

# Convergence\_Poisson\_FV5\_SQUARE

September 29, 2018

## 1 FV5 scheme for Poisson equation

### 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

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

### 1.2 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, we use the formula

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

### 1.3 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

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
    # 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
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.g
            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()

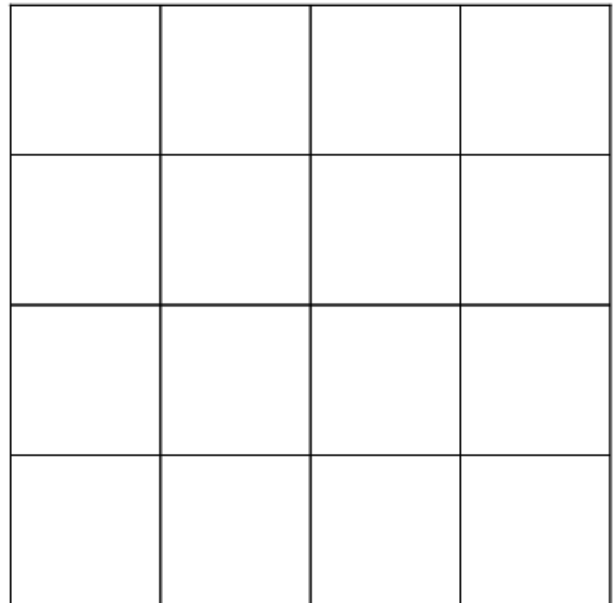
# Automatic postprocessing : save 2D picture and plot diagonal data
#=====
diag_data=VTK_routines.Extract_field_data_over_line_to_numpyArray(my_ResultField,[0
plt.legend()
```

```

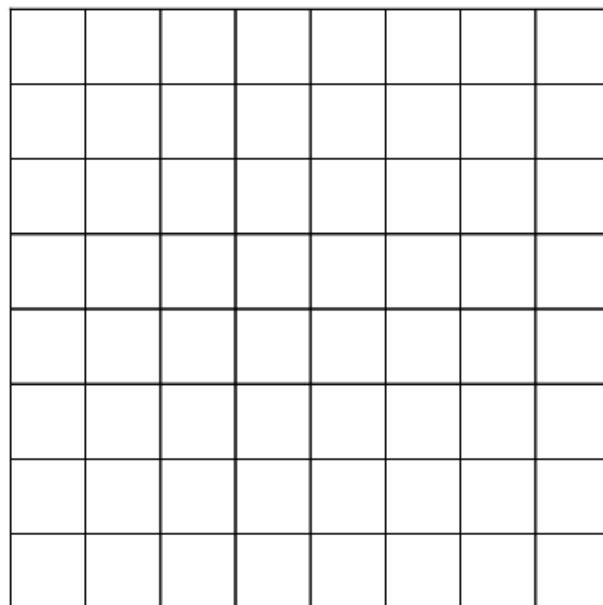
plt.xlabel('Position on diagonal line')
plt.ylabel('Value on diagonal line')
if len(sys.argv) > 1 :
    plt.title('Plot over diagonal line for finite Volumes \n for Laplace operator c
    plt.plot(curv_abs, diag_data, label= str(nbCells)+ ' cells mesh')
    plt.savefig("FiniteVolumes2D_square_ResultField_"+str(nbCells)+ '_cells'+ "_Plot

```

## 1.4 Regular grid

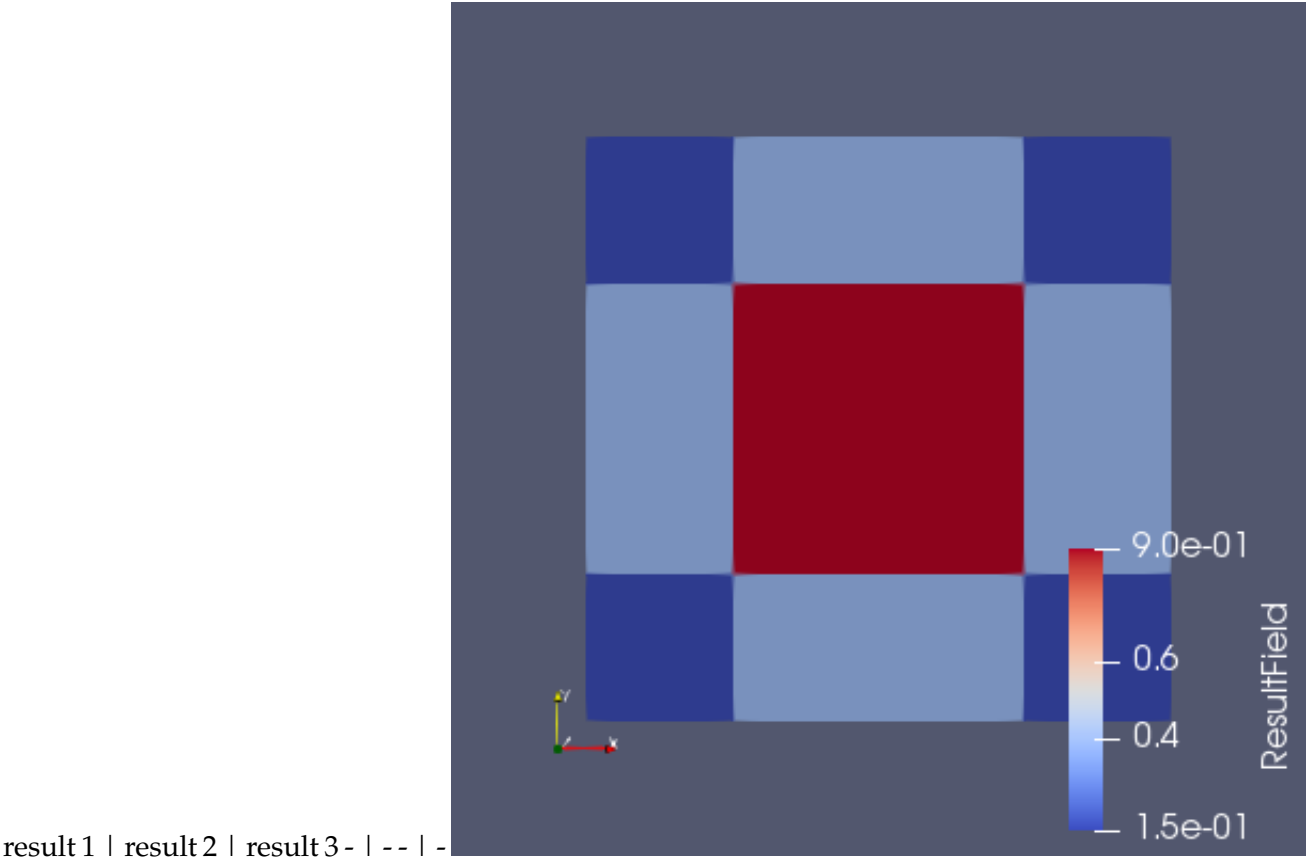
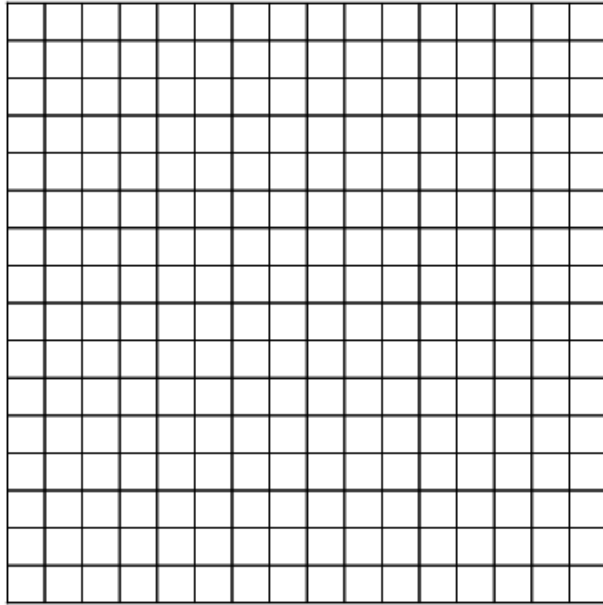


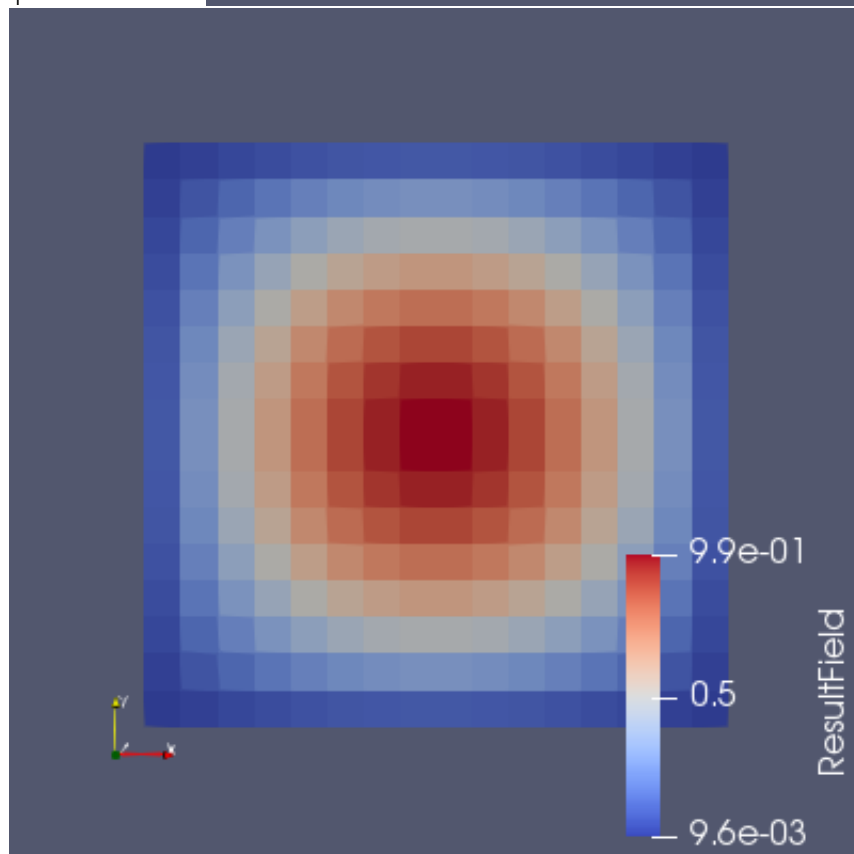
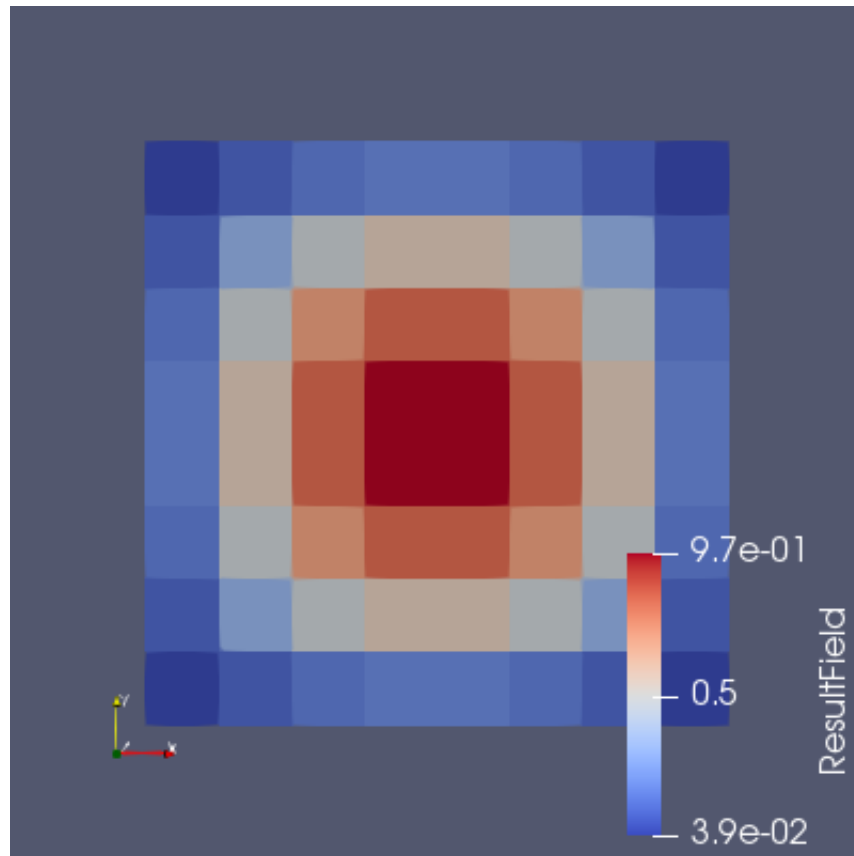
mesh 1 | mesh 2 | mesh 3 - | - - | -

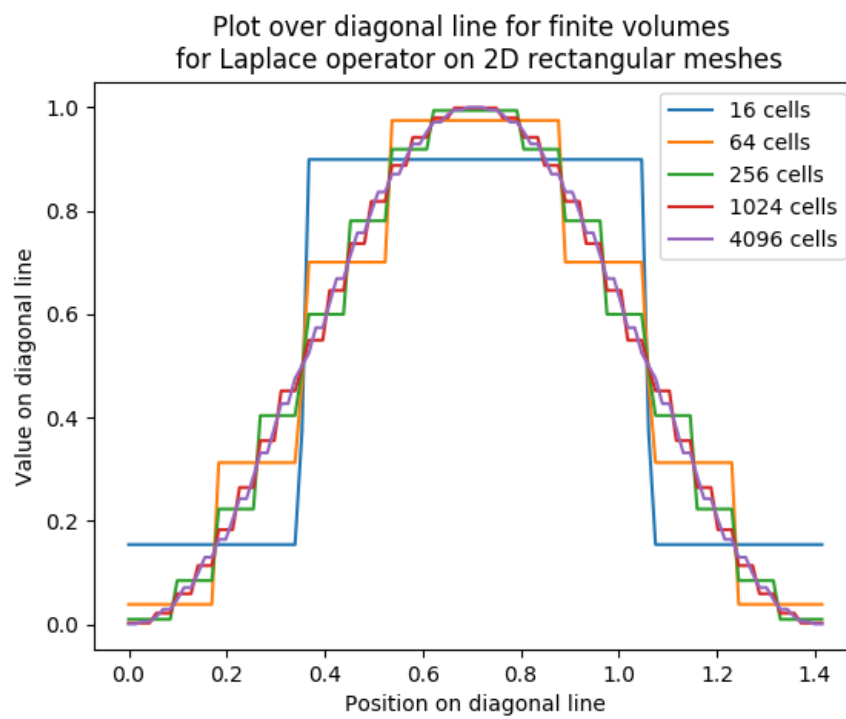
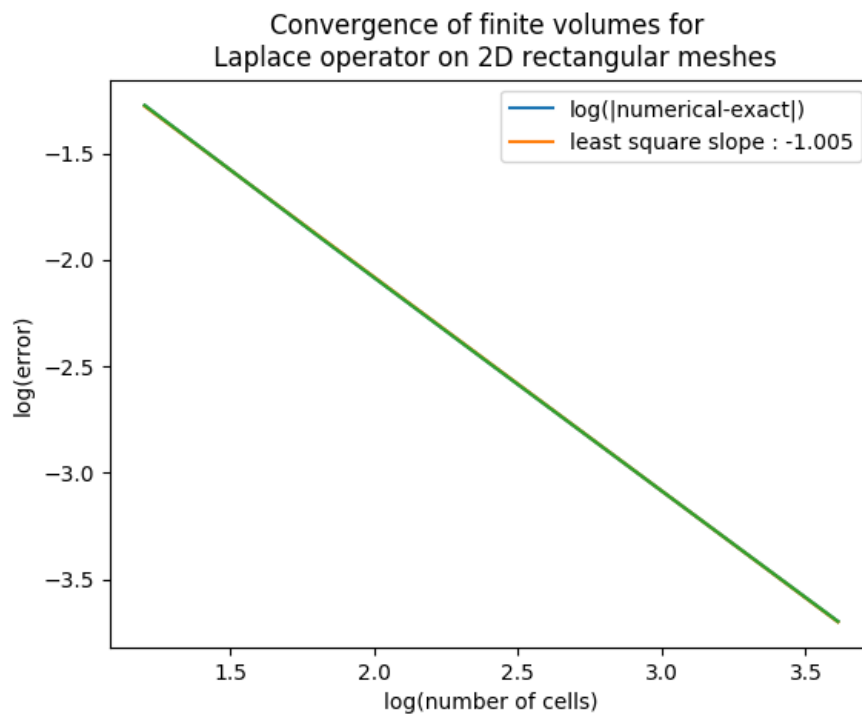


|

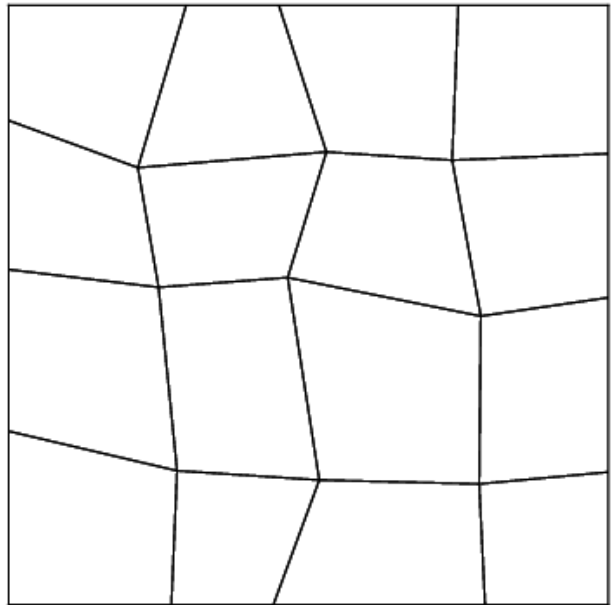
|



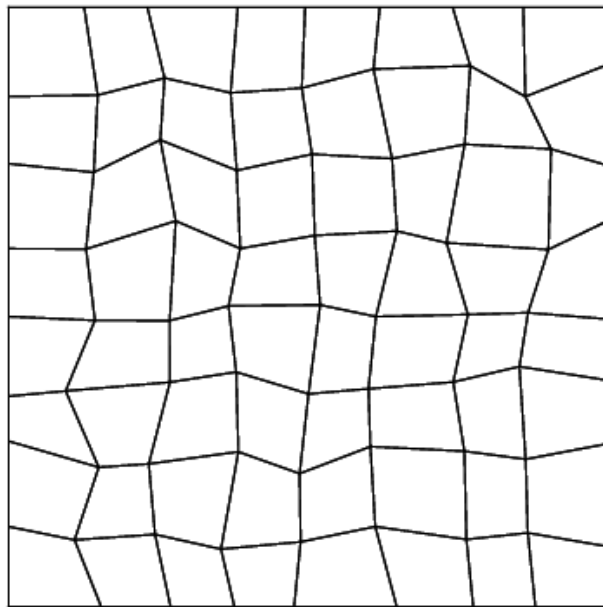




## 1.5 Deformed quadrangles



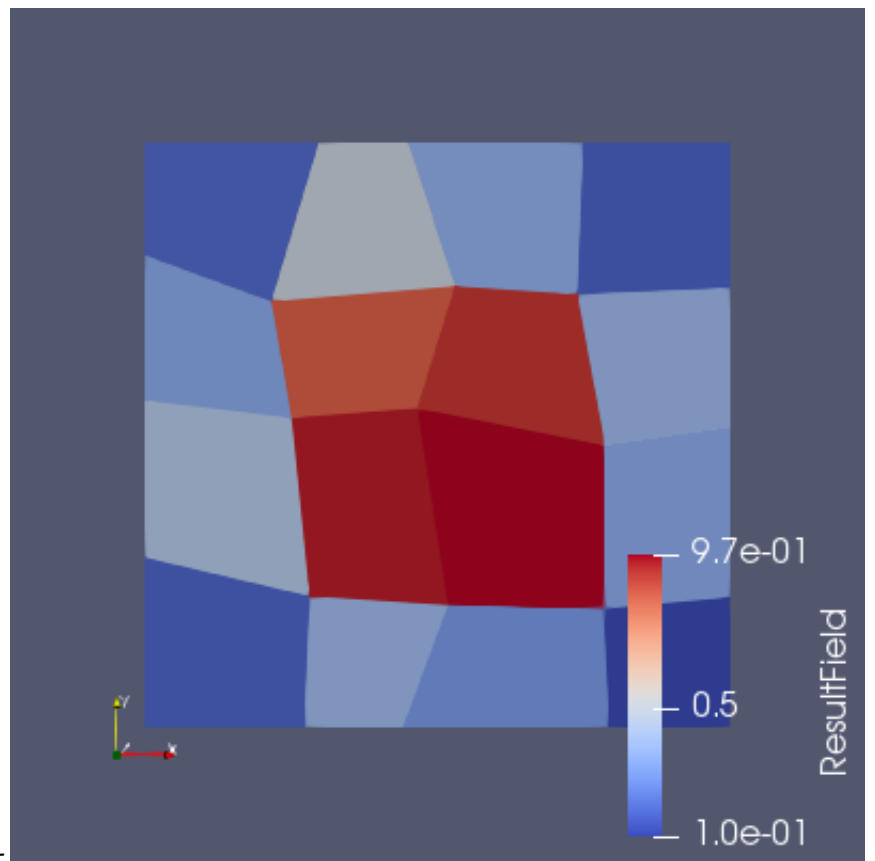
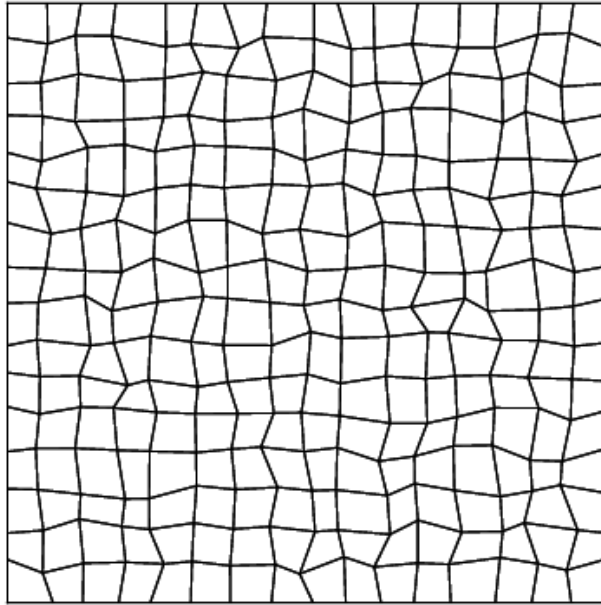
mesh 1 | mesh 2 | mesh 3 - | - - | -



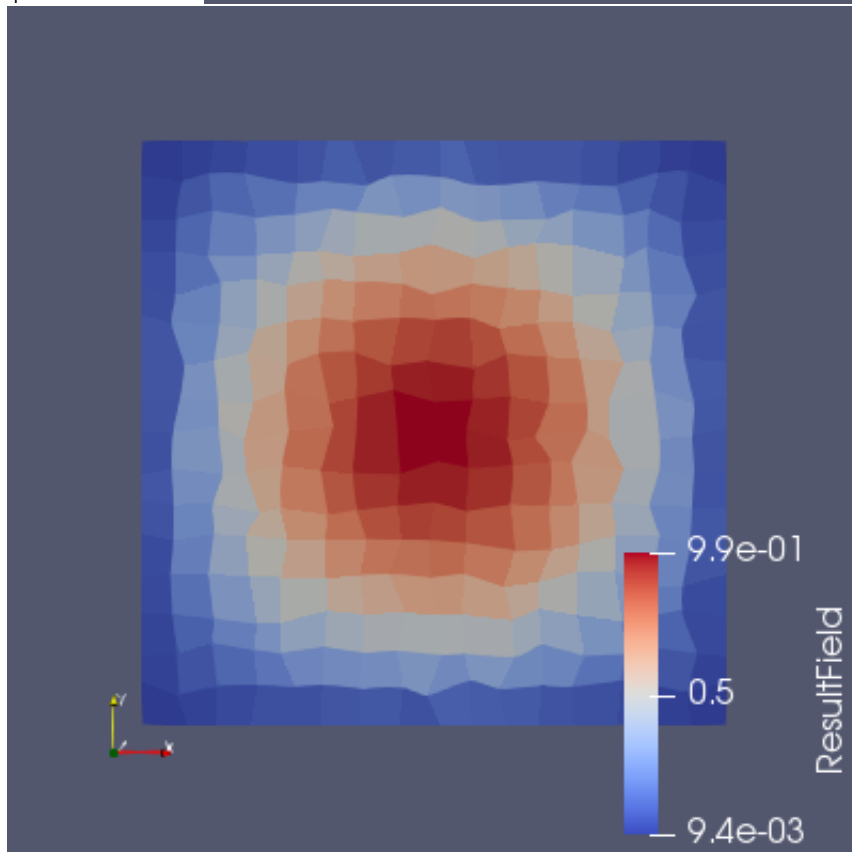
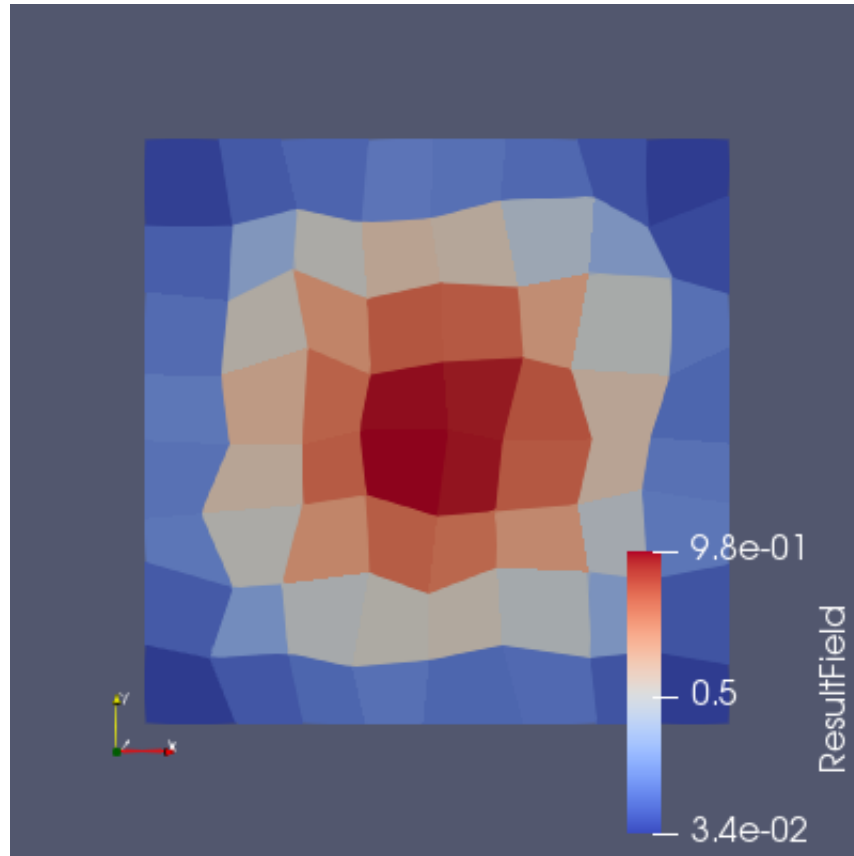
|

|

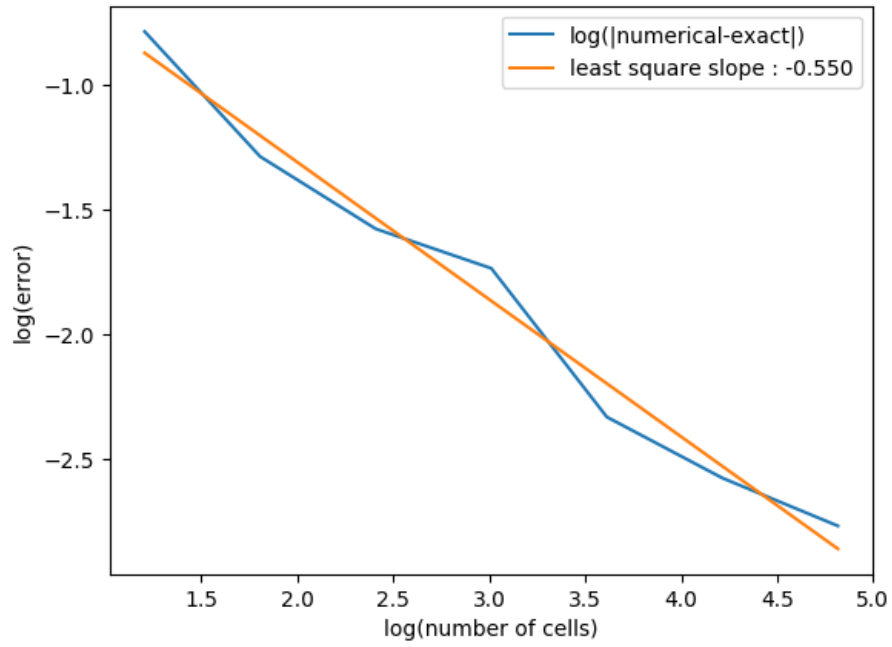




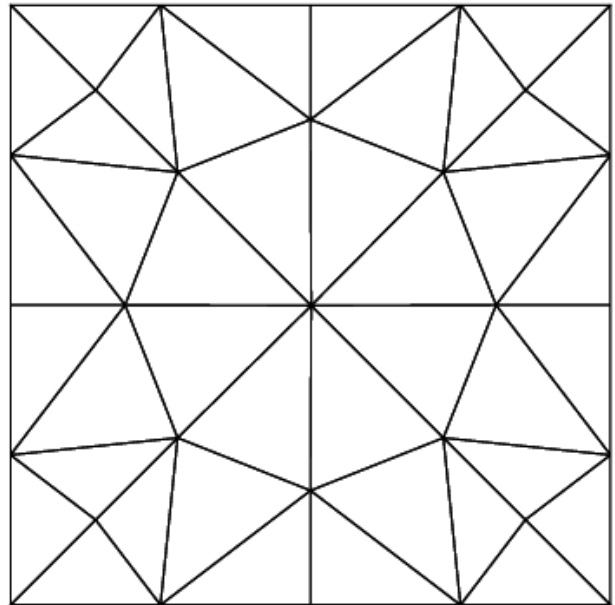
result 1 | result 2 | result 3 - | - - | -



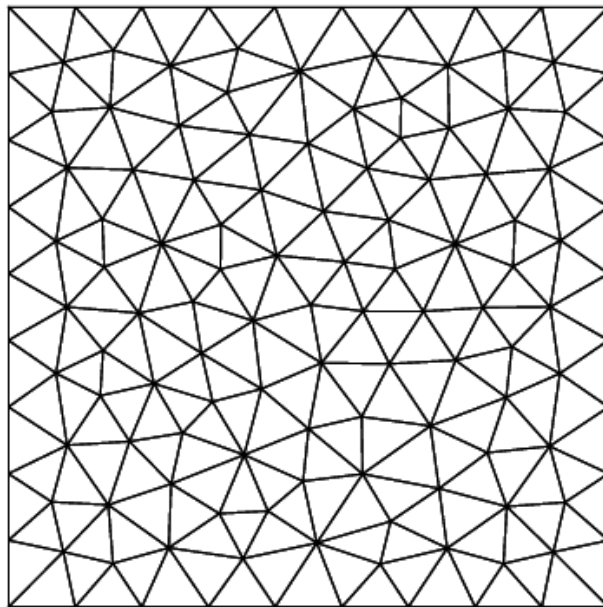
Convergence of finite volumes  
for Laplace operator on a 2D deformed quadrangles meshes



## 1.6 Triangular meshes

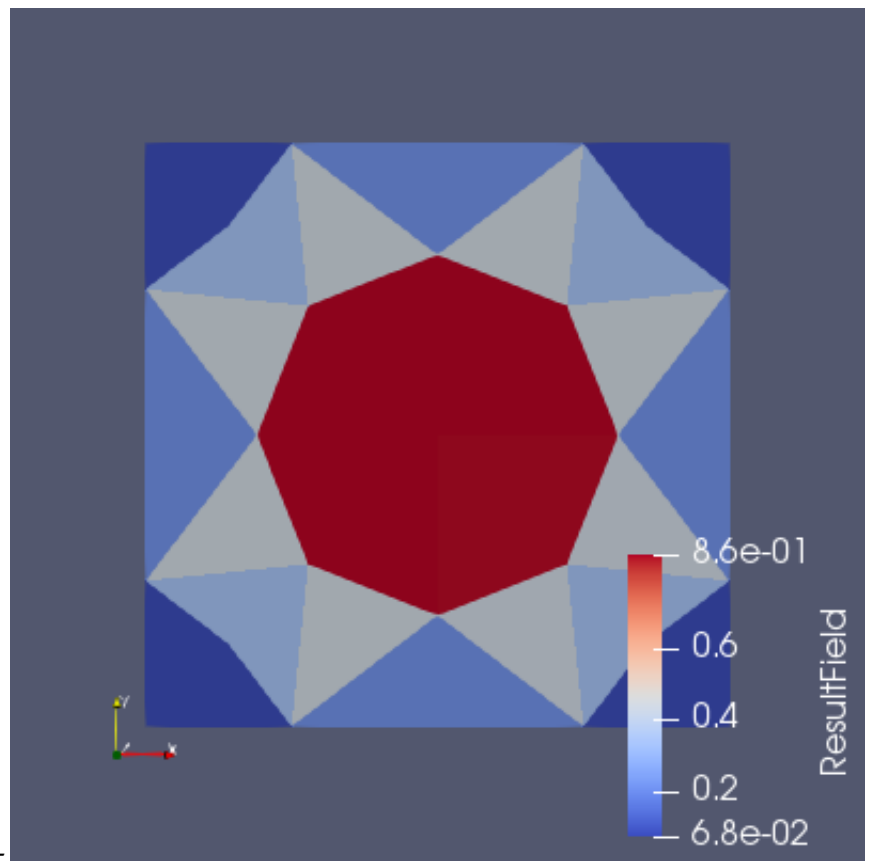
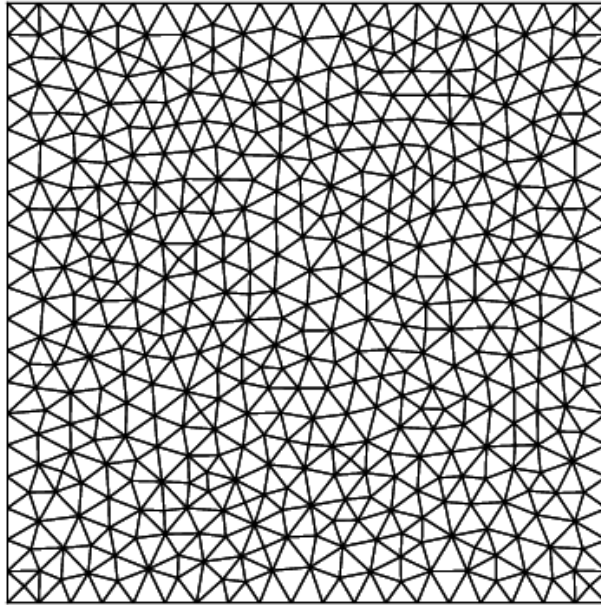


mesh 1 | mesh 2 | mesh 3 - | - - | -

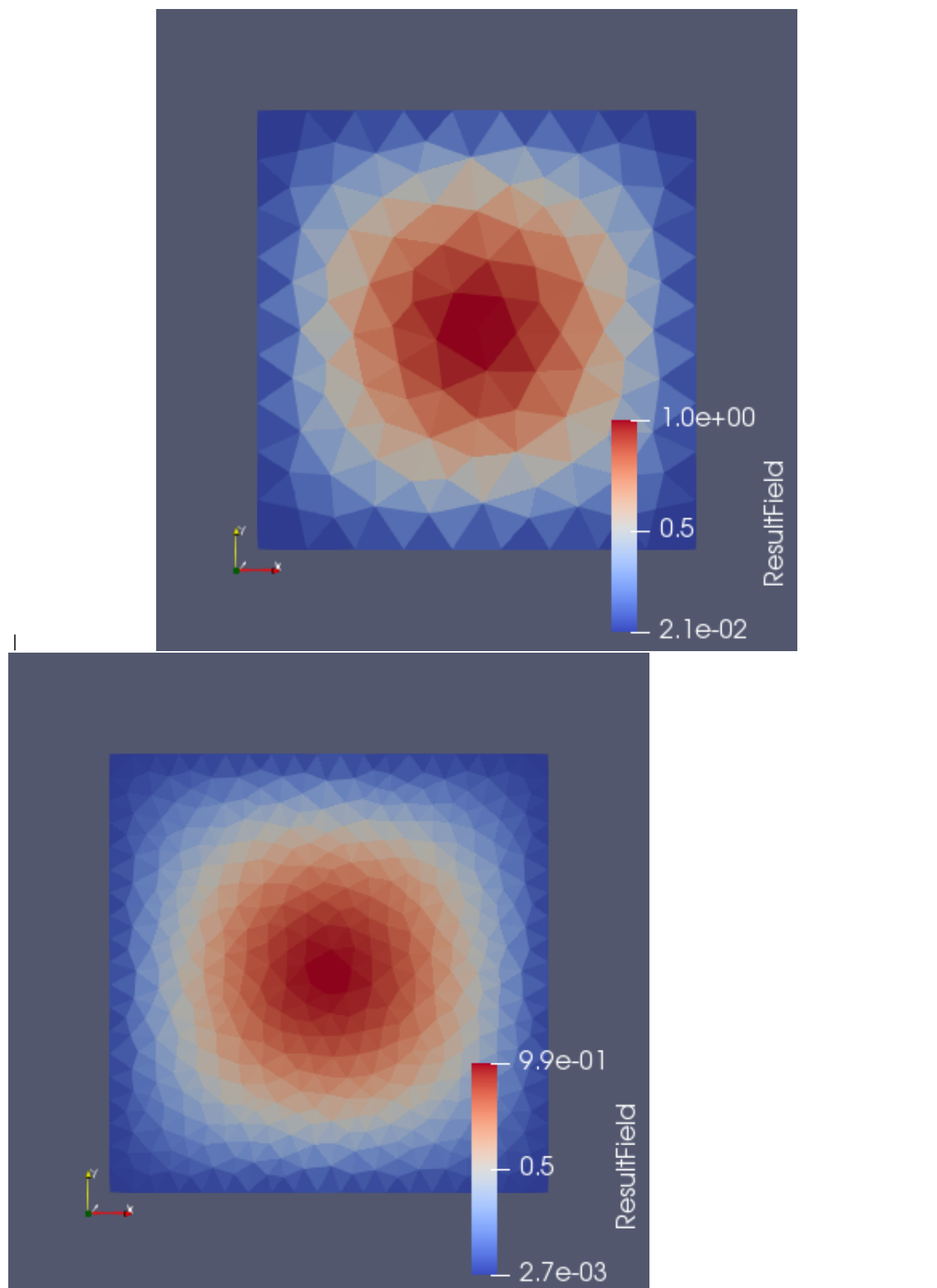


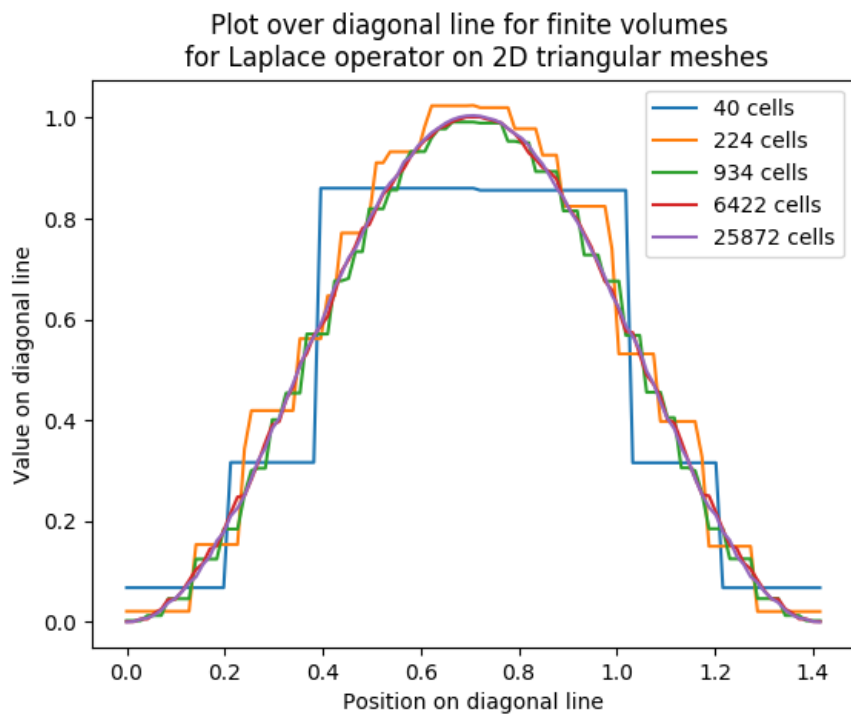
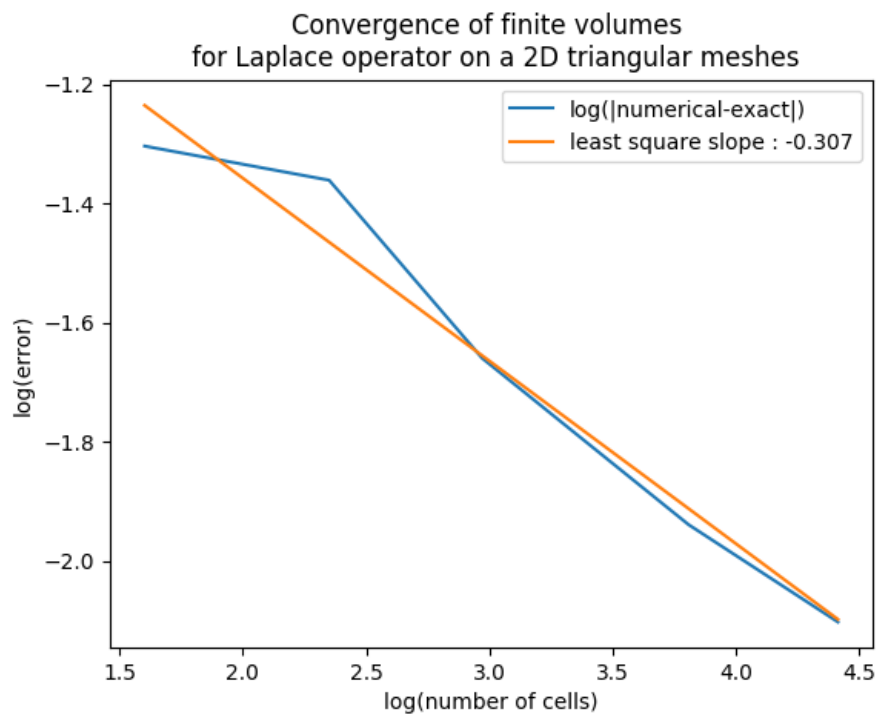
|

|

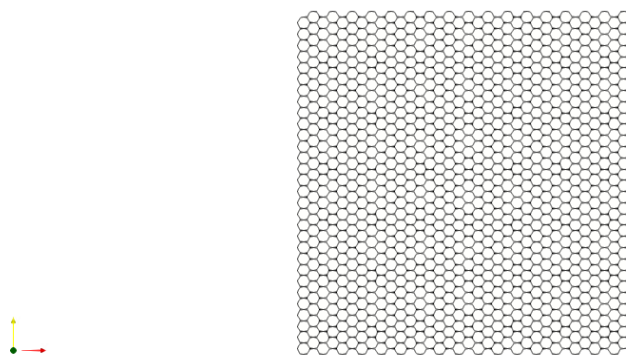
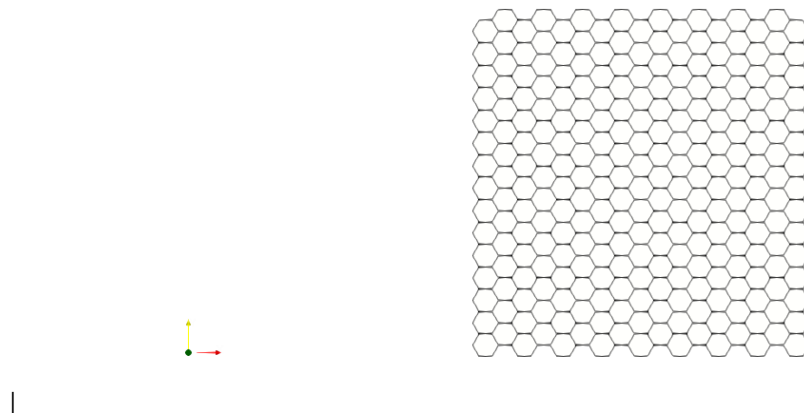
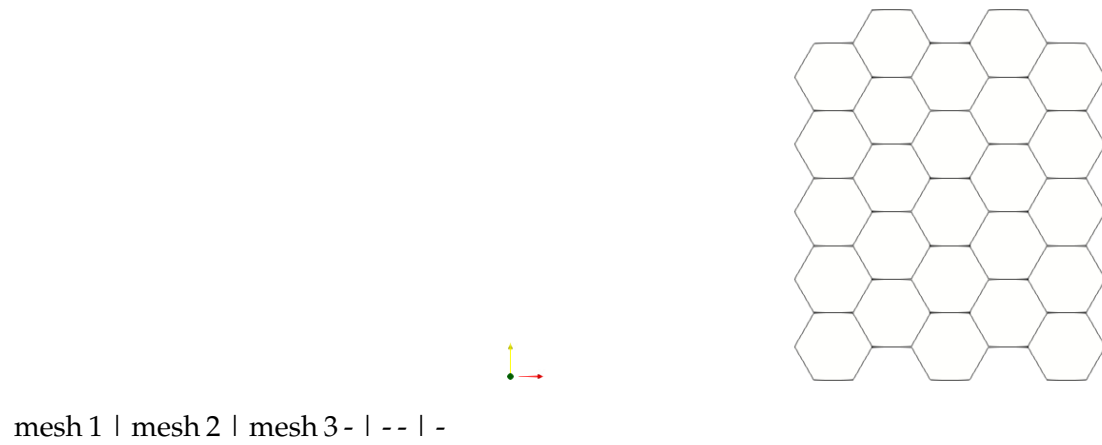


result 1 | result 2 | result 3 - | - - | -

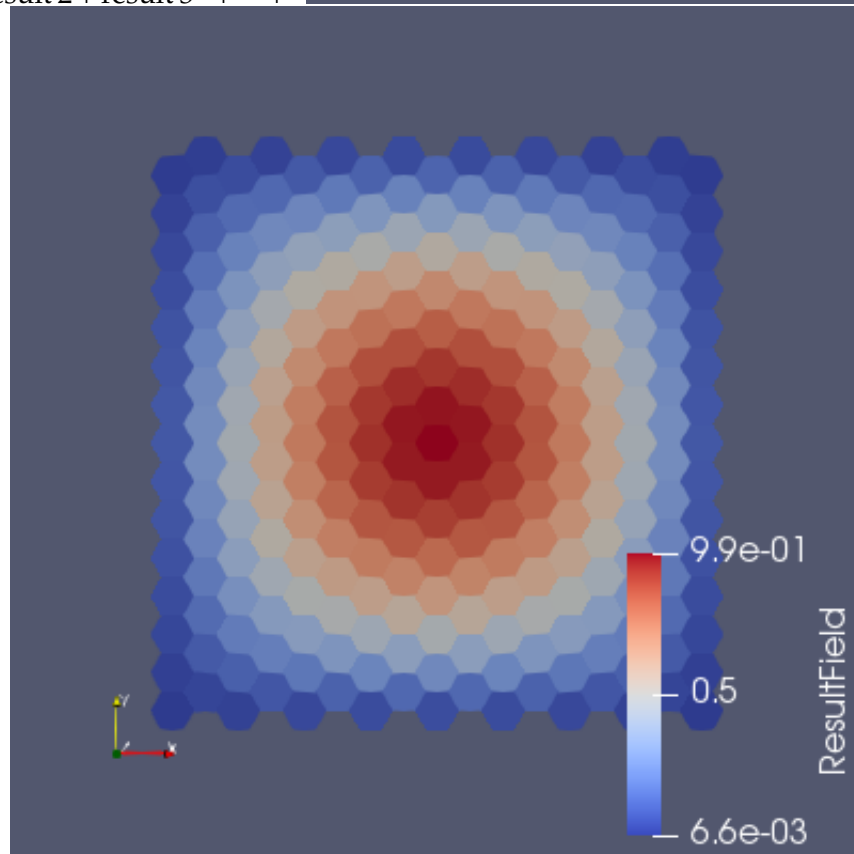
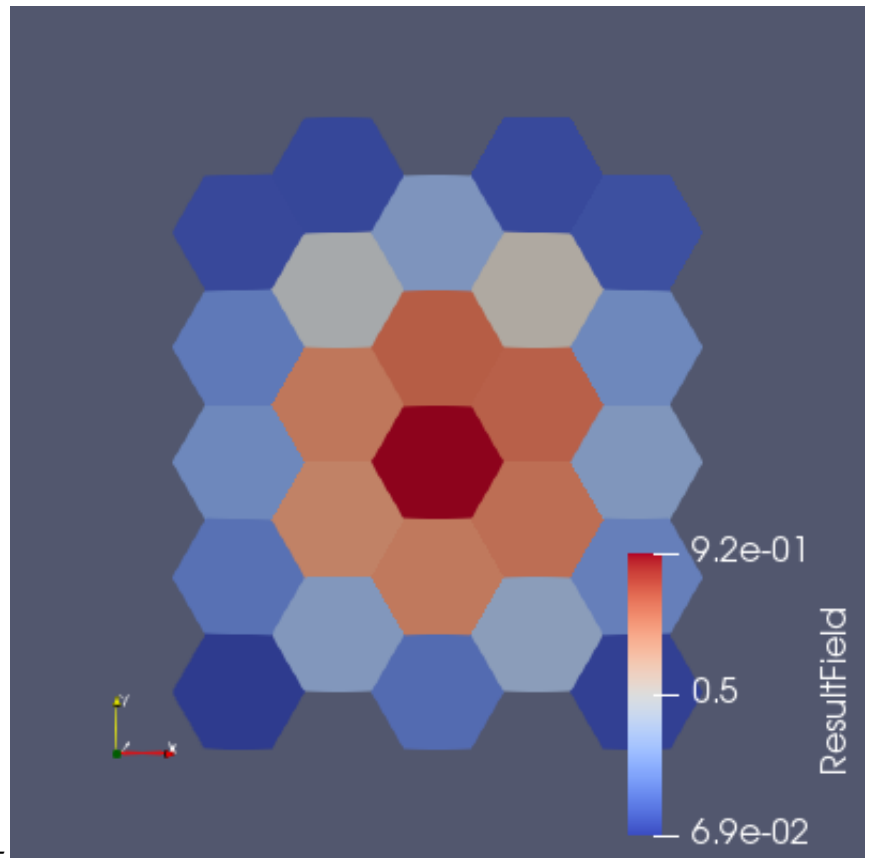


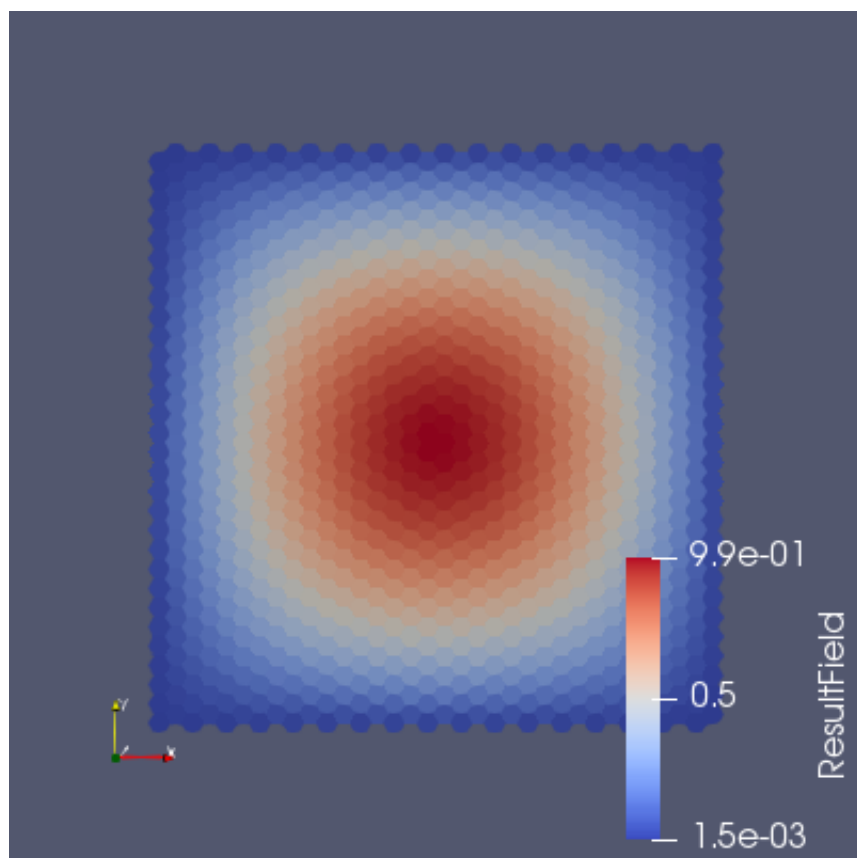
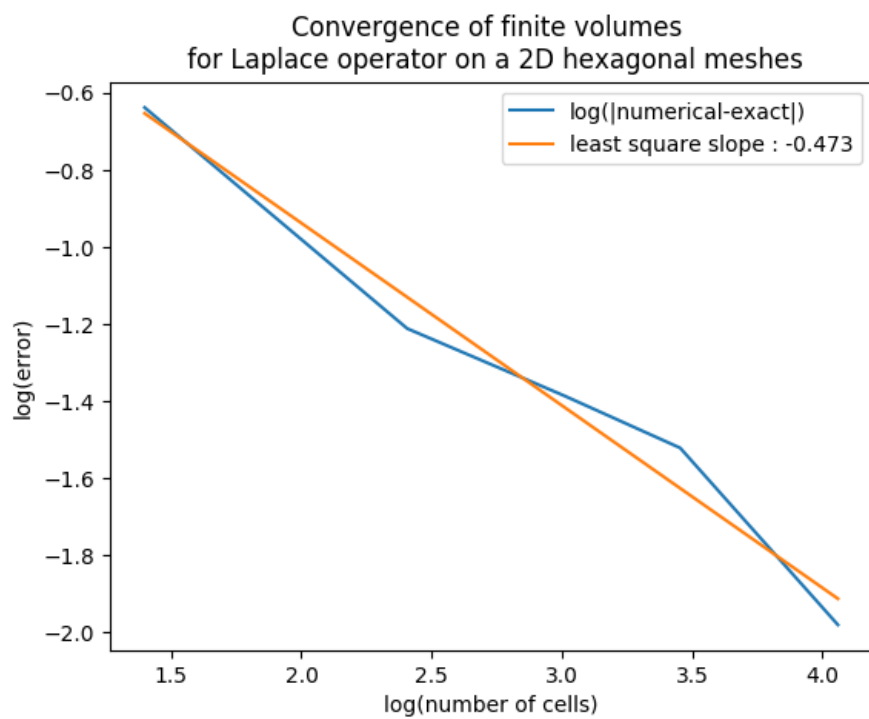


## 1.7 Hexagonal meshes

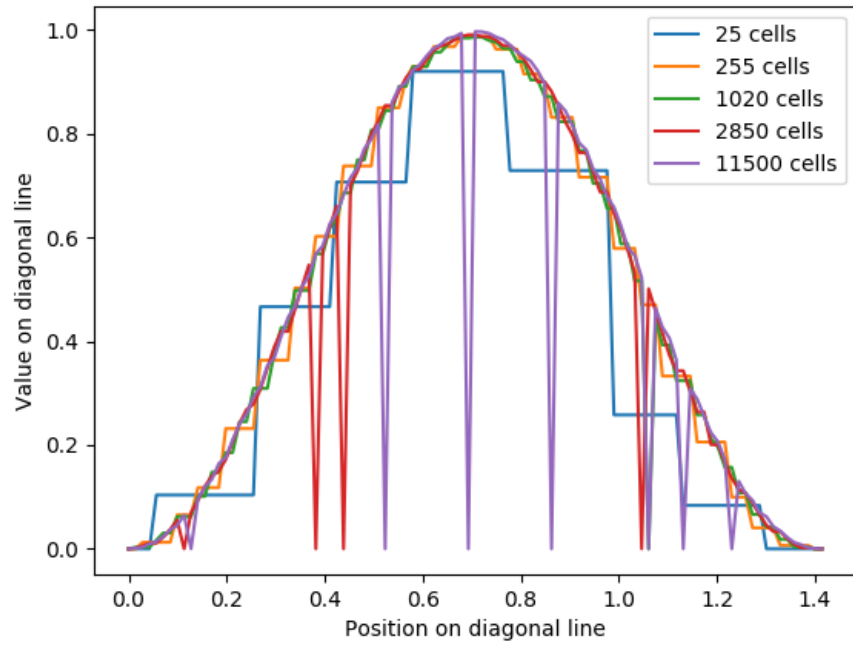




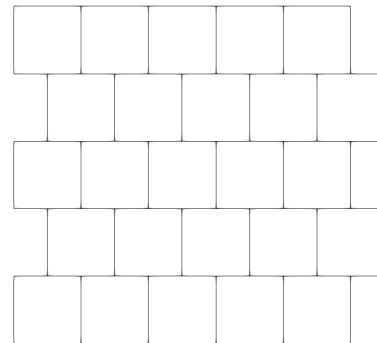




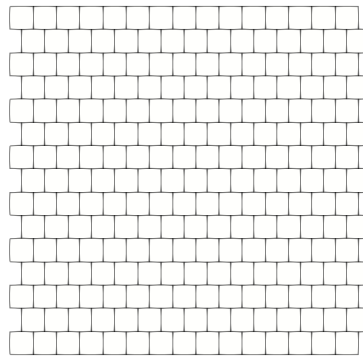
Plot over diagonal line for finite volumes  
for Laplace operator on 2D hexagonal meshes



## 1.8 Brick wall meshes

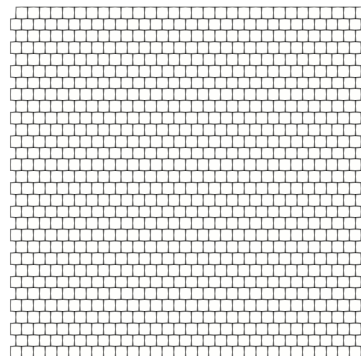


mesh 1 | mesh 2 | mesh 3 - | - - | -

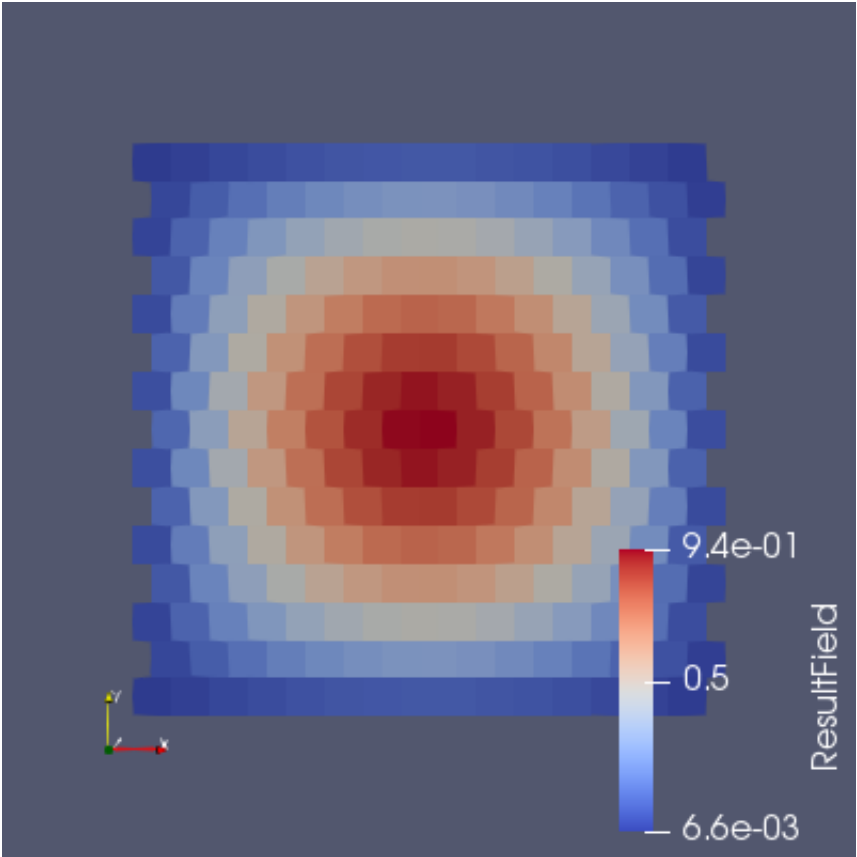
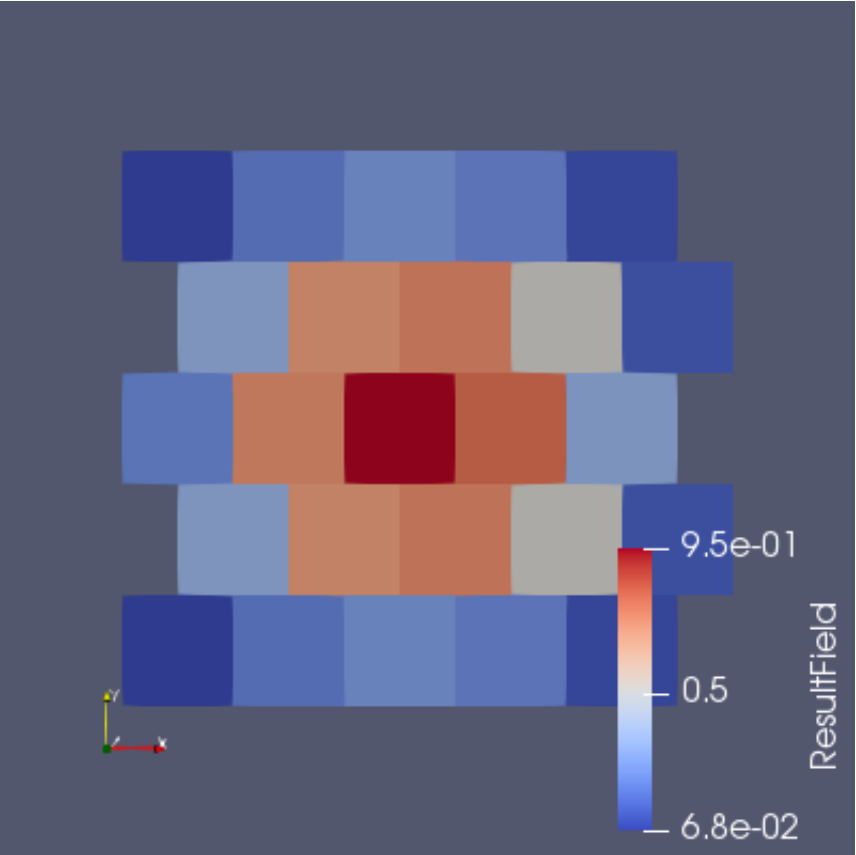


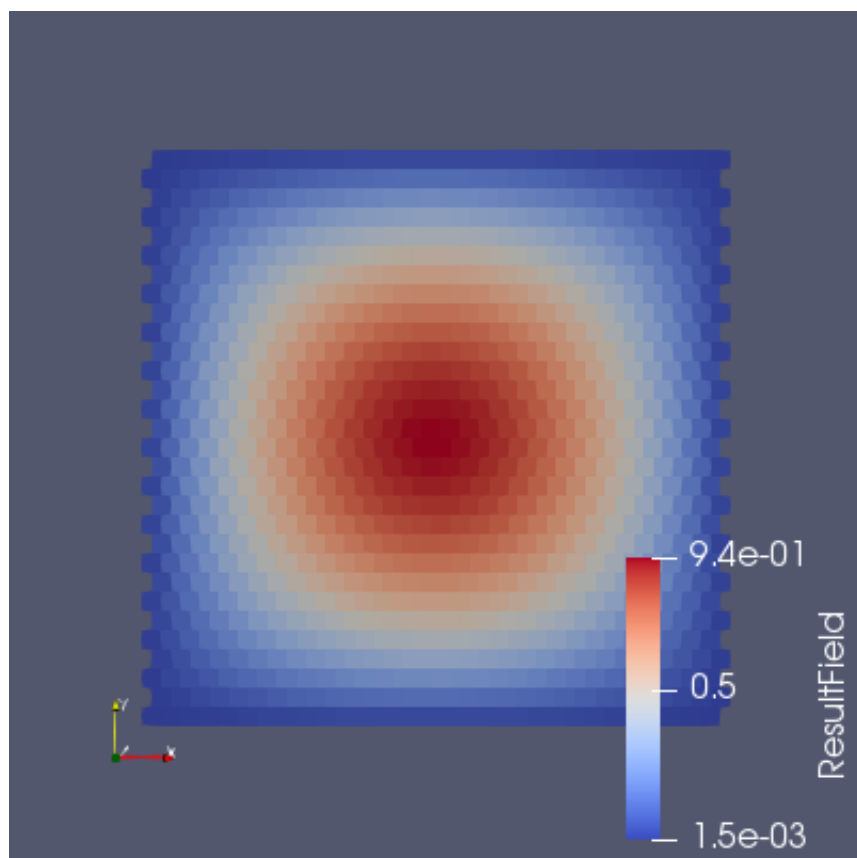
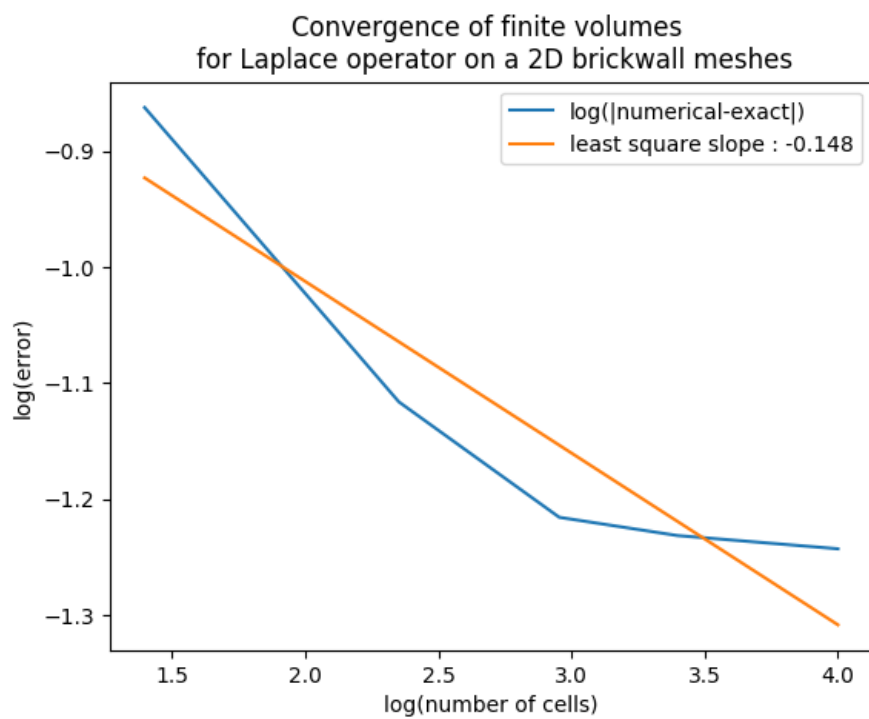
|

|

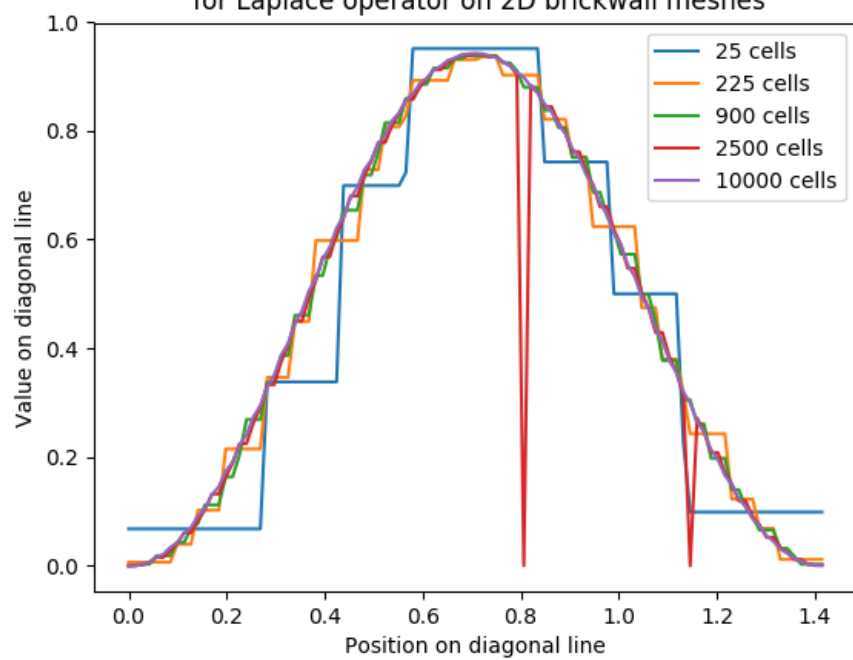


result 1 | result 2 | result 3 - | - - | -

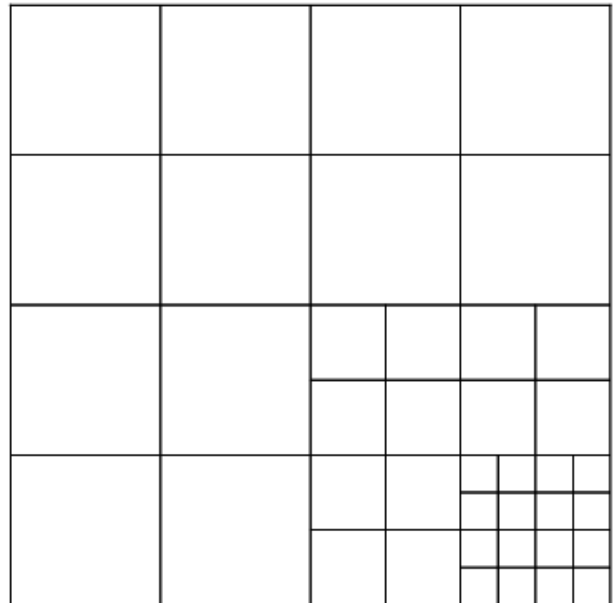




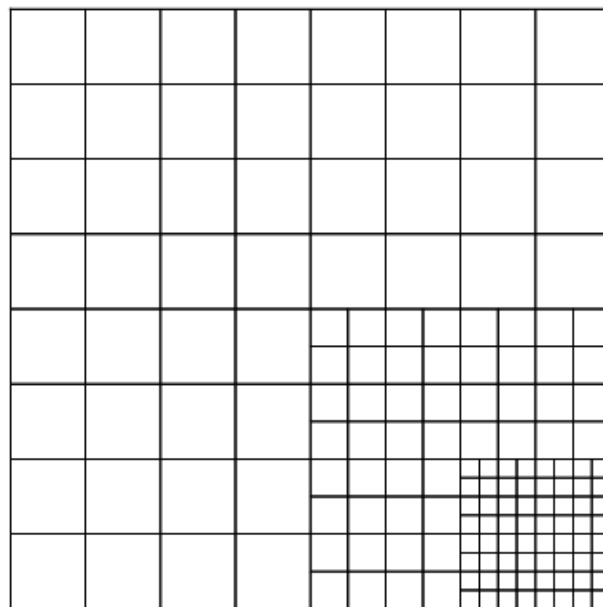
Plot over diagonal line for finite volumes  
for Laplace operator on 2D brickwall meshes



## 1.9 Locally refined meshes



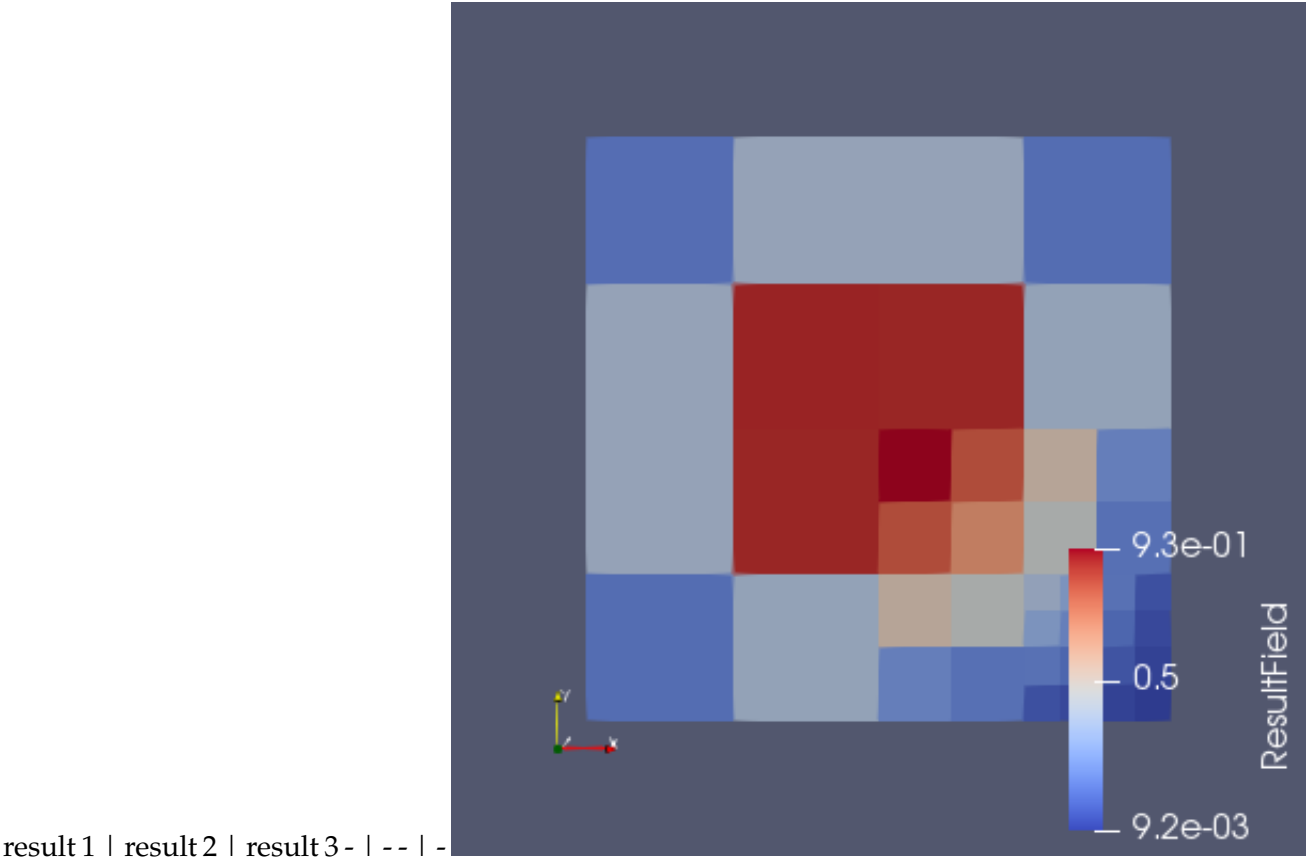
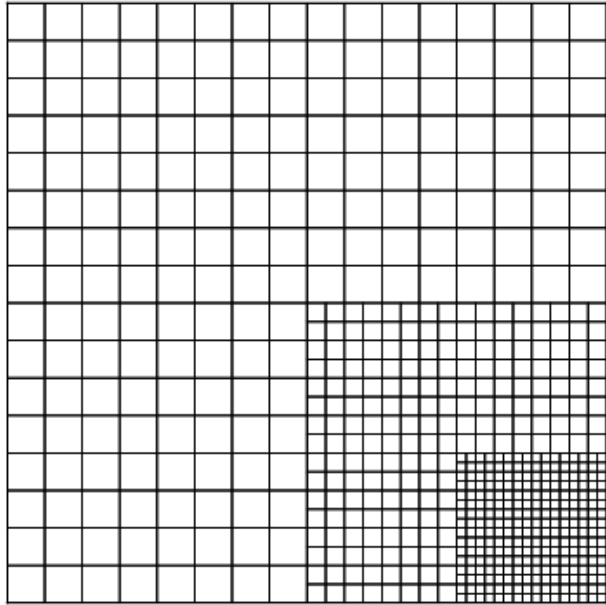
mesh 1 | mesh 2 | mesh 3 - | - - | -

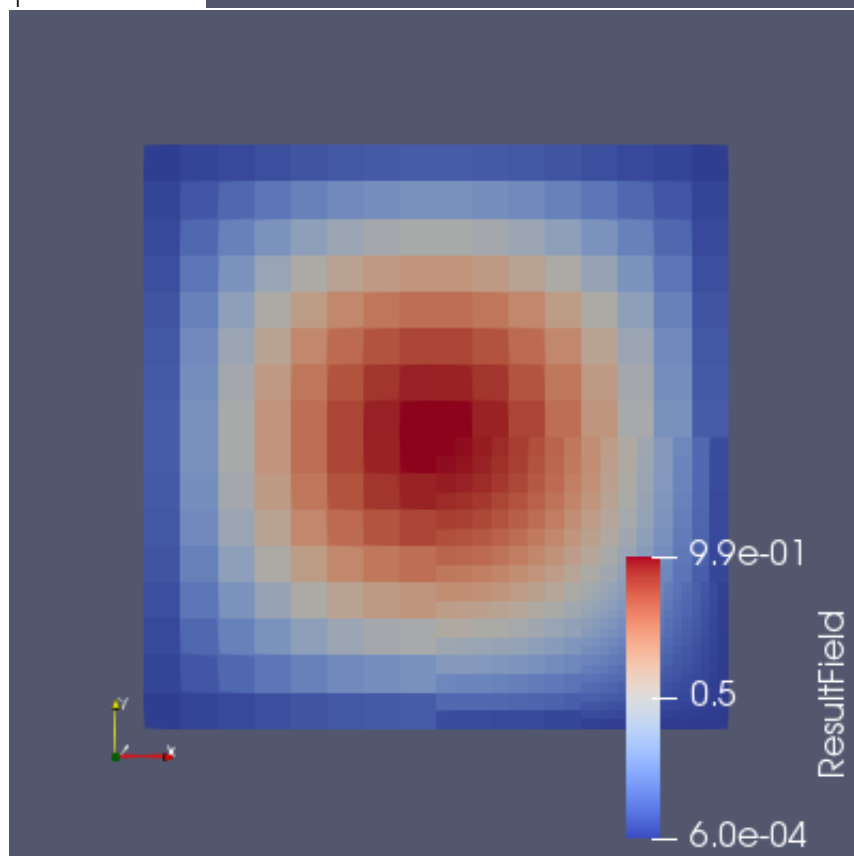
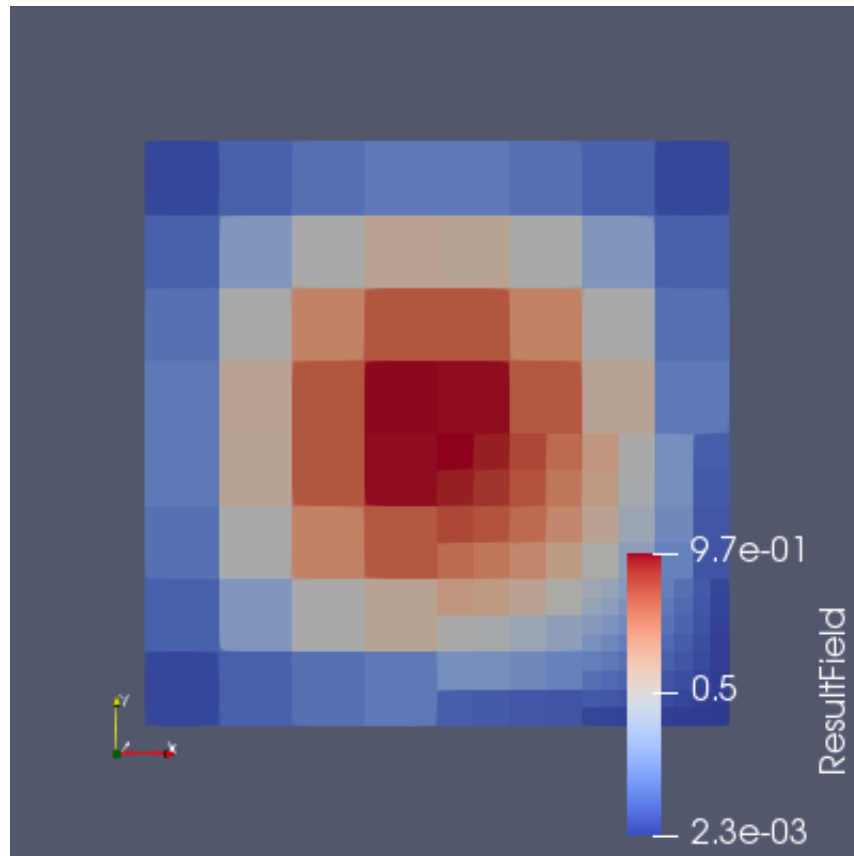


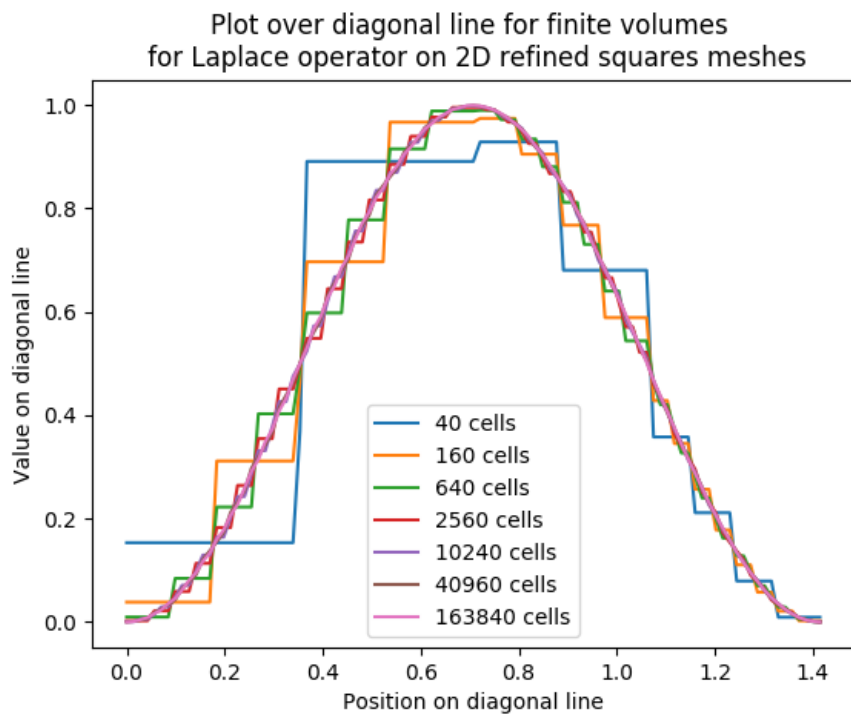
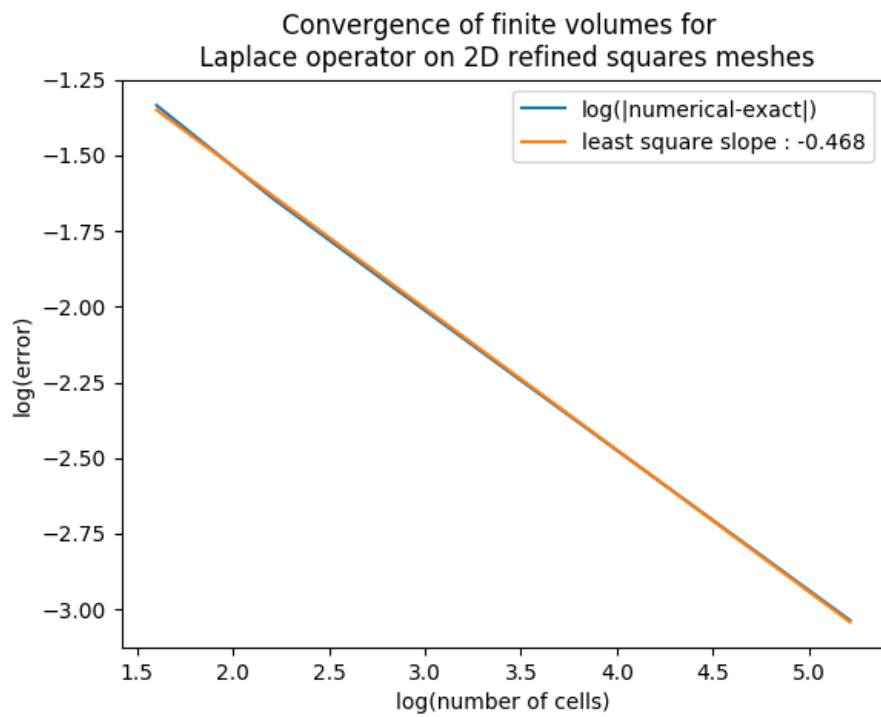
|

|

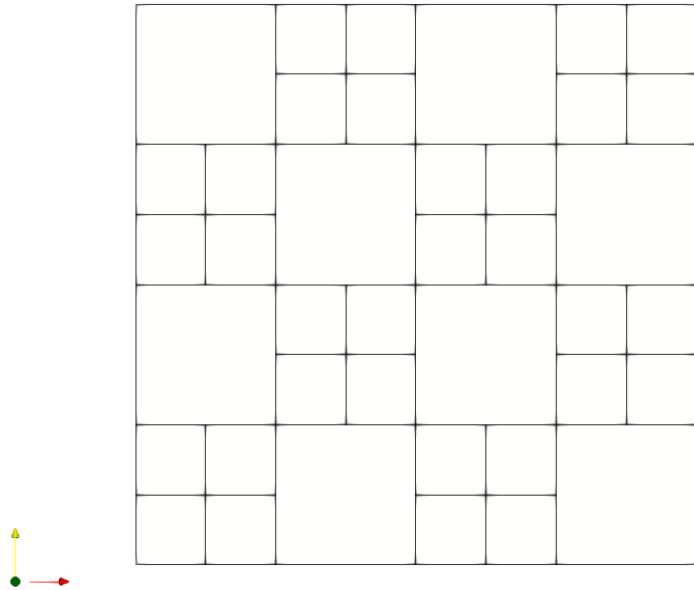




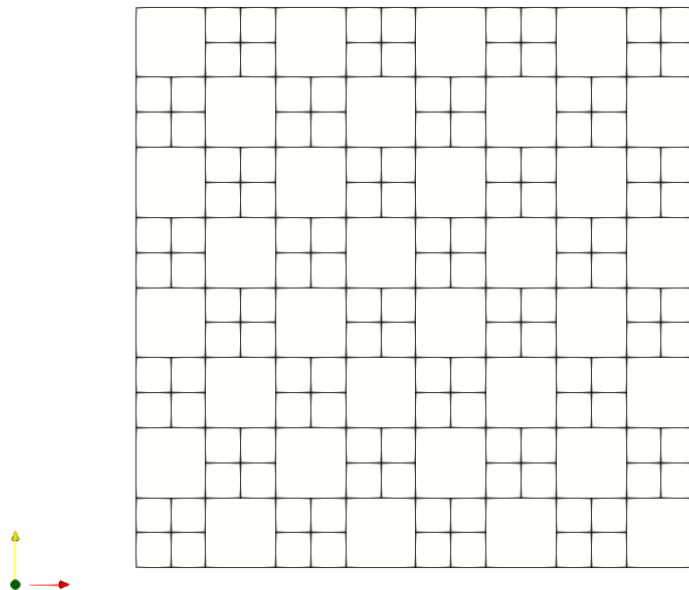




## 1.10 Checkerboard meshes

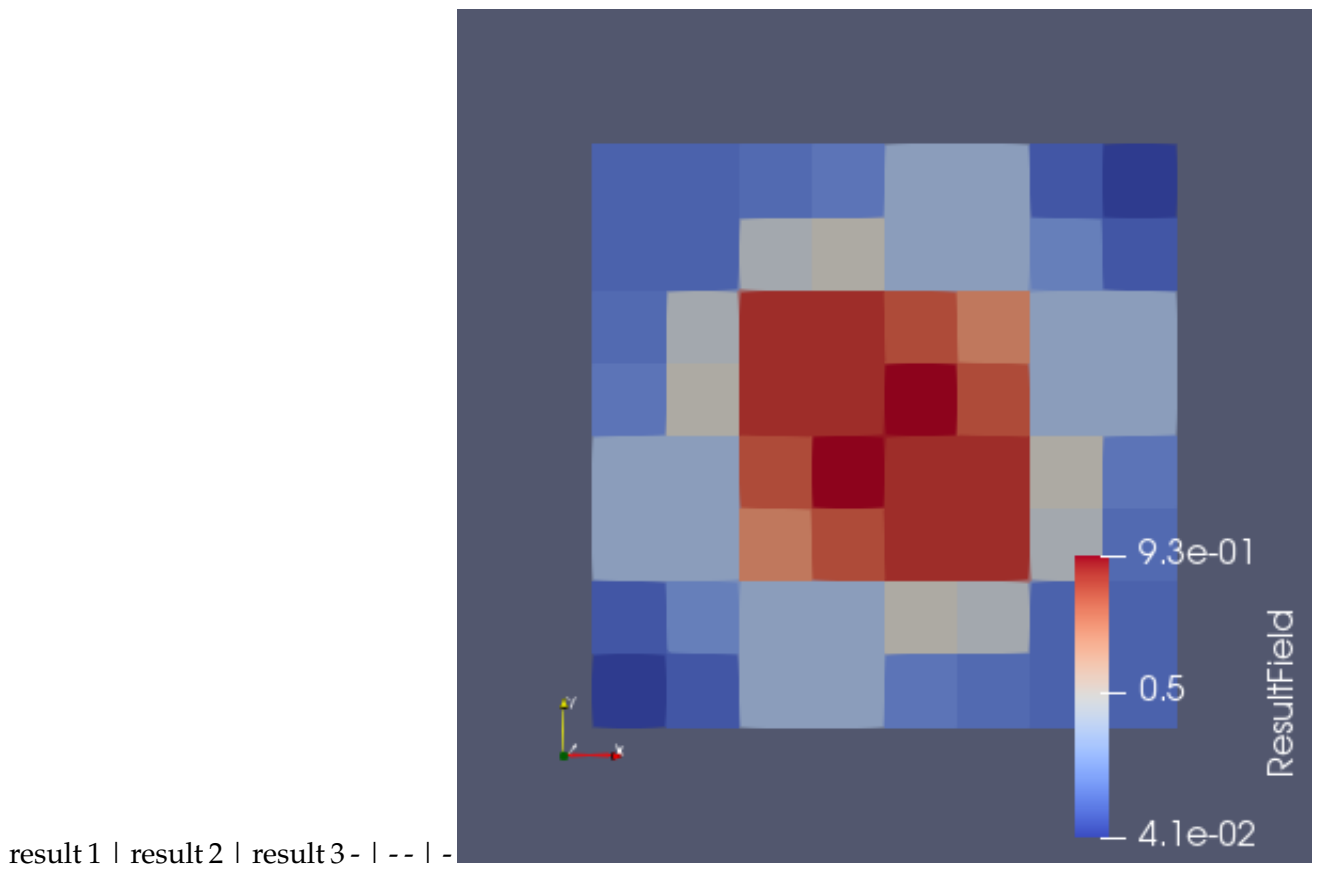
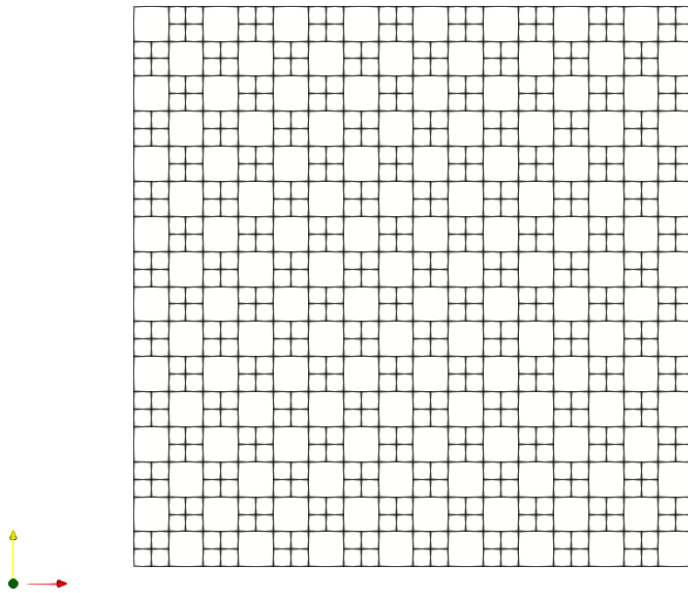


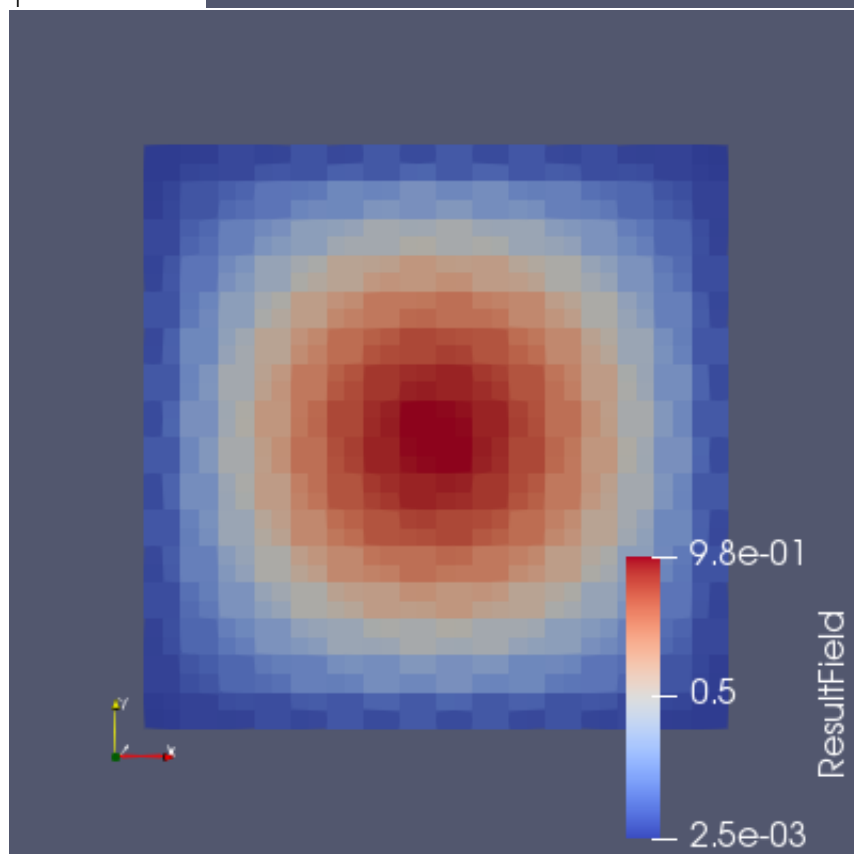
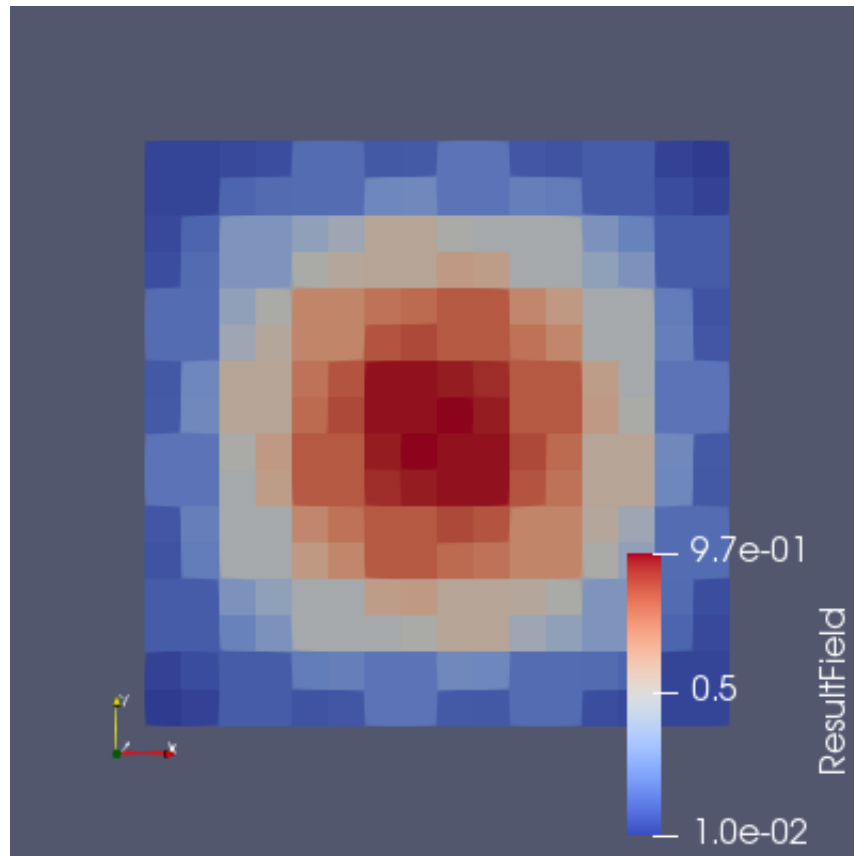
mesh 1 | mesh 2 | mesh 3 - | - - | -



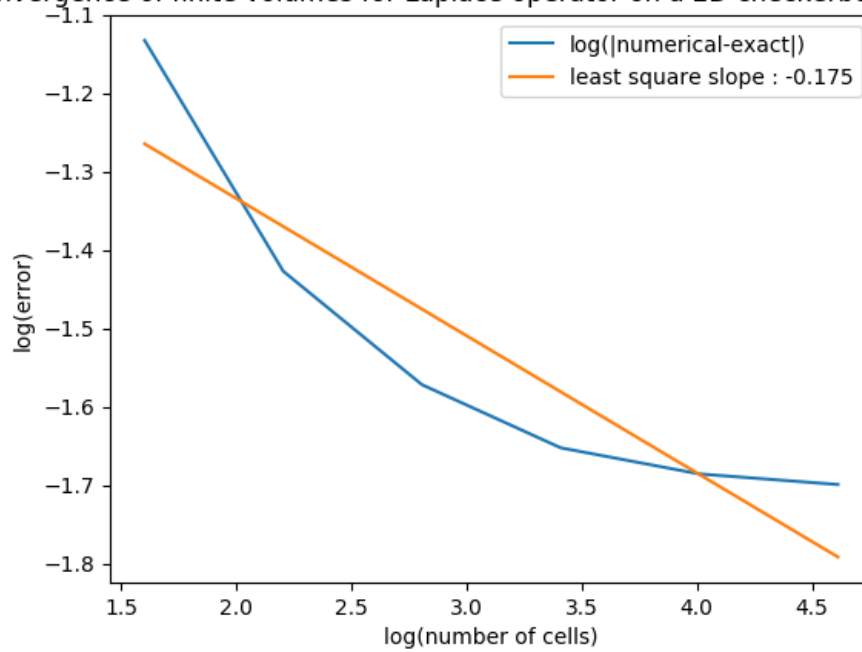
|

|





Convergence of finite volumes for Laplace operator on a 2D checkerboard mesh



Plot over diagonal line for finite volumes for Laplace operator on a 2D checkerboard mesh

