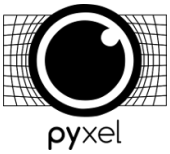
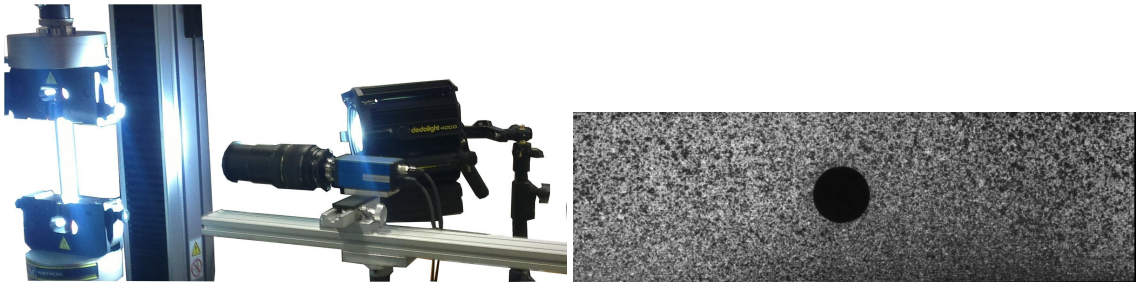
# Finite Element Digital Image Matching
# **Tutorial** and **Exercices**




## Goal of the session

1. Understand an implementation of the DIC (or Image Registration) problem
   https://github.com/jcpassieux/pyxel
2. Implement a Tikhonov regularization based on the Laplace operator
3. Implement an elastic regularization based on mechanical equilibrium
4. Perform an experimental displacement driven simulation
5. Validation of the above mechanical model wrt to experimental field
6. Identify a constitutive parameter from fullfield measurement

Let us consider an openhole glass/epoxy specimen subjected to a tensile test. The experiment is instrumented with a CCD camera.



A set of images is taken before and during the mechanical test. The corresponding images are given in folder `data/dic_composite/`.

## 1 Step by step tutorial

### 1.1 Start by importing some useful libraries

```
In [3]: import numpy as np
        import matplotlib.pyplot as plt
        import scipy.sparse.linalg as splalg
```

```
import scipy as sp
import pyxel as px
import os
```

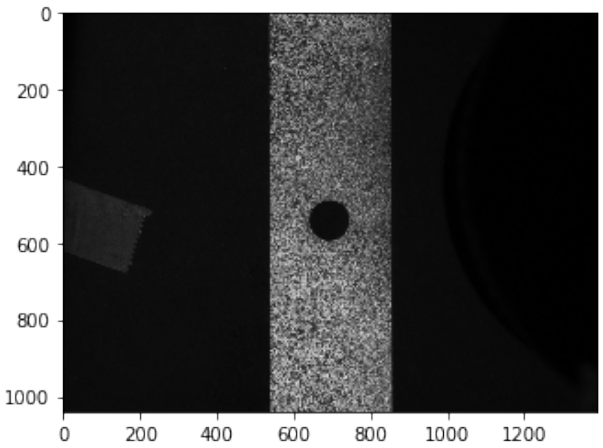
## 1.2 Naming and loading images

First, take a look at the image series named like *zoom-0053.tif* in Folder *'data/dic_composite'*.
It is a set of 11 images of a real mechanical experiment.
The first one being the reference state image.

```
In [6]: imnums=np.array([53,54,57,58,61,62,65,66,69,70,75])
        imagefile=os.path.join('data','dic_composite','zoom-0%03d_1.tif')
```

**Load** and **plot** reference image

```
In [7]: imref = imagefile % imnums[0]
        f=px.Image(imref).Load()
        f.Show()
```




Load the penultimate image

```
In [8]: imdef = imagefile % imnums[-2]
        g = px.Image(imdef).Load()
```

## 1.3 Mesh and associated camera model

An example with a quadrilateral ABAQUS mesh using meter (m) unit.

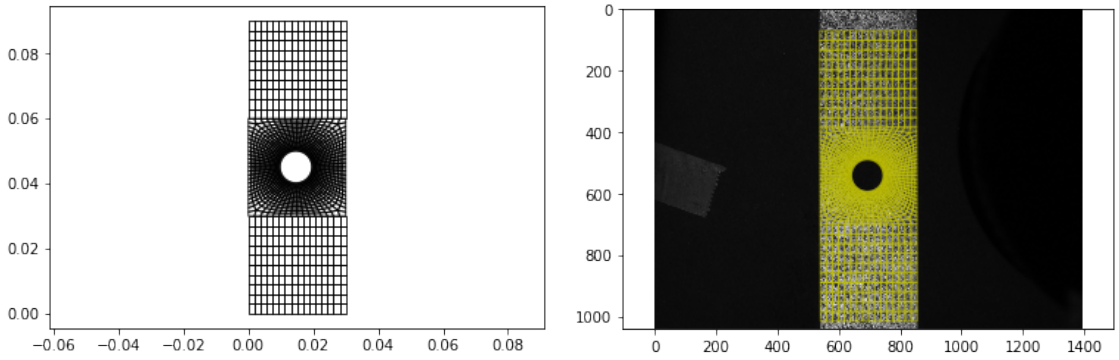
```
In [9]: meshfile=os.path.join('data','dic_composite','abaqus_q4_m.inp')
        m=px.ReadMeshINP(meshfile)
        l0=0.005      # regularization length
```

**Calibration** or **loading** the camera model parameters

```
In [10]: #cam=px.MeshCalibration(f,m,[1,2])
         #print(cam.get_p())
         # or
         cam=px.Camera(np.array([ 1.05168768e+04,  5.13737634e-02,
                                                  -9.65935782e-02, -2.65443047e-03]))
```

Different ways to **plot the mesh**: (1) the mesh or (2) its projection on the image.

```
In [11]: # Visualization of the mesh alone
         m.Plot()
         # Plot Mesh on the reference image
         px.PlotMeshImage(f,m,cam)
```




## 1.4 Preprocessing

Build the **connectivity** table Build the **quadrature** rule + Compute **FE basis functions** and derivatives

```
In [12]: m.Connectivity()
         m.DICIntegration(cam)
```

Optionnal **multiscale initialization** of the displacement field

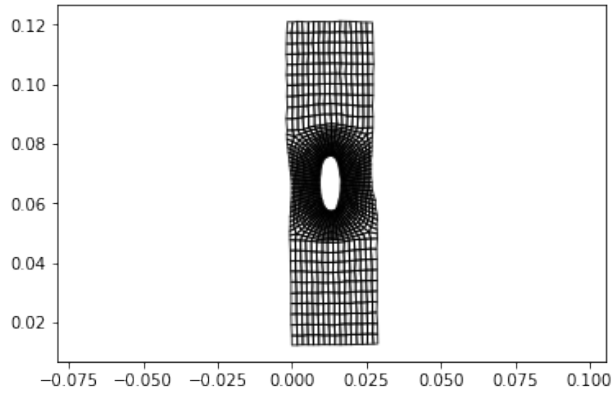
```
In [13]: U0=px.MultiscaleInit(m,f,g,cam,3)
```

```
SCALE  3
Iter #  1 | disc/dyn=10.44 % | dU/U=1.00e+00
Iter #  2 | disc/dyn=7.54 % | dU/U=4.86e-01
Iter #  3 | disc/dyn=5.02 % | dU/U=2.41e-01
Iter #  4 | disc/dyn=3.80 % | dU/U=9.59e-02
Iter #  5 | disc/dyn=3.51 % | dU/U=3.52e-02
Iter #  6 | disc/dyn=3.45 % | dU/U=1.29e-02
Iter #  7 | disc/dyn=3.44 % | dU/U=4.88e-03
Iter #  8 | disc/dyn=3.44 % | dU/U=1.96e-03
Iter #  9 | disc/dyn=3.43 % | dU/U=8.36e-04
```

```
Iter # 10 | disc/dyn=3.43 % | dU/U=3.80e-04
Iter # 11 | disc/dyn=3.43 % | dU/U=1.84e-04
Iter # 12 | disc/dyn=3.43 % | dU/U=9.48e-05
Iter # 13 | disc/dyn=3.43 % | dU/U=5.10e-05
Iter # 14 | disc/dyn=3.43 % | dU/U=2.83e-05
Iter # 15 | disc/dyn=3.43 % | dU/U=1.60e-05
Iter # 16 | disc/dyn=3.43 % | dU/U=9.12e-06
SCALE  2
Iter #  1 | disc/dyn=4.89 % | dU/U=6.09e-02
Iter #  2 | disc/dyn=4.11 % | dU/U=1.53e-02
Iter #  3 | disc/dyn=4.07 % | dU/U=5.11e-03
Iter #  4 | disc/dyn=4.06 % | dU/U=1.90e-03
Iter #  5 | disc/dyn=4.06 % | dU/U=7.52e-04
Iter #  6 | disc/dyn=4.06 % | dU/U=3.10e-04
Iter #  7 | disc/dyn=4.06 % | dU/U=1.32e-04
Iter #  8 | disc/dyn=4.06 % | dU/U=5.71e-05
Iter #  9 | disc/dyn=4.07 % | dU/U=2.53e-05
Iter # 10 | disc/dyn=4.07 % | dU/U=1.14e-05
Iter # 11 | disc/dyn=4.07 % | dU/U=5.18e-06
SCALE  1
Iter #  1 | disc/dyn=3.68 % | dU/U=3.89e-02
Iter #  2 | disc/dyn=3.00 % | dU/U=5.69e-03
Iter #  3 | disc/dyn=2.99 % | dU/U=1.15e-03
Iter #  4 | disc/dyn=2.99 % | dU/U=2.86e-04
Iter #  5 | disc/dyn=2.99 % | dU/U=8.02e-05
Iter #  6 | disc/dyn=2.99 % | dU/U=2.44e-05
Iter #  7 | disc/dyn=2.99 % | dU/U=7.87e-06
```

Plot the coarse initialization

```
In [14]: m.Plot(U0,30)
```



### 1.5 FE-DIC Solver - Remainders

The unknown displacement field $u(x)$ is sought as minimizing the following gray-level conservation quadratic distance:

$$u^\star = \underset{u \in \mathbf{L}^2(\Omega)}{\arg\min} \int_\Omega \left(f(x) - g(x + u(x))\right)^2 dx$$

This problem is a non-linear least-square problem. It is usually solved using Gauss-Newton. One way to introduce Gauss-Newton is to linearize the above functional. Given an approximation $u$ of the solution $u^\star$, we search for a correction $u^\star = u + d$ small enough to perform a first order Taylor expansion of $g$:

$$g(x + u^\star) \approx g(x + u) + d^T \nabla g(x + u)$$

Then the linearized functional reads:

$$d^\star = \underset{d \in \mathbf{L}^2(\Omega)}{\arg\min} \int_\Omega \Big(f(x) - g(x + u) + d^T \nabla g\Big)^2 dx$$

The displacement correction field $d(x)$ is searched for in a finite element approximation subspace:

$$d(x) = \sum_{i=1}^{n} N_i(x)\, d_i = \mathbf{N}(x)\, \mathbf{d}$$

The linearized functional is finally written as:

$$\mathbf{d}^\star = \underset{\mathbf{d} \in \mathbb{R}^n}{\arg\min} \int_\Omega \Big(f(x) - g(x + u) + \mathbf{d}^T \mathbf{N}^T \nabla g\Big)^2 dx$$

which from an algebraic point of view yields:

$$\mathbf{d}^\star = \underset{\mathbf{d} \in \mathbb{R}^n}{\arg\min} \frac{1}{2}\, \mathbf{d}^T \mathbf{H}\, \mathbf{d} - \mathbf{d}^T \mathbf{b}(u)$$

where Hessian $\mathbf{H}$ and right hand side vector $\mathbf{b}(u)$ will be computed by the `pyxel` library. They are defined by:

$$\mathbf{H} = \int_\Omega \mathbf{N}^T \nabla g \nabla g^T \mathbf{N}\, dx \qquad \text{and} \qquad \mathbf{b}(u) = \int_\Omega \mathbf{N}^T \nabla g\, \left(f(x) - g(x + u)\right)\, dx$$

The minimization of the quadratic problem simply leads to the following linear system at iteration $k$:

$$\mathbf{H}\, \mathbf{d} = \mathbf{b}(u)$$

Then the displacement approximation can be updated $\mathbf{u} \leftarrow \mathbf{u} + \mathbf{d}$ and one proceed to next iteration.

## 1.6 FE-DIC Implementation

Initialization of U and DIC engine. **Assembly** and **factorization** of DIC operator **H**.

```
In [15]: U=U0.copy()
         dic=px.DICEngine()
         H=dic.ComputeLHS(f,m,cam)                    # DIC operator assembly
         H_LU=splalg.splu(H)                          # LU Decomposition
```

**Gauss-Newton** iterations (without regularization)

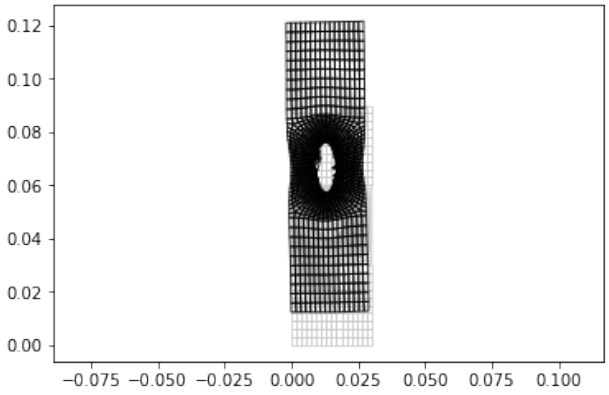
```
In [16]: for ik in range(0,30):
             [b,res]=dic.ComputeRHS(g,m,cam,U)        # DIC rhs assembly
             d=H_LU.solve(b)                          # Forward and backward substitution
             U+=d                                     # Displacement update
             err=np.linalg.norm(d)/np.linalg.norm(U)
             print("Iter # %2d | disc/dyn=%2.2f %% | dU/U=%1.2e" % (ik+1
                                                                   ,np.std(res)/dic.dyn*100,err))
             if err<1e-3:
                 break
```

```
Iter #  1 | disc/dyn=1.22 % | dU/U=1.49e-02
Iter #  2 | disc/dyn=0.96 % | dU/U=1.45e-03
Iter #  3 | disc/dyn=0.96 % | dU/U=2.07e-04
```

## 1.7 Postprocessing

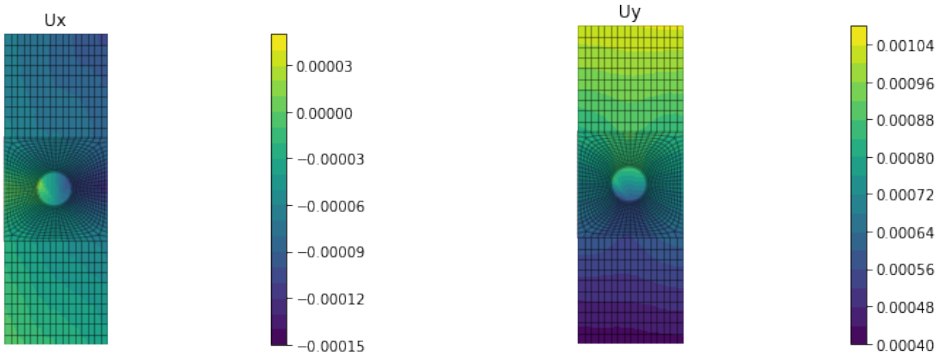
Visualization: **Scaled deformation of the mesh**

```
In [17]: m.Plot(edgecolor='#CCCCCC')
         m.Plot(U,30)
```



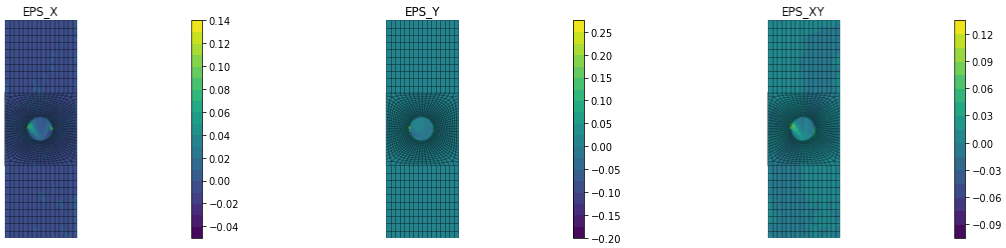

Visualization: **displacement fields**

```
In [18]: m.PlotContourDispl(U)
```



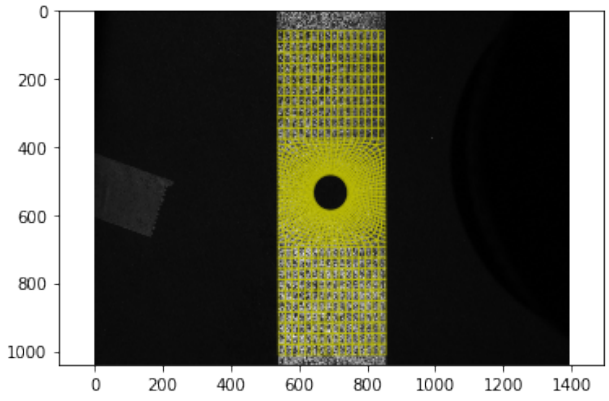

Visualization: **strain fields**

```
In [19]: m.PlotContourStrain(U)
```




Plot deformed **Mesh** on deformed state **image**

```
In [20]: px.PlotMeshImage(g,m,cam,U)
```



# 2 Exercises

## 2.1 Time resolved DIC analysis

**Q.1** Perform the measurement for each time steps and plot the corresponding displacement fields.

```
In [22]: U=np.zeros((m.ndof,10))
         #TODO
```

## 2.2 DIC with Tikhonov regularization

A regularization term is added to the DIC functionnal. The regularized problem becomes:

$$\mathbf{d}^\star = \underset{\mathbf{d}\in\mathbb{R}^n}{\arg\min} \underbrace{\frac{1}{2}\,\mathbf{d}^T\mathbf{H}\,\mathbf{d} - \mathbf{d}^T\mathbf{b}(u)}_{\text{graylevel term}} + \alpha \underbrace{\frac{1}{2}\,(\mathbf{u}+\mathbf{d})^T\mathbf{A}\,(\mathbf{u}+\mathbf{d})}_{\text{regularization term}}$$

with $\mathbf{A}$ a regularization operator. In this section $\mathbf{A}$ is a gradient operator, such that the regularization term decreases when the smoothness of the displacement increases. The weighting/scaling parameter $\alpha$ can be interpreted as a filter cut-off frequency: $\alpha = 0$ corresponds to the standard algorithm without regularisation, $\alpha \to \infty$ correspond to an homogeneous displacement field. Many techniques exists to define a "good" value. For the sake of simplicity, try to find a value by trial and error. The stationnarity of the regularized functionnal with respect to displacement correction $\mathbf{d}$ reads:

$$\mathbf{H}\,\mathbf{d} = \mathbf{b} \qquad \Rightarrow \qquad (\mathbf{H} + \alpha\,\mathbf{A})\,\mathbf{d} = \mathbf{b} - \alpha\,\mathbf{A}\,\mathbf{u}$$

Implement this regularization with $\alpha = 1, 10^5, 10^6$ and $10^{12}$. Comment on the results.

*Indications:*

```
In [23]: imdef = imagefile % imnums[-2]
         g = px.Image(imdef).Load()
         U=U0.copy()
         A=m.Tikhonov()
         #TODO
```

To attempt to give a physical meaning to regularization parameter $\alpha$, let's build a plane shear wave, in other words a displacement field $\mathbf{v}$ such that its vertical component is zero and horizontal is function of $y$:

$$v(y) = l_0 \cdot cos\left(\frac{2\pi y}{l_0}\right)$$

where $l_0$ is a characteristic length in meters. This plane wave can be build as follows:

```
In [23]: V=np.zeros_like(U)
         V[m.conn[:,0]]=l0*np.cos(m.n[:,1]/l0*2*np.pi)
```

**Q.2** Build these waves `V` for different values of $l_0$ (*e.g.* 0.1, 0.05, 0.01, 0.05, 0.001) and plot the corresponding displacement `m.Plot(V,1)`

**Q.3** Run again the regularized registration with $\alpha = \frac{\mathbf{v}^T\mathbf{H}\,\mathbf{v}}{\mathbf{v}^T\mathbf{A}\,\mathbf{v}}$ for the different values of $l_0$. Comment.

## 2.3 DIC with Elastic Regularization

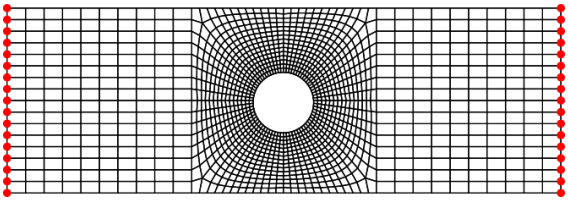
We now want to regularize the displacement field in a more mechanical fashion. Let's consider that we have at our disposal the elastic operator associated to the finite element mesh. The underlying mechanical problem that the unknown displacement is solution to is something like:

$$\mathbf{K}\,\mathbf{u} = \mathbf{F}$$

where $\mathbf{K}$ is the FE stiffness matrix and $\mathbf{F}$ the right-hand side force vector. A good regularization term could be

$$\frac{1}{2}\mathbf{u}^T\mathbf{K}\,\mathbf{u} - \mathbf{u}^T\mathbf{F} \qquad \text{or} \qquad \frac{1}{2}\|\mathbf{K}\,\mathbf{u} - \mathbf{F}\|^2$$

The problem is that, only given the images, the loading (and thus $\mathbf{F}$) is unknown. If we consider that there are no volume forces, the external forces only apply on a reduced number of degrees of freedom $b$ (red circles), whereas most of the dofs $i$ are free from external forces.



If the linear system is renumbered according to $i$ and $b$ dofs, it gives:

$$\begin{bmatrix} \mathbf{K}_{ii} & \mathbf{K}_{ib} \\ \mathbf{K}_{bi} & \mathbf{K}_{bb} \end{bmatrix} \begin{bmatrix} \mathbf{U}_{ii} \\ \mathbf{U}_{bb} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{F}_b \end{bmatrix}$$

Therefore, an idea is to regularize the free nodes $i$ only, using the following regularization term:

$$\frac{1}{2}\|\mathbf{K}\,\mathbf{u} - \mathbf{F}\|_D^2$$

where $\|\cdot\|_D$ is a semi-norm associated to a diagonal operator $\mathbf{D}$ with null diagonal term for non-free boundary nodes and a one instead:

$$\mathbf{D} = \begin{bmatrix} \mathbf{I}_{ni} & 0 \\ 0 & \mathbf{0}_{nb} \end{bmatrix} \qquad \text{such that} \qquad \|\mathbf{K}\,\mathbf{u} - \mathbf{F}\|_D^2 = \|\mathbf{K}\,\mathbf{u}\|_D^2 = \mathbf{u}^T\mathbf{K}^T\mathbf{D}\,\mathbf{K}\,\mathbf{u}$$

Finally, it's the same technique as before, but with a different regularization operator $\mathbf{A} = \mathbf{K}^T\mathbf{D}\,\mathbf{K}$

Define **Stiffness/Hooke tensor** (ortotropic in this case)

```
In [27]: El=1.0e+10
         Et=1.9e+10
         vtl=0.18
         Glt=1.0e+09
         vlt=vtl*El/Et
         alp=1/(1-vlt*vtl)
         hooke=np.array([[alp*El     , alp*vtl*El,0    ],
                         [alp*vlt*Et , alp*Et    ,0    ],
                         [0          , 0         ,2*Glt]])
```

Assemble **stiffness matrix**

```
In [28]: K=m.Stiffness(hooke)
```

Select or load the non-free boundary nodes (here two lines)

```
In [29]: # repb1=px.SelectMeshLine(m)
         # repb2=px.SelectMeshLine(m)
         # or
         repb1=np.array([ 12,  13, 221, 222, 223, 224, 225, 226, 227, 228, 229,
                                                        230, 231, 232, 233, 234, 235])
         repb2=np.array([ 14,  15, 254, 255, 256, 257, 258, 259, 260, 261, 262,
                                                        263, 264, 265, 266, 267, 268])
         repb=np.append(repb1,repb2)
         dofb=m.conn[repb,:].ravel()
         D=np.ones_like(U)
         D[dofb]=0
         D=sp.sparse.diags(D)
         A=K.T.dot(D.dot(K))
```

**Q.4** Run again the registration with elastic regularization.

**Q.5** Try to compute $\alpha$ again using plane waves.

```
In [ ]: #TODO
```

### 2.4 Experimental displacement driven simulation

Here, the goal is to perform a FE simulation on the same domain (same mesh) with measured boundary conditions. Find displacement $\mathbf{v}$ such that:

$$\mathbf{K}\,\mathbf{v} = \mathbf{0} \qquad with \qquad \mathbf{v}|_{\partial_u\Omega} = \mathbf{u}|_{\partial_u\Omega}$$

**Q.6** Extract the experimental dirichlet boundary conditions corresponding to one of the previous measurements. Apply them to the boundaries ($b$ dofs) of an elastic simulation.

```
In [ ]: #TODO
```

### 2.5 Model Validation

Plot the distance map between simulated and measured displacements fields. When using the same FE basis for both simulation and measurement, it simply consists in comparing displacement dof-by-dof:

$$\mathbf{dis} = abs(\mathbf{u} - \mathbf{v})$$

**Q.7** Compare dof-by-dof the displacement field previously computed and a measurement. Plot the discrepandy field.

```
In [31]: #TODO
```

## 2.6 Constitutive parameter identification

For the sake of simplicity, let's try to identify only the Poisson's ratio $\nu_{tl}$ using a very basic FEMU approach. The functionnal reads:

$$\mathcal{J}^{femu} = \frac{1}{2}\|\mathbf{u} - \mathbf{v}(\mathbf{p})\|^2$$

$\mathbf{u}$ being the experimental displacement and $\mathbf{v}$ the simulated displacement fields that depends on the vector of constitutive parameters $\mathbf{p}$

To linearize the problem, the parameter $\mathbf{p}$ is sought in the form of $\mathbf{p} = \mathbf{p}_0 + \mathbf{q}$, the running approximation $\mathbf{p}_0$ begin fixed. The stationnarity of the above functionnal with respect to constitutive parameter update $\mathbf{q}$ reads:

$$\frac{\partial \mathbf{v}}{\partial \mathbf{p}}^T \frac{\partial \mathbf{v}}{\partial \mathbf{p}} \mathbf{q} = -\frac{\partial \mathbf{v}}{\partial \mathbf{p}}^T (\mathbf{u} - \mathbf{v}(\mathbf{p}_0))$$

The sensitivities $\frac{\partial \mathbf{v}}{\partial \mathbf{p}}$ (derivative of the displacement with respect to the parameter) is computed numerically by finite differences.

**Q.8** Try to implement the different brick necessary to update the consititutive parameters of the elastic model to best fit the experimental field.

```
In [30]: #TODO (may require more time than is available...)
```