FEWarp

*Version 1.0*

*FEWarp* is a plugin developed for FEBio that implements the hyperelastic warping algorithm by Weiss et.al. (REF). This document describes how to use the warping plugin with FEBio.

Table of Contents

[1. Introduction 2](#_Toc390865328)

[2. FE model Setup 2](#_Toc390865329)

[2.1. The warping constraint 2](#_Toc390865330)

[template 2](#_Toc390865331)

[target 3](#_Toc390865332)

[range\_min 3](#_Toc390865333)

[range\_max 3](#_Toc390865334)

[blur 3](#_Toc390865335)

[penalty 3](#_Toc390865336)

[laugon 3](#_Toc390865337)

[altol 4](#_Toc390865338)

[2.2 The warping penalty 4](#_Toc390865339)

[2.3. Blurring 4](#_Toc390865340)

[2.4. Warping Output Variables 5](#_Toc390865341)

[3. Running a Warping Model 5](#_Toc390865342)

# 1. Introduction

Running a hyperelastic warping model requires a finite element model describing the model geometry and two sets of image data representing the template and the target images. The finite element model is represented by a standard FEBio input file with a warping constraint applied. We’ll see in the next section how to setup the warping constraint. The image data is typically a stack of raw image data stored in a raw file. Two raw files will be required, one for representing the template image and one for representing the target image.

# 2. FE model Setup

The FE model is defined in a standard FEBio input file and must include the *Control*, *Materials*, *Geometry*, *Boundary*, and *Output* sections. In addition, a *Constraints* section will be added to define the warping constraint and a *LoadData* section to define the warping load curves.

## 2.1. The warping constraint

In FEBio, the warping algorithm is implemented as a constraint. To define a warping constraint, add the following constraint to the *Constraints* section of the input file.

<constraint type="warp-image">  
 <!-- contents here -->  
</constraint>

The warping constraint parameters are defined next. The following table shows the available parameters.

|  |  |
| --- | --- |
| **Warping parameters** |  |
| template | Defines the template image file. |
| target | Defines the target image file. |
| range\_min | x,y,z minimum coordinates defining image domain |
| range\_max | x,y,z maximum coordinates defining image domain |
| blur | image blurring factor |
| penalty | warping penalty factor |
| laugon | augmentation flag |
| altol | augmentation tolerance |

We’ll look at each of these parameters in more detail next. At the end of this section an example is defined illustrating the proper syntax for all these parameters.

### template

The *template* parameter defines the image file for the template. Currently, only 8-bit raw images are supported. The filename is specified in the *file* attribute. The image size is specified in the *size* child element, which takes the x, y, and z dimensions in voxels. For example,

<template file="template.raw">  
 <size>64,64,64</size>  
</template>

### target

The *target* parameter defines the image file for the target. Currently, only 8-bit raw images are supported. The filename is specified in the *file* attribute. The image size is specified in the *size* child element, which takes the x, y, and z dimensions in voxels. For example,

<target file="target.raw">  
 <size>64,64,64</size>  
</target>

### range\_min

The *range\_min* parameter defines the minimum coordinates that define the image domain. In other words, they are the coordinates of the lower-left-bottom corner of the box bounding the image domain.

### range\_max

The *range\_max* parameter defines the maximum coordinates that define the image domain. In other words, they are the coordinates of the upper-right-top corner of the box bounding the image domain.

### blur

This parameter defines the blurring radius. Prior to evaluating the template and target images (and their gradients), the images are blurred. The blurring creates a larger overlap between target and image and may help in registration. Often, the blurring radius is defined via a loadcurve where the loadcurve ramps down from 1 to 0 so that at the final time step no blurring is applied. The blurring radius is defined in voxels.

### penalty

The *penalty* parameter defines the penalty factor that is used to scale the warping forces. In most applications, it is convenient to define a loadcurve for this parameter using the *lc* attribute. That way, the penalty will be ramped up over time, creating a better registration. Keep in mind, that a penalty that is too large may introduce instability in the solution process, resulting in poorer convergence. For best results we recommend using a modest penalty value in conjunction with augmentation (see *laugon* and *altol*).

### laugon

This parameter activates (or deactivates) Lagrangian augmentation. The augmented Lagrangian method is an iterative method to approximate the Lagrange multipliers in a constrained problem. In this case, the image registration is the constraint and the Lagrange multipliers are the forces needed to achieve registration. In each iteration a penalty step is applied followed by an update of the Lagrange multipliers. The iterations continue until the Lagrange multipliers have converged to within a user-defined tolerance (see *altol*). To activate augmented Lagrangian, set *laugon* to 1 (one). To deactivate it, set it to 0 (zero).

### altol

The *altol* parameter defines the convergence tolerance used by the augmented Lagrangian method (see *laugon*).

The following example shows a warping constraint section for a warping problem using augmented Lagrangian and a loadcurve for both the penalty and blur parameters.

<Constraints>

<constraint type="warp-image">

<template file="template.raw">

<size>64,64,1</size>

</template>

<target file="target.raw">

<size>64,64,1</size>

</target>

<range\_min>0,0,0</range\_min>

<range\_max>64,64,0</range\_max>

<penalty lc="1">1</penalty>

<blur lc="2">1</blur>

<laugon>1</laugon>

<altol>0.1</altol>

</constraint>

</Constraints>

## 2.2 The warping penalty

The penalty parameter directly scales the warping forces and consequently is the main parameter in controlling the result of the registration. In general, the larger the penalty, the better the registration.

In most applications it is recommended to use a loadcurve for the *penalty* parameter. This loadcurve is usually defined as a linear ramp from 0 to 1 and will be scaled by the value defined in the *penalty* parameter. The purpose of the penalty loadcurve is to apply the warping forces incrementally, because applying a large warping force in a single step might make the problem unstable and fail to converge.

## 2.3. Blurring

The purpose of blurring the images is that it often results in a smoother overlap between template and target. Blurring is advised when the template and target images have sharp boundaries. Keep in mind that the warping forces are proportional to the target gradient so the effective warping force is determined by both the penalty and the blur parameters.

It is also recommended to use a loadcurve for the *blur* parameter. This loadcurve is usually ramped down from 1 to 0 and will be scaled by the value defined in the *blur* parameter. The resulting value defines the blurring radius (in units of pixels). The result of the down ramp is to create a large area of overlap at the start of the registration, but recover the original template and target images toward the end of the registration.

## 2.4. Warping Output Variables

When running a warping analysis, the following additional output variables can be stored to the FEBio plot file.

|  |  |
| --- | --- |
| **warping output variables** |  |
| warp-template | warping template image data |
| warp-target | warping target image data |
| warp-energy | warping energy |
| warp-force | warping force |

To output a warping output variable, simply add it to the plotfile section. For instance,

<Output>

<plotfile type="febio">

<var type="displacement"/>

<var type="stress"/>

<var type="warp-template"/>

<var type="warp-target"/>

<var type="warp-energy"/>

</plotfile>

</Output>

# 3. Running a Warping Model

In order to run a warping model, the following files will be required.

* The FEBio2 executable.
* The FEBio configuration file, modified to load the warping plugin (see below).
* The FEWarp plugin.
* A warping input file, i.e. an FEBio input file with a warping constraint defined.
* The template and target image data files.

The first step to running a warping problem is to modify the FEBio configuration file so that it loads the warping plugin. The configuration file is usually called febio.xml and can be found in the same directory as the executable. Add the following line to the configuration file.

<import>C:/path/to/plugins/fewarp.dll</import>

Modify the path so that it points to the actual plugin file. When FEBio starts it will load the plugin and print a message whether loading the plugin was successful.

After this is done, running a warping problem is exactly the same as running any other FEBio problem. For example, assume the warping model is stored in a file called *warp.feb*.

>febio –i warp.feb

FEBio will now run the warping input file and generate the usual output files. If you defined additional warping output variables, they will be stored to the plotfile which can be viewed with FEBio’s dedicated post-processor PostView.