposed by A.J.M. Spencer [23]. For this particular material, only two of the five Criscione invariants are used. The strain energy function is defined as follows:

$$W\left(B_1,B_2,\lambda\right)=G_1\tilde{B}_1^2+G_2\tilde{B}_2^2+F_m\left(\tilde{\lambda}\right)+U\left(J\right).$$

The function F_m is the strain energy contribution of the muscle fibers. It is defined as follows:

$$\lambda\frac{\partial F_m}{\partial\lambda}=\sigma_{\max}\left(f_m^{passive}\left(\lambda\right)+\alpha f_m^{active}\left(\lambda\right)\right)\frac{\lambda}{\lambda_{ofl}},$$

where,

$$f_m^{passive}\left(\lambda\right)=\begin{cases}0 & ,\lambda\le\lambda_{ofl}\\ P_1\left(e^{P_2\left(\lambda/\lambda_{ofl}-1\right)}-1\right) & ,\lambda_{ofl}<\lambda<\lambda^*\\ P_3\lambda/\lambda_{ofl}+P_4 & ,\lambda\ge\lambda^*\end{cases},$$

and,

$$f_m^{active}\left(\lambda\right)=\begin{cases}9\left(\lambda/\lambda_{ofl}-0.4\right)^2 & ,\lambda\le 0.6\lambda_{ofl}\\ 9\left(\lambda/\lambda_{ofl}-1.6\right)^2 & ,\lambda\ge 1.4\lambda_{ofl}\\ 1-4\left(1-\lambda/\lambda_{ofl}\right)^2 & ,0.6\lambda_{ofl}<\lambda<1.4\lambda_{ofl}\end{cases},$$

The values P_3 and P_4 are determined by requiring C^0 and C^1 continutity at $\lambda=\lambda^*$.

The parameter α is the activation level and can be specified using the *active_contraction* element. You can specify a loadcurve using the *lc* attribute. The value is interpreted as a scale factor when a loadcurve is defined or as the constant activation level when no loadcurve is defined.

The muscle fiber direction is specified similarly to the transversely isotropic Mooney-Rivlin model.

Example:

```
<material id=”1” type=”muscle material”>
      <g1>500</g1>
      <g2>500</g2>
      <p1>0.05</p1>
      <p2>6.6</p2>
      <smax>3e5</smax>
      <Lofl>1.07</Lofl>
      <lambda>1.4</lambda>
      <k>1e6</k>
      <fiber type=”vector”>1,0,0</fiber>
</material>
```

3.3.28. Tendon Material

The material type for the tendon material is *tendon material*. The tendon material is similar to the muscle material. The only difference is the fiber function. For tendon material this is defined as:

$$\lambda \frac{\partial F_t}{\partial \lambda} = \sigma(\lambda),$$

where

$$\sigma(\lambda) = \begin{cases} 0, & \lambda \le 1 \\ L_1\left(e^{L_2(\lambda-1)} - 1\right) & 1 < \lambda < \lambda^* \\ L_3\lambda + L_4 & \lambda \ge \lambda^* \end{cases}.$$

The parameters L_3 and L_4 are determined by requiring C^0 and C^1 continuity at λ^*.

The material parameters for this material are listed below.

<g1>	along fiber shear modulus
<g2>	cross fiber shear modulus
<l1>	exponential stress coefficients
<l2>	fiber uncrimping factor
<lambda>	fiber stretch for straightened fibers
<k>	bulk modulus

The tendon fiber direction is specified similarly to the transversely isotropic Mooney-Rivlin model.

Example:

```
<material id="1" type="tendon material">
      <g1>5e4</g1>
      <g2>5e4</g2>
      <l1>2.7e6/l1>
      <l2>46.4</l2>
      <lambda>1.03</lambda>
      <k>1e7</k>
      <fiber type="vector">1,0,0</fiber>
</material>
```

3.4. Geometry Section

The geometry section contains all the geometry data, including nodal coordinates and element connectivity. It has the following sub-sections.

- *Nodes*: contains nodal coordinates.
- *Elements*: contains element connectivity.
- *ElementData*: contains additional element data.

3.4.1. Nodes Section

The nodes section contains all the nodal coordinates. Repeat the following XML-element for each node.

```
<node id="n">x,y,z</node>
```

The *id* attribute is a unique number that identifies the node. This *id* is used as a reference in the element connectivity section. The *id*s do not have to be in order, but the lowest id *must* be 1 and numbers can not be skipped. (So if there is a node 4 and a node 6, there must also be a node 5 somewhere).

3.4.2. Elements Section

The element section contains all the element connectivity data. FEBio classifies elements in two classes, namely *solids* and *shells*.

Solid Elements

The following solid element types are defined:

hex8: 8-node hexahedral element
penta6: 6-node pentahedral element
tet4: 4-node tetrahedral element

The value for the XML element is the nodal connectivity:

```
<hex8 id="n" mat="m">n1,n2,n3,n4,n5,n6,n7,n8</hex8>
<penta6 id="n" mat="m">n1,n2,n3,n4,n5,n6</penta6>
<tet4 id="n" mat="m">n1,n2,n3,n4</tet4>
```

Elements have two attributes. First, there is a unique *id* that can be used to reference the element. The same limitations to this id apply as to the nodal ids. The second attribute is the material number. This number is the material *id* that was defined in the material section.

The node numbering has to be defined as in the figure below.

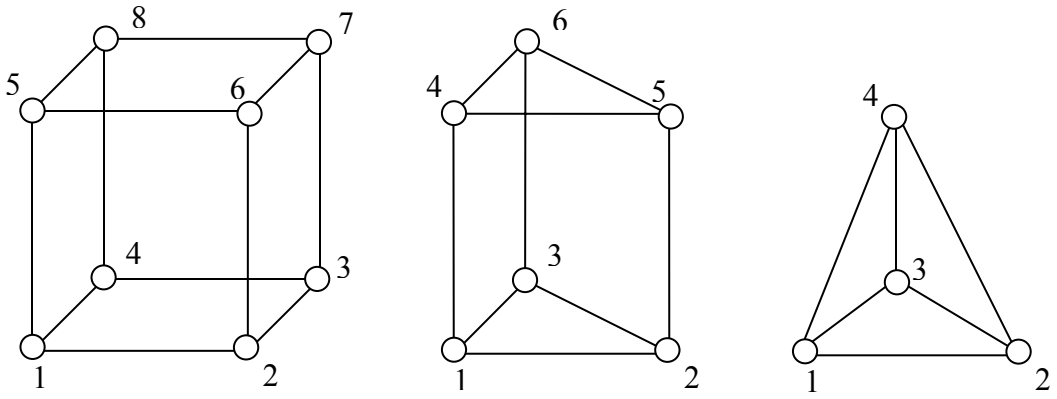

Figure 3-4. Node numbering for solid elements.

Shell Elements

FEBio currently supports a 4-noded quadrilateral and a 3-noded triangular shell element. Use the following XML tags to define the shell element:

```
<quad4 id="n" mat="m">n1,n2,n3,n4</quad>
<tri3 id="n" mat="m">n1,n2,n3</tri3>
```

Like the solid element, the shell element needs two attributes. The first one identifies the unique element ID, while the second is the material number. The shell thickness is specified in the ElementData section.

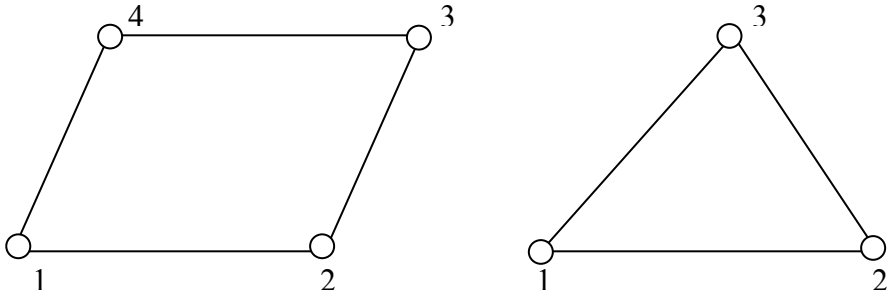

Figure 3-5. Node numbering for shell elements.

Surface Elements

In many cases the surface of some geometry (or part of it) is required. For example, pressure forces are applied to the surface. For this reason surface elements have to be defined. Two surface elements are available.

- *quad4*: 4-node quadrilateral element
- *tri3*: 3-node triangular element

The value for the surface element is the nodal connectivity:

```
<quad4 id="n">n1,n2,n3,n4</quad4>
<tri3 id="n">n1,n2,n3</tri3>
```

Surface elements can not overlap element boundaries. That is, the surface element must belong to a specific element. Surface elements do not contribute to the total number of elements in the mesh. They are also not to be confused with shell elements.

3.4.3. ElementData Section

Additional element data can be specified in the *ElementData* section. The data in this section usually represents material data that can vary from element to element. The following element properties can be defined.

Property	Description
`fiber`	Specify a local fiber direction
`thickness`	Specify the shell element thickness

Note that defining a *fiber* direction here will override the fiber distribution specified in the material definition and only for those elements specified in the *ElementData* section. In other words, you can define a particular fiber distribution in the material section and then override the fiber direction for a subset of elements. See section 3.3 for more information on defining fiber directions for transversely isotropic materials.

The thickness parameter can only be specified for shell elements. The value of this parameter is the shell thickness at each of the shell nodes. For example, for a four-node shell,

```
<element id="n">
      <thickness>0.01,0.01,0.01,0.01</thickness>
</element>
```

To specify a fiber direction for a particular element, enter the following xml-element in the *ElementData* section:

```
<element id="n">
      <fiber>1,0,0</fiber>
</element>
```

The *id* attribute identifies the element to which this fiber direction is applied. The *fiber* element defines a vector in global coordinates. This vector does not have to be of unit length since it is automatically normalized by FEBio.

3.5. Boundary Section

The *Boundary* section defines all boundary conditions that may be applied to the geometry. Several boundary conditions are available: nodal displacements, nodal forces, pressure forces, contact interfaces and rigid joints.

3.5.1. Prescribed Nodal Degrees of Freedom

Nodal degrees of freedom (dof) can be prescribed using the *prescribe* sub-element.

```
<prescribe>
      <node id="n" bc="x" [lc="1"]>2.3</node>
      ...
</prescribe>
```

The *id* attribute indicates to which node this prescribed dof is applied.

The *bc* attribute specifies the particular degree of freedom. The following values are allowed:

x: apply displacement in *x*-direction
y: apply displacement in *y*-direction
z: apply displacement in *z*-direction
p: apply prescribed fluid pressure (only used for poroelastic analysis)

An optional loadcurve can be specified with the *lc* attribute. The value of the *lc* attribute is the ID of the loadcurve that is defined in the *LoadData* section of the input file. If a loadcurve is not specified, the value will be automatically ramped from a value of 0 at time t=0 to the value specified in the xml file at the time corresponding to the end of the analysis (e.g. 2.3 in the example above).

The value of the *node* element (e.g. 2.3 in the example above) is the value for the prescribed displacement. Note that if a loadcurve is specified, this value scales the value determined by the loadcurve.

Degrees of freedom that are fixed (in other words, constrained, or are always zero) can be defined using the *fix* element:

```
<fix>
      <node id="n" bc="x"/>
      ...
</fix>
```

Although the *prescribe* element with a value of zero for the *node* tags can also be used to fix a certain nodal degree of freedom, the user should use the *fix* element whenever possible, since this option causes the equation corresponding to the constrained degree of freedom to be removed from the linear system of equations. This reduces the run time of the FE analysis.

3.5.2. Nodal forces

Nodal forces are applied by the *force* element. These forces always point in the same direction and do not deform with the geometry.

```
<force>
      <node id="n" bc="x" [lc="l"]>3.14</node>
      ...
</force>
```

The *id* attribute indicates to which node this prescribed dof is applied.

The *bc* attribute gives the degree of freedom. The following values are allowed:

x: apply force in *x*-direction
y: apply force in *y*-direction
z: apply force in *z*-direction
p: apply normal fluid flux (only used for poroelastic analysis)

An optional loadcurve can be specified with the *lc* attribute. If a loadcurve is not specified, the value will be automatically ramped from a value of 0 at time t=0 to the value specified in the xml file at the time corresponding to the end of the analysis.

The value of the *node* element (e.g. 3.14 in the example above) is the value for the nodal force. Note that if a loadcurve is specified, this value scales the value determined by the loadcurve.

3.5.3. Pressure forces

Pressure forces are applied to the surface of the geometry and are defined by the *pressure* element:

```
<pressure>
      <quad4 id="n" [lc="l" scale="1.0"]>n1,n2,n3,n4</quad4>
      ...
</pressure>
```

The sub-elements define the geometry of the pressure boundary surface. Each element must be either *quad4* for a four-node quadrilateral or *tri3* for a 3-noded triangular element. The id *n* references the surface element. These pressure forces are also known as *follower forces*; they change direction as the body is deformed and, in this case, are always oriented along the local surface normal. The sign convention is so that a positive pressure will act opposite to the normal, so it will compress the material. The optional parameter *lc* defines a loadcurve for the pressure evolution and *scale* defines a scale factor. The default scale factor is 1.0 and if *lc* is not defined the scale factor is automatically ramped from 0 at time t=0 to the value specified here.

3.5.4. Biphasic Normal Traction

In a biphasic mixture of intrinsically incompressible solid and fluid constituents, the **u**-p formulation adopted in FEBio implies that the total traction is a natural boundary condition. If

this boundary condition is not explicitly prescribed, the code automatically assumes that it is equal to zero. Therefore, biphasic boundaries are traction-free by default.

The *mixture traction* $\mathbf{t}$ is the traction vector corresponding to the mixture (or total) stress $\mathbf{T}$; thus $\mathbf{t}=\mathbf{T}\cdot\mathbf{n}$, where $\mathbf{n}$ is the outward unit normal to the boundary surface. Since $\mathbf{T}=-p\mathbf{I}+\mathbf{T}^e$, where p is the fluid pressure and $\mathbf{T}^e$ is the *effective stress* resulting from strains in the solid matrix, it is also possible to represent the total traction as $\mathbf{t}=-p\mathbf{n}+\mathbf{t}^e$, where $\mathbf{t}^e=\mathbf{T}^e\cdot\mathbf{n}$ is the *effective traction*. Currently, FEBio allows the user to specify only the normal component of the traction, either $t_n=\mathbf{t}\cdot\mathbf{n}$ (the normal component of the mixture traction) or $t_n^e=\mathbf{t}^e\cdot\mathbf{n}$ (the normal component of the effective traction):

```
<normal_traction traction="mixture">
      <quad4 id="n" [lc="l" scale="1.0"]>n1,n2,n3,n4</quad4>
      ...
</normal_traction>
```

or

```
<normal_traction traction="effective">
      <quad4 id="n" [lc="l" scale="1.0"]>n1,n2,n3,n4</quad4>
      ...
</normal_traction>
```

The optional parameter *lc* defines a loadcurve for the normal traction evolution and *scale* defines a scale factor. The default scale factor is 1.0 and if *lc* is not defined the scale factor is automatically ramped from 0 at time *t*=0 to the value specified here.

Recall that, unlike the mixture and effective traction, the fluid pressure p is a nodal variable (see Prescribed Nodal Degrees of Freedom). There may be common situations where the user must apply a combination of related fluid pressure and traction boundary conditions. For example, if a biphasic surface is subjected to a non-zero fluid pressure p_0, the corresponding boundary conditions are $p=p_0$ and $t_n=-p_0$ (or $t_n^e=0$). In FEBio, both boundary conditions must be applied. For example:

```
<Boundary>
      <fix>
            <node id="n1" bc="p" lc="1">1.0</node>
            <node id="n2" bc="p" lc="1">1.0</node>
            <node id="n3" bc="p" lc="1">1.0</node>
            <node id="n4" bc="p" lc="1">1.0</node>
            ...
      </fix>
      <normal_traction traction="mixture">
            <quad4 id="n" lc="l" scale="-1.0" >n1,n2,n3,n4</quad4>
            ...
      </normal_traction>
</Boundary>
<LoadData>
```

```
      <loadcurve id="1">
            <loadpoint>0,0</loadpoint>
            <loadpoint>1,2.0</loadpoint>
      </loadcurve>
</LoadData>
```

3.5.5. Biphasic Fluid Flux

In a biphasic mixture of intrinsically incompressible solid and fluid constituents, the **u**-p formulation adopted in FEBio implies that the normal component of the relative fluid flux is a natural boundary condition. If this boundary condition is not explicitly prescribed, the code automatically assumes that it is equal to zero. Therefore, biphasic boundaries are impermeable by default. (To implement a free-draining boundary, the fluid pressure nodal degrees of freedom should be set to zero.)

The flux of fluid relative to the solid matrix is given by the vector $\mathbf{w}$. Since viscosity is not explicitly modeled in a biphasic material, the tangential component of $\mathbf{w}$ on a boundary surface may not be prescribed. Only the normal component of the relative fluid flux, $w_n = \mathbf{w} \cdot \mathbf{n}$, represents a natural boundary condition. To prescribe a value for w_n on a surface, use

```
<fluidflux type="nonlinear/nonlinear" flux="fluid">
      <quad4 id="n" [lc="l" scale="1.0"]>n1,n2,n3,n4</quad4>
      ...
</fluidflux>
```

The optional parameter *lc* defines a loadcurve for the normal traction evolution and *scale* defines a scale factor. The default scale factor is 1.0 and if *lc* is not defined the scale factor is automatically ramped from 0 at time t=0 to the value specified here.

type="nonlinear" (default) means that the flux matches the prescribed value even if the surface on which it is applied changes in area as it deforms. Therefore the net volumetric flow rate across the surface changes with changes in area. This type of boundary condition is useful if the fluid flux is known in the current configuration.

type="linear" means that the prescribed flux produces a volumetric flow rate based on the undeformed surface area in the reference configuration. Therefore the flux in the current configuration does not match the prescribed value. This type of boundary condition is useful if the net volumetric flow rate across the surface is known. For example: Let Q be the known volumetric flow rate, let A_0 be the surface area in the reference configuration (a constant). Using *"linear"* means that the user prescribes Q/A_0 as the flux boundary condition. (However, regardless of the *type*, the fluid flux saved in the output file has a normal component equal to Q/A, where A=area in current configuration.)

Prescribing w_n on a free surface works only if the nodal displacements of the corresponding faces are also prescribed. If the nodal displacements are not known a priori, the proper boundary

condition calls for prescribing the normal component of the mixture velocity, $v_n = \left(\mathbf{v}^s + \mathbf{w}\right)\cdot\mathbf{n}$. To prescribe the value of v_n on a surface, use

```
<fluidflux type="nonlinear/nonlinear" flux="mixture">
      <quad4 id="n" [lc="l" scale="1.0"]>n1,n2,n3,n4</quad4>
      ...
</fluidflux>
```

For example, this boundary condition may be used when modeling a permeation problem through a biphasic material, when the upstream fluid velocity is prescribed, $v_n = v_0$. If the upstream face is free, the companion boundary condition would be to let $t_n^e = 0$ on that face as well.

3.5.6. Contact Interfaces

Contact boundary conditions are defined with the *contact* element. The *type* attribute specifies the type of contact interface that is defined. For example:

```
<contact type="sliding_with_gaps"> ... </contact>
```

The *type* can be one of the following options:

Type	Description
`sliding_with_gaps`, `facet-to-facet sliding`, `sliding2`	A sliding interface that may separate (with biphasic contact for *sliding2*)
`rigid_wall`	A sliding interface with rigid wall as master surface
`rigid`	A rigid interface
`rigid_joint`	A joint between two rigid bodies
`tied`	A tied interface

Sliding Interfaces

A sliding interface can be used to setup a non-penetration constraint between two surfaces. As of version 1.2, three different sliding contact algorithms are available. Although all three are based on the same contact enforcement method, they all differ slightly in their implementation and have been shown to give different performance for different contact scenarios. Each sliding contact implementation is identified by a different *type* attribute.

- ***sliding_with_gaps (SWG)***: This is FEBio's original implementation of sliding contact. It is based on Laursen's contact formulation [24] which poses the contact problem as a nonlinear constrained optimization problem. In FEBio, the lagrange multipliers that

enforce the contact constraints are computed either using a penalty method or the augmented Lagrangian method.

- ***facet-to-facet sliding (F2F)***: This implementation is identical to the *sliding_with_gaps* implementation but uses a more accurate integration rule: where the former method uses nodal integration, this method uses Gaussian quadrature to integrate the contact equations. This method has been demonstrated to give additional stability and often converges when the former method does not.
- ***sliding2 (S2)***: This method is similar to the *facet-to-facet sliding* but differs in the linearization of the contact forces, which results in a different contact stiffness matrix compared to the previous two methods. This method is described in detail in [25]. This method sometimes performs better than the previous two methods for problems that are dominated by compression. However, the formulation is inherently non-symmetric and therefore will require additional memory and running time. A symmetrized version of this implementation is available (see below), but the symmetric version does not converge as well as the non-symmetric version. This particular contact implementation also supports biphasic contact (see the next section).

The following table lists the properties that are defined for sliding interfaces. It is important to note that the three different sliding implementations cannot be used interchangeably: not all features are available for each method. The third, fourth and fifth column indicate if a parameter is available for a particular implementation.

Parameter	Description	SWG	F2F	S2	Default
`penalty`	normal penalty scale factor (1)	●	●	●	1.0
`auto_penalty`	auto-penalty calculation flag (2)	●	●	●	0
`two_pass`	two-pass flag (3)	●	●	●	0
`laugon`	augmented Lagrangian flag (4)	●	●	●	0
`tolerance`	aug. Lagrangian convergence tolerance (4)	●	●	●	1.0
`gaptol`	tolerance for gap value (4)	●	●	●	0.0 (off)
`minaug`	minimum number of augmentations (4)	●	●		0
`maxaug`	maximum number of augmentations (4)	●	●		10
`fric_coeff`	frictional coefficient (5)	●			0.0
`fric_penalty`	tangential penalty factor (5)	●			0.0
`ktmult`	tangential stiffness multiplier (5)	●			1.0
`seg_up`	maximum number of segment updates (6)	●			0 (off)
`symmetric_stiffness`	symmetric stiffness matrix flag (7)			●	0

The slave and master surfaces, which define the contact interface, are entered by specifying the *surface* element. The *type* attribute is used to specify whether it is a *slave* or *master* surface. The *surface* tag is followed by the definition of the surface elements. For example, a list of facets composing the master surface of the sliding interface could be written as:

```
<surface type="master">
      <quad4 id="n">n1,n2,n3,n4</quad4>
      ...
</surface>
```

Both quadrilateral surface elements (quad4) and triangular elements (tri3) can be used to define the master and/or slave surfaces.

Comments:

1. If the augmented Lagrangian flag is turned off (see comment 4), the penalty method is used to enforce the contact constraint. In this method the contact traction is determined by the gap (i.e. penetration distance) multiplied by the user-defined *penalty* factor. In the augmented Lagrangian method the *penalty* parameter is also used but has a slightly different meaning. In this case, it scales the lagrange multiplier increment. Due to the different meanings, the user might have to adjust the penalty factor when switching between penalty method and augmented Lagrangian method. In general the penalty method requires a larger penalty factor to reach the same gap than the augmented Lagrangian method. See comment 4 for more information on when to use which method.

2. Choosing a good initial penalty parameter can often be a difficult task since this parameter depends on material properties as well as on mesh dimensions. For this reason an algorithm has been implemented in FEBio that attempts to calculate a good initial value for the penalty factor ε_i for a particular node/integration piont i on the contact interface:

$$\varepsilon_i = \frac{EA}{V}.$$

   Here, A is the area of the element the integration point belongs to, V is the element volume and E is a measure of the elasticity modulus, which is calculated from the elasticity tensor of the element. Although the meaning of E depends on the precise material formulation, in general one can regard it as a measure of the small strain Young's modulus of the material.

To use this feature, add the following element to the contact section:

```
<auto_penalty>1</auto_penalty>
```

When the auto-penalty flag is on, the value of the *penalty* parameter serves as a scale factor for the automatically-calculated penalty factor.

3. Each sliding interface consists of a master surface and a slave surface. The slave surface is the surface over which the contact equations are integrated and on which the contact tractions are calculated. The master surface is used to measure the gap function and to define the necessary kinematic quantities such as surface normals and tangents. This approach is usually referred to as the *single-pass* method. When using the single-pass algorithm, the results can be influenced by the choice of slave and master surfaces. It is best to use the most tessellated surface as the slave and the coarsest as the master surface. To resolve the bias issue, one can also use a *two-pass* algorithm for enforcement of the contact constraint. In this case, a single pass is performed first, after which the slave and master surfaces are swapped and another pass is performed. When using the two-pass method, the definition of master and slave surfaces is arbitrary. In most problems, the single pass is sufficient to enforce contact; with a judicious choice of slave-master pair

and contact parameters, good results can be obtained. If however, the single pass does not give good answers, for example, when due to the geometry's curvature the gap cannot be small enough with a single pass, the two-pass method can be used, although at the expense of more calculations.

If one of the contacting surfaces is rigid, a slightly different approach is recommended. In this case, it is best to pick the rigid surface as the master surface and to use a single pass algorithm. The reason is that the nodal degrees of freedom on the rigid surface are condensed to the rigid degrees of freedom and if the rigid surface is the slave surface, the reaction forces may not propagate correctly to the master surface. This is especially true if the rigid degrees of freedom are fixed.[**]

4. In the presence of a sliding interface (and other contact interfaces), FEBio needs to calculate the contact tractions that prevent the two participating surfaces from penetrating. In general these tractions can be found using the method of lagrange multipliers. However, the direct calculation of these multipliers has several computational disadvantages and therefore FEBio approximates the multipliers using one of two alternative methods: the penalty method and the augmented Lagrangian method. In the former method, the multipliers are approximated by the gap (i.e. penetration distance) scaled by a suitably chosen penalty factor. In many cases, this method is sufficient to get good results. Since the correctness of a contact solution is directly determined by the amount of penetration at the converged state, the user has direct control over the quality of the solution. By increasing the penalty factor, the penetration is reduced. However, in some cases, especially in large compression problems, the penalty factor required to achieve an acceptable amount of penetration has to be so large that it causes numerical instabilities in the non-linear solution algorithm due to ill-conditioning of the stiffness matrix. In these cases, the augmented Lagrangian method might be a better choice. In this method, the multipliers are determined iteratively where in each iteration the multiplier's increments are determined with a penalty-like method. The advantage of this method is twofold: due to the iterative nature, the method will work with a smaller penalty factor, and in the limit, the exact lagrange multipliers can be recovered.

   To turn on the augmented Lagrangian method, simply add the following line to the contact section:

   ```
   <laugon>1</laugon>
   ```

   To turn off the augmented Lagrangian method, either set the value to 0 or remove the parameter altogether. The convergence tolerance is set as follows:

   ```
   <tolerance>0.01</tolerance>
   ```

   With this parameter set, the augmented Lagrangian method will iterate until the relative increment in the multipliers is less than the tolerance. For instance, setting the tolerance parameter to 0.01 implies that the augmented Lagrangian method will converge when

[**] In future versions of FEBio rigid surfaces will be automatically picked to be the master.

there is less than a 1% change in the L2 norm of the augmented Lagrangian multiplier vector between successive augmentations. Alternatively, the user can also specify a tolerance for the gap value. In this case, the iterations will terminate when the gap norm, which is defined as the averaged L2 norm, ($\sqrt{\sum_i < g_i >^2} / N$, <•> the Macauley Bracket) is less than the user-specified value:

```
<gaptol>0.001</gaptol>
```

However, one must be careful when specifying a gap tolerance. First note that the gap tolerance is an absolute value (unlike the *tolerance* which is a relative value), so this parameter depends on the dimensions of the model. Also, there are cases when a gap tolerance simply cannot be reached due to the geometry of the model in which case the augmentations may never converge.

Note that both convergence parameters may be defined at once. In that case, FEBio will try to satisfy both convergence criteria. On the other hand, setting a value of zero will turn off the convergence criteria. For example, the default value for *gaptol* is zero, so that FEBio will not check the gap norm by default.

Finally, the *minaug* and *maxaug* can be used to set a minimum and maximum number of augmentations. When the *maxuag* value is reached, FEBio will move on to the next timestep, regardless of whether the force and gap tolerances have been met. When specifying a value for *minaug*, FEBio will perform at least minaug augmentations.

5. The *sliding_with_gaps* contact implementation is currently the only contact algorithm that supports friction. Three parameters control the frictional response: *fric_coeff* is the material's friction coefficient and its value must be in the range from 0.0 to 1.0; *fric_penalty* is the penalty factor that regulates the tangential traction forces, much like the *penalty* parameter regulates the normal traction force component; the parameter *ktmult* is a scale factor for the tangential stiffness matrix. It is default to 1.0, but it is observed that reducing this value might sometimes improve convergence.

6. In a contact problem, FEBio calculates the projection of each slave node on the master surface. As a slave node slides across the master surface, the corresponding master segment can change. However, in some cases, switching segments is undesirable since it might cause instabilities in the solution process or a state in which the node oscillates continuously between two adjacent facets and thus prevents FEBio from meeting the displacement convergence tolerance. The parameter *seg_up* allows the user to control the number of segment updates FEBio will perform during each time step. For example, a value of 4 tells FEBio it can do the segment updates during the first four iterations. After that, slave nodes will not be allowed to switch to new master segments. The default value is 0, which means that FEBio will do a segment update each iteration of each timestep.

7. The *sliding2* contact implementation for sliding contact is an inherently non-symmetric formulation. However, it was found that through a slight modification a symmetric stiffness matrix could be recovered. It was noted, though, that the symmetrized version of

the algorithm did not perform as well as the nonsymmetric version so it is recommended to use the latter for tougher contact problems. The following line controls which version of the algorithm is used.

```
<symmetric_stiffness>1</symmetric_stiffness>
```

A value of 1 uses the symmetric version, where a value of 0 uses the non-symmetric version.

Biphasic Contact

The new *sliding2* implementation for sliding interfaces can also deal with biphasic contact surfaces. It allows for the possibility to track fluid flow across the contact interface. In other words, fluid can flow from one side of the contact interface to the other. Note that for this feature to work properly, the materials on either side of the interface must be defined as poroelastic materials. To use this feature, the user must define an additional contact parameter, namely:

```
<pressure_penalty>1.0</pressure_penalty>
```

In the same way that the *penalty* parameter controls the contact tractions, this parameter controls the penalty value that is used to calculate the lagrange multipliers for the pressure constraint. If the *laugon* flag is set, the augmented Lagrangian method is used to enforce the pressure constraint. And if the *auto_penalty* flag is defined, an initial guess for the pressure penalty is calculated automatically using the following formula.

$$\varepsilon_p = \frac{kA}{V}$$

where A is the element's area, V is the element's volume and k is a measure of the permeability which is defined as one third of the trace of the material's initial permeability tensor.

Rigid Wall Interfaces

A rigid wall interface is similar to a sliding interface, except that the master surface is a rigid wall. The following properties are defined for rigid wall interfaces.

`tolerance`	augmentation tolerance	0.01
`penalty`	penalty factor	1.0
`plane`	the plane equation for the rigid wall	N/A

The *plane* property defines the plane equation for the rigid wall. Its value is an array of four values: a, b, c, d_0. It also takes a loadcurve as an optional attribute to define the motion of the plane as a function of time. The loadcurve defines the offset h from the initial position in the direction of the plane normal:

$$ax + by + cz + d(t) = 0, \quad d(t) = d_0 + h(t)$$

So, for example, a rigid wall that initially lies in the xy-coordinate plane and moves in the z-direction would be specified as follows:

```
<plane lc="1">0,0,1,0</plane>
```

The slave surface is defined by specifying a *surface* element. The *surface* tag is followed by the definition of the surface elements:

```
<surface>
      <quad4 id="n">n1,n2,n3,n4</quad4>
      ...
</surface>
```

Triangular elements (tria3) may also be used instead of quadrilateral elements (quad4).

Tied Interfaces

A tied interface can be used to connect two non-conforming meshes. A tied interface requires the definition of both a slave and a master surface. It is assumed that the nodes of the slave surface are connected to the faces of the master surface. The following control parameters need to be defined:

<penalty>	penalty factor
<tolerance>	augmentation tolerance

The slave and master surfaces are defined similarly as for the sliding interfaces. The *type* attribute is used to specify whether it is a *slave* or *master* surface. The *surface* tag is followed by the definition of the surface elements:

```
<surface type="master">
      <quad4 id="n">n1,n2,n3,n4</quad4>
      ...
</surface>
```

Both quadrilateral surface elements (quad4) and triangular elements (tri3) may be used to define the surface.

Rigid Interfaces

A rigid interface defines a list of nodes that are attached to a rigid body. These nodes will move with the rigid body:

```
<contact type="rigid">
      <node id="n1" rb="1"></node>
      ...
      <node id="n2" rb="1"></node>
</contact>
```

The *id* attribute identifies the node and *rb* is the material id of the rigid body. The value of the node is ignored.

Rigid Joints
A rigid joint connects two rigid bodies at a point in space:

```
<contact type=”rigid joint”>
      <tolerance>0.1</tolerance>
      <penalty>2</penalty>
      <body1>1</body1>
      <body2>2</body2>
      <joint>0,0,0</joint>
</contact>
```

The *tolerance* element defines the augmentation tolerance. That is, when the relative change in the constraint forces (the lagrange multipliers) are less than this value. The *body1* and *body2* elements are the material numbers of the two rigid bodies. The *joint* element defines the position of the joint in world coordinates at the start of the analysis. Note that this point does not have to be inside or on the surface of either of the two bodies.

3.6. Constraints Section

The Constraints section allows the user to enforce additional constraints to the model. Currently, only rigid body constraints can be enforced.

3.6.1. Rigid Body Constraints

Rigid bodies are initially unconstrained which means they can move in all three directions and can rotate about all three axes. To constrain the degrees of freedom of a rigid body you can use the *rigid_body* element.

```
<Constraints>
	<rigid_body>
		<!-- constraints go here -->
	</rigid_body>
</Constraints>
```

The following table lists the elements that can be defined in the *rigid_body* element.

`<trans_x>`	x-translation code. Attribute *type* specifies the type of the translation.
`<trans_y>`	y-translation code. Attribute *type* specifies the type of the translation.
`<trans_z>`	z-translation code. Attribute *type* specifies the type of the translation.
`<rot_x>`	x-rotation code. Attribute *type* specifies the type of the rotation.
`<rot_y>`	y-rotation code. Attribute *type* specifies the type of the rotation.
`<rot_z>`	z-rotation code. Attribute *type* specifies the type of the rotation.

The translation and rotation boundary codes are used to constrain the motion of the rigid body. The full syntax for e.g. the x-translation is as follows:

```
<trans_x type="<type>" [lc="<lc>"]> 1.0 </trans_x>
```

The *type* attribute can take on any of the following values.

Type	**Description**
`fixed`	Degree of freedom is fixed
`prescribed`	Degree of freedom is prescribed by user
`force`	A force is applied in direction of degree of freedom

If the type is *prescribed* or *force* then the *lc* attribute can be used to specify a load curve defining the amplitude of the displacement or force. The value is then interpreted as a scale factor. For all other types the value is ignored. The syntax and interpretation is the same for the other translation and rotation codes.
When the type is *force*, the force is applied at the center of mass for translational degrees of freedom and torque is applied around the center of mass for rotational degrees of freedom. The center of mass of a rigid body is either specified in the material definition or calculated automatically by FEBio.

Example:

```
<rigid_body mat="1">
      <trans_x type="prescribed" lc="1">2.0</trans_x>
      <trans_y type="fixed"/>
      <trans_z type="fixed"/>
      <rot_x type="fixed"/>
      <rot_y type="fixed"/>
      <rot_z type="fixed"/>
</rigid_body>
```

In this example the rigid body that corresponds to material definition *1* has a prescribed displacement defined for the *x* degree of freedom and has all other degrees of freedom fixed.

A force (torque) can be applied at the center of mass by setting the *type* attribute to *force*. Note that specifying a force (torque) will automatically free the corresponding translational (rotational) degree of freedom. For example, applying a force in the *x*-direction while keeping the *y* and *z* directions fixed and the rotational degrees of freedom free, can be done as follows.

```
<rigid_body mat="1" >
      <trans_x type="force" lc="1">1.0</trans_x>
      <trans_y type="fixed"/>
      <trans_z type="fixed"/>
 </rigid_body>
```

3.7. Globals Section

The *Globals* section is used to define some global variables. Currently, the only parameter in this section is the body force. The body force is defined as a 3D vector. Each component can be associated with a load curve to define a time dependent body force. Only the non-zero components need to be defined:

```
<Globals>
	<body_force>
		<x lc="1">1.0</x>
		<y lc="2">1.0</y>
		<z lc="3">1.0</z>
	</body_force>
</Globals>
```

The *lc* attribute defines the load curve to use for the corresponding component. The values of the components can be used to define scale factors for the load values.

3.8. LoadData Section

The *LoadData* section contains the loadcurve data. A loadcurve is defined by the *loadcurve* element. Each loadcurve is defined by repeating the *loadpoint* element for all data points:

```
<loadcurve id="1">
      <loadpoint> 0, 0 </loadpoint>
      ...
      <loadpoint> 1, 1 </loadpoint>
</loadcurve>
```

The *id* attribute is the loadcurve number and is used in other sections of the input file as a means to reference this curve.

FEBio also defines a "zero" loadcurve. This loadcurve is used for all time-dependent parameters that have no loadcurve specified (that is, the *lc* attribute is omitted). This loadcurve will have the effect that parameters are ramped up linearly from zero to their specified value. Imagine for instance, a rigid body defined as follows:

```
<material id="1" type="rigid body">
      <density>1.0</density>
      <trans_x type="prescribed">2.0</trans_x>
</material>
```

The *x* degree of freedom is prescribed, a loadcurve is not specified. FEBio will automatically use the "zero" loadcurve and ramp up the value from 0 at the start time to 2 at the end time.

3.9. Output Section

FEBio usually splits the output in two files: the *logfile,* which contains the same information that was written to the screen during the analysis, and the *plotfile,* which contains (most of) the results. However, in some cases it might be useful to record additional results that are not stored by default. The user can request additional data by defining an *Output* section. FEBio allows the user to store this additional data in the logfile or the plotfile.

3.9.1. Logfile

The user can request FEBio to output additional data in the logfile. This feature is called *data logging*. To use this feature, simply define the following element in the *Output* section of the input file:

```
<Output>
      <logfile [file="<log file>"]>
            <node_data       [attribute list]>item list</node_data>
            <element_data    [attribute list]>item list</element_data>
            <rigid_body_data [attribute list]>item list</rigid_body_data>
      </logfile>
</Output>
```

The optional attribute *file* defines the name of the logfile. If omitted, the default name is used. Additional data is stored by adding one or more of the following elements:

- *node_data*: request nodal data
- *element_data*: request element data
- *rigid_body_data*: request rigid body data

Each of these data classes takes the following attributes:

- *data*: an expression defining the data that is to be stored
- *name:* a descriptive name for the data (optional; default = data expression)
- *file*: the name of the output file where the data is stored. (optional; default = logfile)
- *delim*: the delimiter used to separate data in multi-column format (optional; default = space)

The *data* attribute is the most important one and is mandatory. It contains a mathematical expression using any valid combination of variable names and arithmetic operators. The available variable names depend on the data class and are defined below. A single *data* attribute can define multiple data expressions by separating them with a semicolon (;). For example, the data expression

```
data="x;y;z"
```

will store the variables *x*, *y* and *z* in separate columns. For more examples see below.

The optional *name* attribute is a descriptive name for the data. It is used in the logfile to refer to this data and can be used to quickly find the data record in the logfile. If omitted, the data expression is used as the name.

The *file* attribute defines the name of the output file where the data is to be stored. This attribute is optional and when not specified the data will be stored in the logfile. Note that the filename given is a template. FEBio appends a number at the end of the filename to indicate to which timestep the data belongs. For instance, if you define a file name as follows:

```
file = "data.txt"
```

then the first file that is written will have the name *data000.txt*. After the first converged timestep a file with name *data001.txt* will be written and so on.

The optional *delim* attribute defines the delimiter that is used in multi-column format. As described above, data can be stored in multiple columns and the delimiter is used to separate the columns. The default is a single space.

The value of the data elements is a list of items for which the data is to be stored. For example, for the *node_data* element the value is a list of nodes, for the *element_data* element it is a list of FE elements and for the *rigid_body* element it is a list of rigid bodies. The value may be omitted in which case the data for all items will be stored. For instance, omitting the value for the *node_data* element will store the data for all nodes.

As stated above, the data is either stored in the logfile or in a separate file. In any case, a record is made in the logfile. When storing the data in the logfile, the following entry will be found in the logfile at the end of each converged timestep for each data element:

```
Data Record #<n>
=============================================================
Step = <time step>
Time = <time value>
Data = <data name>
<actual data goes here>
```

The record number *n* corresponds to the *n*th data element in the input file. The *Step* value is the current time step. The *Time* value is the current solution time. The *Data* value is the name of the data element as provided by the *name* attribute (or the *data* attribute if *name* is omitted). The actual data immediately follows this record. If multiple column output is used, the columns are separated by the *delim* attribute of the data element.

When storing the data in a separate file, the format is slightly different:

```
Data Record #<n>
=============================================================
Step = <time step>
Time = <time value>
Data = <data name>
File = <file name>
```

The *File* value is the name of the physical file. Note that this is the name to which the time step number is appended. In addition, the physical file that stores the data contains the following header:

```
*Title = <problem title>
*Step  = <time step>
*Time  = <time value>
*Data  = <data name>
<actual data goes here>
```

The problem title is as defined in the input file.

In either case, the actual data is a multi-column list, separated by the delimiter specified with the *delim* attribute (or a space when omitted). The first column always contains the item number. For example, the following data element:

```
<node_data data="x;y;z" name="nodal coordinates" delim=",">1:4:1</node_data>
```

will result in the following record in the logfile:

```
Data Record #1
Step = 1
Time = 0.1
Data = "nodal coordinates"
1,0.000,0.000,0.000
2,1.000,0.000,0.000
3,1.000,1.000,0.000
4,0.000,1.000,0.000
```

This data record is repeated for each converged time step. The following sections define the data variables that are available for each of the data classes.

Node_Data class

The *node_data* class defines a set of nodal variables. The data is stored for each node that is listed in the item list of the *node_data* element or for all nodes if no list is defined. The following nodal variables are defined.

Node variables	Description
x	x-coordinate of current nodal position
y	y-coordinate of current nodal position
z	z-coordinate of current nodal position
ux	x-coordinate of nodal displacement
uy	y-coordinate of nodal displacement
uz	z-coordinate of nodal displacement

For analyses using poroelastic materials, the following additional variables can be defined.

Node variables	Description
p	fluid pressure
vx	x-component of solid velocity
vy	y-component of solid velocity
vz	z-component of solid velocity

For example, to store the current nodal positions of all nodes, use the following *node_data* element:

```
<node_data data="x;y;z"></node_data>
```

You can store the total nodal displacement for nodes 1 through 100, and all even numbered nodes 200 through 400 as follows:

```
<node_data data="sqrt(ux*ux+uy*uy+uz*uz)">1:100:1,200:400:2</node_data>
```

Element_Data Class

The *element_data* class defines a set of element variables. The data is stored for each element that is listed in the item list of the *element_data* element or for all nodes if no list is defined. The following element variables are defined. Note that the actual value is the average over the element's integration points values (if applicable).

Element variables	Description
sx	xx-component of the Cauchy stress
sy	yy-component of the Cauchy stress
sz	zz-component of the Cauchy stress
sxy	xy-component of the Cauchy stress
syz	yz-component of the Cauchy stress
sxz	xz-component of the Cauchy stress
Ex	xx-component of the Green-Lagrange strain
Ey	yy-component of the Green-Lagrange strain
Ez	zz-component of the Green-Lagrange strain
Exy	xy-component of the Green-Lagrange strain
Eyz	yz-component of the Green-Lagrange strain
Exz	xz-component of the Green-Lagrange strain

For analyses using poroelastic materials, the following additional variables can be defined:

Element variables	Description
wx	x-component of fluid flux
wy	y-component of fluid flux
wz	z-component of fluid flux

For example, to store the (average) Cauchy stress for all elements, define the following data element:

```
<element_data data="sxx;syy;szz;sxy;syz;sxz" name="element stresses" >
</element_data>
```

To store the element pressure, define the following element for elements 1 to 100, and 200 to 300:

```
<element_data data="-0.3*(sxx+syy+szz)">1:100,200:300</element_data>
```

Rigid_Body_Data Class

The *rigid_body_data* class defines a set of variables for each rigid body. The data is stored for each rigid body that is listed in the item list of the *rigid_body_data* element or for all rigid bodies if no list is defined. The following variables are defined. Note that the item referenced in the item list is the material number of the rigid body.

Rigid body variables	Description
x	x-coordinate of center of mass
y	y-coordinate of center of mass
z	z-coordinate of center of mass
qx	x-component of rotation quaternion
qy	y-component of rotation quaternion
qz	z-component of rotation quaternion
qw	w-component of rotation quaternion
Fx	x-component of the rigid body reaction force
Fy	y-component of the rigid body reaction force
Fz	z-component of the rigid body reaction force
Mx	x-component of the rigid body reaction torque
My	y-component of the rigid body reaction torque
Mz	z-component of the rigid body reaction torque

For example, to store the rigid body reaction force of rigid body 2 and 4 add the following data element. Note that the 2 and 4 refer to the rigid body material number as defined in the *Material* section of the input file:

```
<rigid_body_data data="Fx;Fy;Fz">2,4</rigid_body_data>
```

3.9.2. Plotfile

By default, all the results are stored in a binary database, referred to as the *plotfile*. Currently, FEBio uses the LSDYNA database format for the plotfile [1]. This format offers predetermined data storage options. However, in order to provide the user with additional flexibility, FEBio allows the user to customize the data fields in the plotfile. By default, FEBio will choose the data

fields, but you can override them by adding the *plotfile* tag in the Output section of your input file.

Currently, FEBio allows you to redefine the following predefined data fields:

LSDYNA data field	***FEBio data field***
displacement	DISPLACEMENT
velocity	NONE
	VELOCITY
	FLUID_FLUX
	CONTACT_TRACTION
	REACTION_FORCE
	MATERIAL_FIBER
acceleration	NONE
	ACCELERATION
	FLUID_FLUX
	CONTACT_TRACTION
	REACTION_FORCE
	MATERIAL_FIBER
temperature	NONE
	FLUID_PRESSURE
	CONTACT_PRESSURE
	CONTACT_GAP
plastic strain	PLASTIC_STRAIN
	FIBER_STRAIN
	DEV_FIBER_STRAIN

The following example illustrates how to map FEBio data fields to LSDYNA data fields:

```
<Output>
	<plotfile>
		<map field="displacement">DISPLACEMENT</map>
		<map field="velocity">CONTACT_TRACTION</map>
		<map field="acceleration">REACTION_FORCE</map>
		<map field="temperature">CONTACT_GAP</map>
	</plotfile>
</Output>
```

Chapter 4 - Restart Input file

As of version 1.1.6, FEBio will output a restart file. Two files are created when the restart flag is activated: the binary dump file and the text restart input file.

The restart feature in FEBio allows the user to continue a previously terminated analysis. In addition, it allows the user to modify some of the parameters during the restart. To activate the restart feature, define the *restart* element in the Control section:

```
<Control>
      <restart [file="<dump file>"]>1</restart>
      ...
</Control>
```

This describes the format of the restart input file. This file is used to redefine some parameters when restarting a previously terminated run. The structure is very similar to the FEBio input file and also uses XML formatting.

Since the file uses XML, the first line must be the XML header:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

The next line contains the root element of the restart file, and has to be:

```
<febio_restart version="1.0">
```

The restart file is composed of the following sections. These sections are sub-elements of the *febio_restart* root element.

- *Archive*: define the binary dump file used for restarting.
- *Control*: redefine some control parameters
- *LoadData*: redefine some loadcurves.

All sections are optional except for the Archive section, and need only be defined when redefining parameters. In the following paragraphs we describe the different sections in more detail.

4.1. The Archive Section

The Archive section must be the first sub-element of the *febio_restart* root element. This section defines the name of the binary dump file:

```
<Archive>archive.dmp</Archive>
```

4.2. The Control Section

The following control parameters can be redefined:

Parameter	Description
dtol	convergence tolerance for displacements
etol	convergence tolerance for energy
rtol	convergence tolerance for residual
lstol	line search tolerance
max_refs	maximum number of stiffness reformations
max_ups	maximum number of BFGS updates
restart	restart file generation flag
plot_level	defines the frequency of the plot file generation

4.3. The LoadData Section

In the LoadData section the user can redefine some or all of the load curves. The syntax is identical to the LoadData section of the FEBio input file:

```
<LoadData>
	<loadcurve id=”n”>
		<loadpoint>0, 0</loadpoint>
		...
		<loadpoint>1, 0.54</loadpoint>
	</loadcurve>
</LoadData>
```

In this case the loadcurve *id* is the loadcurve number of the loadcurve that the user wishes to redefine.

4.4. Example

The following example defines a restart input file. No parameters are redefined. Only the mandatory *Archive* element is defined. In this case the analysis will simply continue where it left off:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<febio_restart version=”1.0”>
	<Archive>out.dmp</Archive>
</febio_restart>
```

Chapter 5 - Multi-step Analysis

As of version 1.1.6 multi-step problems can be solved with FEBio. A multi-step analysis is defined using multiple steps, where in each step the user can redefine control parameters and boundary conditions. This is useful, for instance, for defining time-dependant boundary conditions or for switching between different analysis types during the simulation.

5.1. The Step section

The multi-step analysis feature introduces a new section to the input file. Each step requires its own *step* section, preferably at the bottom of the input file. In this step section, the user can redefine the control section and the boundary section. The following format is suggested when defining a multi-step analysis:

```
<febio_spec version="1.0">
	<Control>
		<!-- global control parameters -->
	</Control>
	<Material>
		<!-- materials go here -->
	</Material>
	<Geometry>
		<!-- geometry goes here -->
	</Geometry>
	<Boundary>
		<!-- global boundary conditions -->
	</Boundary>
	<LoadData>
		<!-- load curve data goes here -->
	</LoadData>
	<Step>
		<Control>
			<!-- local control settings -->
		</Control>
		<Boundary>
			<!-- local boundary conditions -->
		</Boundary>
	</Step>
</febio_spec>
```

The first part of the file looks similar to a normal input file, except that in the control section only global control parameters should be defined (e.g. the title). Also, the boundary section should only contain global boundary conditions. The differences between local and global parameters are explained below.

After the LoadData section, the user can define as many Step sections as needed. In each Step section, the user can now define the (local) control parameters and boundary conditions.

In a multi-step analysis it is important to understand the difference between local and global settings. The global settings are those settings that remain unchanged during the entire simulation.

5.1.1. Control Settings

All control settings are considered local, except the title. Therefore the title should be the only control setting that is defined in the global control section. All other control parameters, including time step parameters, linear solver parameters, convergence parameters, and so forth, should be defined in the Control section of each step. For example:

```
<febio_spec version=”1.0”>
	<Control>
		<title>This is the title</title>
	</Control>
	...
	<Step>
		<Control>
			<!-- place control parameters here -->
		</Control>
	</Step>
</febio_spec>
```

5.1.2. Boundary Settings

The Boundary section defines the global boundary conditions. It is defined before the first *Step* section. These conditions remain enforced during the entire simulation. The Boundary section defined in each Step section lists the boundary conditions that will remain active only during this step. Currently, the only types of boundary conditions that can be enforced this way are the prescribed displacement, nodal forces and rigid contact boundary conditions.

5.2. An Example

The following example illustrates the use of the multi-step feature of FEBio. This problem defines two steps. In the first step, a single element is stretched using a prescribed boundary condition. In the second step, the boundary condition is removed and the analysis type is switched from quasi-static to a dynamic analysis. Note the presence of the global fixed boundary constraints, which will remain enforced during both steps:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<febio_spec version="1.0">
	<Control>
		<title>Multi-step example</title>
	</Control>
	<Material>
		<material id="1" name="Material 1" type="neo-Hookean">
			<density>1.0</density>
			<E>1</E>
			<v>0.45</v>
		</material>
```

```
	</Material>
	<Geometry>
		<Nodes>
			<node id="1">-2.0,-0.5, 0.0</node>
			<node id="2">-2.0,-0.5, 1.0</node>
			<node id="3">-2.0, 0.5, 0.0</node>
			<node id="4">-2.0, 0.5, 1.0</node>
			<node id="5"> 2.0,-0.5, 0.0</node>
			<node id="6"> 2.0,-0.5, 1.0</node>
			<node id="7"> 2.0, 0.5, 0.0</node>
			<node id="8"> 2.0, 0.5, 1.0</node>
		</Nodes>
		<Elements>
			<hex8 id="1" mat="1">1,5,7,3,2,6,8,4</hex8>
		</Elements>
	</Geometry>
	<Boundary>
		<fix>
			<node id="1" bc="xyz"></node>
			<node id="2" bc="xy"></node>
			<node id="3" bc="xz"></node>
			<node id="4" bc="x"></node>
			<node id="5" bc="yz"></node>
			<node id="6" bc="y"></node>
			<node id="7" bc="z"></node>
		</fix>
	</Boundary>
	<LoadData>
		<loadcurve id="1">
			<loadpoint>0,0</loadpoint>
			<loadpoint>1,0.1</loadpoint>
		</loadcurve>
	</LoadData>
	<Step>
		<Control>
			<time_steps>10</time_steps>
			<step_size>0.1</step_size>
		</Control>
		<Boundary>
			<prescribe>
				<node id="5" bc="x" lc="1">1</node>
				<node id="6" bc="x" lc="1">1</node>
				<node id="7" bc="x" lc="1">1</node>
				<node id="8" bc="x" lc="1">1</node>
			</prescribe>
		</Boundary>
	</Step>
	<Step>
		<Control>
			<time_steps>50</time_steps>
			<step_size>0.5</step_size>
			<analysis type="dynamic"></analysis>
		</Control>
	</Step>
</febio_spec>
```

Chapter 6 - Parameter Optimization

This chapter describes FEBio's parameter optimization module. This module tries to estimate material parameters by solving an inverse finite element problem, where the "solution" of the problem is known, and the problem's material parameters are sought. In this case, the solution will often be an experimentally determined reaction force curve, and the unknown parameters are the material parameters that will recreate the reaction force curve by solving a forward FE problem. The user will need to input this reaction force curve as well as an initial guess for the material parameters. This is done using a separate input file which is described next.

6.1. Optimization input file

Like all FEBio input files, the parameter optimization input file is an xml-formatted text file. The following sections are defined:

- *Model*: defines the FE problem input file.
- *Options*: optional section defining the optimization control parameters
- *Function*: defines the parameter that generates the function values of the solution that are to be fitted.
- *Parameters*: defines the material parameters that are to be determined
- *LoadData*: defines the experimental reaction force using load curves.

In the following paragraphs each section will be explained in detail.

6.1.1. Model section

The model section defines the file name for the FE input problem. This input file defines the problem that will be solved by FEBio repeatedly. This xml-tag only takes a single value, namely the name of a valid FEBio input file. For example,

```
<Model>ex1.feb</Model>
```

6.1.2. Options section

This section defines the control parameters for the optimization. What options can be defined will depend on the chosen optimization method, which is set with an attribute of the Options section. Since only one optimization method is currently available, the attribute can be omitted. Note that this section is optional. When omitted, default values for the control parameters are chosen. The following parameters can be defined.

Parameter	Description	Default
`obj_tol`	Convergence tolerance for objective function. (1)	0.001
`f_diff_scale`	Forward difference scale factor (2)	0.001

Comments:

1. The objective function that is to be minimized is a χ^2 function of the reaction force values.

$$\chi^2(\mathbf{a}) = \sum_{i=1}^{n}\left[\frac{y_i - y(x_i;\mathbf{a})}{\sigma_i}\right]^2$$

Here, $y(x;\mathbf{a})$ is the function that describes the model, **a** is a vector with the (unknown) material parameters and the (x_i, y_i) are the experimentally (or otherwise) obtained data with standard deviation σ_i that approximates the ideal model.

2. The optimization method currently implemented requires the calculation of the gradient of the model function *y* with respect to the model parameters **a**. Since this gradient is not known, FEBio will approximate it using forward differences. For example, the *k*-th component of the gradient is approximated as follows.

$$\frac{\partial y}{\partial a_k} \approx \frac{1}{\delta a_k}\left[y(a_1,\cdots,a_k+\delta a_k,\cdots,a_m) - y(a_1,\cdots,a_k,\cdots,a_m)\right]$$

The value for δa_k is determined from the following formula.

$$\delta a_k = \varepsilon(1+a_k)$$

where, ε is the forward difference scale factor which can be set by the user with the *fdiff_scale* option.

6.1.3. Function section

This section defines the parameter that will generate the function values that are to be fitted to the experimental data. Currently, only one function parameter can be defined and it is defined using the *fnc* tag. For example,

```
<Function>
      <fnc lc="1">rigid.Fx</fnc>
</Function>
```

This example defines the *x*-component of a rigid body material named *rigid* as the model's function parameter. The *lc* attribute defines the loadcurve number that describes the model's experimental data.

Note that currently only rigid body materials can be used to define the function parameter, and only using the following data components.

Component	Description
Fx	x-component of the rigid body reaction force
Fy	y-component of the rigid body reaction force
Fz	z-component of the rigid body reaction force

Mx	x-component of the rigid body reaction moment
My	y-component of the rigid body reaction moment
Mz	z-component of the rigid body reaction moment

6.1.4. Parameters Section

This section defines the material parameters that are to be determined. Each parameter is defined using the *param* element. For example,

```
<Parameters>
      <param name="mat1.E">1.0, 0.0, 5.0</param>
      <param name="mat1.v">0.1, 0.0, 0.5</param>
</Parameters>
```

This example defines two material parameters. The *name* attribute gives the name of the parameter that is to be determined. The first component of the name (here *mat1*) is the name of the material as defined in the model input file (defined in the *Model* section). The second component (here *E* and *v*) are the names of the material parameters. Each parameter takes three values: the first value is the initial guess for this parameter, the second and third values are the minimum and maximum values respectively for this parameter[††].

6.1.5. LoadData Section

This section serves the same purpose as in the regular FEBio input file: it defines all the load curves that are used in the parameter optimization. Currently, only one load curve needs to be defined, namely the experimental data that the model is to be fitted to. The format is identical to that of the usual FEBio input file (see section 3.8).

Each load curve is defined through an array of value pairs (x_i, y_i), where the *x* component refers to the (simulation) time and the *y* component to is the function value.

6.2. Running a parameter optimization

As explained above, a parameter optimization problem is described using two input files. First, a standard FEBio input file that defines the geometry, materials, boundary conditions, etc. The second input file describes the parameter optimization data, such as the objective and which material parameters are to be optimized. The format of the second file is described above. A parameter optimization can only be initiated from the command line. For example,

```
>febio –s optim.feb
```

Comments:

1. To run an optimization file, you need the *–s* command line option instead of the *–i* option which runs a regular FEBio input file.

[††] In the current FEBio implementation, the range values are not used. However, in future versions these values will be used to clamp the material parameter's value within the specified range.

2. The input file (here *optim.feb*) is the name of the parameter optimization input file. The FE model input file does not need to be specified here, since it is specified in the optimization input file.

The output of a parameter optimization analyses is a log file that contains the screen output of the FEBio run as well as the optimized parameter values.

6.3. An example input file

Below follows a complete example of an optimization input file.

```
<?xml version="1.0"?>
<febio_optimize>
	<Model>ex1.feb</Model>
	<Options>
		<obj_tol>0.001</obj_tol>
		<f_diff_scale>0.001</f_diff_scale>
	</Options>
	<Function>
		<fnc lc="1">rigid.Fx</fnc>
	</Function>
	<Parameters>
		<param name="mat1.E">1, 0, 5</param>
		<param name="mat1.v">-0.5, 0, 0.5</param>
	</Parameters>
	<LoadData>
		<loadcurve id="1">
			<point>0.0, 0</point>
			<point>0.5, 1</point>
			<point>1.0, 2</point>
		</loadcurve>
	</LoadData>
</febio_optimize>
```

Comments:

1. Notice that the xml root element is *febio_optimize* for the optimization input file.
2. Here, the FEBio input file that contains the actual FE model data is *ex1.feb*. This file is a standard FEBio input file that defines all geometry, materials, boundary conditions and more.
3. The *Options* section is included here, but can be omitted. If omitted default values will be used for all control parameters.
4. The function parameter and material parameters are defined through “material name”.”parameter name” pairs. The name of the materials and their parameters are defined in the FEBio input file (in this example, *ex1.feb*).

Appendix A. Fixed Format Input

This chapter describes the fixed format for the FEBio input file. Each section of the input file must follow the order specified in this document. This also applies for each line of each section and for each entry on each line. In addition, every entry on each line has a fixed number of columns where its value may appear on the line. The user must pay careful attention to these restrictions, since FEBio may not be able to find all input mistakes related to incorrectly placed parameters. This format is identical to the format used by NIKE3D [1], developed and maintained by the Methods Development Group at Lawrence Livermore National Laboratory, in the sense that FEBio can read and run most NIKE3D input files, though FEBio does not support all NIKE3D features. Although our experience has shown that all FEBio input files using this format can be run with NIKE3D, we have not performed exhaustive testing. The input format for NIKE3D is detailed in the NIKE3D User's Manual. This manual only describes the features that are supported by FEBio.

The fixed format for the input file is composed of a series of sections:

1. Control section
2. Material section
3. Nodal coordinates section
4. Element connectivity section
5. Rigid node and facet section
6. Contact section
7. Load curve section
8. Concentrated nodal force section
9. Pressure boundary section
10. Prescribed displacement section
11. Body force section

Of all these sections, only the first four sections will appear in every input file. The other sections are optional and only need to be specified if the corresponding entry in the control section is non-zero. Note that all sections must appear in the order they are described in this manual. The rest of this chapter describes the sections in detail.

Note that any line that contains an asterisk ("*") in the first column is interpreted as a comment and is ignored by FEBio.

A.1. Control Section

The control section contains entries that describe the size of the problem as well as the solver parameters. It is composed of ten lines, and although the last three lines are ignored in FEBio, their presence is required for compatibility with NIKE3D.

A.1.1. Control Line 1

Columns		Format
1 – 72	Heading to appear on output. (1)	A256

Comments.

1. This line contains the problem title. In NIKE3D this line is limited to 72 characters, but in FEBio this line can be up to 256 characters. Note however that in FEBio (as well as in NIKE3D) only the first forty characters are saved to the plot file, a limitation that is imposed by the TAURUS file format used by NIKE3D and other LLNL codes including DYNA3D.

A.1.2. Control Line 2

Columns		Format
1 – 2	Input format (1)	A2
3 – 5	Number of materials	I3
6 – 15	Number of node points	I10
16 – 25	Number of solid elements (2)	I10
26 – 35	(*not used*)	
36 – 45	Number of shell elements (3)	I10
46 – 50	(*not used*)	
51 – 55	Number of sliding interfaces	I5
56 – 65	(*not used*)	
66 – 70	Number of rigid nodes and facet lines	I5

Comments

1. The input format must be "FL", which corresponds to NIKE3D version 3.0 and later.
2. This number includes all hexahedral ("brick"), pentahedral ("wedge") and tetrahedral elements.
3. Currently FEBio only supports rigid shells. These are shells that are assigned to a rigid material. FEBio also ignores shell thicknesses.

A.1.3. Control Line 3

Columns		Format
1 – 10	Number of time or load steps (*ntime*)	I10
11 – 20	Time or load step size (*dt*)	E10.0
21 – 25	Auto time/load step control flag (1) EQ. " ": use fixed time step size EQ. "AUTO": enable standard automatic time step control	1x, A4
26 – 30	Maximum number of retries allowable per step (2) EQ.0: default = 5	I5
31 – 35	Optimal number of iterations per step (3) EQ.0: default = 11	I5
36 – 45	Minimum allowable step/load size EQ.0: default = *dt / 3*	E10.0
46 – 55	Maximum allowable step/load size EQ.0: default = *dt*3*	E10.0

Comments

1. The automatic time stepping algorithm will adjust the time step size based on convergence information returned by the quasi-Newton solver. The time step size is bracketed by the minimum and maximum values that are input on this line. The algorithm also ensures that FEBio attempts to obtain a converged state at the desired termination time, determined by (*ntime* × *dt*).
2. This is the maximum number of attempts that the auto time stepper retries a failed quasi-Newton iteration. After each attempt the auto time stepper decreases the time step and restarts the iteration. When the minimum time step size is reached before the maximum number of iterations, the program ends in *error termination*.
3. The optimal number of iterations is the expected average number of iterations for each time step. By comparing this number with the actual number of iterations, the auto time stepper decides whether to increase or decrease the time step size.

A.1.4. Control Line 4

Columns		Format
		I5
1 – 5	Number of load curves (1)	
6 – 10	Maximum number of points defining any load curve (2)	I5
11 – 15	Number of concentrated nodal loads	I5
16 – 20	Number of element surfaces with applied pressure loading	I5
21 – 25	Number of displacement boundary conditions	I5
26 – 35	(*not used*)	I5
36 – 40	Body force loads due to base acceleration in x-direction EQ.0: no x-acceleration NE.0: x-acceleration	I5
41 – 45	Body force loads due to base acceleration in y-direction EQ.0: no y-acceleration NE.0: y-acceleration	I5
46 – 50	Body force loads due to base acceleration in z-direction EQ.0: no z-acceleration NE.0: z-acceleration	I5

Comments

1. A *loadcurve* is simply a list of data pairs that represents a curve of time versus load. The load can be interpreted in different ways. For example, it might represent a rigid body displacement, a concentrated nodal force, or any other variable that may depend on time.
2. This number is ignored in the current version of FEBio.

A.1.5. Control Line 5

Columns		Format
1 – 35	(*not used*)	
36 – 40	Dump file creation flag (1) EQ.0: no dump file will be created EQ.1: dump file will be created	I5

Comments

1. The dump file allows the user to restart an unfinished problem at a later time. When the flag is set to 1, FEBio will create a dump file (called *f3dump*) that can be used to restart the analysis from the last converged time step.

A.1.6. Control Line 6

Columns		Format
1 – 30	(*not used*)	
31 – 35	Maximum number of Quasi-Newton (1) equilibrium iterations permitted between stiffness matrix reformations. EQ.0: default set to 10	I5
36 – 40	Maximum number of stiffness matrix reformations per time step (2) EQ.0: default set to 15	I5
41 – 50	Convergence tolerance on displacements EQ.0: default set to 0.001	E10.0
51 – 60	Convergence tolerance on energy EQ.0: default set to 0.01	E10.0
61 – 70	Convergence tolerance on residual EQ.0: default set to 1E+10 (i.e. deactivated)	E10.0
71 – 80	Convergence tolerance on line search EQ.0: default set to 0.9	E10.0

Comments

1. FEBio uses the BFGS method [26] to solve the nonlinear finite element equations. In this method the stiffness matrix does not get recalculated during every iteration (such as in the Full-Newton method). In stead it calculates an approximation of the (inverse of the) stiffness matrix, which is called a *stiffness update*. The second parameter on this control line sets the maximum number of such updates (one during every iteration) before the exact stiffness matrix is reformed.
2. This parameter sets the maximum number of stiffness reformations per time step. If this number is reached the time step fails and, if the auto-time stepper is activated, a smaller time step will be tried.

A.1.7. Control Line 7

Columns		Format
1 – 5	Analysis type EQ. 0: quasi-static analysis	I5

A.1.8. Control Line 8

This line is not used in FEBio.

A.1.9. Control Line 9

This line is not used in FEBio.

A.1.10. Control Line 10

This line is not used in FEBio.

A.2. Material section

The *Material Section* contains all the material parameters of all the materials used in the problem. Repeat the following set of eight lines for each material. If the material is used for shell elements then an additional two lines are expected. Although FEBio reads those lines in, it currently ignores the values. The total number of materials is entered on control line 2. For detailed descriptions of the materials, please see Section 3.3 above. Note that FEBio does not support all materials using the fixed format. It only supports those that are defined in NIKE3D. This was done to guarantee compatibility with NIKE3D: any file in the fixed format can also be run with NIKE3D.

A.2.1. Line 1

Columns		Format
1 – 5	Material identification number	I5
6 – 10	Material Type EQ.1: neo-Hookean material EQ.15: Mooney-Rivlin EQ.18: Transversely isotropic Mooney-Rivlin EQ.20: Rigid body	I5
11 – 20	Density	E10.0
21 – 25	Element class for which this material is used EQ.20: brick element (also pentahedral and tetrahedral element) EQ.2: shell element	I5

A.2.2. Line 2

Columns		Format
1 – 72	Material title	

A.2.3. Material Type 1 – Neo-Hookean

Columns			Format
1 – 10	**Line 3**	Young's modulus	E10.0
1 – 10	**Line 4**	Poisson's ratio	E10.0
	Line 5-8	blank	

A.2.4. Material Type 15 – Mooney Rivlin Hyperelastic

Columns			Format
1 – 10	**Line 3**	Coefficient of first invariant *A*	E10.0
1 – 10	**Line 4**	Coefficient of second invariant *B*	E10.0
1 – 10	**Line 5**	Poisson's ratio ν	E10.0
	Line 6-8	blank	

A.2.5. Material Type 18 – Transversely Isotropic Hyperelastic

Columns			Format
1 – 10	**Line 3**	Isotropic material coefficient C_1	E10.0
11 – 20		Isotropic material coefficient C_2	E10.0
21 – 30		Exponential stress coefficient C_3	E10.0
31 – 40		Fiber uncrimping coefficient C_4	E10.0
41 – 50		Modulus of straightened fibers C_5	E10.0
1 – 10	**Line 4**	Bulk modulus *K*	E10.0
11 – 20		Fiber stretch for straightened fibers λ_m	E10.0
1 – 10	**Line 5**	(*not used*)	
	Line 6	Material axes option, *AOPT* Fiber axis is always aligned with load axis ***a***	E10.0
		EQ.0: local material axes given by local element Nodes specified on line 7 below. Line 8 is blank	
		EQ.1: local material axes determined by a point in space and global location of each element integration point. Line 8 is blank.	
		EQ.2: local material axes determined by normalized vector **a**.	
1 – 30	**Line 7**	*AOPT*.EQ.0: local element nodes (default=1,2,4)	3E10.0
		AOPT.EQ.1: x_p, y_p, z_p	3E10.0
		AOPT.EQ.2: a_1, a_2, a_3	3E10.0
1 – 30	**Line 8**	(not used)	
31 – 40		AC – (active contraction) load curve number	E10.0
41 – 50		AC – intracellular calcium concentration	E10.0
51 – 60		AC – tension-sarcomere length relation constant	E10.0
61 – 70		AC – Unloaded sarcomere length	E10.0
71 – 80		AC – no tension sarcomere length	E10.0

The local fiber direction is specified using the AOPT parameter. For AOPT=0 the material axis is defined by the local element node numbering specified on line 7. For AOPT=1 the material axis is defined a fixed point in space and the global position of the element Gauss points. The normalized vector connecting these two points defines the axis. For AOPT=2 the material axis is defined by a global vector specified on line 7.

A.2.6. Material type 20 – Rigid Body

Columns			Format
	Line 3	blank	
	Line 4	blank	
1 – 10	**Line 5**	*X*-translational boundary code	E10.0
11 – 20		*Y*-translational boundary code	E10.0
21 – 30		*Z*-translational boundary code	E10.0
31 – 40		*X*-rotational boundary code	E10.0
41 – 50		*Y*-rotational boundary code	E10.0
51 – 60		*Z*-rotational boundary code	E10.0
1 – 10	**Line 6**	Center of mass input flag NE.1: FEBio computes center of mass EQ.1: read user supplied coordinates below	E10.0
11 – 20		*X* – coordinate of center of mass	E10.0
21 – 30		*Y* – coordinate of center of mass	E10.0
31 – 40		*Z* – coordinate of center of mass	E10.0
	Line 7-8	blank	

Nodal constraints and displacement boundary conditions on rigid body *nodes* are ignored. Instead, constraints are applied at the rigid body center of mass. The coordinates of the center of mass are either computed by FEBio or may be input by the user. The boundary condition codes are interpreted as:

code < 0:	fixed
code = 0:	free
code > 0:	number of load curve for prescribed displacement or rotation (in radians).

A.3. Nodal coordinates section

The following line is repeated for every node of the mesh. The total number of nodes is entered on line 2 of the Control section.

Columns		Format
1 – 8	Node number	I8
9 – 13	Displacement boundary condition code EQ.0: no constraints EQ.1: constrained *x* displacement EQ.2: constrained *y* displacement EQ.3: constrained *z* displacement EQ.4: constrained *x* and *y* displacement EQ.5: constrained *y* and *z* displacement EQ.6: constrained *z* and *x* displacement EQ.7: constrained *x, y* and *z* displacement	I5
14 – 33	*x* – coordinate	E20.0
34 – 53	*y* – coordinate	E20.0
54 – 73	*z* – coordinate	E20.0
74 – 78	(ignored)	

A.4. Element connectivity section

The following line is repeated for every element in the mesh. The total number of elements is entered on line 2 of the Control section.

Columns		Format
1 – 8	Element number	I8
9 – 13	Material number	I5
14 – 21	Node point 1	I8
22 – 29	Node point 2	I8
…	…	
70 – 77	Node point 8	I8

Nodes 1 through 8 define the corner nodes of the 8-node hexahedral or “brick” element. In the case of a pentahedral or “wedge” element the sixth node is repeated for entry 7 and 8. In the case of a tetrahedral element, the fourth node is repeated four times.

- hex : $n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8$
- penta : $n_1, n_2, n_3, n_4, n_5, n_6, n_6, n_6$
- tet : $n_1, n_2, n_3, n_4, n_4, n_4, n_4, n_4$

A.5. Rigid node and facet section

The number of rigid node and facet lines is defined in the control section on line 2.

Columns		Format
1 – 5	Rigid body number	I5
6 – 13	Node point 1	I8
14 – 21	Node point 2	I8
22 – 29	Node point 3	I8
30 – 37	Node point 4	I8

Rigid nodes and facets move as part of the indicated rigid body. The input is a very general list of node points, four (or less) per line. The nodes need not be unique and need not form a facet of an element, although this method of input is often convenient. This feature gives a convenient way to attach a portion of a deformable mesh to a rigid body without merging node points or using a tied interface. Note that the nodes in this section must be part of a deformable mesh; this section cannot be used to define new geometry.

A.6. Contact section

Contact between objects is defined via sliding interface surfaces. The number of sliding interfaces is defined on control line 1 of the control section. Each sliding interface consists of a master surface and a slave surface. When using a two-pass algorithm for enforcement of the contact constraint, the definition of master and slave surfaces is arbitrary. When using the single-pass algorithm, the results can be influenced by the choice of slave and master surfaces. It is best to use the most tessellated surface as the master. The master and slave contact surfaces are described by "facets" of existing elements in the model. A facet is defined by the node numbers that construct either a quadrilateral or triangular polygon. For each sliding interface the following lines are repeated.

A.6.1. Control line

Columns		Format
1 – 8	Number of slave facets	I8
9 – 16	Number of master facets	I8
17 – 20	Interface type EQ.3: two-pass algorithm EQ.-3: single-pass algorithm	I4
21 – 30	Penalty factor (1)	E10.0
31 – 75	(not used)	
76 – 80	Auxiliary interface control line flag (IAUG) EQ.1: read an auxiliary interface control line flag immediately Following this line.	I5

Comments

1. The penalty factor must be entered as a negative number to be consistent with Nike3D, since in Nike3D a positive number is interpreted as a scale factor to an automatically calculated penalty factor. If a negative number is entered, the absolute value is used as the actual penalty factor. In FEBio the absolute value of this number is always used as the actual penalty factor.

A.6.2. Auxiliary control line

Add this control line immediately after the control line if IAUG equal 1.

Columns		Format
1 – 5	(not used)	
6 – 15	Normal direction convergence tolerance for augmentations (ALTOL) GT.0: converged when force norm < ALTOL	E10.0

EQ.0: DEFAULT = 0.1

A.6.3. Slave facet cards

For each slave facet repeat the following line. There must be at least three non-repeated node numbers but no more than four node numbers per line. Triangular facets are entered by repeating the last node.

Columns		Format
1 – 8	Slave facet number	I8
9 – 16	Node point n1	I8
17 – 24	Node point n2	I8
25 – 32	Node point n3	I8
33 – 40	Node point n4	I8

A.6.4. Master facet cards

The format for entering master facets is the same as for the slave facets. For each master facet repeat the following line.

Columns		Format
1 – 8	Slave facet number	I8
9 – 16	Node point n1	I8
17 – 24	Node point n2	I8
25 – 32	Node point n3	I8
33 – 40	Node point n4	I8

If there are multiple sliding interfaces, the order of the cards is as follows.

```
Control line 1 – sliding interface 1
Control line 2 – sliding interface 1 (optional)
...
Control line 1 – sliding interface n
Control line 2 – sliding interface n (optional)
Slave surface  – sliding interface 1
Master surface – Sliding interface 1
...
Slave surface  – sliding interface n
Master surface – Sliding interface n
```

A.7. Load curve section

A load curve is a list of data pairs that represent a discretized function of time. They are used to specify the magnitude and/or time variation of many types of data, including boundary conditions, material properties and motion of rigid materials. Any boundary condition or property may reference any load curve, and a load curve may be referenced more than once.

Define the number of load curves on Control line 4. Repeat the following lines for every load curve:

A.7.1. Line 1

Columns		Format
1 – 5	Load curve number	I5
6 – 10	Number of points in load curve (pts)	I5

A.7.2. Line 2,…,*pts*

Columns		Format
1 – 10	Time	E10.0
11 – 20	Load curve value	E10.0

The value of a time point in between two load points is calculated using linear interpolation. The value of a time point outside the domain of the load curve is clamped to the nearest data point.

A.8. Concentrated nodal force section

Define the number of concentrated nodal loads as specified on control line 4. Repeat the following line for every nodal load.

Columns		Format
1 – 8	Node point number on which this load acts	I8
9 – 13	Direction in which load acts EQ.1: *x* – direction force EQ.2: *y* – direction force EQ.3: *z* – direction force	I5
14 – 18	Load curve number	I5
19 – 28	Scale factor EQ.0: default set to "1.0"	E10.0

Concentrated nodal loads are defined in global coordinates. In contrast to "follower forces", concentrated nodal loads act in a fixed direction as over the course of the analysis. Hence, loads that were defined normal to a surface in the undeformed configuration may not remain normal if the surface undergoes large deformation. Pressure loads may be used to maintain normality during large deformation.

A.9. Pressure boundary section

Define the number of lines as specified on control card 4. Repeat the following line for every pressure load. For triangular facets repeat the third point.

Columns		Format
1 – 5	Load curve number	I5
6 – 13	Node point 1	I8
14 – 21	Node point 2	I8
22 – 29	Node point 3	I8
30 – 37	Node point 4	I8
38 – 47	Scale factor at node 1	E10.0
48 – 57	Scale factor at node 2	E10.0
58 – 67	Scale factor at node 3	E10.0
68 – 77	Scale factor at node 4	E10.0

Pressure forces are “follower forces”. They always remain normal to the surface even under large deformations. The magnitude of the pressure is determined by the load curve value and the scale factor. When a scale factor is zero, it is *not* replaced by the default value of 1.0. Only when all four scale factors are zero, then they are replaced with the default values.

A.10. Prescribed displacement section

Define the number of lines as specified on control line 4.

Columns		Format
1 – 8	Node point number to which this displacement is applied	I8
9 – 13	Global direction in which node is displaced EQ.1: translation in *x* – direction EQ.2: translation in *y* – direction EQ.3: translation in *z* – direction	I5
14 – 18	Load curve number	I5
19 – 28	Scale factor on load curve EQ.0: default set to "1.0"	E10.0

A.11. Body force section

The body force due to base acceleration is specified on Control card 4. Skip lines 1, 2, and/or 3 if the corresponding flag is zero. The sign convention for body force loads is understood by considering that the coordinate system, or base, is accelerated in the direction specified. Thus, a structure with its base fixed in the X-Y plane and extending upward in the +Z direction will be crushed by a +Z acceleration, and stretched by a –Z acceleration. Material mass density must be specified for all analyses using body forces.

A.11.1. Line 1

Columns		Format
1 – 5	Load curve number	I5
6 – 15	Scale factor on *X* acceleration EQ.0: default set to "1.0"	E10.0

A.11.2. Line 2

Columns		Format
1 – 5	Load curve number	I5
6 – 15	Scale factor on *Y* acceleration EQ.0: default set to "1.0"	E10.0

A.11.3. Line 3

Columns		Format
1 – 5	Load curve number	I5
6 – 15	Scale factor on *Z* acceleration EQ.0: default set to "1.0"	E10.0

Appendix B. Configuration file

As of version 1.2 FEBio requires a configuration file to run. The purpose of this file is to store platform specific settings such as the default linear solver. See section 2.5 for more information on how to use this file. In this section we detail the format of this file.
The configuration file uses an xml format. The root element must be *febio_config*. The required attribute *version* specifies the version number of the format. Currently this value must be set to "1.0". The following elements are defined.

Parameter	Description
`linear_solver`	Set the default linear solver for the platform. (1)

Comments:

1. FEBio supports several linear solvers, such as SuperLU, Pardiso, PSLDLT, Skyline …. Not all solvers are available for all platforms. Only the Skyline solver is available for all platforms and as of version 1.2 the SuperLU and the Pardiso solver are also available on most platforms.

References

[1] Maker, B. N., 1995, "NIKE3D: A nonlinear, implicit, three-dimensional finite element code for solid and structural mechanics," Lawrence Livermore Lab Tech Rept, UCRL-MA-105268.
[2] Bonet, J., and Wood, R. D., 1997, Nonlinear continuum mechanics for finite element analysis, Cambridge University Press.
[3] Simo, J. C., and Taylor, R. L., 1991, "Quasi-incompressible finite elasticity in principal stretches: Continuum basis and numerical algorithms," Computer Methods in Applied Mechanics and Engineering, 85, pp. 273-310.
[4] Arruda, E. M., and Boyce, M. C., 1993, "A Three-Dimensional Constitutive Model for the Large Stretch Behavior of Rubber Elastic Materials," J. Mech. Phys. Solids, 41(2), pp. 389-412.
[5] Veronda, D. R., and Westmann, R. A., 1970, "Mechanical Characterization of Skin - Finite Deformations," J. Biomechanics, Vol. 3, pp. 111-124.
[6] Holmes, M. H., and Mow, V. C., 1990, "The nonlinear characteristics of soft gels and hydrated connective tissues in ultrafiltration," J Biomech, 23(11), pp. 1145-1156.
[7] Ateshian, G. A., Warden, W. H., Kim, J. J., Grelsamer, R. P., and Mow, V. C., 1997, "Finite deformation biphasic material properties of bovine articular cartilage from confined compression experiments," J Biomech, 30(11-12), pp. 1157-1164.
[8] Iatridis, J. C., Setton, L. A., Foster, R. J., Rawlins, B. A., Weidenbaum, M., and Mow, V. C., 1998, "Degeneration affects the anisotropic and nonlinear behaviors of human anulus fibrosus in compression," J Biomech, 31(6), pp. 535-544.
[9] Puso, M. A., and Weiss, J. A., 1998, "Finite element implementation of anisotropic quasi-linear viscoelasticity using a discrete spectrum approximation," J Biomech Eng, 120(1), pp. 62-70.
[10] Quapp, K. M., and Weiss, J. A., 1998, "Material characterization of human medial collateral ligament," J Biomech Eng, 120(6), pp. 757-763.
[11] Weiss, J. A., Maker, B. N., and Govindjee, S., 1996, "Finite element implementation of incompressible, transversely isotropic hyperelasticity," Computer Methods in Applications of Mechanics and Engineering, 135, pp. 107-128.
[12] Fung, Y. C., 1993, Biomechanics : mechanical properties of living tissues, Springer-Verlag, New York.
[13] Fung, Y. C., Fronek, K., and Patitucci, P., 1979, "Pseudoelasticity of arteries and the choice of its mathematical expression," Am J Physiol, 237(5), pp. H620-631.
[14] Ateshian, G. A., and Costa, K. D., 2009, "A frame-invariant formulation of Fung elasticity," J Biomech, 42(6), pp. 781-785.
[15] Mow, V. C., Kuei, S. C., Lai, W. M., and Armstrong, C. G., 1980, "Biphasic creep and stress relaxation of articular cartilage in compression: Theory and experiments," J Biomech Eng, 102(1), pp. 73-84.
[16] Mow, V. C., Kwan, M. K., Lai, W. M., and Holmes, M. H., 1985, "A finite deformation theory for nonlinearly permeable soft hydrated biological tissues," Frontiers in Biomechanics, G. a. W. Schmid-Schonbein, SL-Y and Zweifach, BW, ed., pp. 153-179.
[17] Ateshian, G. A., Ellis, B. J., and Weiss, J. A., 2007, "Equivalence between short-time biphasic and incompressible elastic material response," J Biomech Eng, In press.

[18] Lanir, Y., 1983, "Constitutive equations for fibrous connective tissues," J Biomech, 16(1), pp. 1-12.
[19] Ateshian, G. A., Rajan, V., Chahine, N. O., Canal, C. E., and Hung, C. T., 2009, "Modeling the matrix of articular cartilage using a continuous fiber angular distribution predicts many observed phenomena," J Biomech Eng, 131(6), p. 061003.
[20] Ateshian, G. A., 2007, "Anisotropy of fibrous tissues in relation to the distribution of tensed and buckled fibers," J Biomech Eng, 129(2), pp. 240-249.
[21] Blemker, S., 2004, "3D Modeling of Complex Muscle Architecture and Geometry," Stanford University, Stanford.
[22] Criscione, J., Douglas, S., and Hunter, W., 2001, "Physically based strain invariant set for materials exhibiting transversely isotropic behavior," J. Mech. Phys. Solids, 49, pp. 871-897.
[23] Spencer, A. J. M., 1984, Continuum Theory of the Mechanics of Fibre-Reinforced Composites, Springer-Verlag, New York.
[24] Laursen, T. A., and Maker, B. N., 1995, "Augmented Lagrangian quasi-newton solver for constrained nonlinear finite element applications," International Journal for Numerical Methods in Engineering, 38(21), pp. 3571-3590.
[25] Ateshian, G., Maas, S., and Weiss, J. A., 2009, "Finite Element Algorithm for Frictionless Contact of Porous Permeable Media under Finite Deformation and Sliding," J. Biomech. Engn.
[26] Matthies, H., and Strang, G., 1979, "The solution of nonlinear finite element equations," Intl J Num Meth Eng, 14, pp. 1613-1626.