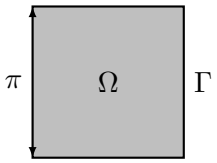
XLiFE++ Practice 2

Eigenvalue computations

The goal of this TP is to see how to compute the eigenvalues and some properties of these computations. You will compute the eigenvalues of the Dirichlet Laplacian on a square of length π (`pi_` in XLiFE++). You may easily know what are the exact eigenvalues. You start by the one element p-version to compare different interpolations and then you will perform the h-version on trianglular meshes with different types of finite elements. I recall the problem:

Find $(\lambda, u) \in \mathbb{R} \times \mathrm{H}_0^1(\Omega)$ such that $u \neq 0$ and

$$(\mathcal{P}) : \begin{cases} -\Delta u = \lambda u & \text{in } \Omega, \\ u = 0 & \text{on } \Gamma. \end{cases}$$



Exercise 1 - p-version (i.e. increasing the interpolation degree).

▶ Create a mesh of **one** quadrangular element of the square Ω and compute with degree ranging from 4 to 20:

- the 6 smallest eigenvalues,
- the condition number of the stiffness matrix,
- the condition number of the mass matrix,

with two types of interpolation:

a) with a uniform interpolation,

b) with a Gauss-Lobatto interpolation.

▶ Compare with the exact eigenvalues. Which interpolation is the best one? Why?

Exercise 2 - h-version (i.e. decreasing the mesh size).

▶ Create a sequence of nested triangular meshes of the square Ω and compute, for each mesh, the 6 smallest eigenvalues with two types of element:

a) with $P1$ conforming Lagrange elements,

b) with Crouzeix-Raviart nonconforming elements.

▶ Observe, with the Carstensen-Gediske estimator, that you obtain bounds on the exact eigenvalues.

▶ Print the mass matrix for the Crouzeix-Raviart elements, what do you observe?

Appendix

To create the geometry of a square with origin (a, b), length l, n points on each side, with domain name `Omega`, and boundary name `Gamma`:

```
Square sq(_origin=Point(a,b),_length=l,_nnodes=n,
          _domain_name="Omega",_side_names="Gamma");
```

To create a structured triangular or quadrangular mesh for a simple geometry:

```
Mesh mesh(sq,_triangle   , 1,_structured);
Mesh mesh(sq,_quadrangle , 1,_structured);
```

To create the interpolation spaces with different elements: (warning: GL works only on tensor product elements):

```
//Lagrange element with Uniform interpolation of degree d
Space VdUI(Omega, Lagrange, d, "VdUI");
//Lagrange element with Gauss-Lobatto interpolation of degree d
Space VdGL(Omega, interpolation(Lagrange, GaussLobatto, d, H1), "VdGL");
//Crouzeix-Raviart element
Space   CR(Omega, CrouzeixRaviart, 1, "CR");
```

To create the essential conditions:

```
EssentialConditions ecs = ((u|Gamma) = 0);
```

To create the matrix of the bilinear form auv and to impose the essential conditions ecs with the pseudo elimination parameter r:

```
TermMatrix A(auv, ecs, ReductionMethod(_pseudoReduction, r), "A");
```

To compute the N largest (STR="LM") or the N smallest (STR="SM") eigenvalues of A:

```
EigenElements eigs = eigenSolve(A,_nev=N,_which=STR);
```

To compute the N largest (STR="LM") or the N smallest (STR="SM") eigenvalues of a generalized eigenproblem $Ax = \lambda Bx$:

```
EigenElements eigs = eigenSolve(A, B,_nev=N,_which=STR);
```

For the N eigenvalues that you compute, you can access the eigenvalues with:

```
eigs.value(i); //for i from 1 to N
```

To save **all** the eigenvalues and the eigenvectors computed:

```
saveToFile("eigs", eigs, yourChoice); //yourChoice in {vtk, vtu, matlab, msh}
//To visualise .vtk or .vtu -> use ParaView
//To visualise .m (matlab)  -> use MATLAB or Octave
//To visualise .msh         -> use Gmsh
```

For more information on how to use the eigenvalues solver see section **7.3** in the documentation.

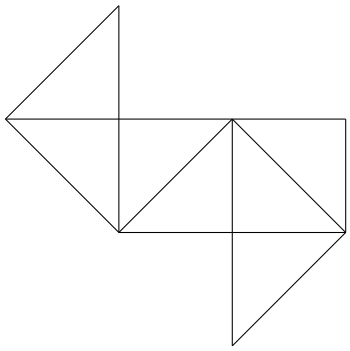
To print a matrix M:

```
M.saveToFile("matM.dat",_coo);    //save in coordinates (i,j,m_{i,j})
M.saveToFile("matM.dat",_dense); //save in a dense way (m_{i,j})
```

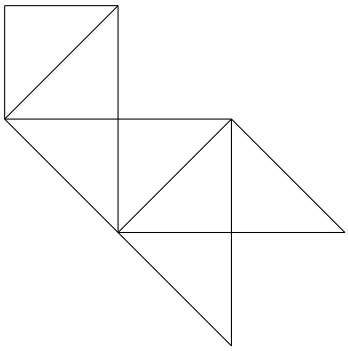
Bonus: isospectral polygons

$\lambda_1 = 2.53794399998$
$\lambda_2 = 3.65550971352$
$\lambda_3 = 5.17555935622$
$\lambda_4 = 6.53755744376$
$\lambda_5 = 7.24807786256$
$\lambda_6 = 9.20929499840$
$\lambda_7 = 10.5969856913$
$\lambda_8 = 11.5413953956$
$\lambda_9 = 12.3370055014$

Cocotte $\sim$ Origami chicken



Flèche $\sim$ Arrow



To define the meshes, copy and paste the following line ($b = 1$ for cocotte and $b = 0$ for flèche):

warning: only work with C++11 or C++14

```
std::vector<Point> vecP;
std::vector<std::vector<number_t>> vecE;
std::vector<std::vector<number_t>> vecB;
if (b){
    vecP = {Point(4.,0.),Point(6.,2.),Point(6.,4.),Point(4.,4.),Point(2.,4.),
            Point(2.,6.),Point(0.,4.),Point(2.,2.),Point(4.,2.)};
    vecE = {{1,2,9},{2,3,4},{2,4,9},{4,5,8},{4,8,9},{5,6,7},{5,7,8}};
  }
else{
    vecP = {Point(4.,0.),Point(4.,2.),Point(6.,2.),Point(4.,4.),Point(2.,4.),
            Point(2.,6.),Point(0.,6.),Point(0.,4.),Point(2.,2.)};
    vecE = {{1,2,9},{2,3,4},{2,4,9},{4,5,9},{5,6,8},{5,8,9},{6,7,8}};
  }
vecB = {{1,2},{2,3},{3,4},{4,5},{5,6},{6,7},{7,8},{8,9},{9,1}};

SetOfElems soe(vecP, vecE, vecB, {"Gamma"},_triangle, nbSubdiv);
Mesh mesh(soe,_triangle, 1,_subdiv);
mesh.saveToFile("mesh",msh); //to see the mesh with gmsh
```

where `nbSubdiv` ($= 0, 1, 2, \cdots$) is the number of subdivisions of the triangles. The domain name is `Omega` and the boundary name is `Gamma`.

▶ Compare the eigenvalues computed for cocotte and flèche, what do you observe?

▶ Perform the h-version, can you explain the convergence rate?

▶ Why does λ_9 converge faster than the others eigenvalues?

▶ **Bonus++** Transform the triangular mesh with 0 subdivision onto a quadrangular mesh and do the p-version. (Hint: use `SetOfElems` with the option `_quadrangle`.)