Such a roadblock may be partially overcome by replacing the linear approximations by *nonlinear* mappings, which may be determined in turn by some type of *manifold learning algorithm*, as done in Ref. [28] for one-scale dynamic problems.

Of paramount importance for the consistency of the overall approach has been to adopt a partitioning framework equipped with fictitious interfaces, and to assign the role of coarse-scale DOFs to the amplitude of the modes of such interfaces. Furthermore, this choice has enabled us to establish a direct link with existing finite element beam theories. Indeed, we have seen in the rectangular beam example of Section 9.1 that our method leads to a 2-node element with 6 DOFs, as in a standard Timoshenko's element. The difference is that, in our approach, *it is the fictitious interfaces that remain plane*, not the faces of the slices, which are free to warp due to torsion and shear. This is why, remarkably, our 2-node beam element is totally *consistent* with the underlying 3D theory —as opposed to Timoshenko's or other analytical theories. Besides, the proposed methodology is able to automatically detect from the provided data when is necessary to enrich the kinematics with more DOFs. This has been fittingly illustrated by the I-shaped cross-section beam problem, in which Algorithm 3 has "discovered" the (not-so-well-known) fact that the kinematics of thin-walled open cross-section cannot be accurately represented with just translation and rotations, but require an additional warping DOF (see Figure 10). Another pivotal ingredient of the proposed approach has been the *hyperreduction* of the internal forces of the subdomains via the Empirical Cubature Method. We have observed in the hexagonal cell example of Section of 9.3 that the number of ECM points increases considerably when passing from the linear to the nonlinear regime, even if the kinematics remains practically the same. Therefore, it would be worthy to explore whether it is possible to further improve the ECM so that it returns less integration points for the same accuracy level. In principle, there is room for improvement, because the ECM, as well are other algorithms either based on nonnegative least-squares (such as the ECWS of Farhat et co-workers [12]), or on linear programming (such as the Empirical Quadrature Method of Patera et co-workers [38]), are not optimal, for they do not exploit the topology of the function they intend to integrate.

Acknowledgements

The research leading to these results has received funding from ANACONDA BIOMED S.L. and the Spanish Ministry of Science, Innovation and Universities via grant RTC-2017-6749-1. The author also acknowledges financial support from the Spanish Ministry of Economy and Competitiveness, through the "Severo Ochoa Programme for Centres of Excellence in R&D" (CEX2018-000797-S)

A. Empirical Cubature Method

A key offline step (see Box 8.1, item 5) is the selection among the Gauss points of the unit cell mesh of a reduced set of points able to accurately integrate (when equipped with specially tailored weights) the internal virtual work, see Eq. (49). Here this task is carried out using an algorithmically improved version —in a sense that will be defined later— of the *Empirical Cubature Method* (ECM), proposed by the author in Ref. [22]. As explained therein, the ECM operates on a snapshot matrix $\boldsymbol{A}^{\boldsymbol{f}}$ that depends on the value of the integrand at all the m_{gs} Gauss points and for all the P configurations . In this case, the integrand is the reduced internal work per unit volume $\boldsymbol{B}^{*T}(\boldsymbol{x}_g)\boldsymbol{\sigma}(\boldsymbol{x}_g,\boldsymbol{\mu})$. Since in the small strain regime $\boldsymbol{B}^{*T}(\boldsymbol{x}^g)\in\mathbb{R}^{p\times s}$ does not depend on the input parameters $\boldsymbol{\mu}$, rather than storing the internal forces at all Gauss points and for all configurations (which requires a matrix of size $m_{gs}\times P\cdot p$), it proves more efficient to solely store the stress solutions in a snapshot matrix $\mathbf{A}^{\boldsymbol{\sigma}}\in\mathbb{R}^{s\,m_{gs}\times P}$ (see Eq. 92). Then, by applying the truncated SVD (with a tolerance $\epsilon_\sigma\sim\epsilon_\lambda$), one can obtain a basis matrix for the stresses:

$$[\boldsymbol{\Lambda},\bullet,\bullet]=\texttt{SVD}(\mathbf{A}^{\boldsymbol{\sigma}},\epsilon_\sigma), \tag{128}$$

where $\boldsymbol{\Lambda}\in\mathbb{R}^{s\cdot m_{gs}\times r}$, r being the number of stress modes (which should be of the same order as the number of straining and self-equilibrating modes p). Having the stress basis matrix at one's disposal, the desired matrix of internal forces is determined as

$$\boldsymbol{A}^{\boldsymbol{f}}:=\begin{bmatrix}\boldsymbol{\mathcal{F}}_1^1 & \cdots & \boldsymbol{\mathcal{F}}_1^p & \boldsymbol{\mathcal{F}}_2^1 & \cdots & \boldsymbol{\mathcal{F}}_2^p & \cdots & \boldsymbol{\mathcal{F}}_2^1 & \cdots & \boldsymbol{\mathcal{F}}_r^p\end{bmatrix}_{m_{gs}\times p\cdot r} \tag{129}$$

where the expression for $\boldsymbol{\mathcal{F}}_j^I\in\mathbb{R}^{m_{gs}}$ reads

$$\boldsymbol{\mathcal{F}}_j^I:=\begin{bmatrix}\sqrt{W_1}\boldsymbol{B}_I^{*T}(\boldsymbol{x}_1)\boldsymbol{\Lambda}(\boldsymbol{b}_1,j)\\ \sqrt{W_2}\boldsymbol{B}_I^{*T}(\boldsymbol{x}_2)\boldsymbol{\Lambda}(\boldsymbol{b}_2,j)\\ \vdots\\ \sqrt{W_{m_{gs}}}\boldsymbol{B}_I^{*T}(\boldsymbol{x}_{m_{gs}})\boldsymbol{\Lambda}(\boldsymbol{b}_{m_{gs}},j)\end{bmatrix} \tag{130}$$

(here $W_g = W(\boldsymbol{x}_g)$ and $\boldsymbol{b}_g = \{s\ (g-1)+1, s\ (g-1)+2 \ldots s\, g\}$). Next we apply again the truncated SVD on $\boldsymbol{A^f}$ to eliminate redundancies and achieve an orthogonal basis matrix for the internal forces

$$[\boldsymbol{\Upsilon}, \bullet, \bullet] = \texttt{SVD}(\boldsymbol{A^f}, \epsilon_f). \tag{131}$$

Algorithm 7: Empirical Cubature Method (enhanced version of the algorithm in Ref. [22])

Function $[\boldsymbol{z}, \boldsymbol{\omega}]$ =ECM($\boldsymbol{G}$,$\boldsymbol{W}$,TOL)
- **Data:** $\boldsymbol{G} \in \mathbb{R}^{p\times M}$, where $\boldsymbol{G}\boldsymbol{G}^T = \boldsymbol{I}$; $\boldsymbol{W} \in \mathbb{R}^M$, $W_i > 0$; tolerance $0 \leq TOL \leq 1$
- **Result:** $\boldsymbol{z} \subset \{1, 2 \ldots M\}$, $\boldsymbol{\omega} > \boldsymbol{0} \in \mathbb{R}^m$, such that $\|\boldsymbol{G}(:, \boldsymbol{z})\boldsymbol{\alpha}\| \leq TOL \|\boldsymbol{G}\sqrt{\boldsymbol{W}}\|$, where $\omega_i = \sqrt{\boldsymbol{W}(\boldsymbol{z}_i)}\alpha_i$
- $\boldsymbol{z} \leftarrow \emptyset$; $\boldsymbol{y} \leftarrow \{1, 2 \ldots M\}$; $\boldsymbol{b} \leftarrow \boldsymbol{G}\sqrt{\boldsymbol{W}}$; $\boldsymbol{r} \leftarrow \boldsymbol{b}$; $\boldsymbol{\alpha} \leftarrow \emptyset$; $\boldsymbol{H} \leftarrow \emptyset$ // Initializations
- $\boldsymbol{y} \leftarrow \boldsymbol{y} \setminus \boldsymbol{h}$, where $\boldsymbol{h} = [h_1, h_2 ...]$ such that $\|\boldsymbol{G}(:, h_i)\| \leq \epsilon$ // Remove points whose associated values in $\boldsymbol{G}$ are lower than a given tolerance ($\epsilon \sim 10^{-6}$)
- **while** $\|\boldsymbol{r}\|/\|\boldsymbol{b}\| > TOL$ ***AND*** length($\boldsymbol{z}$)$< p$ ***AND*** length ($\boldsymbol{y}$)> 0 **do**
  - $i = \arg\max_{i\in\boldsymbol{y}} \boldsymbol{g}_{\boldsymbol{y}}^T \boldsymbol{r}$, where $\boldsymbol{g}_j = \boldsymbol{G}(:, j)/\|\boldsymbol{G}(:, j)\|$ // Select the column most ''positively'' parallel to the residual $\boldsymbol{r}$
  - **if** $\boldsymbol{z} = \emptyset$ **then**
    - $\boldsymbol{H} \leftarrow (\boldsymbol{G}(:, i)^T \boldsymbol{G}(:, i))^{-1}$ // Inverse Hermittian matrix (first iteration)
    - $\boldsymbol{\alpha} \leftarrow \boldsymbol{H}\boldsymbol{G}(:, i)^T \boldsymbol{b}$ // Weights computed through least-squares
  - **else**
    - $[\boldsymbol{\alpha}, \boldsymbol{H}] \leftarrow$ LSTONER($\boldsymbol{\alpha}, \boldsymbol{H}, \boldsymbol{G}(:, \boldsymbol{z}), \boldsymbol{G}(:, i), \boldsymbol{b}$) // Least-squares via one-rank update, see Algorithm 8.
  - **end**
  - $\boldsymbol{z} \leftarrow \boldsymbol{z} \cup i$; $\boldsymbol{y} \leftarrow \boldsymbol{y} \setminus i$; // Move index i from $\boldsymbol{y}$ to $\boldsymbol{z}$
  - $\boldsymbol{n} \leftarrow$ Indexes such that $\boldsymbol{\alpha}_{\boldsymbol{n}} < 0$ // Identify negative weights
  - **if** $\boldsymbol{n} \neq \emptyset$ **then**
    - $\boldsymbol{y} \leftarrow \boldsymbol{y} \cup \boldsymbol{z}(\boldsymbol{n})$; $\boldsymbol{z} \leftarrow \boldsymbol{z} \setminus \boldsymbol{z}(\boldsymbol{n})$; // Remove indexes negative weights
    - $\boldsymbol{H} \leftarrow$ UPHERM($\boldsymbol{H}, \boldsymbol{n}$) // Update inverse Hermitian Matrix (via recursive, one-rank operations, see Algorithm 9)
    - $\boldsymbol{\alpha} = \boldsymbol{H}\boldsymbol{G}(:, \boldsymbol{z})^T \boldsymbol{b}$ // Recalculate weights
  - **end**
  - $\boldsymbol{r} \leftarrow \boldsymbol{b} - \boldsymbol{G}(., \boldsymbol{z})\boldsymbol{\alpha}$ // Update the residual
- **end**
- $\omega_g \leftarrow \alpha_g \sqrt{W(z_g)}$, $g = 1, 2 \ldots$ length($\boldsymbol{z}$)

Algorithm 8: Least-squares via one-rank update

Function $[\boldsymbol{\alpha}_{new}, \boldsymbol{H}_{new}]$ =LSTONER($\boldsymbol{\alpha}, \boldsymbol{H}, \boldsymbol{A}, \boldsymbol{a}, \boldsymbol{b}$)
- **Data:** $\boldsymbol{A} \in \mathbb{R}^{p\times m}$, $\boldsymbol{a} \in \mathbb{R}^p$, $\boldsymbol{H} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}$, $\boldsymbol{b} \in \mathbb{R}^p$, $\boldsymbol{\alpha} = \boldsymbol{H}\boldsymbol{A}^T\boldsymbol{b}$,
- **Result:** $\boldsymbol{H}_{new} = (\boldsymbol{B}^T\boldsymbol{B})^{-1}$ and $\boldsymbol{\alpha}_{new} = \boldsymbol{H}_{new}\boldsymbol{B}^T\boldsymbol{b}$, where $\boldsymbol{B} = \begin{bmatrix}\boldsymbol{A} & \boldsymbol{a}\end{bmatrix}$, via one-rank update.
- $\boldsymbol{c} = \boldsymbol{A}^T\boldsymbol{a}$; $\boldsymbol{d} = \boldsymbol{H}\boldsymbol{c}$; $s = \|\boldsymbol{a}\|^2 - \boldsymbol{c}^T\boldsymbol{d}$
- $\boldsymbol{H}_{new} = \begin{bmatrix}\boldsymbol{H} + \frac{1}{s}\boldsymbol{d}\boldsymbol{d}^T & -\frac{1}{s}\boldsymbol{d} \\ -\frac{1}{s}\boldsymbol{d}^T & \frac{1}{s}\end{bmatrix}$
- $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{\alpha}$; $v = (\boldsymbol{a}^T\boldsymbol{r})/s$
- $\boldsymbol{\alpha}_{new} = \begin{bmatrix}\boldsymbol{\alpha} - v\boldsymbol{d} \\ v\end{bmatrix}$

Having at hand matrix $\boldsymbol{\Upsilon} \in \mathbb{R}^{m_{gs}\times q}$, the error incurred in evaluating the internal forces for any training configuration using the subset of integration points with indexes $\mathbf{Z} \subset \{1, 2 \ldots M\}$) and associated positive weights $\boldsymbol{\alpha} \in \mathbb{R}^{+m^*_{gs}}$ is given

Algorithm 9: Update of the inverse of an Hermitian matrix when several columns are removed

1 **Function** $[\boldsymbol{H}_{new}]$ =UPHERM$(\boldsymbol{H}, \boldsymbol{n})$
  Data: $\boldsymbol{H} \in \mathbb{R}^{m\times m}$, where $\boldsymbol{H}$ is of the form $\boldsymbol{H} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}$, $\boldsymbol{A} \in \mathbb{R}^{p\times m}$; $\quad \boldsymbol{n} \subset \{1, 2 \dots m\}$
  Result: $\boldsymbol{H}_{new} = (\boldsymbol{B}^T\boldsymbol{B})^{-1}$, where $\boldsymbol{B} = \boldsymbol{A}(:, \boldsymbol{c})$, $\boldsymbol{c} = \{1, 2 \dots m\} \setminus \boldsymbol{n}$
2 $\boldsymbol{n} \leftarrow$ sort$(\boldsymbol{n})$ // Sort the indexes in ascending order
3 $\boldsymbol{H}_{new} \leftarrow \boldsymbol{H}$
4 **for** $i = 1$ *to* length$(\boldsymbol{n})$ **do**
5 $j = \boldsymbol{n}(i) - i + 1$
6 $\boldsymbol{H}_{new} \leftarrow$ UPHERMone$(\boldsymbol{H}_{new}, j)$ // Hermitian matrix inverse when 1 column is removed (Alg. 10)
7 **end**

Algorithm 10: Update of the inverse of an Hermitian matrix when one column is removed

1 **Function** $[\boldsymbol{C}_{new}]$ =UPHERMone$(\boldsymbol{C}, j)$ // From https://emtiyaz.github.io/Writings/OneColInv.pdf
  Data: $\boldsymbol{C} \in \mathbb{R}^{m\times m}$, where $\boldsymbol{C}$ is of the form $\boldsymbol{C} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}$, $\boldsymbol{A} \in \mathbb{R}^{p\times m}$; $\quad j \in \{1, 2 \dots m\}$
  Result: $\boldsymbol{C}_{new} = (\boldsymbol{B}^T\boldsymbol{B})^{-1}$, where $\boldsymbol{B} = \boldsymbol{A}(:, \boldsymbol{c})$, $\boldsymbol{c} = \{1, 2 \dots m\} \setminus j$
2 $\boldsymbol{D} \leftarrow \boldsymbol{C}$; $\quad \boldsymbol{r} \leftarrow \{1, 2 \dots m-1\}$
3 **if** $j < m$ **then**
4 $\boldsymbol{a} \leftarrow \{1, 2 \dots j-1\}$; $\quad \boldsymbol{b} \leftarrow \{j+1 \dots m\}$
5 $\boldsymbol{D} \leftarrow \big[\boldsymbol{C}(:, \boldsymbol{a}) \quad \boldsymbol{C}(:, \boldsymbol{b}) \quad \boldsymbol{C}(:, j)\big]$; $\quad \boldsymbol{D} \leftarrow \big[\boldsymbol{D}(\boldsymbol{a}, :) \quad \boldsymbol{D}(\boldsymbol{b}, :) \quad \boldsymbol{D}(j, :)\big]$
6 **end**
7 $\boldsymbol{C}_{new} = \boldsymbol{D}(\boldsymbol{r}, \boldsymbol{r}) - \dfrac{\boldsymbol{D}(\boldsymbol{r}, m)\boldsymbol{D}(m, \boldsymbol{r})}{\boldsymbol{D}(m, m)}$

by[51]

$$e(\boldsymbol{\alpha}, \mathbf{Z}) = \|\boldsymbol{\Upsilon}^T\sqrt{\boldsymbol{W}} - \boldsymbol{\Upsilon}^T(:, \mathbf{Z})\boldsymbol{\alpha}\|. \tag{132}$$

Given a tolerance $0 \leq \epsilon_{ecm} \leq 1$, we seek the smallest set of points $\mathbf{Z}$ and associated positive weights $\boldsymbol{\alpha}$ such that

$$\|\boldsymbol{G}\sqrt{\boldsymbol{W}} - \boldsymbol{G}(:, \mathbf{Z})\boldsymbol{\alpha}\| \leq \epsilon_{ecm}\|\boldsymbol{G}\sqrt{\boldsymbol{W}}\| \tag{133}$$

where $\boldsymbol{G} = \boldsymbol{\Upsilon}^T$ ($\boldsymbol{G}\sqrt{\boldsymbol{W}}$ and $\boldsymbol{G}(:, \mathbf{Z})\boldsymbol{\alpha}$ represent the vectors of exact and approximate integrals, respectively). To solve this minimization problem, we use the function:

$$[\boldsymbol{Z}, \boldsymbol{\omega}] = \text{ECM}(\boldsymbol{G}, \boldsymbol{W}, \epsilon_{ecm}) \tag{134}$$

described in Algorithm 7. As pointed out previously, this (greedy) method is an algorithmically improved version of the one put forward by the author in Ref [22]. The improvement goes as follows: at each iteration, the algorithm has to cope with a least-squares problem to obtain the corresponding weights: $\boldsymbol{\alpha} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b}$, where $\boldsymbol{A} = [\boldsymbol{G}(:, \boldsymbol{z}), \boldsymbol{G}(:, i)]$ is the matrix of coefficients formed by incorporating the incoming column $\boldsymbol{G}(:, i)$. In principle, this takes $\mathcal{O}(qm^2)$ floating-points operations (FLOPs), which implies that, at each iteration, the cost increases by a factor $\mathcal{O}(m^2)$. This constitutes a serious bottleneck for high q, specially when m approaches q (the exact solution, for $\epsilon_{ecm} \approx 0$, is achieved for $m = q$, that is, when $\boldsymbol{A}$ becomes square). To amend this, we have modified the original algorithm in Ref. [22]. Now the the least-squares problem is solved using function LSTONER, described in Alg. 8. This function performs a rank-one update of $(\boldsymbol{A}^T\boldsymbol{A})^{-1}$ using the formula[52] for the inverse of a symmetric 2-by-2 block matrix (cf. [3, 20]), which requires only $\mathcal{O}(qm)$ FLOPs. The same strategy is used for updating the inverse of the Hermitian matrix upon removal of the columns with associated negative weights, see line 16 of Algorithm 7.

[51] If $\sqrt{\boldsymbol{W}}^T\boldsymbol{\Upsilon} = \mathbf{0}$, the problem becomes ill-posed (it admits the trivial solution $\boldsymbol{\alpha} = \mathbf{0}$). To eliminate the issue, it suffices to *expand* $\boldsymbol{\Upsilon}$ as $\boldsymbol{\Upsilon} \leftarrow [\boldsymbol{\Upsilon}, \sqrt{\boldsymbol{W}}]$ (consult Ref. [22] for further details).

[52] Alternatively, when the least-squares problem is solved using the QR decomposition, one can update the QR decomposition of the coefficient matrix. This is the procedure followed by Lawson and Hanson in their pioneering Nonnegative Least-Squares (NNLS) algorithm [27]. In this respect, Chapmat et al. [6] has recently proposed a parallel updatable QR factorizer, which has been used in the context of the energy-conserving sampling and weighting (ECSW) procedure [12, 13].

B. Coarse-scale body forces

The goal of this Section is to show how the expression for the coarse-scale body forces $\boldsymbol{\mathcal{F}}_{\boldsymbol{b}}^e$:

$$\boldsymbol{\mathcal{F}}_{\boldsymbol{b}}^e = \boldsymbol{\mathcal{F}}_{\boldsymbol{se}}^e + \boldsymbol{\mathcal{F}}_{\boldsymbol{eq}}^e \tag{135}$$

(see Eq. 72) can be cast in a reduced-order format, without the need of having at one's disposal the exact nodal expression for the FE nodal external forces $\boldsymbol{F}_{\boldsymbol{ext}}^e$ (which, recall, include FE nodal body forces $\boldsymbol{F}_{\boldsymbol{b}}^e$ and non-interface tractions $\boldsymbol{F}_{\boldsymbol{tr}}^e$, see Eq. 11).

B.1. Body forces

The vector of FE nodal body forces may be expressed as

$$\boldsymbol{F}_{\boldsymbol{b}}^e = \sum_{g=1}^{m_{gs}} \boldsymbol{N}^T(\boldsymbol{x}_g) W(\boldsymbol{x}_g) \boldsymbol{b}^e(\boldsymbol{x}_g), \tag{136}$$

where[53] $\boldsymbol{N}(\boldsymbol{x}_g)$ is the shape function (in its global, sparse format) associated to the Gauss point located at $\boldsymbol{x}_g$ and $\boldsymbol{b}^e(\boldsymbol{x}_g)$ the corresponding body force per unit volume. If self-weight (gravity) is the only body force, then we may write Eq.(136) as

$$\boldsymbol{F}_{\boldsymbol{b}}^e = \sum_{k=1}^{n_{mat}} \boldsymbol{J}_{\boldsymbol{b}}^k \rho^k \hat{\boldsymbol{Q}}^{e^T} \boldsymbol{g}, \tag{137}$$

where

$$\boldsymbol{J}_{\boldsymbol{b}}^k := \sum_{g=1}^{m_{gs}(k)} \boldsymbol{N}^T(\boldsymbol{x}_{g(k)}) W(\boldsymbol{x}_{g(k)}). \tag{138}$$

Here n_{mat} is the number of distinct materials, ρ_k designates the density of the $k-th$ material and $\boldsymbol{g} \in \mathbb{R}^{n_{sd}}$ is the vector of acceleration of gravity (in global coordinates).

B.2. Non-interface traction forces

As for non-interface tractions, suppose that the non-interface boundary $\partial\bar{\Omega}_{non}$ is subdivided into b_{non} portions. In such a case, the expression for the FE nodal contribution can be written as

$$\boldsymbol{F}_{\boldsymbol{tr}}^e = \sum_{k=1}^{b_{non}} \sum_{g=1}^{\bar{m}_{gs}(k)} \bar{\boldsymbol{N}}^T(\bar{\boldsymbol{x}}_{g(k)}) \bar{W}(\bar{\boldsymbol{x}}_{g(k)}) \boldsymbol{t}^e(\bar{\boldsymbol{x}}_{g(k)}), \tag{139}$$

where $\bar{\boldsymbol{N}}^T(\bar{\boldsymbol{x}}_{g(k)})$ stands for the matrix of boundary shape functions at the (boundary) integration point $\bar{\boldsymbol{x}}_{g(k)}$, $\bar{W}(\bar{\boldsymbol{x}}_{g(k)})$ is the corresponding volumetric weight and $\boldsymbol{t}^e(\bar{\boldsymbol{x}}_{g(k)}) \in \mathbb{R}^{n_{sd}}$ the traction vector in the domain reference axes. In the case of curved boundaries, the traction vector is often provided in the coordinate axes intrinsic to the surface: $\boldsymbol{t}^e(\bar{\boldsymbol{x}}_{g(k)}) = \boldsymbol{Q'}(\bar{\boldsymbol{x}}_{g(k)}) \boldsymbol{t'}^e(\bar{\boldsymbol{x}}_{g(k)})$ (here $\boldsymbol{Q'}(\bar{\boldsymbol{x}}_{g(k)})$ is the local rotation matrix[54] at the boundary integration point $\bar{\boldsymbol{x}}_{g(k)}$), and Eq.(139) becomes expressible as

$$\boldsymbol{F}_{\boldsymbol{tr}}^e = \sum_{k=1}^{b_{non}} \boldsymbol{J}_{\boldsymbol{tr}}^k \boldsymbol{t'}^{e(k)}. \tag{140}$$

where

$$\boldsymbol{J}_{\boldsymbol{tr}}^k := \sum_{g=1}^{\bar{m}_{gs}(k)} \bar{\boldsymbol{N}}^T(\bar{\boldsymbol{x}}_{g(k)}) \bar{W}(\bar{\boldsymbol{x}}_{g(k)}) \boldsymbol{Q'}(\bar{\boldsymbol{x}}_{g(k)}) \tag{141}$$

In writing Eq.(140) we have assumed, without loss of generality[55], that the traction vector $\boldsymbol{t'}^{e(k)}$ is uniform over the non-interface boundary $\partial\bar{\Omega}_{non}^k$.

[53] For simplicity, we are assuming that the integration of the body forces are carried out using the same Gauss points as the internal forces

[54] The first vector of the triad of unit vectors is assumed to be, as customary, the one normal to the surface (and pointing outwards).

[55] One may take as partition of the non-interface boundary the finite element partition itself.