A presentation of the library OFELI

Rachid Touzani

Laboratoire de Mathématiques
Université Blaise Pascal
Clermont-Ferrand, France

Outlook

1. What is OFELI ?
2. Levels of use
3. A finite element code example using **OFELI**
4. The library structure
5. The **OFELI** package
6. Current and future developments

What is **OFELI** ?

- **OFELI**: **O**bject **F**inite **E**lement **LI**brary
- The **OFELI** library is a collection of C++ classes C++ and utilities enabling the construction of finite element codes.
- It provides a variety of prototypes codes enabling familiarity with the library usage
- It enables implementation of other approximation methods (finite volumes, finite differences, integral representations, . . .)
- It contains utility programs for:
  - Mesh generation in 2-D
  - Conversion from and to various mesh generators and graphical post-processors
- . . .

What is not **OFELI** ?

- A programming environment (like `Matlab`, `Scilab`, . . .)
- A metalanguage for finite element programming (like `FreeFem`, `Melina`, . . .)

What is **OFELI** ?

- **OFELI**: **O**bject **F**inite **E**lement **LI**brary
- The **OFELI** library is a collection of C++ classes C++ and utilities enabling the construction of finite element codes.
- It provides a variety of prototypes codes enabling familiarity with the library usage
- It enables implementation of other approximation methods (finite volumes, finite differences, integral representations, . . .)
- It contains utility programs for:
  - Mesh generation in 2-D
  - Conversion from and to various mesh generators and graphical post-processors
- . . .

What is not **OFELI** ?

- A programming environment (like `Matlab`, `Scilab`, . . .)
- A metalanguage for finite element programming (like `freefem`, `Melina`, . . .)

What is **OFELI** ?

- **OFELI**: **O**bject **F**inite **E**lement **LI**brary
- The **OFELI** library is a collection of C++ classes C++ and utilities enabling the construction of finite element codes.
- It provides a variety of prototypes codes enabling familiarity with the library usage
- It enables implementation of other approximation methods (finite volumes, finite differences, integral representations, . . .)
- It contains utility programs for:
  - Mesh generation in 2-D
  - Conversion from and to various mesh generators and graphical post-processors
- . . .

What is not **OFELI** ?

- A programming environment (like `Matlab`, `Scilab`, . . .)
- A metalanguage for finite element programming (like `freefem`, `Melina`, . . .)

What is **OFELI** ?

- **OFELI**: **O**bject **F**inite **E**lement **LI**brary
- The **OFELI** library is a collection of C++ classes C++ and utilities enabling the construction of finite element codes.
- It provides a variety of prototypes codes enabling familiarity with the library usage
- It enables implementation of other approximation methods (finite volumes, finite differences, integral representations, . . .)
- It contains utility programs for:
  - Mesh generation in 2-D
  - Conversion from and to various mesh generators and graphical post-processors
- . . .

What is not **OFELI** ?

- A programming environment (like Matlab, Scilab, . . .)
- A metalanguage for finite element programming (like freefem, Melina, . . .)

What is **OFELI** ?

- **OFELI**: **O**bject **F**inite **E**lement **LI**brary
- The **OFELI** library is a collection of C++ classes C++ and utilities enabling the construction of finite element codes.
- It provides a variety of prototypes codes enabling familiarity with the library usage
- It enables implementation of other approximation methods (finite volumes, finite differences, integral representations, . . .)
- It contains utility programs for:
  - Mesh generation in 2-D
  - Conversion from and to various mesh generators and graphical post-processors
- . . .

What is not **OFELI** ?

- A programming environment (like Matlab, Scilab, . . .)
- A metalanguage for finite element programming (like FreeFem, Melina, . . .)

What is **OFELI** ?

- **OFELI**: **O**bject **F**inite **E**lement **LI**brary
- The **OFELI** library is a collection of C++ classes C++ and utilities enabling the construction of finite element codes.
- It provides a variety of prototypes codes enabling familiarity with the library usage
- It enables implementation of other approximation methods (finite volumes, finite differences, integral representations, . . .)
- It contains utility programs for:
  - Mesh generation in 2-D
  - Conversion from and to various mesh generators and graphical post-processors
- . . .

What is not **OFELI** ?

- A programming environment (like Matlab, Scilab, . . .)
- A metalanguage for finite element programming (like FreeFem, Melina, . . .)

What is **OFELI** ?

- **OFELI**: **O**bject **F**inite **E**lement **LI**brary
- The **OFELI** library is a collection of C++ classes C++ and utilities enabling the construction of finite element codes.
- It provides a variety of prototypes codes enabling familiarity with the library usage
- It enables implementation of other approximation methods (finite volumes, finite differences, integral representations, . . .)
- It contains utility programs for:
  - Mesh generation in 2-D
  - Conversion from and to various mesh generators and graphical post-processors
- . . .

What is not **OFELI** ?

- A programming environment (like Matlab, Scilab, . . .)
- A metalanguage for finite element programming (like FreeFem, Melina, . . .)

What is **OFELI** ?

- **OFELI**: **O**bject **F**inite **E**lement **LI**brary
- The **OFELI** library is a collection of C++ classes C++ and utilities enabling the construction of finite element codes.
- It provides a variety of prototypes codes enabling familiarity with the library usage
- It enables implementation of other approximation methods (finite volumes, finite differences, integral representations, . . .)
- It contains utility programs for:
  - Mesh generation in 2-D
  - Conversion from and to various mesh generators and graphical post-processors
- . . .

What is not **OFELI** ?

- A programming environment (like `Matlab`, `Scilab`, . . .)
- A metalanguage for finite element programming (like `freefem`, `Melina`, . . .)

What is **OFELI** ?

- **OFELI**: **O**bject **F**inite **E**lement **LI**brary
- The **OFELI** library is a collection of C++ classes C++ and utilities enabling the construction of finite element codes.
- It provides a variety of prototypes codes enabling familiarity with the library usage
- It enables implementation of other approximation methods (finite volumes, finite differences, integral representations, . . .)
- It contains utility programs for:
  - Mesh generation in 2-D
  - Conversion from and to various mesh generators and graphical post-processors
- . . .

What is not **OFELI** ?

- A programming environment (like `Matlab`, `Scilab`, . . .)
- A metalanguage for finite element programming (like `freefem`, `Melina`, . . .)

What is **OFELI** ?

- **OFELI**: **O**bject **F**inite **E**lement **LI**brary
- The **OFELI** library is a collection of C++ classes C++ and utilities enabling the construction of finite element codes.
- It provides a variety of prototypes codes enabling familiarity with the library usage
- It enables implementation of other approximation methods (finite volumes, finite differences, integral representations, . . .)
- It contains utility programs for:
  - Mesh generation in 2-D
  - Conversion from and to various mesh generators and graphical post-processors
- . . .

What is not **OFELI** ?

- A programming environment (like `Matlab`, `Scilab`, . . .)
- A metalanguage for finite element programming (like `freefem`, `Melina`, . . .)

Levels of use of the library

- Level 0: No knowledge of C++ is required. Use of prototype programs (Demos)
- Level 1: Programming of finite element codes using **OFELI** classes. Possible contribution with classes and functions
  $\Longrightarrow$ Contribution to Demo programs
- Level 2: Contribution to the library's kernel by Implementing equations classes and solvers, ...

What are **objects** in a finite element code ?

Mathematical entities that one manipulates when solving a problem: Mesh, matrices, vectors, equations, solvers (nonlinear problems, optimization, ...), ...

Levels of use of the library

- Level 0: No knowledge of C++ is required. Use of prototype programs (Demos)
- Level 1: Programming of finite element codes using **OFELI** classes. Possible contribution with classes and functions
  $\Longrightarrow$ Contribution to Demo programs
- Level 2: Contribution to the library's kernel by Implementing equations classes and solvers, ...

What are **objects** in a finite element code ?

Mathematical entities that one manipulates when solving a problem: Mesh, matrices, vectors, equations, solvers (nonlinear problems, optimization, ...), ...

Levels of use of the library

- Level 0: No knowledge of C++ is required. Use of prototype programs (Demos)
- Level 1: Programming of finite element codes using **OFELI** classes. Possible contribution with classes and functions
  $\Longrightarrow$ Contribution to Demo programs
- Level 2: Contribution to the library's kernel by Implementing equations classes and solvers, ...

What are **objects** in a finite element code ?

Mathematical entities that one manipulates when solving a problem: Mesh, matrices, vectors, equations, solvers (nonlinear problems, optimization, ...), ...

Levels of use of the library

- Level 0: No knowledge of C++ is required. Use of prototype programs (Demos)
- Level 1: Programming of finite element codes using **OFELI** classes. Possible contribution with classes and functions
  $\Longrightarrow$ Contribution to Demo programs
- Level 2: Contribution to the library's kernel by Implementing equations classes and solvers, ...

What are **objects** in a finite element code ?

Mathematical entities that one manipulates when solving a problem: Mesh, matrices, vectors, equations, solvers (nonlinear problems, optimization, ...), ...

Levels of use of the library

- Level 0: No knowledge of C++ is required. Use of prototype programs (Demos)
- Level 1: Programming of finite element codes using **OFELI** classes. Possible contribution with classes and functions
  $\Longrightarrow$ Contribution to Demo programs
- Level 2: Contribution to the library's kernel by Implementing equations classes and solvers, ...

What are **objects** in a finite element code ?

Mathematical entities that one manipulates when solving a problem: Mesh, matrices, vectors, equations, solvers (nonlinear problems, optimization, ...), ...

An example of program

Consider the following boundary value problem:

$$\begin{aligned} \Delta u &= 0 \qquad \text{in } \Omega \subset \mathbb{R}^2 \quad (\text{or } \mathbb{R}^3) \\ u &= g \qquad \text{on } \partial\Omega \end{aligned}$$

Matrix Formulation

$$\mathbf{Au} = \mathbf{b}$$

where

$$a_{ij} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \, d\mathbf{x}$$

Boundary Conditions:

We enforce $u = g$ by a penalty technique:

$$\sum_{j=1}^{i-1} a_{ij} u_j + \sum_{j=i+1}^{N} a_{ij} u_j + \lambda a_{ii} u_i = \lambda a_{ii} g(x_i) \qquad \lambda \gg 1$$

for each node i on the boundary.

An example of program

Consider the following boundary value problem:

$$\begin{aligned} \Delta u &= 0 \qquad \text{in } \Omega \subset \mathbb{R}^2 \quad (\text{or } \mathbb{R}^3) \\ u &= g \qquad \text{on } \partial\Omega \end{aligned}$$

Matrix Formulation

$$\mathbf{Au} = \mathbf{b}$$

where

$$a_{ij} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \, dx$$

Boundary Conditions:

We enforce $u = g$ by a penalty technique:

$$\sum_{j=1}^{i-1} a_{ij} u_j + \sum_{j=i+1}^{N} a_{ij} u_j + \lambda a_{ii} u_i = \lambda a_{ii} g(x_i) \qquad \lambda \gg 1$$

for each node i on the boundary.

An example of program

Consider the following boundary value problem:

$$\begin{aligned} \Delta u &= 0 \qquad \text{in } \Omega \subset \mathbb{R}^2 \quad (\text{or } \mathbb{R}^3) \\ u &= g \qquad \text{on } \partial\Omega \end{aligned}$$

Matrix Formulation

$$\mathbf{Au} = \mathbf{b}$$

where

$$a_{ij} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \, d\mathbf{x}$$

Boundary Conditions:

We enforce $u = g$ by a penalty technique:

$$\sum_{j=1}^{i-1} a_{ij} u_j + \sum_{j=i+1}^{N} a_{ij} u_j + \lambda a_{ii} u_i = \lambda a_{ii} g(x_i) \qquad \lambda \gg 1$$

for each node i on the boundary.

An example of program

Consider the following boundary value problem:

$$\begin{aligned} \Delta u &= 0 \qquad \text{in } \Omega \subset \mathbb{R}^2 \quad (\text{or } \mathbb{R}^3) \\ u &= g \qquad \text{on } \partial\Omega \end{aligned}$$

Matrix Formulation

$$\mathbf{Au} = \mathbf{b}$$

where

$$a_{ij} = \int_\Omega \nabla \phi_j \cdot \nabla \phi_i \, d\mathbf{x}$$

Boundary Conditions:
We enforce $u = g$ by a penalty technique:

$$\sum_{j=1}^{i-1} a_{ij} u_j + \sum_{j=i+1}^{N} a_{ij} u_j + \lambda a_{ii} u_i = \lambda a_{ii} g(x_i) \qquad \lambda \gg 1$$

for each node i on the boundary.

```
#include "OFELI.h"
#include "Therm.h"
using namespace OFELI;

int main() {
   Mesh ms("test.m");
   SkSMatrix<double> A(ms);
   Vect<double> b(ms.getNbDOF()), bc(ms.getNbDOF());

// Initialize bc

   MeshElements(ms) {
      DC2DT3 eq(theElement);
      eq.Diffusion();
      eq.ElementAssembly(A);
   }

   A.Prescribe(ms,b,bc);
   A.Factor();
   A.Solve(b);

   cout << b;
   return 0;
}
```

```
#include "OFELI.h"
#include "Therm.h"
using namespace OFELI;

int main() {
   Mesh ms("test.m");
   SkSMatrix<double> A(ms);
   Vect<double> b(ms.getNbDOF()), bc(ms.getNbDOF());

// Initialize bc

   MeshElements(ms) {
      DC2DT3 eq(theElement);
      eq.Diffusion();
      eq.ElementAssembly(A);
   }

   A.Prescribe(ms,b,bc);
   A.Factor();
   A.Solve(b);

   cout << b;
   return 0;
}
```

```cpp
#include "OFELI.h"
#include "Therm.h"
using namespace OFELI;

int main() {
   Mesh ms("test.m");
   SkSMatrix<double> A(ms);
   Vect<double> b(ms.getNbDOF()), bc(ms.getNbDOF());

// Initialize bc

   MeshElements(ms) {
      DC2DT3 eq(theElement);
      eq.Diffusion();
      eq.ElementAssembly(A);
   }

   A.Prescribe(ms,b,bc);
   A.Factor();
   A.Solve(b);

   cout << b;
   return 0;
}
```

```
#include "OFELI.h"
#include "Therm.h"
using namespace OFELI;

int main() {
   Mesh ms("test.m");
   SkSMatrix<double> A(ms);
   Vect<double> b(ms.getNbDOF()), bc(ms.getNbDOF());

// Initialize bc

   MeshElements(ms) {
      DC2DT3 eq(theElement);
      eq.Diffusion();
      eq.ElementAssembly(A);
   }

   A.Prescribe(ms,b,bc);
   A.Factor();
   A.Solve(b);

   cout << b;
   return 0;
}
```

```
#include "OFELI.h"
#include "Therm.h"
using namespace OFELI;

int main() {
   Mesh ms("test.m");
   SkSMatrix<double> A(ms);
   Vect<double> b(ms.getNbDOF()), bc(ms.getNbDOF());

// Initialize bc

   MeshElements(ms) {
      DC2DT3 eq(theElement);
      eq.Diffusion();
      eq.ElementAssembly(A);
   }

   A.Prescribe(ms,b,bc);
   A.Factor();
   A.Solve(b);

   cout << b;
   return 0;
}
```

```cpp
#include "OFELI.h"
#include "Therm.h"
using namespace OFELI;

int main() {
   Mesh ms("test.m");
   SkSMatrix<double> A(ms);
   Vect<double> b(ms.getNbDOF()), bc(ms.getNbDOF());

// Initialize bc

   MeshElements(ms) {
      DC2DT3 eq(theElement);
      eq.Diffusion();
      eq.ElementAssembly(A);
   }

   A.Prescribe(ms,b,bc);
   A.Factor();
   A.Solve(b);

   cout << b;
   return 0;
}
```

```
#include "OFELI.h"
#include "Therm.h"
using namespace OFELI;

int main() {
   Mesh ms("test.m");
   SkSMatrix<double> A(ms);
   Vect<double> b(ms.getNbDOF()), bc(ms.getNbDOF());

// Initialize bc

   MeshElements(ms) {
      DC2DT3 eq(theElement);
      eq.Diffusion();
      eq.ElementAssembly(A);
   }

   A.Prescribe(ms,b,bc);
   A.Factor();
   A.Solve(b);

   cout << b;
   return 0;
}
```

```
#include "OFELI.h"
#include "Therm.h"
using namespace OFELI;

int main() {
   Mesh ms("test.m");
   SkSMatrix<double> A(ms);
   Vect<double> b(ms.getNbDOF()), bc(ms.getNbDOF());

// Initialize bc

   MeshElements(ms) {
      DC2DT3 eq(theElement);
      eq.Diffusion();
      eq.ElementAssembly(A);
   }

   A.Prescribe(ms,b,bc);
   A.Factor();
   A.Solve(b);

   cout << b;
   return 0;
}
```

```
#include "OFELI.h"
#include "Therm.h"
using namespace OFELI;

int main() {
   Mesh ms("test.m");
   SkSMatrix<double> A(ms);
   Vect<double> b(ms.getNbDOF()), bc(ms.getNbDOF());

// Initialize bc

   MeshElements(ms) {
      DC2DT3 eq(theElement);
      eq.Diffusion();
      eq.ElementAssembly(A);
   }

   A.Prescribe(ms,b,bc);
   A.Factor();
   A.Solve(b);

   cout << b;
   return 0;
}
```

```
#include "OFELI.h"
#include "Therm.h"
using namespace OFELI;

int main() {
   Mesh ms("test.m");
   SkSMatrix<double> A(ms);
   Vect<double> b(ms.getNbDOF()), bc(ms.getNbDOF());

// Initialize bc

   MeshElements(ms) {
      DC2DT3 eq(theElement);
      eq.Diffusion();
      eq.ElementAssembly(A);
   }

   A.Prescribe(ms,b,bc);
   A.Factor();
   A.Solve(b);

   cout << b;
   return 0;
}
```

Classes in OFELI

Some conventions

- Class names always begin with a capital letter.
- Member function names begin with capital letters except if the name starts with a verb. et les `get...`.
- Class members that modify a class have generally names that start with the verb `set` (*e.g.*, `setSize()`)
- Class members that return an information have generally names that start with the verb `get`. (*e.g.*, `getNbNodes()`)
- Most of classes have an overload of the operator `<<`

To each phase in the procedure corresponds a *family of classes* :

1. Mesh classes

Construction of a mesh:	`Mesh ms("test.m");`
Output of a mesh :	`cout << ms;`
Loop over elements:	`for (ms.topElement(); (theElement=ms.getElement());)` `cout << *theElement;`
or equivalently:	`MeshElements(ms)`
Get pointer to a node:	`Node *nd = el->getPtrNode(2);`
Creation of boundary sides:	`ms.getBoundarySides();`
Creation of all sides:	`ms.getAllSides();`
Change of unknown support:	`ms.setDOFSupport(SIDE_DOF);`

To each phase in the procedure corresponds a *family of classes* :

1. Mesh classes

Construction of a mesh:	`Mesh ms("test.m");`
Output of a mesh :	`cout << ms;`
Loop over elements:	`for (ms.topElement(); (theElement=ms.getElement());)` `    cout << *theElement;`
or equivalently:	`MeshElements(ms)`
Get pointer to a node:	`Node *nd = el->getPtrNode(2);`
Creation of boundary sides:	`ms.getBoundarySides();`
Creation of all sides:	`ms.getAllSides();`
Change of unknown support:	`ms.setDOFSupport(SIDE_DOF);`

To each phase in the procedure corresponds a *family of classes* :

1. Mesh classes

Construction of a mesh:	`Mesh ms("test.m");`
Output of a mesh :	`cout << ms;`
Loop over elements:	`for (ms.topElement(); (theElement=ms.getElement());)`
	`    cout << *theElement;`
or equivalently:	`MeshElements(ms)`
Get pointer to a node:	`Node *nd = el->getPtrNode(2);`
Creation of boundary sides:	`ms.getBoundarySides();`
Creation of all sides:	`ms.getAllSides();`
Change of unknown support:	`ms.setDOFSupport(SIDE_DOF);`

To each phase in the procedure corresponds a *family of classes* :

1. Mesh classes

Construction of a mesh:	`Mesh ms("test.m");`
Output of a mesh :	`cout << ms;`
Loop over elements:	`for (ms.topElement(); (theElement=ms.getElement());)` `    cout << *theElement;`
or equivalently:	`MeshElements(ms)`
Get pointer to a node:	`Node *nd = el->getPtrNode(2);`
Creation of boundary sides:	`ms.getBoundarySides();`
Creation of all sides:	`ms.getAllSides();`
Change of unknown support:	`ms.setDOFSupport(SIDE_DOF);`

To each phase in the procedure corresponds a *family of classes* :

1. Mesh classes

Construction of a mesh: `Mesh ms("test.m");`

Output of a mesh : `cout << ms;`

Loop over elements:
```
for (ms.topElement(); (theElement=ms.getElement());)
    cout << *theElement;
```

or equivalently: `MeshElements(ms)`

Get pointer to a node: `Node *nd = el->getPtrNode(2);`

Creation of boundary sides: `ms.getBoundarySides();`

Creation of all sides: `ms.getAllSides();`

Change of unknown support: `ms.setDOFSupport(SIDE_DOF);`

To each phase in the procedure corresponds a *family of classes* :

1. Mesh classes

Construction of a mesh:	`Mesh ms("test.m");`
Output of a mesh :	`cout << ms;`
Loop over elements:	`for (ms.topElement(); (theElement=ms.getElement());)` `    cout << *theElement;`
or equivalently:	`MeshElements(ms)`
Get pointer to a node:	`Node *nd = el->getPtrNode(2);`
Creation of boundary sides:	`ms.getBoundarySides();`
Creation of all sides:	`ms.getAllSides();`
Change of unknown support:	`ms.setDOFSupport(SIDE_DOF);`

To each phase in the procedure corresponds a *family of classes* :

1. Mesh classes

Construction of a mesh:	`Mesh ms("test.m");`
Output of a mesh :	`cout << ms;`
Loop over elements:	`for (ms.topElement(); (theElement=ms.getElement());)` `cout << *theElement;`
or equivalently:	`MeshElements(ms)`
Get pointer to a node:	`Node *nd = el->getPtrNode(2);`
Creation of boundary sides:	`ms.getBoundarySides();`
Creation of all sides:	`ms.getAllSides();`
Change of unknown support:	`ms.setDOFSupport(SIDE_DOF);`

To each phase in the procedure corresponds a *family of classes* :

1. Mesh classes

Construction of a mesh:	`Mesh ms("test.m");`
Output of a mesh :	`cout << ms;`
Loop over elements:	`for (ms.topElement(); (theElement=ms.getElement());)`
	`    cout << *theElement;`
or equivalently:	`MeshElements(ms)`
Get pointer to a node:	`Node *nd = el->getPtrNode(2);`
Creation of boundary sides:	`ms.getBoundarySides();`
Creation of all sides:	`ms.getAllSides();`
Change of unknown support:	`ms.setDOFSupport(SIDE_DOF);`

To each phase in the procedure corresponds a *family of classes* :

1. Mesh classes

Construction of a mesh:	`Mesh ms("test.m");`
Output of a mesh :	`cout << ms;`
Loop over elements:	`for (ms.topElement(); (theElement=ms.getElement());)` `cout << *theElement;`
or equivalently:	`MeshElements(ms)`
Get pointer to a node:	`Node *nd = el->getPtrNode(2);`
Creation of boundary sides:	`ms.getBoundarySides();`
Creation of all sides:	`ms.getAllSides();`
Change of unknown support:	`ms.setDOFSupport(SIDE_DOF);`

2. Vector classes

A wide variety of Template classes for vectors
The template parameter is the data type for vector entries
A vector class called `Vect<T_>`.

Class `Vect<T_>` :

Construction of a vector:	`Vect<double> v(ms.getNbNodes());`
Assignment:	`v(1) = 5; v[0] = 5;`
	`v = -10;`
Other operations:	`v += w;`
	`v *= 5;`
Assembly:	`v.Assembly(el,ve);`
Euclidean norm:	`double x = v.getNorm2();`
Vector size:	`int n = v.getSize();`

2. Vector classes

A wide variety of Template classes for vectors

The template parameter is the data type for vector entries

A vector class called `Vect<T_>`.

Class `Vect<T_>` :

Construction of a vector:	`Vect<double> v(ms.getNbNodes());`
Assignment:	`v(1) = 5; v[0] = 5;`
	`v = -10;`
Other operations:	`v += w;`
	`v *= 5;`
Assembly:	`v.Assembly(el,ve);`
Euclidean norm:	`double x = v.getNorm2();`
Vector size:	`int n = v.getSize();`

2. Vector classes

A wide variety of Template classes for vectors
The template parameter is the data type for vector entries
A vector class called `Vect<T_>`.

Class `Vect<T_>` :

Construction of a vector:	`Vect<double> v(ms.getNbNodes());`
Assignment:	`v(1) = 5; v[0] = 5;`
	`v = -10;`
Other operations:	`v += w;`
	`v *= 5;`
Assembly:	`v.Assembly(el,ve);`
Euclidean norm:	`double x = v.getNorm2();`
Vector size:	`int n = v.getSize();`

2. Vector classes

A wide variety of Template classes for vectors
The template parameter is the data type for vector entries
A vector class called `Vect<T_>`.

Class `Vect<T_>` :

Construction of a vector:	`Vect<double> v(ms.getNbNodes());`
Assignment:	`v(1) = 5; v[0] = 5;`
	`v = -10;`
Other operations:	`v += w;`
	`v *= 5;`
Assembly:	`v.Assembly(el,ve);`
Euclidean norm:	`double x = v.getNorm2();`
Vector size:	`int n = v.getSize();`

2. Vector classes

A wide variety of Template classes for vectors
The template parameter is the data type for vector entries
A vector class called `Vect<T_>`.

Class `Vect<T_>` :

Construction of a vector:	`Vect<double> v(ms.getNbNodes());`
Assignment:	`v(1) = 5; v[0] = 5;`
	`v = -10;`
Other operations:	`v += w;`
	`v *= 5;`
Assembly:	`v.Assembly(el,ve);`
Euclidean norm:	`double x = v.getNorm2();`
Vector size:	`int n = v.getSize();`

2. Vector classes

A wide variety of Template classes for vectors
The template parameter is the data type for vector entries
A vector class called Vect<T_>.

Class Vect<T_> :

Construction of a vector:	`Vect<double> v(ms.getNbNodes());`
Assignment:	`v(1) = 5; v[0] = 5;`
	`v = -10;`
Other operations:	`v += w;`
	`v *= 5;`
Assembly:	`v.Assembly(el,ve);`
Euclidean norm:	`double x = v.getNorm2();`
Vector size:	`int n = v.getSize();`

2. Vector classes

A wide variety of Template classes for vectors
The template parameter is the data type for vector entries
A vector class called `Vect<T_>`.

Class `Vect<T_>` :

Construction of a vector:	`Vect<double> v(ms.getNbNodes());`
Assignment:	`v(1) = 5; v[0] = 5;`
	`v = -10;`
Other operations:	`v += w;`
	`v *= 5;`
Assembly:	`v.Assembly(el,ve);`
Euclidean norm:	`double x = v.getNorm2();`
Vector size:	`int n = v.getSize();`

2. Vector classes

A wide variety of Template classes for vectors
The template parameter is the data type for vector entries
A vector class called `Vect<T_>`.

Class `Vect<T_>` :

Construction of a vector:	`Vect<double> v(ms.getNbNodes());`
Assignment:	`v(1) = 5; v[0] = 5;`
	`v = -10;`
Other operations:	`v += w;`
	`v *= 5;`
Assembly:	`v.Assembly(el,ve);`
Euclidean norm:	`double x = v.getNorm2();`
Vector size:	`int n = v.getSize();`

2. Vector classes

A wide variety of Template classes for vectors
The template parameter is the data type for vector entries
A vector class called `Vect<T_>`.

Class `Vect<T_>` :

Construction of a vector:	`Vect<double> v(ms.getNbNodes());`
Assignment:	`v(1) = 5; v[0] = 5;`
	`v = -10;`
Other operations:	`v += w;`
	`v *= 5;`
Assembly:	`v.Assembly(el,ve);`
Euclidean norm:	`double x = v.getNorm2();`
Vector size:	`int n = v.getSize();`

2. Vector classes

A wide variety of Template classes for vectors
The template parameter is the data type for vector entries
A vector class called `Vect<T_>`.

Class `Vect<T_>` :

Construction of a vector:	`Vect<double> v(ms.getNbNodes());`
Assignment:	`v(1) = 5; v[0] = 5;`
	`v = -10;`
Other operations:	`v += w;`
	`v *= 5;`
Assembly:	`v.Assembly(el,ve);`
Euclidean norm:	`double x = v.getNorm2();`
Vector size:	`int n = v.getSize();`

2. Vector classes

A wide variety of Template classes for vectors
The template parameter is the data type for vector entries
A vector class called `Vect<T_>`.

Class `Vect<T_>` :

Construction of a vector:	`Vect<double> v(ms.getNbNodes());`
Assignment:	`v(1) = 5; v[0] = 5;`
	`v = -10;`
Other operations:	`v += w;`
	`v *= 5;`
Assembly:	`v.Assembly(el,ve);`
Euclidean norm:	`double x = v.getNorm2();`
Vector size:	`int n = v.getSize();`

2. Vector classes:

Class `NodeVect<T_>`: "Node" oriented vector

Construction of a vector:

```
NodeVect<double> v(ms);
NodeVect(Mesh &mesh, const Vect<T_> &v, int nb_dof,
         int first_dof, char *name, double time);
```

Assignment:

```
v(n,i) = 5;
```

Class `ElementVect<T_>`: "Element" oriented vector

Class `SideVect<T_>`: "Side" oriented vector

Class `LocalVect<T_,N_>`: Small size vector `LocalVect<double,4> v;`

2. Vector classes:

Class `NodeVect<T_>`: "Node" oriented vector

Construction of a vector:
```
NodeVect<double> v(ms);
NodeVect(Mesh &mesh, const Vect<T_> &v, int nb_dof,
         int first_dof, char *name, double time);
```

Assignment:
```
v(n,i) = 5;
```

Class `ElementVect<T_>`: "Element" oriented vector

Class `SideVect<T_>`: "Side" oriented vector

Class `LocalVect<T_,N_>`: Small size vector `LocalVect<double,4> v;`

2. Vector classes:

Class `NodeVect<T_>`: "Node" oriented vector

Construction of a vector:

```
NodeVect<double> v(ms);
NodeVect(Mesh &mesh, const Vect<T_> &v, int nb_dof,
         int first_dof, char *name, double time);
```

Assignment:

```
v(n,i) = 5;
```

Class `ElementVect<T_>`: "Element" oriented vector

Class `SideVect<T_>`: "Side" oriented vector

Class `LocalVect<T_,N_>`: Small size vector `LocalVect<double,4> v;`

2. Vector classes:

Class `NodeVect<T_>`: "Node" oriented vector

Construction of a vector:
```
NodeVect<double> v(ms);
NodeVect(Mesh &mesh, const Vect<T_> &v, int nb_dof,
         int first_dof, char *name, double time);
```

Assignment:
```
v(n,i) = 5;
```

Class `ElementVect<T_>`: "Element" oriented vector

Class `SideVect<T_>`: "Side" oriented vector

Class `LocalVect<T_,N_>`: Small size vector `LocalVect<double,4> v;`

2. Vector classes:

Class `NodeVect<T_>`: "Node" oriented vector

Construction of a vector:

```
NodeVect<double> v(ms);
NodeVect(Mesh &mesh, const Vect<T_> &v, int nb_dof,
         int first_dof, char *name, double time);
```

Assignment:

```
v(n,i) = 5;
```

Class `ElementVect<T_>`: "Element" oriented vector

Class `SideVect<T_>`: "Side" oriented vector

Class `LocalVect<T_,N_>`: Small size vector `LocalVect<double,4> v;`

2. Vector classes:

Class `NodeVect<T_>`: "Node" oriented vector

Construction of a vector:

```
NodeVect<double> v(ms);
NodeVect(Mesh &mesh, const Vect<T_> &v, int nb_dof,
         int first_dof, char *name, double time);
```

Assignment:

```
v(n,i) = 5;
```

Class `ElementVect<T_>`: "Element" oriented vector

Class `SideVect<T_>`: "Side" oriented vector

Class `LocalVect<T_,N_>`: Small size vector `LocalVect<double,4> v;`

3. Matrix classes:

Template classes for storage and manipulation of matrices using principal storage types:

- Dense storage: `DMatrix<T_>` and `DSMatrix<T_>`
- Skyline storage: `SkMatrix<T_>` and `SkSMatrix<T_>`
- Sparse storage: `SpMatrix<T_>`
- `TrMatrix<T_>`, `LocalMatrix<T_,NR_,NC_>`, ...

3. Matrix classes:

Template classes for storage and manipulation of matrices using principal storage types:

- Dense storage: `DMatrix<T_>` and `DSMatrix<T_>`
- Skyline storage: `SkMatrix<T_>` and `SkSMatrix<T_>`
- Sparse storage: `SpMatrix<T_>`
- `TrMatrix<T_>`, `LocalMatrix<T_,NR_,NC_>`, ...

3. Matrix classes:

Template classes for storage and manipulation of matrices using principal storage types:

- Dense storage: `DMatrix<T_>` and `DSMatrix<T_>`
- Skyline storage: `SkMatrix<T_>` and `SkSMatrix<T_>`
- Sparse storage: `SpMatrix<T_>`
- `TrMatrix<T_>`, `LocalMatrix<T_,NR_,NC_>`, ...

3. Matrix classes:

Template classes for storage and manipulation of matrices using principal storage types:

- Dense storage: `DMatrix<T_>` and `DSMatrix<T_>`
- Skyline storage: `SkMatrix<T_>` and `SkSMatrix<T_>`
- Sparse storage: `SpMatrix<T_>`
- `TrMatrix<T_>`, `LocalMatrix<T_,NR_,NC_>`, ...

3. Matrix classes:

Template classes for storage and manipulation of matrices using principal storage types:

- Dense storage: DMatrix<T_> and DSMatrix<T_>
- Skyline storage: SkMatrix<T_> and SkSMatrix<T_>
- Sparse storage: SpMatrix<T_>
- TrMatrix<T_>, LocalMatrix<T_,NR_,NC_>, ...

3. Matrix classes:

Template classes for storage and manipulation of matrices using principal storage types:

- Dense storage: DMatrix<T_> and DSMatrix<T_>
- Skyline storage: SkMatrix<T_> and SkSMatrix<T_>
- Sparse storage: SpMatrix<T_>
- TrMatrix<T_>, LocalMatrix<T_,NR_,NC_>, ...

4. Equation classes:

- An element equation is an object
- Each term of the equation is a member of the class that contributes to the left and/or the right-hand side
- **OFELI** contains a collection of classes specific to problems:
  - Laplace : Various numerical methods to solve the Laplace equation
  - Therm: Diffusion-convection problem with phase change
  - Solid: Elasticity problem
  - Fluid: Incompressible Navier-Stokes equations
  - Electromagnetics: Electromagnetic and Eddy Current problems
  - CL: Systems of Conservation Laws

4. Equation classes:

- An element equation is an object
- Each term of the equation is a member of the class that contributes to the left and/or the right-hand side
- **OFELI** contains a collection of classes specific to problems:
  - Laplace : Various numerical methods to solve the Laplace equation
  - Therm: Diffusion-convection problem with phase change
  - Solid: Elasticity problem
  - Fluid: Incompressible Navier-Stokes equations
  - Electromagnetics: Electromagnetic and Eddy Current problems
  - CL: Systems of Conservation Laws

4. Equation classes:

- An element equation is an object
- Each term of the equation is a member of the class that contributes to the left and/or the right-hand side
- **OFELI** contains a collection of classes specific to problems:
  - Laplace : Various numerical methods to solve the Laplace equation
  - Therm: Diffusion-convection problem with phase change
  - Solid: Elasticity problem
  - Fluid: Incompressible Navier-Stokes equations
  - Electromagnetics: Electromagnetic and Eddy Current problems
  - CL: Systems of Conservation Laws

4. Equation classes:

- An element equation is an object
- Each term of the equation is a member of the class that contributes to the left and/or the right-hand side
- **OFELI** contains a collection of classes specific to problems:
  - Laplace : Various numerical methods to solve the Laplace equation
  - Therm: Diffusion-convection problem with phase change
  - Solid: Elasticity problem
  - Fluid: Incompressible Navier-Stokes equations
  - Electromagnetics: Electromagnetic and Eddy Current problems
  - CL: Systems of Conservation Laws

5. Shape function classes:

To each finite element interpolation corresponds a class (e.g., 3-Node triangles (P_1): `Triang3`)
Available shape function classes: `Line2`, `Line3`, `Triang3`, `Triang6S`, `Quad4`, `Tetra4`, `Hexa8`.

6. Solvers:

OFELI contains some template functions enabling the solution of specific problems.

- Direct and iterative solvers (with preconditioners) for linear systems
- Optimization problems can be solved by using a template function. The Objective function and its gradient are given through a user defined class.

5. Shape function classes:

To each finite element interpolation corresponds a class (e.g., 3-Node triangles (P_1): `Triang3`)

Available shape function classes: `Line2`, `Line3`, `Triang3`, `Triang6S`, `Quad4`, `Tetra4`, `Hexa8`.

6. Solvers:

OFELI contains some template functions enabling the solution of specific problems.

- Direct and iterative solvers (with preconditioners) for linear systems
- Optimization problems can be solved by using a template function. The Objective function and its gradient are given through a user defined class.

5. Shape function classes:

To each finite element interpolation corresponds a class (e.g., 3-Node triangles (P_1): `Triang3`)
Available shape function classes: `Line2`, `Line3`, `Triang3`, `Triang6S`, `Quad4`, `Tetra4`, `Hexa8`.

6. Solvers:

OFELI contains some template functions enabling the solution of specific problems.

- Direct and iterative solvers (with preconditioners) for linear systems
- Optimization problems can be solved by using a template function. The Objective function and its gradient are given through a user defined class.

5. Shape function classes:

To each finite element interpolation corresponds a class (e.g., 3-Node triangles (P_1): `Triang3`)
Available shape function classes: `Line2`, `Line3`, `Triang3`, `Triang6S`, `Quad4`, `Tetra4`, `Hexa8`.

6. Solvers:

OFELI contains some template functions enabling the solution of specific problems.

- Direct and iterative solvers (with preconditioners) for linear systems
- Optimization problems can be solved by using a template function. The Objective function and its gradient are given through a user defined class.

5. Shape function classes:

To each finite element interpolation corresponds a class (e.g., 3-Node triangles (P_1): `Triang3`)
Available shape function classes: `Line2`, `Line3`, `Triang3`, `Triang6S`, `Quad4`, `Tetra4`, `Hexa8`.

6. Solvers:

OFELI contains some template functions enabling the solution of specific problems.

- Direct and iterative solvers (with preconditioners) for linear systems
- Optimization problems can be solved by using a template function. The Objective function and its gradient are given through a user defined class.

5. Shape function classes:

To each finite element interpolation corresponds a class (e.g., 3-Node triangles (P_1): `Triang3`)
Available shape function classes: `Line2`, `Line3`, `Triang3`, `Triang6S`, `Quad4`, `Tetra4`, `Hexa8`.

6. Solvers:

OFELI contains some template functions enabling the solution of specific problems.

- Direct and iterative solvers (with preconditioners) for linear systems
- Optimization problems can be solved by using a template function. The Objective function and its gradient are given through a user defined class.

5. Shape function classes:

To each finite element interpolation corresponds a class (e.g., 3-Node triangles (P_1): `Triang3`)
Available shape function classes: `Line2`, `Line3`, `Triang3`, `Triang6S`, `Quad4`, `Tetra4`, `Hexa8`.

6. Solvers:

OFELI contains some template functions enabling the solution of specific problems.

- Direct and iterative solvers (with preconditioners) for linear systems
- Optimization problems can be solved by using a template function. The Objective function and its gradient are given through a user defined class.

Handling of material properties:

- Each material is defined in a file having its name (e.g., Copper.md).
- A user can defined his own material file.
- A generic material enables giving standard properties of a material.

Handling of material properties:

- Each material is defined in a file having its name (e.g., Copper.md).
- A user can defined his own material file.
- A generic material enables giving standard properties of a material.

Handling of material properties:

- Each material is defined in a file having its name (e.g., Copper.md).
- A user can defined his own material file.
- A generic material enables giving standard properties of a material.

Handling of material properties:

- Each material is defined in a file having its name (e.g., Copper.md).
- A user can defined his own material file.
- A generic material enables giving standard properties of a material.

Data structures in OFELI

OFELI use the XML syntax for all input and output files.
All types of data are to be introduced by means of an XML tag:Various data files can be used:

- `Project`: To introduce various parameters for a main program
- `Domain`: To describe a domain via its geometric properties (for 2-D mesh generation)
- `Mesh`: To describe mesh data
- `Material`: To describe properties of a material (*i.e.* a used defined material)
- `Field`: To give any input or output field (by nodes, elements or sides).

A typical XML OFELI File

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<OFELI_File>
<info>
   <title></title>
   <date></date>
   <author></author>
</info>
...
...
</OFELI_File>
```

Data structures in OFELI

OFELI use the XML syntax for all input and output files.
All types of data are to be introduced by means of an XML tag:Various data files can be used:

`Project`:	To introduce various parameters for a main program
`Domain`:	To describe a domain via its geometric properties (for 2-D mesh generation)
`Mesh`:	To describe mesh data
`Material`:	To describe properties of a material (*i.e.* a used defined material)
`Field`:	To give any input or output field (by nodes, elements or sides).

A typical XML OFELI File

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<OFELI_File>
<info>
   <title></title>
   <date></date>
   <author></author>
</info>
...
...
</OFELI_File>
```

Data structures in OFELI

OFELI use the XML syntax for all input and output files.
All types of data are to be introduced by means of an XML tag:Various data files can be used:

`Project`:	To introduce various parameters for a main program
`Domain`:	To describe a domain via its geometric properties (for 2-D mesh generation)
`Mesh`:	To describe mesh data
`Material`:	To describe properties of a material (*i.e.* a used defined material)
`Field`:	To give any input or output field (by nodes, elements or sides).

A typical XML OFELI File

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<OFELI_File>
<info>
   <title></title>
   <date></date>
   <author></author>
</info>
...
...
</OFELI_File>
```

An example of mesh file:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<OFELI_File>
<info>
  <title></title>
  <date></date>
  <author></author>
</info>
<Mesh dim="2">
    <Nodes>
      0.00  0.00  2    1.00  0.00  0
      1.00  1.00  2    0.00  1.00  0
      0.50  0.50  1
    </Nodes>
    <Elements shape="triangle" nodes="3">
      1  2  5    2  3  5
      3  4  5    3  4  5
      4  1  5
    </Elements>
</Mesh>
</OFELI_File>
```

The OFELI package

The **OFELI** library is free and under the *GPL* license (GNU General Public License). It is available on the web site http://www.ofeli.net.
The package contains:

1. Source files of the library (kernel + problem dependent classes: Laplace, Thermics, Solid mechanics, Fluid dynamics, Electromagnetics).
2. Documentation in HTML and PDF. The documentation is automatically generated by doxygen.
3. A tutorial with examples of finite element codes with increasing difficulty
4. Demos: Multiple finite element programs
5. A mesh generator for 2–D meshes
6. Utility programs: conversion

The OFELI package

The **OFELI** library is free and under the *GPL* license (GNU General Public License). It is available on the web site http://www.ofeli.net.
The package contains:

1. Source files of the library (kernel + problem dependent classes: Laplace, Thermics, Solid mechanics, Fluid dynamics, Electromagnetics).
2. Documentation in HTML and PDF. The documentation is automatically generated by doxygen.
3. A tutorial with examples of finite element codes with increasing difficulty
4. Demos: Multiple finite element programs
5. A mesh generator for 2–D meshes
6. Utility programs: conversion

The OFELI package

The **OFELI** library is free and under the *GPL* license (GNU General Public License). It is available on the web site http://www.ofeli.net.
The package contains:

1. Source files of the library (kernel + problem dependent classes: Laplace, Thermics, Solid mechanics, Fluid dynamics, Electromagnetics).
2. Documentation in HTML and PDF. The documentation is automatically generated by doxygen.
3. A tutorial with examples of finite element codes with increasing difficulty
4. Demos: Multiple finite element programs
5. A mesh generator for 2–D meshes
6. Utility programs: conversion

The OFELI package

The **OFELI** library is free and under the *GPL* license (GNU General Public License). It is available on the web site http://www.ofeli.net.
The package contains:

1. Source files of the library (kernel + problem dependent classes: Laplace, Thermics, Solid mechanics, Fluid dynamics, Electromagnetics).
2. Documentation in HTML and PDF. The documentation is automatically generated by doxygen.
3. A tutorial with examples of finite element codes with increasing difficulty
4. Demos: Multiple finite element programs
5. A mesh generator for 2–D meshes
6. Utility programs: conversion

The OFELI package

The **OFELI** library is free and under the *GPL* license (GNU General Public License). It is available on the web site http://www.ofeli.net.
The package contains:

1. Source files of the library (kernel + problem dependent classes: Laplace, Thermics, Solid mechanics, Fluid dynamics, Electromagnetics).
2. Documentation in HTML and PDF. The documentation is automatically generated by doxygen.
3. A tutorial with examples of finite element codes with increasing difficulty
4. Demos: Multiple finite element programs
5. A mesh generator for 2–D meshes
6. Utility programs: conversion

The OFELI package

The **OFELI** library is free and under the *GPL* license (GNU General Public License). It is available on the web site http://www.ofeli.net.
The package contains:

1. Source files of the library (kernel + problem dependent classes: Laplace, Thermics, Solid mechanics, Fluid dynamics, Electromagnetics).
2. Documentation in HTML and PDF. The documentation is automatically generated by doxygen.
3. A tutorial with examples of finite element codes with increasing difficulty
4. Demos: Multiple finite element programs
5. A mesh generator for 2–D meshes
6. Utility programs: conversion

The OFELI package

The **OFELI** library is free and under the *GPL* license (GNU General Public License). It is available on the web site http://www.ofeli.net.
The package contains:

1. Source files of the library (kernel + problem dependent classes: Laplace, Thermics, Solid mechanics, Fluid dynamics, Electromagnetics).
2. Documentation in HTML and PDF. The documentation is automatically generated by doxygen.
3. A tutorial with examples of finite element codes with increasing difficulty
4. Demos: Multiple finite element programs
5. A mesh generator for 2–D meshes
6. Utility programs: conversion

The Demos

These are partitioned in physical problems:

- Therm: Diffusion–Convection (steady state and transient)
- Solid: Linear Elasticity 2–D et 3–D
- Fluid: Incompressible Navier-Stokes equations
- Electromagnetics: Helmholtz and Eddy Current equations s in 2–D
- CL: Systems of Conservation Laws

The Demos

These are partitioned in physical problems:

- Therm: Diffusion–Convection (steady state and transient)
- Solid: Linear Elasticity 2–D et 3–D
- Fluid: Incompressible Navier-Stokes equations
- Electromagnetics: Helmholtz and Eddy Current equations s in 2–D
- CL: Systems of Conservation Laws

The Demos

These are partitioned in physical problems:

- Therm: Diffusion–Convection (steady state and transient)
- Solid: Linear Elasticity 2–D et 3–D
- Fluid: Incompressible Navier-Stokes equations
- Electromagnetics: Helmholtz and Eddy Current equations s in 2–D
- CL: Systems of Conservation Laws

The Demos

These are partitioned in physical problems:

- Therm: Diffusion–Convection (steady state and transient)
- Solid: Linear Elasticity 2–D et 3–D
- Fluid: Incompressible Navier-Stokes equations
- Electromagnetics: Helmholtz and Eddy Current equations s in 2–D
- CL: Systems of Conservation Laws

The Demos

These are partitioned in physical problems:

- Therm: Diffusion–Convection (steady state and transient)
- Solid: Linear Elasticity 2–D et 3–D
- Fluid: Incompressible Navier-Stokes equations
- Electromagnetics: Helmholtz and Eddy Current equations s in 2–D
- CL: Systems of Conservation Laws

The Demos

These are partitioned in physical problems:

- Therm: Diffusion–Convection (steady state and transient)
- Solid: Linear Elasticity 2–D et 3–D
- Fluid: Incompressible Navier-Stokes equations
- Electromagnetics: Helmholtz and Eddy Current equations s in 2–D
- CL: Systems of Conservation Laws

The utilities

- Conversion of mesh files issued from: EasyMesh, EMC2, BAMG, Netgen, Gmsh, Gambit, Triangle and Matlab
- Conversion of mesh and output files issued from: Matlab, Tecplot, Gnuplot and Paraview (VTK)
- Generation of 2-D meshes by BAMG ou Triangle

The utilities

- Conversion of mesh files issued from: EasyMesh, EMC2, BAMG, Netgen, Gmsh, Gambit, Triangle and Matlab
- Conversion of mesh and output files issued from: Matlab, Tecplot, Gnuplot and Paraview (VTK)
- Generation of 2-D meshes by BAMG ou Triangle

The utilities

- Conversion of mesh files issued from: EasyMesh, EMC2, BAMG, Netgen, Gmsh, Gambit, Triangle and Matlab
- Conversion of mesh and output files issued from: Matlab, Tecplot, Gnuplot and Paraview (VTK)
- Generation of 2-D meshes by BAMG ou Triangle

The utilities

- Conversion of mesh files issued from: EasyMesh, EMC2, BAMG, Netgen, Gmsh, Gambit, Triangle and Matlab
- Conversion of mesh and output files issued from: Matlab, Tecplot, Gnuplot and Paraview (VTK)
- Generation of 2-D meshes by BAMG ou Triangle

Current and future developments

- Level Set and Fast Marching methods in 2–D and 3–D
- Efficient solvers for coupled equations
- Implementation of domain decomposition methods (coupling with Metis)
- Implementation of mixed finite volumes for elliptic problems (C. Chainais)

Current and future developments

- Level Set and Fast Marching methods in 2–D and 3–D
- Efficient solvers for coupled equations
- Implementation of domain decomposition methods (coupling with Metis)
- Implementation of mixed finite volumes for elliptic problems (C. Chainais)

Current and future developments

- Level Set and Fast Marching methods in 2–D and 3–D
- Efficient solvers for coupled equations
- Implementation of domain decomposition methods (coupling with Metis)
- Implementation of mixed finite volumes for elliptic problems (C. Chainais)

Current and future developments

- Level Set and Fast Marching methods in 2–D and 3–D
- Efficient solvers for coupled equations
- Implementation of domain decomposition methods (coupling with Metis)
- Implementation of mixed finite volumes for elliptic problems (C. Chainais)

Current and future developments

- Level Set and Fast Marching methods in 2–D and 3–D
- Efficient solvers for coupled equations
- Implementation of domain decomposition methods (coupling with Metis)
- Implementation of mixed finite volumes for elliptic problems (C. Chainais)

Current and future developments

- Level Set and Fast Marching methods in 2–D and 3–D
- Efficient solvers for coupled equations
- Implementation of domain decomposition methods (coupling with Metis)
- Implementation of mixed finite volumes for elliptic problems (C. Chainais)