

# Efficient and User-friendly Computation of Local Stiffness Matrices

Jochen Härdtlein, and Christoph Pflaum \*  
University of Erlangen-Nuremberg,  
jochen.haerdtlein@informatik.uni-erlangen.de

## Abstract

The finite element method is a way to discretize partial differential equations. The problem is transformed to its weak formulation, approximated by ansatz and testing functions, in order to reach a linear system of equations. Although, this method is very flexible and powerful, many students and scientists of application ranges flee from the finite element method, since it has a complex mathematical background.

Therefore, we introduce *Colsamm*. Starting from an interface very close to the weak formulation of partial differential equations, *Colsamm* computes the local discretization stencil restricted to an element. Further, users can assemble the global stiffness matrices, by using their specific grid informations. Additionally, *Colsamm* provides many opportunities to adapt the finite element method to specific problems; e.g. easy changing of ansatz and testing functions.

## 1 Introduction

### 1.1 Motivation

The finite element method (FEM) is a well-known way to discretize differential operators. Thereby, a partial differential equation (PDE) is transformed to its weak formulation and approximated by ansatz and testing functions. The result is a linear system of equations, that has to be solved by suitable methods.

In comparison to other methods; e.g. the finite difference method (FDM), the FEM is more powerful and more flexible in use. However, in opposite to FDM, that is based on a very intuitive idea, the FEM base on more complex mathematical backgrounds; e.g. see [Bra97]. These backgrounds must be understood at least in parts, in order to be capable of applying and implementing the FEM. However, the complex mathematical base terrifies many students and scientists of application ranges, just intending a fast solution of their problem. Those users, that are more interested in results than in the method of discretization, ought to benefit from the advantages of the FEM without concerning oneself with the mathematical background in detail. The users only have to know, that this ansatz discretizes differential operators and yields the local stiffness matrices. On that account, *Colsamm* (**Computation Of Local Stiffness And Mass Matrices**) was developed to enable an easy use of the FEM without an intensive study of the mathematical basics.

---

\*Department of Computer Science 10, Systemsimulation, Cauerstr. 6, D-91058 Erlangen, Germany

## 1.2 Introducing Example

To offer a first impression of *Colsamm*, we present an example, computing the local stiffness matrices for Poisson's equation, based on a grid of non-regular hexahedrons. By transforming the equation to the weak formulation, we yield

$$\int_{\Omega_h} \nabla u \cdot \nabla v \, d\mu = \int_{\Omega_h} u \cdot v \, d\mu,$$

where  $\Omega_h$  represents the discretized domain. First, a reference hexahedron is initialized, containing the accuracy of the Gaussian quadrature formula as template parameter. Thereby, the linear basis functions are defined. Extensions for more complex elements are presented later on. Hence, the user can start the computations of the local stiffness matrix. Therefore, the vertices of the actual hexahedron are assigned to `my_element` as array or STL vector, enumerated clockwise in the x-y-plane. `v()` and `w()` represent the ansatz and testing functions, respectively.

```
Hexahedron<Gauss2> my_element;

for(int i = 0; i < number_elements; ++i){
    // put the verticies of the actual element in right order
    vertices = element[i].get_verticies_as_array();
    // compute stiffness matrices of the element;
    loc_stenc_1 = my_element(vertices).int_all(grad(v())*grad(w()));
    loc_stenc_2 = my_element.int_all(v() * w());

    for(int k = 0; k < size_of_ansatz_functions; ++k){
        global_k = element.loc_to_glob_numbering(k);
        for(int l = 0; l < size_of_testing_functions; ++l){
            global_l = element.loc_to_glob_numbering(l);
            global_A[global_k][global_l] += loc_stenc_1[k][l];
            global_H[global_k][global_l] += loc_stenc_2[k][l];
        }
    }
}
```

The entries of the two element stiffness matrices `loc_stenc_1` and `loc_stenc_2` are added to the corresponding positions of the global discretization matrices. Applying these steps to all hexahedrons of the grid, we yield the global discretization matrix. While this was a very simple and common example, we further will present the implementation techniques of the library, in order to point out the features realized in *Colsamm*.

## 2 Policies and Applications of *Colsamm*

### 2.1 C++ Template Libraries

Nearly all control parameters in *Colsamm* are template parameters, in order to evaluate the problem specific decisions at compile-time. Since the computation of stiffness matrices

implies repeated calls of the library, every decision at run-time causes a significant lack to performance. Thus, no object file of *Colsamm* can be achieved, as every template parameter must be known at compile-time. Due to that fact, *Colsamm* just consists of header files, that have to be included. The way the problem-specific parameters are set is presented below. Operator overloading enables a programmer to build math-like and user-friendly interfaces. Since traditional operator overloading suffers from several performance lacks, we apply the Fast Expression Template technique (FET), see [HLP05]. As improvement of Expression Templates, where temporary variables are already avoided, FET reach better performance especially for small amounts data. Those occur during the computations for the basis functions and the transformation formula for an element.

## 2.2 Local Stiffness Matrices

In order to keep *Colsamm* very flexible in use, the library does not work on the whole grid. The user provides the vertices of one actual element in a local numbering, concerning the reference element. The resulting matrix contains the computations of the integrals, in the local numbering, as well. The mapping to the global numbering has to be realized by the user. Due to this fact, *Colsamm* can be applied to any grids, even if they cover different element types.

Concerning the FEM, the user has to know, how to combine these local stiffness matrices restricted to an element to the complete local stencil. This is determined by the supports of the used basis functions and the structure of the grid. A basis function is 1 at one discretization point and vanishes at every other point. The complete local stencil is computed by adding all local restricted stiffness matrices of those elements, that build the support of the basis function.

## 2.3 User-Defined Element Types

*Colsamm* covers all standard elements in 1D, 2D, and 3D, concerning linear basis functions. The library also contains all formulas for the computations of the determinants, and the dimension-dependent substitution formula. Therefore, it is easy for a user to realize any other types of elements. An element is defined by

1. the shape of the reference element including the dimension and vertices,
2. the transformation formula defining the mapping from the reference element to a general element, and
3. the sets of the ansatz and testing functions defined on the reference element.

An user-defined element has to provide these informations in an appropriate element class. Most parameters are delivered to the parent `Element` class, managing the storage and the interface. Since the derivation from this parent element is realized at compile-time, the parent class needs the type of the child as template parameter. In the following we present the manner a triangle covering linear basis functions is defined:

```

template <Int_Mod t>
struct Triangle : public _Domain_<
    Triangle<t>, // type of element
    One_Set<3>, // # of basis functions
    3, // # of vertices
    2, // dimension
    Gauss<t,triangle>, // Gauss quadrature
    double >{ // real or complex
Triangle() { // Setting of the basis functions
    Set(1.-X(1)-Y(1));
    Set(X(1));
    Set(Y(1));
    finalize(X()*Y()); //precomputations & initialization
}
inline static void Transformation(){ // Transformation
    Mapping(C() , P_(0));
    Mapping(X(1) , P_(1)-P_(0));
    Mapping( Y(1) , P_(2)-P_(0));
    Interior_2D(); // Starting computations
}
};

```

This very short example ought to show the syntax, for building a new element. Thereby, the user has the opportunity to implement any element, by writing such a class. Every type of basis functions, and every mapping of polynomial type can be realized; e.g. iso-parametric elements. Standard boundary elements; e.g. a surface triangle or quadrangle in 3D, are contained in *Colsamm*, too. In order to deal with mixed finite elements, the user changes the `One_Set<. >` to `Two_Sets<., .>`. Thereby, one can define different sets for ansatz and testing functions, respectively.

Hence, the trial and error of different basis function approaches is very easy. The user has only to care about the varying assembling of the global stiffness matrices.

## 2.4 Integrands

*Colsamm* supports the common differential operations, that occur in the weak formulations of PDEs. Starting from the basic arithmetic operations, the ansatz function `v()` and testing function `w()`, directional derivatives (`d_dx(.)`, `d_dy(.)`, `d_dz(.)`), and the gradient (`grad(.)`). Due to the usage of FET, a unaesthetic side-effect arises: all constants and user-defined functions have to be enumerated by a template integer, that has to be unique inside the integrand. This enumerated constant is implemented by `D_<template int>(double)`. This list of possible operations is not complete, since the features provided by *Colsamm* are best described in precise examples. However, this would blow the limit of this article.

Up to now, only integrands that describe scalar fields can be computed. This is not a hard restriction, while the user can split the PDE into suitable parts, yet. However, to increase the user-friendliness, we are working on computing vector fields, as well.

### 3 Computation and Performance Issues

The main topics arising during the implementation of *Colsamm*, were to reach user-friendliness and high-performance, as well. Focusing on performance, we first faced the problem, that same terms were computed multiple times. In order to get rid of re-computations, we divided the computations in three layers. First, we need to know the type and the accuracy of the Gaussian quadrature formula (see [SB02]) to be applied. The three computation layers are:

1. During initializing the element: evaluation of the basis functions and derivatives at the Gaussian points on the reference element. These values are stored as they necessary for nearly all integral computations.
2. After the vertices of the actual element are set, the mapping of the actual element and its derivatives are computed. Subsequently, the library again calculates the values at the Gaussian points, which are stored, too, because they are fix for each element.
3. While starting the integration, *Colsamm* computes the expression-dependent integrals using the corresponding precomputed values

Therefore, and due to the fact of the high grade of template programming, *Colsamm* became a high efficient code for stencil computations. Applying FET to the performance relevant parts we nearly reach the efficiency of hand-crafted C-code that has not any user-friendliness.

The template evaluations, that are computed at compile-time, cause a longer time for compilation. Compared to the compile-time for a whole reasonable problem, however, this yields no significant delays. For tests without any performance issue, we suppose to omit the `-O3` option to decrease compile-time.

*Colsamm* is able to calculate with complex numbers, as well, however, performance extremely decreases. This is caused by the STL implementation of the complex numbers, that uses traditional operator overloading. This implies creating, copying, and deleting temporary objects. One solution is to add a library-embedded complex number implementation, to do the stencil computations in suitable time.

### 4 Downloading *Colsamm*

In order to ease the usage of FEM for everyone, we provide *Colsamm* for download. It is a free software underlying the GNU General Public License (see [GNU91]). *Colsamm* can be downloaded at the following address:

`www10.informatik.uni-erlangen.de/~jochen/colsamm.html`.

The zip-file contains, besides the *Colsamm*-library, an example file, that demonstrates the usage *Colsamm* in different cases. Additionally, a how-to is enclosed, describing the library in more detail. Remarks and improvements are welcome at `jochen.haerdtlein@informatik.uni.erlangen.de`.

## 5 Conclusions and Future Work

*Colsamm* is already in use for many different application ranges; e.g. simulation of lasers, simulation of bioelectric fields, global simulation of melting furnace. Motivated by these applications we are working several extensions of *Colsamm*, that ease usage and realize some user-specific requests.

- Extending the transformation formulas to any formula type, not restricted to polynomials. This empowers *Colsamm* to compute local stiffness matrices on elements with spherical segments.
- Integration of vector fields, matrices and tensors. Thereby, the usage of the library would become much easier, since the integrands have not to be split in scalar-fields anymore.
- Providing computations for second derivatives. Up to now, only first derivatives can be calculated. This extension would open up more application ranges.
- Front-end for the calculation of the complete local stencil, to ease the use for users with low experiences with FEM.

Besides, we permanently pay attention to the performance of the template library, to keep it a rational way for computing the stiffness and mass matrices.

Thus, *Colsamm* is a senseful, efficient, and user-friendly software, that spares time while designing a simulation. And scientists and students of application ranges can focus on getting the solution, instead of implementing the suitable discretization method.

## References

- [Bra97] D. Braess. *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*. Cambridge University Press, second edition, 1997.
- [GNU91] Gnu general public license, 1991. Version 2, [www.gnu.org/licenses/gpl.html](http://www.gnu.org/licenses/gpl.html).
- [HLP05] J. Härdtlein, A. Linke, and C. Pflaum. Fast expression templates. In V.S. Suneram, G.D.v. Albada, P.M.A. Sloot, and J.J. Dongarra, editors, *Computational Science - ICCS 2005*, volume 3515 of *LNCS*, pages 1055 – 1063. Springer, May 2005. ISBN-10 3-540-26043-9, ISBN-13 978-3-540-26043-1, ISSN 03-2-9743.
- [SB02] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer, third edition, 2002.