High Order Numerical Quadrature Functions for XFEM

Linbo ZHANG

State Key Laboratory of Scientific and Engineering Computing,
Academy of Mathematics and Systems Science, Chinese Academy of Sciences.

September 23, 2022

Abstract

This document presents functions available in the PHG toolbox for numerical quadrature in subdomains of an element cut by an implicit curved interface. Currently four element types, including triangle, rectangle, tetrahedron and cuboid, are supported.

Contents

1 Introduction

Let E be an element, which can be a triangle, rectangle, tetrahedron or cuboid, and $L(\mathbf{x})$ a smooth level set function. PHG provides C functions for computing highly accurate approximations of the following integrals using the algorithm presented in [1] for triangles and tetrahedra, and a modified version of the algorithm proposed in [2] for rectangles and cuboids:

$$I^- = \int_{E\cap\Omega^-} f(\mathbf{x})\,\mathrm{d}\mathbf{x}, \quad I^+ = \int_{E\cap\Omega^+} f(\mathbf{x})\,\mathrm{d}\mathbf{x}, \quad I^0 = \int_{E\cap\Gamma} f(\mathbf{x})\,\mathrm{d}\Gamma, \tag{1}$$

where $\Omega^- := \{\mathbf{x} \mid L(\mathbf{x}) < 0\}$, $\Omega^+ := \{\mathbf{x} \mid L(\mathbf{x}) > 0\}$ and $\Gamma := \{\mathbf{x} \mid L(\mathbf{x}) = 0\}$. The level set function $L(\mathbf{x})$ and integrand $f(\mathbf{x})$ are assumed smooth in E.

These functions are implemented in the following files:

```
include/phg/quad-interface.h
include/phg/quad-cuboid.hpp
src/quad-interface.c
src/quad-interface-triangle.c
src/quad-interface-cuboid.cxx
```

They can be used in implementations of extended finite element methods (XFEM) or other high order methods for solving PDEs on unfitted meshes. Note that two functions, namely `phgQuadInterface` and `phgQuadInterfaceMarkGrid`, are intended for PHG users only, while all other functions are for general users and can be used in either PHG or non PHG programs.

2 Functions for general users

2.1 The function `phgQuadInterfaceTetrahedron`

This function is used to compute quadrature rules for the integrals listed in (1), where E is a tetrahedron.

An example code on using this function is provided in the program `test/quad_test3.c`.

2.1.1 Prototype

```
void phgQuadInterfaceTetrahedron(FUNC_3D ls, int ls_order, FUNC_3D ls_grad,
                FLOAT const tet[4][3], int quad_order,
                FLOAT **rule_m, FLOAT **rule_0, FLOAT **rule_p);
```

2.1.2 Arguments

- The level set function $L(\mathbf{x})$ and its gradient $\nabla L(\mathbf{x})$ are specified by the arguments `ls`, `ls_order` and `ls_grad`.
  - `ls` and `ls_grad` are pointers to functions of type `FUNC_3D`, which compute respectively the value of the level set function $L(\mathbf{x})$ and its gradient $\nabla L(\mathbf{x})$ at a given point.

    The `FUNC_3D` type is defined as follows:

    ```
    typedef void (*FUNC_2D)(FLOAT x, FLOAT y, FLOAT z, FLOAT *res);
    ```

    It should compute the value(s) of the function at the given point, and return the result(s) in `res[]`.
  - `ls_order`, if nonnegative, specifies the polynomial order of $L(\mathbf{x})$, which is used by the roots finding algorithm. If `ls_order` < 0, then $L(\mathbf{x})$ is non polynomial, in this case the roots finding algorithm will first use an interpolating polynomial of order $|$`ls_order`$|$ to get initial guesses of the roots, and then apply Newton iterations or bisections to refine the roots.
- The argument `tet` gives the coordinates of the four vertices of the tetrahedron.
- The order of the 1D Gaussian quadrature rule used in the function is specified by the argument `quad_order`. Since Gaussian rules have odd orders, the Gaussian rule of order `quad_order` $+ 1$ will be used if `quad_order` is even. In some cases, for examples, if the interface is planar, or `quad_order` ≤ 1, or the element is very small, the integral may be approximated using a planar approximation of the interface, and in this case quadrature rules for triangles and tetrahedra of order `quad_order` presented in [3] are used.
- The computed quadrature rules for I^-, I^0 and I^+ are returned by the arguments `*rule_m`, `*rule_0` and `*rule_p`, respectively. They will be set to point to the corresponding quadrature rule. One (or more) of `rule_m`, `rule_0` and `rule_p` can be `NULL`, then the corresponding quadrature rule(s) are not computed. Buffers for the computed quadrature rules are dynamically allocated and are to be freed by the calling function when no longer needed.

  The data format for a quadrature rule is an array of `FLOAT`s consisting of a header followed by a list of quadrature points and weights, and optionally a list of unit normal vectors of the interface at the quadrature points (for I^0 only). It is ensured that in the computed quadrature rules, the points are strictly inside the integration domain and the weights are all positive. The functions phgQuadInterfaceRuleInfo or phgQuadInterfaceRuleApply can be used to retrieve data from a quadrature rule, or apply it to specific integrands.

For convenience, and as a special convention, the argument ls can be a NULL pointer, in this case a quadrature rule for the integral over the whole element will be computed and returned in rule_m, or rule_p if rule_m == NULL.

2.2 The function phgQuadInterfaceTriangle

This function is used to compute quadrature rules for the integrals listed in (1), where E is a triangle.

A test program, test/quad_test3-triangle.c, is available which can serve as a sample code on using this function.

2.2.1 Prototype

```
void phgQuadInterfaceTriangle(FUNC_2D ls, int ls_order, FUNC_2D ls_grad,
                FLOAT const triangle[3][2], int quad_order,
                FLOAT **rule_m, FLOAT **rule_0, FLOAT **rle_p);
```

2.2.2 Arguments

- ls and ls_grad are pointers to functions of type FUNC_2D, which compute respectively the value of the level set function $L(\mathbf{x})$ and its gradient $\nabla L(\mathbf{x})$ at a given point.

  The FUNC_2D type is defined as follows:

  ```
  typedef void (*FUNC_2D)(FLOAT x, FLOAT y, FLOAT *res);
  ```

  It should compute the value(s) of the function at the given point, and return the result(s) in res[].

- The element E is specified by the argument triangle, which gives the coordinates of the three vertices of the triangle.

- See the function phgQuadInterfaceTetrahedron for the meanings of the other arguments.

2.3 The function phgQuadInterfaceCuboid

This function is used to compute the integrals listed in (1), where E is a cuboid.

A test program, test/quad_test3-cuboid.c, is available which can serve as a sample code on using this function.

2.3.1 Prototype

```
void phgQuadInterfaceCuboid(FUNC_3D ls, int ls_order, FUNC_3D ls_grad,
                FLOAT const cuboid[2][3], int quad_order,
                FLOAT **rule_m, FLOAT **rule_0, FLOAT **rule_p);
```

2.3.2 Arguments

- The element E is specified by the argument cuboid, which gives the coordinates of lower-left-front and upper-right-back corners of the cuboid.

- See the function phgQuadInterfaceTetrahedron for the meanings of the other arguments.

2.4 The function phgQuadInterfaceRectangle

This function is used to compute quadrature rules for the integrals listed in (1), where E is a rectangle.

A test program, test/quad_test3-cuboid.c, is available which can serve as a sample code on using this function.

2.4.1 Prototype

```
void phgQuadInterfaceRectangle(FUNC_2D ls, int ls_order, FUNC_2D ls_grad,
                FLOAT const rectangle[2][2], int quad_order,
                FLOAT **rule_m, FLOAT **rule_0, FLOAT **rule_p);
```

2.4.2 Arguments

- The element E is specified by the argument `rectangle`, which gives the coordinates of lower-left and upper-right corners of the rectangle.
- See the function `phgQuadInterfaceTetrahedron` for the meanings of the other arguments.

2.5 The function `phgQuadInterfaceTetrahedronFace`

This function is used to compute quadrature rules for the integrals listed in (1), where E is a face of a tetrahedron, with the 3D level set function projected to the face. The actual computations in this function are done by calling `phgQuadInterfaceTriangle`.

2.5.1 Prototype

```
void phgQuadInterfaceTetrahedronFace(FUNC_3D ls, int ls_order, FUNC_3D ls_grad,
                FLOAT const tet[4][3], int face, int quad_order,
                FLOAT **rule_m, FLOAT **rule_0, FLOAT **rule_p);
```

2.5.2 Arguments

- The argument `face` ($\in \{0,1,2,3\}$) specifies the face number. The face is opposite to the vertex with the number `face`. The the three vertices of the face are given by:

$$\left\{\texttt{tet[}i\texttt{][]} \mid i \in \{0,1,2,3\} \text{ and } i \neq \texttt{face}\right\}$$

- See `phgQuadInterfaceTetrahedron` and `phgQuadInterfaceTriangle` for the meanings of the other arguments.

For convenience, and as a special convention, the argument `ls` can be a `NULL` pointer, in this case a quadrature rule for the integral over the whole face will be computed and returned in `rule_m`, or `rule_p` if `rule_m` == `NULL`.

2.6 The function `phgQuadInterfaceCuboidFace`

This function is used to compute quadrature rules for the integrals listed in (1), where E is a face of a cubid, with the 3D level set function projected to the face. The actual computations in this function are done by calling `phgQuadInterfaceRectangle`.

2.6.1 Prototype

```
void phgQuadInterfaceCuboidFace(FUNC_3D ls, int ls_order, FUNC_3D ls_grad,
                FLOAT const cuboid[2][3], int face, int quad_order,
                FLOAT **rule_m, FLOAT **rule_0, FLOAT **rule_p);
```

2.6.2 Arguments

- The argument `face` ($\in \{0,1,2,3,4,5\}$) specifies the face. Here the six faces of a cuboid is numbered as $x-$, $x+$, $y-$, $y+$, $z-$ and $z+$.
- See `phgQuadInterfaceCuboid` and `phgQuadInterfaceRectangle` for the meanings of the other arguments.

For convenience, and as a special convention, the argument `ls` can be a `NULL` pointer, in this case a quadrature rule for the integral over the whole face will be computed and returned in `rule_m`, or `rule_p` if `rule_m` == `NULL`.

2.7 The function `phgQuadInterfaceRuleCreate`

Auxiliary function. It is used to create a rule with the specified points, weights, and optionally normal vectors.

2.7.1 Prototype

```
FLOAT *phgQuadInterfaceRuleCreate(int dim, int np, const FLOAT *pts, int pinc,
                const FLOAT *wgts, int winc, FLOAT scale,
const FLOAT *nv, int ninc);
```

The return value of the function is the pointer to the quadrature rule, with the memory space for the rule dynamically allocated which should be freed by the calling function.

2.7.2 Arguments

- The argument `dim` specifies the space dimension of the quadtrture rule (2 or 3).
- The argument `np` specifies the number of points in the quadtrture rule.
- The arguments `pts` and `pinc` specify the list of points $\{\mathbf{x}_i \mid i = 0, \dots, \mathtt{np} - 1\}$ of the quadrature rule, with:
$$\mathbf{x}_i = (x_i, y_i, z_i) = (\mathtt{pts[}i\mathtt{*pinc]}, \mathtt{pts[}i\mathtt{*pinc+1]}, \mathtt{pts[}i\mathtt{*pinc+2]}),$$
for $\mathtt{dim} = 3$, and:
$$\mathbf{x}_i = (x_i, y_i) = (\mathtt{pts[}i\mathtt{*pinc]}, \mathtt{pts[}i\mathtt{*pinc+1]}),$$
for $\mathtt{dim} = 2$.
- The arguments `wgts`, `winc` and `scale` specify the list of weights $\{w_i \mid i = 0, \dots, \mathtt{np} - 1\}$ of the quadrature rule, with:
$$w_i = \mathtt{wgts[}i\mathtt{*winc]\ *\ scale}.$$
- The arguments `nv` and `ninc`, if $\mathtt{nv} \neq \mathtt{NULL}$, specify the list of unit normal vectors of the interface at the quadrature points $\{\mathbf{n}_i \mid i = 0, \dots, \mathtt{np} - 1\}$, with:
$$\mathbf{n}_i = (\mathbf{n}_{i,x}, \mathbf{n}_{i,y}, \mathbf{n}_{i,z}) = (\mathtt{nv[}i\mathtt{*ninc]}, \mathtt{nv[}i\mathtt{*ninc+1]}, \mathtt{nv[}i\mathtt{*ninc+2]}),$$
for $\mathtt{dim} = 3$, and:
$$\mathbf{n}_i = (\mathbf{n}_{i,x}, \mathbf{n}_{i,y}) = (\mathtt{nv[}i\mathtt{*ninc]}, \mathtt{nv[}i\mathtt{*ninc+1]}),$$
for $\mathtt{dim} = 2$.

2.8 The function `phgQuadInterfaceRuleApply`

This function computes the integral of a given integrand using a quadrature rule obtained by the numerical quadrature functions described in the previous subsections. The integral to compute has the following form:
$$\mathtt{result} := \int_D \mathcal{P}\big(u(\mathbf{x})\big),$$
where D denotes the integration domain matching the quadrature rule, which can be either $E \cap \Omega^-$ (I^-), $E \cap \Omega^+$ (I^+) or $E \cap \Gamma$ (I^0), and $\mathcal{P}$ denotes a projection operator with respect to the unit normal vector of Γ (for I^0 only).

2.8.1 Prototype

```
int phgQuadInterfaceRuleApply(void *func, int dim, PROJ proj,
                              FLOAT *rule, FLOAT *res);
```

The return value of the function is the number of points in the quadrature rule.

2.8.2 Arguments

- The integrand $\mathcal{P}\big(u(\mathbf{x})\big)$ is specified by the arguments `func`, `dim` and `proj`.
  - `func` points to the function for evaluating $u(\mathbf{x})$. The pointer sould be of type `FUNC_2D` for 2D (triangle and rectangle) elements, and of type `FUNC_3D` for 3D (tetrahedron and cuboid) elements.
  - `dim` $\equiv \mathbf{dim}\Big(\mathcal{P}\big(u(\mathbf{x})\big)\Big)$, here $\mathbf{dim}(f)$ denotes the dimension, or number of components, of a function f.
  - `proj` specifies the projection with respect to the unit normal vector of Γ. Valid values for `proj` are given in the table below:

`proj`	$\mathcal{P}\big(u(\mathbf{x})\big)$	$\mathbf{dim}\big(u(\mathbf{x})\big)$
`PROJ_NONE`	$u(\mathbf{x})$	`dim`
`PROJ_DOT`	$u(\mathbf{x}) \cdot \vec{\mathbf{n}}$	`dim` $\times d$
`PROJ_CROSS`	$u(\mathbf{x}) \times \vec{\mathbf{n}}$	`dim` $\times c$

  where $\vec{\mathbf{n}}$ denotes the unit normal vector of Γ, d denotes the space dimension (2 or 3), and $c = d$ (for $d = 2$) or 1 (for $d = 3$). Note that the values of `proj` other than `PROJ_NONE` can only be used with I^0.

- The argument `rule` points to the quadrature rule, computed by one of the functions presented in the previous subsections.
- The computed integral is returned in `res`, which points to a buffer of FLOATs of size `dim`.

2.9 The function `phgQuadInterfaceRuleInfo`

This function is used to retrieve information and data in a quadrature rule.

2.9.1 Prototype

```
int phgQuadInterfaceRuleInfo(FLOAT *rule, int *dim, FLOAT **pw, FLOAT **nv);
```

The return value of the function is the number of points in the quadrature rule.

2.9.2 Arguments

- The argument `rule` points to the quadrature rule.
- The argument `*dim`, if `dim` $\neq$ `NULL`, will be set to the space dimension (2 or 3) of the rule.
- The argument `*pw`, if `pw` $\neq$ `NULL`, will be set to point to the actual quadrature rule data, which is an array of `FLOAT`s consisting of $\{\text{point}, \text{weight}\}$ pairs. Let n be the number of points in the rule (the return value of this function), then the size of the array is $n \times (\texttt{dim} + 1)$, and we have:

  $$\texttt{*pw} = \{x_1, y_1, w_1, \ldots, x_n, y_n, w_n\}$$

  for `dim` $= 2$, and:

  $$\texttt{*pw} = \{x_1, y_1, z_1, w_1, \ldots, x_n, y_n, z_n, w_n\}$$

  for `dim` $= 3$.

- The argument `*nv`, if `nv` $\neq$ `NULL`, will be set to point to the list of unit normal vectors of the interface at the quadrature points, in the format:

  $$\texttt{*nv} = \{\vec{\mathbf{n}}_{1,x}, \vec{\mathbf{n}}_{1,y}, \ldots, \vec{\mathbf{n}}_{n,x}, \vec{\mathbf{n}}_{n,y}\},$$

  for 2D elements (`dim` $= 2$), and:

  $$\texttt{*nv} = \{\vec{\mathbf{n}}_{1,x}, \vec{\mathbf{n}}_{1,y}, \vec{\mathbf{n}}_{1,z}, \ldots, \vec{\mathbf{n}}_{n,x}, \vec{\mathbf{n}}_{n,y}, \vec{\mathbf{n}}_{n,z}\}.$$

  for 3D elements (`dim` $= 3$). If the unit normal vectors for the quadrature rule are not available (e.g., the integration domain is not on the interface, or the option "`+qi_nv_flag`" was used when computing the quadrature rule), then `*nv` will be set to `NULL`. Note that storing normal vectors of Γ in the quadrature rule can be disabled by the command-line option "`+qi_nv_flag`" if the normal vectors are not needed.

2.10 The function `phgQuadInterfaceMarkElement`

This function returns an integer indicating the relative position of the given element with respect to the interface: -1 if the element is entirely contained in Ω^-, 1 if the element is entirely contained in Ω^+, and 0 if the element *might* intersect the interface.

Note that this function requires that the element is not too big w.r.t. the interface. It may return wrong results if, for example, the interface is entirely contained in the element.

2.10.1 Prototype

```
int phgQuadInterfaceMarkElement(void *ls, int ls_order, void *ls_grad,
                                ELEMENT_TYPE elem_type, void *E);
```

2.10.2 Arguments

- The arguments `ls`, `ls_order` and `ls_grad` specify the level set function and its gradient. `ls` and `ls_grad` are of type `FUNC_2D` for 2D elements, and of type `FUNC_3D` for 3D elements, please see functions `phgQuadInterfaceTetrahedron` and `phgQuadInterfaceTriangle` for information about these types.
- `elem_type` specifies the type of the element. Available types are: `ET_TRIANGLE`, `ET_RECTANGLE`, `ET_TETRAHEDRON` and `ET_CUBOID`.
- `E` points to the element, in the same format as in the corresponding numerical quadrature functions, in the form "`FLOAT const E[`v`][`d`]`", where d is the space dimension (2 or 3) and v is the number of vertices in the element (3 for triangle, 4 for rectangle and tetrahedron, and 8 for cuboid).

3 Functions for PHG users

The functions in this section are intended for PHG users only, in which the element is specified using PHG's `ELEMENT` type, and the level set function and its gradient are specified using PHG's `DOF` type.

3.1 The function `phgQuadInterface`

This function is used to compute quadrature rules for the integrals listed in (1), where E is a tetrahedral element.

A test program, `test/quad_test2.c`, is available which can serve as a sample code on using this function.

3.1.1 Prototype

```
void phgQuadInterface(DOF *ls, DOF *ls_grad, ELEMENT *e, int quad_order,
                      FLOAT **rule_m, FLOAT **rules_0, FLOAT **rule_p);
```

3.1.2 Arguments

- The level set function $L(\mathbf{x})$ and its gradient $\nabla L(\mathbf{x})$ are specified by the arguments `ls` and `ls_grad`. `ls` can represent a finite element function (usually a piecewise polynomial) or an analytic function (with the type `DOF_ANALYTIC`). `ls_grad` is optional and can be set to `NULL` if `ls` is an ordinary finite element function, since in this case $\nabla L(\mathbf{x})$ can be computed using `ls`.
- The element E is specified by the argument `e`.
- See the function `phgQuadInterfaceTetrahedron` for the meanings of the other arguments.

For convenience, and as a special convention, the argument `ls` can be set to `NULL` and then a quadrature rule for the integral over the whole element will be computed and returned in `rule_m`, or `rule_p` if `rule_m` is `NULL`. In this case `ls_grad` must not be `NULL`, and can point to any `DOF` such that `ls_grad->g` is valid.

3.2 The function phgQuadInterfaceFace

This function is used to compute quadrature rules for the integrals listed in (1), where E is a face of a tetrahedron, with the 3D level set function projected to the face. The actual computations in this function are done by calling `phgQuadInterfaceTriangle`.

3.2.1 Prototype

```
void phgQuadInterfaceTetrahedronFace(DOF *ls, DOF *ls_grad,
                ELEMENT *e, int face, int quad_order,
                FLOAT **rule_m, FLOAT **rule_0, FLOAT **rule_p);
```

3.2.2 Arguments

- The argument `face` ($\in \{0, 1, 2, 3\}$) specifies the face number. The face is opposite to the vertex with the number `face`.
- See `phgQuadInterface` and `phgQuadInterfaceTriangle` for the meanings of the other arguments.

For convenience, and as a special convention, the argument `ls` can be a `NULL` pointer, in this case a quadrature rule for the integral over the whole face will be computed and returned in `rule_m`, or `rule_p` if `rule_m` == `NULL`.

3.3 The function phgQuadInterfaceMarkGrid

This function sets the "`mark`" member of all elements in the mesh `ls->g` to 0 if the element might intersect the interface, -1 if the element is entirely contained in Ω^-, and 1 if the element is entirely contained in Ω^+. It helps to determine for which elements the functions presented in this document, which are much more expensive then ordinary numerical quadrature functions, need to be used.

Note that this function requires that the interface elements are relatively small w.r.t. the interface. It may produce wrong results otherwise.

3.3.1 Prototype

```
INT phgQuadInterfaceMarkGrid(DOF *ls);
```

The return value of the function is the number of elements marked with 0.

3.3.2 Arguments

- The level set function $L(\mathbf{x})$ is given by the argument `ls`.

3.4 The functions phgQuadGetRule*

These functions construct a numerical quadrature rule, which can be used by the `phgQCSetRule` function (see `doc/quad-cache.pdf`), for numerical integration in a whole element (`phgQuadGetRule3D`), a whole element face (`phgQuadGetRule2D`) or a whole element edge (`phgQuadGetRule1D`).

Note for all these functions, the returned rule is always in 3D and in physical coordinates.

3.4.1 Prototype

```
FLOAT *phgQuadGetRule3D(GRID *g, ELEMENT *e, int order);
FLOAT *phgQuadGetRule2D(GRID *g, ELEMENT *e, int face_no, int order);
FLOAT *phgQuadGetRule1D(GRID *g, ELEMENT *e, int edge_no, int order);
```

The return value of the functions is the pointer to the rule, in the same format as the rules generated by `phgQuadInterface*` functions.

3.4.2 Arguments

- The element is given by the argument `e`.
- The order of the quarature rule is given by the argument `order`.
- The arguments `face_no` and `edge_no` give respectively the face and edge number.

4 Command-line options

A set of command-line options, in the category "`quad_interface`", are available for setting internal parameters or debugging the algorithms. They are prefixed with "`-qi_`", and can be used either in the command-line, or at run-time by using the `phgOptionsSet*` functions in the program, for example:

```
phgOptionsPush();         /* save current settings */
phgOptionsSetOptions("-qi_threshold=0.9 -qi_subdiv_type=regular");
phgQuadInterfaceTetrahedron(... ...);
phgOptionsPop();          /* restore saved settings */
```

The default values for these options have been fine-tuned, the underlying algorithms are sensitive to some of them, and improper values may lead to poor precision or even program failure. So please don't change them unless you know what you are doing.

The full list of options available in this category can be obtained by running any program linked to the PHG library, and in which `phgInit` is called, with the command-line option "`-help quad_interface`".

Note that for the command-line options to be effective, the `phgInit` function must be called by the user program.

All (or most) options in this category are effective for the tetrahedron element type. For other element types, only a subset of them are effective. Below is an incomplete list of the options which are effective for the triangle, rectangle and cuboid element types:

```
-qi_eps, -qi_threshold, -qi_nv_flag, -qi_subdiv_limit,
-qi_newton_maxits, -qi_newton_porder, -qi_newton,
-qi_dbg_elem, -qi_show_recursions, -qi_show_directions, -qi_dbg_vtk
```

5 Acknowledgements

The code for the rectangle and cuboid element types was originally written by WEN Yundi (温韵迪).

The code for the triangle element type was originally written by LIU Ziyang (刘子扬).

Other people who have taken part in the development of this library include CUI Tao (崔涛), LENG Wei (冷伟) and ZHENG Weiying (郑伟英).

References

[1] Tao Cui, Wei Leng, Huaqing Liu, Linbo Zhang and Weiying Zheng, High-order numerical quadratures in a tetrahedron with an implicitly defined curved interface, ACM Transactions on Mathematical Software, 46, 1, Article 3 (March 2020), 18 pages. https://doi.org/10.1145/3372144

[2] R. I. Saye, High-order quadrature methods for implicitly defined surfaces and volumes in hyperrectangles, SIAM J. Sci. Comput., Vol. 37, No. 2, pp. A993–A1019, 2015

[3] Linbo Zhang, Tao Cui and Hui Liu, A set of symmetric quadrature rules on triangles and tetrahedra, Journal of Computational Mathematics, 27, 1, 2009, 89–96.