High Order Numerical Quadrature for XFEM

December 30, 2019

Contents

Introduction

Let T be a tetrahedron and $L(\mathbf{x})$ be a smooth level set function. PHG provides C functions for computing highly accurate approximations of the following types of integrals using the algorithm presented in [1]:

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

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 $u(\mathbf{x})$ must be smooth in T.

These functions are implemented in `src/quad-interface.c` and `include/phg/quad-interface.h`. They can be used in implementations of extended finite element methods (XFEM) or other immersed interface or immersed boundary methods on tetrahedral meshes.

1 The function `phgQuadInterface`

Main user interface. It can be used to compute any of the integrals listed in (1).

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

1.1 Prototype

```
int phgQuadInterface(DOF *ls, DOF *ls_grad, ELEMENT *e,
                     DOF_USER_FUNC func, int dim, DOF_PROJ proj,
                     int quad_type, int quad_order,
                     FLOAT *res, FLOAT **prule_data);
```

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

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`.

  For convenience, and as a special convention, the argument `ls` can be set to `NULL` and then the integral on the whole tetrahedron will be computed if `quad_type` $\neq 0$ (see below). In this case `ls_grad` must not be `NULL`, and can be set to any valid `DOF` attached to the grid such that `ls_grad->g` is valid.

- The tetrahedron T is specified by the argument `e`.

- The integrand $u(\mathbf{x})$ is specified by the arguments `func`, `dim` and `proj`. The type `DOF_USER_FUNC` is defined as:

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

  which evaluates the function at the point (x, y, z) and returns the result in `res`, and can encapsulate either a scalar function or a vector function of any dimension. `dim` specifies the dimension of $u(\mathbf{x})$. `proj` is only effective for the surface integral, it specifies the projection operation using the normal direction of the interface Γ. Let $\mathbf{n}(\mathbf{x})$ be the unit normal vector of Γ at $\mathbf{x}$ then:

  $$u(\mathbf{x}) = \begin{cases} \texttt{func}(\mathbf{x}), & \text{if } \texttt{proj} = \texttt{DOF\_PROJ\_NONE} \text{ or } \texttt{quad\_type} \neq 0, \\ \texttt{func}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}), & \text{if } \texttt{proj} = \texttt{DOF\_PROJ\_DOT} \text{ and } \texttt{quad\_type} = 0, \\ \texttt{func}(\mathbf{x}) \times \mathbf{n}(\mathbf{x}), & \text{if } \texttt{proj} = \texttt{DOF\_PROJ\_CROSS} \text{ and } \texttt{quad\_type} = 0. \end{cases}$$

- The integration type is specified by the argument `quad_type`:
  - volume integral in $T \cap \Omega^-$ if `quad_type` < 0;
  - volume integral in $T \cap \Omega^+$ if `quad_type` > 0, where $\Omega^+ := \{\mathbf{x} \in \Omega \mid L(\mathbf{x}) > 0\}$;

– surface integral on $\Gamma \cap T$ if `quad_type` $= 0$.

- The order of the quadrature rules to use is specified by the argument `quad_order`. Since 1D Gaussian rules have odd orders, the rule of order `quad_order` $+ 1$ is used if `quad_order` is even. In the case where the integral is computed by a planar approximation of the interface (for example when the interface is locally planar, or `quad_order` ≤ 1, or the tetrahedron is very small), quadrature rules for triangles and tetrahedra of order `quad_order`.

- The computed integral is returned in the argument `res`, which should be a pointer to an array of `FLOAT`s of size $\geq$ `dim`.

- Finally `prule_data`, if $\neq$ `NULL`, returns a pointer to a dynamically allocated buffer, to be freed by the calling function, containing the data for the computed quadrature rule. If `prule_data` $\neq$ `NULL`, the argument `func` can be `NULL` and the integral is actually computed iff `func` $\neq$ `NULL`.

  The data for the quadrature rule saved in `*prule_data` is an array of `FLOAT`s consisting of an one number header followed by a list of quadrature points and weights. The points are strictly inside the integration domain and the weights are all nonnegative. Please see comments in the source code of the function for detailed format of the data, which can be reused to compute integrals of the same type on the same element more efficiently, either directly or by calling the function `phgQuadInterfaceApplyRule` (see 5).

1.3 Command-line options

These options are for setting internal parameters or debugging the algorithm. They are prefixed with "`-qi_`". and can be used either as command-line options or at run-time with the `phgOptionsSet*` functions, for example:

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

The full list of options available in this category can be obtained by running any PHG program with the command-line option "`-help quad_interface`".

2 The function `phgQuadInterfaceGetRule`

Wrapper function for `phgQuadInterface`. It computes and returns the quadrature rule without actually computing an integral.

2.1 Prototype

```
int phgQuadInterfaceGetRule(DOF *ls, DOF *ls_grad, ELEMENT *e, DOF_PROJ proj,
                int quad_type, int quad_order, FLOAT **prule);
```

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

2.2 Arguments

The arguments have exactly the same meanings as those in `phgQuadInterface`.

3 The function phgQuadInterface2

Wrapper function for non PHG users. It calls `phgQuadInterface` for the actual computation. It doesn't use PHG's data structures thus can be used in non PHG programs

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

3.1 Prototype

```
int phgQuadInterface2(DOF_USER_FUNC ls, int ls_order,
                      DOF_USER_FUNC ls_grad, FLOAT (*tet)[3],
                      DOF_USER_FUNC func, int dim, DOF_PROJ proj,
                      int quad_type, int quad_order,
                      FLOAT *res, FLOAT **prule_data);
```

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

3.2 Arguments

- The argument `ls_order` specifies the polynomial order of the level set function (`ls_order` < 0 means `ls` is non polynomial).
- The array `tet[4][3]` gives the coordinates of the four vertices of the tetrahedron.
- The other arguments have the same meanings as in `phgQuadInterface`.

4 The function phgQuadInterface2GetRule

Wrapper function for `phgQuadInterface2`. It computes and returns the quadrature rule without actually computing an integral.

4.1 Prototype

```
int phgQuadInterface2GetRule(DOF_USER_FUNC ls, int ls_order,
                DOF_USER_FUNC ls_grad, FLOAT (*tet)[Dim], DOF_PROJ proj,
                int quad_type, int quad_order, FLOAT **prule);
```

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

4.2 Arguments

The arguments have exactly the same meanings as those in `phgQuadInterface2`.

5 The function phgQuadInterfaceApplyRule

Auxiliary function. It computes the integral of a given function using a quadrature rule returned by the `prule_data` argument of `phgQuadInterface` or `phgQuadInterface2`.

5.1 Prototype

```
int phgQuadInterfaceApplyRule(DOF_USER_FUNC func, int dim,
                        const FLOAT *rule_data, FLOAT *res);
```

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

5.2 Arguments

- The arguments `func`, `dim` and `res` are the same as in `phgQuadInterface` and `phgQuadInterface2`.
- The argument `rule_data` points to the quadrature rule.

6 The function `phgQuadInterfaceDumpRule`

Utility function. It prints out the quadrature points and weights in CSV (comma separated values) format.

6.1 Prototype

```
int phgQuadInterfaceDumpRule(const FLOAT *rule_data, FILE *fp);
```

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

6.2 Arguments

- The argument `rule_data` points to the quadrature rule.
- The argument `fp` is the output stream. If `fp == NULL` then rule is not printed (this can be used to get the number of points in the rule).

7 The function `phgQuadInterfaceMarkElements`

Auxiliary function. It sets the "`mark`" member of all elements in the mesh `ls->g` to 0 if the element might intersect with 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 is much more expensive then ordinary numerical quadrature functions, needs to be used.

7.1 Prototype

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

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

7.2 Arguments

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

References

[1] T. Cui, W. Leng, H. Liu, L. Zhang and W. Zheng, High-order numerical quadratures in a tetrahedron with an implicitly defined curved interface, to appear in ACM Trans. Math. Softw.