

# Convergence\_Poisson\_FV5\_SQUARE

October 21, 2018

```
In [1]: from IPython.display import display, Markdown
        with open('PoissonProblemOnSquare.md', 'r') as file1:
            PoissonProblemOnSquare = file1.read()
        with open('DescriptionFV5PoissonProblem.md', 'r') as file2:
            DescriptionFV5PoissonProblem = file2.read()
        with open('CodeFV5PoissonProblem.md', 'r') as file3:
            CodeFV5PoissonProblem = file3.read()
        with open('BibliographyFV5.md', 'r') as file4:
            BibliographyFV5=file4.read()
```

## 1 FV5 scheme for Poisson equation

```
In [2]: display(Markdown(PoissonProblemOnSquare))
```

### 1.1 The Poisson problem on the square

We consider the following Poisson problem with Dirichlet boundary conditions

$$\begin{cases} -\Delta u = f & \text{on } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases}$$

on the square domain  $\Omega = [0, 1] \times [0, 1]$  with

$$f = 2\pi^2 \sin(\pi x) \sin(\pi y).$$

The unique solution of the problem is

$$u = \sin(\pi x) \sin(\pi y).$$

The Poisson equation is a particular case of the diffusion problem

$$-\nabla \cdot (K \vec{\nabla} u) = f$$

and the associated diffusion flux is

$$F(u) = K \nabla u.$$

We are in the particular case where  $K = 1$ .

```
In [3]: display(Markdown(BibliographyFV5))
```

## 1.2 Some bibliographical remarks

- Order 1 convergence on triangular meshes  
*R. Herbin, An error estimate for a four point finite volume scheme for the convection-diffusion equation on a triangular mesh, Num. Meth. P.D.E., 165-173, 1995.*
- On triangular meshes, the FV5 scheme order is 2 provided
  - the center of the circumscribed circle is used instead of the center of mass in each cell
  - the Delaunay conditions are satisfied (no neighboring cell is included in the circumscribed circle of an arbitrary cell)
- Non convergence on highly deformed meshes  
*K. Domelevo, P. Omnes, A finite volume method for the Laplace equation on almost arbitrary 2D grids, Mathematical Modelling and Numerical Analysis, 2005*
- The scheme is order 1 if the mesh is conforming except on a line  
*J. Droniou, C. Le Potier, Construction and Convergence Study of Schemes Preserving the Elliptic Local Maximum Principle, SIAM Journal on Numerical Analysis, 2011*
- The scheme is order 2 if Delaunay type conditions,  $f \in H^1$  and meshes are generated from an initial mesh either by subdivisions, symmetry or translation  
*J. Droniou, Improved  $L^2$  estimate for gradient schemes and super-convergence of the TPFA finite volume scheme, 2018*
- It is possible to converge with order 1 on the gradient, but only order 1 on the function ie there is no equivalent of the Aubin-Nitsche lemma in the finite volume context  
*P. Omnes, Error estimates for a finite volume method for the Laplace equation in dimension one through discrete Green functions. International Journal on Finite Volumes 6(1), 18p., electronic only, 2009*

In [4] : `display(Markdown(DescriptionFV5PoissonProblem))`

## 1.3 The FV5 scheme for the Laplace equation

The domain  $\Omega$  is decomposed into cells  $C_i$ .

$|C_i|$  is the measure of the cell  $C_i$ .

$f_{ij}$  is the interface between two cells  $C_i$  and  $C_j$ .

$s_{ij}$  is the measure of the interface  $f_{ij}$ .

$d_{ij}$  is the distance between the centers of mass of the two cells  $C_i$  and  $C_j$ .

The discrete Poisson problem is

$$-\frac{1}{|C_i|} \sum s_{ij} F_{ij} = f_i,$$

where  $u_i$  is the approximation of  $u$  in the cell  $C_i$ ,

$f_i$  is the approximation of  $f$  in the cell  $C_i$ ,

$F_{ij}$  is a numerical approximation of the outward normal diffusion flux from cell  $i$  to cell  $j$ .

In the case of the scheme FV5, the flux formula are

$$F_{ij} = \frac{u_j - u_i}{d_{ij}},$$

for two cells  $i$  and  $j$  inside the domain,  
and

$$F_{boundary} = \frac{u(x_f) - u_i}{d_{if}},$$

for a boundary face with center  $x_f$ , inner cell  $i$  and distance between face and cell centers  $d_{if}$

In [5]: display(Markdown(CodeFV5PoissonProblem))

## 1.4 The script

```
#Discrétisation du second membre et extraction du nb max de voisins d'une cellule
=====
my_RHSfield = cdmath.Field("RHS_field", cdmath.CELLS, my_mesh, 1)
maxNbNeighbours=0#This is to determine the number of non zero coefficients in the sparse finite

for i in range(nbCells):
    Ci = my_mesh.getCell(i)
    x = Ci.x()
    y = Ci.y()

    my_RHSfield[i]=2*pi*pi*sin(pi*x)*sin(pi*y)#mettre la fonction definie au second membre de l
    # compute maximum number of neighbours
    maxNbNeighbours= max(1+Ci.getNumberOfFaces()),maxNbNeighbours)

# Construction de la matrice et du vecteur second membre du système linéaire
=====
Rigidite=cdmath.SparseMatrixPetsc(nbCells,nbCells,maxNbNeighbours)# warning : third argument is
RHS=cdmath.Vector(nbCells)
#Parcours des cellules du domaine
for i in range(nbCells):
    RHS[i]=my_RHSfield[i] #la valeur moyenne du second membre f dans la cellule i
    Ci=my_mesh.getCell(i)
    for j in range(Ci.getNumberOfFaces()):# parcours des faces voisines
        Fj=my_mesh.getFace(Ci.getFaceId(j))
        if not Fj.isBorder():
            k=Fj.getCellId(0)
            if k==i :
                k=Fj.getCellId(1)
            Ck=my_mesh.getCell(k)
            distance=Ci.getBarryCenter().distance(Ck.getBarryCenter())
            coeff=Fj.getMeasure()/Ci.getMeasure()/distance
            Rigidite.setValue(i,k,-coeff) # terme extradiagonal
        else:
            coeff=Fj.getMeasure()/Ci.getMeasure()/Ci.getBarryCenter().distance(Fj.getBarryCenter)
            #For the particular case where the mesh boundary does not coincide with the domain b
            x=Fj.getBarryCenter().x()
            y=Fj.getBarryCenter().y()
            RHS[i]+=coeff*sin(pi*x)*sin(pi*y)#mettre ici la condition limite du problème de Dirichlet
```

```

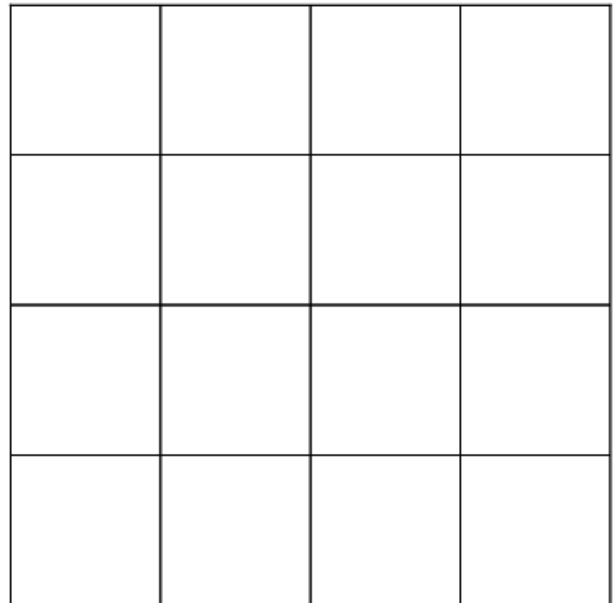
Rigidite.addValue(i,i,coeff) # terme diagonal

# Résolution du système linéaire
#=====
LS=cdmath.LinearSolver(Rigidite,RHS,500,1.E-6,"GMRES","ILU")
SolSyst=LS.solve()

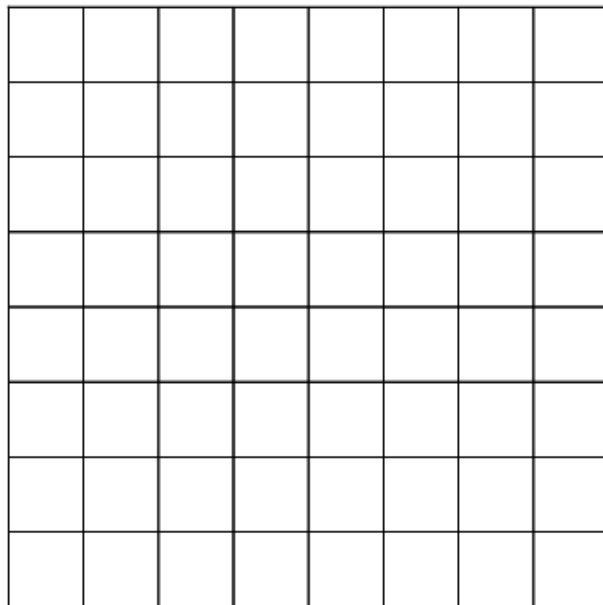
# Automatic postprocessing : save 2D picture and plot diagonal data
#=====
PV_routines.Save_PV_data_to_picture_file("my_ResultField_0.vtu","ResultField",'CELLS',"my_ResultField_0.vtu")
diag_data=VTK_routines.Extract_field_data_over_line_to_numpyArray(my_ResultField,[0,1,0],[1,0,0])
plt.plot(curv_abs, diag_data, label= str(nbCells)+ ' cells mesh')
plt.savefig("FV5_on_square_PlotOverDiagonalLine.png")

```

## 1.5 Regular grid

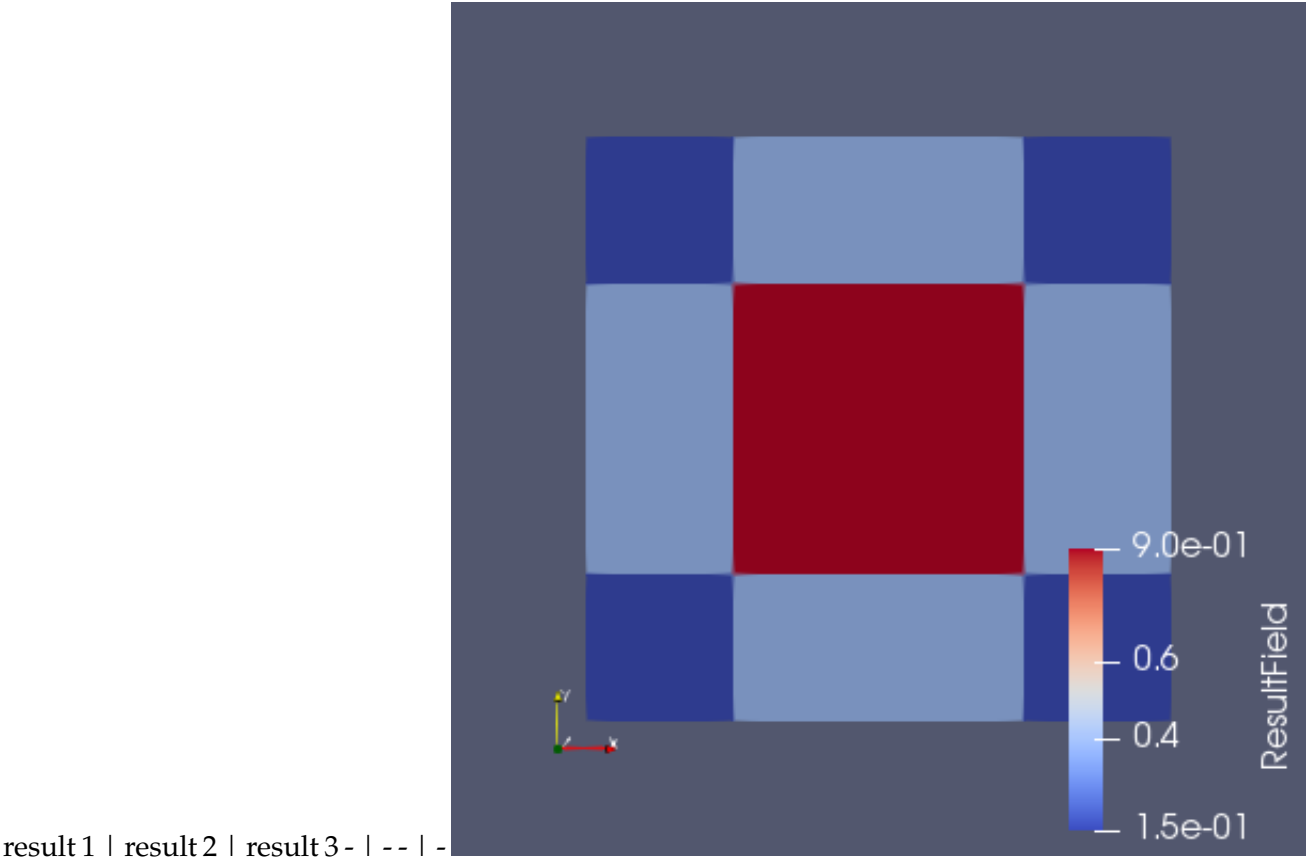
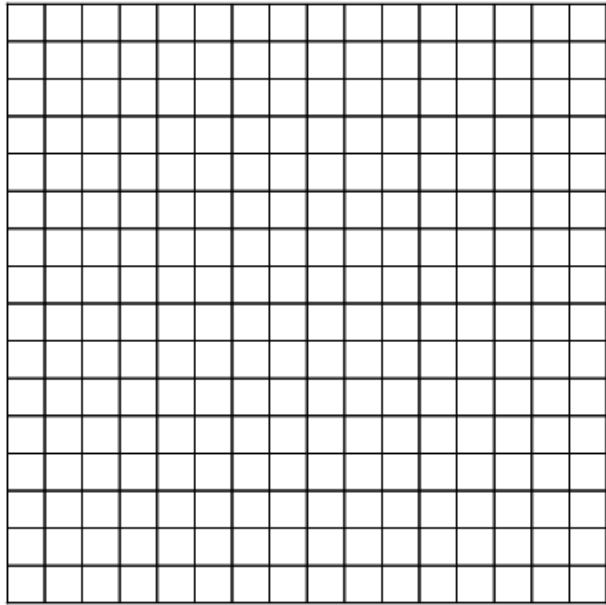


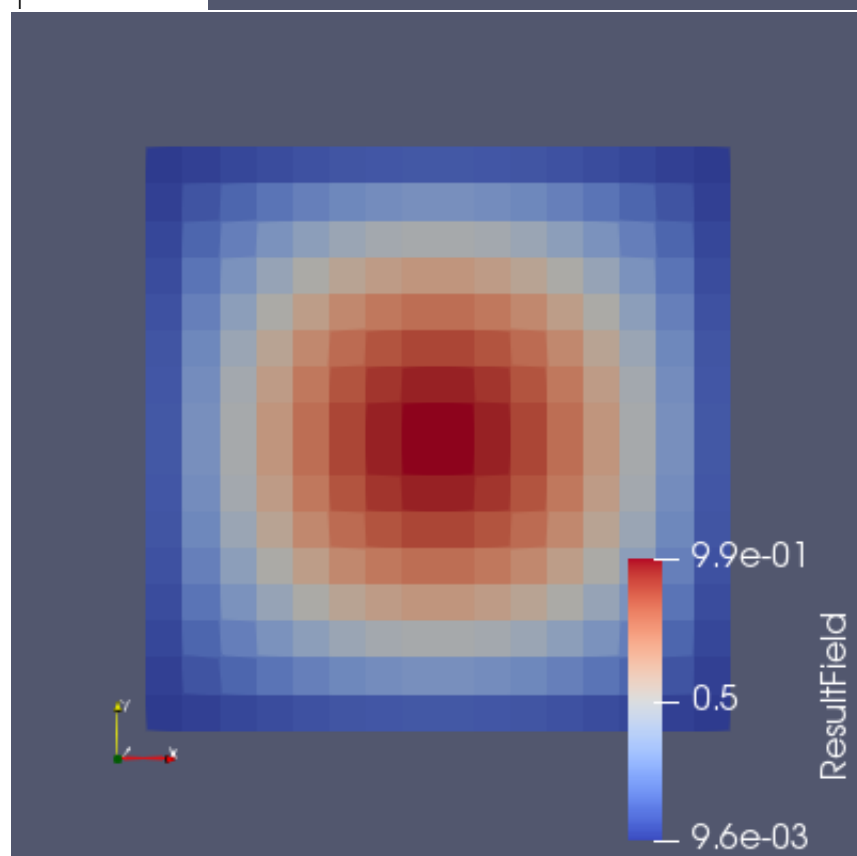
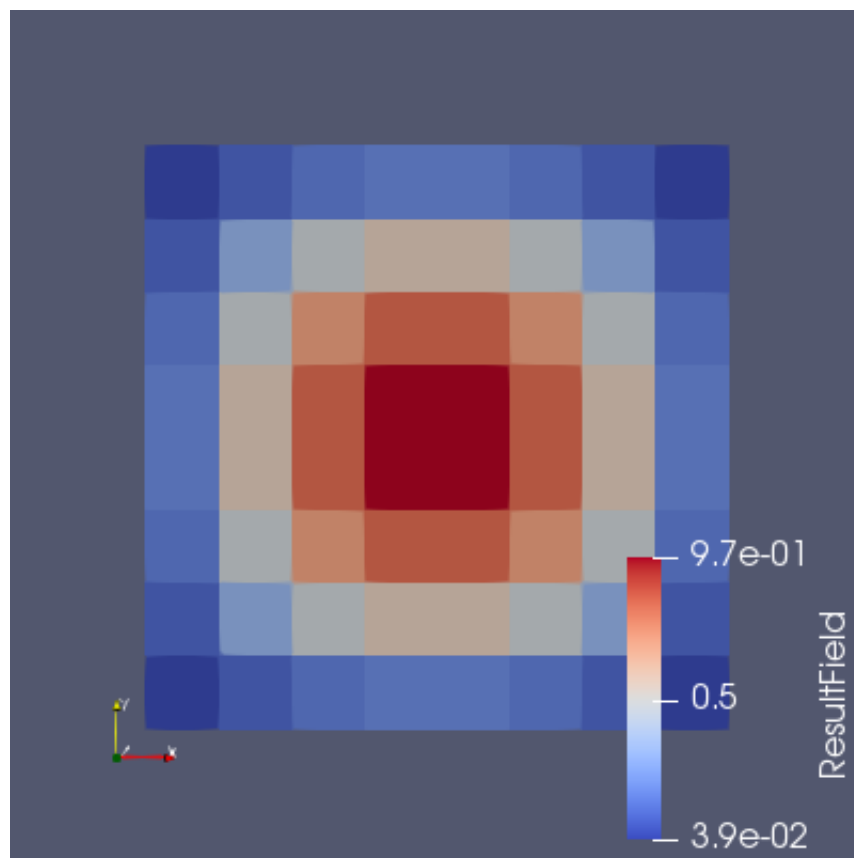
mesh 1 | mesh 2 | mesh 3 - | - - | -



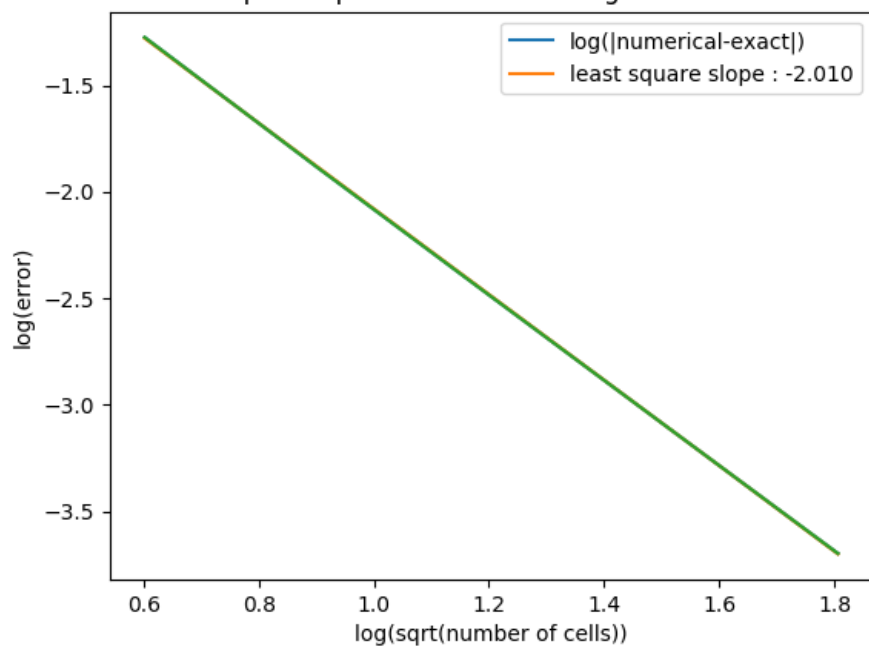
|

|



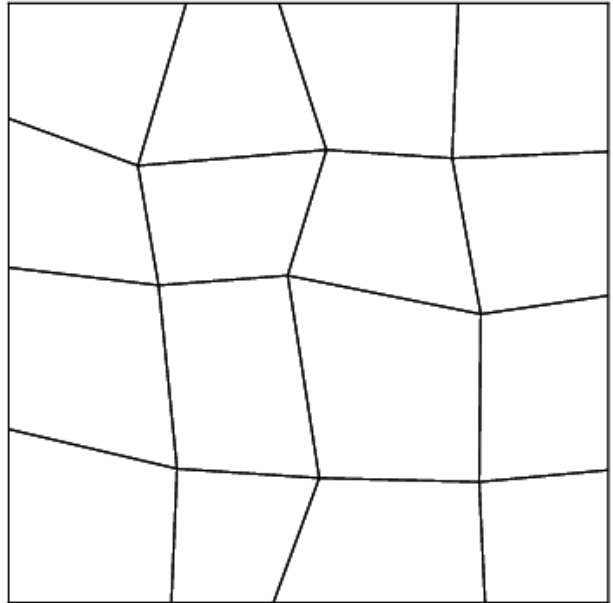


Convergence of finite volumes for  
Laplace operator on 2D rectangular meshes

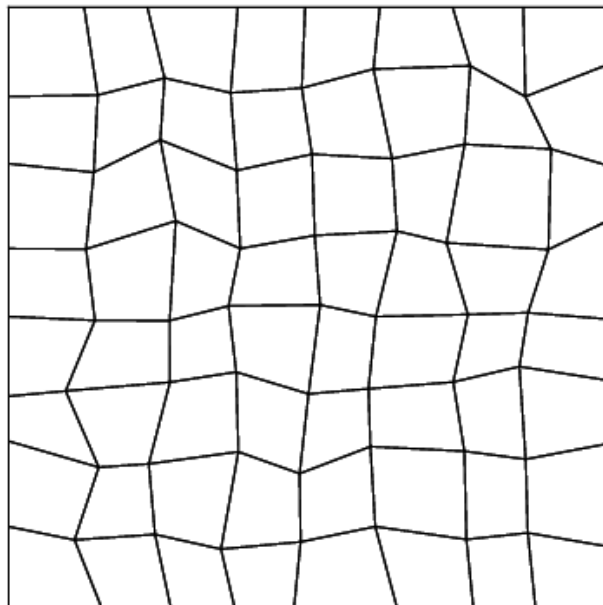




## 1.6 Deformed quadrangles

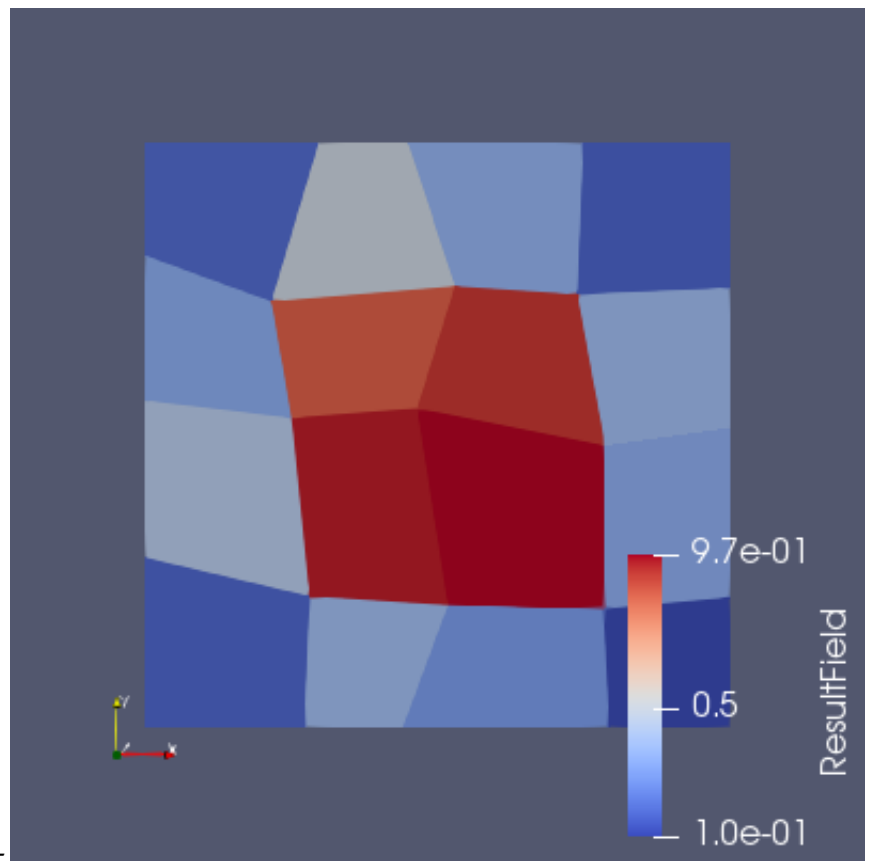
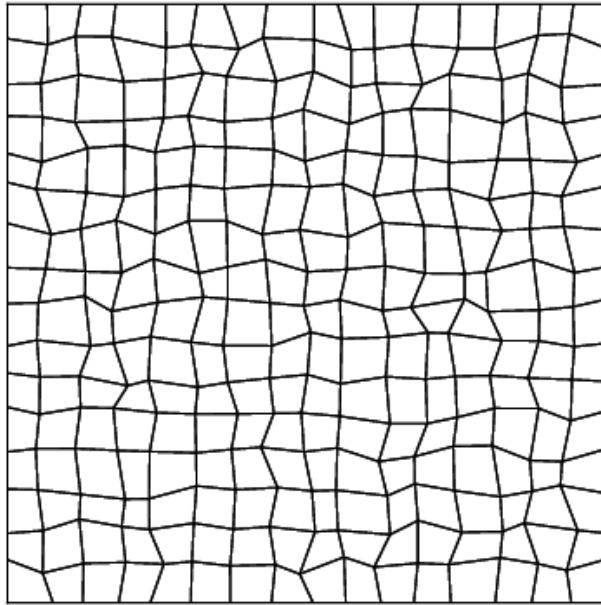


mesh 1 | mesh 2 | mesh 3 - | - - | -

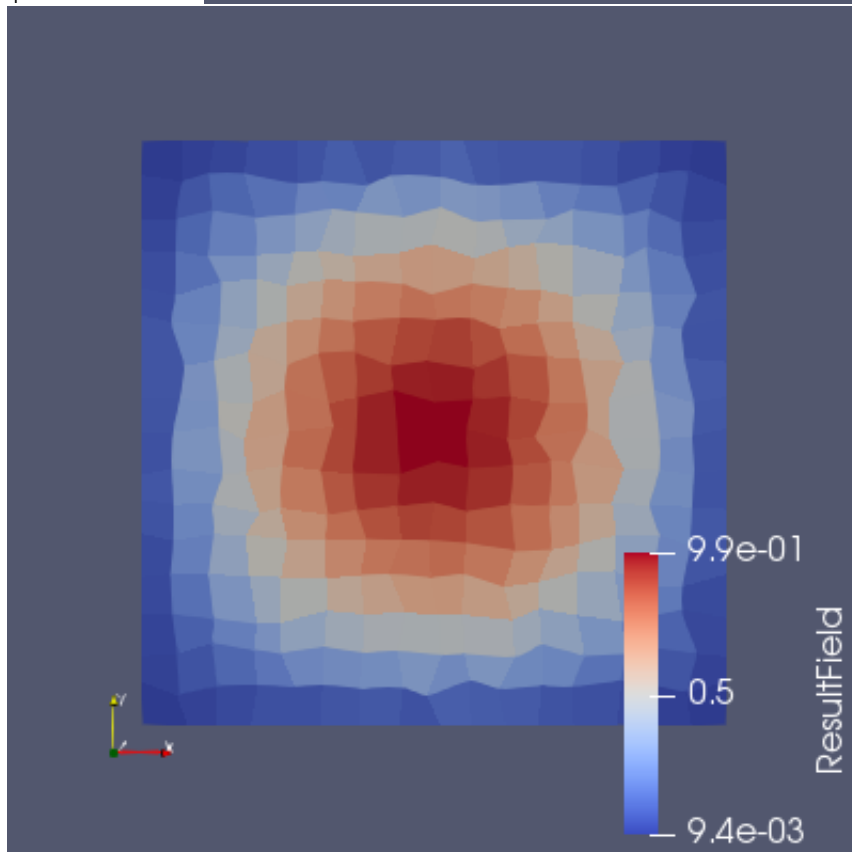
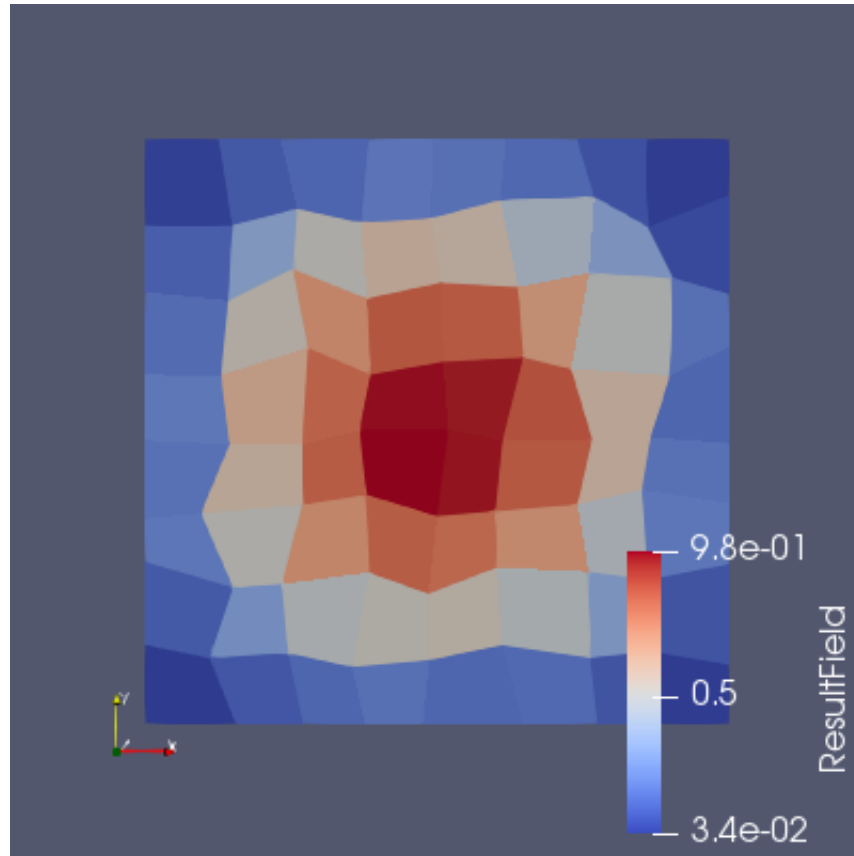


|

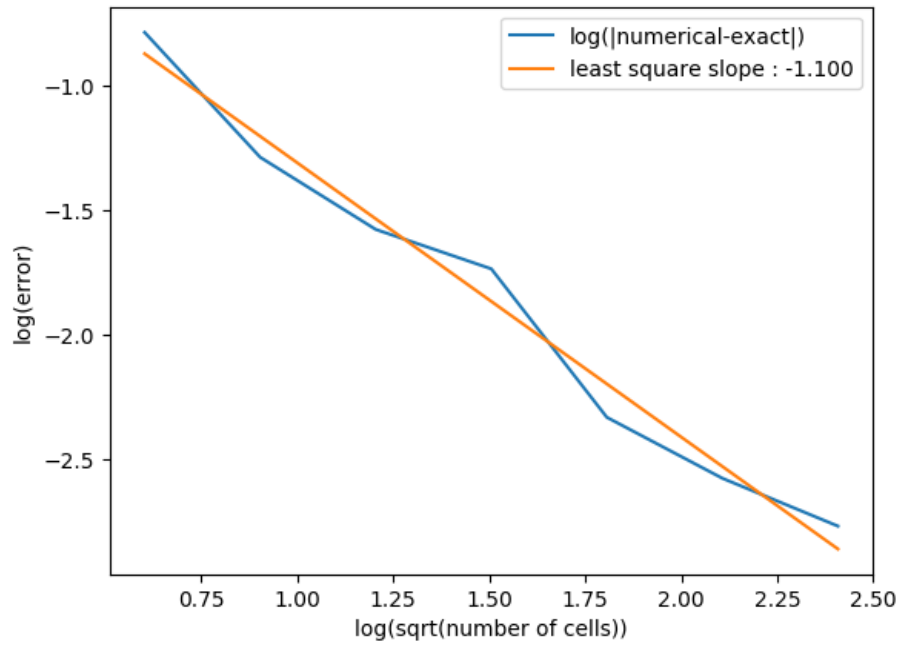
|



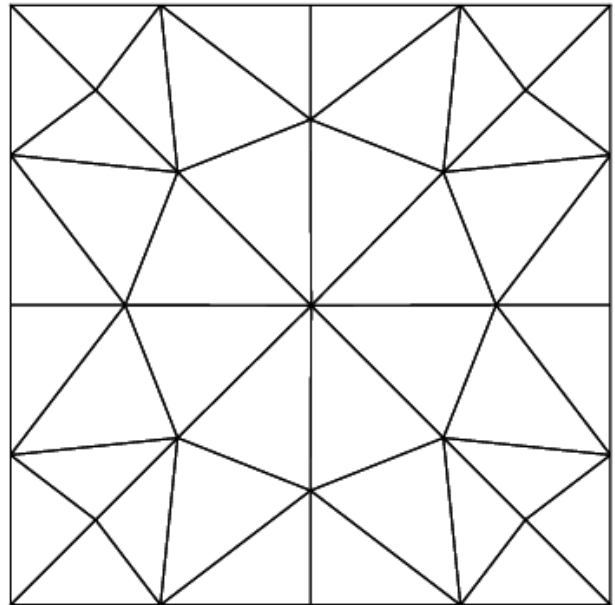
result 1 | result 2 | result 3 - | - - | -



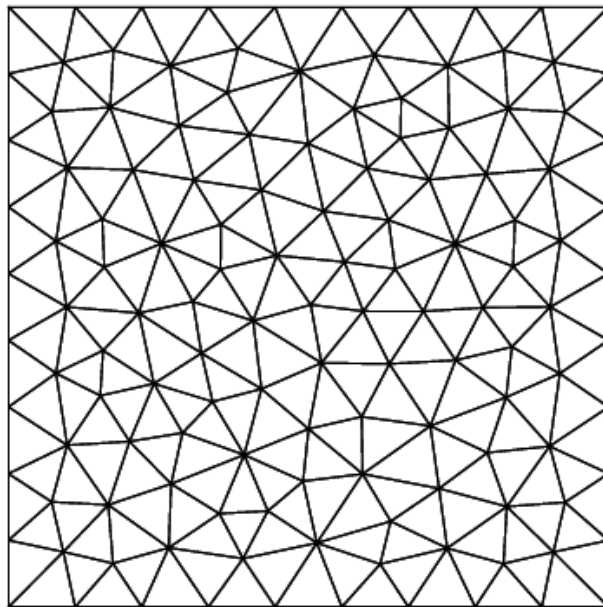
Convergence of finite volumes  
for Laplace operator on a 2D deformed quadrangles meshes



## 1.7 Triangular meshes

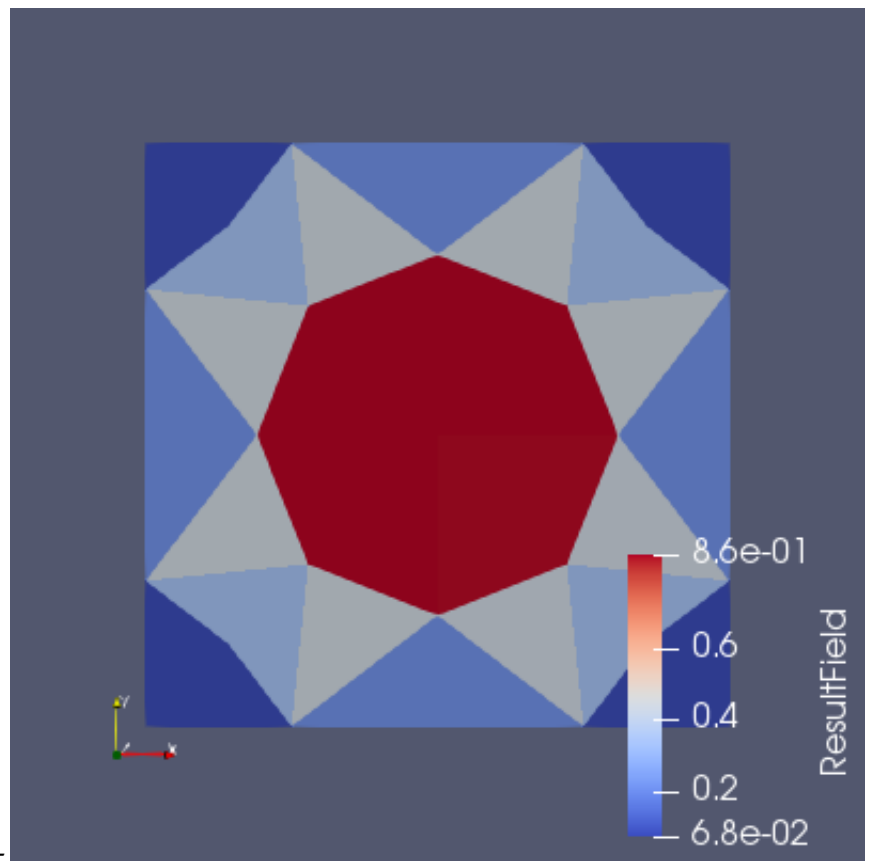
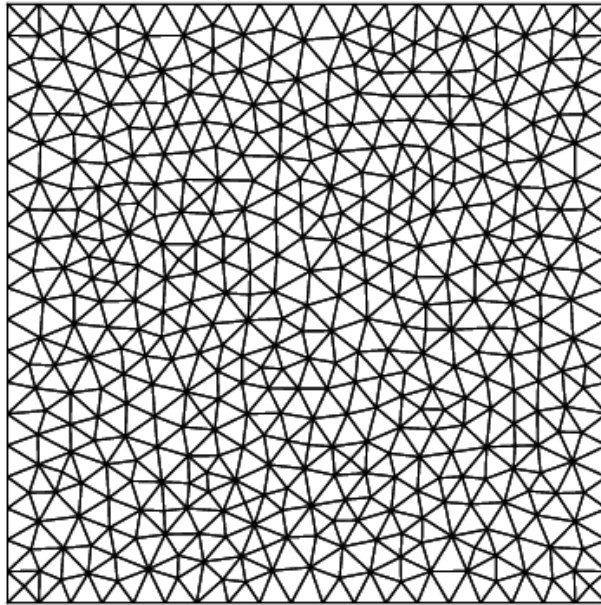


mesh 1 | mesh 2 | mesh 3 - | - - | -

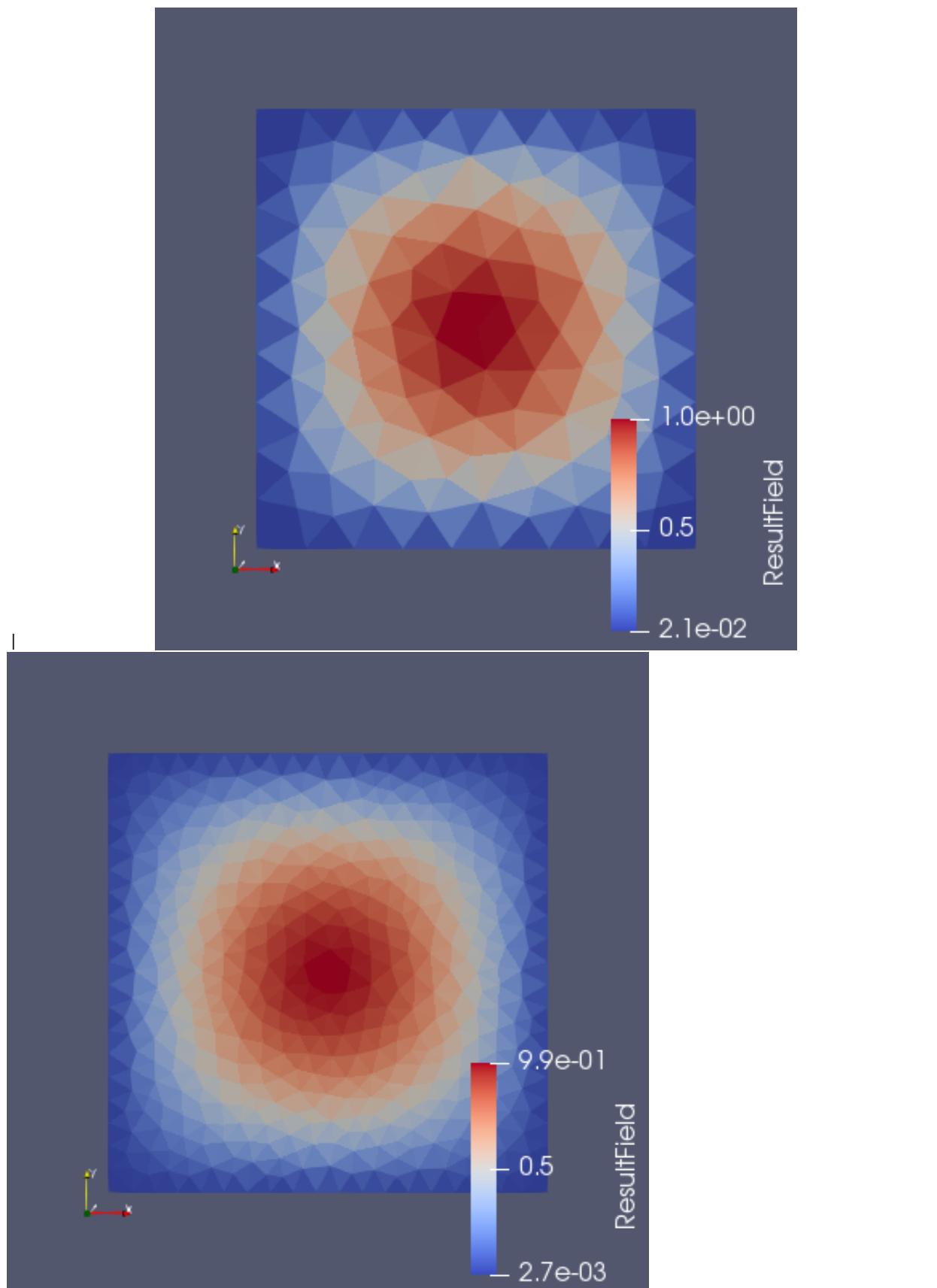


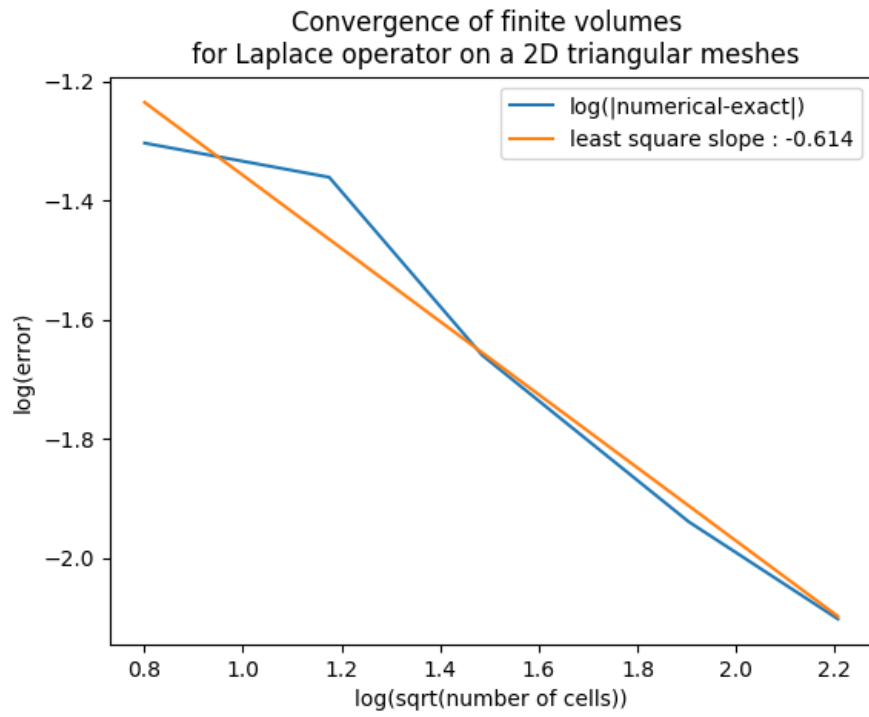
|

|



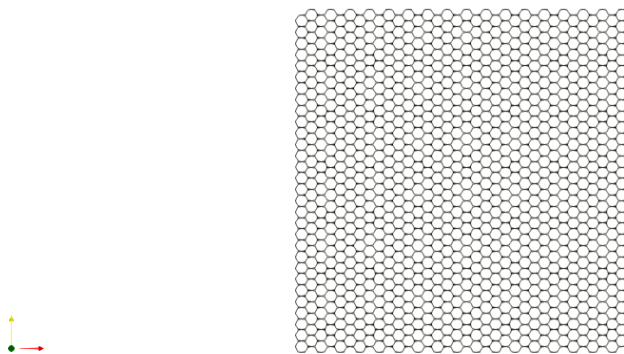
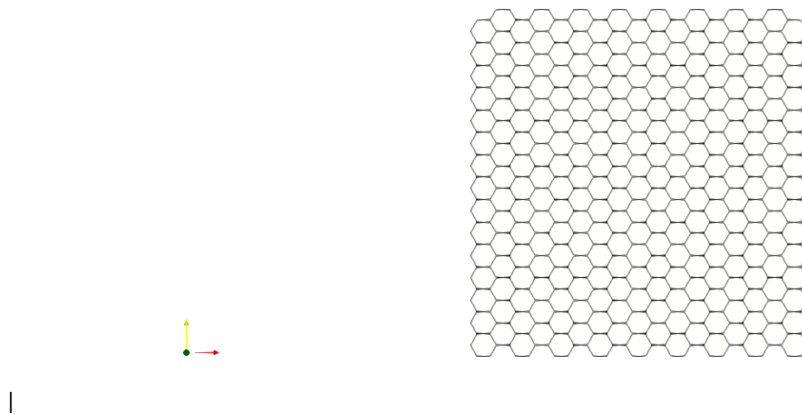
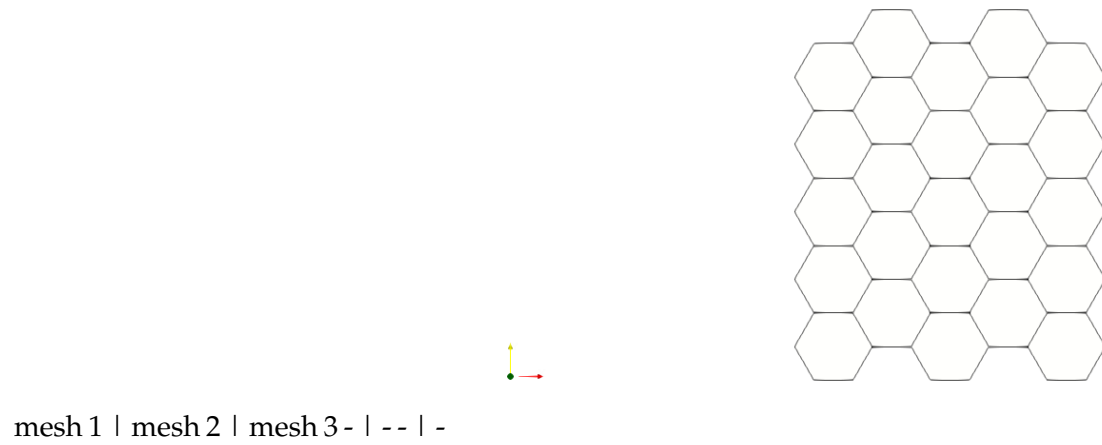
result 1 | result 2 | result 3 - | - - | -

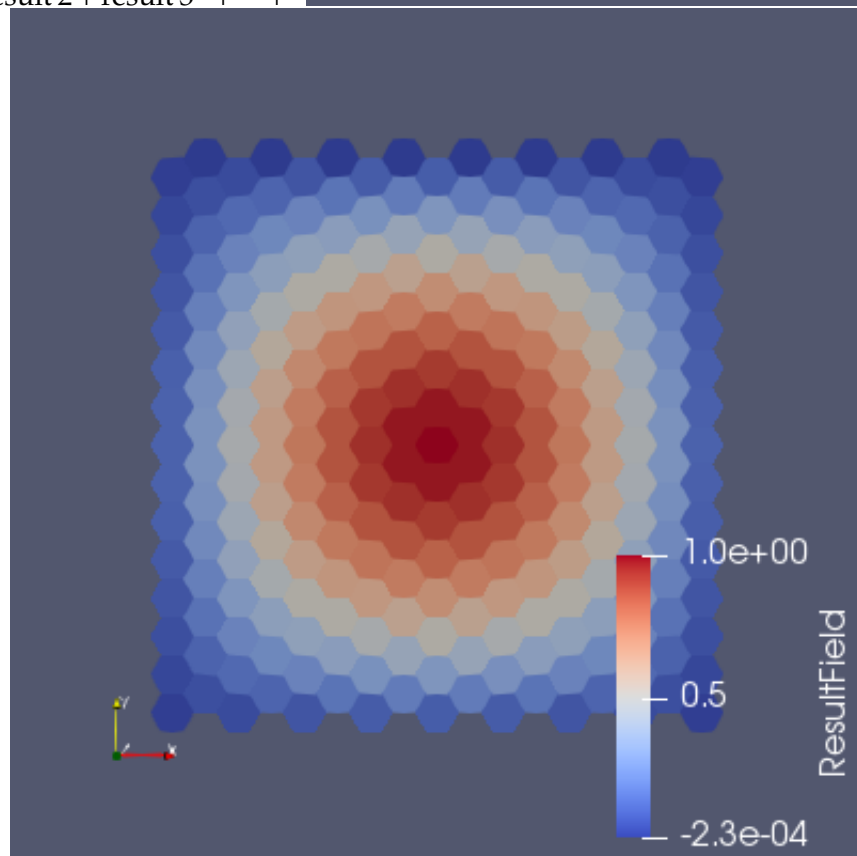
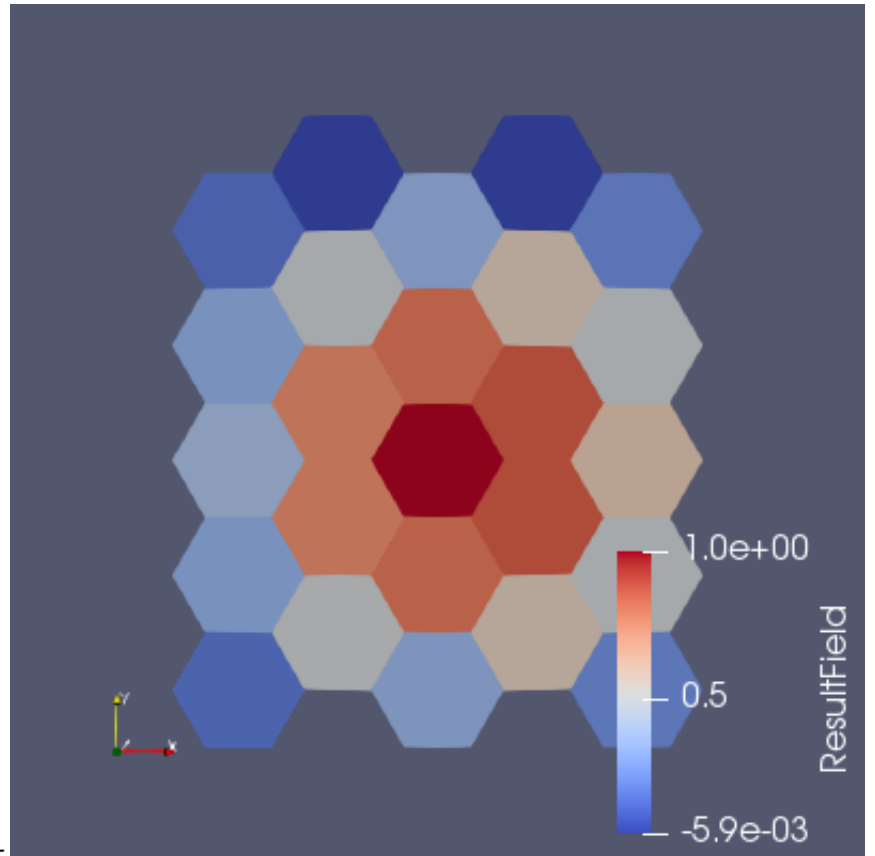


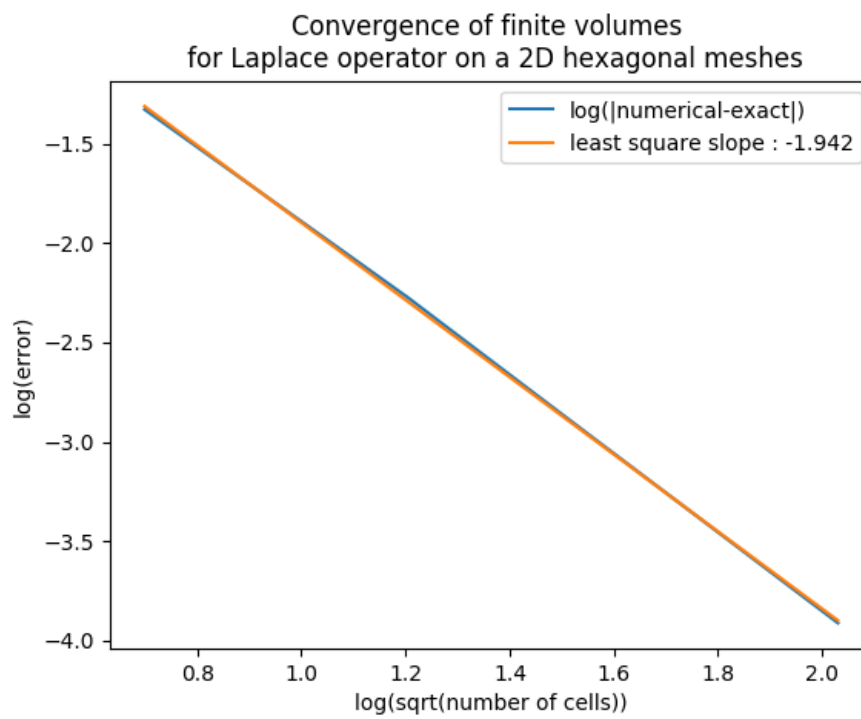
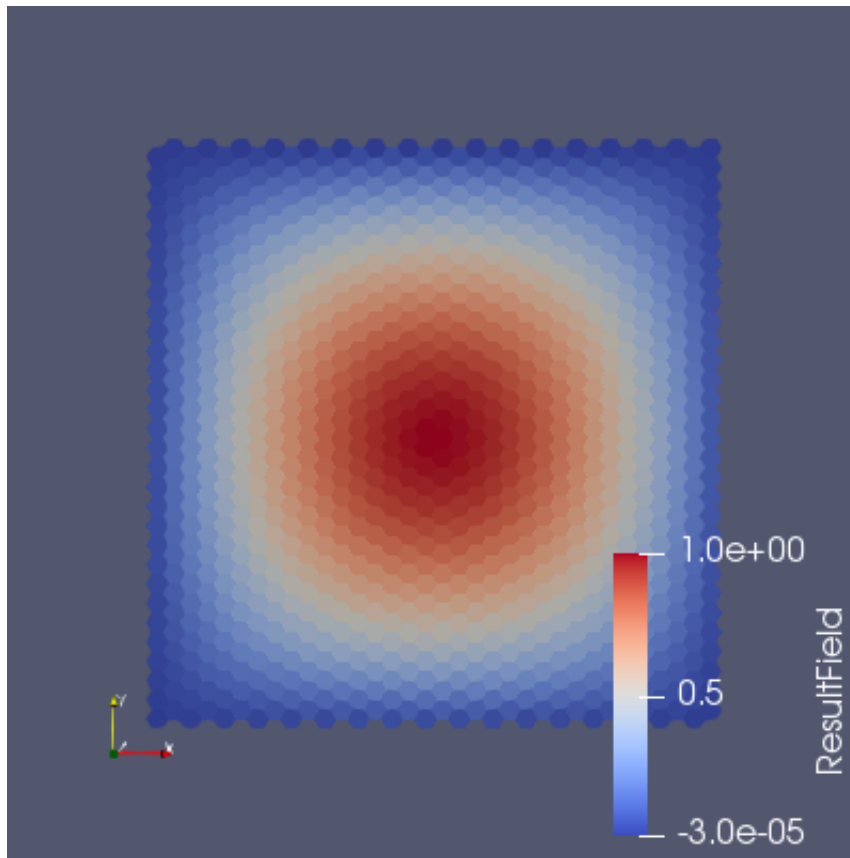




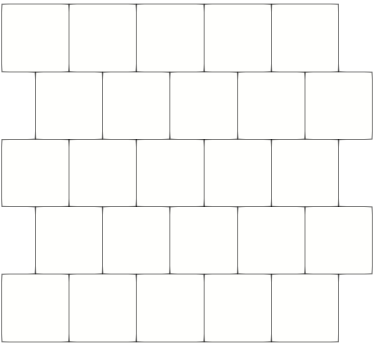
## 1.8 Hexagonal meshes



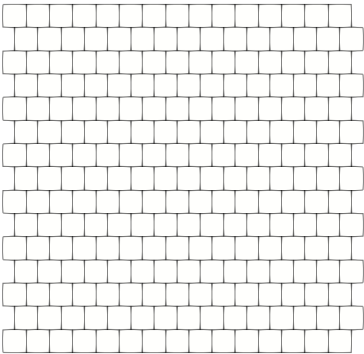




1.9 Brick wall meshes

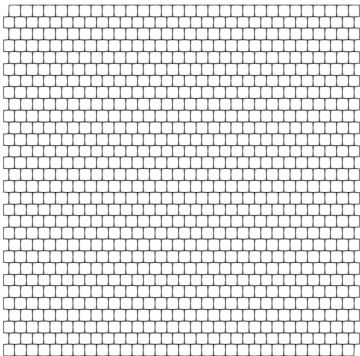


mesh 1 | mesh 2 | mesh 3 - | - - | -

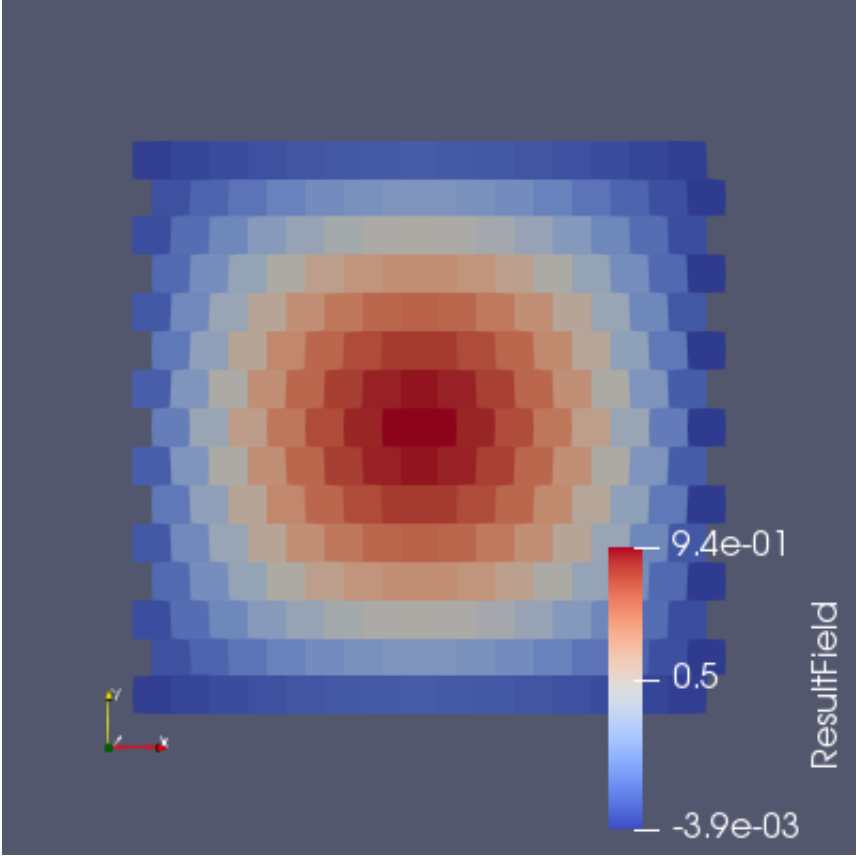
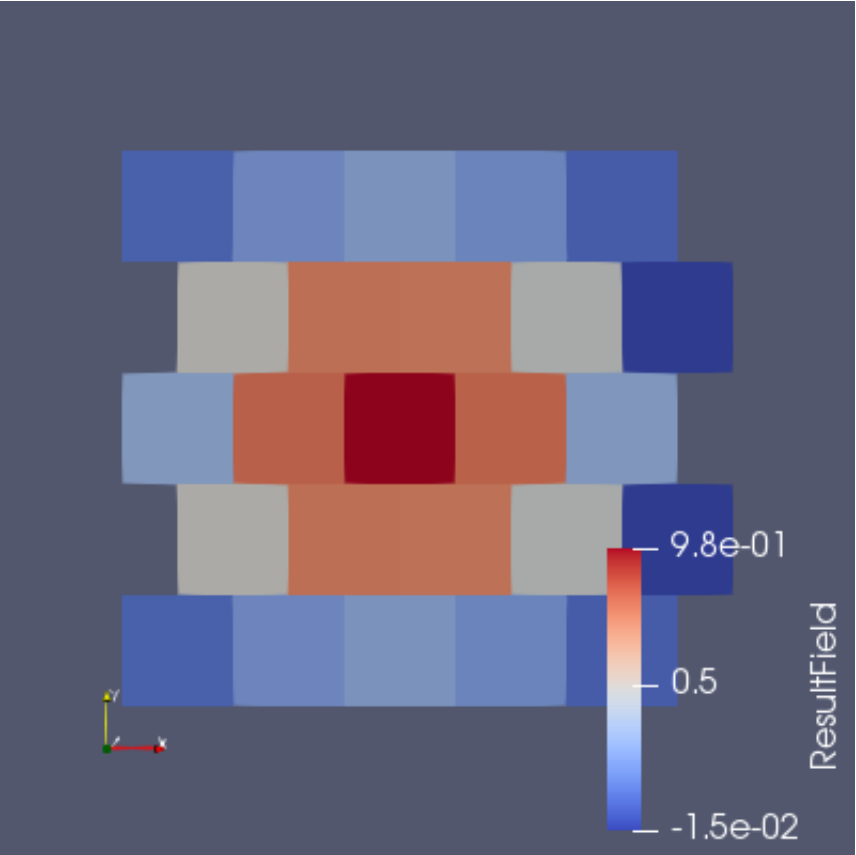


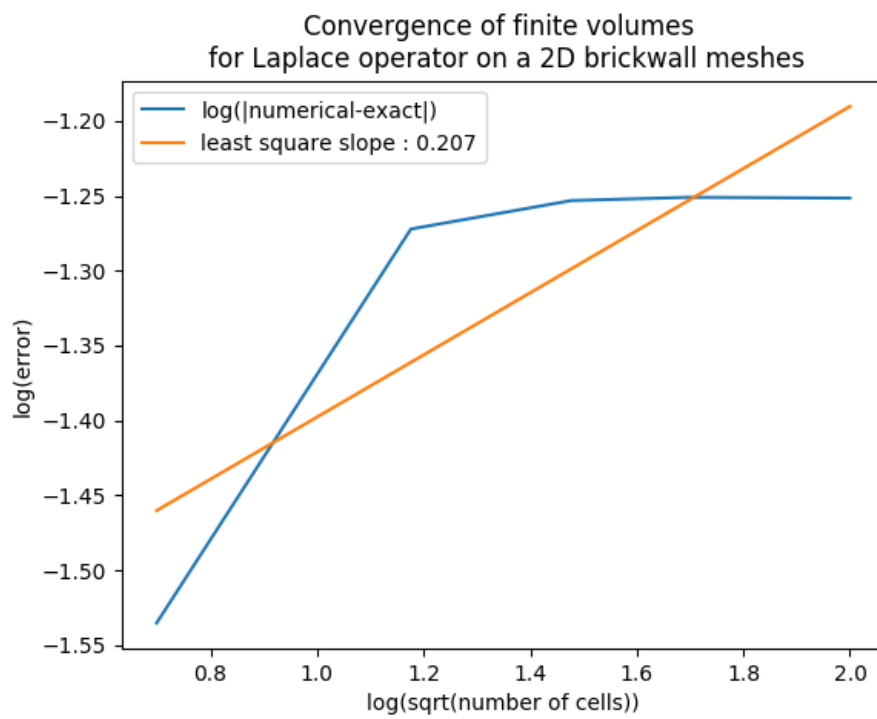
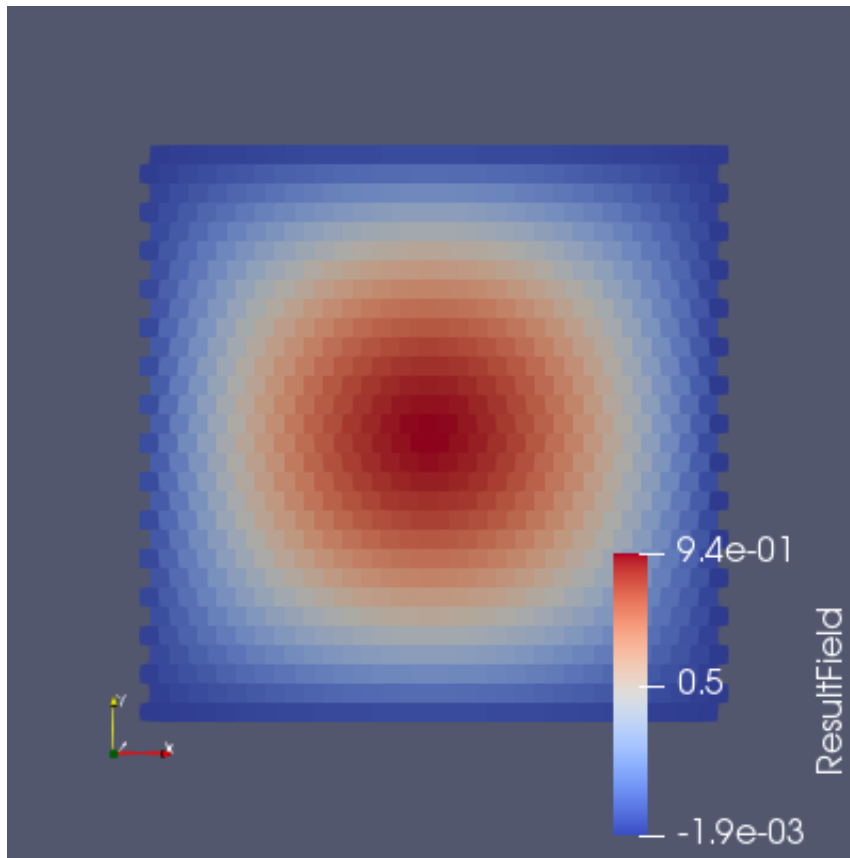
|

|

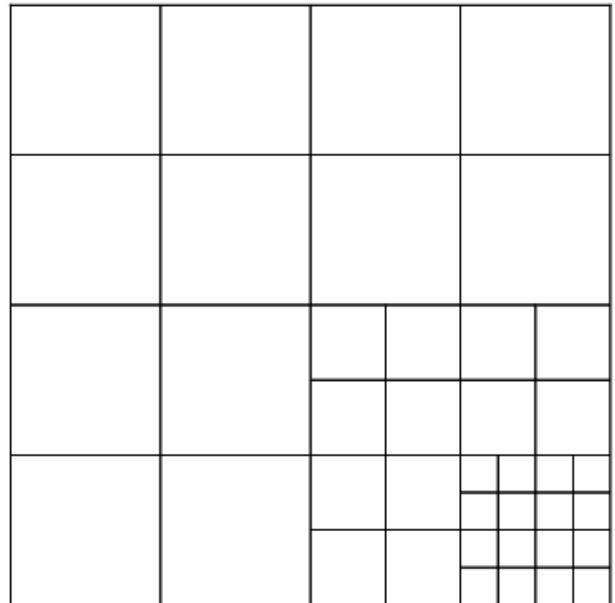


result 1 | result 2 | result 3 - | - - | -

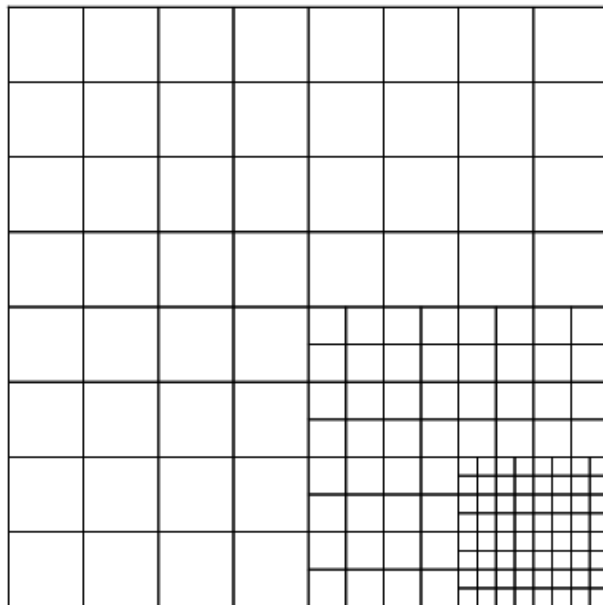




## 1.10 Locally refined meshes

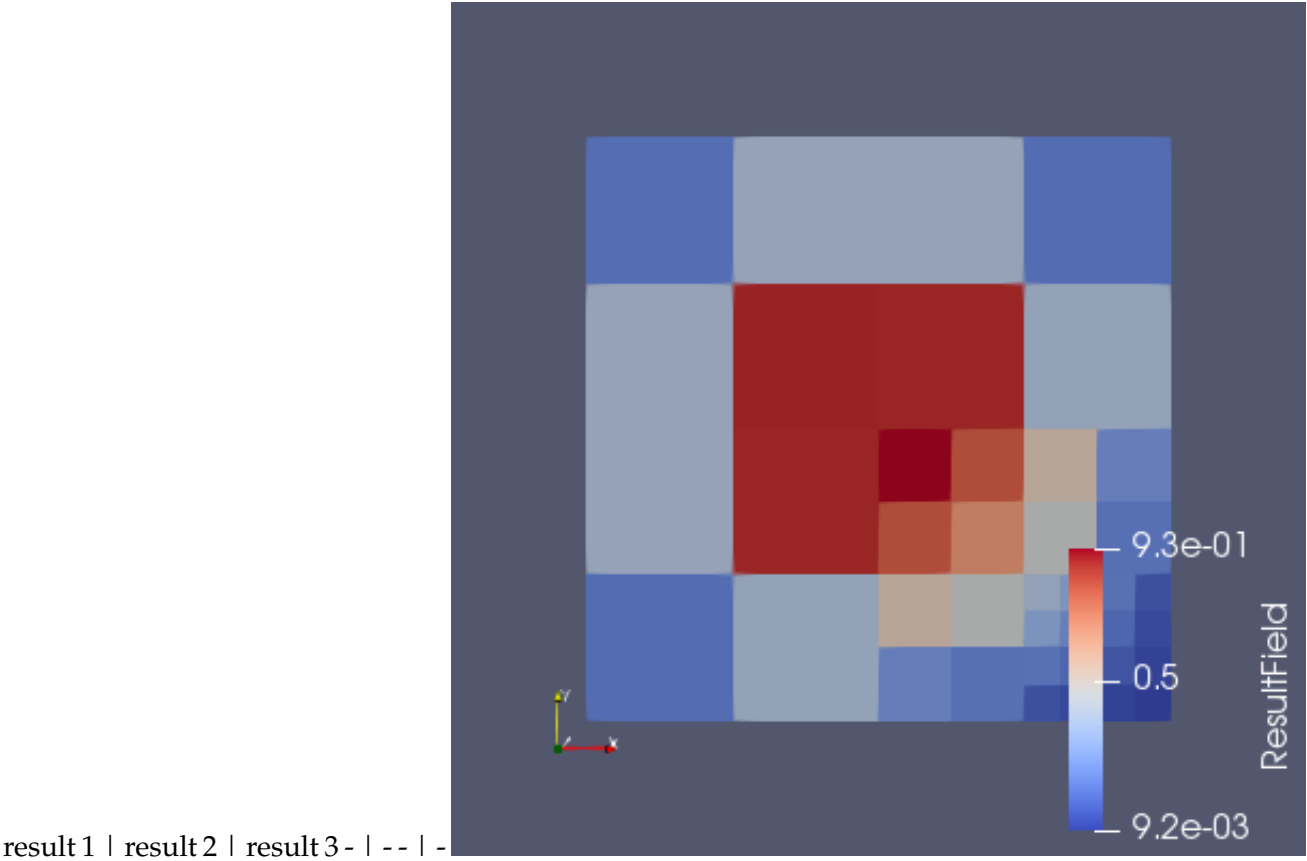
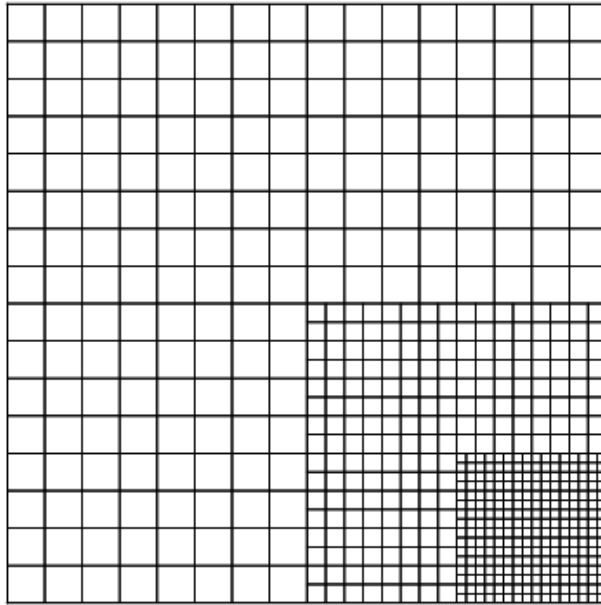


mesh 1 | mesh 2 | mesh 3 - | - - | -

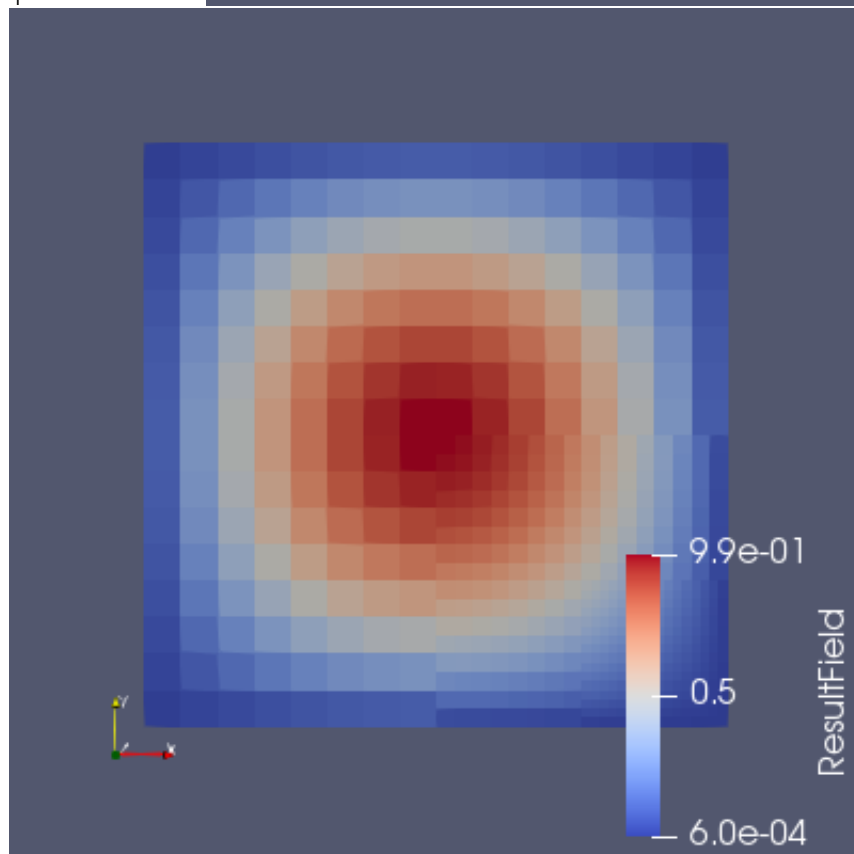
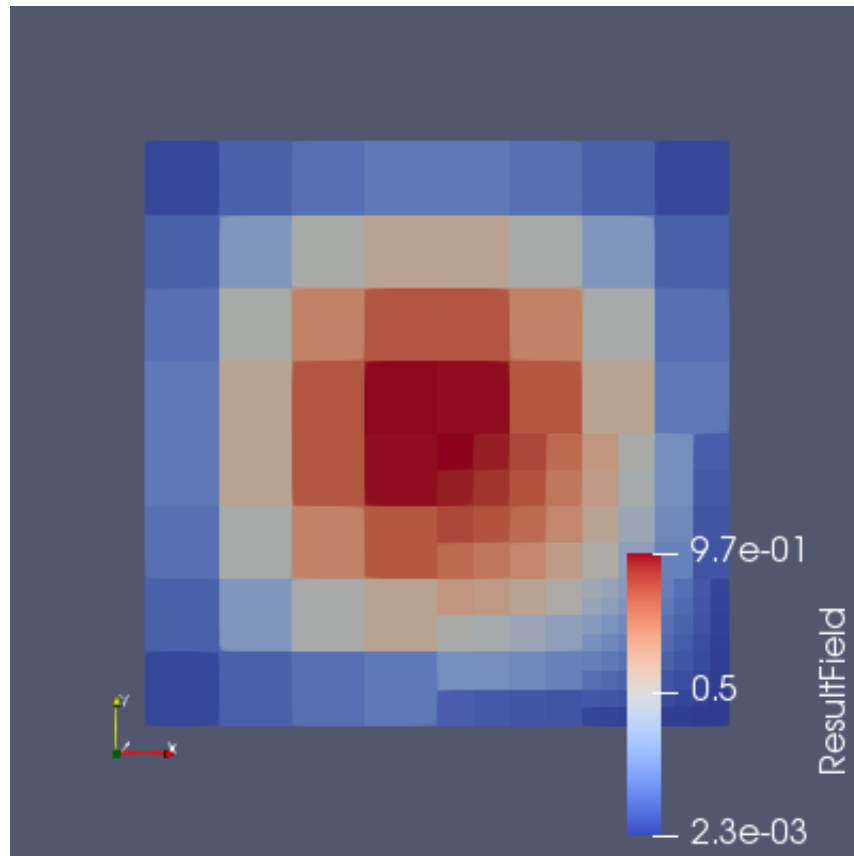


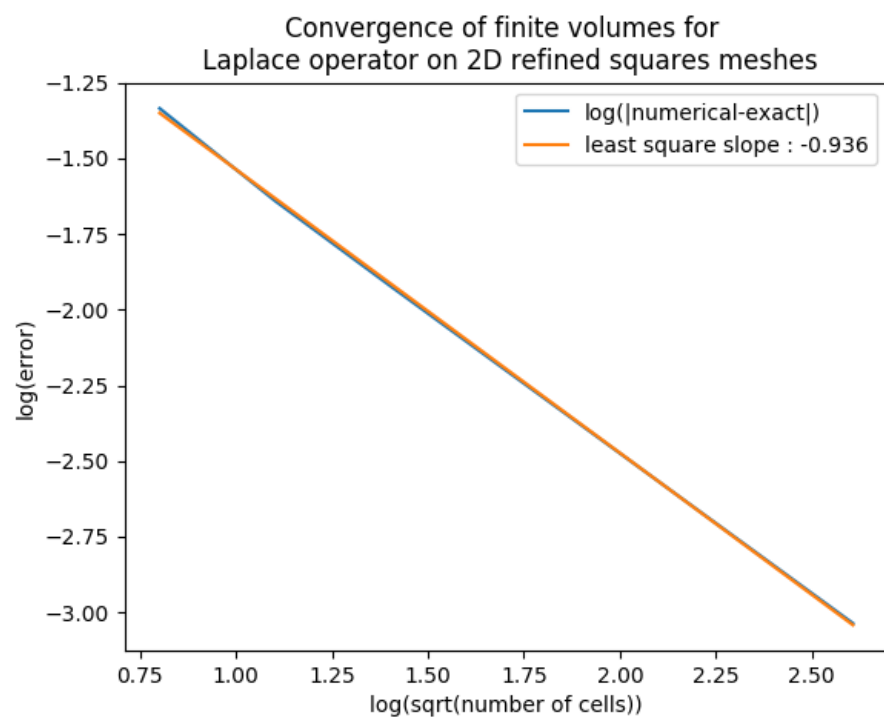
|

|

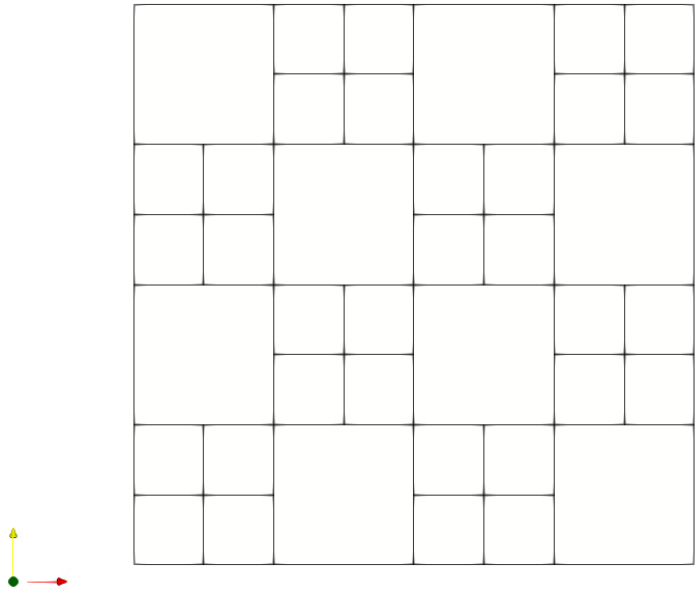




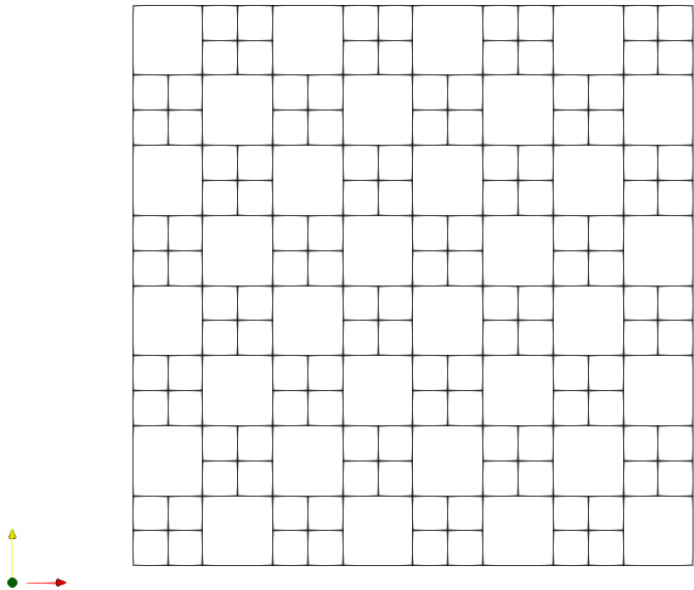




1.11 Checkerboard meshes

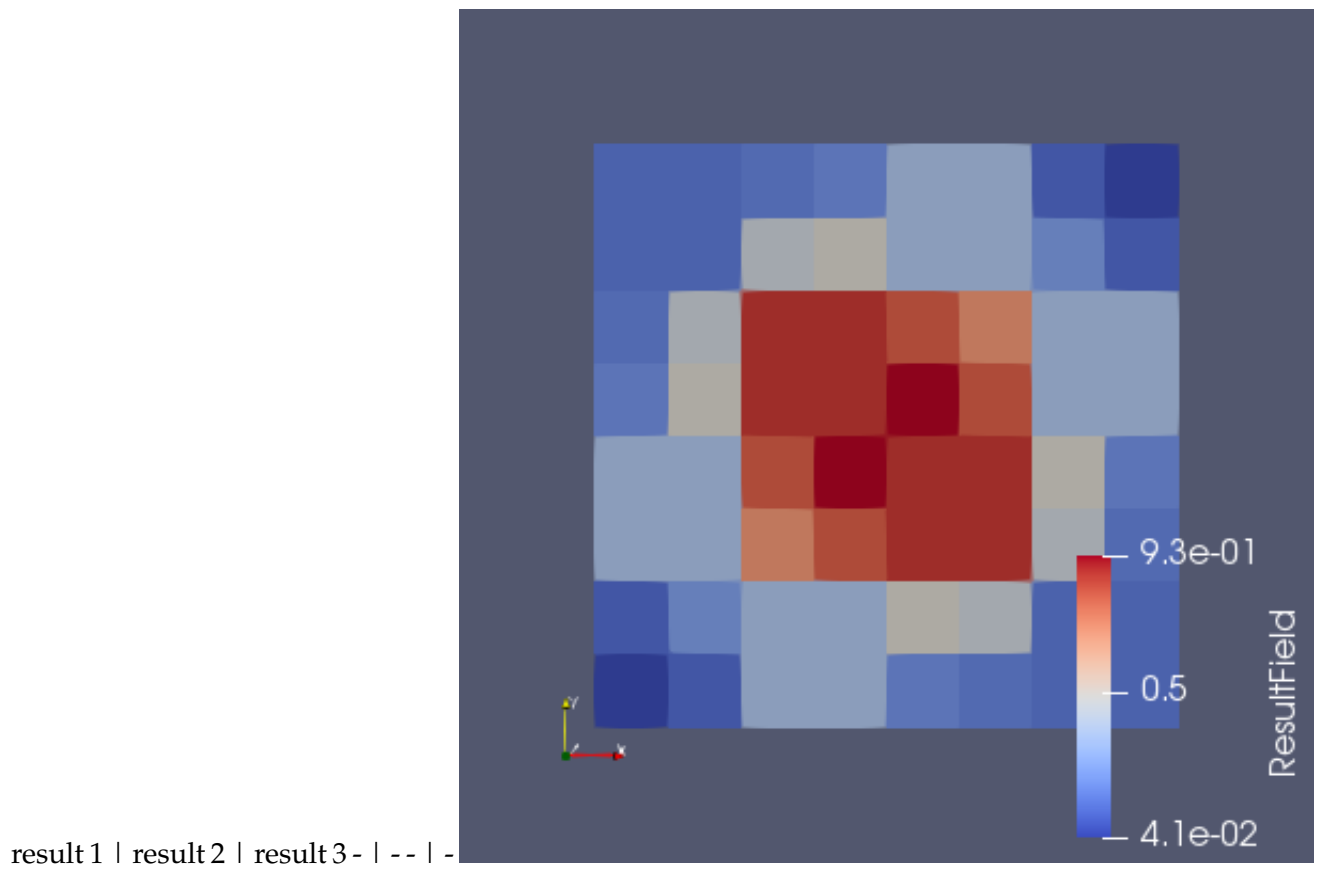
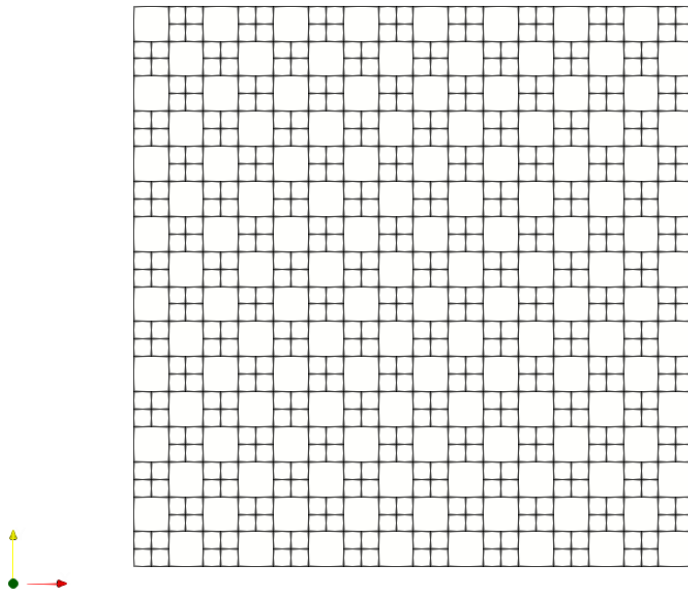


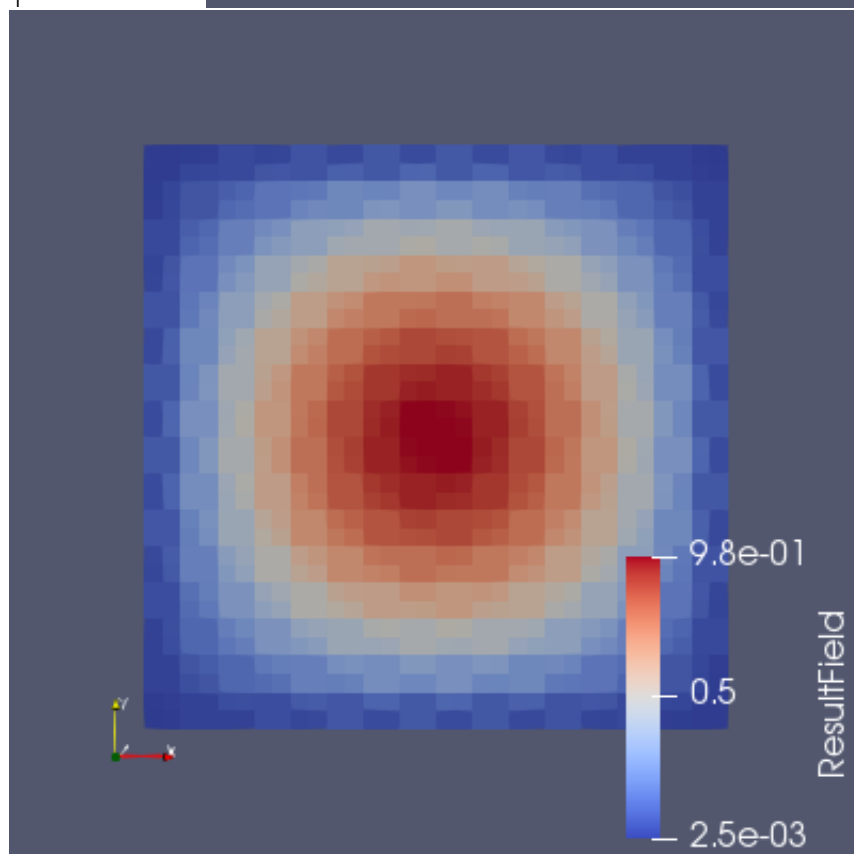
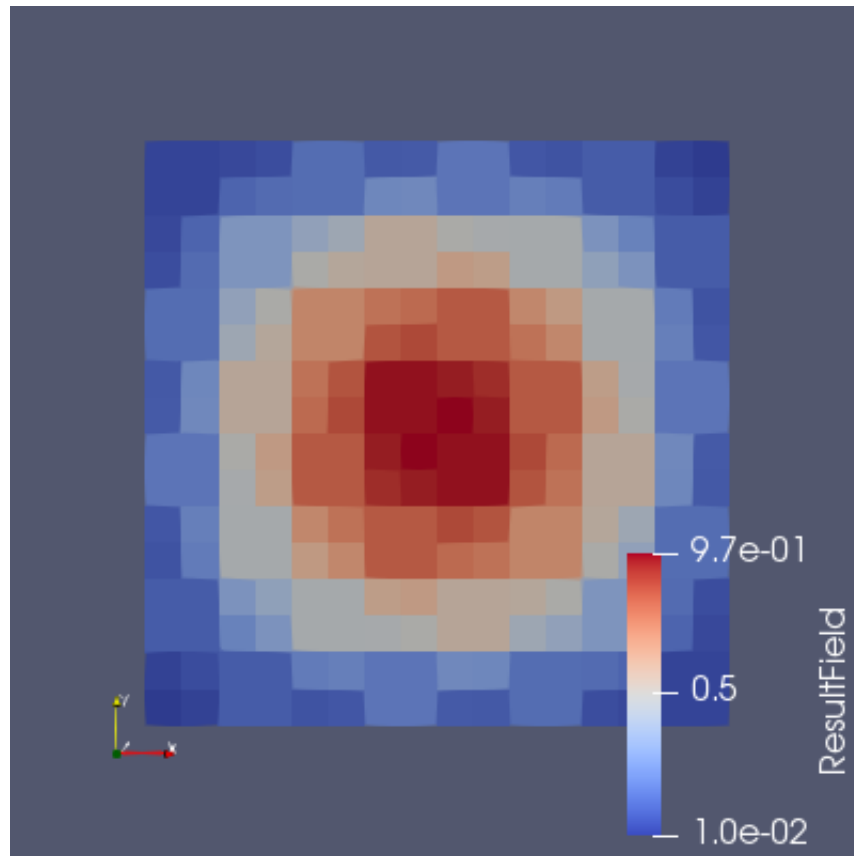
mesh 1 | mesh 2 | mesh 3 - | - - | -

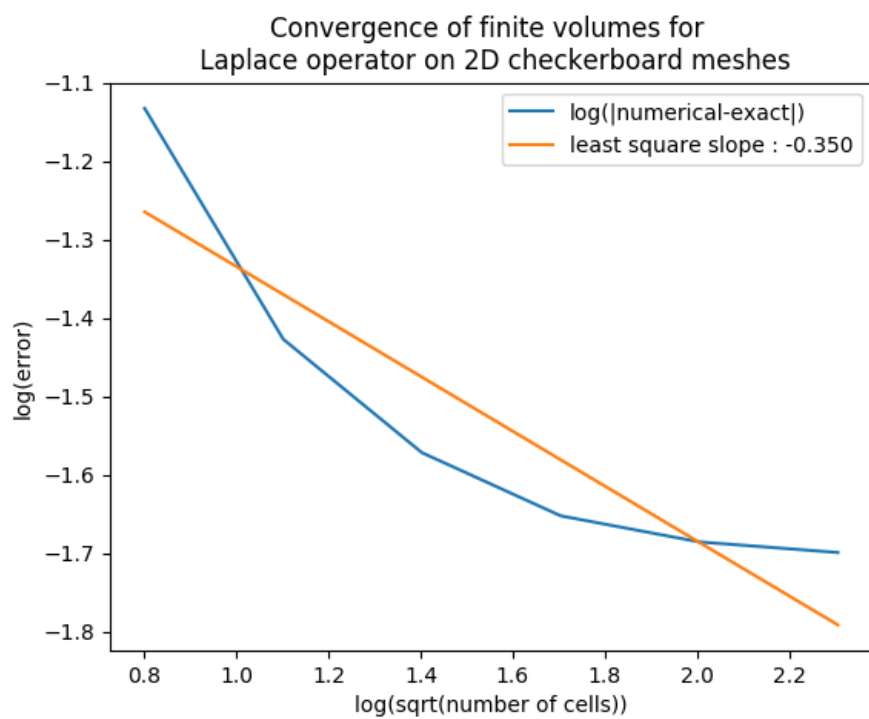


|

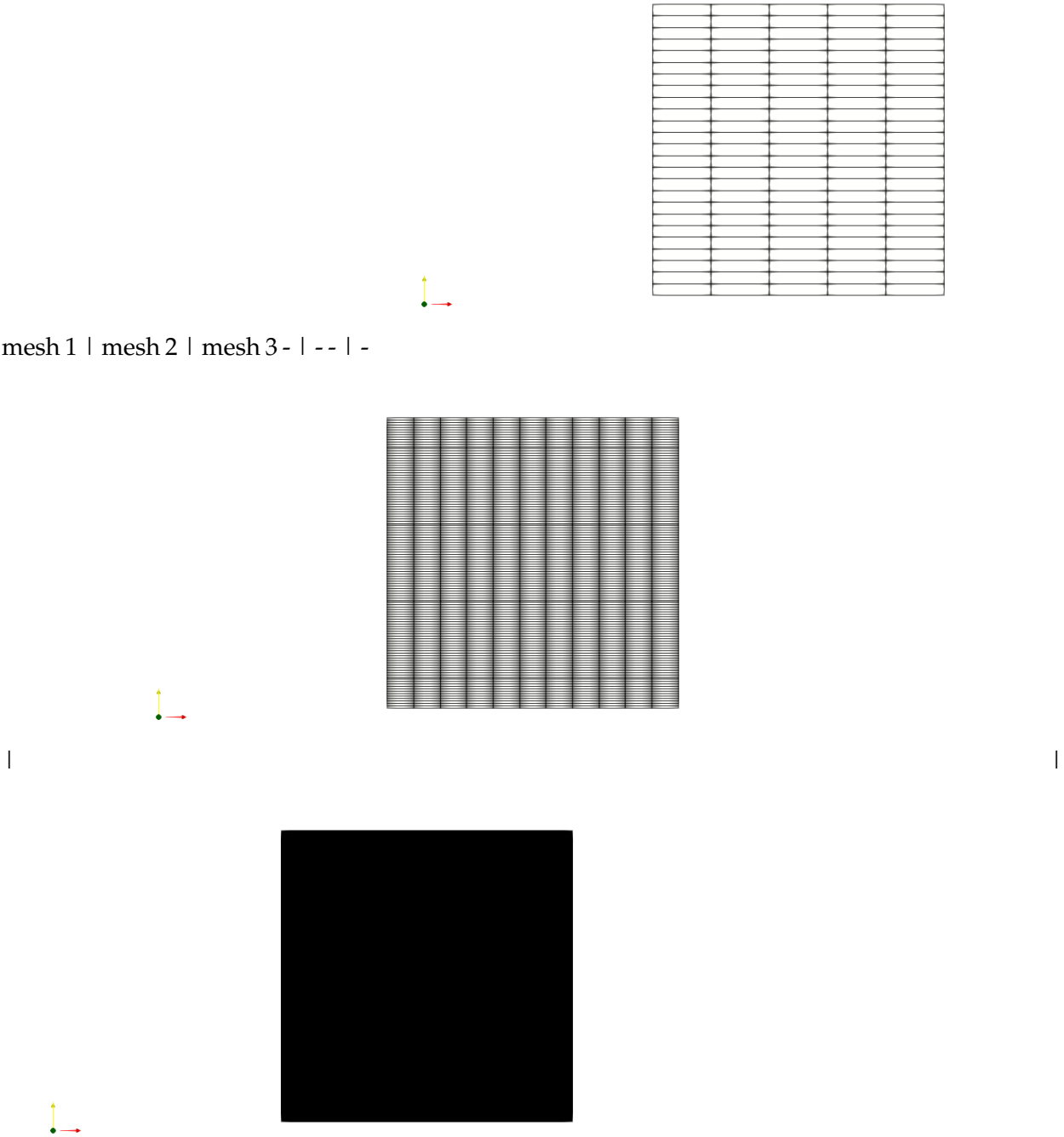
|

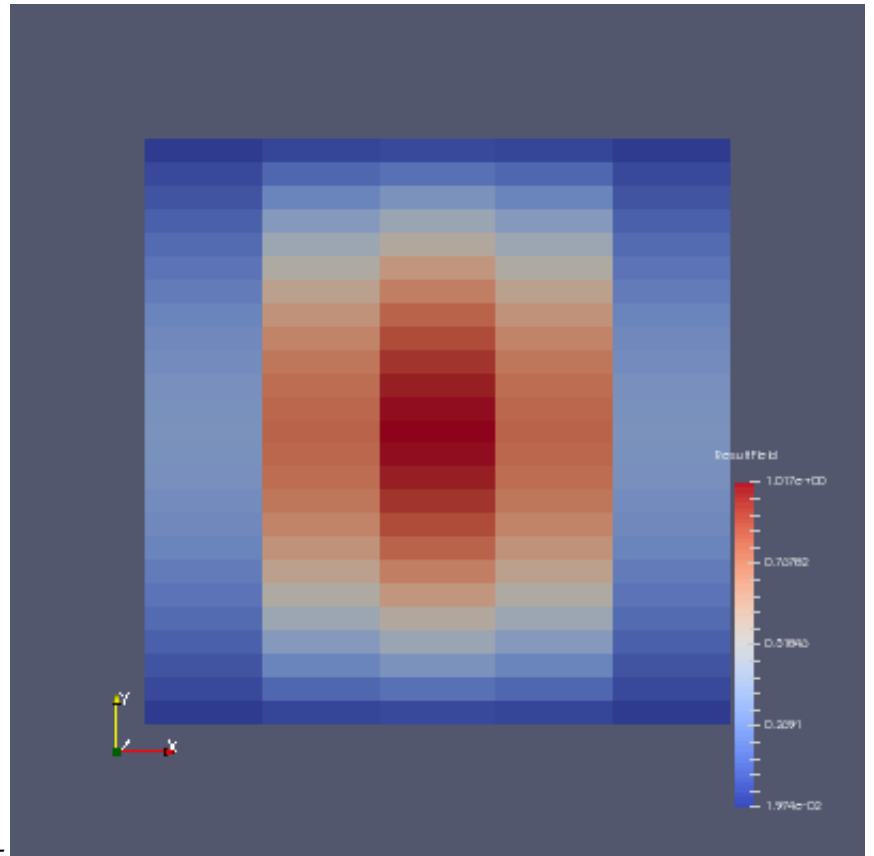




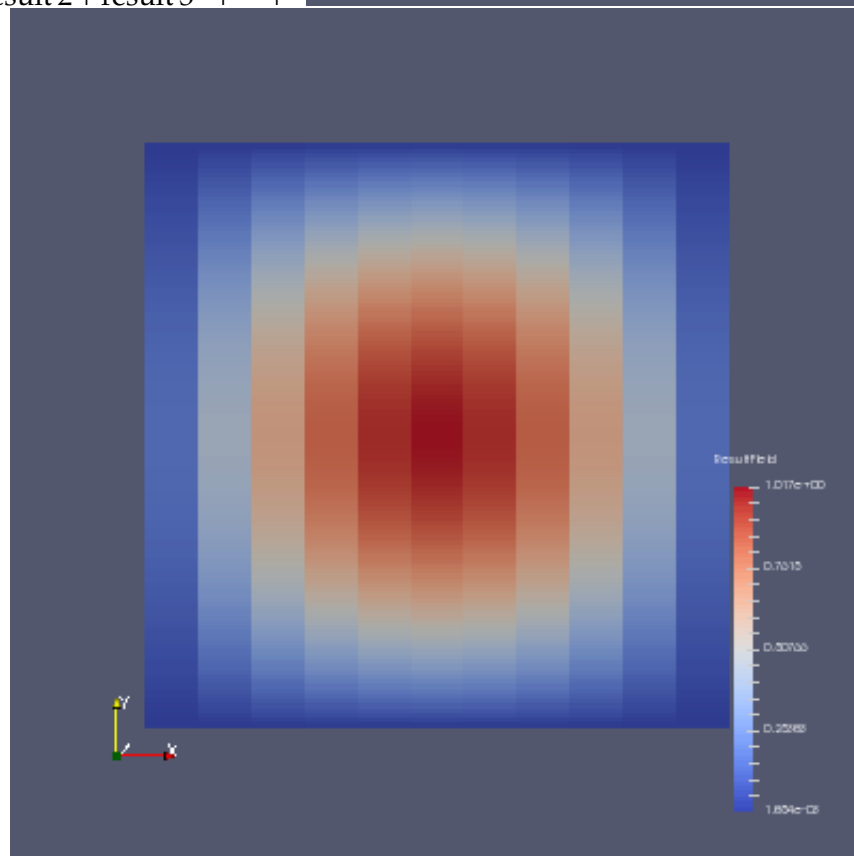


1.12 Long rectangle meshes

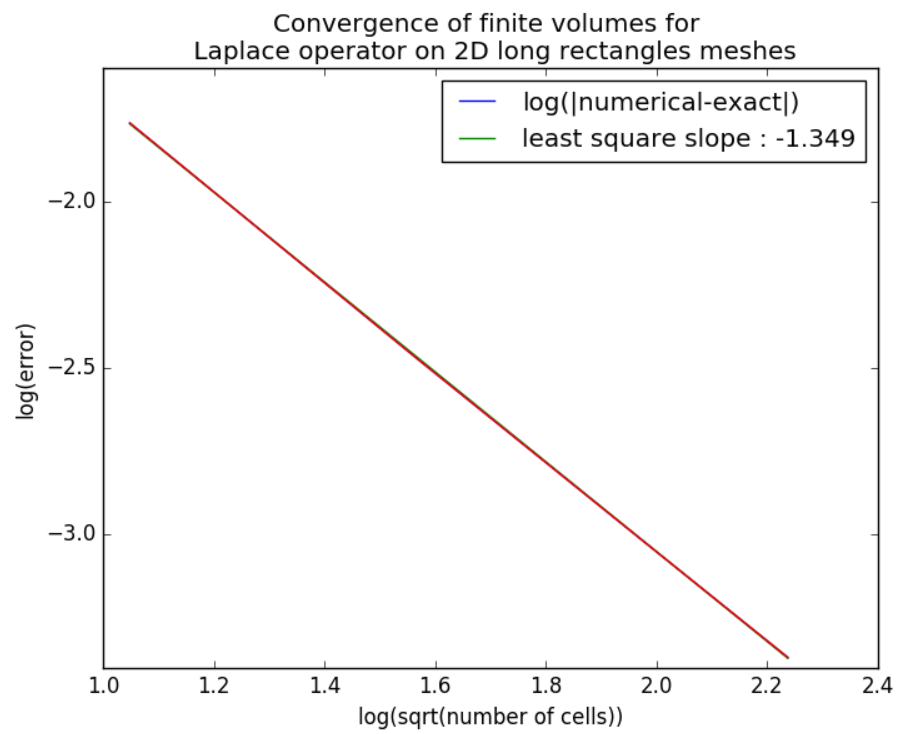
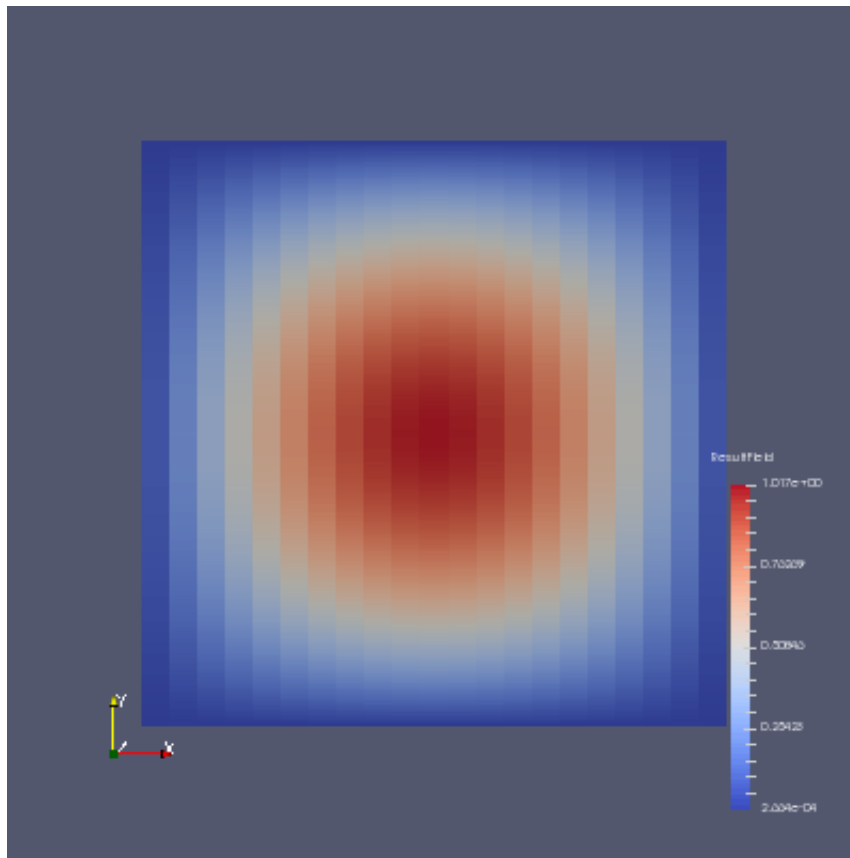




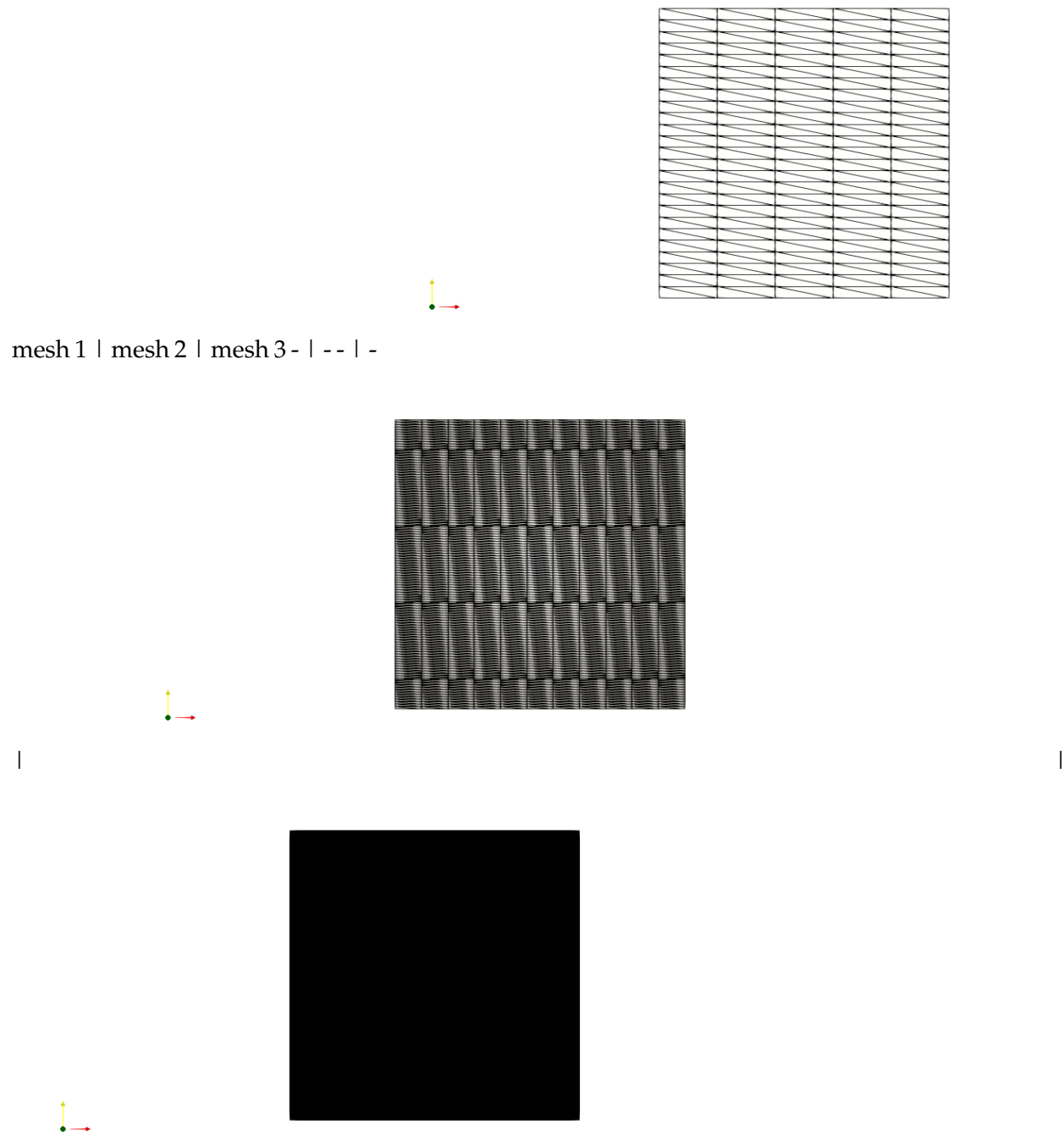
result 1 | result 2 | result 3 - | - - | -

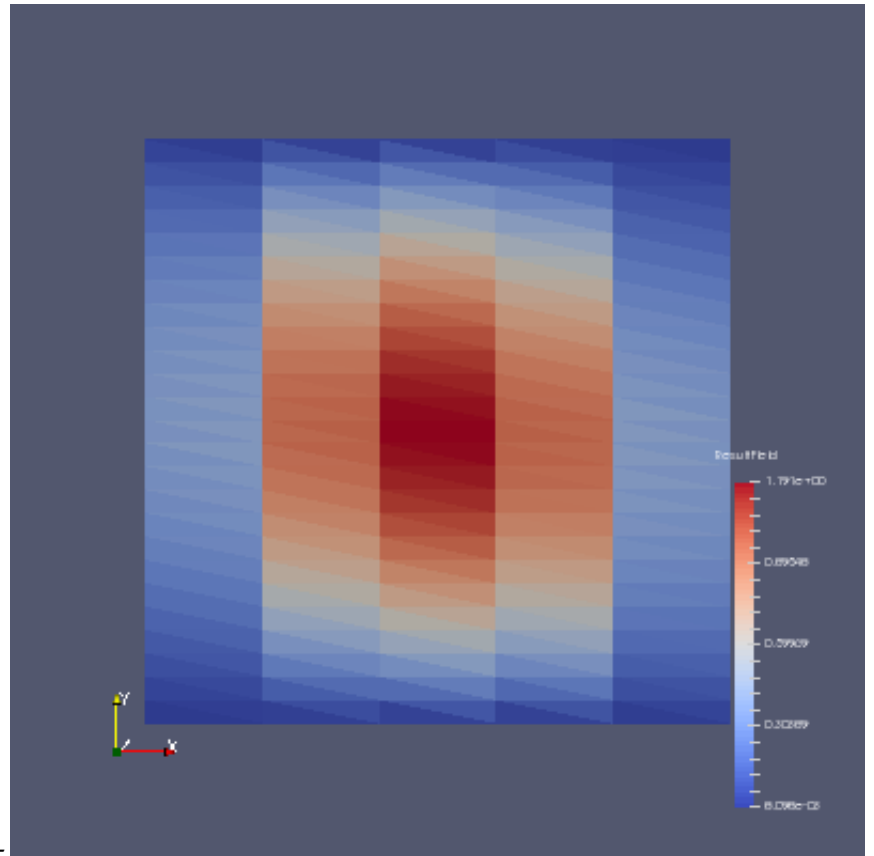






### 1.13 Long triangle meshes





result 1 | result 2 | result 3 - | -- | -

