A brief user guide to Nifty Sim
Version 2.4

Contents

Introduction

Nifty Sim is a high-performance nonlinear dynamic finite element (FE) solver. It employs a total Lagrangian kinematic framework and explicit time integration. A key feature of the package is the option of graphics processing unit- (GPU-) based execution, which allows the solver to significantly out-perform equivalent commercial packages. Technical details of most features of the solver may be found in [5, 6, 7]. We would appreciate it if you would cite these publications when using Nifty Sim in your work.

1 Compiling Nifty Sim

Requirements:

- Cmake [`http://www.cmake.org`].
- CUDA (optional, required for GPU-based components) [`http://www.nvidia.com`].
- VTK (optional, required for visualisation components) [`http://www.vtk.org`].
- BOOST (optional, used for CPU-parallelsim, conflicts with CUDA) [`http://www.boost.org`]

The code can be built as follows (starting in the Nifty Sim root folder, `nifty_sim/`):
Create a `build` folder, and move to it:

```
>> mkdir build
>> cd build
```

Call ccmake in order to enter the build options:

```
>> ccmake ../
```

Press "c" to configure.
Now, set the following variables:

- `CMAKE_BUILD_TYPE`: [`Debug` | `Release` | `RelWithDebInfo` | `MinSizeRel`]
- `CMAKE_INSTALL_PREFIX`: installation root path
- `USE_CUDA`: [`On` | `Off`] to compile the GPU-based components
- `USE_VIZ`: [`On` | `Off`] to compile the visualisation components
- `USE_BOOST`: [`On` | `Off`] to compile the Boost-based CPU-parallel components

Press "c" again to reconfigure.
To compile the GPU components, you will need to set the following additional variables:
With CUDA toolkit versions < 5.0:

- `CUDA_SDK_ROOT_DIR`: path to the CUDA SDK, e.g. `/usr/local/cuda-sdk/C`
- `CUDA_TOOLKIT_ROOT_DIR`: path to the CUDA toolkit, e.g. `/usr/local/cuda`

With CUDA toolkit versions ≥ 5.0:

- `CUDA_SDK_ROOT_DIR`: path to the CUDA examples (packaged with toolkit) root directory, e.g. `/usr/local/cuda-sdk`. If correctly set, you should see another variable, `CUDA_SDK_COMMON_INCLUDE_DIR` pointing to the SDK sub-directory containing the helper headers, appearing on the advanced options screen.
- `CUDA_TOOLKIT_ROOT_DIR`: path to the CUDA toolkit, e.g. `/usr/local/cuda`

If Boost support is enabled, the location of the Boost libraries is normally automatically determined. Should this fail, or you wish to use other libraries than the default ones of your system, the path to the Boost thread library can be set manually in the advanced configuration settings, accessible by pressing "t". Boost and CUDA support are *mutually exclusive*.

To compile the visualisation components, you will need to set the following additional variable:

- `VTK_DIR`: directory containing Cmake configuration file for VTK, e.g. `/usr/lib/vtk-5.4`

Press "c" again to reconfigure after any changes, then press "g" to generate.
Generate the make file:

```
>> cmake .
```

Finally, build it:

```
>> make
```

and install:

```
>> make install
```

The `niftysim` executable should now be in `CMAKE_INSTALL_PREFIX/bin`, while libraries are in `CMAKE_INSTALL_PREFIX/lib`.

2 Running Nifty Sim

Nifty Sim is executed using

```
>> niftysim -x <file> [options]
```

Required inputs:

Switches	*Arguments*	*Description*
`-x`	`<file>`	XML model file name (full path and extension)

Optional inputs:

Switches	*Arguments*	*Description*
`-sport`	`<none>`	Use GPU execution[†]
`-plot`	`<none>`	Plot solution results[‡]
`-print`	`<NF|LNF|LNFS|ND|EK|ES>`	Print some solution results (for sanity check, really)
	`NF`	All nodal forces
	`LNF`	Nodal forces of loaded nodes

	LNFS	Sums of nodal forces of loaded nodes
	ND	All nodal displacements
	ES	Total strain energy of final configuration
	EK	Total kinetic energy in last time step
-output-prefix	<prefix>	Custom prefix for outputs requested in XML simulation definition (see Sect. 3.8).
-device	<device id>	Select CUDA device to run on. By default the most powerful device installed in your system is used.†
-export-mesh	<destination path>	Exports the simulation solid mesh and final configuration displacements as a VTK file, with the latter stored as an attribute labeled with "displacements".‡
-export-membrane	<destination path>	Export the simulation membrane mesh and final configuration displacements as a VTK file.‡
-export-submeshes	<destination prefix>	Export the meshes of the simulation's sub-models with their respective final displacements individually as VTK files. For an explanation of the sub-model concept, please refer to the relevant chapter in this document.‡

†Requires CUDA
‡Requires VTK

See usage information with

```
>> niftysim -help
```

3 Model files

Input to Nifty Sim is via XML model files, which define all aspects of an analysis. The root *element* of the file is a `Model` element. Model features and simulation settings are then defined by a hierarchy of child elements. For users unfamiliar with XML the following features should be noted:

- XML *elements* (not to be confused with finite elements!) comprise opening and closing tags, e.g. `<Nodes>...</Nodes>`, and everything in between. To avoid confusion with said finite elements, we will refer to them as *xelements* hereafter.
- Xelements may contain other xelements (called *child xelements*) and/or text.
- Xelement opening tags may include *attributes* which provide further information about the xelement, e.g. `<Nodes DOF="3" NumNodes="8">` indicates that this xelement defines a node listing, each node has 3 degrees of freedom, and there are 8 of them.
- Comments (ignored by the parser) may be included as `<!--this is a comment-->`.

Note that element and node numbers are 0-based, i.e. the first element is element 0.

3.1 An example

A simple example is given below. In this example a model comprising a single hexahedral finite element with neo-Hookean hyperelastic material properties is stretched by 30%.

```
======================================================================
<?xml version="1.0" encoding="utf-8"?>
<Model>
   <Nodes DOF="3" NumNodes="8">
      0.0 0.0 0.0
      0.1 0.0 0.0
      0.1 0.1 0.0
      0.0 0.1 0.0
      0.0 0.0 0.1
      0.1 0.0 0.1
      0.1 0.1 0.1
      0.0 0.1 0.1
   </Nodes>

   <Elements NumEls="1" Type="H8">
      0 1 2 3 4 5 6 7
   </Elements>

   <ElementSet Size="1">
      <Material Type="NH">
         <ElasticParams NumParams="2">
            1052 52249
         </ElasticParams>
      </Material>
      0
   </ElementSet>

   <Constraint DOF="0" LoadShape="POLY345" NumNodes="4" Type="Disp">
      <Nodes>
         1 2 5 6
      </Nodes>
      <Magnitudes Type = "UNIFORM">
         0.03
      </Magnitudes>
   </Constraint>

   <Constraint DOF="0" NumNodes="4" Type="Fix">
      <Nodes>
```

```
         0 3 4 7
      </Nodes>
   </Constraint>

   <Constraint DOF="1" NumNodes="8" Type="Fix">
      <Nodes>
         0 1 2 3 4 5 6 7
      </Nodes>
   </Constraint>

   <Constraint DOF="2" NumNodes="8" Type="Fix">
      <Nodes>
         0 1 2 3 4 5 6 7
      </Nodes>
   </Constraint>

   <SystemParams>
      <TimeStep>0.01</TimeStep>
      <TotalTime>1</TotalTime>
      <DampingCoeff>5</DampingCoeff>
      <HGKappa>0.075</HGKappa>
      <Density>1000</Density>
   </SystemParams>
</Model>
=====================================================================
```

Descriptions of the various components of a model file are given in subsequent sections.

3.2 Defining geometry and topology

Define a list of node coordinates and element connectivities:

```
<Nodes DOF="3" NumNodes="8">
   <!--Node coords listed here,
         (DOF*NumNodes) values expected-->
</Nodes>

<Elements NumEls="1" Type="H8">
   <!--Element node connectivities listed here,
         (NumEls*Nodes-per-element) values expected-->
</Elements>
```

3.2.1 Nodes

Used for defining nodal coordinates. Values are listed as

$$x_1 \quad y_1 \quad z_1 \quad x_2 \quad y_2 \quad z_2 \quad \ldots \quad x_{N_n} \quad y_{N_n} \quad z_{N_n},$$

where x_i y_i z_i are the coordinates of node i and N_n = `NumNodes`.
***Attributes*:**

- `DOF`: degrees of freedom per node (currently only 3D analyses are supported).
- `NumNodes`: the number of nodes included in the listing.

3.2.2 `Elements`

Used for defining finite elements in terms of their nodal connectivity. Values are listed as

$$n_1^1 \quad n_2^1 \ \ldots \ n_M^1 \quad n_1^2 \quad n_2^2 \ \ldots \ n_M^2 \ \ldots \ n_1^{N_e} \quad n_2^{N_e} \ \ldots \ n_M^{N_e},$$

where n_j^i is the j^{th} node of element i, N_e = `NumEls`, and M is the number of nodes per element.
***Attributes*:**

- `NumEls`: the number of elements in the listing.
- `Type`: element type, options include
  - `H8`: 8-node hexahedral elements with linear interpolation and reduced integration.
  - `T4`: 4-node tetrahedral elements with linear interpolation.
  - `T4ANP`: 4-node tetrahedral elements which use an improved nodal-averaged pressure formulation [4].

3.3 Reading Geometry from Files

Alternatively, NiftySim can read geometry definitions from GMSH and VTK unstructured-grid files. Reading of a GMSH file can be accomplished by placing a `MSHMesh` tag inside the model. The `MSHMesh` tag expects a type attribute whose definition is analogous to that of the `Elements` tag, i.e. it can hold a value of `T4`, `T4ANP`, or `H8`. The tag contains the path to the mesh file. This path can be either absolute (recommended) or is taken to be relative to the execution directory.
For example, the below model gets its geometry (a tetrahedral mesh) from a file called `box.msh`, placed in NiftySim's execution directory.

```
<Model>
  ...
  <MSHMesh Type="T4">
    box.msh
  </MSHMesh>
  ...
</Model>
```

The file readers can also perform some simple geometrical transforms on the loaded meshes. These transforms are limited to translation, rotation, and scaling. Translations are defined

with the `Translation` tag and scaling with the `ScaleFactor` tag. The format for rotations (`Rotation`) is

$$c_x\ c_y\ c_z\ \alpha_x\ \alpha_y\ \alpha_z$$

where the c_i are the components of the centre of rotation, the α_i denote the rotations about the coordinate system axes.
The order of application is rotation - scaling - translation. Example:

```
<Model>
  ...
  <MSHMesh Type="T4">
    box.msh
    <Translation>
      2 0 0
    </Translation>
    <Rotation>
      0 0 0 30 0 0
    </Rotation>
    <ScaleFactor>
      3.5
    </ScaleFactor>
  </MSHMesh>
  ...
</Model>
```

In this example the box is *first* rotated by 30° about the x-axis and coordinate-system origin, then scaled by a factor of 3.5, finally translated by two units in positive x-direction.

The behaviour of the `VTKMesh` tag, used for reading VTK mesh files, is identical to that of `MSHMesh`.

3.4 Defining materials

Define a material model and associate it with a group of elements:

```
<ElementSet Size="1">
   <Material Type="NHV">
      <ElasticParams NumParams="2">
         <!--Elastic material parameters listed here,
            (NumParams) values expected-->
      </ElasticParams>
      <ViscoParams NumIsoTerms="1" NumVolTerms="0">
         <!--Prony series parameters listed here,
            (NumIsoTerms + NumVolTerms)*2 values expected-->
      </ViscoParams>
   </Material>
   <!--Element numbers listed here,
```

```
        (Size) values expected
        or, if consecutive element numbers are included,
        only the first is required-->
</ElementSet>
```

3.4.1 `ElementSet`

Used for defining the group of elements with which the material is associated. Each element in the analysis must be associated with one and only one element set/material. The element numbers are listed. Alternatively, if consecutive elements are included in the list, only the first is required. A `Material` child xelement must be included.
***Attributes*:**

- `Size`: the number of elements in the set.

3.4.2 `Material`

Used for defining a material (constitutive) model. Hyperelastic materials must include an `ElasticParams` child xelement. Visco-hyperelastic materials must also include a `ViscoParams` child xelement.
***Attributes*:**

- `Type`: material model identifier, options include
  - `NH`: neo-Hookean hyperelastic
  - `AB`: Arruda-Boyce hyperelastic
  - `PY`: polynomial ($N = 2$) hyperelastic
  - `TI`: transversely isotropic hyperelastic
  - `NHV`: neo-Hookean visco-hyperelastic
  - `TIV`: transversely isotropic visco-hyperelastic

3.4.3 `ElasticParams`

Used for defining the elastic params for a material model. Values are listed as

$$\text{Param}_1 \quad \text{Param}_2 \ \ldots \ \text{Param}_{N_P},$$

where $N_P =$ `NumParams` and is different for each model – see appendix A for details.
***Attributes*:**

- `NumParams`: the number of elastic parameters in the model.

3.4.4 `ViscoParams`

Used for defining the Prony series parameters for a visco-hyperelastic model. Values are listed as

$$\alpha_1^I \quad \tau_1^I \quad \alpha_2^I \quad \tau_2^I \quad \ldots \quad \alpha_{N_{IP}}^I \quad \tau_{N_{IP}}^I \quad \alpha_1^V \quad \tau_1^V \quad \alpha_2^V \quad \tau_2^V \quad \ldots \quad \alpha_{N_{VP}}^V \quad \tau_{N_{VP}}^V,$$

where α_i^I and α_i^V are the i^{th} relaxation coefficients for the isochoric and volumetric terms, respectively, τ_i^I and τ_i^V are the i^{th} relaxation time constants for the isochoric and volumetric terms, respectively, N_{IP} = `NumIsoTerms`, and N_{VP} = `NumVolTerms`. See appendix A for details.

***Attributes*:**

- `NumIsoTerms`: the number of isochoric Prony series terms in the model (≥ 0).
- `NumVolTerms`: the number of volumetric Prony series terms in the model (≥ 0).

3.4.5 `Density`

Material mass density. If none is specified, the default density taken from the `SystemParams` is used. If neither is available, NiftySim will exit with an error.

3.5 Defining loads and boundary conditions

All loads and boundary conditions are defined using `Constraint` xelements:

```
<Constraint DOF="0" LoadShape="POLY345" NumNodes="4" Type="Disp">
   <Nodes>
      <!--Node numbers listed here,
         (NumNodes) values expected-->
   </Nodes>
   <Magnitudes Type="UNIFORM">
      <!--Load magnitudes listed here,
         the number of values expected varies-->
   </Magnitudes>
</Constraint>
```

Each constraint applies to a set of nodes and to one degree of freedom.

3.5.1 `Constraint`

Used for defining a constraint. Must contain a `Nodes` child xelement. If `Disp` or `Force` types are used, a `Magnitudes` child xelement is required also.

***Attributes*:**

- `DOF`: (optional) the degree of freedom to which the constraint applies (0, 1, 2, or "all"). Default: apply to all nodal DOFs.
- `SpecType`: Optional attribute specifying the boundary specification method:

  - `NODES` (default) Specification through a node index list
  - `NORMAL` Specification through surface normal, see below for details.

- `NumNodes`: the number of nodes to which the constraint applies (only required with node index specification).

- `Type`: the type of constraint, options include:

  - `Disp`: directly impose displacements on the listed nodes.
  - `Force`: directly impose forces on the listed nodes.
  - `Gravity`: Gravity body forces. Requires specification of the acceleration magnitude and direction with a `AccelerationMagnitude` and `AccelerationDirection` child node respectively. The direction is defined as a 3-component vector.
  - `Pressure`: Surface pressure. See "Surface Constraints" (3.5.4)
  - `Traction`: Surface traction. See "Surface Constraints" (3.5.4)
  - `Fix`: constrain the relevant DOF of the listed nodes to 0.

- `LoadShape`: the curve shape (time-course) of the imposed load (required for `Disp`, `Gravity`, and `Force` constraints), options include:

  - `POLY345`: polynomial: $L(t_r) = L_{\text{mag}}(10t_r^3 - 15t_r^4 + 6t_r^5)$.
  - `RAMP`: linear: $L(t_r) = L_{\text{mag}}t_r$.
  - `STEP`: full magnitude imposed in a single step: $L(t_r) = L_{\text{mag}}$.
  - `HILLY`: a peaky function: $L(t_r) = L_{\text{mag}}t_r \exp(\cos(4\pi t_r) - 1)$.

In the above $t_r = t/T$ is the current relative simulation time, t is the current absolute simulation time, T is the total simulation time, $L(t_r)$ is the amplitude of the imposed load over the simulation, and L_{mag} is the load magnitude defined in the `Magnitudes` xelement.

3.5.2 `Nodes`

Used for defining the nodes to which a constraint is applied. This can be done either by explicitly listing node indices, or - by using the `SpecType=''NORMAL''` attribute - through a boundary normal. If the latter method is chosen a `Normal` child tag is expected with a mandatory threshold attribute `ToleranceAngle` (in degrees). If the angle between a mesh-surface facet's normal and the passed normal is smaller than `ToleranceAngle`, all its vertices are added to the constraint node list.

List-mode

```
<Constraint Type=''TYPE'' NumNodes=''NUMBER'' LoadShape=''TYPE'' NumNodes=''NUMBER''>
  <Nodes>
   INDEX 1
   INDEX 2
   ...
  </Nodes>
  ...
</Constraint>
```

NORMAL-mode

```
<Constraint Type=''TYPE'' NumNodes=''NUMBER'' LoadShape=''TYPE'' SpecType=''NORMAL''>
  <Normal ToleranceAngle="ANGLE (DEG)">
     NX NY NZ
  </Normal>
  ...
</Constraint>
```

NORMAL-mode also supports the use of bounding volumes to restrict the regions of the mesh to which a constraint applies. The two currently supported bounding volume types are boxes and spheres. The feature is enabled with the following XML code.

Box

```
<Constraint Type=''TYPE'' NumNodes=''NUMBER'' LoadShape=''TYPE'' SpecType=''NORMAL''>
  <Normal ...>
    ...
  </Normal>
  <RestrictTo Type=''Box''>
    MIN_X MAX_X MIN_Y MAX_Y MIN_Z MAX_Z
  </RestrictTo>
  ...
</Constraint>
```

Sphere

```
<Constraint Type=''TYPE'' NumNodes=''NUMBER'' LoadShape=''TYPE'' SpecType=''NORMAL''>
  <Normal ...>
    ...
  </Normal>
  <RestrictTo Type=''Sphere''>
    RADIUS CENTRE_X CENTRE_Y CENTRE_Z
  </RestrictTo>
  ...
</Constraint>
```

3.5.3 Magnitudes

Used for defining the load magnitudes for Disp and Force constraints.
***Attributes*:**

- Type: the type of magnitude listing, options include:
  - UNIFORM: a single magnitude is applied to all listed nodes, one value required.
  - DIFFORM: a different magnitude is applied to each listed node, NumNodes values required.
  - FILE: magnitudes are stored in a separate file.

For FILE type, the name of an ascii text file is given instead of the magnitude values. The file must contain load values for each affected node for each time step in the analysis, i.e. (NumNodes*NumSteps) values. Values must be listed as

$$L_1^{t_1} \quad L_2^{t_1} \ \ldots \ L_{N_n}^{t_1} \quad L_1^{t_2} \quad L_2^{t_2} \ \ldots \ L_{N_n}^{t_2} \ \ldots \ L_1^{t_{N_s}} \quad L_2^{t_{N_s}} \ \ldots \ L_{N_n}^{t_{N_s}}$$

where $L_i^{t_j}$ is the load magnitude of the i^{th} node at time step j, N_n is the number of nodes affected by the constraint, and N_s is the number of time steps in the analysis.

3.5.4 Surface Constraints

The group of "surface constraints" supported by NiftySim comprises Pressure and Traction. These require that a part of the mesh surface be specified either as an explicit list of surface-facet connectivities, or through a surface normal (requires attribute SpecType=''NORMAL'' on Constraint xelement). A Nodes xelement must not be specified (undefined behaviour), instead if the former mode of boundary specification is used, the list of facet definitions is expected inside a Faces child-xelement, with the number of face-definitions to be read specified as an attribute NumFaces on the parent Constraint xelement. Alternatively, if the entire surface of the mesh is to be used, the list can be omitted if all is set for the value of the NumFaces attribute. Specifications through normals is analogous to the node constraint case.

If the SpecType=''NORMAL''-mode is *not* used, the surface-facet type must also be specified through a FaceType attribute on the corresponding Constraint xelement. This attribute can be set to either ''Quad'' or ''Tri'', and must be consistent with the type of the solid mesh (triangles for tetrahedral meshes, quads for hexahedral meshes). If the NumFaces attribute is set to all, or normals are used for defining the boundary, FaceType is optional as the surface type is determined automatically via the solid mesh type.

Examples are given below:

List-mode

```
<Constraint Type="Pressure" NumFaces="100" FaceType="Tri" LoadShape="RAMP">
  <Faces>
    0 1 2
    1 2 3
    ...
  </Faces>
  <Magnitude>1</Magnitude>
</Constraint>
```

all-mode

```
<Constraint Type="Pressure" NumFaces="all" LoadShape="RAMP">
  <Magnitude>1</Magnitude>
</Constraint>
```

Normal-mode

```
<Constraint Type="Pressure" SpecType="NORMAL" LoadShape="RAMP">
  <Normal ToleranceAngle="5">
    1 0 0
  </Normal>
  <Magnitude>1</Magnitude>
</Constraint>
```

The magnitude in the case of pressure constraints is specified by means of an `Magnitude` xelement which can hold a single floating point value. Tractions can be specified as either `UNIFORM`, in which case an xelement of the following type is used:

```
<Magnitudes Type="UNIFORM">
   X Y Z
</Magnitudes>
```

where `X`, `Y`, `Z` are the components of the uniformly applied traction. Alternatively, tractions can be specified per surface facet with

```
<Magnitudes Type="DIFFORM">
   X1 Y1 Z1
   ....
   XN YN ZN
</Magnitudes>
```

Doing so requires that the number of traction vectors specified match the number of facets in the `Faces` xelement.

3.6 Membrane and Shell Elements

3.6.1 Mesh Definition

Solid meshes can be wrapped in virtual membranes by including a `ShellElements` tag. There are two possibilities for adding a membrane to a simulation, either only a part of the simulation's geometry is covered, or all geometry is wrapped in membrane. For the prior configuration an explicit listing of surface facets is required. The format of such a list closely resembles the definition of solid mesh elements, consisting of an element type attribute - T3 for triangles (currently the only supported element type) - and a list of node index tuples.

```
<Model>
  ...
  <ShellElements Type="T3">
    0 2 3
    2 3 4
    1 5 6
    ....
  </ShellElements>
  ...
</Model>
```

The node indices appearing in this list refer to the node set of the solid mesh.
The second possibility is to specify "SURFACE" for the shell element type which leads to NiftySim automatically extracting the entire surface of the solid mesh.

```
<Model>
  ...
  <ShellElements Type="SURFACE" />
  ...
</Model>
```

With this option no particular order of the elements can be guaranteed.

3.6.2 Element Sets

One or more element sets form the second component of a membrane simulation definition. Membrane element sets consist of a list of element indices and a material definition. The element indices can be either explicitly given

```
<Model>
  <ShellElementSet Size="123">
     0
     1
     5
     ...
     276
     <Material ...>
       ...
     </Material>
  </ShellElementSet>
```

As an implicit sequence of indices (in the below example ranging from 150 to 273)

```
<Model>
  <ShellElementSet Size="123">
     150
     <Material ...>
        ...
     </Material>
  </ShellElementSet>
  ...
</Model>
```

Or, by specifying "all" for the element set size, all membrane elements are included in the element set.
The material specification is done in a child node of the `ShellElementSet` with the `Material` tag. The specification comprises a material type and parameters, in turn given by elasticity parameters and the membrane thickness and density. As of the time of writing, the

supported material types are linear (standard plane-stress linear elasticity), incompressible Neo-Hookean [1], and a rotation-free thin-shell element [2] with a neo-Hookean membrane component is available if bending stiffness is required. The linear elastic model requires a Young's modulus and a Poisson ratio; the Neo-Hookean model only takes one parameter which is interpreted as the material's shear modulus; the shell element takes two pairs of Young's moduli and Poisson ratios, where the second pair is used for the modelling of the bending response:

```
<Material Type="Linear">
  E nu
  <Density>rho</Density>
  <Thickness>t</Thickness>
</Material>
```

```
<Material Type="NeoHookean">
  mu
  <Density>rho</Density>
  <Thickness>t</Thickness>
</Material>
```

```
<Material Type="LinearShell">
  E nu E nu
  <Density>rho</Density>
  <Thickness>t</Thickness>
</Material>
```

Examples of membrane/shell simulations can be found in `box_membrane.xml`, `box_membrane_explicit_mesh.xml`, `box_shell.xml`, and `box_membrane_nonlin.xml`, in the `models` subdirectory below the source root directory.

3.7 Submodels

A submodel is defined with the `SubModel` tag. Submodels allow for a recursive definition of simulation models. This is particularly interesting when defining a simulation comprising multiple mesh files. Mixing mesh types, however, is not supported, i.e. all meshes must be of either `T4`, `T4ANP`, or `H8` type. Just a like a model, a submodel must contain a geometry specification, a complete list of element material definitions, and can have multiple constraints. SubModel Xelements do not require a `SystemParams` child node; if they do have one, it is wholly ignored.
A simple example of how to use this tag can be found in `submodel_demo.xml` in the `models` directory.

Submodels can also be used to assemble larger meshes from smaller building blocks by identifying nodes shared between meshes. This requires the specification of a distance threshold below which surface nodes of the constituent meshes are considered identical and shared. The distance specification is done with the `SubModelNodeMergerDistance` tag. An example of mesh merging can be found in `submodel_merging_demo.xml`.

3.8 Output requests

Request saving of a solution variable (in this case, every 100 steps):

```
<Output Freq="100">
   <Variable>U</Variable>
</Output>
```

3.8.1 `Output`

Used for defining a save request. Must contain a `Variable` child xelement.
***Attributes*:**

- `Freq`: the save frequency in solution steps.

By default the values are written to files in the directory containing the simulation file. In the case of displacements, the file is called `U.txt`, internal forces are written to `F.txt`, etc. A custom file prefix can be specified on the command line.

3.8.2 `Variable`

Used to specify a solution variable to save. Options include `U` (nodal displacements) and `F` (internal nodal forces - computed from element stresses), `S` (SPK stress), `E` (Green-Lagrange strain), `EKin` (total kinetic energy), `EStrain` (total strain energy).

3.9 Contact modelling

NiftySim provides to kinds of contact modelling: 1) contacts between deformable geometry and analytically defined rigid surfaces; 2) mesh-based contact modelling (deformable-body contacts and deformable-, rigid body contacts).

3.9.1 Analytical Surface Contacts

The rigid surfaces are defined with custom XML elements at `Model`-level. Options include:

- `ContactPlate`: 2D rectangular, flat surface. Specification format:

```
<ContactPlate>
  <a>XA YA ZA</a>
  <b>XB YB ZB</a>
  <c>XB YB ZB</c>
  <Disp>TX TY TZ</Disp>
  <SlvNodes ... />
</ContactPlate>
```

  The `a`, `b`, and `c` elements define three corners of the plate, s.t.

```
<ContactPlate>
  <a>0 0 0</a>
  <b>1 0 0</a>
  <c>0 1 0</c>
</ContactPlate>
```

defines the unit square with the plane normal pointing upward, $\mathbf{n} = (0, 0, 1)^{\mathrm{T}}$.

`Disp` defines the motion carried out by the plate during the simulation. The displacement formula for the corner $\mathbf{a}$ is given by:

$$\mathbf{a}(t_R) = \mathbf{a}_0 + t_R^2(10t_R - t_R^2(15 - 6t_R))\mathbf{d} \tag{1}$$

and analogously for the other corners of the plate. In (1) t_R was used to denote the relative time, i.e. current time divided by total simulation time, $\mathbf{d}$ was the input displacement vector.

- `ContactCylinder`: Cylinder defined by an axis, radius, length, and origin:

```
<ContactCylinder>
  <Origin>OX OY OZ</Origin>
  <Axis>AX AY AZ</Axis>
  <Radius>R</Radius>
  <Length>L</Length>
  <OrigDisp>OX OY OZ</OrigDisp>
  <RadChange>M</RadChange>
  <SlvNodes ... />
</ContactCylinder>
```

The role of `OrigDisp` is analogous to `Disp` in the contact plate case. The radius change `RadChange` follows the same time profile and is applied as follows:

$$r(t_R) = r_0 + t_R^2(10t_R - t_R^2(15 - 6t_R))M \tag{2}$$

- `ContactUSProbe`: An idealised TRUS (transrectal ultrasound) probe. In terms of shape and specification format it is mostly identical to `ContactCylinder` except for having a rounded, hemi-spherical tip:

```
<ContactUSProbe>
  <Origin>OX OY OZ</Origin>
  <Axis>AX AY AZ</Axis>
  <Radius>R</Radius>
  <Length>L</Length>
  <OrigDisp>OX OY OZ</OrigDisp>
  <RadChange>M</RadChange>
  <SlvNodes ... />
</ContactUSProbe>
```

All of these models require a list of solid-mesh nodes that can be affected by the contact, in a sub-element of the following type:

```
<SlvNodes NumNodes="N">
  NODE INDEX 1
  NODE INDEX 2
  ...
  NODE INDEX N
</SlvNodes>
```

3.9.2 Mesh-Based Contact Modelling

Contacts between deformable bodies and self-collisions are enabled in the analysis settings (Sec. 3.10) via `DoDeformableCollision` and `DoSelfCollision` xelements, respectively. E.g.:

```
<SystemParams>
  ...
  <DoDeformableCollision>1</DoDeformableCollision>
  ...
</SystemParams>
```

The extraction of the mesh surface and construction of the bounding volume hierarchies is fully automatic and no further user input is required. However, to accelerate repeated simulations on the same geometry the contact modelling data can be exported and subsequently imported with the command line switches `-precompute` and `-initialise-with`.

Rigid contact surfaces can be incorporated with `ContactSurface` xelements. The inline-definition format of rigid contact surfaces is analogous to the main simulation mesh format, i.e. a `Nodes` xelement followed by a `Elements` xelement. The only supported element type is `T3`, i.e. triangles.

Alternatively, rigid contact surfaces can be read from VTK PolyData files by employing a `VTKSurface` xelement to specify their location:

```
<ContactSurface>
  <VTKSurface Type="T3">
     /path/to/mesh.vtk
  </VTKSurface>
</ContactSurface>
```

Self-collision detection is a CPU-only feature. Mesh-based GPU contact modelling requires setting of the `USE_GPU_GP_CONTACT` flag at compile time, and CUDA hardware with at least Fermi architecture.

3.10 Analysis settings

Define a set of analysis settings:

```
<SystemParams>
   <TimeStep>0.001</TimeStep>
   <TotalTime>1.0</TotalTime>
   <DampingCoeff>5</DampingCoeff>
   <Density>1000</Density>
   <HGKappa>0.075</HGKappa>
   <ROM>0</ROM>
</SystemParams>
```

3.10.1 SystemParams

Used to enclose a set of analysis settings. Must contain `TimeStep`, `TotalTime`, and `DampingCoeff` child xelements. If `H8` elements are used in the analysis, an `HGKappa` child xelement must be included also. To use a reduced order model an `ROM` xelement must be included also. `Density` is the optional default mass density that is used when the simulation contains element sets without their own density value.

3.10.2 TimeStep

Used to specify a solution time step size.

3.10.3 TotalTime

Used to specify a total simulation time.

3.10.4 DampingCoeff

Used to specify a structural (Rayleigh) damping coefficient.

3.10.5 Density

Used to specify a mass density for elements in the model.

3.10.6 HGKappa

Used to specify an hourglass control parameter for reduced integration hexahedral (`H8`) elements. Not required for other elements.

3.10.7 ROM

Used to control use of reduced order modelling. A value of 0 or 1 (off/on) may be passed. If no `ROM` xelement is defined, reduced order modelling is switched off.

3.11 Reduced Order Modelling

Define a reduced basis for a reduced order model:

```
<ReducedBasis NumVectors="4" File="/home/rom.txt">
   <!--Basis vectors listed here,
      (NumVectors*NumNodes*3) values required.
      Or, if the File attribute is defined, leave this space blank-->
</ReducedBasis>
```

Reduced order modelling will only actually be used if an `ROM` xelement is used to switch it on – see Sect 3.10.7.

NB: Reduced order modelling is currently only implemented for GPU execution.

3.11.1 `ReducedBasis`

Used to define a set of basis vectors for a reduced order model. Values are listed as

$$\phi_1^1 \quad \phi_1^2 \ \ldots \ \phi_1^{N_v} \quad \phi_2^1 \quad \phi_2^2 \ \ldots \ \phi_2^{N_v} \ \ldots \ \phi_{N_d}^1 \quad \phi_{N_d}^2 \ \ldots \ \phi_{N_d}^{N_v}$$

where ϕ_j^i is the j^{th} component of vector i, $N_v =$ `NumVectors`, $N_d = 3N_n$, and N_n is the number of nodes in the model. Alternatively, if the `File` attribute is defined, these values must appear in the specified file.
***Attributes*:**

- `NumVectors`: the number of vectors comprising the basis.
- `File`: specify a file containing the reduced basis values.

See [7] for further details.

4 Units

No units are assumed in Nifty Sim; users must ensure that all units in their analysis specification are consistent.

5 Bugs and comments

Please send comments and bug reports to Zeike Taylor (ztaylor@itee.uq.edu.au).

References

[1] J Bonet, R D Wood, J Mahaney, and P Heywood. Finite element analysis of air supported membrane structures. *Computer Methods in Applied Mechanics and Engineering*, 190(5-7):579–595, November 2000.

[2] Fernando G. Flores and Eugenio Oñate. Improvements in the membrane behaviour of the three node rotation-free BST shell triangle using an assumed strain approach. *Computer Methods in Applied Mechanics and Engineering*, 194(6-8):907–932, February 2005.

[3] G A Holzapfel. *Nonlinear Solid Mechanics: A Continuum Approach for Engineering*. John Wiley & Sons, Chichester, 2000.

[4] G R Joldes, A Wittek, and K Miller. Non-locking tetrahedral finite element for surgical simulation. *Communications in Numerical Methods in Engineering*, 25(7):827–836, 2009.

[5] Z A Taylor, M Cheng, and S Ourselin. High-speed nonlinear finite element analysis for surgical simulation using graphics processing units. *IEEE Transactions on Medical Imaging*, 27(5):650–663, 2008.

[6] Z A Taylor, O Comas, M Cheng, J Passenger, D J Hawkes, D Atkinson, and S Ourselin. On modelling of anisotropic viscoelasticity for soft tissue simulation: numerical solution and GPU execution. *Medical Image Analysis*, 13(2):234–244, 2009.

[7] Z A Taylor, S Crozier, and S Ourselin. Real-time surgical simulation using reduced order finite element analysis. In *13th International Conference on Medical Image Computing and Computer Assisted Intervention*, Beijing, October 2010.

A Constitutive models

A.1 Hyperelastic models

Hyperelastic models are defined by a strain energy function $\Psi(\mathbf{C})$ which is a function of deformation, in this case the right Cauchy-Green deformation tensor $\mathbf{C} = \mathbf{F}^{\mathrm{T}}\mathbf{F}$. $\mathbf{F}$ is the deformation gradient. Nifty Sim constitutive models employ the well-known multiplicative split of the deformation gradient into isochoric (volume-preserving) and volumetric components. This allows the strain energy to be expressed as a sum of corresponding isochoric and volumetric terms also:

$$\Psi\left(\mathbf{C}\right) = \Psi^{\mathrm{iso}}\left(\bar{\mathbf{C}}\right) + \Psi^{\mathrm{vol}}\left(J\right), \tag{3}$$

where $\bar{\mathbf{C}} = J^{-2/3}\mathbf{C}$ is the modified right Cauchy-Green deformation and $J = \det \mathbf{F}$ is the volume ratio. Isochoric components are often expressed in terms of the principal invariants of $\bar{\mathbf{C}}$: $\bar{I}_1 = \mathrm{tr}\bar{\mathbf{C}}$, $\bar{I}_2 = \left[(\mathrm{tr}\bar{\mathbf{C}})^2 - \mathrm{tr}\bar{\mathbf{C}}^2\right]/2$. The second Piola-Kirchhoff stress is then given by

$$\mathbf{S} = \mathbf{S}^{\mathrm{iso}} + \mathbf{S}^{\mathrm{vol}} = 2\frac{\partial \Psi^{\mathrm{iso}}}{\partial \mathbf{C}} + 2\frac{\partial \Psi^{\mathrm{vol}}}{\partial \mathbf{C}}. \tag{4}$$

For further details of the mentioned quantities and the evaluation of the derivatives in (4), refer to [3].

Details of the models available in Nifty Sim are given below.

A.1.1 Neo-Hookean

Strain energy function:

$$\Psi^{\mathrm{NH}} = \frac{\mu}{2}\left(\bar{I}_1 - 3\right) + \frac{\kappa}{2}\left(J-1\right)^2, \tag{5}$$

where μ and κ are shear and bulk moduli, respectively.

Xelement definition:

```
<Material Type="NH">
   <ElasticParams NumParams="2">
       μ    κ
   </ElasticParams>
</Material>
```

A.1.2 Arruda-Boyce

Strain energy function:

$$\Psi^{\mathrm{AB}} = \mu \sum_{i=1}^{5} \frac{C_i}{\lambda_m^{2i-2}}\left(\bar{I}_1^i - 3^i\right) + \frac{\kappa}{2}\left(\frac{J^2-1}{2} - \ln J\right), \tag{6}$$

where μ and κ are shear and bulk moduli, respectively, λ_m is the locking stretch, and C_i, $(i = 1, \ldots, 5)$ are constants: $C_1 = 1/2$, $C_2 = 1/20$, $C_3 = 11/1050$, $C_4 = 19/7000$, $C_5 = 519/673750$.

Xelement definition:

```
<Material Type="AB">
   <ElasticParams NumParams="3">
       μ    λ_m    κ
   </ElasticParams>
</Material>
```

A.1.3 Polynomial ($N = 2$)

Strain energy function:

$$\Psi^{\mathrm{PY}} = \sum_{i+j=1}^{N} C_{ij}\left(\bar{I}_1 - 3\right)^i \left(\bar{I}_2 - 3\right)^j + \frac{\kappa}{2}\left(J-1\right)^2, \tag{7}$$

where C_{ij} are material parameters (related to the initial shear modulus as $\mu = 2(C_{10} + C_{01})$) and κ is the initial bulk modulus. Nifty Sim uses the popular form in which $N = 2$. It should be noted that Eqn (7) is slightly different from the model implemented in some commercial packages (e.g. Abaqus, Ansys), wherein the volumetric term is $\sum_{i=1}^{N} 1/D_i(J-1)^{2i}$, where D_i are material parameters. We adopt the simplification $N = 1$ for this term only and note the simple relation between D_1 and the intitial bulk modulus, $\kappa = 2/D_1$, which leads to the expression in (7).

Xelement definition:

```
<Material Type="PY">
   <ElasticParams NumParams="6">
       C10   C01   C20   C02   C11   κ
   </ElasticParams>
</Material>
```

A.1.4 Transversely isotropic

Strain energy function:

$$\Psi^{\mathrm{TI}} = \frac{\mu}{2}\left(\bar{I}_1 - 3\right) + \frac{\eta}{2}\left(\bar{I}_4 - 1\right)^2 + \frac{\kappa}{2}\left(J-1\right)^2, \tag{8}$$

where μ and κ are shear and bulk moduli, respectively, $\bar{I}_4 = \mathbf{a}\cdot\bar{\mathbf{C}}\mathbf{a}$ is a so-called pseudo-invariant of $\bar{\mathbf{C}}$ and the preferred material direction $\mathbf{a} = [a_1\ a_2\ a_3]^{\mathrm{T}}$, and η is an additional material parameter that controls stiffness in the preferred direction $\mathbf{a}$. The latter should be a unit vector, and will be normalised if it is not.

Xelement definition:

```
<Material Type="TI">
   <ElasticParams NumParams="6">
       μ   κ   η   a1   a2   a3
   </ElasticParams>
</Material>
```

A.2 Visco-hyperelastic models

Visco-hyperelastic constitutive models are defined by time-dependent strain energy functions $\hat{\Psi}$, expressed in the form of a convolution integral:

$$\hat{\Psi}\left(\Psi, t\right) = \int_0^t \alpha\left(t-s\right)\frac{\partial \Psi}{\partial s}\,\mathrm{d}s, \tag{9}$$

where t is time and Ψ is an underlying hyperelastic strain energy function. The relaxation functions $\alpha(t)$ commonly assume the form of a Prony series:

$$\alpha\left(t\right) = \alpha_\infty + \sum_{i=1}^{N_P}\alpha_i \mathrm{e}^{-t/\tau_i}, \tag{10}$$

where α_∞, α_i, and τ_i are positive real constants, and N_P is the number of Prony terms in the series. Using the condition $\alpha_\infty + \sum_{i=1}^{N_P}\alpha_i = 1$, it is only necessary to specify the constants α_i and τ_i. Viscoelasticity may be applied to either, or both, of the isochoric and volumetric strain energy terms. Two visco-hyperelastic models, based on the corresponding hyperelastic models defined above, are available in Nifty Sim: neo-Hookean (`NHV`) and transversely isotropic (`TIV`). Viscoelastic parameters are specified as described in Sect 3.4.4.