A presentation of the library OFELI

Rachid Touzani

Université Blaise Pascal
Clermont-Ferrand, France

Outlook

1. What is **OFELI**?
2. Objects in **OFELI**
3. A finite element code example
4. The library structure
5. The **OFELI** package
6. Recent and future developments
7. Example Codes

What is **OFELI** ?

- OFELI: **O**bject **F**inite **E**lement **LI**brary
- The OFELI library is a collection of C++ classes and utilities enabling the construction of finite element codes.
- It provides a variety of prototype 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 and utilities enabling the construction of finite element codes.
- It provides a variety of prototype 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 and utilities enabling the construction of finite element codes.
- It provides a variety of prototype 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 and utilities enabling the construction of finite element codes.
- It provides a variety of prototype 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 and utilities enabling the construction of finite element codes.
- It provides a variety of prototype 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 and utilities enabling the construction of finite element codes.
- It provides a variety of prototype 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 and utilities enabling the construction of finite element codes.
- It provides a variety of prototype 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 and utilities enabling the construction of finite element codes.
- It provides a variety of prototype 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 and utilities enabling the construction of finite element codes.
- It provides a variety of prototype 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 and utilities enabling the construction of finite element codes.
- It provides a variety of prototype 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`, . . .)

Objects in C++

- C++ is a language to manipulate objects rather than data.
- Classes are structures that may contain data and functions to handle them. Objects are instances of classes.
- Example 1: Integer numbers can be considered as instances (objects) of a class called integer
- Example 2: A node can be considered as a class. Its members are for example: the node's label, coordinates, ...

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, ...), ...

Objects in C++

- C++ is a language to manipulate objects rather than data.
- Classes are structures that may contain data and functions to handle them. Objects are instances of classes.
- Example 1: Integer numbers can be considered as instances (objects) of a class called integer
- Example 2: A node can be considered as a class. Its members are for example: the node's label, coordinates, ...

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, ...), ...

Objects in C++

- C++ is a language to manipulate objects rather than data.
- Classes are structures that may contain data and functions to handle them. Objects are instances of classes.
- Example 1: Integer numbers can be considered as instances (objects) of a class called integer
- Example 2: A node can be considered as a class. Its members are for example: the node's label, coordinates, . . .

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, . . .), . . .

Objects in C++

- C++ is a language to manipulate objects rather than data.
- Classes are structures that may contain data and functions to handle them. Objects are instances of classes.
- Example 1: Integer numbers can be considered as instances (objects) of a class called integer
- Example 2: A node can be considered as a class. Its members are for example: the node's label, coordinates, . . .

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, . . .), . . .

Objects in C++

- C++ is a language to manipulate objects rather than data.
- Classes are structures that may contain data and functions to handle them. Objects are instances of classes.
- Example 1: Integer numbers can be considered as instances (objects) of a class called integer
- Example 2: A node can be considered as a class. Its members are for example: the node's label, coordinates, ...

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, ...), ...

Objects in C++

- C++ is a language to manipulate objects rather than data.
- Classes are structures that may contain data and functions to handle them. Objects are instances of classes.
- Example 1: Integer numbers can be considered as instances (objects) of a class called integer
- Example 2: A node can be considered as a class. Its members are for example: the node's label, coordinates, ...

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 (P_1 Finite Elements)

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

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:

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

Matrix Formulation (P_1 Finite Elements)

$$\mathbf{A}\mathbf{u} = \mathbf{b}$$

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:

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

Matrix Formulation (P_1 Finite Elements)

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

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:

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

Matrix Formulation (P_1 Finite Elements)

$$\mathbf{A}\mathbf{u} = \mathbf{b}$$

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

   Element *el;
   for (ms.topElement(); (el=ms.getElement());) {
      DC2DT3 eq(el);
      eq.Diffusion();
      a.Assembly(el,eq.A());
   }

   a.Prescribe(ms,b,bc);
   A.FactorAndSolve(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

   Element *el;
   for (ms.topElement(); (el=ms.getElement());) {
      DC2DT3 eq(el);
      eq.Diffusion();
      a.Assembly(el,eq.A());
   }

   a.Prescribe(ms,b,bc);
   A.FactorAndSolve(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

   Element *el;
   for (ms.topElement(); (el=ms.getElement());) {
      DC2DT3 eq(el);
      eq.Diffusion();
      a.Assembly(el,eq.A());
   }

   a.Prescribe(ms,b,bc);
   A.FactorAndSolve(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

   Element *el;
   for (ms.topElement(); (el=ms.getElement());) {
      DC2DT3 eq(el);
      eq.Diffusion();
      a.Assembly(el,eq.A());
   }

   a.Prescribe(ms,b,bc);
   A.FactorAndSolve(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

   Element *el;
   for (ms.topElement(); (el=ms.getElement());) {
      DC2DT3 eq(el);
      eq.Diffusion();
      a.Assembly(el,eq.A());
   }

   a.Prescribe(ms,b,bc);
   A.FactorAndSolve(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

   Element *el;
   for (ms.topElement(); (el=ms.getElement());) {
      DC2DT3 eq(el);
      eq.Diffusion();
      a.Assembly(el,eq.A());
   }

   a.Prescribe(ms,b,bc);
   A.FactorAndSolve(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

   Element *el;
   for (ms.topElement(); (el=ms.getElement());) {
      DC2DT3 eq(el);
      eq.Diffusion();
      a.Assembly(el,eq.A());
   }

   a.Prescribe(ms,b,bc);
   A.FactorAndSolve(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

   Element *el;
   for (ms.topElement(); (el=ms.getElement());) {
      DC2DT3 eq(el);
      eq.Diffusion();
      a.Assembly(el,eq.A());
   }

   a.Prescribe(ms,b,bc);
   A.FactorAndSolve(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

   Element *el;
   for (ms.topElement(); (el=ms.getElement());) {
      DC2DT3 eq(el);
      eq.Diffusion();
      a.Assembly(el,eq.A());
   }

   a.Prescribe(ms,b,bc);
   A.FactorAndSolve(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

   Element *el;
   for (ms.topElement(); (el=ms.getElement());) {
      DC2DT3 eq(el);
      eq.Diffusion();
      a.Assembly(el,eq.A());
   }

   a.Prescribe(ms,b,bc);
   A.FactorAndSolve(b);

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

Classes in OFELI

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:	`Element *el;`
	`for (ms.topElement(); (el=ms.getElement());)`
	`    cout << *el;`
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(ELEMENT_DOF);`

Classes in OFELI

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:	`Element *el;`
	`for (ms.topElement(); (el=ms.getElement());)`
	`cout << *el;`
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(ELEMENT_DOF);`

Classes in OFELI

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:

```
Element *el;
for (ms.topElement(); (el=ms.getElement());)
    cout << *el;
```

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(ELEMENT_DOF);`

Classes in OFELI

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:	`Element *el;`
	`for (ms.topElement(); (el=ms.getElement());)`
	`    cout << *el;`

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(ELEMENT_DOF);`

Classes in OFELI

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:
```
Element *el;
for (ms.topElement(); (el=ms.getElement());)
    cout << *el;
```
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(ELEMENT_DOF);
```

Classes in OFELI

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:

```
Element *el;
for (ms.topElement(); (el=ms.getElement());)
    cout << *el;
```

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(ELEMENT_DOF);`

Classes in OFELI

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:	`Element *el;`
	`for (ms.topElement(); (el=ms.getElement());)`
	`    cout << *el;`

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(ELEMENT_DOF);`

Classes in OFELI

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:	`Element *el;`
	`for (ms.topElement(); (el=ms.getElement());)`
	`    cout << *el;`
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(ELEMENT_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();`

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: Eddy Current problems

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: Eddy Current problems

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: Eddy Current problems

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: Eddy Current problems

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: Eddy Current problems

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: Eddy Current problems

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: Eddy Current problems

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: Eddy Current problems

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: Eddy Current problems

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: Eddy Current problems

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.

Data structures in OFELI

Mesh file:

- Mesh files must be in the MDF format (*Mesh Data File*).
- Some utility function enable converting from and to (gmsh,BAMG, Easymesh, Gnuplot, Matlab, Triangle, Gambit, Tecplot, . . .) formats.
- An XML file format will be soon available.

Data structures in OFELI

Mesh file:

- Mesh files must be in the MDF format (*Mesh Data File*).
- Some utility function enable converting from and to (gmsh,BAMG, Easymesh, Gnuplot, Matlab, Triangle, Gambit, Tecplot, . . .) formats.
- An XML file format will be soon available.

Data structures in OFELI

Mesh file:

- Mesh files must be in the MDF format (*Mesh Data File*).
- Some utility function enable converting from and to (gmsh,BAMG, Easymesh, Gnuplot, Matlab, Triangle, Gambit, Tecplot, . . .) formats.
- An XML file format will be soon available.

Data structures in OFELI

Mesh file:

- Mesh files must be in the MDF format (*Mesh Data File*).
- Some utility function enable converting from and to (gmsh,BAMG, Easymesh, Gnuplot, Matlab, Triangle, Gambit, Tecplot, . . .) formats.
- An XML file format will be soon available.

Field files

- Field data and result can be stored in the format FDF (*Field Data File*).
- Utility programs enable converting from and to the formats: Tecplot, Gnuplot, Matlab, ...
- An XML file format is being developed and will be available soon

Parameter files

A file enabling giving parameters of a program (Data file names, values of parameters, ...) in the format IPF (*Input Parameter File*).

Field files

- Field data and result can be stored in the format FDF (*Field Data File*).
- Utility programs enable converting from and to the formats: Tecplot, Gnuplot, Matlab, . . .
- An XML file format is being developed and will be available soon

Parameter files

A file enabling giving parameters of a program (Data file names, values of parameters, . . .) in the format IPF (*Input Parameter File*).

Field files

- Field data and result can be stored in the format FDF (*Field Data File*).
- Utility programs enable converting from and to the formats: Tecplot, Gnuplot, Matlab, ...
- An XML file format is being developed and will be available soon

Parameter files

A file enabling giving parameters of a program (Data file names, values of parameters, ...) in the format IPF (*Input Parameter File*).

Field files

- Field data and result can be stored in the format FDF (*Field Data File*).
- Utility programs enable converting from and to the formats: Tecplot, Gnuplot, Matlab, ...
- An XML file format is being developed and will be available soon

Parameter files

A file enabling giving parameters of a program (Data file names, values of parameters, ...) in the format IPF (*Input Parameter File*).

Field files

- Field data and result can be stored in the format FDF (*Field Data File*).
- Utility programs enable converting from and to the formats: Tecplot, Gnuplot, Matlab, ...
- An XML file format is being developed and will be available soon

Parameter files

A file enabling giving parameters of a program (Data file names, values of parameters, ...) in the format IPF (*Input Parameter 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 (The PDF reference guide is more than 600 pages).
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 (The PDF reference guide is more than 600 pages).
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: Eddy Currents in 2–D

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: Eddy Currents in 2–D

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: Eddy Currents in 2–D

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: Eddy Currents in 2–D

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: Eddy Currents in 2–D

The utilities

- Conversion de fichiers de maillage provenant de : EasyMesh, EMC2, BAMG, Netgen, Gmsh, Gambit, Triangle et Matlab
- Conversion de fichiers de maillage et de résultats vers : Matlab, Tecplot, Gnuplot et Paraview (VTK)
- A 2-D mesh generator using BAMG or Triangle

The utilities

- Conversion de fichiers de maillage provenant de : EasyMesh, EMC2, BAMG, Netgen, Gmsh, Gambit, Triangle et Matlab
- Conversion de fichiers de maillage et de résultats vers : Matlab, Tecplot, Gnuplot et Paraview (VTK)
- A 2-D mesh generator using BAMG or Triangle

The utilities

- Conversion de fichiers de maillage provenant de : EasyMesh, EMC2, BAMG, Netgen, Gmsh, Gambit, Triangle et Matlab
- Conversion de fichiers de maillage et de résultats vers : Matlab, Tecplot, Gnuplot et Paraview (VTK)
- A 2-D mesh generator using BAMG or Triangle

The utilities

- Conversion de fichiers de maillage provenant de : EasyMesh, EMC2, BAMG, Netgen, Gmsh, Gambit, Triangle et Matlab
- Conversion de fichiers de maillage et de résultats vers : Matlab, Tecplot, Gnuplot et Paraview (VTK)
- A 2-D mesh generator using BAMG or Triangle

Recent and future developments

- Level Set and Fast Marching methods in 2–D and 3–D
- Solvers for coupled equations
- Incorporation de solvers for conservation laws using finite volumes (S. Clain, V. Clauzon)
- Implementation of domain decomposition methods (coupling with Metis)
- Implementation of mixed finite volumes for elliptic problems (C. Chainais)
- Solvers for shape optimization problems (A. Kooli)

Recent and future developments

- Level Set and Fast Marching methods in 2–D and 3–D
- Solvers for coupled equations
- Incorporation de solvers for conservation laws using finite volumes (S. Clain, V. Clauzon)
- Implementation of domain decomposition methods (coupling with Metis)
- Implementation of mixed finite volumes for elliptic problems (C. Chainais)
- Solvers for shape optimization problems (A. Kooli)

Recent and future developments

- Level Set and Fast Marching methods in 2–D and 3–D
- Solvers for coupled equations
- Incorporation de solvers for conservation laws using finite volumes (S. Clain, V. Clauzon)
- Implementation of domain decomposition methods (coupling with Metis)
- Implementation of mixed finite volumes for elliptic problems (C. Chainais)
- Solvers for shape optimization problems (A. Kooli)

Recent and future developments

- Level Set and Fast Marching methods in 2–D and 3–D
- Solvers for coupled equations
- Incorporation de solvers for conservation laws using finite volumes (S. Clain, V. Clauzon)
- Implementation of domain decomposition methods (coupling with Metis)
- Implementation of mixed finite volumes for elliptic problems (C. Chainais)
- Solvers for shape optimization problems (A. Kooli)

Recent and future developments

- Level Set and Fast Marching methods in 2–D and 3–D
- Solvers for coupled equations
- Incorporation de solvers for conservation laws using finite volumes (S. Clain, V. Clauzon)
- Implementation of domain decomposition methods (coupling with Metis)
- Implementation of mixed finite volumes for elliptic problems (C. Chainais)
- Solvers for shape optimization problems (A. Kooli)

Recent and future developments

- Level Set and Fast Marching methods in 2–D and 3–D
- Solvers for coupled equations
- Incorporation de solvers for conservation laws using finite volumes (S. Clain, V. Clauzon)
- Implementation of domain decomposition methods (coupling with Metis)
- Implementation of mixed finite volumes for elliptic problems (C. Chainais)
- Solvers for shape optimization problems (A. Kooli)

Recent and future developments

- Level Set and Fast Marching methods in 2–D and 3–D
- Solvers for coupled equations
- Incorporation de solvers for conservation laws using finite volumes (S. Clain, V. Clauzon)
- Implementation of domain decomposition methods (coupling with Metis)
- Implementation of mixed finite volumes for elliptic problems (C. Chainais)
- Solvers for shape optimization problems (A. Kooli)

Example 1: A 1-D Problem

```
double Lmin=0, Lmax=1;
int N = 20;
double f(double x);
Mesh ms(Lmin,Lmax,N);

TrMatrix<double> a(N-1);
Vect<double> b(N-1);
double h = (Lmax-Lmin)/double(N);

for (int i=2; i<N-1; i++) {
   double x = ms.getPtrNode(i)->getCoord(1);
   a(i,i) = 2./h;
   a(i,i+1) = -1./h;
   a(i,i-1) = -1./h;
   b(i) = f(x)*h;
}
a(1,1) = 2./h;
a(1,2) = -1./h;
a(N-1,N-2) = -1./h;
a(N-1,N-1) = 2./h;
```

Example 1: A 1-D Problem

Go to Example 2

```
double Lmin=0, Lmax=1;
int N = 20;
double f(double x);
Mesh ms(Lmin,Lmax,N);

TrMatrix<double> a(N-1);
Vect<double> b(N-1);
double h = (Lmax-Lmin)/double(N);

for (int i=2; i<N-1; i++) {
   double x = ms.getPtrNode(i)->getCoord(1);
   a(i,i) = 2./h;
   a(i,i+1) = -1./h;
   a(i,i-1) = -1./h;
   b(i) = f(x)*h;
}
a(1,1) = 2./h;
a(1,2) = -1./h;
a(N-1,N-2) = -1./h;
a(N-1,N-1) = 2./h;
```

Example 1: A 1-D Problem

Go to Example 2

```
double Lmin=0, Lmax=1;
int N = 20;
double f(double x);
Mesh ms(Lmin,Lmax,N);

TrMatrix<double> a(N-1);
Vect<double> b(N-1);
double h = (Lmax-Lmin)/double(N);

for (int i=2; i<N-1; i++) {
   double x = ms.getPtrNode(i)->getCoord(1);
   a(i,i) = 2./h;
   a(i,i+1) = -1./h;
   a(i,i-1) = -1./h;
   b(i) = f(x)*h;
}
a(1,1) = 2./h;
a(1,2) = -1./h;
a(N-1,N-2) = -1./h;
a(N-1,N-1) = 2./h;
```

Example 1: A 1-D Problem

Go to Example 2

```
double Lmin=0, Lmax=1;
int N = 20;
double f(double x);
Mesh ms(Lmin,Lmax,N);

TrMatrix<double> a(N-1);
Vect<double> b(N-1);
double h = (Lmax-Lmin)/double(N);

for (int i=2; i<N-1; i++) {
   double x = ms.getPtrNode(i)->getCoord(1);
   a(i,i) = 2./h;
   a(i,i+1) = -1./h;
   a(i,i-1) = -1./h;
   b(i) = f(x)*h;
}
a(1,1) = 2./h;
a(1,2) = -1./h;
a(N-1,N-2) = -1./h;
a(N-1,N-1) = 2./h;
```

Example 1: A 1-D Problem

Go to Example 2

```
double Lmin=0, Lmax=1;
int N = 20;
double f(double x);
Mesh ms(Lmin,Lmax,N);

TrMatrix<double> a(N-1);
Vect<double> b(N-1);
double h = (Lmax-Lmin)/double(N);

for (int i=2; i<N-1; i++) {
   double x = ms.getPtrNode(i)->getCoord(1);
   a(i,i) = 2./h;
   a(i,i+1) = -1./h;
   a(i,i-1) = -1./h;
   b(i) = f(x)*h;
}
a(1,1) = 2./h;
a(1,2) = -1./h;
a(N-1,N-2) = -1./h;
a(N-1,N-1) = 2./h;
```

Example 1: A 1-D Problem

Go to Example 2

```
double Lmin=0, Lmax=1;
int N = 20;
double f(double x);
Mesh ms(Lmin,Lmax,N);

TrMatrix<double> a(N-1);
Vect<double> b(N-1);
double h = (Lmax-Lmin)/double(N);

for (int i=2; i<N-1; i++) {
   double x = ms.getPtrNode(i)->getCoord(1);
   a(i,i) = 2./h;
   a(i,i+1) = -1./h;
   a(i,i-1) = -1./h;
   b(i) = f(x)*h;
}
a(1,1) = 2./h;
a(1,2) = -1./h;
a(N-1,N-2) = -1./h;
a(N-1,N-1) = 2./h;
a.Solve(b);
```

Example 2: A *Black Box* Code (1/2)

Go To Example 3

A Black Box Finite Element Code:
Diffusion-Convection Equation.

```
Mesh ms(data.getMeshFile(),true);
VDF vdf(ms,data.getDataFile());

DC2DT3 eq;
eq.setMesh(ms);
SpMatrix<double> A;
eq.setMatrix(A);

Vect<double> u(ms.getNbDOF());
eq.setInput(SOLUTION,u);

Vect<double> bc(ms.getNbDOF());
vdf.Get(BOUNDARY_CONDITION,bc);
eq.setInput(BOUNDARY_CONDITION,bc);
```

Example 2: A *Black Box* Code (1/2)

Go To Example 3

A Black Box Finite Element Code:
Diffusion-Convection Equation.

```
Mesh ms(data.getMeshFile(),true);
VDF vdf(ms,data.getDataFile());

DC2DT3 eq;
eq.setMesh(ms);
SpMatrix<double> A;
eq.setMatrix(A);

Vect<double> u(ms.getNbDOF());
eq.setInput(SOLUTION,u);

Vect<double> bc(ms.getNbDOF());
vdf.Get(BOUNDARY_CONDITION,bc);
eq.setInput(BOUNDARY_CONDITION,bc);
```

Example 2: A *Black Box* Code (1/2)

Go To Example 3

A Black Box Finite Element Code:
Diffusion-Convection Equation.

```
Mesh ms(data.getMeshFile(),true);
VDF vdf(ms,data.getDataFile());

DC2DT3 eq;
eq.setMesh(ms);
SpMatrix<double> A;
eq.setMatrix(A);

Vect<double> u(ms.getNbDOF());
eq.setInput(SOLUTION,u);

Vect<double> bc(ms.getNbDOF());
vdf.Get(BOUNDARY_CONDITION,bc);
eq.setInput(BOUNDARY_CONDITION,bc);
```

Example 2: A *Black Box* Code (1/2)

Go To Example 3

A Black Box Finite Element Code:
Diffusion-Convection Equation.

```
Mesh ms(data.getMeshFile(),true);
VDF vdf(ms,data.getDataFile());

DC2DT3 eq;
eq.setMesh(ms);
SpMatrix<double> A;
eq.setMatrix(A);

Vect<double> u(ms.getNbDOF());
eq.setInput(SOLUTION,u);

Vect<double> bc(ms.getNbDOF());
vdf.Get(BOUNDARY_CONDITION,bc);
eq.setInput(BOUNDARY_CONDITION,bc);
```

Example 2: A *Black Box* Code (1/2)

Go To Example 3

A Black Box Finite Element Code:
Diffusion-Convection Equation.

```
Mesh ms(data.getMeshFile(),true);
VDF vdf(ms,data.getDataFile());

DC2DT3 eq;
eq.setMesh(ms);
SpMatrix<double> A;
eq.setMatrix(A);

Vect<double> u(ms.getNbDOF());
eq.setInput(SOLUTION,u);

Vect<double> bc(ms.getNbDOF());
vdf.Get(BOUNDARY_CONDITION,bc);
eq.setInput(BOUNDARY_CONDITION,bc);
```

Example 2: A *Black Box* Code (2/2)

```
  Vect<double> body_f(ms.getNbDOF());
  vdf.Get(SOURCE,body_f);
  eq.setInput(SOURCE,body_f);

  NodeVect<double> v(ms.getDim());
  FDF ff(data.getAuxFile(1),FDF_READ);
  ff.Get(v);
  eq.setInput(VELOCITY,v.getVect());

  eq.setTerms(DIFFUSION|CONVECTION);

// Solve options
  eq.getLinearSolver().setSolver(GMRES_SOLVER);
  eq.getLinearSolver().setPreconditioner(ILU_PREC);

// Formation and solution of the linear system
  eq.run();

  cout << u;
  if (data.getSave()) {
    FDF pf(data.getSaveFile(),FDF_WRITE);
    pf.Put(u);
  }
```

Example 2: A *Black Box* Code (2/2)

```
  Vect<double> body_f(ms.getNbDOF());
  vdf.Get(SOURCE,body_f);
  eq.setInput(SOURCE,body_f);

  NodeVect<double> v(ms.getDim());
  FDF ff(data.getAuxFile(1),FDF_READ);
  ff.Get(v);
  eq.setInput(VELOCITY,v.getVect());

  eq.setTerms(DIFFUSION|CONVECTION);

// Solve options
  eq.getLinearSolver().setSolver(GMRES_SOLVER);
  eq.getLinearSolver().setPreconditioner(ILU_PREC);

// Formation and solution of the linear system
  eq.run();

  cout << u;
  if (data.getSave()) {
    FDF pf(data.getSaveFile(),FDF_WRITE);
    pf.Put(u);
  }
```

Example 2: A *Black Box* Code (2/2)

```
  Vect<double> body_f(ms.getNbDOF());
  vdf.Get(SOURCE,body_f);
  eq.setInput(SOURCE,body_f);

  NodeVect<double> v(ms.getDim());
  FDF ff(data.getAuxFile(1),FDF_READ);
  ff.Get(v);
  eq.setInput(VELOCITY,v.getVect());

  eq.setTerms(DIFFUSION|CONVECTION);

// Solve options
  eq.getLinearSolver().setSolver(GMRES_SOLVER);
  eq.getLinearSolver().setPreconditioner(ILU_PREC);

// Formation and solution of the linear system
  eq.run();

  cout << u;
  if (data.getSave()) {
    FDF pf(data.getSaveFile(),FDF_WRITE);
    pf.Put(u);
  }
```

Example 2: A *Black Box* Code (2/2)

```
   Vect<double> body_f(ms.getNbDOF());
   vdf.Get(SOURCE,body_f);
   eq.setInput(SOURCE,body_f);

   NodeVect<double> v(ms.getDim());
   FDF ff(data.getAuxFile(1),FDF_READ);
   ff.Get(v);
   eq.setInput(VELOCITY,v.getVect());

   eq.setTerms(DIFFUSION|CONVECTION);

// Solve options
   eq.getLinearSolver().setSolver(GMRES_SOLVER);
   eq.getLinearSolver().setPreconditioner(ILU_PREC);

// Formation and solution of the linear system
   eq.run();

   cout << u;
   if (data.getSave()) {
      FDF pf(data.getSaveFile(),FDF_WRITE);
      pf.Put(u);
   }
```

Example 2: A *Black Box* Code (2/2)

```
   Vect<double> body_f(ms.getNbDOF());
   vdf.Get(SOURCE,body_f);
   eq.setInput(SOURCE,body_f);

   NodeVect<double> v(ms.getDim());
   FDF ff(data.getAuxFile(1),FDF_READ);
   ff.Get(v);
   eq.setInput(VELOCITY,v.getVect());

   eq.setTerms(DIFFUSION|CONVECTION);

// Solve options
   eq.getLinearSolver().setSolver(GMRES_SOLVER);
   eq.getLinearSolver().setPreconditioner(ILU_PREC);

// Formation and solution of the linear system
   eq.run();

   cout << u;
   if (data.getSave()) {
      FDF pf(data.getSaveFile(),FDF_WRITE);
      pf.Put(u);
   }
```

Example 2: A *Black Box* Code (2/2)

```
   Vect<double> body_f(ms.getNbDOF());
   vdf.Get(SOURCE,body_f);
   eq.setInput(SOURCE,body_f);

   NodeVect<double> v(ms.getDim());
   FDF ff(data.getAuxFile(1),FDF_READ);
   ff.Get(v);
   eq.setInput(VELOCITY,v.getVect());

   eq.setTerms(DIFFUSION|CONVECTION);

// Solve options
   eq.getLinearSolver().setSolver(GMRES_SOLVER);
   eq.getLinearSolver().setPreconditioner(ILU_PREC);

// Formation and solution of the linear system
   eq.run();

   cout << u;
   if (data.getSave()) {
      FDF pf(data.getSaveFile(),FDF_WRITE);
      pf.Put(u);
   }
```

Example 3: An optimization Problem (1/3)

Go To Example 4

Consider the following problem:

$$\mathbf{u} \in \mathscr{V};\ W(\mathbf{u}) = \inf_{\mathbf{v} \in \mathscr{V}} W(\mathbf{v})$$

where

$$\mathscr{V} := \{\mathbf{v} \in \mathbb{R}^N;\ a_i \leq v_i \leq b_i,\ 1 \leq i \leq N\}$$

To solve this problem with **OFELI**, we write a C++ code:

Example 3: An optimization Problem (1/3)

Go To Example 4

Consider the following problem:

$$\mathbf{u} \in \mathscr{V};\ W(\mathbf{u}) = \inf_{\mathbf{v} \in \mathscr{V}} W(\mathbf{v})$$

where

$$\mathscr{V} := \{\mathbf{v} \in \mathbb{R}^N;\ a_i \leq v_i \leq b_i,\ 1 \leq i \leq N\}$$

To solve this problem with **OFELI**, we write a C++ code:

Example 3: An optimization Problem (1/3)

Go To Example 4

Consider the following problem:

$$\mathbf{u} \in \mathscr{V};\ \ W(\mathbf{u}) = \inf_{\mathbf{v} \in \mathscr{V}} W(\mathbf{v})$$

where

$$\mathscr{V} := \{\mathbf{v} \in \mathbb{R}^N;\ a_i \le v_i \le b_i,\ 1 \le i \le N\}$$

To solve this problem with **OFELI**, we write a C++ code:

Example 3: An optimization Problem (1/3)

Go To Example 4

Consider the following problem:

$$\mathbf{u} \in \mathscr{V}; \;\; W(\mathbf{u}) = \inf_{\mathbf{v} \in \mathscr{V}} W(\mathbf{v})$$

where

$$\mathscr{V} := \{\mathbf{v} \in \mathbb{R}^N; \;\; a_i \leq v_i \leq b_i, \;\; 1 \leq i \leq N\}$$

To solve this problem with OFELI, we write a C++ code:

Example 3: An optimization Problem (1/3)

Go To Example 4

Consider the following problem:

$$\mathbf{u} \in \mathscr{V}; \; W(\mathbf{u}) = \inf_{\mathbf{v} \in \mathscr{V}} W(\mathbf{v})$$

where

$$\mathscr{V} := \{\mathbf{v} \in \mathbb{R}^N; \; a_i \leq v_i \leq b_i, \; 1 \leq i \leq N\}$$

To solve this problem with **OFELI**, we write a C++ code:

Example 3: An optimization Problem (2/3)

```
Mesh ms("test.m");
User ud(ms);
Vect<double> x(ms.getNbDOF());
Vect<double> low(ms.getNbDOF()), up(ms.getNbDOF());
Vect<int> pivot(ms.getNbDOF());

Vect<double> bc(ms.getNbDOF());
ud.setDBC(bc);

Opt theOpt(ms,ud);
BCAsConstraint(ms,bc,up,low);

OptimTN<Opt>(theOpt,x,low,up,pivot,100,1.e-8,1);
```

Example 3: An optimization Problem (3/3)

The class `Opt` is defined as follows:

```
class Opt {

   public:
      Opt(Mesh &ms, User &ud);
      void Objective(const Vect<double> &x, double &f, Vect<double> &g);

   private:
      Mesh *_ms;
      User *_ud;
};
```

Example 3: An optimization Problem (3/3)

The class `Opt` is defined as follows:

```
class Opt {
  public:
    Opt(Mesh &ms, User &ud);
    void Objective(const Vect<double> &x, double &f, Vect<double> &g);
  private:
    Mesh *_ms;
    User *_ud;
};
```

Example 4: Mixed Hybrid Finite Elements (1/6)

This example illustrates the use of non standard methods in **OFELI** (Mixed Elements, Finite Volumes, ...) Consider the problem

$$\Delta u = 0 \qquad \text{in } \Omega \subset \mathbb{R}^2$$
$$u = g \qquad \text{on } \partial\Omega$$

This problem is equivalent to:

$$\mathbf{p} - \nabla u = 0, \; -\nabla\cdot\mathbf{p} = f \qquad \text{in } \Omega \subset \mathbb{R}^2, \qquad u = g \qquad \text{on } \partial\Omega$$

The approximation by mixed hybrid finite elements consists in defining the spaces:

$$\mathscr{V} = \{v \in L^2(\Omega);\; v_{|T} = \text{Const.} \quad \forall\; T \in \mathscr{T}\},$$
$$\mathscr{Q} = \{\mathbf{q} \in L^2(\Omega)^2;\; q_{|T} = \mathbf{a}_T + b_T\mathbf{x},\; \mathbf{a}_T \in \mathbb{R}^2,\; b_T \in \mathbb{R} \;\forall\; T \in \mathscr{T}\},$$
$$\mathscr{M} = \{\mu;\; \mu_{|e} = \text{Const.} \quad \forall\; e \in \mathscr{E}\}.$$

where $\mathscr{T}$: triangles, $\mathscr{E}$: edges.

Example 4: Mixed Hybrid Finite Elements (1/6)

This example illustrates the use of non standard methods in **OFELI** (Mixed Elements, Finite Volumes, . . .) Consider the problem

$$\Delta u = 0 \qquad \text{in } \Omega \subset \mathbb{R}^2$$
$$u = g \qquad \text{on } \partial\Omega$$

This problem is equivalent to:

$$\mathbf{p} - \nabla u = 0, \ -\nabla \cdot \mathbf{p} = f \qquad \text{in } \Omega \subset \mathbb{R}^2, \qquad u = g \qquad \text{on } \partial\Omega$$

The approximation by mixed hybrid finite elements consists in defining the spaces:

$$\mathscr{V} = \{v \in L^2(\Omega);\ v_{|T} = \text{Const.} \quad \forall\ T \in \mathscr{T}\},$$
$$\mathscr{Q} = \{\mathbf{q} \in L^2(\Omega)^2;\ q_{|T} = \mathbf{a}_T + b_T\mathbf{x},\ \mathbf{a}_T \in \mathbb{R}^2,\ b_T \in \mathbb{R}\ \forall\ T \in \mathscr{T}\},$$
$$\mathscr{M} = \{\mu;\ \mu_{|e} = \text{Const.} \quad \forall\ e \in \mathscr{E}\}.$$

where $\mathscr{T}$: triangles, $\mathscr{E}$: edges.

Example 4: Mixed Hybrid Finite Elements (1/6)

This example illustrates the use of non standard methods in **OFELI** (Mixed Elements, Finite Volumes, ...) Consider the problem

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

This problem is equivalent to:

$$\mathbf{p} - \nabla u = 0, \ \ -\nabla \cdot \mathbf{p} = f \qquad \text{in } \Omega \subset \mathbb{R}^2, \qquad u = g \qquad \text{on } \partial\Omega$$

The approximation by mixed hybrid finite elements consists in defining the spaces:

$$\begin{aligned} &\mathscr{V} = \{v \in L^2(\Omega);\ v_{|T} = \text{Const.} \quad \forall\ T \in \mathscr{T}\}, \\ &\mathscr{Q} = \{\mathbf{q} \in L^2(\Omega)^2;\ q_{|T} = \mathbf{a}_T + b_T\mathbf{x},\ \mathbf{a}_T \in \mathbb{R}^2,\ b_T \in \mathbb{R}\ \forall\ T \in \mathscr{T}\}, \\ &\mathscr{M} = \{\mu;\ \mu_{|e} = \text{Const.} \quad \forall\ e \in \mathscr{E}\}. \end{aligned}$$

where $\mathscr{T}$: triangles, $\mathscr{E}$: edges.

Example 4: Mixed Hybrid Finite Elements (1/6)

This example illustrates the use of non standard methods in **OFELI** (Mixed Elements, Finite Volumes, ...) Consider the problem

$$\Delta u = 0 \quad \text{in } \Omega \subset \mathbb{R}^2$$
$$u = g \quad \text{on } \partial\Omega$$

This problem is equivalent to:

$$\mathbf{p} - \nabla u = 0, \ -\nabla\cdot\mathbf{p} = f \quad \text{in } \Omega \subset \mathbb{R}^2, \quad u = g \quad \text{on } \partial\Omega$$

The approximation by mixed hybrid finite elements consists in defining the spaces:

$$\mathscr{V} = \{v \in L^2(\Omega);\ v_{|T} = \text{Const.} \quad \forall\ T \in \mathscr{T}\},$$
$$\mathscr{Q} = \{\mathbf{q} \in L^2(\Omega)^2;\ q_{|T} = \mathbf{a}_T + b_T\mathbf{x},\ \mathbf{a}_T \in \mathbb{R}^2,\ b_T \in \mathbb{R}\ \forall\ T \in \mathscr{T}\},$$
$$\mathscr{M} = \{\mu;\ \mu_{|e} = \text{Const.} \quad \forall\ e \in \mathscr{E}\}.$$

where $\mathscr{T}$: triangles, $\mathscr{E}$: edges.

Mixed Hybrid Finite Elements (2/6)

We then look for a triple $(u,\mathbf{p},\lambda)\in\mathscr{V}\times\mathscr{Q}\times\mathscr{M}$ such that:

$$\begin{aligned}
&\int_T \mathbf{p}\cdot\mathbf{q}\,d\mathbf{x}+\int_T u\,\nabla\cdot\mathbf{q}\,d\mathbf{x}-\sum_{e\in\mathscr{E}_T}\int_e\lambda\mathbf{q}\cdot\mathbf{n}\,ds=\sum_{e\in\mathscr{E}_T^D}\int_e g\mathbf{q}\cdot\mathbf{n}\,ds && \forall\ \mathbf{q}\in\mathscr{Q},\ T\in\mathscr{T},\\
&\int_T\nabla\cdot\mathbf{p}\,d\mathbf{x}=-\int_T f\,d\mathbf{x}, && \forall\ T\in\mathscr{T},\\
&\sum_{T\in\mathscr{T}}\sum_{e\in\mathscr{E}_T}\int_e\mu\,\mathbf{p}\cdot\mathbf{n}\,ds=0 && \forall\ \mu\in\mathscr{M}
\end{aligned}$$

After some calculus, we obtain for λ the linear system

$$\sum_{T\in\mathscr{T}_e}\Big(\frac{1}{|T|}\sum_{e'\in\mathscr{E}_T}\ell_e\ell_{e'}\mathbf{n}_T^e\cdot\mathbf{n}_T^{e'}\Big)\lambda_{e'}=-\sum_{T\in\mathscr{T}_e}\ell_e\mathbf{n}_T^e\cdot\Big(\frac{1}{2}f_T\,(\mathbf{c}_e-\mathbf{c}_T)+\sum_{e'\in\mathscr{E}_T^D}g_{e'}\ell_{e'}\mathbf{n}_T^{e'}\Big)\quad e\in\mathscr{E}$$

Mixed Hybrid Finite Elements (2/6)

We then look for a triple $(u, \mathbf{p}, \lambda) \in \mathscr{V} \times \mathscr{Q} \times \mathscr{M}$ such that:

$$\int_T \mathbf{p}\cdot\mathbf{q}\,d\mathbf{x} + \int_T \mathbf{u}\,\nabla\cdot\mathbf{q}\,d\mathbf{x} - \sum_{e\in\mathscr{E}_T}\int_e \lambda\mathbf{q}\cdot\mathbf{n}\,ds = \sum_{e\in\mathscr{E}_T^D}\int_e g\mathbf{q}\cdot\mathbf{n}\,ds \qquad \forall\ \mathbf{q}\in\mathscr{Q},\ T\in\mathscr{T},$$

$$\int_T \nabla\cdot\mathbf{p}\,d\mathbf{x} = -\int_T f\,d\mathbf{x}, \qquad \forall\ T\in\mathscr{T},$$

$$\sum_{T\in\mathscr{T}}\sum_{e\in\mathscr{E}_T}\int_e \mu\,\mathbf{p}\cdot\mathbf{n}\,ds = 0 \qquad \forall\ \mu\in\mathscr{M}$$

After some calculus, we obtain for λ the linear system

$$\sum_{T\in\mathscr{T}_e}\Big(\frac{1}{|T|}\sum_{e'\in\mathscr{E}_T}\ell_e\ell_{e'}\mathbf{n}_T^e\cdot\mathbf{n}_T^{e'}\Big)\lambda_{e'} = -\sum_{T\in\mathscr{T}_e}\ell_e\mathbf{n}_T^e\cdot\Big(\frac{1}{2}f_T\,(\mathbf{c}_e-\mathbf{c}_T)+\sum_{e'\in\mathscr{E}_T^D}g_{e'}\ell_{e'}\mathbf{n}_T^{e'}\Big) \quad e\in\mathscr{E}$$

Mixed Hybrid Finite Elements (3/6)

Implementation: The main program

```
   Mesh ms("test.m");
   ms.setDOFSupport(SIDE_DOF);
   ms.removeImposedDOF();

   SpMatrix<double> A(ms);
   Vect<double> b(ms.getNbEq()), lambda(ms.getNbSides());
   Vect<double> f(ms.getNbElements()), g(ms.getNbSides());
// Initialize vectors f and g
   ...

   Laplace2DMHRT0 eq(ms,A,b);
   eq.build(f,g);
   eq.solve(lambda);
```

Mixed Hybrid Finite Elements (3/6)

Implementation: The main program

```
   Mesh ms("test.m");
   ms.setDOFSupport(SIDE_DOF);
   ms.removeImposedDOF();

   SpMatrix<double> A(ms);
   Vect<double> b(ms.getNbEq()), lambda(ms.getNbSides());
   Vect<double> f(ms.getNbElements()), g(ms.getNbSides());
// Initialize vectors f and g
   ...

   Laplace2DMHRT0 eq(ms,A,b);
   eq.build(f,g);
   eq.solve(lambda);
```

Mixed Hybrid Finite Elements (3/6)

Implementation: The main program

```
   Mesh ms("test.m");
   ms.setDOFSupport(SIDE_DOF);
   ms.removeImposedDOF();

   SpMatrix<double> A(ms);
   Vect<double> b(ms.getNbEq()), lambda(ms.getNbSides());
   Vect<double> f(ms.getNbElements()), g(ms.getNbSides());
// Initialize vectors f and g
   ...

   Laplace2DMHRT0 eq(ms,A,b);
   eq.build(f,g);
   eq.solve(lambda);
```

Mixed Hybrid Finite Elements (3/6)

Implementation: The main program

```
   Mesh ms("test.m");
   ms.setDOFSupport(SIDE_DOF);
   ms.removeImposedDOF();

   SpMatrix<double> A(ms);
   Vect<double> b(ms.getNbEq()), lambda(ms.getNbSides());
   Vect<double> f(ms.getNbElements()), g(ms.getNbSides());
// Initialize vectors f and g
   ...

   Laplace2DMHRT0 eq(ms,A,b);
   eq.build(f,g);
   eq.solve(lambda);
```

Mixed Hybrid Finite Elements (4/6)

Implementation: The class Laplace2DMHRT0

```
class Laplace2DMHRT0 :  virtual public FE_Laplace<double,3,3,2,2> {

public :
   Laplace2DMHRT0();
   Laplace2DMHRT0(Mesh &ms, SpMatrix<double> &A, Vect<double> &b);
   ~Laplace2DMHRT0();
   void build(const Vect<double> &f, const Vect<double> &g);
   void Post(const Vect<double> &lambda, const Vect<double> &f
             Vect<double> &v, Vect<Point<double> > &p, Vect<double> &u);
   int solve(Vect<double> &u);

private :
   SpMatrix<double> *_A;
   Vect<double> *_b;
   const Vect<double> *_f, *_g;
   Triang3 *_tr;
   Side *_sd1, *_sd2, *_sd3;
   LocalVect<Point<double>,3> _n, _ce;
   void ElementSet(const Element *el);
   void LM_LHS();
   void LM_RHS();
```

Mixed Hybrid Finite Elements (4/6)

Implementation: The class Laplace2DMHRT0

```
class Laplace2DMHRT0 : virtual public FE_Laplace<double,3,3,2,2> {
public :
   Laplace2DMHRT0();
   Laplace2DMHRT0(Mesh &ms, SpMatrix<double> &A, Vect<double> &b);
   ~Laplace2DMHRT0();
   void build(const Vect<double> &f, const Vect<double> &g);
   void Post(const Vect<double> &lambda, const Vect<double> &f
             Vect<double> &v, Vect<Point<double> > &p, Vect<double> &u);
   int solve(Vect<double> &u);

private :
   SpMatrix<double> *_A;
   Vect<double> *_b;
   const Vect<double> *_f, *_g;
   Triang3 *_tr;
   Side *_sd1, *_sd2, *_sd3;
   LocalVect<Point<double>,3> _n, _ce;
   void ElementSet(const Element *el);
   void LM_LHS();
   void LM_RHS();
```

Mixed Hybrid Finite Elements (4/6)

Implementation: The class Laplace2DMHRT0

```
class Laplace2DMHRT0 :  virtual public FE_Laplace<double,3,3,2,2> {

public :
   Laplace2DMHRT0();
   Laplace2DMHRT0(Mesh &ms, SpMatrix<double> &A, Vect<double> &b);
   ~Laplace2DMHRT0();
   void build(const Vect<double> &f, const Vect<double> &g);
   void Post(const Vect<double> &lambda, const Vect<double> &f
             Vect<double> &v, Vect<Point<double> > &p, Vect<double> &u);
   int solve(Vect<double> &u);

private :
   SpMatrix<double> *_A;
   Vect<double> *_b;
   const Vect<double> *_f, *_g;
   Triang3 *_tr;
   Side *_sd1, *_sd2, *_sd3;
   LocalVect<Point<double>,3> _n, _ce;
   void ElementSet(const Element *el);
   void LM_LHS();
   void LM_RHS();
```

Mixed Hybrid Finite Elements (4/6)

Implementation: The class Laplace2DMHRT0

```
class Laplace2DMHRT0 : virtual public FE_Laplace<double,3,3,2,2> {

public :
   Laplace2DMHRT0();
   Laplace2DMHRT0(Mesh &ms, SpMatrix<double> &A, Vect<double> &b);
   ~Laplace2DMHRT0();
   void build(const Vect<double> &f, const Vect<double> &g);
   void Post(const Vect<double> &lambda, const Vect<double> &f
             Vect<double> &v, Vect<Point<double> > &p, Vect<double> &u);
   int solve(Vect<double> &u);

private :
   SpMatrix<double> *_A;
   Vect<double> *_b;
   const Vect<double> *_f, *_g;
   Triang3 *_tr;
   Side *_sd1, *_sd2, *_sd3;
   LocalVect<Point<double>,3> _n, _ce;
   void ElementSet(const Element *el);
   void LM_LHS();
   void LM_RHS();
```

Mixed Hybrid Finite Elements (4/6)

Implementation: The class Laplace2DMHRT0

```
class Laplace2DMHRT0 :  virtual public FE_Laplace<double,3,3,2,2> {

public :
   Laplace2DMHRT0();
   Laplace2DMHRT0(Mesh &ms, SpMatrix<double> &A, Vect<double> &b);
   ~Laplace2DMHRT0();
   void build(const Vect<double> &f, const Vect<double> &g);
   void Post(const Vect<double> &lambda, const Vect<double> &f
             Vect<double> &v, Vect<Point<double> > &p, Vect<double> &u);
   int solve(Vect<double> &u);

private :
   SpMatrix<double> *_A;
   Vect<double> *_b;
   const Vect<double> *_f, *_g;
   Triang3 *_tr;
   Side *_sd1, *_sd2, *_sd3;
   LocalVect<Point<double>,3> _n, _ce;
   void ElementSet(const Element *el);
   void LM_LHS();
   void LM_RHS();
```

Mixed Hybrid Finite Elements (4/6)

Implementation: The class `Laplace2DMHRT0`

```
class Laplace2DMHRT0 :  virtual public FE_Laplace<double,3,3,2,2> {

public :
   Laplace2DMHRT0();
   Laplace2DMHRT0(Mesh &ms, SpMatrix<double> &A, Vect<double> &b);
   ~Laplace2DMHRT0();
   void build(const Vect<double> &f, const Vect<double> &g);
   void Post(const Vect<double> &lambda, const Vect<double> &f
             Vect<double> &v, Vect<Point<double> > &p, Vect<double> &u);
   int solve(Vect<double> &u);

private :
   SpMatrix<double> *_A;
   Vect<double> *_b;
   const Vect<double> *_f, *_g;
   Triang3 *_tr;
   Side *_sd1, *_sd2, *_sd3;
   LocalVect<Point<double>,3> _n, _ce;
   void ElementSet(const Element *el);
   void LM_LHS();
   void LM_RHS();
```

Mixed Hybrid Finite Elements (4/6)

Implementation: The class Laplace2DMHRT0

```
class Laplace2DMHRT0 :  virtual public FE_Laplace<double,3,3,2,2> {

public :
   Laplace2DMHRT0();
   Laplace2DMHRT0(Mesh &ms, SpMatrix<double> &A, Vect<double> &b);
   ~Laplace2DMHRT0();
   void build(const Vect<double> &f, const Vect<double> &g);
   void Post(const Vect<double> &lambda, const Vect<double> &f
             Vect<double> &v, Vect<Point<double> > &p, Vect<double> &u);
   int solve(Vect<double> &u);

private :
   SpMatrix<double> *_A;
   Vect<double> *_b;
   const Vect<double> *_f, *_g;
   Triang3 *_tr;
   Side *_sd1, *_sd2, *_sd3;
   LocalVect<Point<double>,3> _n, _ce;
   void ElementSet(const Element *el);
   void LM_LHS();
   void LM_RHS();
```

Mixed Hybrid Finite Elements (4/6)

Implementation: The class Laplace2DMHRT0

```
class Laplace2DMHRT0 :  virtual public FE_Laplace<double,3,3,2,2> {
public :
   Laplace2DMHRT0();
   Laplace2DMHRT0(Mesh &ms, SpMatrix<double> &A, Vect<double> &b);
   ~Laplace2DMHRT0();
   void build(const Vect<double> &f, const Vect<double> &g);
   void Post(const Vect<double> &lambda, const Vect<double> &f
             Vect<double> &v, Vect<Point<double> > &p, Vect<double> &u);
   int solve(Vect<double> &u);
private :
   SpMatrix<double> *_A;
   Vect<double> *_b;
   const Vect<double> *_f, *_g;
   Triang3 *_tr;
   Side *_sd1, *_sd2, *_sd3;
   LocalVect<Point<double>,3> _n, _ce;
   void ElementSet(const Element *el);
   void LM_LHS();
   void LM_RHS();
```

Mixed Hybrid Finite Elements (5/6)

```
Laplace2DMHRT0::Laplace2DMHRT0(Mesh &ms, SpMatrix<double> &A, Vect<double> &b)
{
   _theMesh = &ms;
   _A = &A;
   _b = &b;
}

void Laplace2DMHRT0::ElementSet(const Element *el)
{
// Geometric calculations
}

void Laplace2DMHRT0::LM_LHS()
{
   for (size_t i=1; i<=3; i++)
      for (size_t j=1; j<=3; j++)
         eMat(i,j) = _n(i)*_n(j)/_area;
}
```

Mixed Hybrid Finite Elements (5/6)

```
Laplace2DMHRT0::Laplace2DMHRT0(Mesh &ms, SpMatrix<double> &A, Vect<double> &b)
{
   _theMesh = &ms;
   _A = &A;
   _b = &b;
}

void Laplace2DMHRT0::ElementSet(const Element *el)
{
// Geometric calculations
}

void Laplace2DMHRT0::LM_LHS()
{
   for (size_t i=1; i<=3; i++)
      for (size_t j=1; j<=3; j++)
         eMat(i,j) = _n(i)*_n(j)/_area;
}
```

Mixed Hybrid Finite Elements (5/6)

```
Laplace2DMHRT0::Laplace2DMHRT0(Mesh &ms, SpMatrix<double> &A, Vect<double> &b)
{
   _theMesh = &ms;
   _A = &A;
   _b = &b;
}

void Laplace2DMHRT0::ElementSet(const Element *el)
{
// Geometric calculations
}

void Laplace2DMHRT0::LM_LHS()
{
   for (size_t i=1; i<=3; i++)
      for (size_t j=1; j<=3; j++)
         eMat(i,j) = _n(i)*_n(j)/_area;
}
```

Mixed Hybrid Finite Elements (6/6)

```
void Laplace2DMHRT0::build(const Vect<double> &f, const Vect<double> &g)
{
  Element *el;
  for (_theMesh->topElement(); (el=_theMesh->getElement());) {
    ElementSet(el);
    _g = &g;
    _f = &f;
    LM_LHS();
    LM_RHS();
    SideAssembly(*el,EA(),*_A);
    SideAssembly(*el,Eb(),*_b);
  }
}

int Laplace2DMHRT0::solve(Vect<double> &u)
{
  double toler = 1.e-8;
  Vect<double> x;
  int nb_it = CG(*_A,Prec<double>(*_A,ILU_PREC),*_b,x,1000,toler,1);
  return nb_it;
}
```

Mixed Hybrid Finite Elements (6/6)

```
void Laplace2DMHRT0::build(const Vect<double> &f, const Vect<double> &g)
{
   Element *el;
   for (_theMesh->topElement(); (el=_theMesh->getElement());) {
      ElementSet(el);
      _g = &g;
      _f = &f;
      LM_LHS();
      LM_RHS();
      SideAssembly(*el,EA(),*_A);
      SideAssembly(*el,Eb(),*_b);
   }
}

int Laplace2DMHRT0::solve(Vect<double> &u)
{
   double toler = 1.e-8;
   Vect<double> x;
   int nb_it = CG(*_A,Prec<double>(*_A,ILU_PREC),*_b,x,1000,toler,1);
   return nb_it;
}
```

Mixed Hybrid Finite Elements (6/6)

```
void Laplace2DMHRT0::build(const Vect<double> &f, const Vect<double> &g)
{
   Element *el;
   for (_theMesh->topElement(); (el=_theMesh->getElement());) {
      ElementSet(el);
      _g = &g;
      _f = &f;
      LM_LHS();
      LM_RHS();
      SideAssembly(*el,EA(),*_A);
      SideAssembly(*el,Eb(),*_b);
   }
}

int Laplace2DMHRT0::solve(Vect<double> &u)
{
   double toler = 1.e-8;
   Vect<double> x;
   int nb_it = CG(*_A,Prec<double>(*_A,ILU_PREC),*_b,x,1000,toler,1);
   return nb_it;
}
```