FCMLab: A Finite Cell Research Toolbox for MATLAB

N. Zander[a,*], T. Bog[a], M. Elhaddad[a], R. Espinoza[a], H. Hu[a], A. Joly[b], C. Wu[a], P. Zerbe[a], A. Düster[c], S. Kollmannsberger[a], J. Parvizian[d], M. Ruess[e], D. Schillinger[f], E. Rank[a]

[a] *Chair for Computation in Engineering, Technische Universität München, Arcisstr. 21, 80333 München, Germany*
[b] *École des Ponts ParisTech, 6 et 8 avenue Blaise Pascal, 77455 Marne-la-Valleé cedex 2, France*
[c] *Numerische Strukturanalyse mit Anwendungen in der Schiffstechnik, Technische Universität Hamburg-Harburg, Schwarzenbergstr. 95c, 21073 Hamburg, Germany*
[d] *Department of Mechanical Engineering, Isfahan University of Technology, Isfahan 84156 83111, Iran*
[e] *Aerospace Structures and Computational Mechanics, Delft University of Technology, Kluyverweg 1, 2629 HS Delft, The Netherlands*
[f] *Department of Civil Engineering, University of Minnesota, 500 Pillsbury Drive S.E., Minneapolis, 55455 MN, USA*

Abstract

The recently introduced Finite Cell Method combines the fictitious domain idea with the benefits of high-order finite elements. Although previous publications demonstrated the method's excellent applicability in various contexts, the implementation of a three-dimensional Finite Cell code is challenging. To lower the entry barrier, this work introduces the object-oriented MATLAB toolbox FCMLab allowing for an easy start into this research field and for rapid prototyping of new algorithmic ideas. The paper reviews the essentials of the methods applied and explains in detail the class structure of the framework. Furthermore, the usage of the toolbox is discussed by means of different two- and three-dimensional examples demonstrating all important features of FCMLab (http://fcmlab.cie.bgu.tum.de/).

Keywords: Finite Cell Method, fictitious domain methods, MATLAB, object-oriented finite elements, high-order finite elements, *p*-FEM

1. Introduction

For decades, the Finite Element Method (FEM) has been among the most prominent approaches to solve partial differential equations (PDEs) numerically. Many theoretical and algorithmic enhancements of the method now allow to simulate very complex scenarios in science and technology. What remains unchanged, however, is the central idea to describe the PDE's solution *and* the domain geometry using the *same* mesh. This dual-use of elements for the approximation of the geometric shape and solution characteristics implies e.g. that distorted elements have to be avoided, as they would degrade the numerical accuracy. For non-trivial geometries, this constraint complicates the mesh generation processes considerably and renders the method's intrinsic need for a geometry-conforming discretization as one often limiting factor in daily engineering practice.

Alternative numerical approaches try to drop this limitation by separating the discretization of the solution from that of the geometry. Prominent examples are meshless and element-free methods [1, 2]. Also the Generalized Finite Element, Immersed Boundary, Fictitious Domain, Fat Boundary, Cartesian Grid-FEM and certain variants of extended Finite Element Methods follow this approach [3–16]. For a comprehensive review see [17–19]. A common idea of these methods is to approximate the PDE's solution using a non geometry-conforming discretization (approximation mesh) and to capture the actual domain geometry on a separate geometry mesh. Many of these approaches use a rather fine approximation mesh on which linear shape functions are spanned. This *h*-FEM-like concept has been extended to higher orders by the Finite Cell Method (FCM), combining a fictitious domain approach with the benefits of high-order finite elements (*p*-FEM). The Finite Cell Method was first introduced in [20, 21], where its potential was demonstrated for linear-elastic examples in two and three dimensions. Various extensions of the FCM confirm its versatility in the context of topology optimization [22], geometrically nonlinear continuum mechanics [23], adaptive mesh-refinement [24–27], computational steering [28, 29], biomedical engineering [30], numerical homogenization [31], elastoplasticity [32], wave propagation in heterogeneous materials [33, 34], local enrichment for material interfaces [35], convection diffusion problems [36, 37], thin-walled structures [38], design-through-analysis and iso-geometric-analysis [26, 27, 39, 40], weak coupling of non-matching, multi-patch geometries [41], and multi-physical applications [42]. The potential of high-order-fictitious-domain methods is also demonstrated in the context of Kantorovich methods [43, 44], immersed B-Spline methods (I-Splines) [45, 6, 46], and eXtended Finite Element Methods (XFEM) [13–16].

Yet, research in this field essentially demands for the availability of a high-order Finite Element code, which is not trivial to implement. To lower this entry barrier, the goal of the present

*Corresponding author.
Email address: nils.zander@tum.de (N. Zander)

work is to introduce FCMLab, an object-oriented MATLAB toolbox tailored for quick prototyping of new algorithmic ideas. Similar open-source software projects have been launched recently in the communities of hp-FEM, XFEM, meshless methods, and isogeometric analysis (see e.g. [47–54]). The authors hope that this work will be helpful for researches interested in high-order fictitious domain methods. For this purpose, the second section of this paper briefly explains the essential concepts of high-order Finite Elements and of the Finite Cell Method. The most important parts of the code design are outlined in the third section. The fourth section explains how to use the toolbox by different examples in two and three dimensions.

2. High-order Finite Elements and Finite Cells

As indicated in the introduction, this section outlines the essential ideas of the Finite Cell Method. As a model problem, linear elasticity will be considered and some basics of the p-version of the Finite Element Method, being a prerequisite for the FCM will be sketched, briefly.

2.1. Model problem

Consider a two- or three dimensional domain Ω with boundary $\partial\Omega$ divided into Neumann and Dirichlet parts Γ_N and Γ_D, respectively, such that

$$\Gamma_N \cap \Gamma_D = \emptyset \quad \text{and} \quad \Gamma_N \cup \Gamma_D = \partial\Omega.$$

Assuming linear-elastic behaviour, the deformation $\boldsymbol{u}$ caused by a body force $\hat{\boldsymbol{b}}$ is described by the following set of partial differential equations:

$$\begin{aligned}
\nabla \cdot \boldsymbol{\sigma} + \hat{\boldsymbol{b}} &= \boldsymbol{0} && \forall\, \boldsymbol{x} \in \Omega && \text{(1a)}\\
\boldsymbol{\sigma} &= \boldsymbol{C} : \boldsymbol{\varepsilon} && \forall\, \boldsymbol{x} \in \Omega && \text{(1b)}\\
\boldsymbol{\varepsilon} &= \frac{1}{2}\left[\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^{\top}\right] && \forall\, \boldsymbol{x} \in \Omega && \text{(1c)}\\
\boldsymbol{\sigma} \cdot \boldsymbol{n} &= \hat{\boldsymbol{t}} && \forall\, \boldsymbol{x} \in \Gamma_N && \text{(1d)}\\
\boldsymbol{u} &= \hat{\boldsymbol{u}} && \forall\, \boldsymbol{x} \in \Gamma_D. && \text{(1e)}
\end{aligned}$$

Here, $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$ denote the stress and strain tensor, respectively, and $\boldsymbol{C}$ is the linear-elastic constitutive tensor. $\hat{\boldsymbol{t}}$ and $\hat{\boldsymbol{u}}$ are the traction and displacement vectors prescribed on the respective boundaries. $\boldsymbol{n}$ is the outward-pointing normal vector on the boundary.

Following the principal of virtual work described e.g. in [55–58], the above equation is transferred into its corresponding weak form:

Find $\boldsymbol{u} \in H^1(\Omega)$ such that

$$\begin{aligned}
a(\boldsymbol{u}, \boldsymbol{v}) &= f(\boldsymbol{v}) \quad \forall\, \boldsymbol{v} \in H^1(\Omega) && \text{(2a)}\\
\text{with} \quad a(\boldsymbol{u}, \boldsymbol{v}) &= \int_{\Omega} \boldsymbol{\varepsilon}(\boldsymbol{v}) : \boldsymbol{C} : \boldsymbol{\varepsilon}(\boldsymbol{u})\, \mathrm{d}\Omega && \text{(2b)}\\
\text{and} \quad f(\boldsymbol{v}) &= \int_{\Omega} \boldsymbol{v} \cdot \hat{\boldsymbol{b}}\, \mathrm{d}\Omega + \int_{\Gamma_N} \boldsymbol{v} \cdot \hat{\boldsymbol{t}}\, \mathrm{d}\Gamma, && \text{(2c)}
\end{aligned}$$

with H^1 denoting the Sobolev space of order 1.

On the basis of the weak formulation, the Finite Element Method is applied to approximate the analytical solution. To this end, the infinite-dimensional Sobolev space is restricted to an n-dimensional subspace $V_h \subset H^1$ being spanned by the basis $B = \{N_1, N_2, \ldots, N_n\}$. This allows to express the numerical approximation $\boldsymbol{u}_h \in V_h$ as a linear combination of basis functions:

$$\boldsymbol{u}_h = \sum_{i=1}^{n} \boldsymbol{N}_i \tilde{\boldsymbol{u}}_i, \tag{3}$$

with $\tilde{\boldsymbol{u}}_i$ denoting the degrees of freedom related to shape functions $\boldsymbol{N}_i$. Following a Bubnov-Galerkin approach, the space of test functions $\boldsymbol{v}$ is reduced to the same space V_h [55]. Thus, the same basis representation can be employed for the test functions $\boldsymbol{v}_h \in V_h$. To determine the unknown degrees of freedom, the above representation is inserted into the weak form (2) yielding a system of linear equations

$$\boldsymbol{K}\tilde{\boldsymbol{u}} = \boldsymbol{b}, \tag{4}$$

which has to be solved for the unknowns $\tilde{\boldsymbol{u}}$. Typically, $\boldsymbol{K}$ and $\boldsymbol{b}$ are denoted as stiffness matrix and external force vector.

2.2. p-version of the Finite Element Method

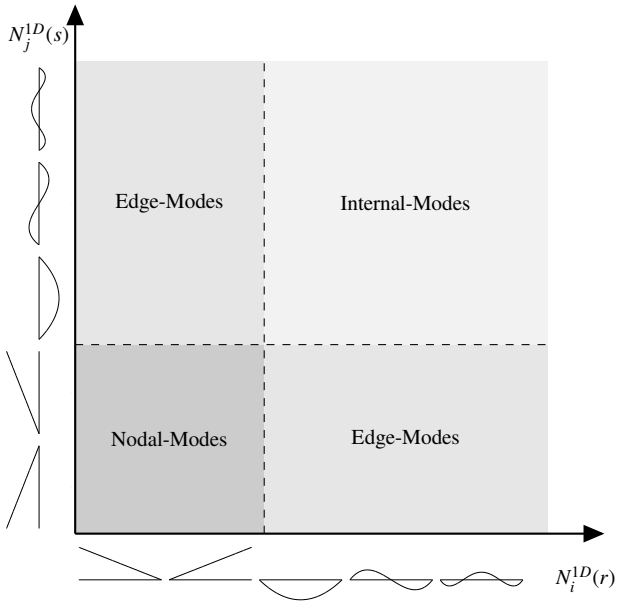
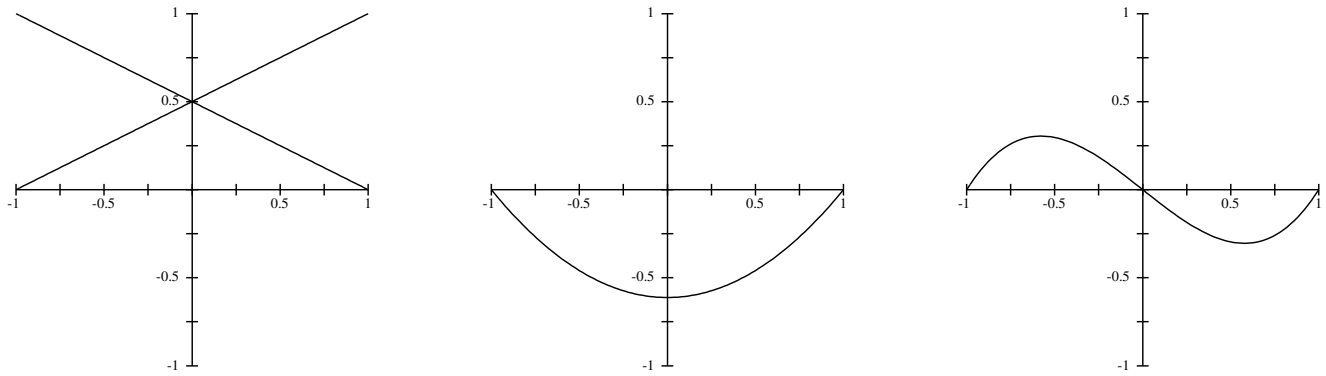


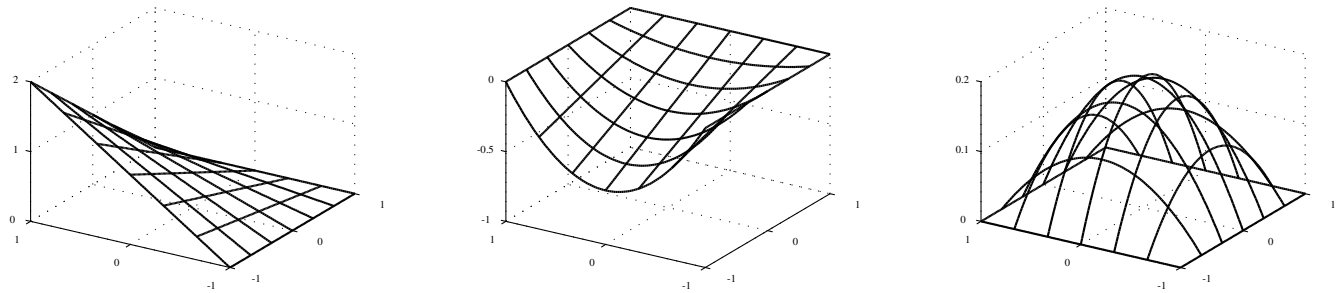
Figure 2: Tensor-product structure of legendre-based shape functions

The type of basis functions N has a major influence on the characteristics of the numerical method. For the Finite Cell Method, any high-order basis can be utilized. So far p-FEM shape functions [57, 59], NURBS, being used in isogeometric analysis [60], and Gauss-Lobatto-Legendre polynomials, used in the Spectral Cell Method [33], have been applied successfully. Currently, FCMLab only provides basis functions from the p-version.



(a) Linear modes (b) Quadratic mode (c) Cubic mode

Figure 1: 1D Legendre-based shape functions



(a) Nodal-mode (b) Edge-mode (c) Internal-mode

Figure 3: Two-dimensional mode types

Its one dimensional shape functions N_i^{1D} (see Figure 1) are defined as integrated Legendre polynomials [57]. They yield a hierarchical basis, which is superior to the classical Lagrange basis from a numerical point of view as the condition number of the corresponding stiffness matrix remains constant for an increasing polynomial degree (in case of Laplace problems). Although this property does not carry over to higher dimensions, a comparatively low condition number is maintained.

To obtain a two- or three-dimensional basis on a unit-square or -cube, the one-dimensional shape functions are combined in a tensor product

$$\begin{aligned} N_{i,j}^{2D}(r,s) &= N_i^{1D}(r)\, N_j^{1D}(s) \\ \text{and} \quad N_{i,j,k}^{3D}(r,s,t) &= N_{i,j}^{2D}(r,s)\, N_k^{1D}(t)\,. \end{aligned}$$

As depicted in Figure 2, the elements of the two-dimensional tensor-product space can be sorted into the following three groups:

Nodal-mode The combination of two one-dimensional nodal-modes gives rise to the well known bi-linear modes (see Figure 3a). Just as in 1D, these can be associated directly to one node giving the group its name.

Edge-mode The combination of a one-dimensional internal-mode and a one-dimensional nodal-mode results in a function that is non-zero at only one edge (see Figure 3b). Therefore, the mode can be directly associated to that edge and, thus, identified as a two-dimensional edge-mode.

Internal-mode The combination of two one-dimensional internal-modes yields a function that is zero on all four nodes and edges (see Figure 3c). The mode can, therefore, be directly associated to the face and thus, identified as a bubble- or internal-mode.

This mode concept naturally extends into three-dimensions, where also internal- or volume-modes have to be considered.

Different alternatives to the tensor-product space, such as a trunk-space or anisotropic polynomial-degree-templates, are discussed in [61, 57]. They are currently not implemented in `FCMLab` but might be added in the future.

2.3. The Finite Cell Method

As discussed in the introduction, mesh-generation is often time consuming and sometimes even a limiting factor when using FEM for industrial applications. This problem becomes even more severe for the *p*-version of the Finite Element

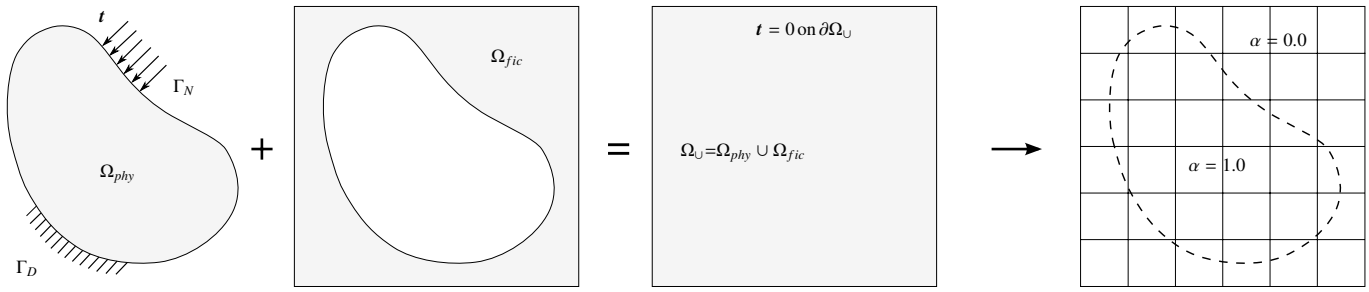


Figure 4: The Finite Cell idea [23]

Method: although the solution can be represented by much less and thus coarser elements compared to the *h*-version, they have to assume possibly complex shapes of the domain's surface, to deliver accurate results [57]. The Finite Cell Method (FCM) [20, 21], combining the benefits of higher-order Finite Elements with the ideas of fictitious domain methods, overcomes these mesh generation problems. The basics of the method are briefly reviewed in the first part of this section, whereas the second part addresses the imposition of boundary conditions for non boundary-conforming discretizations.

2.3.1. Basic concept

The essential idea of the Finite Cell Method is to simplify the mesh-generation process by separating the approximation of the analytical solution from the geometry representation of the physical domain. For this purpose, two independent discretizations are introduced: the *solution* and the *integration* mesh. The first mesh is used to approximate the analytical solution and, thus, spans the shape functions. However, it does not resolve the geometry of the domain. This is done by the second mesh, which is completely independent of the first one and does not introduce any additional degrees of freedom.

Solution mesh. The first step to separate the two meshes is to embed the actual physical domain Ω_{phy} in a fictitious domain Ω_{fic} such that their union $\Omega_\cup$ yields a simple shape (see Figure 4). To recover the original boundary value problem on the new domain $\Omega_\cup$, an indicator function α is defined as

$$\alpha(x) = \begin{cases} 1 & \forall\, x \in \Omega_{phy} \\ 0 & \forall\, x \notin \Omega_{phy}{}^{1}. \end{cases} \tag{5}$$

[1] Choosing α as zero outside of the physical domain negatively influences the condition number of the stiffness matrix. To control the conditioning, the value of α is typically set to a small value that allows to solve the system of equations without numerical problems. Experience shows that the direct solvers implemented in MATLAB can handle values of $\alpha \approx 10^{-10}$ without major difficulties.

The bi-linear form $a(\cdot,\cdot)$ can then be rewritten as

$$\begin{aligned} a(\boldsymbol{u},\boldsymbol{v}) &= \int_{\Omega_{phy}} \boldsymbol{\varepsilon}(\boldsymbol{v}) : \boldsymbol{C} : \boldsymbol{\varepsilon}(\boldsymbol{u}) \, \mathrm{d}\Omega \\ &= \int_{\Omega_{phy}} 1 \cdot \boldsymbol{\varepsilon}(\boldsymbol{v}) : \boldsymbol{C} : \boldsymbol{\varepsilon}(\boldsymbol{u}) \, \mathrm{d}\Omega \\ &+ \int_{\Omega_{fic}} 0 \cdot \boldsymbol{\varepsilon}(\boldsymbol{v}) : \boldsymbol{C} : \boldsymbol{\varepsilon}(\boldsymbol{u}) \, \mathrm{d}\Omega \\ &\approx \int_{\Omega_\cup} \alpha \cdot \boldsymbol{\varepsilon}(\boldsymbol{v}) : \boldsymbol{C} : \boldsymbol{\varepsilon}(\boldsymbol{u}) \, \mathrm{d}\Omega. \end{aligned}$$

The right hand side $f(\cdot)$ can be treated analogously. In this way, the meshing of the complex physical domain can be simplified drastically, as the simply-shaped domain $\Omega_\cup$ can be discretized instead using a Cartesian-shaped solution mesh, which renders the mesh-generation process trivial. To avoid a confusion of names, the resulting non boundary-conforming, high-order elements are denoted as Finite Cells giving the method its name.

Integration mesh. The second step is to recover the domain geometry using a separate integration mesh. In principle, this second mesh can be generated following various strategies. One possibility is to use conforming integration simplices by triangulating the physical domain. XFEM- and Level Set-based methods commonly apply this strategy [14, 15]. Typically, this approach is used with linear simplices. However, the method can also be extended to higher-orders as shown in [13, 62–64, 16, 65, 66].

An alternative strategy, which is followed within this work, is to use a space-tree as integration mesh (see e.g. [67]). Starting from the non-conforming solution mesh, cut cells are recursively refined towards the domain boundary yielding a quad- or octree structure (see Figure 5). This approach has the advantage that it requires only a simple inside-outside test. It can therefore also be applied in conjunction with simple geometry modelling techniques such as voxel-models, whereas the triangulation approach demands for a more advanced geometry representation.

2.3.2. Boundary conditions

A challenging task being a consequence of the non boundary-conforming discretization is the imposition of boundary conditions.

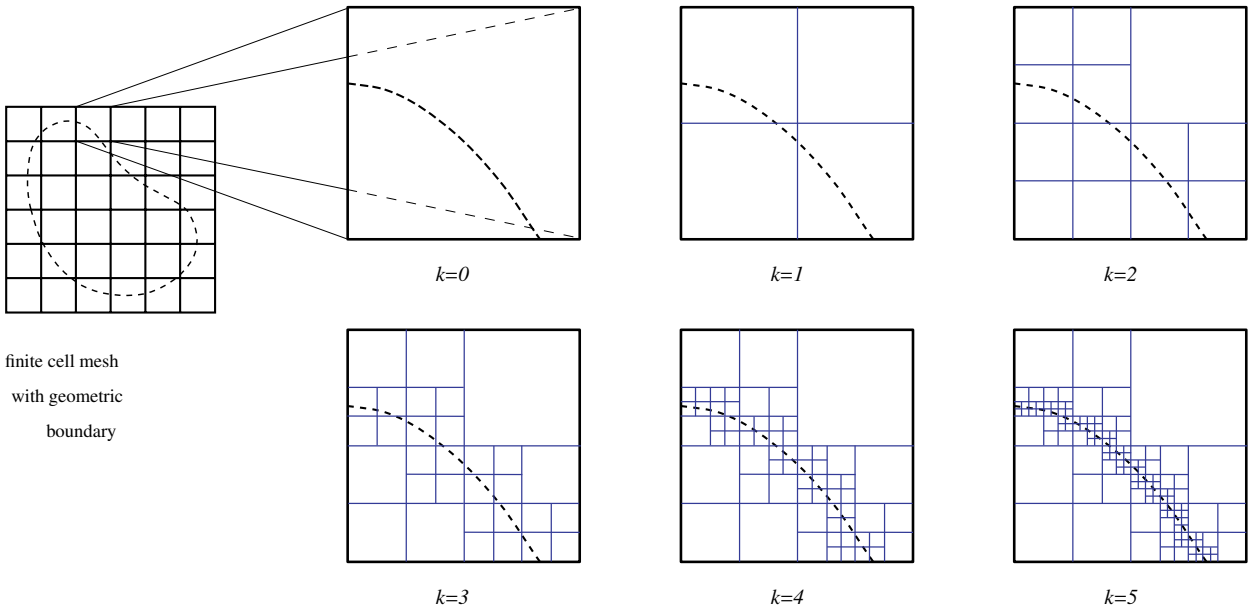


Figure 5: Integration via space-trees [23]

Like in the classical FEM, homogeneous, 'natural' Neumann boundary conditions need no special treatment in the FCM. Inhomogeneous Neumann boundary conditions demand to explicitly include the boundary integral term of the linear form (2c). As the boundary is *not* resolved by edges of the finite cells, a classical integration over the element edges is not possible. Instead, an explicit surface discretization is introduced on which the traction integral can be evaluated. Just as the previously described integration mesh, this *surface integration mesh* is independent of the actual solution mesh and does not introduce additional degrees of freedom. A convenient way to generate this mesh – supported by `FCMLab` – is to provide the surface of interest as an STL[2] file. The STL format is supported by most CAD systems and represents surfaces as a collection of linear triangles over which the integration can be performed.

In contrast to the Neumann case, the incorporation of Dirichlet boundary conditions is more challenging. Classically, they are imposed by explicitly choosing the shape functions from kinematically admissible spaces, such that the prescribed values are met on the boundary by definition [57]. This imposes the boundary conditions in the strong sense and can also be followed in the case of non-boundary conforming discretizations by adapting the admissible function spaces for the particular geometry under consideration. Popular methods following this idea are for example web- and i-spline based methods (see e.g. [68, 45]). An alternative approach followed in this work is to extend the weak formulation such that it directly incorporates the Dirichlet boundary conditions. Possible strategies for this purpose are for example the Lagrange Multiplier or Penalty Method [56, 69, 23, 70]. An alternative is to use an approach originally introduced by Nitsche in [71] for the Laplace problem, which was later extended to other applications [72–75]. The idea of this method is to extend the original weak form (2) by additional constraining expressions as follows:

$$\tilde{a}(\boldsymbol{u},\boldsymbol{v}) = a(\boldsymbol{u},\boldsymbol{v}) - \int_{\Gamma_D} (\boldsymbol{\sigma}(\boldsymbol{u})\cdot\boldsymbol{n})\cdot\boldsymbol{v}\,\mathrm{d}\Gamma - \int_{\Gamma_D} (\boldsymbol{\sigma}(\boldsymbol{v})\cdot\boldsymbol{n})\cdot\boldsymbol{u}\,\mathrm{d}\Gamma + \beta\int_{\Gamma_D}\boldsymbol{v}\cdot\boldsymbol{u}\,\mathrm{d}\Gamma$$

$$\text{and}\quad \tilde{f}(\boldsymbol{v}) = f(\boldsymbol{v}) - \int_{\Gamma_D} (\boldsymbol{\sigma}(\boldsymbol{v})\cdot\boldsymbol{n})\cdot\hat{\boldsymbol{u}}\,\mathrm{d}\Gamma + \beta\int_{\Gamma_D}\boldsymbol{v}\cdot\hat{\boldsymbol{u}}\,\mathrm{d}\Gamma.$$

The first additional integral of the bi-linear form follows naturally from the divergence theorem when the test functions are not restricted to be homogeneous on the boundary. It, therefore, insures the method's consistency. The second additional term of $\tilde{a}$ and the first additional term of $\tilde{f}$ are introduced to yield a symmetric stiffness matrix. The two remaining integrals are introduced for numerical stabilization, since coercivity might be lost when subtracting the consistency terms from the bilinear form. With this extension, the discrete problem can be reformulated incorporating the Dirichlet boundary conditions in the weak sense as follows:

$$\text{Find } \boldsymbol{u_h} \in V(\Omega) \text{ such that}$$
$$\tilde{a}(\boldsymbol{u_h},\boldsymbol{v_h}) = \tilde{f}(\boldsymbol{v_h}) \quad \forall\, \boldsymbol{v_h} \in V(\Omega).$$

In analogy to Neumann boundary conditions, the constraining expressions are evaluated using a separate surface integration mesh.

The major advantage of Nitsche-like methods is their inherent consistency, which ensures that the solution of the original strong problem (1) also solves the modified weak problem

[2] Surface Tesselation Language

[73, 76, 74]. However, the major challenge of this approach is to select the stability parameter β large enough to ensure the coercivity of the modified weak form. As shown in [74, 72, 73], the correct value of β is dependent on the mesh-size h, the polynomial degree p, and the material parameters. [73–75] suggest to solve an auxiliary eigenvalue-problem to estimate β. However, their investigations show that choosing a value higher than the threshold has no significant effect on the numerical result. Following the work of [23, 40, 42], who showed that also an empirical choice of the stability parameter yields good results, the eigenvalue-estimate is omitted in this work.

3. Code design

`FCMLab` is an object-oriented MATLAB toolbox, which is intended to provide an easy entry point into the research field of higher-order fictitious domain methods. This section presents a compact outline of the code design. To this end, the two most frequently applied design patterns are introduced in the first part of the section. In the second part, the most important classes of the toolkit and their mutual dependencies are explained. The description of the code expects the reader to be familiar with general MATLAB programming and knowledge of object oriented concepts in MATLAB, as described e.g. in [78, 79].

3.1. Applied design patterns

In the context of object-oriented software engineering, certain types of design problems keep on re-appearing, independent of the actual application under consideration. For this reason, *design patterns* have been developed as reusable template solutions for the most prominent design problems. A classical reference in this context is the textbook by Gamma, Helm, Johnson, and Vlissides [77].

3.1.1. Strategy pattern

The strategy pattern is a behavioral pattern aiming to decouple a family of algorithms from the actually calling code. For this purpose, different algorithms are implemented as separate classes. They all inherit from the same base class which defines the common interface. The client code then implements against this abstract interface and not against the actual implementations. Thus, the calling codes and the algorithms are decoupled, allowing to refactor and maintain the implementation of different algorithms without affecting the client code. Furthermore, the family of algorithms can be extended by simply adding new sub-classes. A numeric-oriented example is depicted in Figure 6a.

3.1.2. Factory method pattern

The factory method pattern is a creational pattern aiming to simplify the creation of complex products. For this purpose, the creational process is decoupled from the client code by introducing an abstract creator class. This specifies the interface functions that create the respective object. These factory-methods are then implemented in different sub-classes deciding in which way the final product is constructed. This approach allows the client code to implement against the abstract interface without knowing about the details of the different creator classes. In this way, the implementation of the creational process can be exchanged easily as the calling code remains unaffected. A numeric-oriented example is depicted in Figure 6b.

The factory idea can be extended to the abstract factory pattern, which allows to create complex product families easily. Within `FCMLab`, both versions of the pattern are applied and combined.

3.2. Code Verification

Besides the aspects of extensibility and maintainability, a third important request when developing a software framework is to ensure and maintain code correctness and consistency. For this purpose, `FCMLab` is developed in a test-driven manner (see e.g. [80, 81]). The framework is, thus, equipped with a test suite, which verifies the correctness of the individual modules following the idea of unit tests. Furthermore, the full simulation pipeline is tested using examples with analytical solutions which allow to verify the numerical results. To ensure code correctness during the on-going development process, a continuous integration system is employed, which automatically executes the entire test suite as soon as changes are committed to the version control system. To this end, we utilize the MATLAB xUnit Test Framework [82], the continuous integration server Jenkins (formally known as Hudson) [83], and Apache Subversion (SVN) [84].

3.3. Program structure

The aim of `FCMLab` is to provide a prototyping framework in which new algorithmic ideas can be tested easily. Therefore, the framework has to be structured such that different components can be exchanged easily without affecting other parts of the code. The design must thus aim for high cohesion of the individual modules – in the sense that all libraries, classes, and functions have *one* clearly defined responsibility–and low coupling between the modules. To address these issues, the program is decomposed hierarchically into the sub-systems depicted in Figure 7. These are then arranged as layers such that each module only depends on lower layers. This architecture

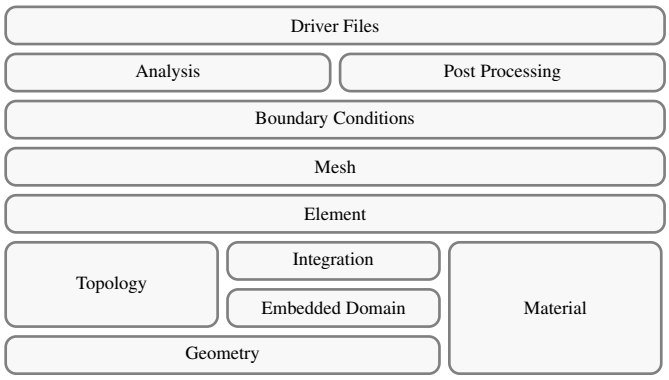
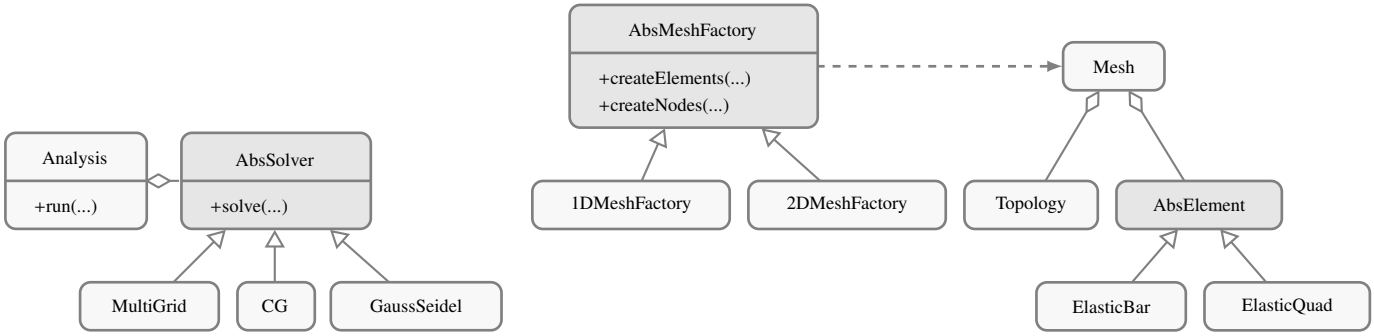


Figure 7: Layered decomposition of `FCMLab`



(a) Strategy Pattern

(b) Factory Method Pattern

Figure 6: Commonly used design patterns [77]

allows to regard each layer as a virtual machine with a clearly defined task, whose implementation can be exchanged without affecting up-stream elements. The details of the different packages are explained in the following sections.

3.3.1. Geometry

The geometry packages provide essential geometric objects such as vertices, lines, quadrilaterals, and hexahedra (see Figure 8). These are realized as separate classes and grouped according to their dimensionality as abstract curves, areas, and volumes. These abstractions all implement the class `AbsGeometry` that specifies the interface functions listed in Table 1.

Function Name	Parameter
`calcJacobian(...)`	local coordinates
`mapLocalToGlobal(...)`	local coordinates
`mapGlobalToLocal(...)`	global coordinates

Table 1: Interface-functions of geometry package

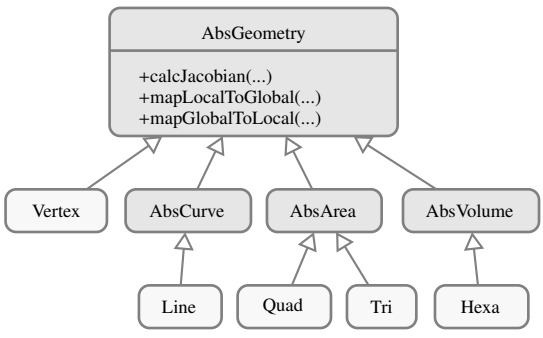


Figure 8: Class structure of geometry package

3.3.2. Topology

As discussed in Section 2.2, the Legendre-based shape functions can be associated to the different topological components. Therefore, the topology package, depicted in Figure 9, is responsible for handling the degrees of freedom. To this end, a `Dof` class is defined, which is then aggregated by the `AbsTopology` class. This interface is implemented by the classes `Node`, `Edge`, `Face`, and `Solid` that hold objects of their respective geometric counterparts. Furthermore, the classes store the associated polynomial degree of the shape functions.

3.3.3. Embedded domain

As discussed in Section 2.3, the essential idea of the Finite Cell Method is to decouple the solution mesh from the geometry and to recover the original domain on the integration level. This demands for a separate geometry representation, which is provided by the embedded domain package (see Figure 10). It contains the abstract `AbsEmbeddedDomain` class, which specifies the interface function `getDomainIndex(...)` determining the domain in which a point under consideration lies. This interface is implemented by different concrete domains, such as simple geometric objects that can be specified explicitly (e.g. sphere, rectangle, ellipse). Furthermore, `FCMLab` provides an implementation that allows to create embedded domains on the basis of CT-scan-data. This `VoxelDomain` class reads in the CT-data from a single ASCII file format, which linearizes the three-dimensional matrix. The origin and the bounding box are extracted automatically. Based on this information, the Finite Cell mesh can be created. The list of domain types can be extended easily to suit new needs.

In addition to the domain description, the FCM demands for an explicit representation of the boundary for application of boundary conditions. For this purpose, the package provides a `AbsBoundaryFactory` class, which specifies the `getBoundary(...)` function returning a surface description as a list of geometric simplices. Presently, several explicit geometries such as circles and rectangles are provided as well as an implementation to read STL files[3], which can be exported from a CAD model. Again, this list can be extended easily for new applications.

[3]For this purpose, the `stlread` function provided by Doron Harlev is used, which is available on the MATLAB Central File Exchange platform: `http://www.mathworks.com/matlabcentral/fileexchange/6678-stlread`

3.3.4. Integration

As discussed in Section 2.3, the domain integration for the FCM is carried out over a separate integration mesh. For this purpose, the integration package defines the non-abstract class `Integrator`. This class provides the service function `integrate(...)`, taking the integrand and the integration domain as arguments. To generate the integration mesh, the integrator aggregates the abstract classes `AbsPartitioner` and `AbsIntegrationScheme`. The `AbsPartitioner` serves as an interface for different partitioning strategies (see Figure 11). Currently, `FCMLab` provides one-, two-, and three-dimensional space trees as concrete partitioning strategies. These could be extended by different schemes, such as triangulations. The class `AbsIntegrationScheme` provides interfaces to get one-dimensional integration coordinates and weights. At present, the quadrature rule for Gauss-Legendre integration is available, which is used mostly in Finite Element Analysis [55].

3.3.5. Material

The material package contains different constitutive relations. To encapsulate these from the other parts of the code, the `AbsMaterial` class is defined, which specifies the interface functionality listed in Table 2.

In addition to providing the constitutive properties, the material package is also responsible for defining the Finite Cell indicator function α introduced in Section 2.3. For this purpose, the third method in the table is specified, which returns the α-value of the respective material. The idea is to associate one material for each domain: on the physical domains, the "real" materials are defined with $\alpha = 1.0$; on the fictitious domains, auxiliary "void" materials are defined with $\alpha \to 0$ [4]. This concept allows to handle all domains and materials in the same way during the integration process.

As depicted in Figure 12, `FCMLab` provides material laws for linear-elastic simulations. Again, this list can be extended by new materials easily as the other parts of the code implement against the interface defined by `AbsMaterial`.

[4] To avoid numerical difficulties due to badly conditioned matrices, an α-value of 10^{-10} is chosen for the numerical examples of Section 4

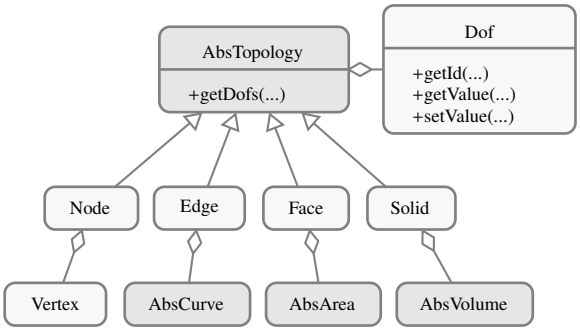
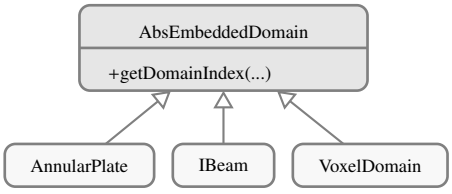
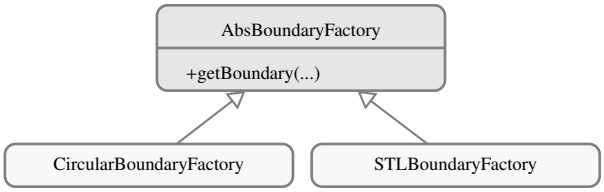


Figure 9: Class structure of topology package



(a) Selection of provided domains



(b) Selection of provided boundaries

Figure 10: Class structure of embedded domain package

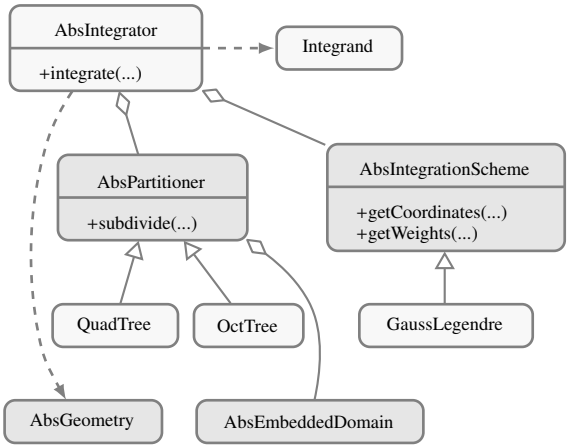


Figure 11: Class structure of integration package

Function Name	Parameter
`getMaterialMatrix(...)`	none
`getDensity(...)`	none
`getScalingFactor(...)`	none

Table 2: Interface-functions of material package

3.3.6. Element

The element package is the kernel of the toolbox. It is responsible for computing the stiffness matrix and the right-hand-side vector of an element. For this purpose, the abstract `AbsElement` class is defined aggregating its topological components, a material, and a domain and offers the interface functions listed in Table 3. To implement these services, the shape functions and their derivatives need to be evaluated and arranged in matrices. As this process depends on the dimensionality of the problem, the strategy pattern is applied by encapsulating the shape-function evaluation into respective sub-classes (see Figure 13). In this way, the higher-level parts of the code

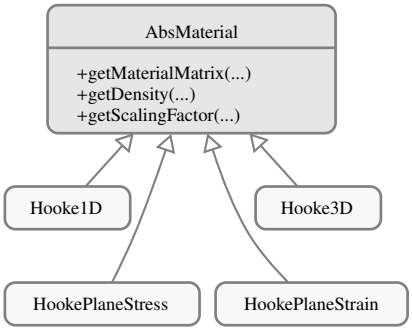


Figure 12: Class structure of material package

can use the abstract element without having to know about the details of the setup and the dimensionality. As the creation of an element is rather complex, different factories are provided to generate the most commonly used types.

Function Name	Parameter
`calcStiffnessMatrix(...)`	none
`calcMassMatrix(...)`	none
`calcBodyLoadVector(...)`	load function
`getLocationMatrix(...)`	none
`getDofVector(...)`	load case id

Table 3: Interface-functions of element package

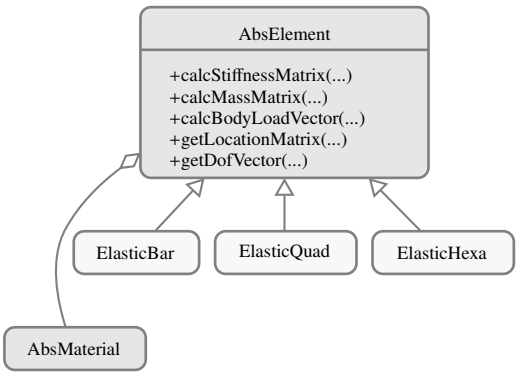


Figure 13: Class structure of element package

3.3.7. Mesh

The mesh package provides components to create and operate on one-, two-, and three-dimensional Cartesian meshes. For this purpose, a non-abstract `Mesh` class is defined aggregating the respective topological components as well as all elements (see Figure 14). As services, it offers the functions listed in Table 4. Furthermore, the mesh package offers different schemes to number the degrees of freedom. To this end, the `AbsNumberingScheme` is defined, which assigns the degrees of freedoms to the topological components. The interface is implemented by concrete strategies sorting the unknowns either according to their associated polynomial degree or to their topological component.

Following the abstract factory pattern, the mesh creation process is encapsulated into the abstract `AbsMeshFactory` class that handles the setup of the topology, the elements, and the numbering scheme.

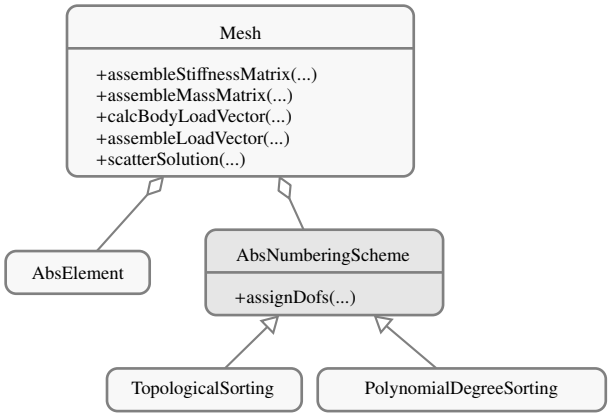


Figure 14: Class structure of mesh package

Function Name	Parameter
`assembleStiffnessMatrix(...)`	none
`assembleMassMatrix(...)`	none
`assembleLoadVector(...)`	load case id
`scatterSolution(...)`	solution

Table 4: Interface-functions of mesh package

3.3.8. Boundary condition

The boundary condition package offers the possibility to constrain the numerical system using Neumann and Dirichlet boundary conditions. As described in Section 2.3, the latter can be applied in the strong and in the weak sense. To break the complexity of the package, it is split into sub-systems, which are explained in the following paragraphs[5].

Dirichlet boundary conditions. To apply Dirichlet boundary conditions, the abstract `AbsDirichletBoundaryCondition` class is introduced. It provides the interface function `modifyLinearSystem(...)`, which takes the mesh and the system of linear equations as arguments and applies the boundary conditions. From this super class, the `StrongDirichletBoundaryCondition` and `WeakDirichletBoundaryCondition` are derived.

Strong Dirichlet boundary conditions directly operate on degrees of freedom. Since the latter are associated to

[5]Within the figures of the different packages, the terms DBC and NBC are used as abbreviations for Dirichlet and Neumann boundary conditions. Within `FCMLab`, the full class names are spelled out.

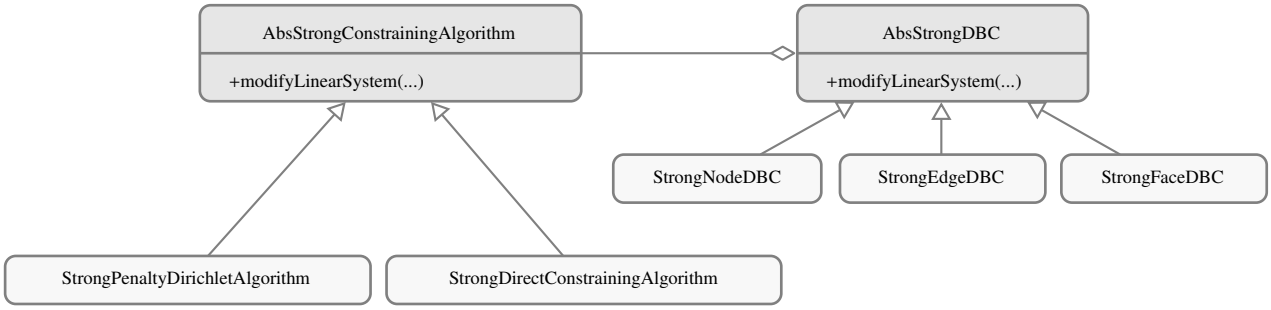


Figure 15: Class structure of strong boundary condition Package[5]

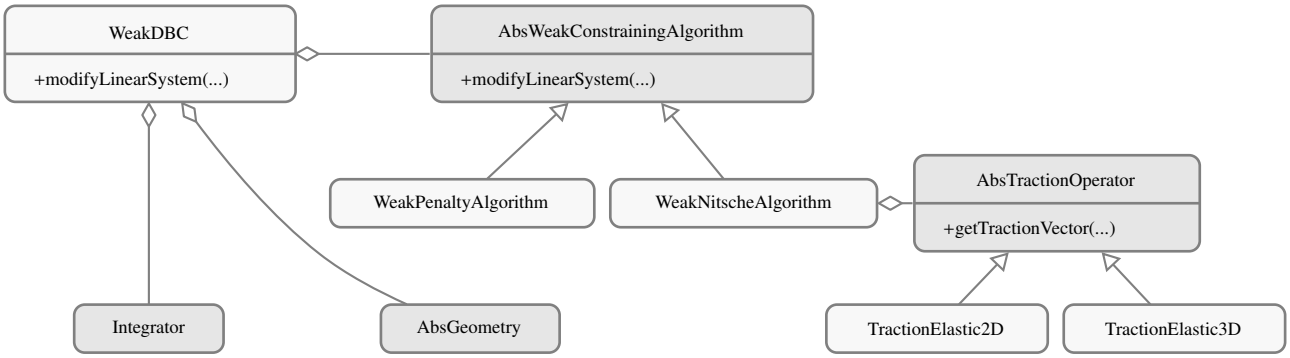


Figure 16: Class structure of weak boundary condition Package[5]

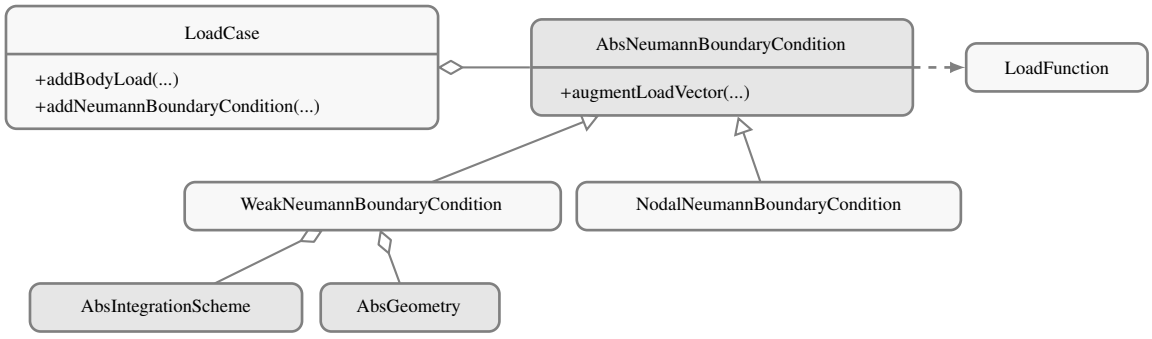


Figure 17: Class structure of neumann boundary condition Package[5]

the topological components (see Section 3.3.2), three different classes are derived from `AbsStrongDirichletBoundaryCondition`. They handle the corresponding tasks for nodes, edges, and faces, respectively (see Figure 15). In principle, strong Dirichlet boundary conditions can be applied in various ways. To allow for this flexibility, the strategy pattern is applied by introducing the interface class `AbsStrongConstrainingAlgorithm`. Presently, two implementations of this interface are provided: The first is the `StrongPenaltyDirichletAlgorithm`, which constrains the respective degrees of freedom by adding a specified penalty value on the main diagonal of the system matrix. The second strategy condenses the influence of the constrained degrees of freedom to the right hand side and thus results in a better conditioning of the stiffness matrix (see e.g. [85]).

As described in Section 2.3, weak Dirichlet boundary conditions can be e.g. applied following the penalty or a Nitsche-like method. Both approaches introduce additional boundary integrals constraining the solution appropriately. To evaluate these integrals, the `WeakDirichletBoundaryCondition` class aggregates a list of geometric objects as boundary description and an integrator object (see Figure 16). The actual constraining algorithm is encapsulated by introducing the `AbsWeakConstrainingAlgorithm` interface class, which is then implemented by the `WeakNitscheAlgorithm` and `WeakPenaltyAlgorithm`.

Neumann boundary conditions. Also in the case of Neumann boundary conditions, two different cases have to be considered.

In the general case, the traction integral in (2c) has to be evaluated over the Neumann boundary. To this end, the `WeakNeumannBoundaryCondition` class is introduced, which, in analogy to the weak Dirichlet case, aggregates a boundary description and an integrator object. The actual integrand is specified as a handle to a load function (see Figure 17).

The second case is the application of a load vector, which lumps the applied tractions in the nodal degrees of freedom [58]. For this purpose, the `NodalNeumannBoundaryCondition` is defined.

To compute the mechanical reactions of the system under consideration for different load settings, a `LoadCase` class is introduced to which different body and surface loads can be associated.

3.3.9. Analysis

The analysis package is at the top level of the `FCMLab` toolbox. Here the numerical problem is set up and run. For this purpose, the abstract class `AbsAnalysis` is introduced, which aggregates the mesh and boundary conditions (see Figure 18) and specifies the interface functions listed in Table 5. These are then implemented by concrete analyses. In its current state `FCMLab` provides a quasi-static and an eigenmode analysis (see Figure 18).

3.3.10. Post-processing

The final module to be discussed here is the post-processing package. `FCMLab` offers two different ways to post-process numerical results: visual post-processing depicting the results as line or surface plots and integrational post-processing allowing the calculation of integral values, such as strain energy and mass. Again following the strategy pattern, both post-processor types operate on point-processors via an interface class `AbsPointProcessor`. These convert the numerical solution into the physical quantities of interest, such as strains, stresses, and energies (see Figure 19).

In the context of the Finite Cell Method, both post-processing types have to be enhanced to operate on the non boundary-conforming discretization. In case of integrational post-processing, the quadrature package outlined in Section 3.3.4 is used for this purpose. For visual post-processing, a separate post-processing mesh is introduced to visualize the results in two or three dimensions. Different factories provided by `FCMLab` allow for an easy creation of these meshes. The user can also choose between different types of visualization, such as line- or path- and surface-plots. For the latter plot type, also the finite cells, the integration cells, and the boundary condition mesh can be visualized.

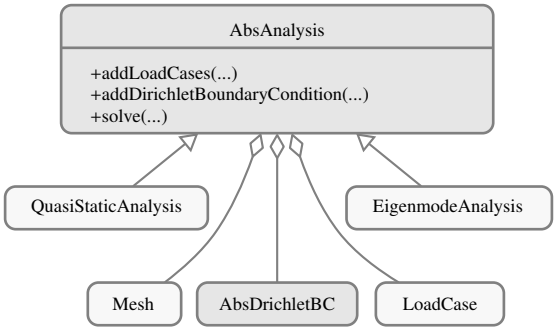


Figure 18: Class structure of analysis package[5]

Function Name	Parameter
`addLoadCases(...)`	list of load cases
`addDBC(...)`	abstract Dirichlet BC
`solve(...)`	none

Table 5: Interface-functions of analysis package

3.4. Extensions to alternative differential operators and non-linear problems

Currently the toolbox only supports simulations of linear elastic problems. As shown by e.g. Schillinger *et al.*, however, the Finite Cell Method also yields excellent results in the context of geometrically non-linear elasticity (see e.g [27]). The modular setup of `FCMLab`, described in the previous sections, allows to easily extend the toolbox for applications of this category. As a guideline for users of the toolbox, the necessary steps shall be briefly outlined in the following.

First, new elements have to be defined, which allow for large deformations. To this end, new element classes have to be created in the element package, which implement the `AbsElement`

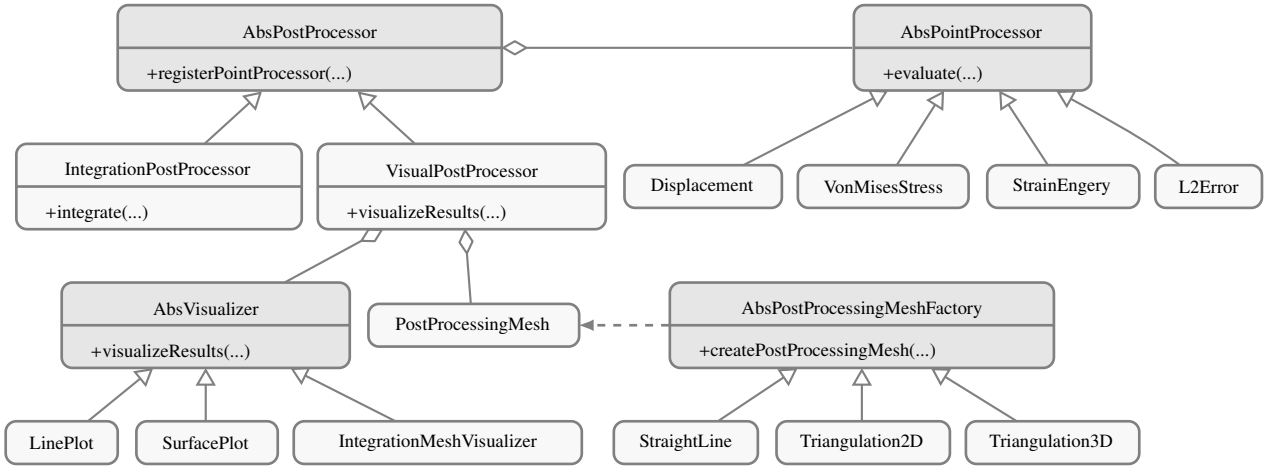


Figure 19: Class Structure of Post-Processing Package. Note that only a selection of the available point-processors, mesh factories, and visualizers is depicted.

interface.

In a second step, the respective material models (e.g. Moony-Rivlin or Neo-Hooke [86]) have to be implemented in the framework. This can be done easily by creating new sub-classes in the material package that implement the `AbsMaterial` interface.

The final step is the implementation of a non-linear solver, which solves the non-linear system of equations iteratively e.g. using the Newton's method or a fix-point approach (see e.g. [86]). This has to be carried out in the analysis package by creating new classes, which implement the `AbsAnalysis` interface.

Similarly, the framework could be extended into the domain of fluid dynamics, where void and structure essentially interchange their role, see, e.g. [87]. For a convection diffusion problem the penalization factor α is applied to the diffusion coefficient in the 'obstacle' regime of the flow.

4. Examples

Having outlined the theoretical background and the basic program structure in the previous sections, this part of the work aims at demonstrating the usage of `FCMLab`. For this purpose, examples from previous FCM-related publications are reused for highlighting the different features of the toolbox. The section starts with a two-dimensional benchmark example with an explicit geometry description. In the second example, the mechanical behaviour of a three-dimensional femur (thigh bone) is investigated using a voxel model. The section concludes with a third example demonstrating the possibility of eigenmode analyses using the Finite Cell Method.

4.1. Analytical benchmark

This first example outlines the structure of a driver file to setup an analysis within `FCMLab`. For this purpose, the displacement of the annular plate depicted in Figure 20a shall be computed, following [23, 42, 40]. Assuming a plane stress configuration and the denoted material properties, the displacement in polar coordinates caused by the stated radial loading can be given analytically as:

$$u_r = -\frac{r}{2}\frac{\ln r}{\ln 2}$$
$$u_\theta = 0.$$

The corresponding polar stresses read

$$\sigma_r = \varepsilon_r = \frac{\partial u_r}{\partial r} = -\frac{1}{2}\frac{\ln(r)+1}{\ln(2)}$$
$$\sigma_\theta = \varepsilon_\theta = \frac{1}{r}\frac{\partial u_\theta}{\partial \theta} + \frac{u_r}{r} = -\frac{1}{2}\frac{\ln(r)}{\ln(2)}$$
$$\sigma_{r\theta} = \varepsilon_{r\theta} = \frac{1}{2}\left(\frac{1}{r}\frac{\partial u_r}{\partial \theta} + \frac{\partial u_\theta}{\partial r} - \frac{u_\theta}{r}\right) = 0.$$

The analysis can be separated into the parameter configuration and the analysis setup. These two sections are explained individually in the following paragraphs.

Parameter configuration. The driver file starts with the description of the discretization and material parameters:

```
MeshOrigin = [-1.1 -1.1 0.0];
MeshLengthX = 2.2;
MeshLengthY = 2.2;
NumberOfXDivisions = 4;
NumberOfYDivisions = 4;
PolynomialDegree = 4;
NumberOfGaussPoints = PolynomialDegree+1;

YoungsModulus = 1.0;
PoissonsRatio = 0.0;
Density = 1.0;
```

In a second step, the numerical parameters of the Finite Cell Method are specified as follows:

```
SpaceTreeDepth = 3;
Alpha = 1e-10;
Beta = 1e3;
```

Here the value of the indicator function α outside of the physical domain is chosen to be a small positive number in order to avoid ill-conditioning of the system matrix.

The third step is to define the inner and outer radius of the annular plate and the discretization of the boundary mesh:

```
Center = [ 0.0 0.0 0.0 ];
OuterRadius = 1.0;
InnerRadius = 0.25;
NumberOfSegments = 500;
```

Finally, the body forces and the boundary values need to be specified as function handles:

```
bPolar = @(r,theta) [ 1/(r*log(2)) ; 0];

uxPolar = @(r,theta) 0.25*cos(theta);
uyPolar = @(r,theta) 0.25*sin(theta);
```

Analysis setup. After having specified the different numerical parameters, the analysis can be setup. For this purpose, two different materials are defined for the physical and the void domain:

```
Materials(1) = HookePlaneStress( YoungsModulus, ...
    PoissonsRatio, Density, Alpha );
Materials(2) = HookePlaneStress( YoungsModulus, ...
     PoissonsRatio, Density, 1 );
```

The next step is to define the physical domain and its boundary. As the geometry can be described explicitly, the `AnnularPlate` implementation of the `AbsDomain` interface introduced in Section 3.3.3 is used:

```
AnnularPlate = AnnularPlate( ...
    Center, OuterRadius, InnerRadius );

CounterClockWise = true;
OuterBoundaryFactory = ...
    CircularBoundaryFactory( Center, ...
      OuterRadius, NumberOfSegments, ...
      CounterClockWise );

CounterClockWise = false;
InnerBoundaryFactory = ...
    CircularBoundaryFactory( Center, ...
      InnerRadius, NumberOfSegments, ...
      CounterClockWise );
```

In a third step, the discretization is created using the element and mesh factories described in Section 3.3.6 and 3.3.7:

```
DofDimension = 2;

ElementFactory = ...
    ElementFactoryElasticQuadFCM( ...
       Materials, AnnularPlate, ...
       NumberOfGaussPoints, SpaceTreeDepth );

MeshFactory = MeshFactory2DUniform( ...
    NumberOfXDivisions, NumberOfYDivisions,...
    PolynomialDegree, PolynomialDegreeSorting, ...
    DofDimension, MeshOrigin, MeshLengthX, ...
    MeshLengthY, ElementFactory );
```

With these entities, the quasi-static analysis can now be created:

```
Analysis = QuasiStaticAnalysis( MeshFactory );
```

To apply the body forces, a new load case is set up and registered using the function handles defined above:

```
BodyForce = @(x,y,z) ...
    polarToCartesian2DVector( x, y, bPolar );

LoadCase = LoadCase();
LoadCase.addBodyLoad( BodyForce );

Analysis.addLoadCases( LoadCase );
```

To specify the weak Dirichlet boundary conditions, a Nitsche constraining algorithm and an integration scheme are specified. These are then used to create three boundary conditions for the inner and outer boundary in x- and y-direction, which are then registered to the analysis:

```
% Select constraining strategy
ConstrainingAlgorithm = ...
    WeakNitscheDirichlet2DAlgorithm( Beta );
IntegrationScheme = ...
    GaussLegendre( NumberOfGaussPoints );

% Create outer boundary condition
Boundary = ...
    OuterBoundaryFactory.getBoundary();
zeroBoundary = @(x,y,z) 0;
ConstrainedDirections = [1 1];
OuterCondition = WeakDirichletBoundaryCondition( ...
    zeroBoundary, ConstrainedDirections, ...
    IntegrationScheme, Boundary, ...
    ConstrainingAlgorithm );

% Create inner boundary condition in X
Boundary = ...
    InnerBoundaryFactory.getBoundary();
Ux = @(x,y,z) polarToCartesian(x,y,uxPolar);
ConstrainedDirections = [1 0];
InnerConditionX = WeakDirichletBoundaryCondition( ...
    Ux, ConstrainedDirections, IntegrationScheme,...
    Boundary, ConstrainingAlgorithm );

% Create inner boundary condition in Y
Uy = @(x,y,z) polarToCartesian(x,y,uyPolar);
ConstrainedDirections = [0 1];
InnerConditionY = WeakDirichletBoundaryCondition( ...
    Uy, ConstrainedDirections, IntegrationScheme,...
    Boundary, ConstrainingAlgorithm );

% Register boundary conditions at analysis
Analysis.addDirichletBoundaryCondition( ...
    OuterCondition );
Analysis.addDirichletBoundaryCondition( ...
    InnerConditionX );
Analysis.addDirichletBoundaryCondition( ...
    InnerConditionY );
```

With this setup, the analysis can be executed as follows:

```
Analysis.solve();
```

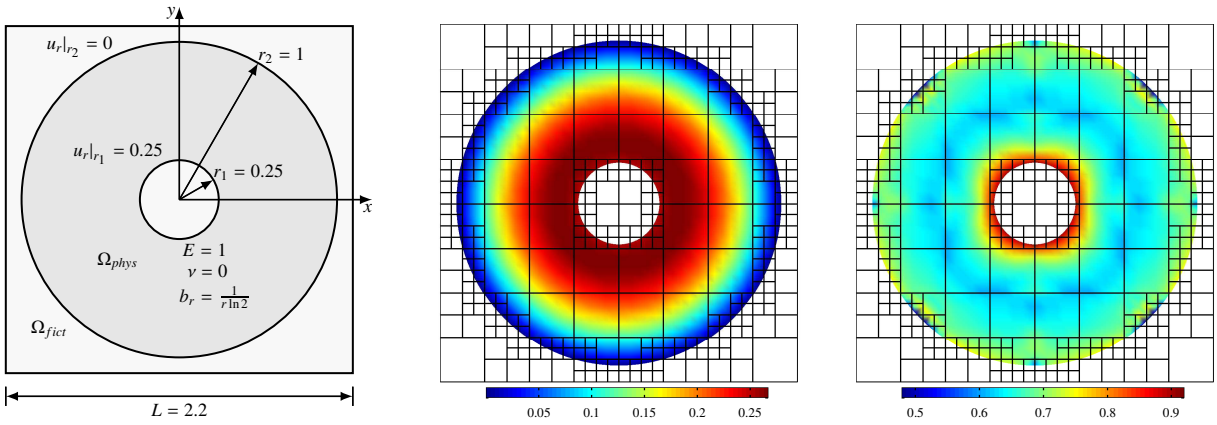
As discussed in Section 3.3.10, `FCMLab` offers the possibilities of visual and integral post-processing, which both work on point-processors. These can, for example, be defined as follows:

```
indexOfPhysicalDomain = 2;
loadCaseToVisualize = 1;

displacementNorm = DisplacementNorm( ...
    loadCaseToVisualize );
vonMises = VonMisesStress( loadCaseToVisualize, ...
    indexOfPhysicalDomain );
strainEnergy = StrainEnergy( loadCaseToVisualize, ...
    indexOfPhysicalDomain );
```

To visualize these results, a visual post-processor is created to which different point-processors are registered:

```
FeMesh = Analysis.getMesh();
gridSize = 0.1;
postProcessingFactory = ...
    VisualPostProcessingFactory2DFCM( ...
        FeMesh, AnnularPlate, ...
        indexOfPhysicalDomain, gridSize, ...
        {OuterBoundaryFactory, ...
        InnerBoundaryFactory} );
postProcessor = ...
    postProcessingFactory.creatVisualPostProcessor();
```



(a) Benchmark setup (b) Displacement norm (c) von Mises stress

Figure 20: Visualization of annular plate results

```
postProcessor.registerPointProcessor( ...
    displacementNorm );
postProcessor.registerPointProcessor( vonMises );

postProcessor.visualizeResults( FeMesh );
```

The visualization of the displacement and stress obtained from the registered post-processors is depicted in Figure 20.

In analogy, the results can also be integrated over the domain:

```
postProcessor = IntegrationPostProcessos( );

postProcessor.registerPointProcessor( strainEnergy );
postProcessor.registerPointProcessor( l2Error );

integrals = postProcessor.integrate( FeMesh );
```

4.2. Femur

The following example aims at demonstrating the ability of `FCMLab` to work on geometries defined by voxel data. To this end, the mechanical reaction of a human femur (thigh bone) subjected to a hip-contact force is computed. Typically, inhomogeneous material models are applied in this context to capture the physical behaviour correctly (see e.g. [30, 88, 89]). However, as the focus of this paper lies on demonstrating the usage of different geometry models, a homogeneous, linear-elastic material description is used.

As the analysis is again quasi-static, the general setup of the driver file remains unchanged. However, an important difference is that the bone's geometry is represented by a voxel-model obtained from a CT[6]-scan of the patient. The data file is loaded using the `VoxelDomain` implementation of the `AbsDomain` interface described in Section 3.3.3:

```
VoxelDataFile = 'BoneFF1_CTdata_cut.txt';
Bone = VoxelDomain( VoxelDataFile );
```

The second important difference is that the hip-contact force has to be applied on the non boundary-conforming discretization. To this end, a separate surface discretization of the hip-cap is loaded from an STL-file and then used to create a Neumann boundary condition:

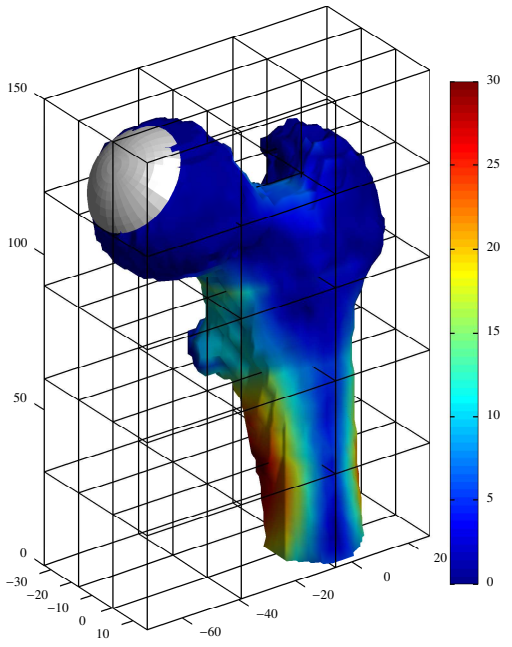


Figure 21: von Mises stress distribution in MPa

[6]Computed Tomography

```
BinarySTLFile = 'HipCap.stl';
LoadSurfaceFactory = ...
    STLBoundaryFactory( BinarySTLFile );
LoadBoundary = ...
    LoadSurfaceFactory.getBoundary();

BoundaryIntegrator = ...
    GaussLegendre( NumberOfGaussPoints );

pressure = -1;
Traction = @(x,y,z) [0;0; pressure];
Load = WeakNeumannBoundaryCondition( ...
    Traction, BoundaryIntegrator, ...
    LoadBoundary );

LoadCase = LoadCase();
LoadCase.addNeumannBoundaryCondition( Load );
Analysis.addLoadCases( LoadCase );
```

The remaining parts of the driver file follow in analogy to the first example and are, thus, not recapitulated here.

Of course, also three-dimensional results can be post-processed using FCMLab. Figure 21 depicts exemplarily the von Mises' stress distribution in the bone caused by the loading. The figure also shows the traction surface used to apply the hip-contact forces on the bones' head.

4.3. Eigenmode analysis

This final example aims at demonstrating the use of FCMLab in the context of an eigenmode analysis. For this purpose, eigenmodes of the I-section beam depicted in Figure 22 are computed.

In line with the annular plate example, the physical domain is represented by an explicit geometry description. As shown in Figure 22a, the beam is embedded in a slightly larger fictitious domain being discretized by $5 \times 5 \times 5$ finite cells of polynomial order 3. The driver file for this setup follows in analogy to the previous two examples. The important difference is that in this application no quasi-static analysis is performed. Instead, the EigenmodeAnalysis implementation of the AbsAnalysis interface is applied in the following way:

```
NumberOfModesToCompute = 10;
Analysis = EigenmodeAnalysis( ...
    MeshFactory, NumberOfModesToCompute );
```

Results are depicted in Figure 23, and the corresponding frequencies are listed in Table 6. The comparison to references obtained from an ANSYS computation using 3200 quadratic elements of type Solid186 (see [90]) demonstrates that the Finite Cell Method does not only allow to significantly simplify the mesh generation process but also yields highly accurate results.

5. Conclusion

This work introduced FCMLab as an object-oriented MATLAB toolbox for the Finite Cell Method allowing rapid-prototyping of new algorithmic methods in the context of high-order fictitious domain methods.

[7] Degrees of Freedom

	FCMLab	ANSYS	Deviation
1^{st} Frequency	7.308	7.307	1.37e-02 %
2^{nd} Frequency	9.637	9.635	2.08e-02 %
3^{rd} Frequency	33.452	33.325	3.81e-01 %
4^{th} Frequency	44.317	44.305	2.71e-02 %
5^{th} Frequency	54.783	54.706	1.41e-01 %
6^{th} Frequency	104.241	103.816	4.09e-01 %
7^{th} Frequency	118.374	118.242	1.12e-01 %
8^{th} Frequency	126.189	126.189	1.29e-05 %
9^{th} Frequency	136.045	135.601	3.27e-01 %
10^{th} Frequency	185.731	184.869	4.66e-01 %
Number of Dofs[7]	12288	54720	

Table 6: Eigenfrequencies of I-section beam [Hz]

To offer researchers an entry point for code extensions, the paper firstly explained the different modules of the framework in detail by pointing out the class structure and its internal dependencies. Secondly, the usage of the toolbox was explained by means of different examples demonstrating how explicit and voxel-based geometry models can be used for quasi-static and eigenmode analyses within FCMLab.

At the current stage, the toolbox's scope allows to apply the Finite Cell Method in the context of one-, two-, and three-dimensional linear-elasticity. However, the framework is explicitly designed to allow for easy extensibility. It is, therefore, conceivable to broaden the applicability to non-linear problems or to implement alternative integration schemes. Also, the development of new approaches for imposing Dirichlet boundary conditions on the non-conforming discretization would be very interesting. We, therefore, explicitly invite interested researchers to join our development group at http://fcmlab.cie.bgu.tum.de/. Here, you can download the full source code of FCMLab and find a detailed *Getting Started* guide explaining the installation of the framework and how to setup and run your own examples.

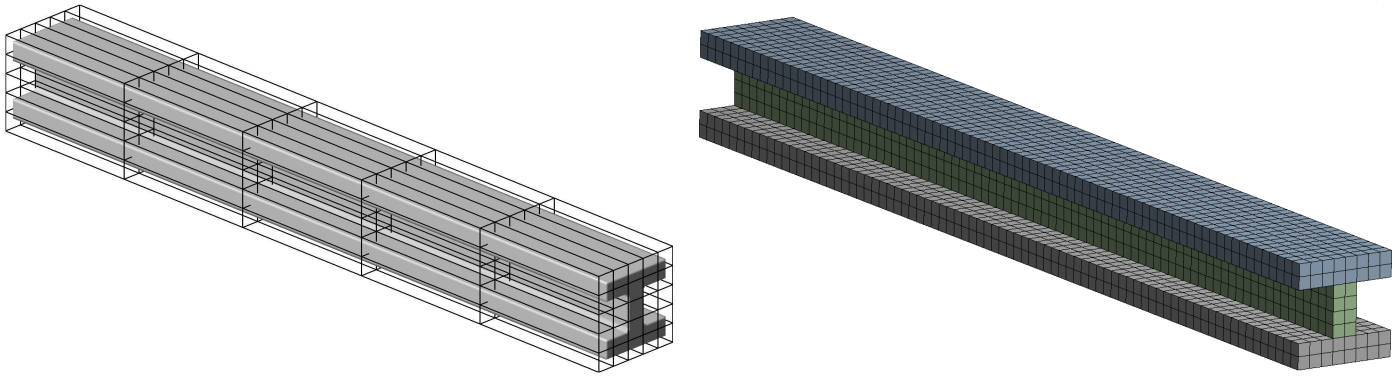
Acknowledgements

The first two and the last author gratefully acknowledge the financial support of the German Research Foundation (DFG) under grant RA 624/19-1 and RA 624/15-2. D. Schillinger acknowledges support from the Institute for Computational Engineering and Sciences (ICES) at the University of Texas at Austin and the German Research Foundation (DFG) under grant SCHI 1249/1-2.

References

[1] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, P. Krysl, Meshless methods: An overview and recent developments, Computer Methods in Applied Mechanics and Engineering 139 (1-4) (1996) 3–47, ISSN 00457825, doi:10.1016/S0045-7825(96)01078-X.

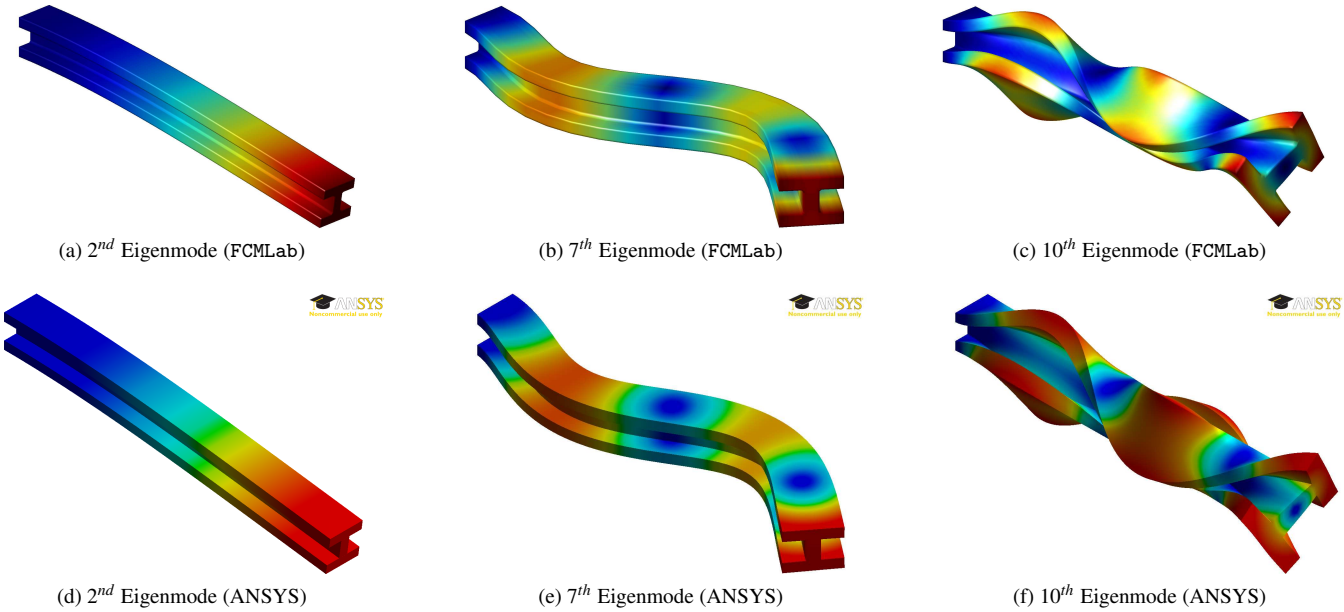
[2] T. Belytschko, Y. Y. Lu, L. Gu, Element-free Galerkin methods, International Journal for Numerical Methods in Engineering 37 (2) (1994) 229–256, ISSN 0029-5981, doi:10.1002/nme.1620370205.



(a) FCM Discretization within FCMLab (Polynomial Degree $p = 3$)

(b) FEM Discretization within ANSYS (Polynomial Degree $p = 2$)

Figure 22: Discretization of I-section beam



(a) 2^{nd} Eigenmode (FCMLab)

(b) 7^{th} Eigenmode (FCMLab)

(c) 10^{th} Eigenmode (FCMLab)

(d) 2^{nd} Eigenmode (ANSYS)

(e) 7^{th} Eigenmode (ANSYS)

(f) 10^{th} Eigenmode (ANSYS)

Figure 23: Comparison of the beams' eigenmodes

[3] T. Strouboulis, I. Babuska, K. Copps, The design and analysis of the Generalized Finite Element Method, Computer Methods in Applied Mechanics and Engineering 181 (1-3) (2000) 43–69, ISSN 00457825, doi:10.1016/S0045-7825(99)00072-9.

[4] T. Strouboulis, K. Copps, I. Babuska, The generalized finite element method, Computer methods in applied mechanics and engineering 190 (32) (2001) 4081–4193.

[5] T. Fries, T. Belytschko, The extended/generalized finite element method: An overview of the method and its applications, International Journal for Numerical Methods in Engineering 84 (3) (2010) 253–304, ISSN 00295981, doi:10.1002/nme.2914.

[6] T. Rüberg, F. Cirak, Subdivision-stabilised immersed b-spline finite elements for moving boundary flows, Computer Methods in Applied Mechanics and Engineering 209-212 (2012) 266–283, ISSN 00457825, doi:10.1016/j.cma.2011.10.007.

[7] R. Mittal, G. Iaccarino, Immersed boundary methods, Annu. Rev. Fluid Mech. 37 (2005) 239–261.

[8] C. S. Peskin, The immersed boundary method, Acta Numerica 11 (January 2002), ISSN 0962-4929, doi:10.1017/S0962492902000077.

[9] R. Löhner, J. R. Cebral, F. F. Camelli, J. D. Baum, E. L. Mestreau, O. A. Soto, Adaptive Embedded/Immersed Unstructured Grid Techniques, Archives of Computational Methods in Engineering 14 (3) (2007) 279–301, ISSN 1134-3060, doi:10.1007/s11831-007-9008-4.

[10] I. Ramière, P. Angot, M. Belliard, A fictitious domain approach with spread interface for elliptic problems with general boundary conditions, Computer Methods in Applied Mechanics and Engineering 196 (4-6) (2007) 766–781, ISSN 00457825, doi:10.1016/j.cma.2006.05.012.

[11] E. Nadal, J. J. Ródenas, J. Albelda, M. Tur, J. E. Tarancón, F. J. Fuenmayor, Efficient Finite Element Methodology Based on Cartesian Grids: Application to Structural Shape Optimization, Abstract and Applied Analysis 2013 (2013) 1–19, ISSN 1085-3375, 1687-0409, doi:10.1155/2013/953786.

[12] O. Marco Alacid, A shape sensitivity analysis module with geometric representation by NURBs for a 2D finite element program based on Cartesian meshes independent of the geometry, Master's thesis, Universitat Politècnica de València, Valencia, Spain, 2013.

[13] K. W. Cheng, T. Fries, Higher-order XFEM for curved strong and weak discontinuities, International Journal for Numerical Methods in Engineering 82 (5) (2009) 564–590, ISSN 00295981, doi:10.1002/nme.2768.

[14] K. Dréau, N. Chevaugeon, N. Moës, Studied X-FEM enrichment to handle material interfaces with higher order finite element, Computer Methods in Applied Mechanics and Engineering 199 (29-32) (2010) 1922–1936, ISSN 00457825, doi:10.1016/j.cma.2010.01.021.

[15] G. Legrain, N. Chevaugeon, K. Dréau, High order X-FEM and levelsets for complex microstructures: Uncoupling geometry and approximation, Computer Methods in Applied Mechanics and Engineering 241-244 (2012) 172–189, ISSN 00457825, doi:10.1016/j.cma.2012.06.001.

[16] G. Legrain, A NURBS enhanced extended finite element approach for unfitted CAD analysis, Computational Mechanics 52 (4) (2013) 913–929, ISSN 0178-7675, 1432-0924, doi:10.1007/s00466-013-0854-7.

[17] M. Moumnassi, S. Belouettar, E. Béchet, S. P. Bordas, D. Quoirin, M. Potier-Ferry, Finite element analysis on implicitly defined domains: An accurate representation based on arbitrary parametric surfaces, Computer Methods in Applied Mechanics and Engineering 200 (5-8) (2011) 774–796, ISSN 00457825, doi:10.1016/j.cma.2010.10.002.

[18] P. Vos, R. van Loon, S. J. Sherwin, A comparison of fictitious domain methods appropriate for spectral/hp element discretisations, Computer Methods in Applied Mechanics and Engineering 197 (25-28) (2007) 2275–2289, ISSN 00457825, doi:10.1016/j.cma.2007.11.023.

[19] H. Lian, S. Bordas, R. Sevilla, R. Simpson, Recent Developments in the Integration of Computer Aided Design and Analysis, Computational Technology Reviews 6 (2012) 1–36, ISSN 2044-8430, doi:10.4203/ctr.6.1.

[20] J. Parvizian, A. Düster, E. Rank, Finite cell method, Computational Mechanics 41 (1) (2007) 121–133, ISSN 0178-7675, doi:10.1007/s00466-007-0173-y.

[21] A. Düster, J. Parvizian, Z. Yang, E. Rank, The finite cell method for three-dimensional problems of solid mechanics, Computer Methods in Applied Mechanics and Engineering 197 (45-48) (2008) 3768–3782, ISSN 00457825, doi:10.1016/j.cma.2008.02.036.

[22] J. Parvizian, A. Düster, E. Rank, Topology optimization using the finite cell method, Optimization and Engineering 13 (1) (2011) 57–78, ISSN 1389-4420, doi:10.1007/s11081-011-9159-x.

[23] D. Schillinger, M. Ruess, N. Zander, Y. Bazilevs, A. Düster, E. Rank, Small and large deformation analysis with the p- and B-spline versions of the Finite Cell Method, Computational Mechanics 50 (4) (2012) 445–478, ISSN 0178-7675, doi:10.1007/s00466-012-0684-z.

[24] D. Schillinger, E. Rank, An unfitted hp-adaptive finite element method based on hierarchical B-splines for interface problems of complex geometry, Computer Methods in Applied Mechanics and Engineering 200 (47-48) (2011) 3358–3380, ISSN 00457825, doi:10.1016/j.cma.2011.08.002.

[25] D. Schillinger, A. Düster, E. Rank, The hp-d-adaptive finite cell method for geometrically nonlinear problems of solid mechanics, International Journal for Numerical Methods in Engineering 89 (9) (2012) 1171–1202, ISSN 1097-0207, doi:10.1002/nme.3289.

[26] D. Schillinger, L. Dedè, M. A. Scott, J. A. Evans, M. J. Borden, E. Rank, T. J. Hughes, An isogeometric design-through-analysis methodology based on adaptive hierarchical refinement of NURBS, immersed boundary methods, and T-spline CAD surfaces, Computer Methods in Applied Mechanics and Engineering 249-252 (2012) 116–150, ISSN 00457825, doi:10.1016/j.cma.2012.03.017.

[27] D. Schillinger, The p- and B-spline versions of the geometrically nonlinear finite cell method and hierarchical refinement strategies for adaptive isogeometric and embedded domain analysis, Doctoral thesis, Technische Universität München, Chair for Computation in Engineering, 2012.

[28] Z. Yang, M. Ruess, S. Kollmannsberger, A. Düster, E. Rank, An efficient integration technique for the voxel-based finite cell method, International Journal for Numerical Methods in Engineering 91 (5) (2012) 457–471, ISSN 00295981, doi:10.1002/nme.4269.

[29] Z. Yang, S. Kollmannsberger, A. Düster, M. Ruess, E. G. Garcia, R. Burgkart, E. Rank, Non-standard bone simulation: interactive numerical analysis by computational steering, Computing and Visualization in Science 14 (5) (2012) 207–216, ISSN 1432-9360, doi:10.1007/s00791-012-0175-y.

[30] M. Ruess, D. Tal, N. Trabelsi, Z. Yosibash, E. Rank, The finite cell method for bone simulations: verification and validation., Biomechanics and modeling in mechanobiology 11 (3-4) (2012) 425–37, ISSN 1617-7940, doi:10.1007/s10237-011-0322-2.

[31] A. Düster, H. Sehlhorst, E. Rank, Numerical homogenization of heterogeneous and cellular materials utilizing the finite cell method, Computational Mechanics 50 (4) (2012) 413–431, ISSN 0178-7675, 1432-0924, doi:10.1007/s00466-012-0681-2.

[32] A. Abedian, J. Parvizian, A. Düster, E. Rank, The finite cell method for the J2 flow theory of plasticity, Finite Elements in Analysis and Design 69 (2013) 37–47, ISSN 0168874X, doi:10.1016/j.finel.2013.01.006.

[33] S. Duczek, M. Joulaian, A. Düster, U. Gabbert, Numerical analysis of Lamb waves using the finite and spectral cell method, International Journal for Numerical Methods in Engineering (Accepted for publication, 2014).

[34] S. Duczek, M. Joulaian, A. Düster, U. Gabbert, Simulation of Lamb waves using the spectral cell method, in: T. Kundu (Ed.), Proc. SPIE 8695, Health Monitoring of Structural and Biological Systems, doi:10.1117/12.2009983, 2013.

[35] M. Joulaian, A. Düster, Local enrichment of the finite cell method for problems with material interfaces, Computational Mechanics 52 (4) (2013) 741–762, ISSN 0178-7675, 1432-0924, doi:10.1007/s00466-013-0853-8.

[36] Q. Cai, S. Kollmannsberger, R. Mundani, E. Rank, The finite cell method for solute transport problems in porous media, in: Proceedings of the International Conference on Finite Elements in Flow Problems, 2011.

[37] Q. Cai, S. Kollmannsberger, R. Mundani, E. Rank, The Finite Cell Method for spatially varying dispersions in coupled multispecies reactive transport problems, in: Proc. of Coupled Problems 2011: Computational Methods for Coupled Problems in Science and Engineering, 2011.

[38] E. Rank, S. Kollmannsberger, C. Sorger, A. Düster, Shell Finite Cell Method: A high order fictitious domain approach for thin-walled structures, Computer Methods in Applied Mechanics and Engineering 200 (45-46) (2011) 3200–3209, ISSN 00457825, doi:10.1016/j.cma.2011.06.005.

[39] E. Rank, M. Ruess, S. Kollmannsberger, D. Schillinger, A. Düster, Geometric modeling, isogeometric analysis and the finite cell method, Computer Methods in Applied Mechanics and Engineering 249-252 (2012)

104–115, ISSN 00457825, doi:10.1016/j.cma.2012.05.022.
[40] M. Ruess, D. Schillinger, Y. Bazilevs, V. Varduhn, E. Rank, Weakly enforced essential boundary conditions for NURBS-embedded and trimmed NURBS geometries on the basis of the finite cell method, International Journal for Numerical Methods in Engineering 95 (10) (2013) 811–846, ISSN 00295981, doi:10.1002/nme.4522.
[41] M. Ruess, D. Schillinger, A. I. Özcan, E. Rank, Weak coupling for isogeometric analysis of non-matching and trimmed multi-patch geometries, Computer Methods in Applied Mechanics and Engineering 269 (2014) 46–71, ISSN 00457825, doi:10.1016/j.cma.2013.10.009.
[42] N. Zander, S. Kollmannsberger, M. Ruess, Z. Yosibash, E. Rank, The Finite Cell Method for linear thermoelasticity, Computers & Mathematics with Applications 64 (11) (2012) 3527–3541, ISSN 08981221, doi:10.1016/j.camwa.2012.09.002.
[43] I. Tsukanov, V. Shapiro, Meshfree modeling and analysis of physical fields in heterogeneous media, Advances in Computational Mathematics 23 (1-2) (2005) 95–124, ISSN 1019-7168, doi:10.1007/s10444-004-1835-3.
[44] M. Aghdam, N. Shahmansouri, M. Mohammadi, Extended Kantorovich method for static analysis of moderately thick functionally graded sector plates, Mathematics and Computers in Simulation 86 (2012) 118–130, ISSN 03784754, doi:10.1016/j.matcom.2010.07.029.
[45] R. Sanches, P. Bornemann, F. Cirak, Immersed b-spline (i-spline) finite element method for geometrically complex domains, Computer Methods in Applied Mechanics and Engineering 200 (13-16) (2011) 1432–1445, ISSN 00457825, doi:10.1016/j.cma.2010.12.008.
[46] P. Bornemann, F. Cirak, A subdivision-based implementation of the hierarchical b-spline finite element method, Computer Methods in Applied Mechanics and Engineering 253 (2013) 584–598, ISSN 00457825, doi:10.1016/j.cma.2012.06.023.
[47] T. Vejchodský, P. Šolín, M. Zítka, Modular hp-FEM system HERMES and its application to Maxwell's equations, Mathematics and Computers in Simulation 76 (1-3) (2007) 223–228, ISSN 03784754, doi:10.1016/j.matcom.2007.02.001.
[48] Y. Renard, J. Pommier, GetFEM++, An open-source finite element library, http://download.gna.org/getfem/html/homepage/, 2014.
[49] W. Bangerth, R. Hartmann, G. Kanschat, deal.II—A general-purpose object-oriented finite element library, ACM Transactions on Mathematical Software 33 (4) (2007) 24–es, ISSN 00983500, doi:10.1145/1268776.1268779.
[50] W. Bangerth, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin, T. D. Young, THE deal. II LIBRARY, VERSION 8.1, arXiv preprint arXiv:1312.2266 .
[51] S. Bordas, P. V. Nguyen, C. Dunant, A. Guidoum, H. Nguyen-Dang, An extended finite element library, International Journal for Numerical Methods in Engineering 71 (6) (2007) 703–732, ISSN 00295981, 10970207, doi:10.1002/nme.1966.
[52] V. P. Nguyen, T. Rabczuk, S. Bordas, M. Duflot, Meshless methods: A review and computer implementation aspects, Mathematics and Computers in Simulation 79 (3) (2008) 763–813, ISSN 03784754, doi:10.1016/j.matcom.2008.01.003.
[53] C. de Falco, A. Reali, R. Vázquez, GeoPDEs: A research tool for Isogeometric Analysis of PDEs, Advances in Engineering Software 42 (12) (2011) 1020–1034, ISSN 0965-9978, doi:10.1016/j.advengsoft.2011.06.010.
[54] V. P. Nguyen, S. P. A. Bordas, T. Rabczuk, Isogeometric analysis: an overview and computer implementation aspects, arXiv:1205.2129 .
[55] T. J. R. Hughes, The finite element method: linear static and dynamic finite element analysis, Dover Publications, Mineola, NY, ISBN 0486411818, 2000.
[56] O. Zienkiewicz, R. Taylor, J. Zhu, The finite element method: its basis and fundamentals, Butterworth-Heinemann, 6 edn., ISBN 0750663200, 2005.
[57] B. A. Szabó, I. Babuska, Finite element analysis, John Wiley & Sons, New York, ISBN 0471502731, 1991.
[58] K. J. Bathe, Finite element procedures, Prentice Hall, ISBN 097900490X, 2007.
[59] B. A. Szabó, A. Düster, E. Rank, The p-version of the finite element method, in: E. Stein (Ed.), Encyclopedia of Computational mechanics, John Wiley & Sons, Ltd, ISBN 9780470091357, 2004.
[60] J. Cottrell, T. J. Hughes, Y. Bazilevs, Isogeometric Analysis: Toward Integration of CAD and FEA, John Wiley & Sons, New York, 2009.
[61] A. Düster, H. Bröker, E. Rank, The p-version of the finite element method for three-dimensional curved thin walled structures, International Journal for Numerical Methods in Engineering 52 (7) (2001) 673–703.
[62] R. Sevilla, S. Fernández-Méndez, A. Huerta, Comparison of high-order curved finite elements, International Journal for Numerical Methods in Engineering 87 (8) (2011) 719–734, ISSN 00295981, doi:10.1002/nme.3129.
[63] R. Sevilla, S. Fernández-Méndez, A. Huerta, NURBS-enhanced finite element method (NEFEM), International Journal for Numerical Methods in Engineering 76 (1) (2008) 56–83, ISSN 00295981, 10970207, doi:10.1002/nme.2311.
[64] R. Sevilla, S. Fernández-Méndez, A. Huerta, 3D NURBS-enhanced finite element method (NEFEM), International Journal for Numerical Methods in Engineering 88 (2) (2011) 103–125, ISSN 00295981, doi:10.1002/nme.3164.
[65] A. Stavrev, The role of higher-order geometry approximation and accurate quadrature in NURBS based immersed boundary methods, Master's thesis, Technische Universität München, 2012.
[66] L. Kudela, Highly Accurate Subcell Integration in the Context of The Finite Cell Method, Master's thesis, Technische Universität München, Chair for Computation in Engineering, 2013.
[67] A. Abedian, J. Parvizian, A. Düster, H. Khademyzadeh, E. Rank, Performance of Different Integration Schemes in Facing Discontinuities in the Finite Cell Method, International Journal of Computational Methods 10 (03) (2013) 1350002, ISSN 0219-8762, 1793-6969, doi:10.1142/S0219876213500023.
[68] K. Höllig, U. Reif, J. Wipper, Weighted extended B-spline approximation of Dirichlet problems, SIAM Journal on Numerical Analysis 39 (2) (2001) 442–462.
[69] X. Zhuang, Meshless methods: theory and application in 3D fracture modelling with level sets, Ph.D. thesis, University of Durham, 2010.
[70] A. Gerstenberger, W. A. Wall, An embedded Dirichlet formulation for 3D continua, International Journal for Numerical Methods in Engineering 82 (5) (2010) 537–563, ISSN 1097-0207, doi:10.1002/nme.2755.
[71] J. Nitsche, Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind, Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg 36 (1) (1971) 9–15, ISSN 0025-5858, doi:10.1007/BF02995904.
[72] Y. Bazilevs, T. Hughes, Weak imposition of Dirichlet boundary conditions in fluid mechanics, Computers & Fluids 36 (1) (2007) 12–26, ISSN 00457930, doi:10.1016/j.compfluid.2005.07.012.
[73] S. Fernández-Méndez, A. Huerta, Imposing essential boundary conditions in mesh-free methods, Computer Methods in Applied Mechanics and Engineering 193 (12-14) (2004) 1257–1275, ISSN 00457825, doi:10.1016/j.cma.2003.12.019.
[74] A. Embar, J. Dolbow, I. Harari, Imposing Dirichlet boundary conditions with Nitsche's method and spline-based finite elements, International Journal for Numerical Methods in Engineering 83 (March) (2010) 877–898, ISSN 00295981, doi:10.1002/nme.2863.
[75] P. Hansbo, Nitsche's method for interface problems in computational mechanics, GAMM-Mitteilungen 28 (2) (2005) 183–206.
[76] D. N. Arnold, F. Brezzi, B. Cockburn, L. D. Marini, Unified Analysis of Discontinuous Galerkin Methods for Elliptic Problems, SIAM Journal on Numerical Analysis 39 (5) (2002) 1749–1779, ISSN 0036-1429, doi:10.1137/S0036142901384162.
[77] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design patterns: Elements of reusable object-oriented software, Addison-Wesley, 1 edn., ISBN 0201633612, 1995.
[78] The MathWorks, Inc., Getting Started with MATLAB, Tech. Rep., 2013.
[79] C. B. Moler, Numerical Computing with MATLAB, Society for Industrial and Applied Mathematics, 2 edn., ISBN 978-0898716603, 2010.
[80] R. C. Martin (Ed.), Clean code: a handbook of agile software craftsmanship, Prentice Hall, Upper Saddle River, NJ, ISBN 0132350882, 2009.
[81] K. Beck, C. Andres, Extreme programming explained, Addison-Wesley, Boston, 2nd edn., ISBN 0321278658, 2005.
[82] The MathWorks, Inc., MATLAB Unit Testing Framework, http://www.mathworks.de/de/help/matlab/matlab-unit-test-framework.html, 2014.
[83] K. Kawaguchi, Jenkins CI, http://jenkins-ci.org/, 2014.

[84] Apache Software Foundation, Apache Subversion, http://subversion.apache.org/, 2014.
[85] C. A. Felippa, Introduction to Finite Element Methods, 2013.
[86] J. Bonnet, R. D. Wood, Nonlinear Continuum Mechanics For Finite Element Analysis, Cambrigde University Press, 2nd edn., 2008.
[87] Q. Cai, Finite Cell Method for Transport Problems in Porous Media, Doctoral thesis, Technische Universität München, Munich, 2013.
[88] N. Trabelsi, Z. Yosibash, C. Wutte, P. Augat, S. Eberle, Patient-specific finite element analysis of the human femur–a double-blinded biomechanical validation., Journal of Biomechanics 44 (9) (2011) 1666–72, ISSN 1873-2380, doi:10.1016/j.jbiomech.2011.03.024.
[89] H. Wille, E. Rank, Z. Yosibash, Prediction of the mechanical response of the femur with uncertain elastic properties., Journal of Biomechanics 45 (7) (2012) 1140–8, ISSN 1873-2380, doi:10.1016/j.jbiomech.2012.02.006.
[90] ANSYS, Inc., ANSYS Release 14.0, Help System, Element Reference, 2011.