

QUAD_CACHE — A Numerical Integration Interface for Finite Element Methods

Linbo ZHANG

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

March 22, 2022

Abstract

The `QUAD_CACHE` module included in the PHG toolbox is a generic numerical integration interface for implementation of finite element methods. It provides the `QCACHE` object with a set of supporting functions, which can be used to transparently precompute and store values of basis functions and their derivatives (gradient, divergence, curl, etc.), as well as values of user functions, at the quadrature points of a given quadrature rule, and then use them to compute numerical integrations of various variational forms. It is intended to be the new user interface for implementing numerical integrations in PHG, and PHG users are encouraged to use it in place of the old style `phgQuadXXXX` functions.

This document briefly describes the design principle and main functions of the `QCACHE` object.

Contents

1	Introduction	1
2	The QDESC Object	3
3	User Functions	3
3.1	The function <code>phgQCNew</code>	3
3.2	The function <code>phgQCFree</code>	3
3.3	The functions <code>phgQCAdd*</code>	4
3.4	The functions <code>phgQCSetRule</code> and <code>phgQCSetQuad</code>	5
3.5	The functions <code>phgQCIntegrate*</code>	5
3.6	The functions <code>phgQCClone</code>	6
3.7	The functions <code>phgQCGetNP</code>	6
4	Sample programs	6

1 Introduction

In the implementation of finite element methods, one needs to compute integrals of the form:

$$\int_D f(x) dx$$

where the integration domain D is an element or a subdomain of an element (a face, an edge, etc.), and the integrand $f(x)$ is a product of basis functions, derivatives of basis functions and user functions, optionally projected along the normal or tangent directions of D in the case D is a lower dimensional surface or curve. These integrals are generally computed with numerical quadrature using a numerical quadrature rule [2].

Originally, PHG provides a set of functions for computing integrals, either in an element or on a face of an element, of various types of variational forms. For example, the function `phgQuadGradBasAGradBas` is used to compute integrals of the following form:

$$\int_e \nabla \varphi_j \mathbb{A} \nabla \varphi_i \, dx,$$

where e is an element, φ_j and φ_i are respectively the j -th and i -th basis function in the element, and \mathbb{A} is a coefficient function which can be either a scalar function or a matrix function. In the `phgQuadXXXXX` functions the values of the basis functions and their derivatives at the quadrature points are precomputed and stored (cached) in the corresponding `DOF_TYPE` object for reuse, in a transparent way across successive calls. See [3], Section 2.4, for more details.

The main problem with the above user interface for computing element-wise variational forms is that the set of functions is becoming increasingly large, and still users often need to write their own functions for types of variational forms not covered by available functions in the library, which is both non trivial and redundantly done by different users.

The `QCACHE` object is intended to replace the aforementioned numerical quadrature functions in PHG. It is implemented in the files `include/phg/quad-cache.h` and `src/quad-cache.c` in the PHG source package. It can be easily configured to interface with any finite element package, through providing a `QDESC` object with a small set of member functions. The interface for PHG is defined by the `QDESC` object `QD_PHG` in the file `src/quad-cache.c`, which can serve as a sample code if you wish to create your own `QDESC` objects for other finite element packages. Another `QDESC` example is the `QD_P4EST`, which defines an interface for the PHG-to-p4est module (see files in the `p4est` subdirectory in the PHG's source files.)

Let's illustrate with a simple example. For solving the Poisson equation $-\Delta u = f$ using finite element methods, we need to compute the following integrals:

$$\int_e \nabla \varphi_j \cdot \nabla \varphi_i \, dx, \quad \int_e f \varphi_i \, dx,$$

where e is an element, and φ_i ($0 \leq i < n$) are the basis functions in e . The pseudo code for computing the above integrals with a `QCACHE` object is as follows:

```

ELEMENT *e;
DOF *u_h = phgDofNew(g, DOF_P3, "u_h", NULL); /* u_h is the FE solution */
QCACHE *qc = phgQCNew(QD_PHG, u_h); /* u_h for the FE space */
int Q_f = phgQCAddXYZFunction(qc, func_f, 1); /* Q_f is the FID for f */
ForAllElements(g, e) {
    INT eno = e->index;
    int n = qc->qd->n.bas(qc, eno);
    QUAD *quad = phgQuadGetQuad3D(4); /* 4th order quadrature rule */
    FLOAT vol = phgGeomGetVolume(g, e);
    phgQCSetQuad(qc, quad, vol);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            val = phgQCIntegrate(qc, eno, Q_GRAD, j, qc, eno, Q_GRAD, i);
            /* add val to the stiffness matrix */
            ... ...
        }
        val = phgQCIntegrate(qc, eno, Q_BAS, i, qc, eno, Q_f, 0);
        /* add val to the RHS vector */
        ... ...
    }
}
phgQCFree(&qc);

```

In the above example the P_3 element is used, g is the pointer to the `GRID` object, and f is defined by the user function `func_f`.

2 The QDESC Object

The QDESC object is used to define an interface to a specific finite element package. For more information please refer to the comments on the struct QDESC in `include/phg/quad-cache.h`, and the code defining QD_PHG in `src/quad-cache.c`.

3 User Functions

In this section we briefly introduce user functions for the QCACHE object.

3.1 The function phgQCNew

```
QCACHE *phgQCNew(QDESC *qd, void *fe);
```

It creates a new QCACHE object and returns a pointer to it. `qd` points to a predefined QDESC object (QD_PHG for PHG users). `fe` is a pointer to a struct which provides information about the underlying finite element space. The actual type of struct pointed to by `fe` depends on `qd` (for QD_PHG, `fe` points to an DOF struct.)

A QCACHE object has functions attached to it, which are identified with an id of type `int` (called function id, or FID) and can be used in the numerical integration functions presented later.

Below is a list of types of attached functions in a QCACHE object:

- The basis functions of the underlying finite element space. They are identified with the FID `Q_BAS`. The member function `n_bas()` in the QDESC object returns the number of basis functions in the element (predefined).
- The gradients of the basis functions of the underlying finite element space. They are identified with the FID `Q_GRAD` (predefined).
- The divergences of the basis functions of the underlying finite element space. They are identified with the FID `Q_DIV` (predefined, only valid for vector basis functions, i.e., when the dimension of the basis functions equals the space dimension).
- The curls of the basis functions of the underlying finite element space. They are identified with the FID `Q_CURL` (predefined, only valid for vector basis functions, i.e., when the dimension of basis functions equals the space dimension).
- An user function, attached through calling one of the functions `phgQCAdd*Function`.
- The result of multiplying a predefined or previously attached function of any type with an user function (the latter is called *the coefficient function*). This type of functions are attached by calling one of the functions `phgQCAdd*Coefficient`.
- The result of projecting a predefined or previously attached attached function of any type using the normal vectors either defined by calling the function `phgQCSetConstantNormal`, or available in the quadrature rule. This type of functions are created by calling the function `phgQCAddProjection`.
- The result of applying an operator to a predefined or previously attached function of any type. This is done through calling the function `phgQCAddOperator`. It provides a mechanism to apply a general transformation to an attached function. In fact `Q_DIV`, `Q_CURL` and `phgQCAddProjection` are internally implemented as operators.

3.2 The function phgQCFree

```
void phgQCFree(QCACHE **qc);
```

It frees the QCACHE object pointed by `*qc`, and sets `qc` to `NULL`.

3.3 The functions phgQCAAdd*

These functions attach an user function or a coefficient, or apply a projection or an operator to an existing function, to the given QCACHE object. They return the FID of the resulting new attached function.

- The function `phgQCAAddFEFunction`

```
int phgQCAAddFEFunction(QCACHE *qc, void *fe);
```

It attaches the finite element function `fe` (a DOF object for QD_PHG) to `qc`.

- The function `phgQCAAddXYZFunction`

```
int phgQCAAddXYZFunction(QCACHE *qc, FUNC_3D func, int dim);
```

It attaches the function `func` to `qc`. The dimension (number of components) of the function is given by `dim`.

- The function `phgQCAAddConstantFunction`

```
int phgQCAAddConstantFunction(QCACHE *qc, FLOAT *data, int dim);
```

It attaches a constant function to `qc`. The values of the function are given by `data`, and the dimension (number of components) of the function is given by `dim`.

- The function `phgQCAAddFECoefficient`

```
int phgQCAAddFECoefficient(QCACHE *qc, void *fe, int base_fid);
```

It adds a coefficient function. The resulting function, which is attached to `qc` with the returned FID, is the product of `fe` with the previously attached function whose FID is `base_fid`.

Let m be the dimension of the function `fe` and n the dimension of the function `base_fid`. The following conventions on the type of product and the dimension of the resulting function are used, which are subject to future extensions:

- If $n = 1$ or $m = 1$, then the product is regarded as a scalar function multiplied by a scalar or vector function, and the dimension of the resulting function is nm .
- If $n > 1$ and $m = n$, then `fe` is regarded as a diagonal $n \times n$ matrix, and the dimension of the resulting function is n .
- If $n > 1$ and $m = n^2$, then `fe` is regarded as a full $n \times n$ matrix, and the dimension of the resulting function is n .

- The function `phgQCAAddXYZCoefficient`

```
int phgQCAAddXYZCoefficient(QCACHE *qc, FUNC_3D func, int dim, int base_fid);
```

It's similar to `phgQCAAddFECoefficient`, but adds a `FUNC_3D` function of dimension `dim`, instead of a finite element function.

See `phgQCAAddFECoefficient` for conventions on the dimensions of the functions.

- The function `phgQCAAddConstantCoefficient`

```
int phgQCAAddConstantCoefficient(QCACHE *qc, void *data, int dim, int base_fid);
```

It's similar to `phgQCAAddFECoefficient`, but adds a constant function of dimension `dim` whose values are given by `data`.

See `phgQCAAddFECoefficient` for conventions on the dimensions of the functions.

- The function `phgQCAAddFIDCoefficient`

```
int phgQCAAddFIDCoefficient(QCACHE *qc, int fid, int base_fid);
```

It's similar to `phgQCAAddFECoefficient`, but the coefficient function is a previously attached function whose FID is `fid`.

See `phgQCAAddFECoefficient` for conventions on the dimensions of the functions.

- The function `phgQCAddProjection`

```
int phgQCAddProjection(QCACHE *qc, PROJ proj, int fid);
```

It attaches to `qc` the new function obtained by projecting the existing function with the FID `fid`. `proj` specifies type of projection (`PROJ_NONE`, `PROJ_DOT` and `PROJ_CROSS`). To use this function normal vectors at the quadrature points must be available, either provided by the current quadrature rule, or defined by `phgQCSetConstantNormal`.

- The function `phgQCAddOperator`

```
int phgQCAddOperator(QCACHE *qc, OP_FUNC op, int fid);
```

It creates and attaches to `qc` the new function obtained by applying the operator `op` to an existing function with the FID `fid`.

- The function `phgQCSetConstantNormal`

```
void phgQCSetConstantNormal(QCACHE *qc, const FLOAT *nv);
```

It defines the constant normal vector (the same vector at all the quadrature points) used in the projections.

3.4 The functions `phgQCSetRule` and `phgQCSetQuad`

These two functions set (or change) the quadrature rule of the `QCACHE` object. Calling one of them will clear stored (cached) values of all function values as well as normal vectors.

The argument `scale` is the scaling factor (the weights will be multiplied by `scale`). It's also used to indicate whether the quadrature points are in physical coordinates (`scale < 0`, and in this case the weights are not scaled), or in reference coordinates (`scale ≥ 0`).

- The function `phgQCSetRule`

```
void phgQCSetRule(QCACHE *qc, FLOAT *rule, FLOAT scale);
```

It sets the quadrature rule of `qc` to `rule`. Note the quadrature rule can be obtained by calling one of the `phgQuadGetRule*` and `phgQuadInterface*` functions, see [1] and [doc/quad-XFEM.pdf](#) for more information.

- The function `phgQCSetQuad` (for PHG only)

```
void phgQCSetQuad(QCACHE *qc, QUAD *quad, FLOAT scale);
```

It sets the quadrature rule of `qc` to `quad`.

3.5 The functions `phgQCIntegrate*`

These functions compute integrals using the attached functions, and the current quadrature rule and normal vectors of the specified `QCACHE` objects. Currently only integration of bilinear forms are implemented.

- The function `phgQCIntegrate`

```
FLOAT phgQCIntegrate(QCACHE *qc1, INT e1, int fid1, int i1,  
                    QCACHE *qc2, INT e2, int fid2, int i2);
```

It computes and returns the integral of the (dot) product of two functions, where `fid1` and `fid2` are FIDs, `e1` and `e2` are element indices, with `fid1` attached to `qc1` and evaluated in `e1`, and `fid2` attached to `qc2` and evaluated in `e2`. If `fid1` (resp. `fid2`) is derived from a basis function, then `i1` (resp. `i2`) gives the index of the basis function.

For this function the dimensions (number of components) of `fid1` and `fid2` must be the same.

- The function `phgQCIntegrateM`

```

FLOAT *phgQCIntegrateM(QCACHE *qc1, INT e1, int fid1, int i1,
                      QCACHE *qc2, INT e2, int fid2, int i2,
                      int M, int N, int K, FLOAT *res);

```

It computes integral of the product of two functions and returns the result in `res`. Here `fid1` (resp. `fid2`) is regarded as a matrix function of dimension $M \times K$ (resp. $K \times N$) and the integrand is regarded as a matrix function of dimension $M \times N$, and `res` points to an array of `FLOAT` whose length is at least $M \times N$ for returning the result. The other arguments (`qc1` through `i2`) have the same meanings as in the function `phgQCIntegrate`.

See comments at the top of the function `phgQCIntegrate_` for more information.

- The function `phgQCIntegrateFace`

```

FLOAT phgQCIntegrateFace(QCACHE *qc1, INT e1, int face1, int fid1,
                        PROJ proj1, int i1,
                        QCACHE *qc2, INT e2, int face2, int fid2,
                        PROJ proj2, int i2);

```

It computes and returns the integral of the (dot) product of two functions on a face (in 2D a “face” is actually an edge). The face can be an element face identified by both (`e1`, `face1`) and (`e2`, `face2`) (they should point to the same face), or the intersection of a surface (the interface) with an element for interface problems. `proj1` (resp. `proj2`) is the projection to apply to `fid1` (resp. `fid2`). The other arguments have the same meanings as in the function `phgQCIntegrate`.

- The function `phgQCIntegrateFaceM`

```

FLOAT *phgQCIntegrateFaceM(QCACHE *qc1, INT e1, int face1, int fid1,
                          PROJ proj1, int i1,
                          QCACHE *qc2, INT e2, int face2, int fid2,
                          PROJ proj2, int i2,
                          int M, int N, int K, FLOAT *res);

```

It computes integral of the product of two matrix functions and returns the result in `res`. The arguments `M`, `N` and `K` have the same meanings as in the function `phgQCIntegrateM`, while the other arguments have the same meanings as in the function `phgQCIntegrateFace`.

3.6 The functions `phgQCClone`

```

QCACHE *phgQCClone(QCACHE *qc0);

```

It returns a `QCACHE` object with the same FE space, functions, coefficients, operators, projections, and constant normal vector as that of `qc0`.

3.7 The functions `phgQCGetNP`

```

int phgQCGetNP(QCACHE *qc);

```

It returns the number of points of the current quadrature rule in `qc`.

4 Sample programs

The programs `ipdg.c` (an IPDG code for the Poisson equation) and `interface.c` (an IPDG code for an elliptic interface problem) in the `examples/` subdirectory in PHG’s source tree can serve as sample programs on using the `QUAD_CACHE` interface.

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] 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.
- [3] 崔涛, PHG 中线性棱单元的实现及三维电磁时谐场问题并行自适应数值模拟实验, 硕士学位论文, 中国科学院数学与系统科学研究院, 2006.