The GalerkinTools library: Efficient implementation of Galerkin type finite element formulations

Sebastian Stark[1]

[1]*Centre for Research in Computational and Applied Mechanics, University of Cape Town, 7701 Rondebosch, South Africa*

1 Introduction

The numerical solution of problems from continuum mechanics and related fields by means of Galerkin type finite element methods usually follows the same general steps. Despite this fact, most commercial finite element codes are limited to certain standard applications. In the case of non-standard applications on the contrary, it is necessary to develop custom codes. In this context, open source finite element libraries like deal.II (Alzetta et al., 2018; Bangerth et al., 2007) provide a powerful framework comprising implementations of the standard building blocks of finite element software. Still, a significant amount of coding is often necessary to implement a numerical scheme for a particular problem. This applies especially to the assembly of the finite element system. Although, for most problems, the really problem specific part boils down to a few simple functions emerging already in the continuous setting (and certain derivatives thereof), the implementation of the assembly procedures is usually a tedious and error prone task. This has motivated the development of the GalerkinTools library, which is based on deal.II and assists the user during assembly for the case that the finite element formulation is obtained based on a standard Galerkin approach. In particular, the task of the user is essentially reduced to the definition of the underlying weak form in a suitable way, the finite element mesh, the finite element space, and, possibly, constraints applying to the latter. The main purpose of this note is to describe (i) the problem type which can be handled by the GalerkinTools library, (ii) the main steps necessary to define the problem in practice, and (iii) some of the internal procedures of the GalerkinTools library.

2 Spatially continuous problem under consideration

In the following, the weak form under consideration is described. In this context, it is for formal reasons pretended that a scalar valued potential function Π exists such that the weak form follows from the requirement that the first variation of Π vanishes. However, the problems which can be handled by the GalerkinTools library need not necessarily be associated with such a potential.

It is also remarked that no explicit dependence on time is considered. However, time dependent problems can still be handled if the weak form is already discrete in time.

2.1 Spatial domain and interfaces

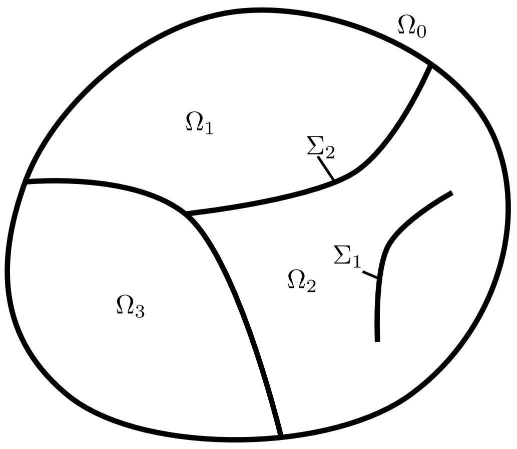

Figure 1: Domain under consideration.

The Euclidean n-space $\mathbb{R}^n$ with $n = 2$ or $n = 3$ is considered. $\mathbb{R}^n$ is split into the "computational domain" $\Omega \subset \mathbb{R}^n$ and the "environment" $\Omega_0 = \mathbb{R}^n \setminus \Omega$, see also Figure 1. The computational domain is further subdivided into "domain portions" Ω_α, where $\alpha \in A = \{1, \ldots, N^\Omega\}$, $\bigcup_{\alpha \in A} \Omega_\alpha = \Omega$, and $\Omega_\alpha \cap \Omega_\beta = 0$ if $\alpha \neq \beta$. Additionally, the $n-1$ dimensional "interface" Σ is introduced. This interface is further subdivided into "interface portions" Σ_γ, where $\gamma \in \Gamma = \{1, \ldots, N^\Sigma\}$, and $\Sigma_\gamma \cap \Sigma_\delta = 0$ if $\gamma \neq \delta$. Interface portions may align with interfaces between the domain portions Ω_α or the interface between Ω and Ω_0, but need not to. Together with the interface, a piecewise continuous unit normal field $\boldsymbol{n}(\boldsymbol{X})$ satisfying $\boldsymbol{n}(\boldsymbol{X}) \cdot \boldsymbol{n}(\boldsymbol{X}) = 1$ is defined. Using this normal field, the '+' and '−' sides of the interface at a point $\boldsymbol{X} \in \Sigma$ are defined such that $\boldsymbol{n}(\boldsymbol{X})$ points from − to +.

2.2 Unknowns of the problem: independent fields and scalars

The unknowns of the problem are a set of "independent fields" and, possibly, a set of "independent scalars". The independent fields may either be related to the domain (or portions thereof) or to the interface (or portions thereof), while the independent scalars have no spatial association.

The "domain related independent fields" $u^\Omega_\epsilon \in \mathcal{V}^\Omega_\epsilon \subseteq \mathcal{L}^2(\mathbb{R}^n)$ are introduced according to[1]

$$u^\Omega_\epsilon : \begin{cases} \mathbb{R}^n \rightarrow \mathbb{R} \\ \boldsymbol{X} \mapsto u^\Omega_\epsilon(\boldsymbol{X}), \end{cases} \tag{1}$$

where $\epsilon \in E = \{1, \ldots, N^{\mathrm{u},\Omega}\}$, and $\boldsymbol{X} \in \mathbb{R}^n$.

[1]Defining u^Ω_ϵ on the entire $\mathbb{R}^n$ instead of on Ω has the advantage that the boundary of the domain can formally be considered as an interface. The latter viewpoint is also physically motivated: A boundary may be seen as a special interface, where the situation on one side of it is known to some extent (in particular, to the extent it influences the situation on the other side of the boundary).

Similarly, the “interface related independent fields” $u_\eta^\Sigma \in \mathcal{V}_\eta^\Sigma \subseteq \mathcal{L}^2(\Sigma)$ are introduced according to

$$u_\eta^\Sigma : \begin{cases} \Sigma \to \mathbb{R} \\ \boldsymbol{X} \mapsto u_\eta^\Sigma(\boldsymbol{X}), \end{cases} \tag{2}$$

where $\eta \in H = \{1, \ldots, N^{\mathrm{u},\Sigma}\}$.

It will generally be assumed that $u_\epsilon^\Omega(\boldsymbol{X}) = 0$ if $\boldsymbol{X} \in \Omega_0$. Also, sets $A_{\epsilon,u\neq 0} \subset A$ and $\Gamma_{\eta,u\neq 0} \subset \Gamma$ are introduced. If $\alpha \notin A_{\epsilon,u\neq 0}$, then $u_\epsilon^\Omega(\boldsymbol{X}) = 0$ if $\boldsymbol{X} \in \Omega_\alpha$. Similarly, if $\gamma \notin \Gamma_{\eta,u\neq 0}$, then $u_\eta^\Sigma(\boldsymbol{X}) = 0$ if $\boldsymbol{X} \in \Sigma_\gamma$. I.e., the sets $A_{\epsilon,u\neq 0}$ and $\Gamma_{\eta,u\neq 0}$ determine on which domain portions and interface portions the unknown fields u_ϵ^Ω and u_η^Σ may be nonzero, and on which domain and interface portions they are constrained to zero. This approach effectively allows for different sets of unknown fields on different domain/interface portions. In addition, further problem specific constraints may apply to u_ϵ^Ω and u_η^Σ. It is understood that all constraints as well as regularity requirements on the independent fields are incorporated by choosing appropriate spaces $\mathcal{V}_\epsilon^\Omega$ and $\mathcal{V}_\eta^\Sigma$. Moreover, it is noted that, although each independent field is scalar valued, tensor valued unknown fields can easily be accounted for by combining several scalar valued fields.

Finally, the “independent scalars” $C_\iota \in \mathbb{R}$ are introduced, where $\iota \in I = \{1, \ldots, N^{\mathrm{C}}\}$. Hence, the unknowns of the problem are $(u_\epsilon^\Omega, u_\eta^\Sigma, C_\iota)$.

2.3 Dependent fields

The “domain related dependent fields” are introduced according to

$$e_\lambda^\Omega = \sum_{\epsilon\in E} \left[a_{\lambda\epsilon}^\Omega u_\epsilon^\Omega + \boldsymbol{b}_{\lambda\epsilon}^\Omega \cdot (\nabla u_\epsilon^\Omega)\right] + \sum_{\iota\in I} c_{\lambda\iota}^\Omega C_\iota + d_\lambda^\Omega. \tag{3}$$

Here, $\lambda \in \Lambda = \{1, \ldots, N^{\mathrm{e},\Omega}\}$; $a_{\lambda\epsilon}^\Omega$, $c_{\lambda\iota}^\Omega$ and d_λ^Ω are constants; and $\boldsymbol{b}_{\lambda\epsilon}^\Omega$ are constant vectors.

Similarly, the “interface related dependent fields” are introduced according to

$$\begin{aligned} e_\nu^\Sigma = &\sum_{\eta\in H} \left[a_{\nu\eta}^\Sigma u_\eta^\Sigma + \boldsymbol{b}_{\nu\eta}^\Sigma \cdot (\nabla u_\eta^\Sigma)\right] \\ &+ \sum_{\epsilon\in E} \left[a_{\nu\epsilon}^+ (u_\epsilon^\Omega)^+ + \boldsymbol{b}_{\nu\epsilon}^+ \cdot (\nabla u_\epsilon^\Omega)^+\right] \\ &+ \sum_{\epsilon\in E} \left[a_{\nu\epsilon}^- (u_\epsilon^\Omega)^- + \boldsymbol{b}_{\nu\epsilon}^- \cdot (\nabla u_\epsilon^\Omega)^-\right] \\ &+ \sum_{\iota\in I} c_{\nu\iota}^\Sigma C_\iota + d_\nu^\Sigma. \end{aligned} \tag{4}$$

Here, $\nu \in N = \{1, \ldots, N^{\mathrm{e},\Sigma}\}$; and $a_{\nu\eta}^\Sigma$, $a_{\nu\epsilon}^+$, $a_{\nu\epsilon}^-$, $c_{\nu\iota}^\Sigma$ and d_ν^Σ are constants; and $\boldsymbol{b}_{\nu\eta}^\Sigma$, $\boldsymbol{b}_{\nu\epsilon}^+$, $\boldsymbol{b}_{\nu\epsilon}^-$ are constant vectors. The superscript $+/-$ on a domain related independent field indicates that it is evaluated on the $+/-$ side of the interface.

The concept of dependent fields is used to define quantities which are derived from the set of unknowns in terms of linear relations. For illustration of the concept, consider an electrostatics problem with the scalar potential φ as an independent field defined on the domain. Then, dependent fields would be the electric field components, which are the components of the negative of the gradient of φ. It is also remarked that the trivial case that a dependent field equals an independent field is explicitly allowed for.

2.4 Scalar functionals

A set of scalar valued "domain related scalar functionals" is introduced according to

$$H_\rho^\Omega = \int_\Omega h_\rho^\Omega(e_\lambda^\Omega, \boldsymbol{X})\mathrm{d}V, \tag{5}$$

where $\rho \in P = \{1, \ldots, N^{\mathrm{H},\Omega}\}$, and the h_ρ^Ω are problem specific functions, which may be nonlinear.

Similarly, a set of scalar valued "interface related scalar functionals" is introduced according to

$$H_\tau^\Sigma = \int_\Sigma h_\tau^\Sigma(e_\nu^\Sigma, \boldsymbol{X})\mathrm{d}S, \tag{6}$$

where $\tau \in T = \{1, \ldots, N^{\mathrm{H},\Sigma}\}$, and the h_τ^Σ are again problem specific functions, which may be nonlinear.

In the same way as for the independent fields, sets $A_{\rho,h\neq 0} \subset A$ and $\Gamma_{\tau,h\neq 0} \subset \Gamma$ are introduced. If $\alpha \notin A_{\rho,h\neq 0}$, then $h_\rho^\Omega(e_\lambda^\Omega, \boldsymbol{X}) = 0$ if $\boldsymbol{X} \in \Omega_\alpha$. Similarly, if $\gamma \notin \Gamma_{\tau,h\neq 0}$, then $h_\tau^\Sigma(e_\nu^\Sigma, \boldsymbol{X}) = 0$ if $\boldsymbol{X} \in \Sigma_\gamma$. I.e., the sets $A_{\epsilon,h\neq 0}$ and $\Gamma_{\eta,h\neq 0}$ determine on which domain portions and interface portions the functions h_ρ^Ω and h_τ^Σ may be non-zero, and on which domain and interface portions they are set to zero. This approach effectively allows for different sets of scalar functionals on different domain/interface portions.

2.5 Scalar valued potential and weak form

Based on the previously introduced quantities, the scalar valued potential is finally written as

$$\Pi = \Pi(H_\rho^\Omega, H_\tau^\Sigma, C_\iota); \tag{7}$$

and the equations determining the unknown fields are obtained by requiring that the first variation of Π be stationary.

Using the definitions

$$\begin{aligned}
\Pi_\rho^\Omega &= \frac{\partial \Pi}{\partial H_\rho^\Omega}, \\
\Pi_\tau^\Sigma &= \frac{\partial \Pi}{\partial H_\tau^\Sigma}, \\
\Pi_\iota^\mathrm{C} &= \frac{\partial \Pi}{\partial C_\iota}, \\
h_{\rho\lambda}^\Omega &= \frac{\partial h_\rho^\Omega}{\partial e_\lambda^\Omega}, \\
h_{\tau\nu}^\Sigma &= \frac{\partial h_\tau^\Sigma}{\partial e_\nu^\Sigma}
\end{aligned} \tag{8}$$

and the usual notation of variational calculus, the condition for Π to be stationary may be expressed as

$$\begin{aligned}
\delta\Pi(u_\epsilon^\Omega, u_\eta^\Sigma, C_m, \delta u_\epsilon^\Omega, \delta u_\eta^\Sigma, \delta C_\iota) = \sum_{\rho\in P}\sum_{\lambda\in\Lambda} \Pi_\rho^\Omega \int_\Omega h_{\rho\lambda}^\Omega \delta e_\lambda^\Omega \mathrm{d}V + \sum_{\tau\in T}\sum_{\nu\in N} \Pi_\tau^\Sigma \int_\Sigma h_{\tau\nu}^\Sigma \delta e_\nu^\Sigma \mathrm{d}S \\
+ \sum_{\iota\in I} \Pi_\iota^\mathrm{C} \delta C_\iota = 0 \quad \forall(\delta u_\epsilon^\Omega, \delta u_\eta^\Sigma, \delta C_\iota).
\end{aligned} \tag{9}$$

The latter weak form defines the problem under consideration. In the following, it will be assumed that this problem is well-posed.

2.6 Remarks

1. As indicated earlier, Eq. (9) can still be used for the determination of the unknowns $(u^{\Omega}_{\epsilon}, u^{\Sigma}_{\eta}, C_{\iota})$ if Π does not exist and only Π^{Ω}_{ρ}, Π^{Σ}_{τ}, and Π^{C}_{ι} are defined. Furthermore, if Π^{Ω}_{ρ}, Π^{Σ}_{τ}, and Π^{C}_{ι} are all independent of a particular H^{Ω}_{ρ} (or H^{Σ}_{τ}), it is not necessary to give a relation for that H^{Ω}_{ρ} (or H^{Σ}_{τ}). Rather, it is then sufficient, to define $h^{\Omega}_{\rho\lambda}$ (or $h^{\Sigma}_{\tau\nu}$). However, existence of Π is a desirable property because (i) the mathematical analysis of the problem is typically simplified, and (ii) the resulting finite element system is usually symmetric.

2. A restriction of (9) is that only up to first derivatives of the unknown fields are allowed for. Despite this limitation, problems involving higher derivatives may still be treated within the present framework by using auxiliary independent fields representing higher derivatives of the unknown fields and incorporating the resulting constraints by a Lagrangian multiplier approach. In the context of finite element analysis, this approach is often also advantageous because the requirement of higher order continuous finite elements is circumvented.

3. The introduction of the independent scalars C_{ι} together with the assumption of a general function for Π (instead of just adding the scalar functions H^{Ω}_{ρ} and H^{Σ}_{τ} up to a scalar valued potential) allow for the incorporation of certain integral type equations. These do, e.g., emerge in an electrostatics problem if a total electrical charge is to be prescribed on an electrode.

3 Spatially discrete problem under consideration

The spatially discrete problem is obtained based on the weak form (9) by the Galerkin approach. I.e., the approximate solution is given by those $u^{\Omega,\mathrm{h}}_{\epsilon} \in \mathcal{V}^{\Omega,\mathrm{h}}_{\epsilon}$, $u^{\Sigma,\mathrm{h}}_{\eta} \in \mathcal{V}^{\Sigma,\mathrm{h}}_{\eta}$, $C^{\mathrm{h}}_{\iota} \in \mathbb{R}$ satisfying

$$\delta\Pi(u^{\Omega,\mathrm{h}}_{\epsilon}, u^{\Sigma,\mathrm{h}}_{\eta}, C^{\mathrm{h}}_{\iota}, \delta u^{\Omega,\mathrm{h}}_{\epsilon}, \delta u^{\Sigma,\mathrm{h}}_{\eta}, \delta C^{\mathrm{h}}_{\iota}) = 0 \quad \forall(\delta u^{\Omega,\mathrm{h}}_{\epsilon}, \delta u^{\Sigma,\mathrm{h}}_{\eta}, \delta C^{\mathrm{h}}_{\iota}), \tag{10}$$

where $\mathcal{V}^{\Omega,\mathrm{h}}_{\epsilon} \subset \mathcal{V}^{\Omega}_{\epsilon}$ and $\mathcal{V}^{\Sigma,\mathrm{h}}_{\eta} \subset \mathcal{V}^{\Sigma}_{\eta}$ are finite dimensional spaces.

Let $(w^{\Omega,\mathrm{h}}_{\epsilon\phi}, w^{\Sigma,\mathrm{h}}_{\eta\phi}, w^{\mathrm{C},\mathrm{h}}_{\iota\phi})$ with $\phi \in \Phi = \{1, \ldots, N^{\mathrm{dof}}\}$ be the ϕ-th basis vector of a basis of the N^{dof}-dimensional space $\mathcal{V}^{\Omega,\mathrm{h}}_{0} \times \ldots \times \mathcal{V}^{\Omega,\mathrm{h}}_{N^{\mathrm{u},\Omega}} \times \mathcal{V}^{\Sigma,\mathrm{h}}_{0} \times \ldots \times \mathcal{V}^{\Sigma,\mathrm{h}}_{N^{\mathrm{u},\Sigma}} \times \mathbb{R}^{N^{\mathrm{C}}}$. Then, $u^{\Omega,\mathrm{h}}_{\epsilon}$, $u^{\Sigma,\mathrm{h}}_{\eta}$ and C^{h}_{ι} may be expressed in terms of the scalar unknowns $\hat{u}_{\phi}$ according to

$$\begin{aligned} u^{\Omega,\mathrm{h}}_{\epsilon} &= \sum_{\phi\in\Phi} \hat{u}_{\phi} w^{\Omega,\mathrm{h}}_{\epsilon\phi} \\ u^{\Sigma,\mathrm{h}}_{\eta} &= \sum_{\phi\in\Phi} \hat{u}_{\phi} w^{\Sigma,\mathrm{h}}_{\eta\phi} \\ C^{\mathrm{h}}_{\iota} &= \sum_{\phi\in\Phi} \hat{u}_{\phi} w^{\mathrm{C},\mathrm{h}}_{\iota\phi}. \end{aligned} \tag{11}$$

Also, it is readily seen that (10) is equivalent to require that

$$\delta\Pi(u_\epsilon^{\Omega,\mathrm{h}}, u_\eta^{\Sigma,\mathrm{h}}, C_\iota^{\mathrm{h}}, w_{\epsilon\phi}^{\Omega,\mathrm{h}}, w_{\eta\phi}^{\Sigma,\mathrm{h}}, w_{\iota\phi}^{\mathrm{C,h}}) = 0 \quad \forall \phi \in \Phi, \tag{12}$$

which are N^{dof} equations for the determination of the N^{dof} unknowns $\hat{u}_\phi$.

Assuming that the spatial discretization is based on the finite element method, the structure of the resulting finite element system will be discussed in the following. For this purpose, first the quantities

$$\begin{aligned}
h_{\rho\lambda\mu}^{\Omega} &= \frac{\partial h_{\rho\lambda}^{\Omega}}{\partial e_\mu^{\Omega}},\\
h_{\tau\nu\xi}^{\Sigma} &= \frac{\partial h_{\tau\nu}^{\Sigma}}{\partial e_\xi^{\Sigma}},\\
\Pi_{\rho\sigma}^{\Omega\Omega} &= \frac{\partial \Pi_\rho^{\Omega}}{\partial H_\sigma^{\Omega}}, \quad \Pi_{\rho\tau}^{\Omega\Sigma} = \frac{\partial \Pi_\rho^{\Omega}}{\partial H_\tau^{\Sigma}}, \quad \Pi_{\rho\iota}^{\Omega\mathrm{C}} = \frac{\partial \Pi_\rho^{\Omega}}{\partial C_\iota},\\
\Pi_{\tau\rho}^{\Sigma\Omega} &= \frac{\partial \Pi_\tau^{\Sigma}}{\partial H_\rho^{\Omega}}, \quad \Pi_{\tau\upsilon}^{\Sigma\Sigma} = \frac{\partial \Pi_\tau^{\Sigma}}{\partial H_\upsilon^{\Sigma}}, \quad \Pi_{\tau\iota}^{\Sigma\mathrm{C}} = \frac{\partial \Pi_\tau^{\Sigma}}{\partial C_\iota},\\
\Pi_{\iota\rho}^{\mathrm{C}\Omega} &= \frac{\partial \Pi_\iota^{\mathrm{C}}}{\partial H_\rho^{\Omega}}, \quad \Pi_{\iota\tau}^{\mathrm{C}\Sigma} = \frac{\partial \Pi_\iota^{\mathrm{C}}}{\partial H_\tau^{\Sigma}}, \quad \Pi_{\iota\kappa}^{\mathrm{CC}} = \frac{\partial \Pi_\iota^{\mathrm{C}}}{\partial C_\kappa},\\
v_{\lambda\phi}^{\Omega} &= \sum_{\epsilon\in E}\left[a_{\lambda\epsilon}^{\Omega} w_{\epsilon\phi}^{\Omega,\mathrm{h}} + \boldsymbol{b}_{\lambda\epsilon}^{\Omega}\cdot(\nabla w_{\epsilon\phi}^{\Omega,\mathrm{h}})\right] + \sum_{\iota\in I} c_{\lambda\iota}^{\Omega} w_{\iota\phi}^{\mathrm{C,h}},\\
v_{\nu\phi}^{\Sigma} &= \sum_{\eta\in H}\left[a_{\nu\eta}^{\Sigma} w_{\eta\phi}^{\Sigma,\mathrm{h}} + \boldsymbol{b}_{\nu\eta}^{\Sigma}\cdot(\nabla w_{\eta\phi}^{\Sigma,\mathrm{h}})\right]\\
&\quad + \sum_{\epsilon\in E}\left[a_{\nu\epsilon}^{+}(w_{\epsilon\phi}^{\Omega,\mathrm{h}})^{+} + \boldsymbol{b}_{\nu\epsilon}^{+}\cdot(\nabla w_{\epsilon\phi}^{\Omega,\mathrm{h}})^{+}\right]\\
&\quad + \sum_{\epsilon\in E}\left[a_{\nu\epsilon}^{-}(w_{\epsilon\phi}^{\Omega,\mathrm{h}})^{-} + \boldsymbol{b}_{\nu\epsilon}^{-}\cdot(\nabla w_{\epsilon\phi}^{\Omega,\mathrm{h}})^{-}\right]\\
&\quad + \sum_{\iota\in I} c_{\nu\iota}^{\Sigma} w_{\iota\phi}^{\mathrm{C,h}},\\
v_{\iota\phi}^{\mathrm{C}} &= w_{\iota\phi}^{\mathrm{C,h}}
\end{aligned} \tag{13}$$

are introduced. Furthermore, the vectors $\boldsymbol{f}^{\Omega}$, $\boldsymbol{f}^{\Sigma}$, $\boldsymbol{f}^{\mathrm{C}}$, and the matrices $\boldsymbol{K}^{\Omega}$, $\boldsymbol{K}^{\Sigma}$, $\boldsymbol{L}^{\Omega}$, $\boldsymbol{L}^{\Sigma}$

and $\boldsymbol{L}^{\mathrm{C}}$ are defined by

$$\begin{aligned}
f^{\Omega}_{\phi} &= \sum_{\rho\in P}\sum_{\lambda\in\Lambda}\Pi^{\Omega}_{\rho}\int_{\Omega} h^{\Omega}_{\rho\lambda}v^{\Omega}_{\lambda\phi}\mathrm{d}V,\\
f^{\Sigma}_{\phi} &= \sum_{\tau\in T}\sum_{\nu\in N}\Pi^{\Sigma}_{\tau}\int_{\Sigma} h^{\Sigma}_{\tau\nu}v^{\Sigma}_{\nu\phi}\mathrm{d}S,\\
f^{\mathrm{C}}_{\phi} &= \sum_{\iota\in I}\Pi^{\mathrm{C}}_{\iota}v^{\mathrm{C}}_{\iota\phi},\\
K^{\Omega}_{\phi\chi} &= \sum_{\rho\in P}\sum_{\lambda\in\Lambda}\sum_{\mu\in\Lambda}\Pi^{\Omega}_{\rho}\int_{\Omega} h^{\Omega}_{\rho\lambda\mu}v^{\Omega}_{\lambda\phi}v^{\Omega}_{\mu\chi}\mathrm{d}V,\\
K^{\Sigma}_{\phi\chi} &= \sum_{\tau\in T}\sum_{\nu\in N}\sum_{\xi\in N}\Pi^{\Sigma}_{\tau}\int_{\Sigma} h^{\Sigma}_{\tau\nu\xi}v^{\Sigma}_{\nu\phi}v^{\Sigma}_{\xi\chi}\mathrm{d}S,\\
L^{\Omega}_{\phi\rho} &= \sum_{\lambda\in\Lambda}\int_{\Omega} h^{\Omega}_{\rho\lambda}v^{\Omega}_{\lambda\phi}\mathrm{d}V,\\
L^{\Sigma}_{\phi\tau} &= \sum_{\nu\in N}\int_{\Sigma} h^{\Sigma}_{\tau\nu}v^{\Sigma}_{\nu\phi}\mathrm{d}S,\\
L^{\mathrm{C}}_{\phi\iota} &= v^{\mathrm{C}}_{\iota\phi}.
\end{aligned}\tag{14}$$

Using these definitions, linearization of (12) at a point $\hat{\boldsymbol{u}}$ yields the linear system

$$\underbrace{\boldsymbol{f}^{\Omega}+\boldsymbol{f}^{\Sigma}+\boldsymbol{f}^{\mathrm{C}}}_{\boldsymbol{f}}+\left[\underbrace{\boldsymbol{K}^{\Omega}+\boldsymbol{K}^{\Sigma}}_{\boldsymbol{K}}+\underbrace{\begin{pmatrix}\boldsymbol{L}^{\Omega} & \boldsymbol{L}^{\Sigma} & \boldsymbol{L}^{\mathrm{C}}\end{pmatrix}}_{\boldsymbol{L}}\underbrace{\begin{pmatrix}\boldsymbol{\Pi}^{\Omega\Omega} & \boldsymbol{\Pi}^{\Omega\Sigma} & \boldsymbol{\Pi}^{\Omega\mathrm{C}}\\ \boldsymbol{\Pi}^{\Sigma\Omega} & \boldsymbol{\Pi}^{\Sigma\Sigma} & \boldsymbol{\Pi}^{\Sigma\mathrm{C}}\\ \boldsymbol{\Pi}^{\mathrm{C}\Omega} & \boldsymbol{\Pi}^{\mathrm{C}\Sigma} & \boldsymbol{\Pi}^{\mathrm{CC}}\end{pmatrix}}_{\boldsymbol{\Pi}}\begin{pmatrix}\boldsymbol{L}^{\Omega^{\top}}\\ \boldsymbol{L}^{\Sigma^{\top}}\\ \boldsymbol{L}^{\mathrm{C}^{\top}}\end{pmatrix}\right]\Delta\hat{\boldsymbol{u}}=\boldsymbol{0},\tag{15}$$

which can finally be written as

$$\boldsymbol{f}+\left(\boldsymbol{K}+\boldsymbol{L}\boldsymbol{\Pi}\boldsymbol{L}^{\top}\right)\Delta\hat{\boldsymbol{u}}=0.\tag{16}$$

The $N^{\mathrm{dof}}\times N^{\mathrm{dof}}$ matrix $\boldsymbol{K}$ is, in the context of the finite element method, usually sparse. Furthermore, $\boldsymbol{\Pi}$ will typically be a relatively small matrix, with dimension $(N^{\mathrm{H},\Omega}+N^{\mathrm{H},\Sigma}+N^{\mathrm{C}})\times(N^{\mathrm{H},\Omega}+N^{\mathrm{H},\Sigma}+N^{\mathrm{C}})$, where $N^{\mathrm{H},\Omega}+N^{\mathrm{H},\Sigma}+N^{\mathrm{C}}\ll N^{\mathrm{dof}}$. Despite this fact, the matrix $\boldsymbol{L}\boldsymbol{\Pi}\boldsymbol{L}^{\top}$ (and, therefore, also $\boldsymbol{K}+\boldsymbol{L}\boldsymbol{\Pi}\boldsymbol{L}^{\top}$) may be dense if $\boldsymbol{\Pi}\neq\boldsymbol{0}$. Iterative solvers requiring only the computation of matrix vector products can circumvent this issue by using an appropriate order of multiplication when evaluating $\boldsymbol{L}\boldsymbol{\Pi}\boldsymbol{L}^{\top}\Delta\hat{\boldsymbol{u}}$. However, solving (16) with a direct solver is not feasible in general. Thus, we shall rewrite (16) in a form more suitable for the application of a direct solver. To start with, $\boldsymbol{\Pi}$ is written as

$$\boldsymbol{\Pi}=\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\top},\tag{17}$$

where $\boldsymbol{D}$ is a diagonal matrix with its diagonal entries being either 1 or -1, and $\boldsymbol{U}=\boldsymbol{V}$ if $\boldsymbol{\Pi}=\boldsymbol{\Pi}^{\top}$. Such a decomposition can easily be obtained based on a singular value decomposition. Using (17), it follows that the solution $\Delta\hat{\boldsymbol{u}}$ of (16) is also part of the solution of the auxiliary system

$$\begin{pmatrix}\boldsymbol{f}\\ \boldsymbol{0}\end{pmatrix}+\begin{pmatrix}\boldsymbol{K} & \boldsymbol{L}\boldsymbol{U}\\ \boldsymbol{V}^{\top}\boldsymbol{L}^{\top} & -\boldsymbol{D}\end{pmatrix}\begin{pmatrix}\Delta\hat{\boldsymbol{u}}\\ \Delta\hat{\boldsymbol{\lambda}}\end{pmatrix}=\boldsymbol{0}.\tag{18}$$

Considering the facts that (i) the $N^{\mathrm{dof}} \times N^{\mathrm{dof}}$ matrix $\boldsymbol{K}$ is sparse in the context of the finite element method and (ii) the number of rows and columns, respectively, of $\boldsymbol{D}$ is $N^{\mathrm{H},\Omega} + N^{\mathrm{H},\Sigma} + N^{\mathrm{C}} \ll N^{\mathrm{dof}}$, the system matrix in (18) can be considered sparse. Therefore, direct solvers can be applied for the solution of (18). In this context, it is remarked that storage of the "stretched" matrix appearing in (18) as a whole may be disadvantageous in parallel computations, where certain processors own certain rows of the matrix. This is because a "natural" partitioning of the lower left block of the matrix is defined by the partitioning of the finite element mesh. However, this partitioning cannot be respected because the ownership of a particular row cannot be distributed to several processors. Therefore it is in parallel advantageous to store the matrices $\boldsymbol{K}$, $\boldsymbol{LU}$, $\boldsymbol{LV}$, and $\boldsymbol{D}$ separately. For similar reasons it may be desirable to store the rows and columns of $\boldsymbol{K}$ associated with independent scalars together with $\boldsymbol{LU}$, $\boldsymbol{LV}$, and $\boldsymbol{D}$, instead of including them into $\boldsymbol{K}$.

4 Problem definition, assembly and solution of the finite element system in practice

4.1 Problem definition

For the definition of the problem the following steps are required:

1. generation of the finite element mesh for domain and interface, and partitioning into domain and interface portions,
2. definition of the independent fields and independent scalars,
3. definition of the dependent fields,
4. definition of the scalar functionals (including first and second derivatives of the integrands h^{Ω}_{ρ} and h^{Σ}_{τ} w.r.t. the dependent fields),
5. definition of Π (including first and second derivatives w.r.t. the scalar functionals and the independent scalars).

These steps are briefly described below; and commented C++ example programs included in the documentation of the library show how these steps translate into source code.

4.1.1 Mesh generation and partitioning

As a first step, a coarse mesh of the domain Ω needs to be generated. For this, the standard sequential or distributed parallel triangulation features of deal.II are utilized. Subsequently, the domain is split into domain portions based on the *material_id* property of the cells in the coarse mesh. In the next step, an instance of the *TriangulationSystem* class needs to be created. This object knows of the triangulation of the domain, and is used to define the interface based on it. In particular, the interface is defined by selected faces of domain cells, with the convention that the underlying domain cell is on the "$-$" side of the interface. Internally, a coarse interface mesh is generated from the selected faces, and the interface portions are defined by the *material_id* property of the cells of the coarse interface mesh. It is remarked in this context that the interface elements do currently not share edges and vertices wherever two interface portions meet (i.e., the edges and vertices are duplicated). Another important aspect is that the identifiers for the manifold description of interface cells are taken over from the underlying faces of the domain cells in order to make sure that domain mesh and interface mesh remain consistent with each other upon mesh refinement. However, it is necessary to explicitly supply the *TriangulationSystem* object with the respective co-dimension 1 counterparts of the manifolds used for the domain triangulation because this cannot be easily automated.

After creation of the coarse domain and interface mesh (or at any later instant), the domain mesh may be refined. Upon refinement of the domain mesh, the interface mesh is automatically updated such that each coarse interface cell is refined to the same degree as the most refined of the two adjacent domain cells.

4.1.2 Definition of the independent fields and independent scalars

For the definition of independent fields, instances of the *IndependentField* class of the GalerkinTools library have to be created. In this context, the finite element used for discretization of the respective independent field is defined together with the domain/interface portions on which the independent field is non-zero. In order to allow for vector valued finite elements, it is also possible to define several independent fields at the same time and assign a vector valued finite element to these.

Independent scalars are as well defined in terms of instances of the *IndependentField* class.

4.1.3 Definition of dependent fields

The definition of dependent fields is done by creating instances of the *DependentField* class of the GalerkinTools library. In this context, methods are available which allow for specification of the relations (3) and (4) in terms of the *IndependentField* objects.

4.1.4 Definition of scalar functionals

For the definition of scalar functionals, the abstract class *ScalarFunctional* is available. Classes inheriting from this class essentially have to implement the methods *get_h_omega* and *get_h_sigma* for domain and interface related scalar functionals, respectively[2]. These methods are used to implement the functions h^{Ω}_{ρ} and h^{Σ}_{τ} and their first and second derivatives w.r.t. the dependent fields[3]. Besides the current values of the dependent fields, also a user-defined number of reference solutions (e.g. related to previous time steps) is passed into these functions for information purposes. Furthermore, it is possible to access an array storing data on the quadrature point level for each scalar functional, which allows for "hidden variables" like they are needed for classical plasticity theory models. When an instance of a derived class of *ScalarFunctional* is constructed, the dependent fields the scalar functional depends on must be supplied together with the domain/interface portions on which the scalar functional is non-zero.

[2]The main difference between *get_h_omega* and *get_h_sigma* is that the latter takes the normal vector as an additional argument.

[3]Depending on the situation, it may be sufficient to fill the data structures for the first and second "derivatives", see remark 1 in Sect. 2.6.

Optionally, the method *get_maximum_step* may be implemented within a class deriving from *ScalarFunctional*. This method can be useful within Newton-Raphson procedures in order to avoid that inaccessible states are attained after a Newton step. In particular, an increment direction for the dependent field values is passed into the method *get_maximum_step*, and the output is the maximum step which can be taken in this direction.

4.1.5 Definition of the potential Π

The potential Π is defined through the sum

$$\Pi = \sum_{i=1}^{N^\Pi} \Pi_i(H_\rho^\Omega, H_\tau^\Sigma, C_\iota). \tag{19}$$

The main reason for this split is to account for cases, where certain scalar functionals H_ρ^Ω or H_τ^Σ enter Π only as a summand (in the terminology of the library: they enter Π "primitively"), in which case the rows and columns of $\mathbf{\Pi}$ related to these particular scalar functionals are zero. This in turn allows to reduce the size of $\mathbf{\Pi}$ in practice. I.e., only those scalar functionals factor into $\mathbf{\Pi}$, which enter Π in a "non-primitive" (and, in particular, non-linear) way. Another reason for choosing the split in (19) is to allow for a clearer structuring of the problem, which may help to reduce computational cost, see also below.

Individual Π_i are defined through the *TotalPotentialContribution* class of the GalerkinTools library, or through derived classes. In this context, there are two options: (i) the Π_i is chosen to be equal to either one domain related scalar functional or one interface related scalar functional, (ii) the Π_i depends (in a typically non-linear way) on different scalar functionals and, possibly, independent scalars. In the latter case, the function Π_i must be defined together with its first and second derivatives w.r.t. the scalar functionals and independent scalars in a derived class of *TotalPotentialContribution*[4].

Subsequently, an instance of the class *TotalPotential* of the GalerkinTools library gathers all potential contributions Π_i into Π.

Finally, the problem definition is completed by joining the *TotalPotential* and the *TriangulationSystem* in an instance of the *AssemblyHelper* class of the GalerkinTools library. At this point it is also possible to specify the mappings used on the domain and the interface, respectively, to map the reference cells to real space. Upon creation of an instance of *AssemblyHelper* the degrees of freedom are distributed (i.e., they are globally enumerated).

4.2 Assembly of the finite element system

The assembly process consists of three steps:

1. definition of constraints,

2. generation of a sparsity pattern for the system matrix,

[4]If no potential exists, it is sufficient to implement the first and second "derivatives" of Π_i.

3. computation of the system matrix and the right hand side

$$\boldsymbol{K}^{\mathrm{s}} = \begin{pmatrix} \boldsymbol{K} & \boldsymbol{L}\boldsymbol{U} \\ \boldsymbol{V}^\top \boldsymbol{L}^\top & -\boldsymbol{D} \end{pmatrix} \tag{20}$$

   and the r.h.s. vector

$$-\boldsymbol{f}^{\mathrm{s}} = -\begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{pmatrix}. \tag{21}$$

These are explained below.

4.2.1 Definition of constraints

The constraints are gathered in a deal.II *AffineConstraints* object. Though an entirely manual definition of the constraints is possible (information about degrees of freedom can be obtained through the member functions of *AssemblyHelper*), the GalerkinTools library also provides with functionality to define usual constraint types. Firstly, hanging node constraints can automatically be generated. And, secondly, Dirichlet constraints of the form $(u^\Omega_\epsilon)^{+/-} = b^\Omega_\epsilon + c^\Omega_\epsilon C_\iota$, with b^Ω_ϵ and c^Ω_ϵ being constants, can be defined by instances of the class *DirichletConstraint* and then generated by a member of *AssemblyHelper*.

4.2.2 Generation of the sparsity pattern

Before generation of the sparsity pattern, it is necessary to decide whether the matrix $\boldsymbol{K}^{\mathrm{s}}$ is to be stored as a whole or block-wise, with the latter being the only reasonable option for parallel calculations. In case $\boldsymbol{K}^{\mathrm{s}}$ is stored as a whole, the standard deal.II (dynamic) sparsity pattern can be used, while the *TwoBlockSparsityPattern* offered by the GalerkinTools library is appropriate for block-wise storage. The generation of the sparsity pattern is then in both cases done by the member function *generate_sparsity_pattern_by_simulation* of the *AssemblyHelper* class.

4.2.3 Computation of the system matrix and the right hand side

Before computation of the system matrix and the right hand side, the following data structures have to be set up:

- the current solution vector, the content of which may be relevant or not depending on the form of the scalar functionals;
- possibly a number of reference solution vectors typically being solutions from previous time steps, as knowledge of states other than the current one may be required by scalar functionals (e.g. in transient calculations),
- the system matrix, which needs to be initialized with the sparsity pattern generated before (in case a *TwoBlockSparsityPattern* is used, the system matrix has to be an instance of the *TwoBlockMatrix* class provided by the GalerkinTools library),
- the right hand side vector (in case a *TwoBlockSparsityPattern* is used, this must be an appropriate *BlockVector* from the deal.II library).

Subsequently, a single call to the member function *assemble_system* of the *AssemblyHelper* class is required to assemble the system and, possibly, compute the value of Π if a potential exists. It is noted in this context that the strategy followed in the GalerkinTools library is to eliminate constraints immediately when assembling the system such that no condensation steps are required later.

4.3 Solution of the finite element system

The GalerkinTools library provides with direct solvers based on UMFPACK (Davis, 2004) and MUMPS (Amestoy et al., 2001, 2006). While the class *SolverWrapperUMFPACK* handles problems where $\boldsymbol{K}^{\mathrm{s}}$ is stored as a whole, the classes *BlockSolverWrapperUMFPACK* and *SolverWrapperPETSc* work with block-wise stored $\boldsymbol{K}^{\mathrm{s}}$, with *SolverWrapperPETSc* being the only option in parallel.

In case an iterative solver is needed, it must be implemented by the user.

5 Internal procedures

In the following, some of the internal procedures are briefly discussed. For further details, the reader is referred to the documentation of the GalerkinTools library.

5.1 Mesh

In general, separate meshes are stored in *TriangulationSystem* for the domain and the interface, with the relation between interface cells and faces of underlying domain cells being kept track of by additional internal data structures. Furthermore, internal triggers make sure that the interface mesh is updated automatically upon a change of the domain mesh.

An important aspect for parallel calculations is that the partitioning of the interface mesh is assumed to follow the partitioning of the cells on the minus side of the interface. This is necessary in order to avoid complex procedures and excessive communication between processors when calculating interface related scalar functionals. Currently, the approach to ensure that the domain and interface partitioning remain consistent is to first partition the domain mesh (which is automatically done by deal.II upon mesh generation or refinement based on the p4est library), and then completely rebuild the interface mesh based on the domain mesh, with the partitioning matching the one of the domain mesh. An implication of this procedure is that any transfer of data related to interface cells upon mesh refinement/repartitioning must be done through the underlying domain cells. However, the latter is not implemented yet, and, hence, a solution transfer upon mesh refinement is not easily possible at the moment.

5.2 Degree of freedom handling

The degrees of freedom are handled by the *DoFHandlerSystem* class of the GalerkinTools library. An instance of this class involves a domain related deal.II *hp::DoFHandler* and an interface related deal.II *hp::DoFHandler*. Here, the choice of the hp version is necessary in order to account for the fact that independent fields may be zero on certain portions of the domain/interface. In addition to the domain and interface related degrees of freedom, *DoFHandlerSystem* allows for a number of degrees of freedom not related to a mesh (these represent the independent scalars).

The standard convention for the numbering of degrees of freedom is that the domain related degrees of freedom are numbered first, followed by the interface related degrees of freedom and the degrees of freedom not related to a mesh. However this approach is not feasible in parallel, because the degrees of freedom owned by the processors would not form contiguous ranges due to the required consistency of the partitionings of the domain mesh and the interface mesh. To circumvent this issue, the standard numbering of the degrees of freedom can be changed.

In general, the degrees of freedom not related to a mesh are stored on the last processor. The decision not to distribute these degrees of freedom to different processors is based on the fact that usually only a few degrees of freedom not related to a mesh exist, and this number does also not increase upon mesh refinement.

5.3 Assembly

In order to facilitate the assembly process, the information contained in the instances of *TotalPotential* and *TriangulationSystem* is distributed into several data structures internal to *AssemblyHelper* when an instance of the latter is created. Though these data structures may seem cumbersome, they are designed to make the assembly process as efficient as possible.

An important aspect of the assembly process is that the values of all scalar functionals entering the total potential non-primitively must be computed beforehand. This is accomplished by the function *get_nonprimitive_scalar_functional_values*, which is called in the beginning of the *assemble_system* function.

Another aspect worth mentioning is the order of traversal of cells and scalar functionals during assembly (as well as during generation of the sparsity pattern and calculation of the values of the scalar functionals entering the total potential non-primitively). In this regard, the outer loop is over the cells, and for each cell it is looped over the scalar functionals. For a particular scalar functional on a particular cell, its contribution to the system matrix and right hand side is computed and distributed immediately. The latter approach has the advantage that, for each scalar functional on the cell, only those degrees of freedom need to be considered, which are affected by the scalar functional according to its dependency on independent scalars. For illustration of the consequences of this approach, consider a case with three domain related independent fields u_1^Ω, u_2^Ω and u_3^Ω. Further, it is assumed that the total potential has the form $\Pi = H_1^\Omega[e_1^\Omega(u_1^\Omega, u_2^\Omega)] + H_2^\Omega[e_2^\Omega(u_2^\Omega, u_3^\Omega)]$. When the contribution of H_1^Ω to the finite element system is computed, only degrees of freedom related to u_1^Ω and u_2^Ω have to be taken into account; and when the contribution of H_2^Ω to the finite element system is computed, only degrees of freedom related to u_2^Ω and u_3^Ω have to be taken into account. Thus, on the whole, degrees of freedom related to u_1^Ω do not couple to degrees of freedom related to u_3^Ω, and no corresponding entries have to be computed and allocated in the sparsity pattern of the system matrix. In contrast, if the scalar functionals H_1^Ω and H_2^Ω would be combined in a single scalar functional, the routines would not be able to detect that u_1^Ω does not couple to u_3^Ω and, consequently, compute the related entries in the finite element system (which are all zeros) and allocate the entries in the sparsity pattern of the system matrix. Due to this aspect, the user can reduce the computational effort by suitable structuring of the total potential.

6 Future work

Future work may, for example, include the following aspects:

- extension towards the automatic calculation of terms on boundaries between cells for discontinuous Galerkin methods,
- implementation of automatic solution transfer upon mesh refinement (including hidden variables at the quadrature points),
- extension towards degrees of freedom on lines (in three dimensions) and vertices (in three and two dimensions),
- allowing for a different trial and test function space.

References

G. Alzetta, D. Arndt, W. Bangerth, V. Boddu, B. Brands, D. Davydov, R. Gassmoeller, T. Heister, L. Heltai, K. Kormann, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells. The `deal.II` library, version 9.0. *Journal of Numerical Mathematics*, 26(4):173–183, 2018. doi: 10.1515/jnma-2018-0054.

P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

W. Bangerth, R. Hartmann, and G. Kanschat. deal.II – a general purpose object oriented finite element library. *ACM Trans. Math. Softw.*, 33(4):24/1–24/27, 2007.

T. A. Davis. Algorithm 832: UMFPACK v4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):196–199, 2004. doi: 10.1145/992200.992206.