# Convergence_Poisson_FV5_SQUARE

September 26, 2019

```
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

$$\left\{\begin{array}{c} -\Delta u = f \text{ on } \Omega \\ u = 0 \text{ on } \partial\Omega \end{array}\right.$$

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

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

and the associated diffusion flux is

$$F(u) = D\nabla u,$$

where $D$ is the diffusion matrix.

We investigate the particular case where $D$ is the identity matrix.

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

### 1.2 Some bibliographical remarks about the two points finite volume scheme

- Order 2 convergence on orthogonal meshes : neighbouring cells $C_i$ and $C_j$ must be separated by a face (or edge in 2D) $f_{ij}$ that is perpendicular to the straight line connecting the center of masses $x_i$ of $C_i$ and $x_j$ of $C_j$
  *R. Eymard, T. Gallouët, R. Herbin, Finite Volume Methods, Handbook for Numerical Analysis, Ph. Ciarlet, J.L. Lions eds, North Holland, 2000, 715-1022.

- Order 1 convergence on not too deformed triangular meshes : the triangles edges must be in $O(h)$ and the triangle areas must be in $O(h^2)$ (angles must not shrink to $0°$ nor $180°$)
  *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.*

- Order 2 convergence on triangular meshes, provided

  - the center of the circumscribed circle is used instead of the center of mass in each cell for the evaluation of the source term and analytical solution
  - the Delaunay conditions are satisfied (no neighboring cell is included in the circumscribed circle of an arbitrary cell)

- Non convergence on flat degenerating triangular meshes
  *K. Domelevo, P. Omnes, A finite volume method for the Laplace equation on almost arbitrary 2D grids, Mathematical Modelling and Numerical Analysis, 2005*

- 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*

- Order 2 on triangular meshes provided 1) Delaunay type conditions are satisfied and 2) $f \in H^1$ 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, IMA Journal of Numerical Analysis, 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

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

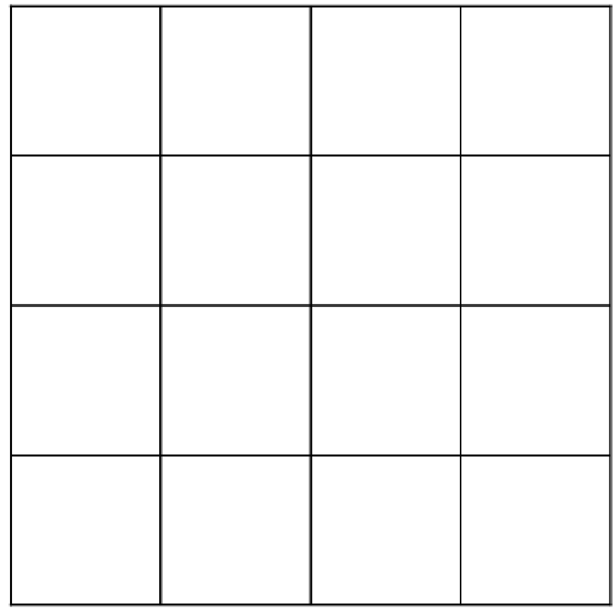
for two cells $i$ and $j$ inside the domain,

and

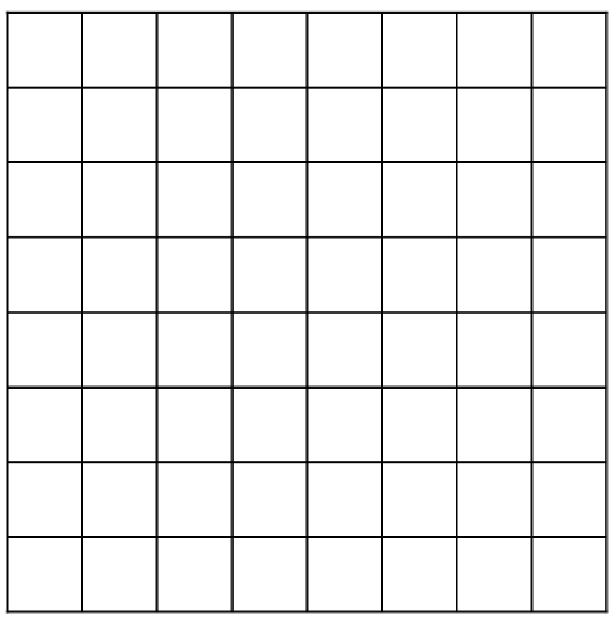
$$F_{boundary} = \frac{u(x_f) - u_i}{d_{if}},$$

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

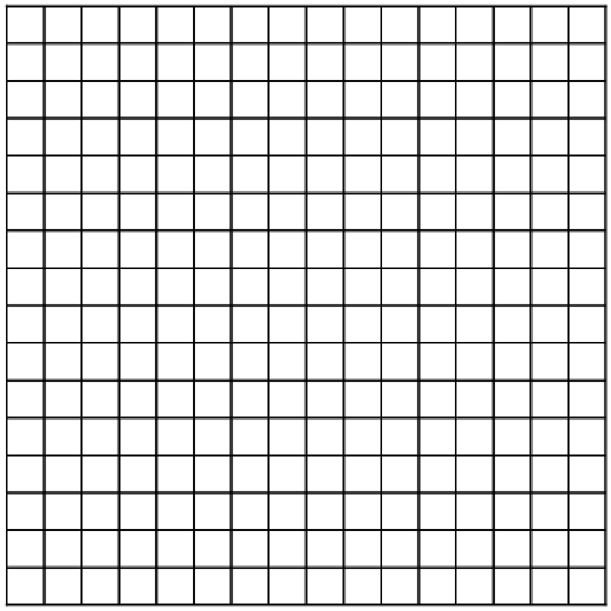
## 1.4 Regular grid



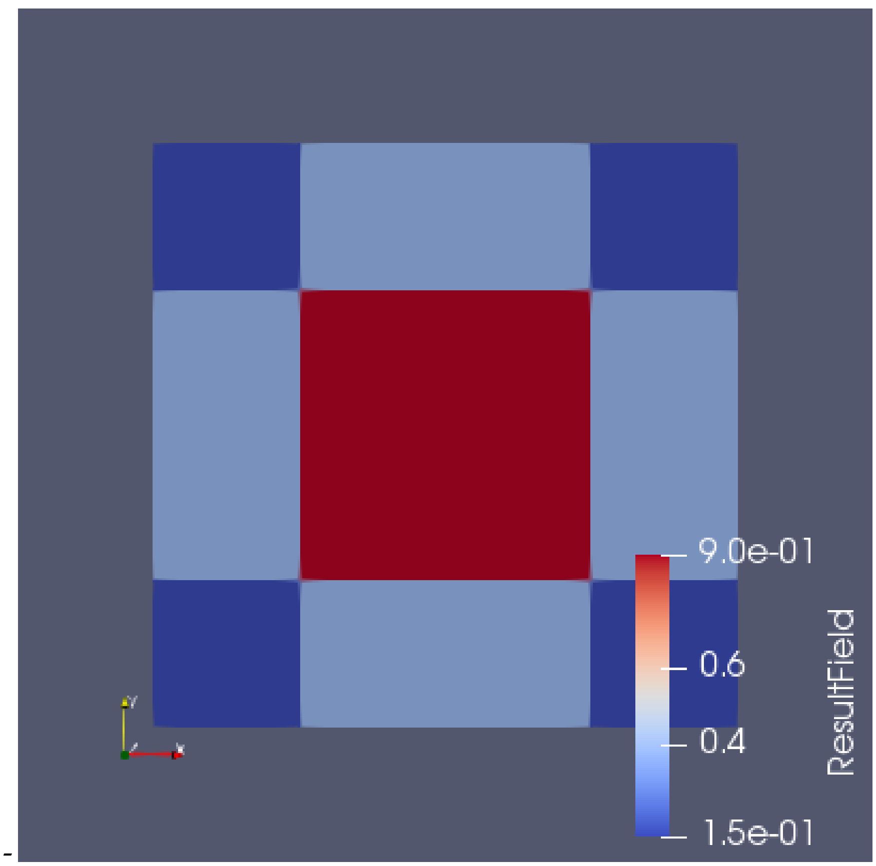
mesh 1 | mesh 2 | mesh 3 - | - - | -

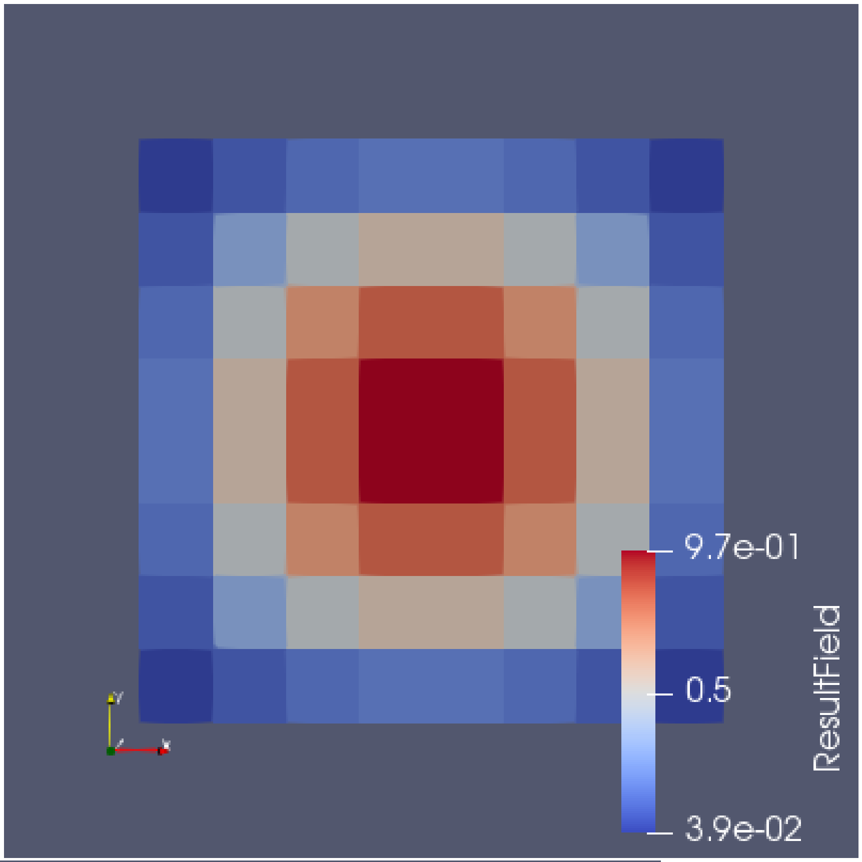


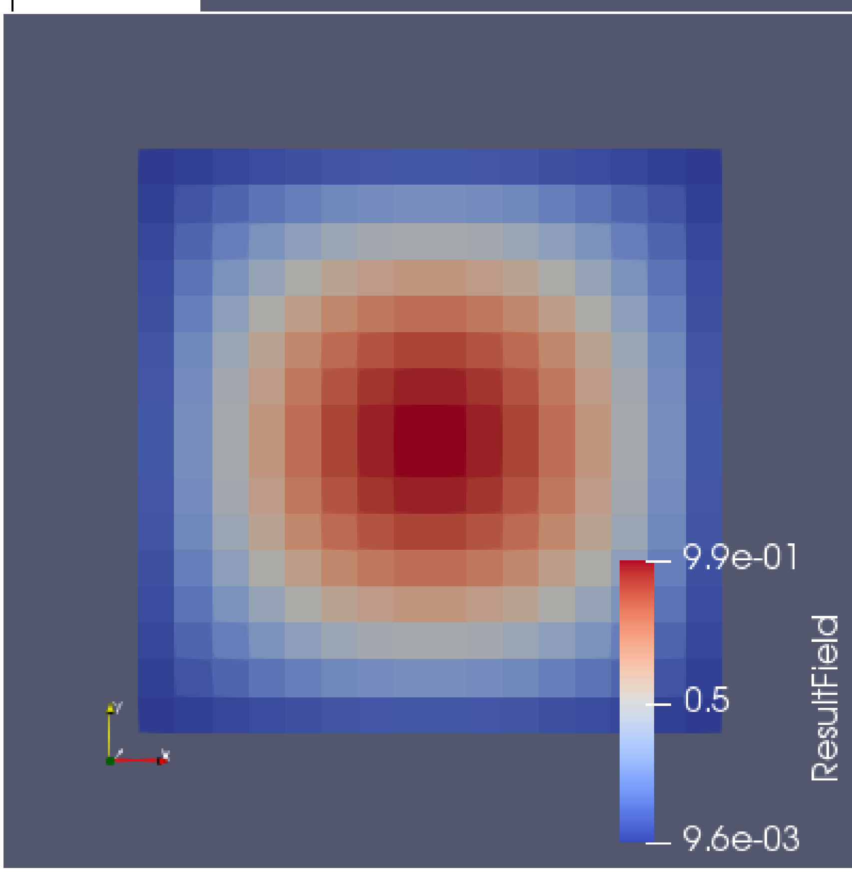
| |

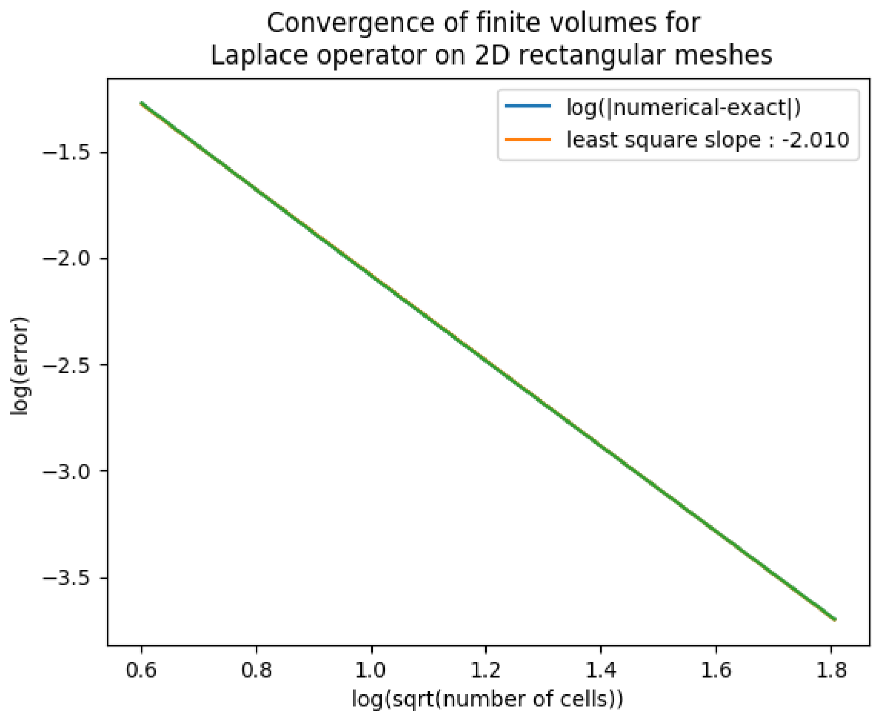


result 1 | result 2 | result 3 - | - - | -

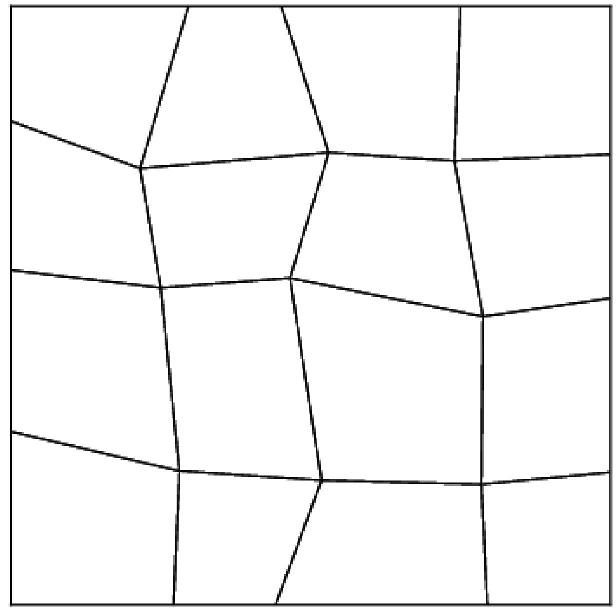



9.7e-01
0.5
3.9e-02
ResultField

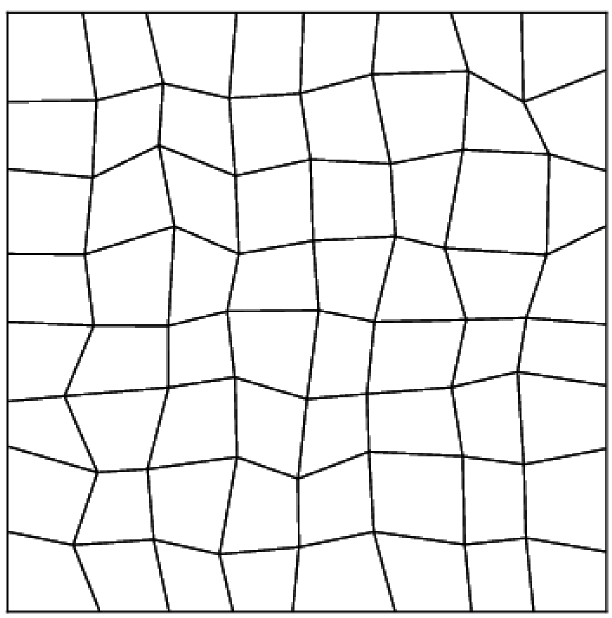

9.9e-01
0.5
9.6e-03
ResultField


Convergence of finite volumes for
Laplace operator on 2D rectangular meshes
log(|numerical-exact|)
least square slope : -2.010
log(error)
−1.5
−2.0
−2.5
−3.0
−3.5
0.6
0.8
1.0
1.2
1.4
1.6
1.8
log(sqrt(number of cells))

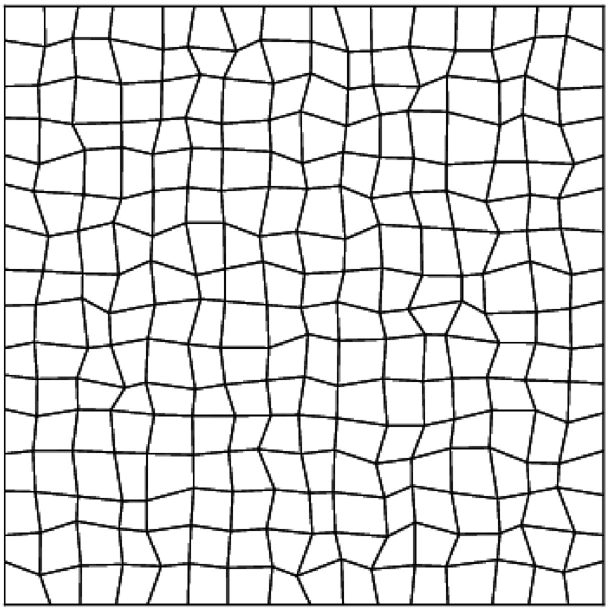
### 1.5 Deformed quadrangles



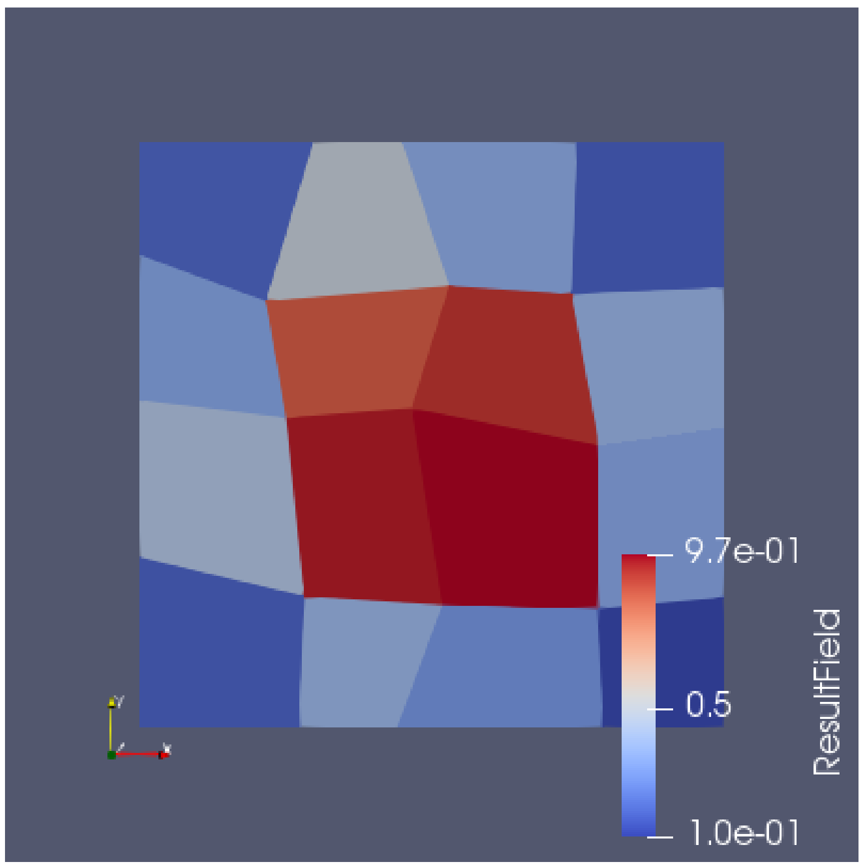
mesh 1 | mesh 2 | mesh 3 - | - - | -

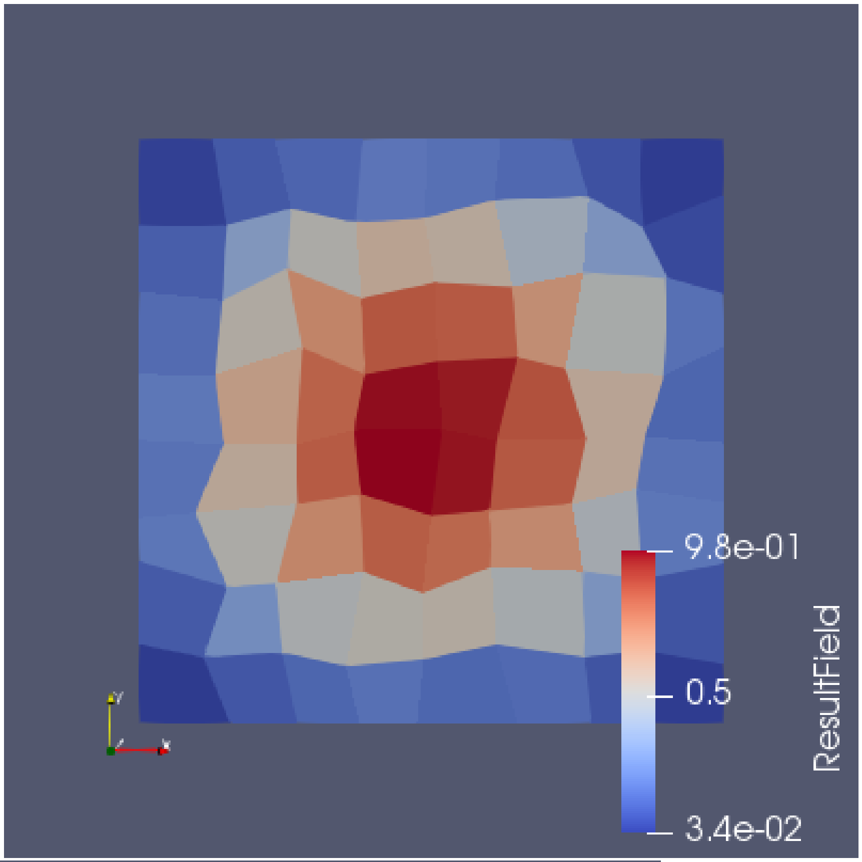


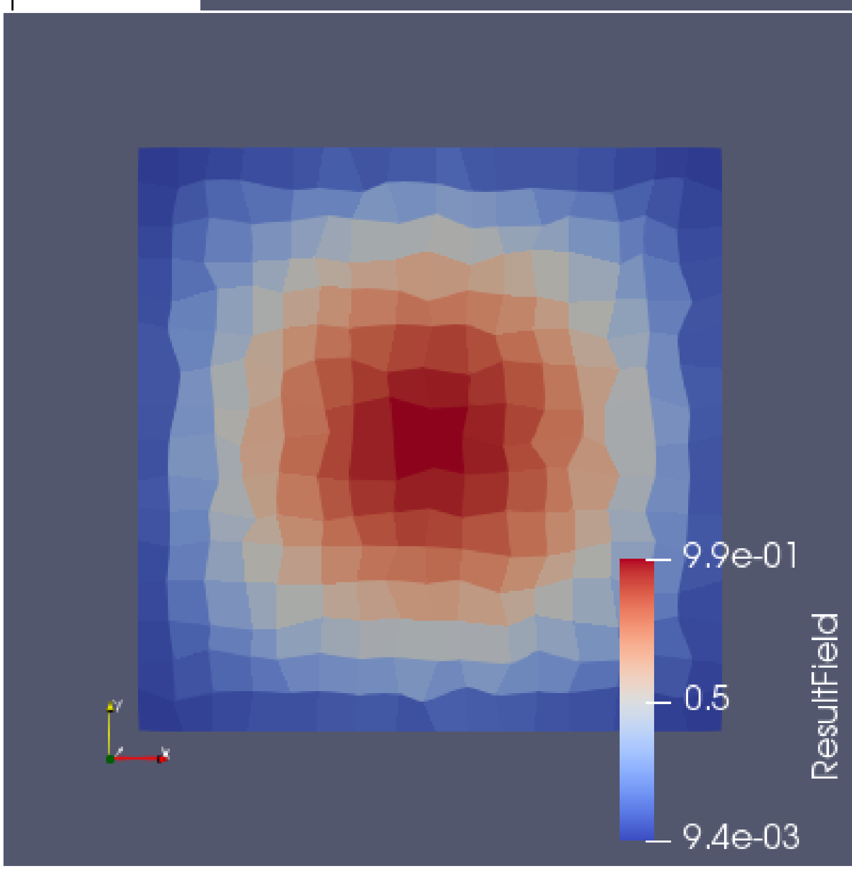
| |

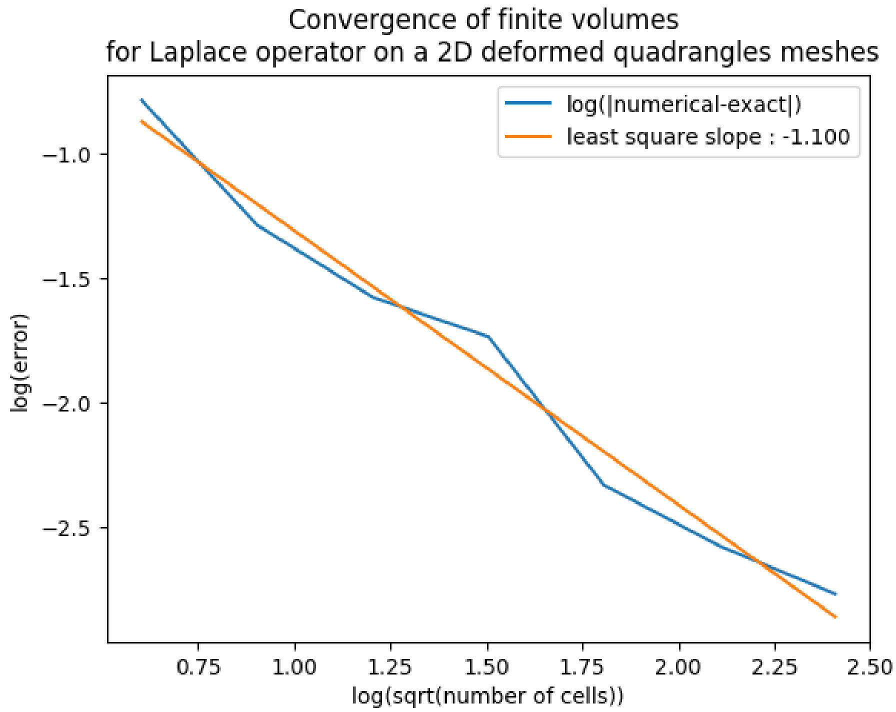


result 1 | result 2 | result 3 - | -- | -

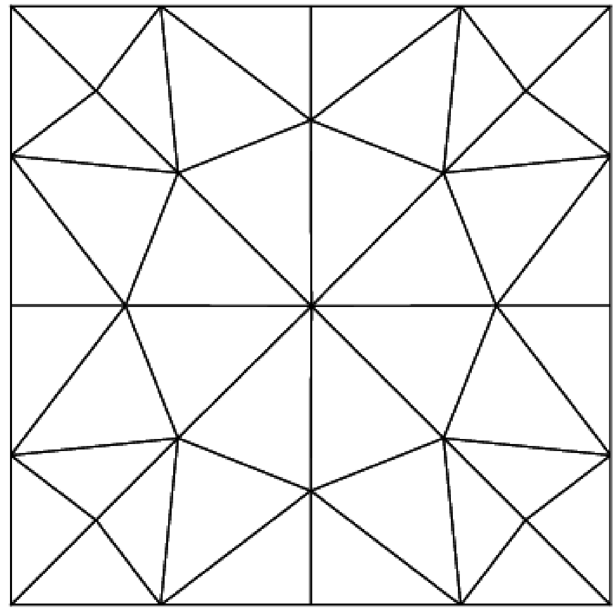



9.8e-01
0.5
3.4e-02
ResultField

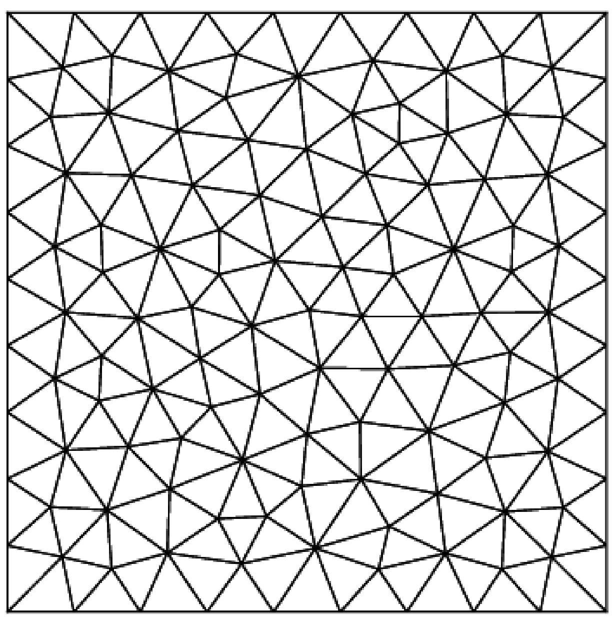

9.9e-01
0.5
9.4e-03
ResultField


Convergence of finite volumes
for Laplace operator on a 2D deformed quadrangles meshes
log(|numerical-exact|)
least square slope : -1.100
log(error)
−1.0
−1.5
−2.0
−2.5
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50
log(sqrt(number of cells))

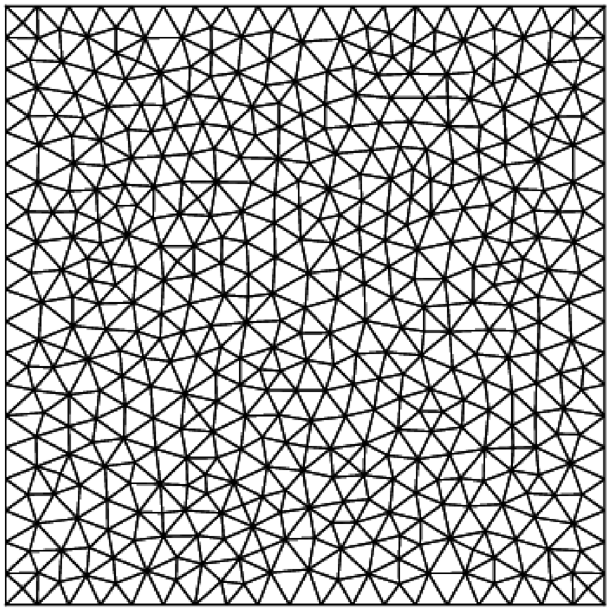
## 1.6 Delaunay triangular meshes

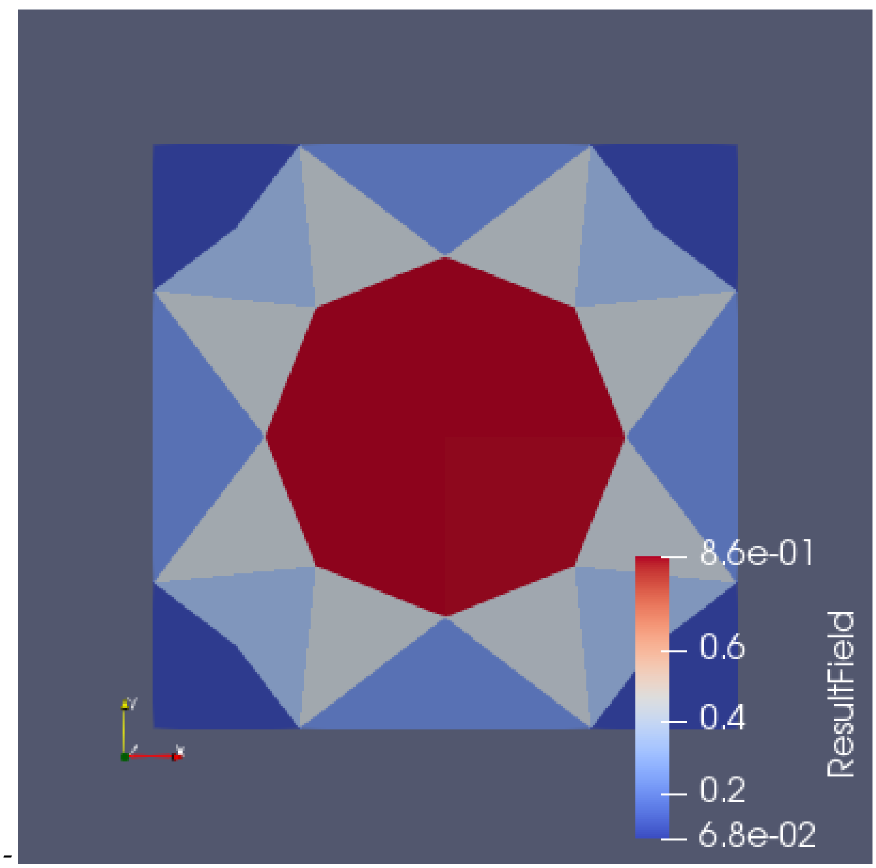


mesh 1 | mesh 2 | mesh 3 - | - - | -

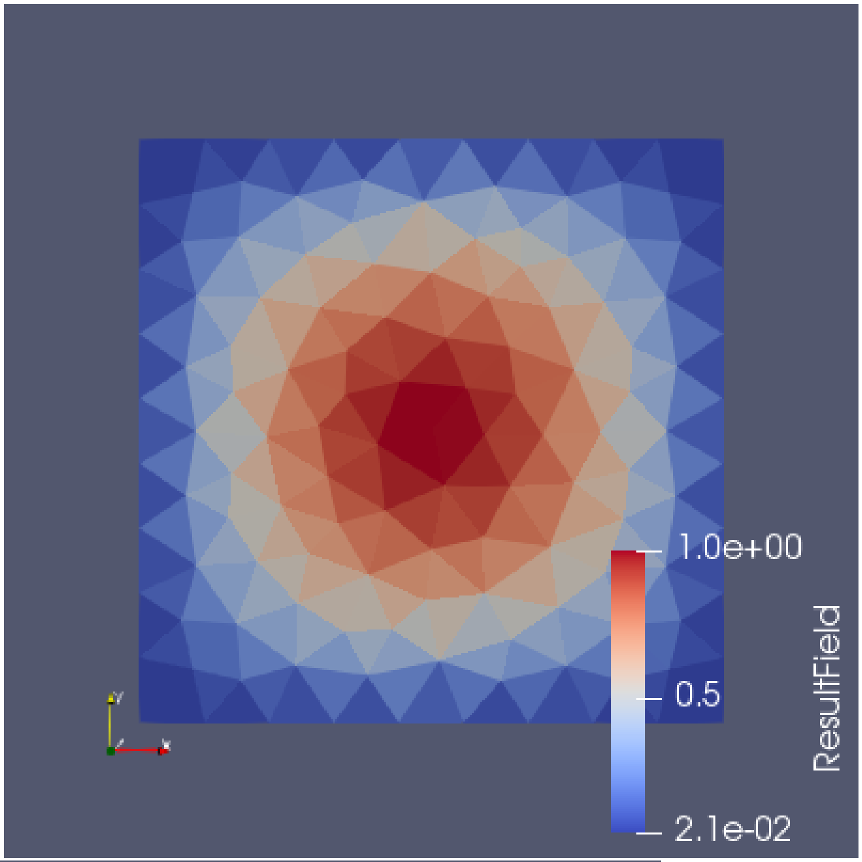


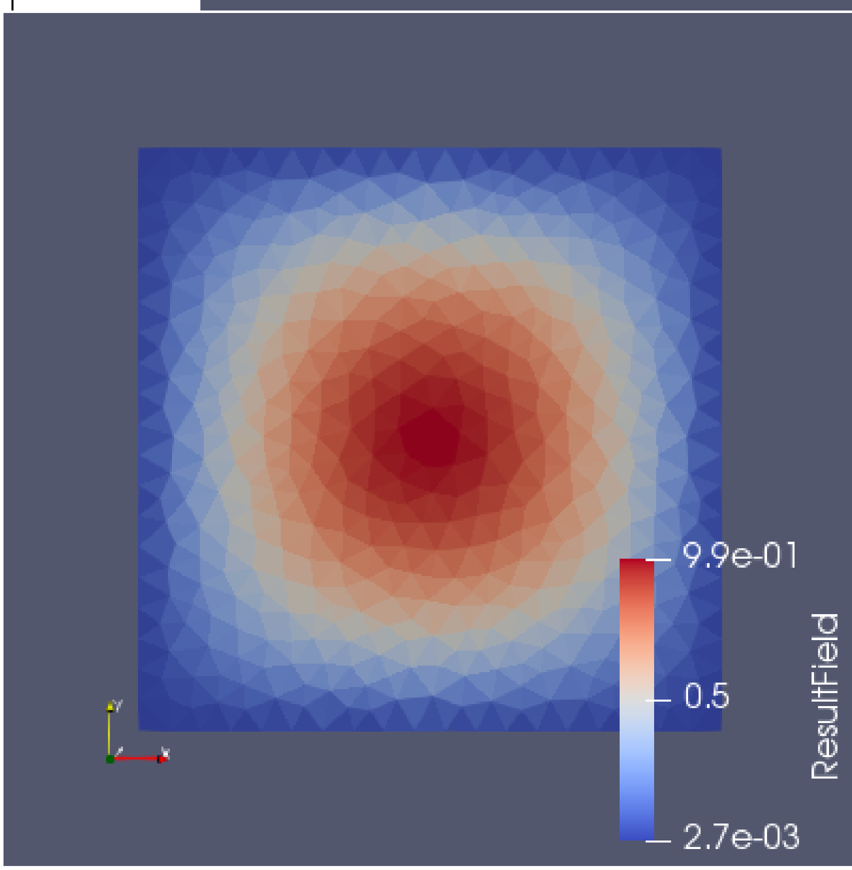
| |

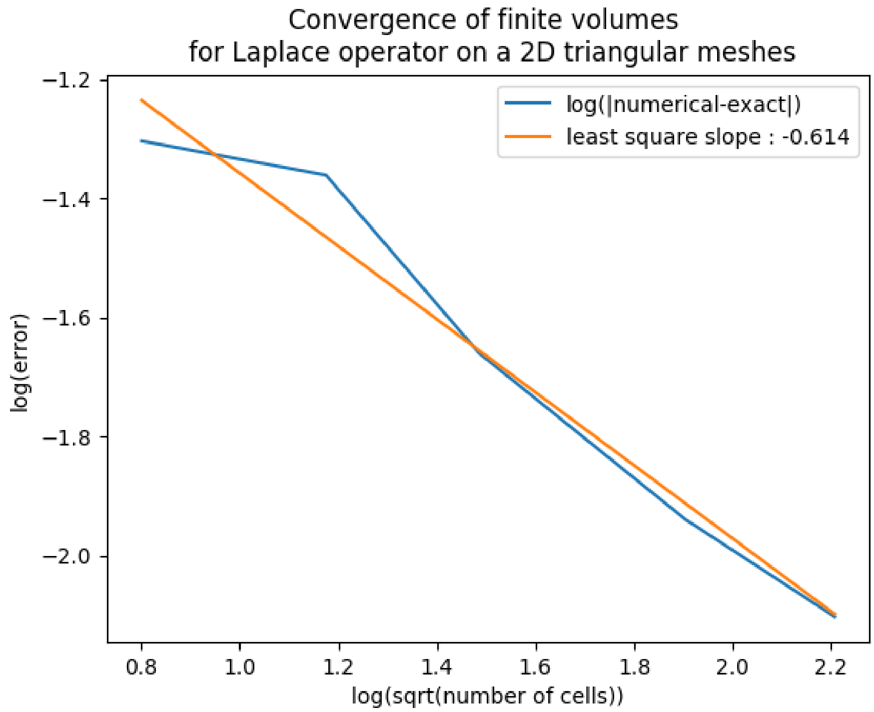




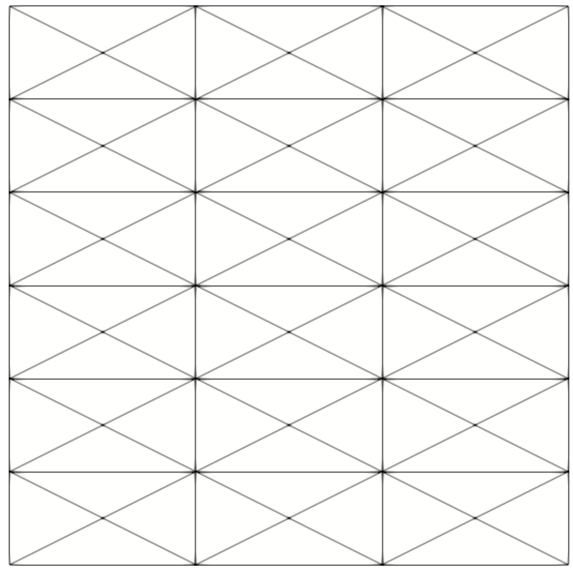

result 1 | result 2 | result 3 - | - - | -


1.0e+00
0.5
2.1e-02
ResultField

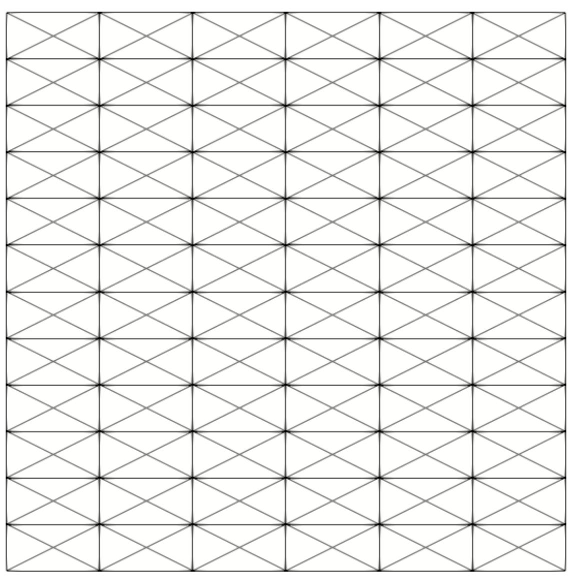

9.9e-01
0.5
2.7e-03
ResultField


Convergence of finite volumes
for Laplace operator on a 2D triangular meshes
log(|numerical-exact|)
least square slope : -0.614
−1.2
−1.4
−1.6
−1.8
−2.0
log(error)
0.8
1.0
1.2
1.4
1.6
1.8
2.0
2.2
log(sqrt(number of cells))

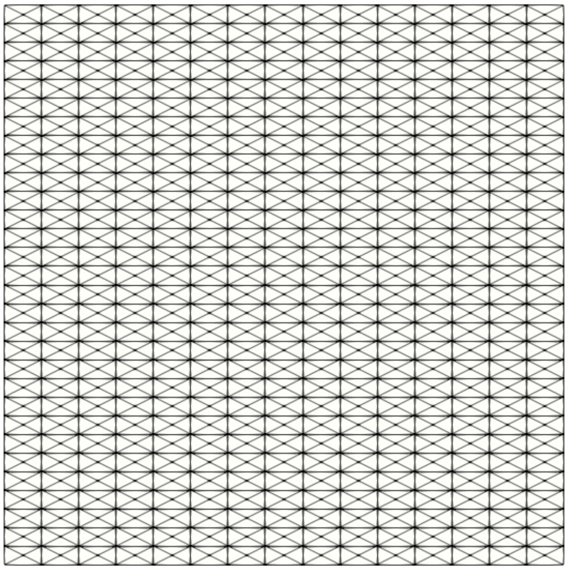
### 1.7 Cross triangle meshes (from a $(n, 2n)$ rectangular grid)



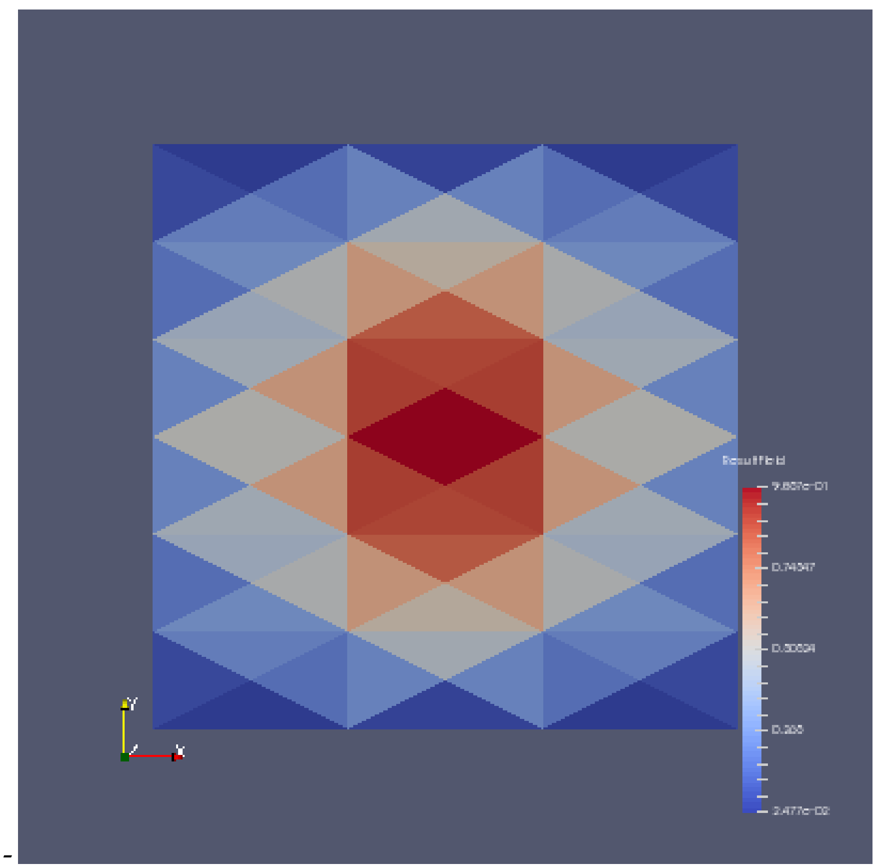
mesh 1 | mesh 2 | mesh 3 - | - - | -

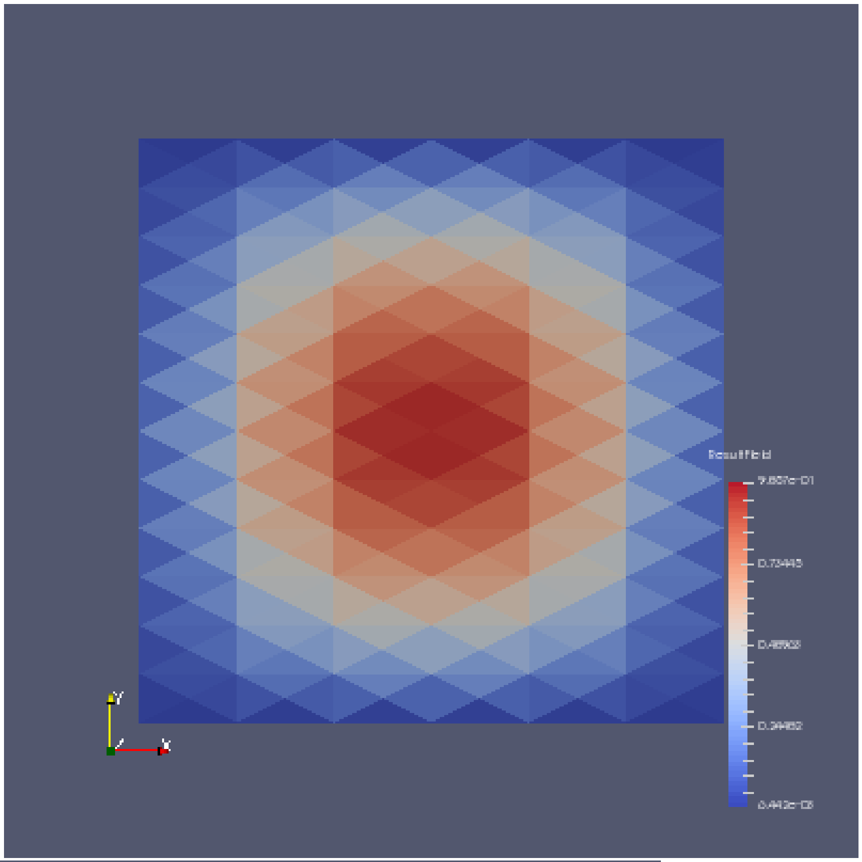


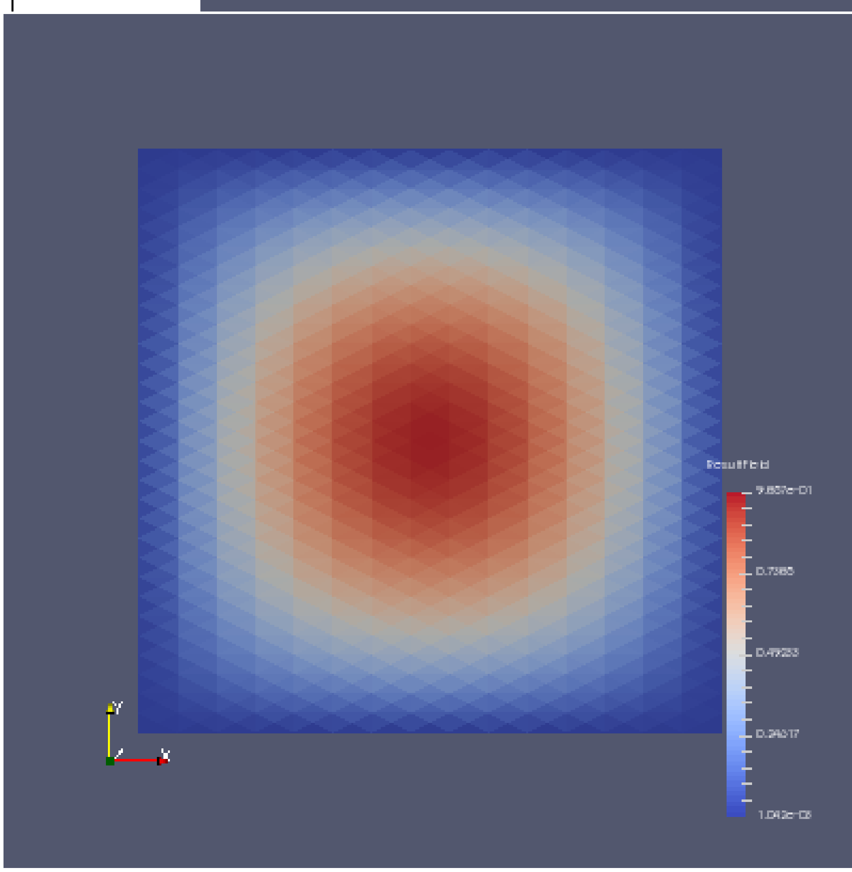
| |

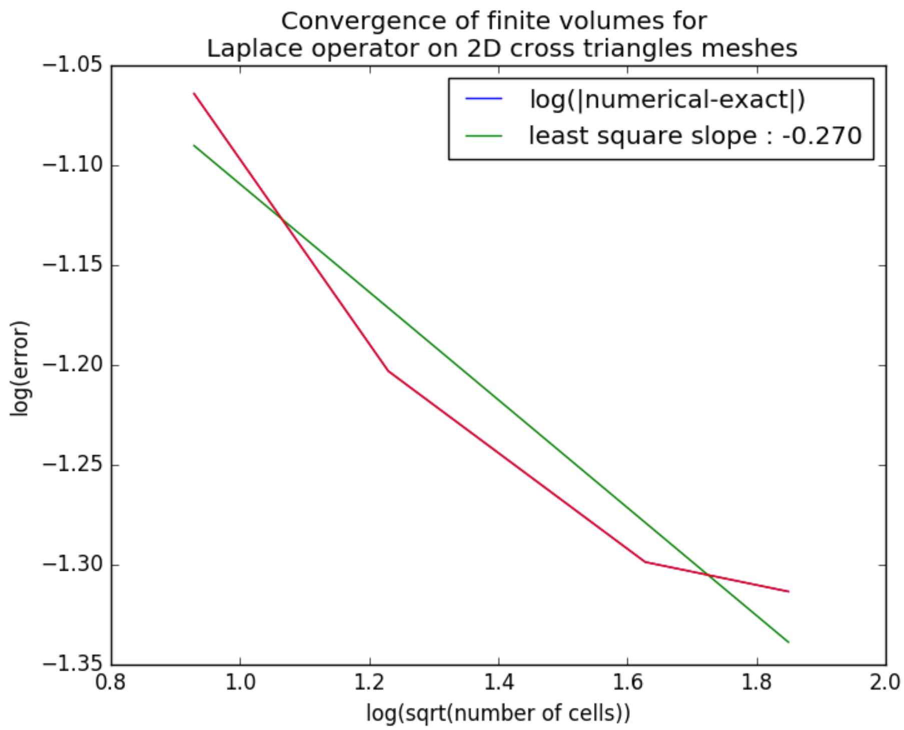


result 1 | result 2 | result 3 - | -- | -

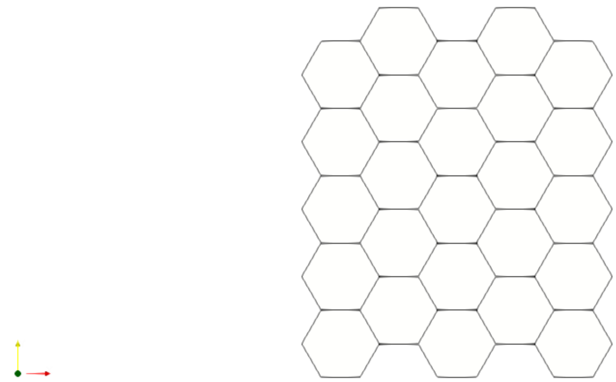




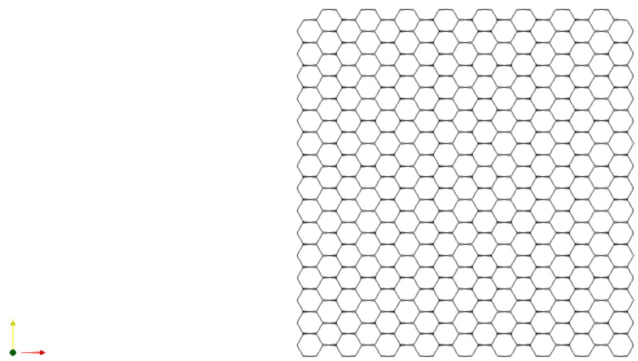



Convergence of finite volumes for
Laplace operator on 2D cross triangles meshes
log(|numerical-exact|)
least square slope : -0.270
log(error)
log(sqrt(number of cells))
−1.05
−1.10
−1.15
−1.20
−1.25
−1.30
−1.35
0.8
1.0
1.2
1.4
1.6
1.8
2.0

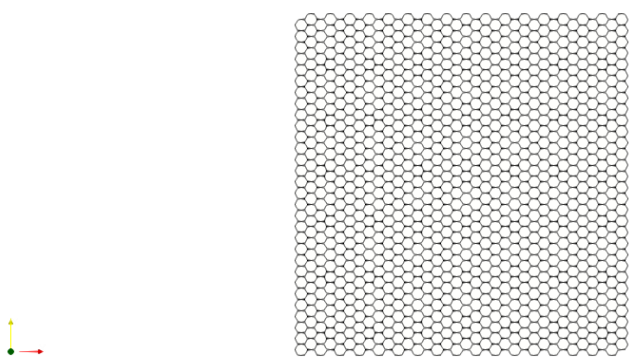
## 1.8 Hexagonal meshes

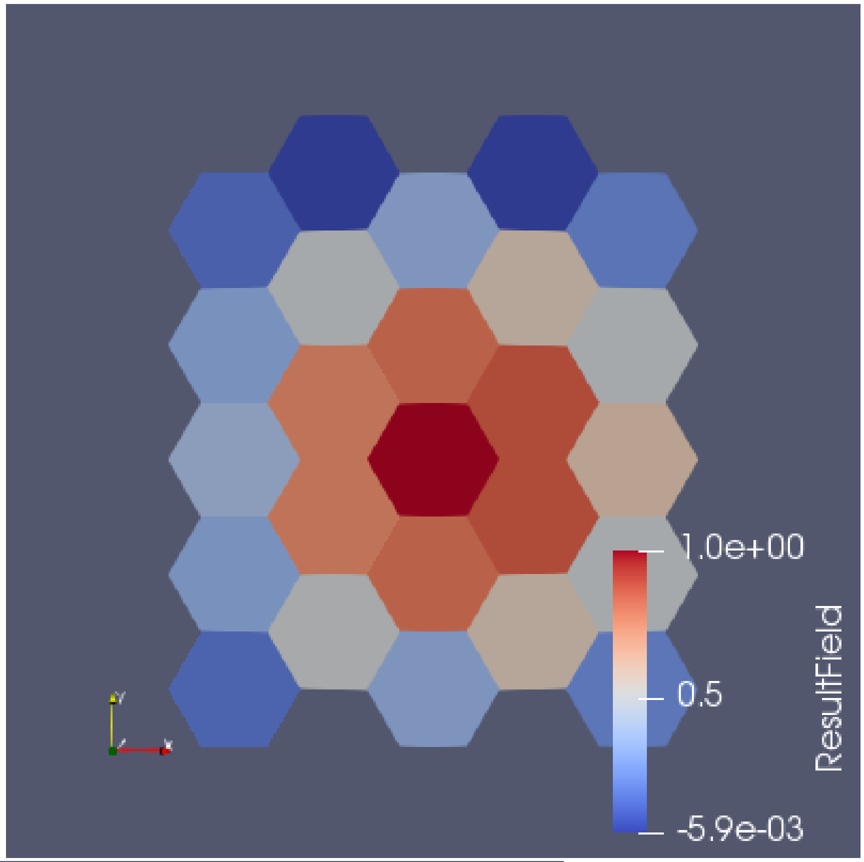


mesh 1 | mesh 2 | mesh 3 - | - - | -

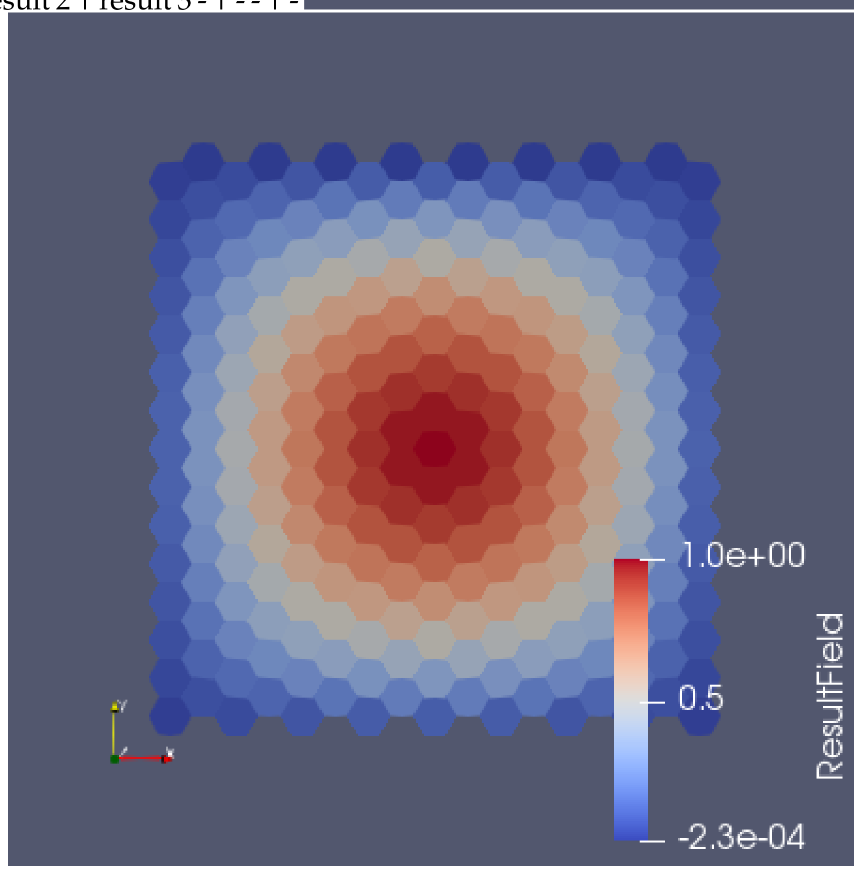


| |

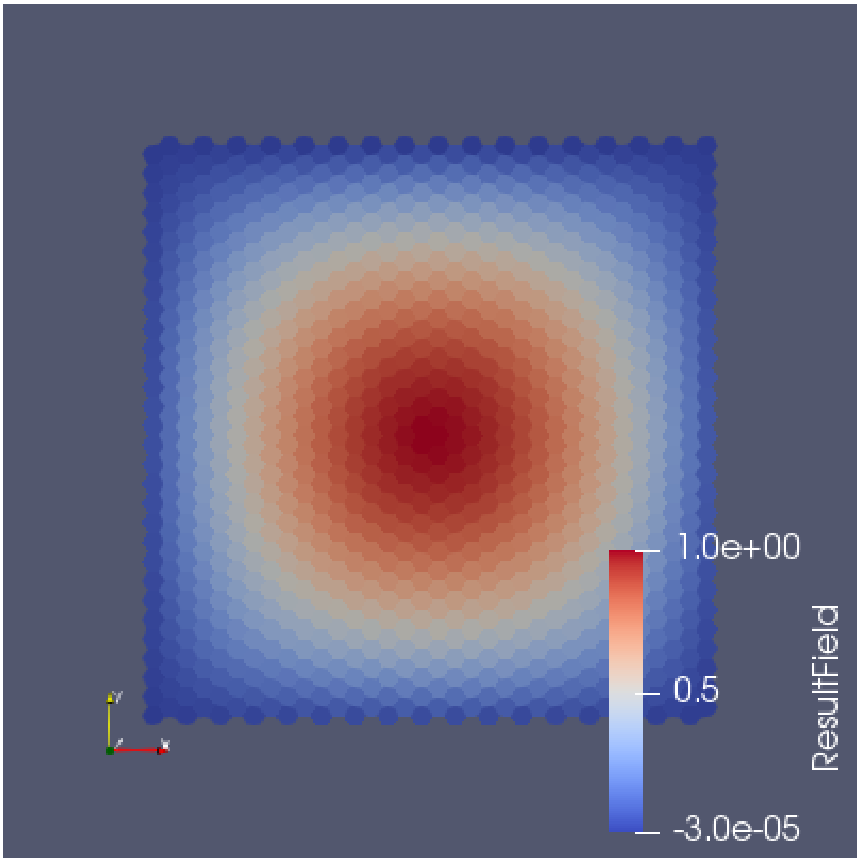


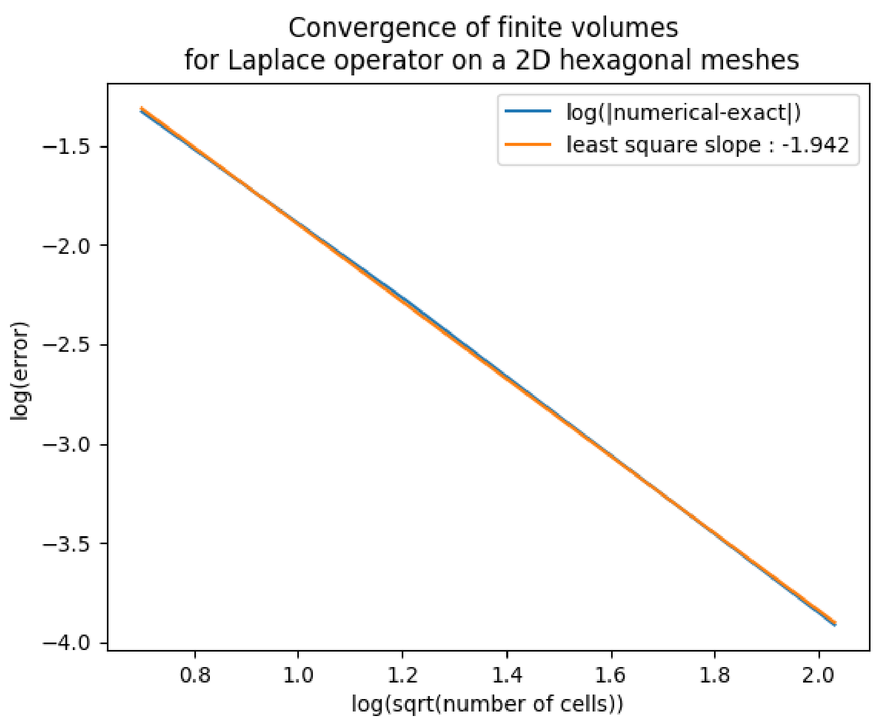



result 1 | result 2 | result 3 - | - - | -

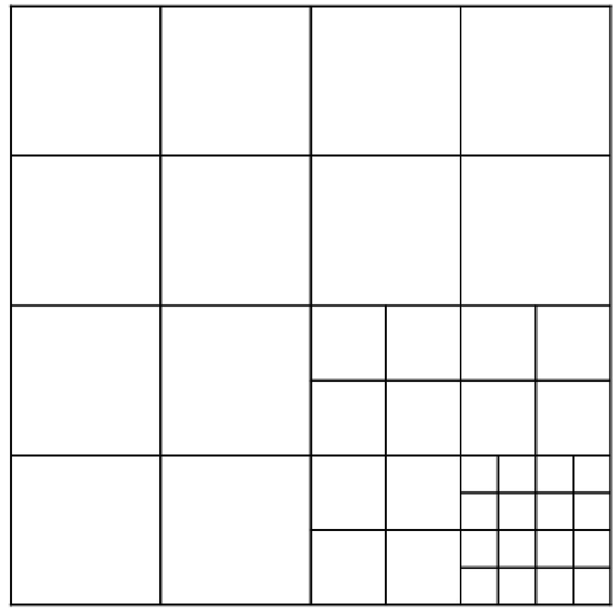



| |
|---|

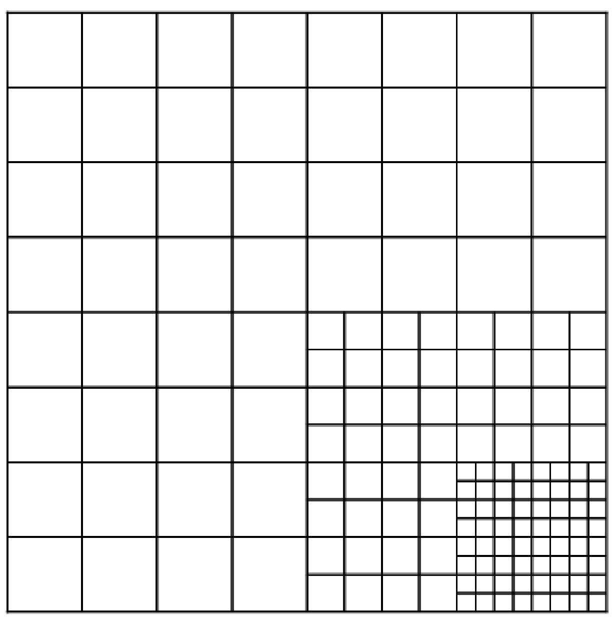

1.0e+00
0.5
-3.0e-05
ResultField


Convergence of finite volumes
for Laplace operator on a 2D hexagonal meshes
log(|numerical-exact|)
least square slope : -1.942
log(error)
log(sqrt(number of cells))

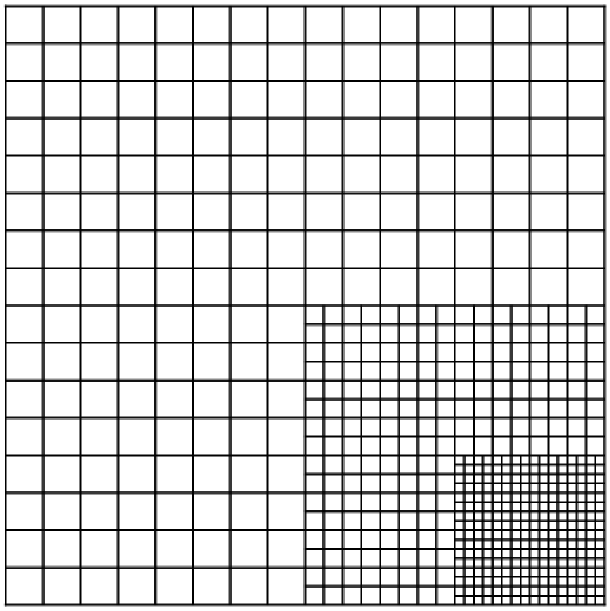
## 1.9 Locally refined meshes



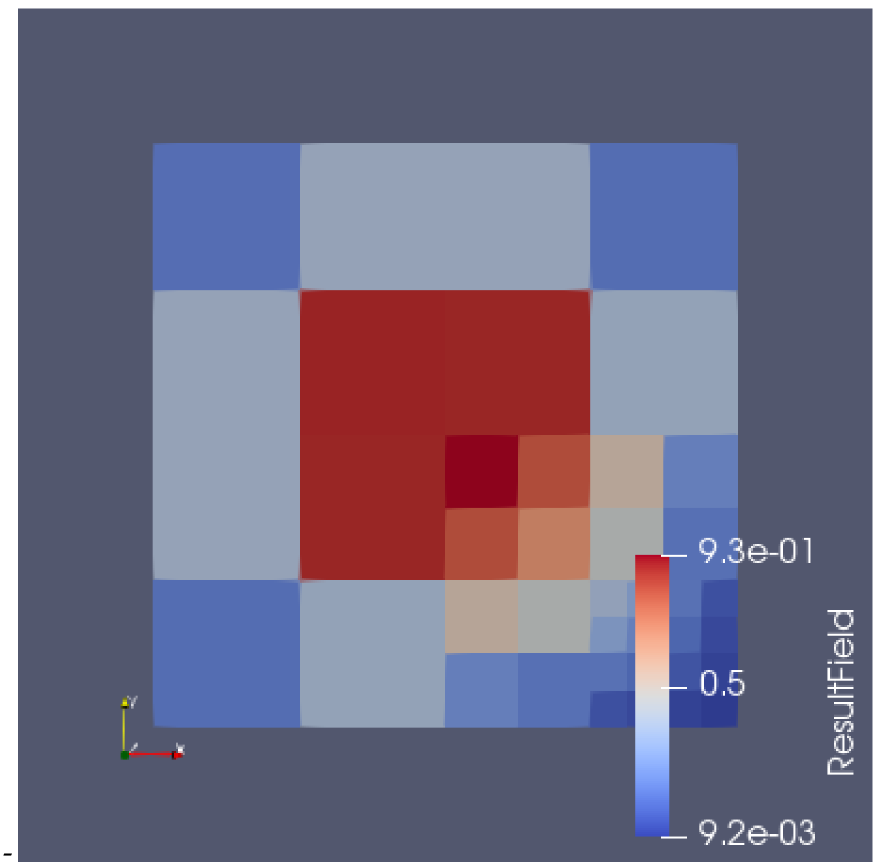
mesh 1 | mesh 2 | mesh 3 - | - - | -

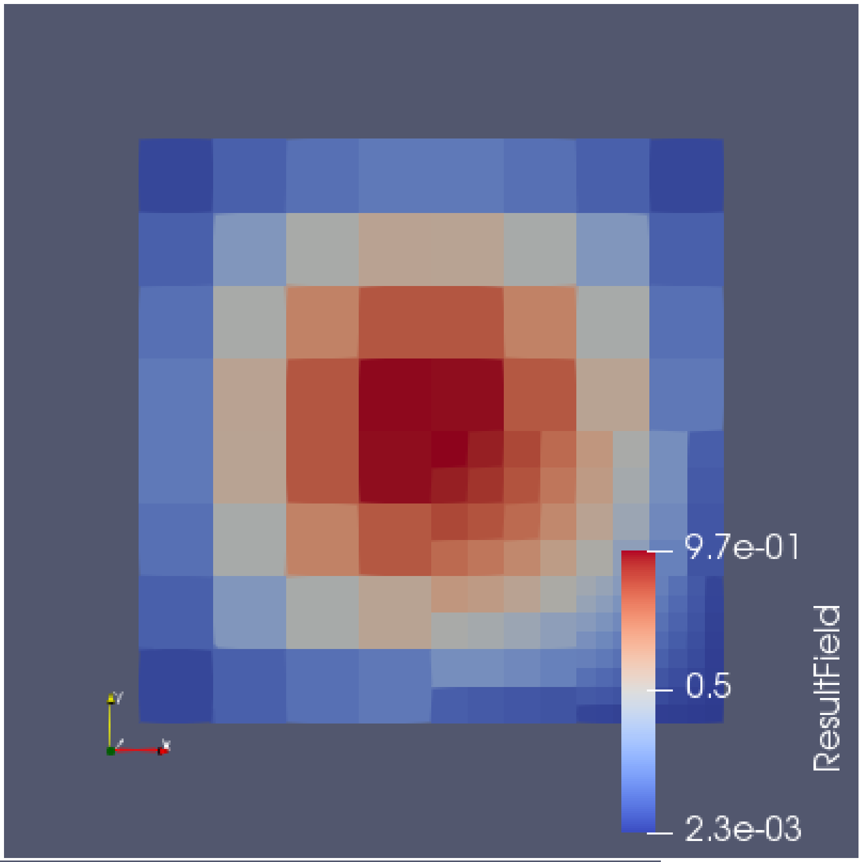


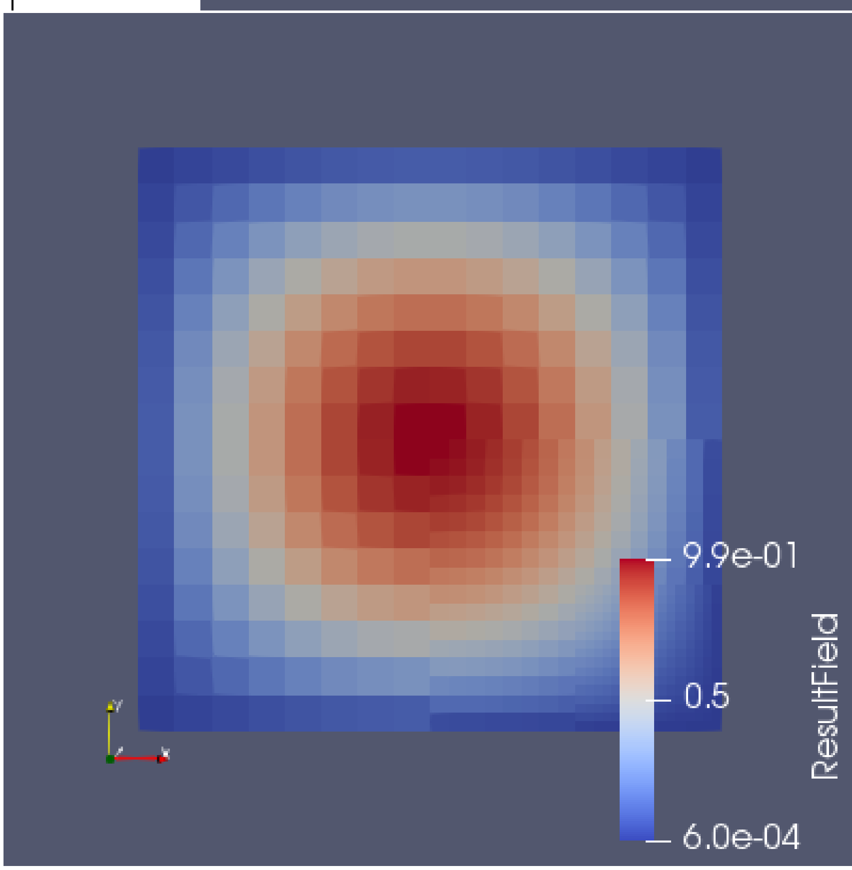
| |

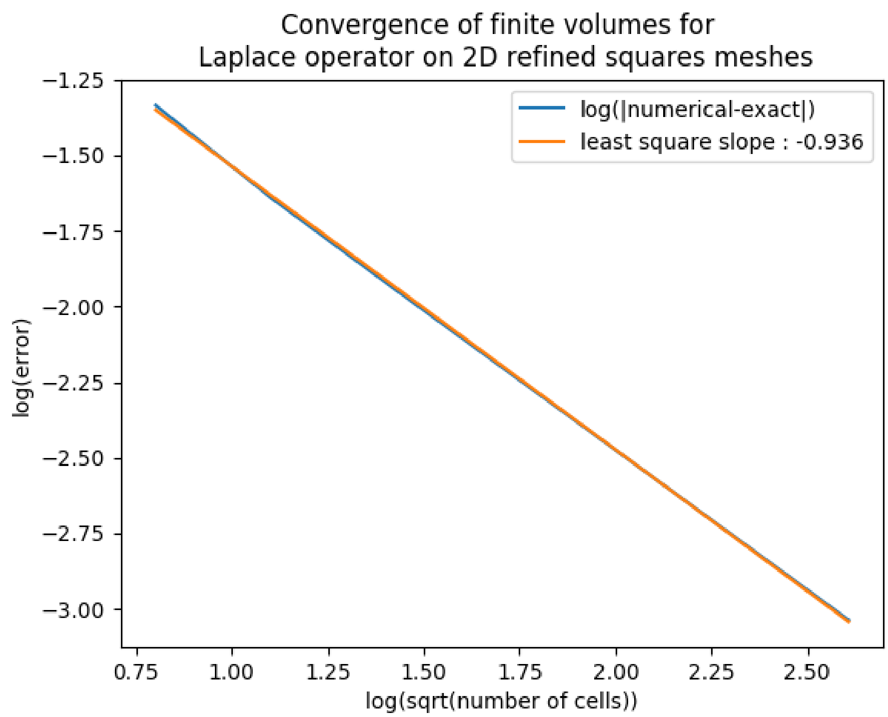


result 1 | result 2 | result 3 - | - - | -

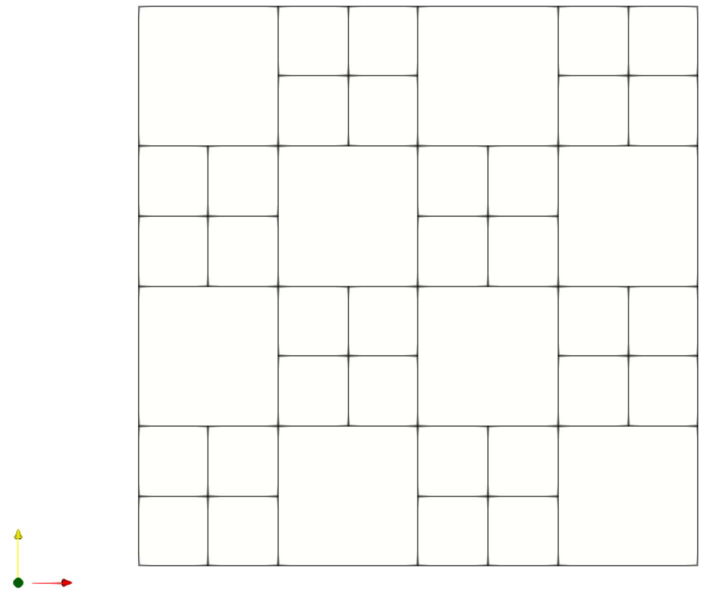



9.7e-01
0.5
2.3e-03
ResultField

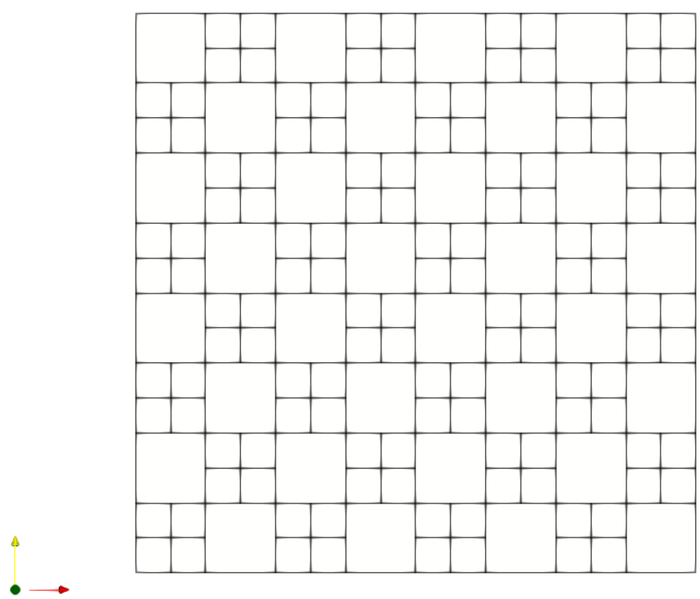

9.9e-01
0.5
6.0e-04
ResultField


Convergence of finite volumes for
Laplace operator on 2D refined squares meshes
log(|numerical-exact|)
least square slope : -0.936
log(error)
log(sqrt(number of cells))
−1.25
−1.50
−1.75
−2.00
−2.25
−2.50
−2.75
−3.00
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50

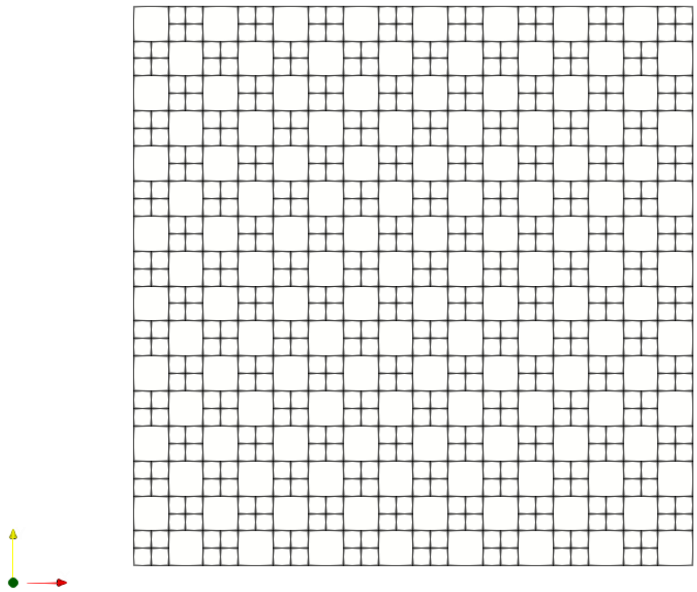
## 1.10 Checkerboard meshes



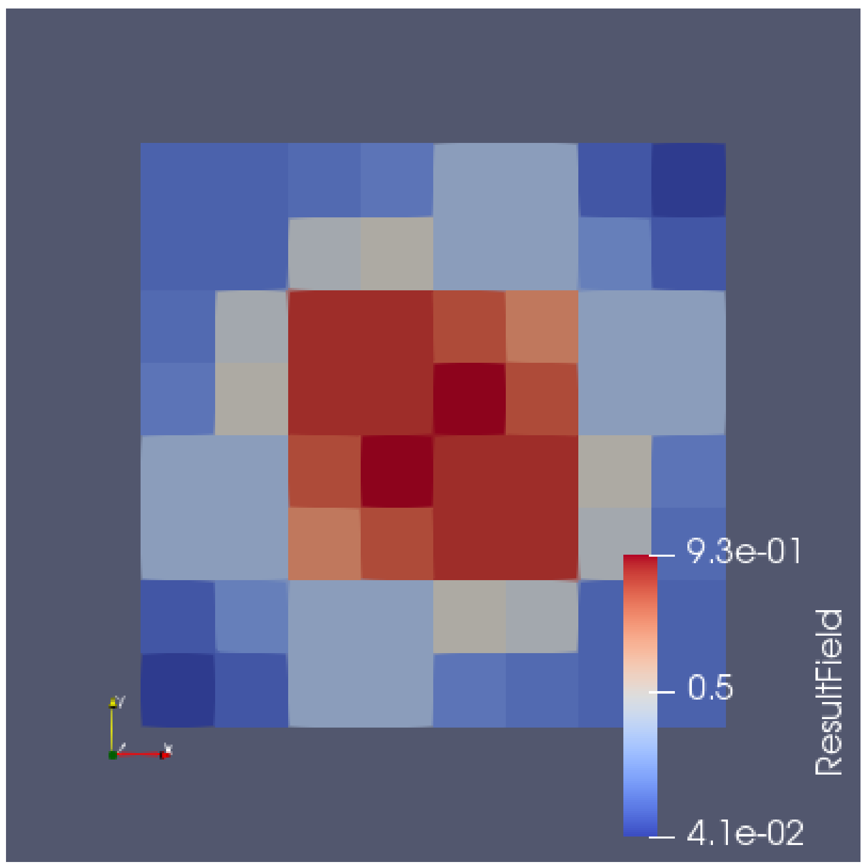
mesh 1 | mesh 2 | mesh 3 - | - - | -

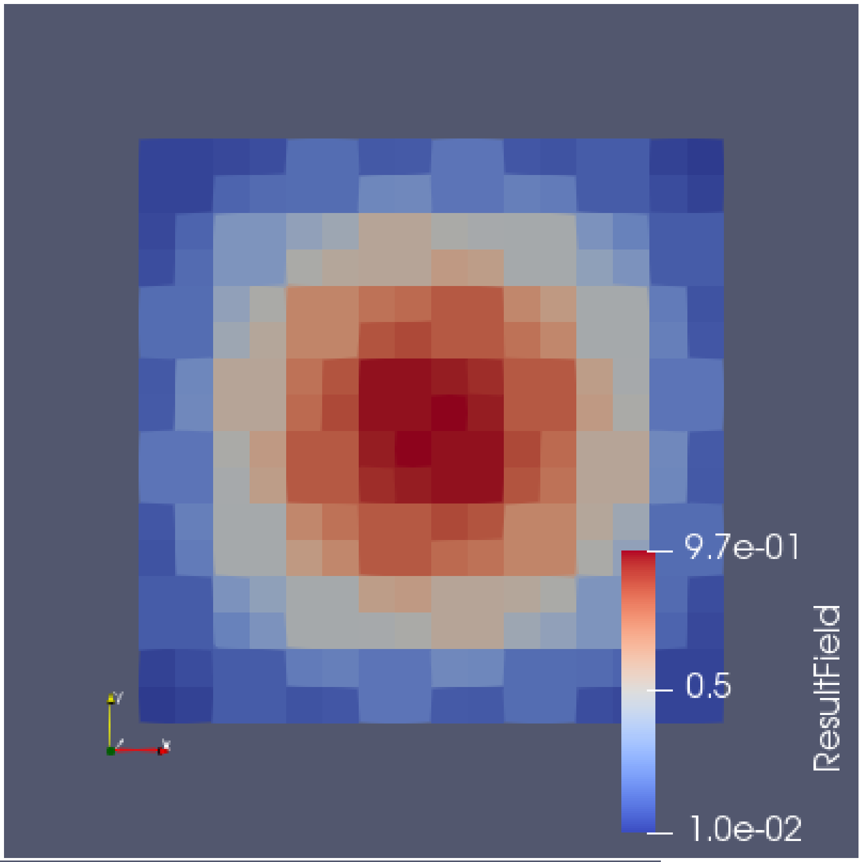


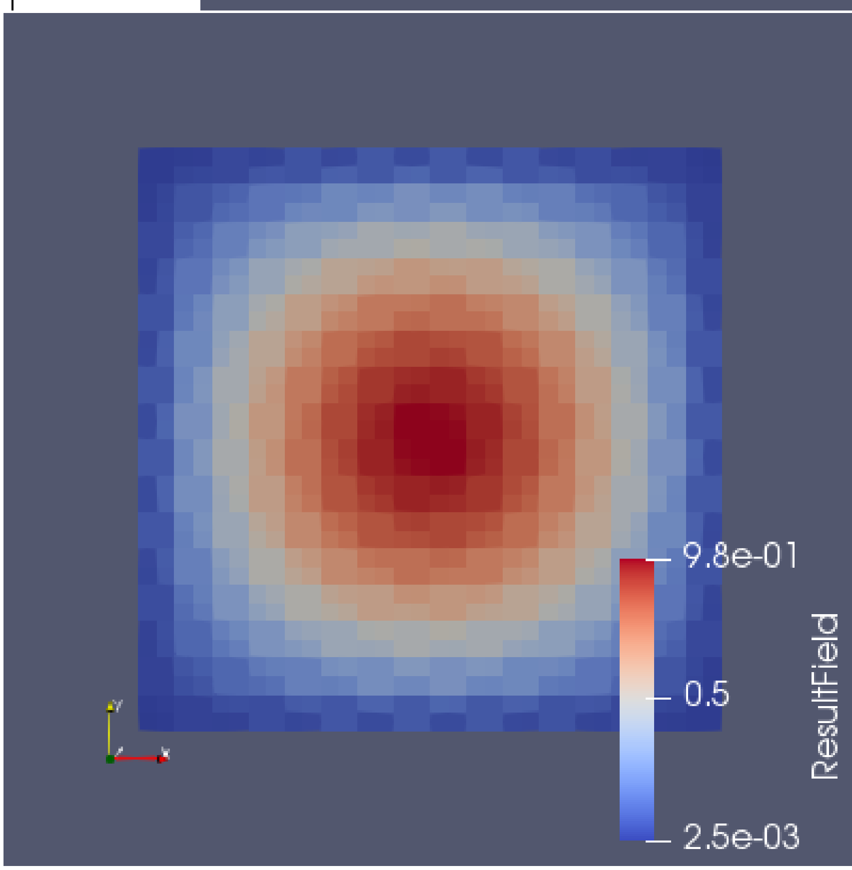
| |

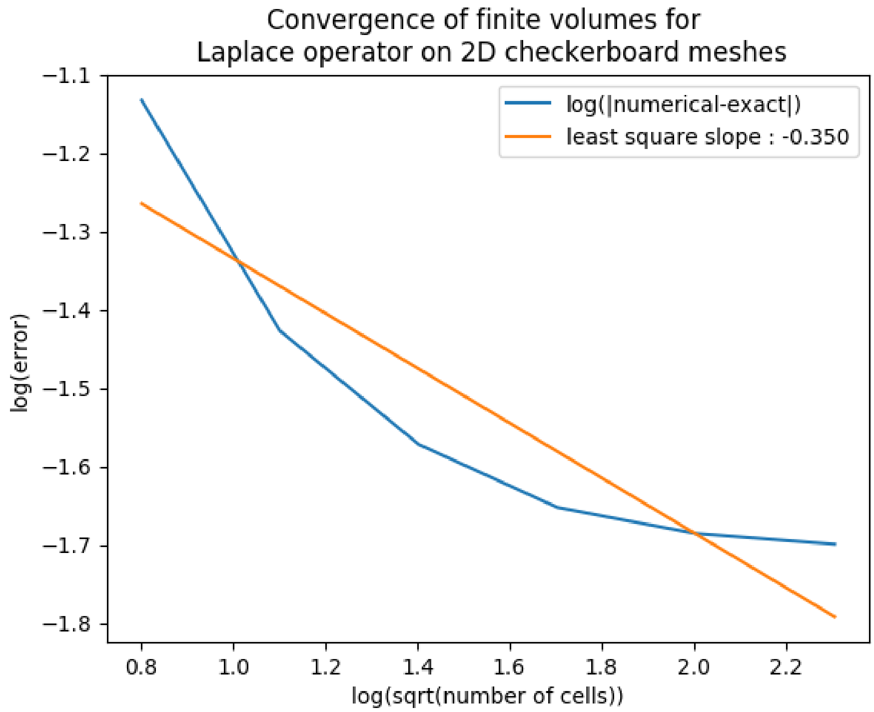


result 1 | result 2 | result 3 - | - - | -

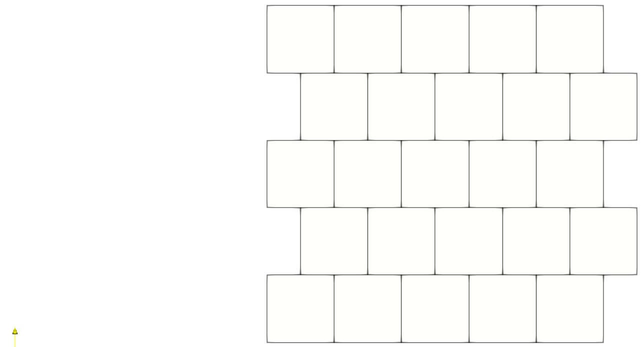



9.7e-01
0.5
1.0e-02
ResultField

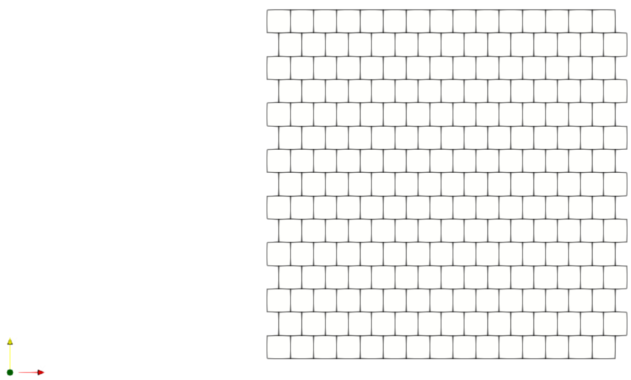

9.8e-01
0.5
2.5e-03
ResultField


Convergence of finite volumes for
Laplace operator on 2D checkerboard meshes
log(|numerical-exact|)
least square slope : -0.350
log(error)
−1.1
−1.2
−1.3
−1.4
−1.5
−1.6
−1.7
−1.8
0.8
1.0
1.2
1.4
1.6
1.8
2.0
2.2
log(sqrt(number of cells))

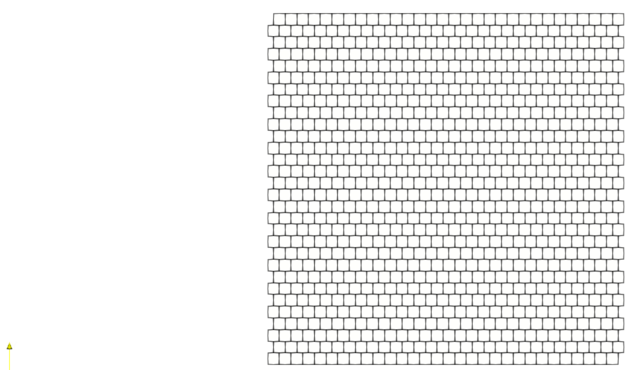
## 1.11 Brick wall meshes

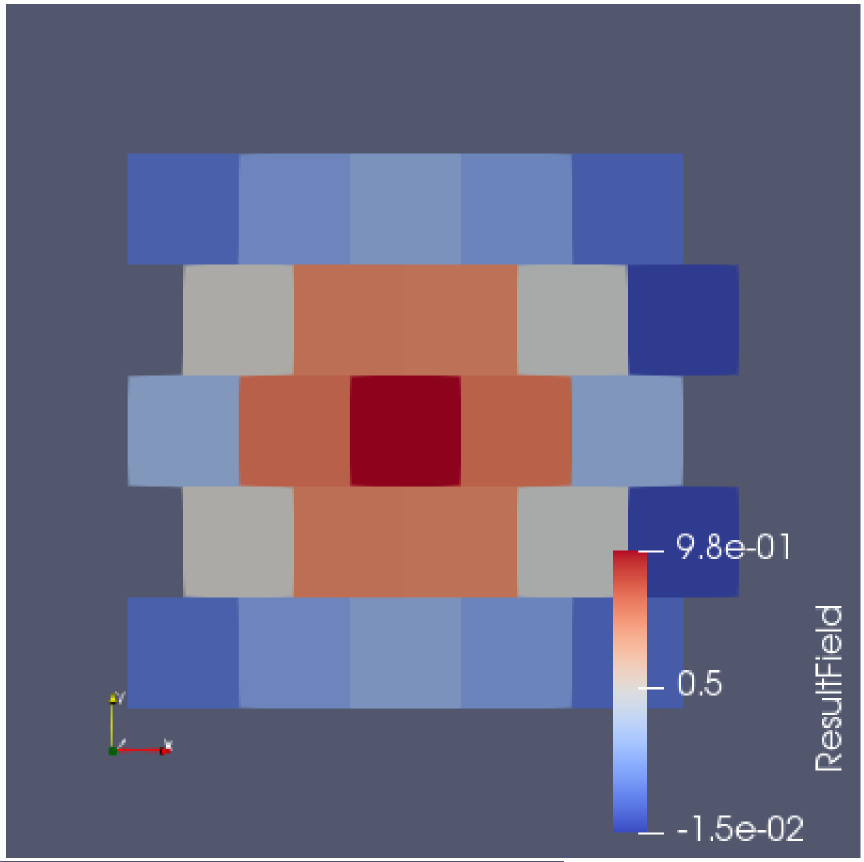


mesh 1 | mesh 2 | mesh 3 - | - - | -

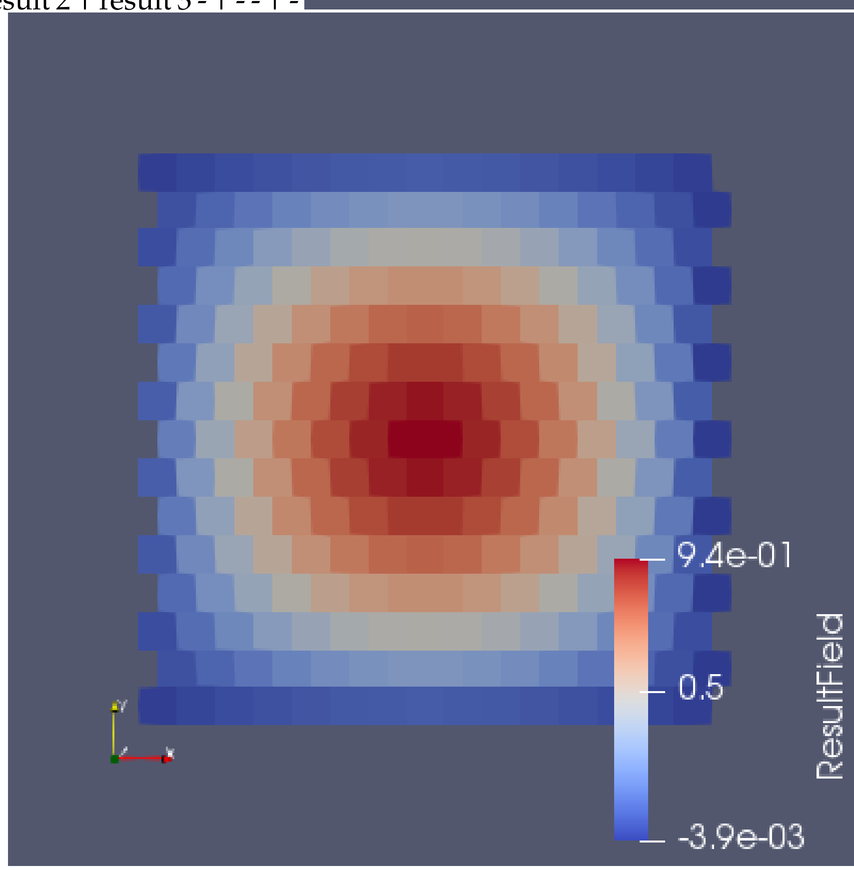


| |

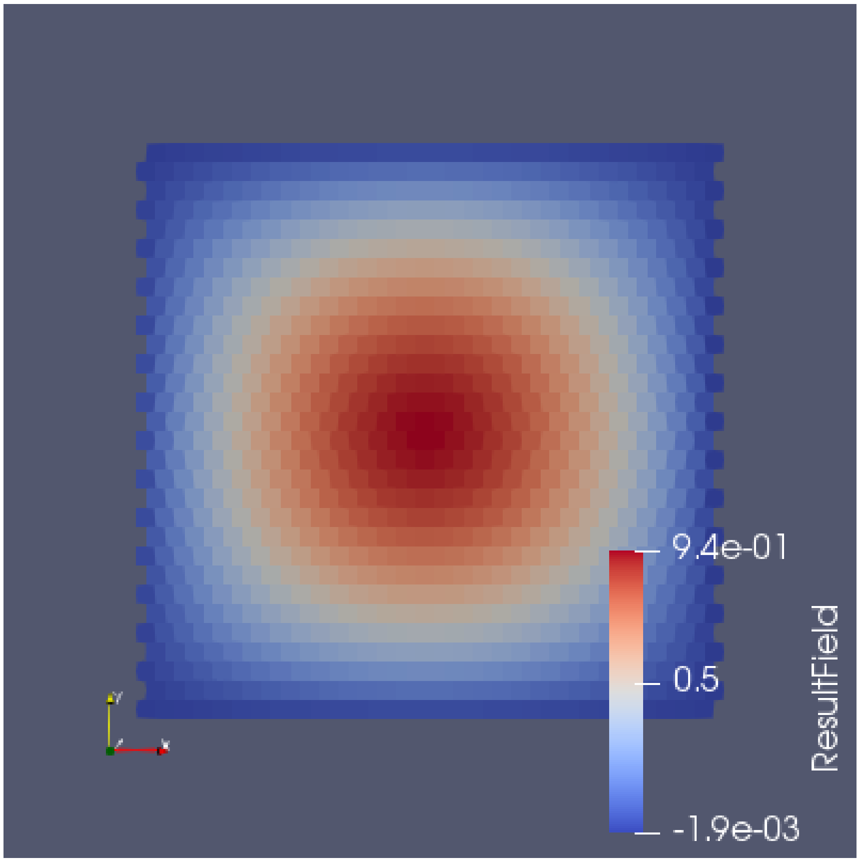


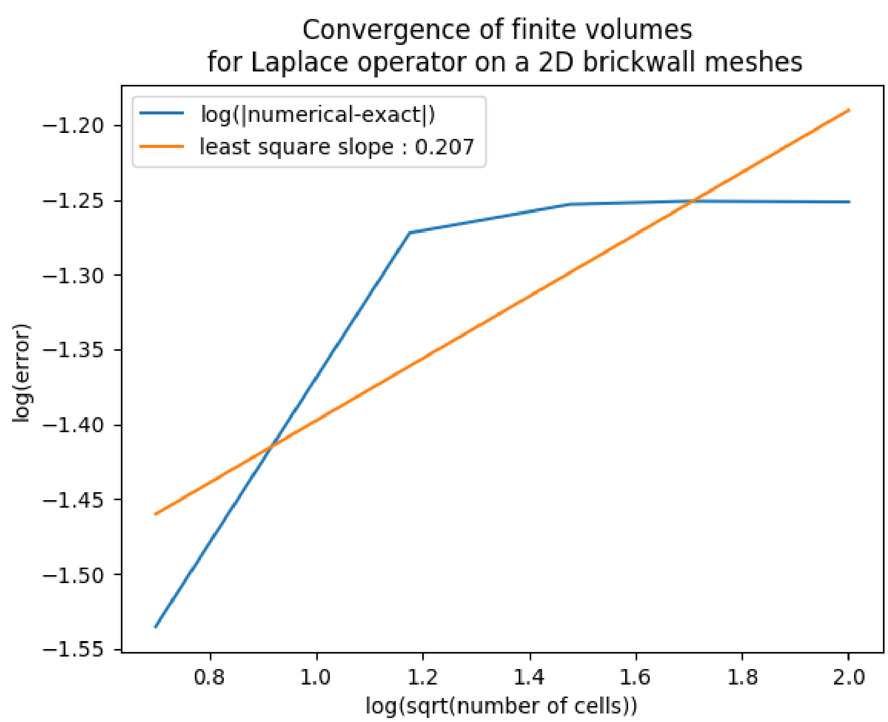



result 1 | result 2 | result 3 - | - - | -

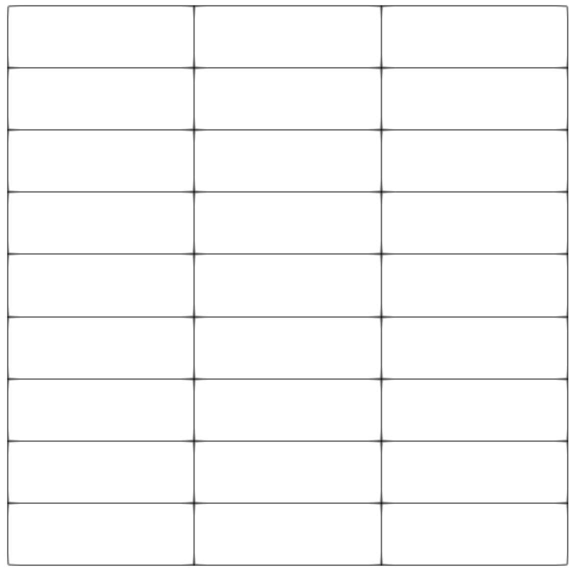



| |

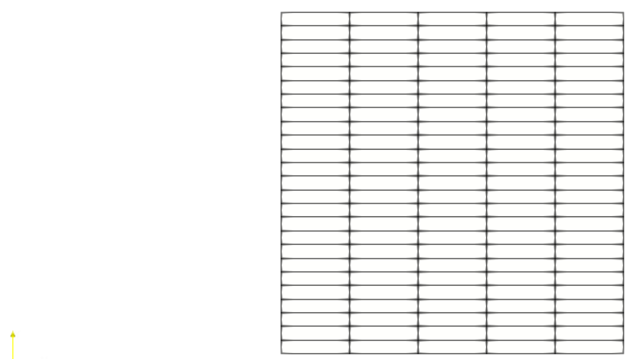

9.4e-01
0.5
-1.9e-03
ResultField


Convergence of finite volumes
for Laplace operator on a 2D brickwall meshes
log(|numerical-exact|)
least square slope : 0.207
log(error)
log(sqrt(number of cells))

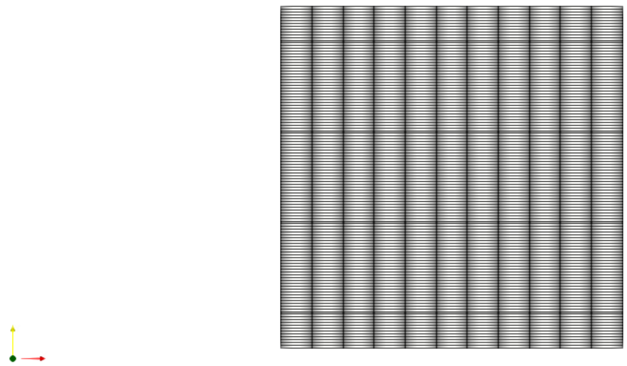
### 1.12 Long rectangle meshes ( $(n, n^2)$ rectangular grid)

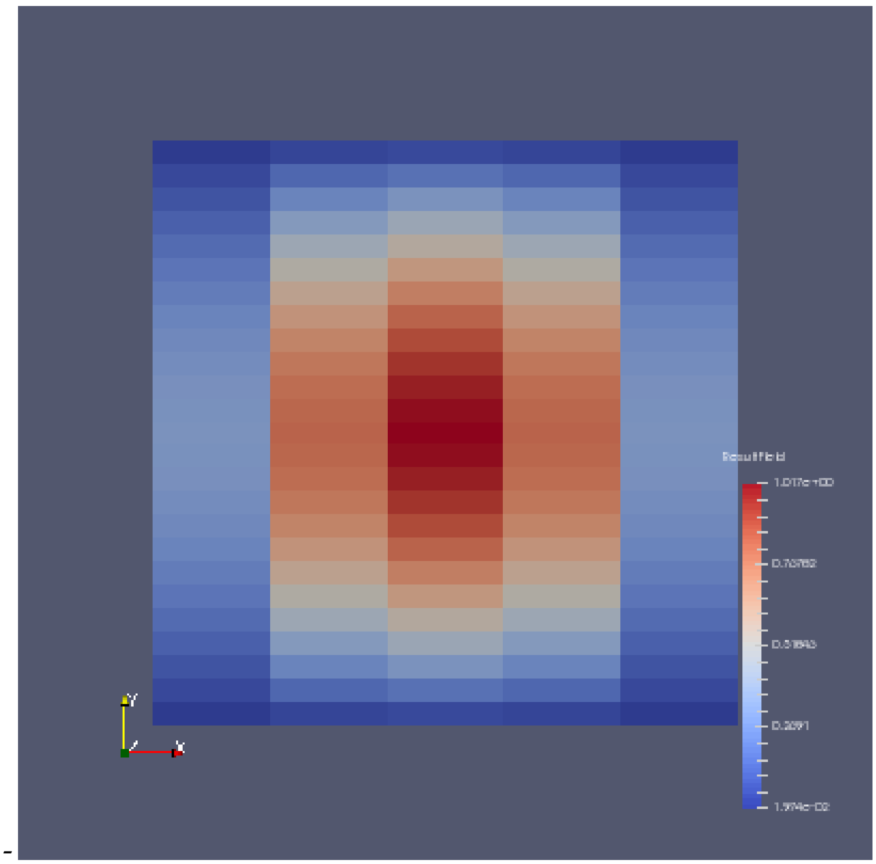


mesh 1 | mesh 2 - | - -

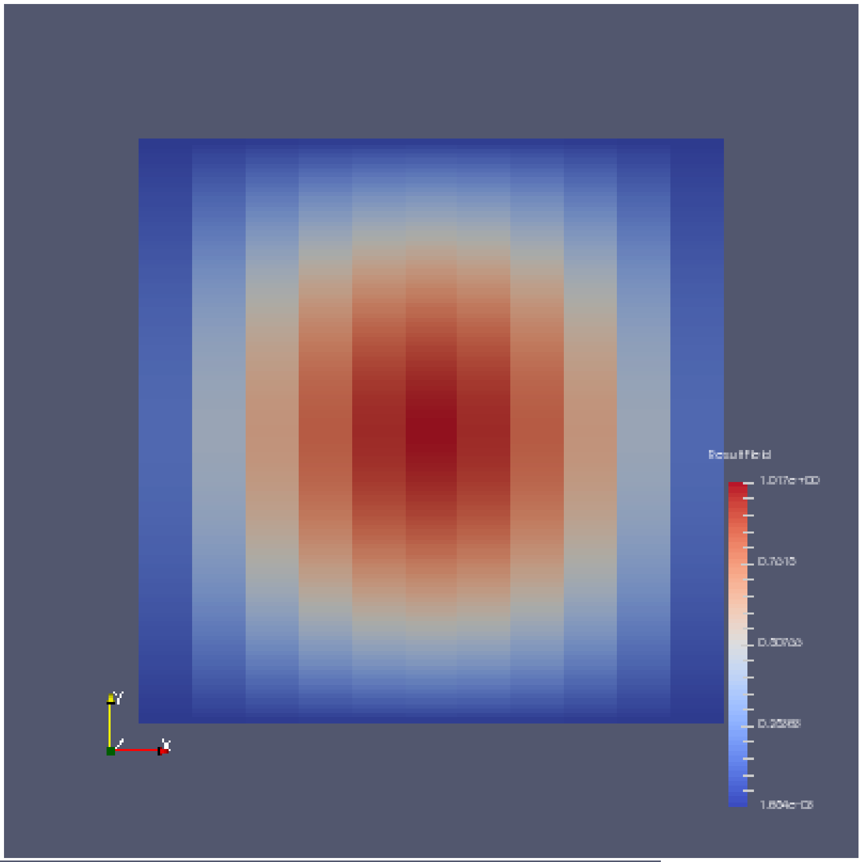


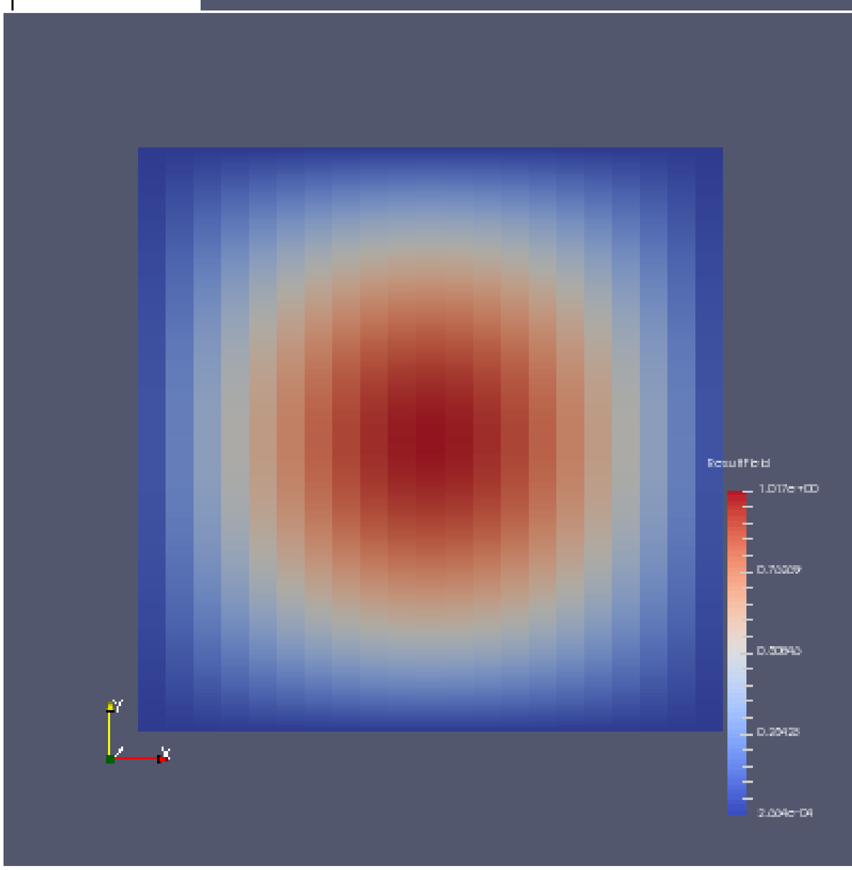
| |

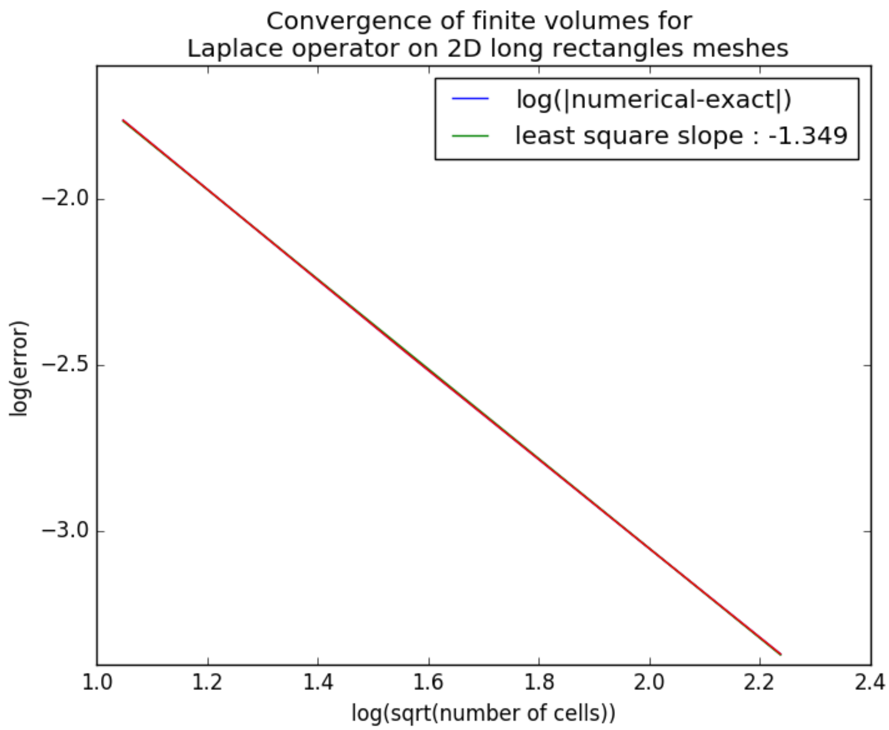




result 1 | result 2 | result 3 - | -- | -

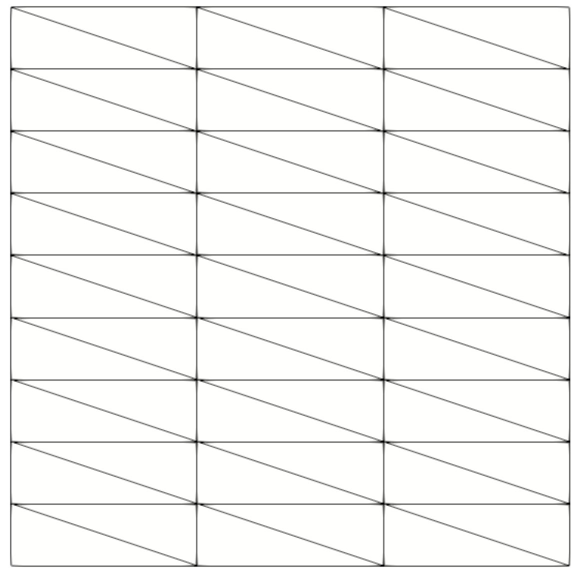

ResultField
1.017e+00
0.7615
0.50763
0.2538
1.854e-03
Y
X


ResultField
1.017e+00
0.76289
0.50843
0.25428
2.004e-04
Y
X

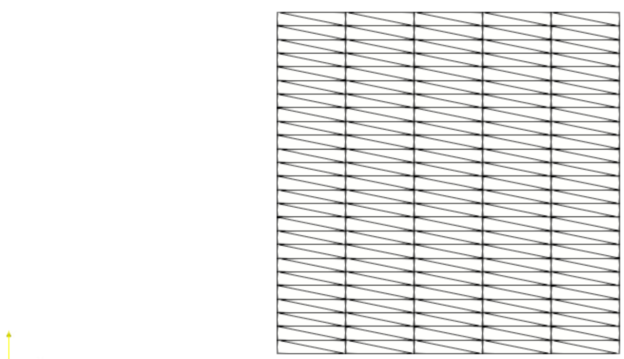


Convergence of finite volumes for
Laplace operator on 2D long rectangles meshes
log(|numerical-exact|)
least square slope : -1.349
−2.0
−2.5
−3.0
log(error)
1.0
1.2
1.4
1.6
1.8
2.0
2.2
2.4
log(sqrt(number of cells))

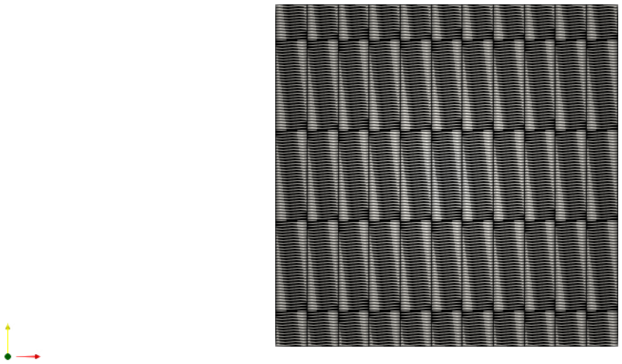
## 1.13 Long right triangle meshes (from a $(n, n^2)$ rectangular grid)

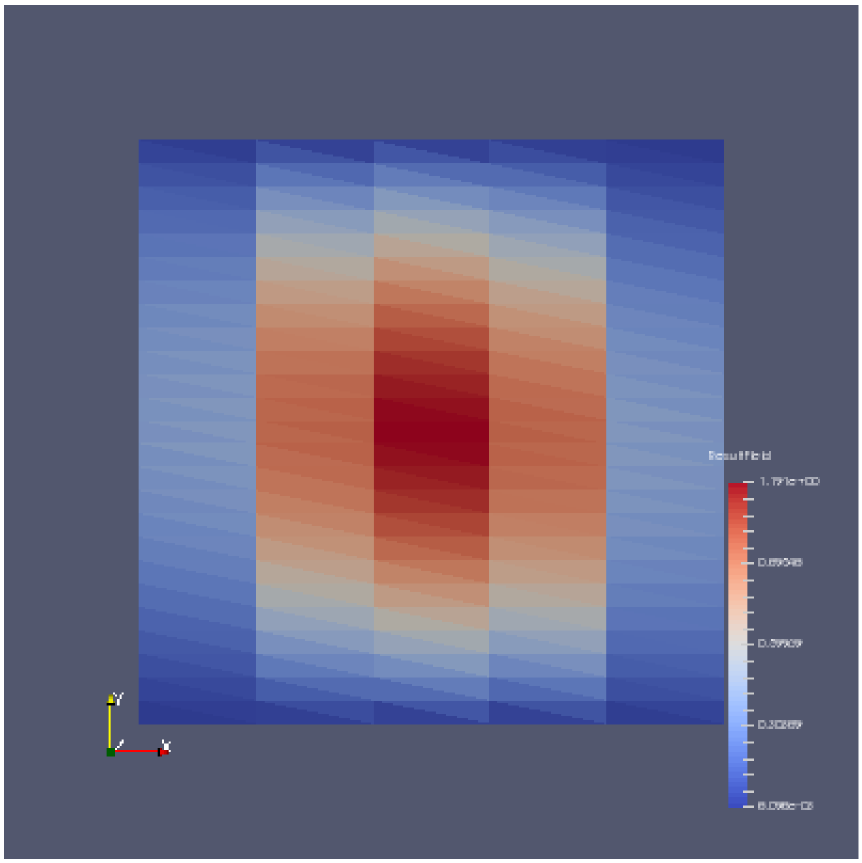


mesh 1 | mesh 2 | mesh 3 - | - - | - -

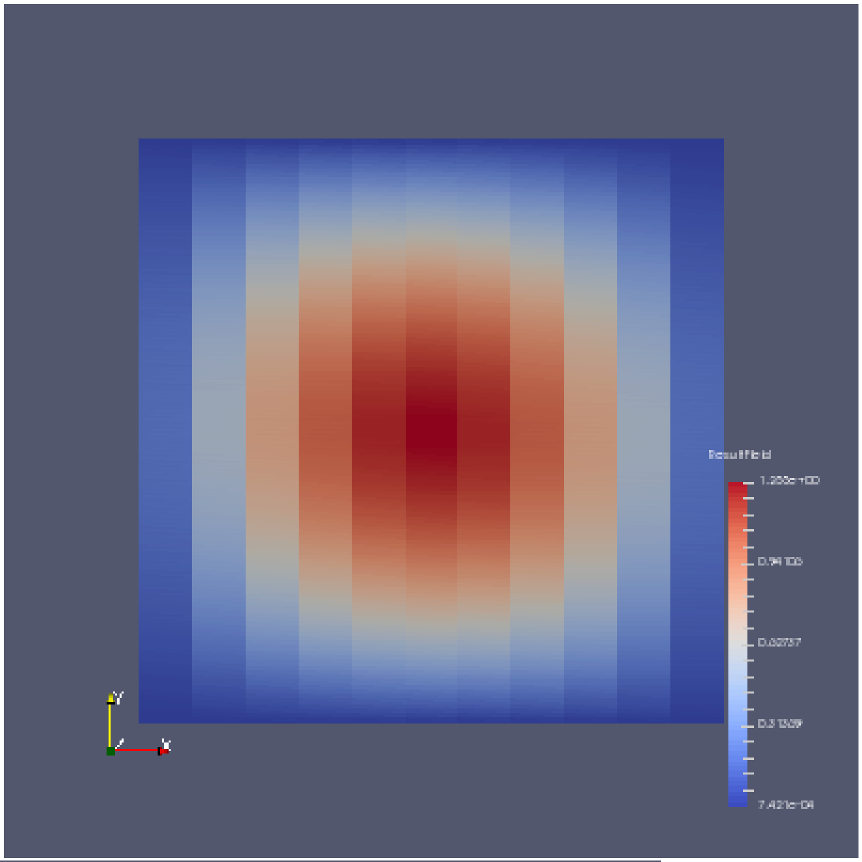


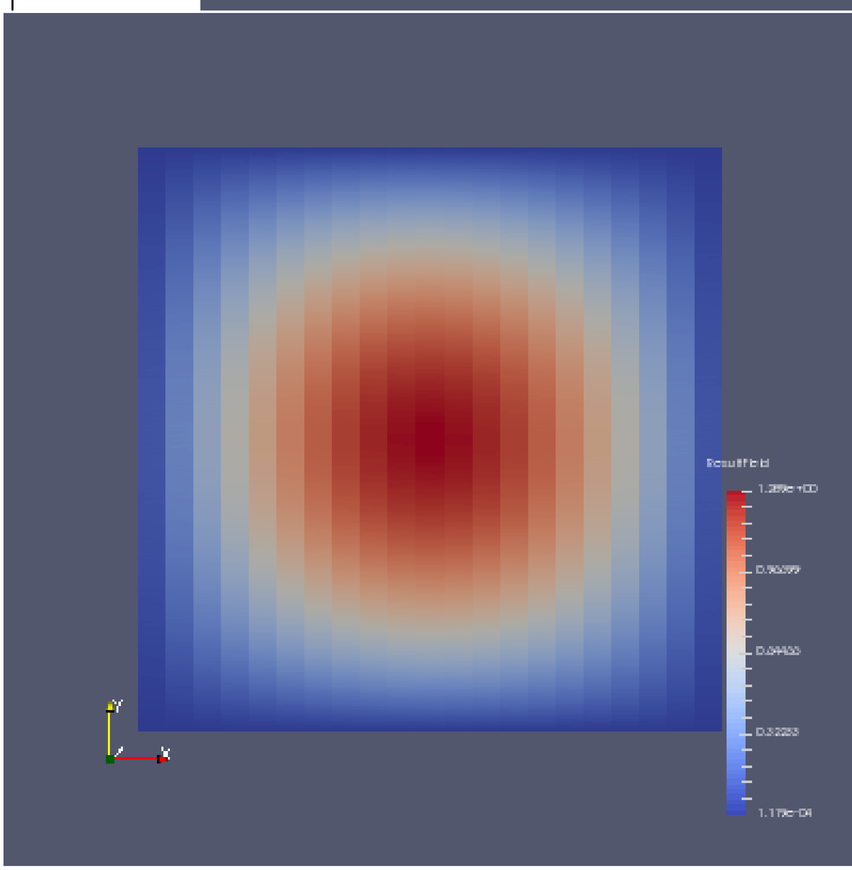
| |

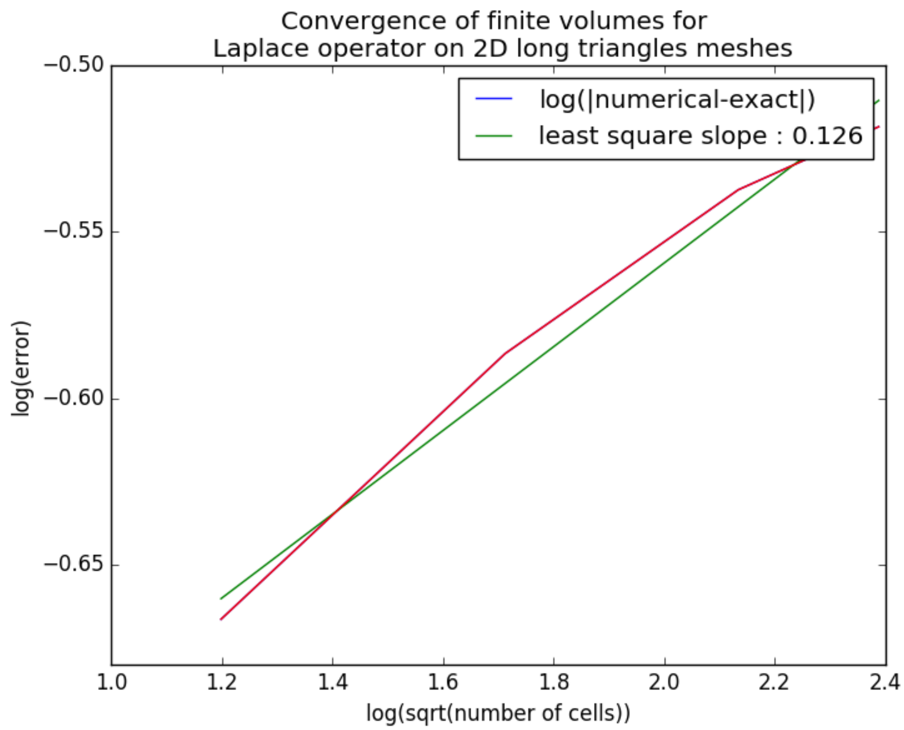





result 1 | result 2 | result 3 - | - - | -

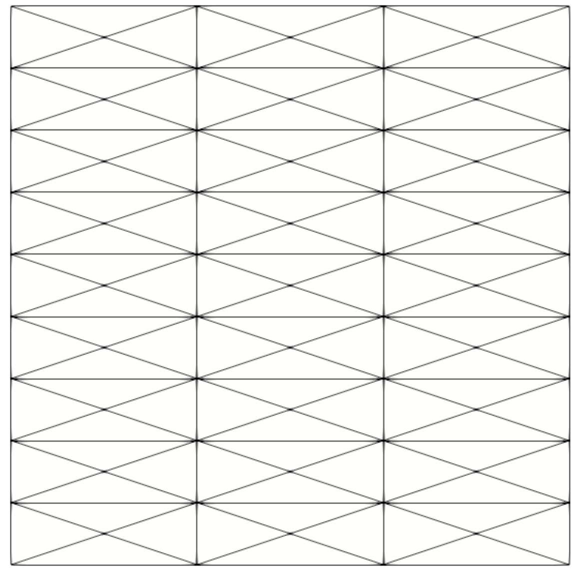

Result field
1.255e+00
0.94103
0.62737
0.31369
7.421e-04
Y
X


Result field
1.289e+00
0.96699
0.64466
0.32233
1.119e-04
Y
X

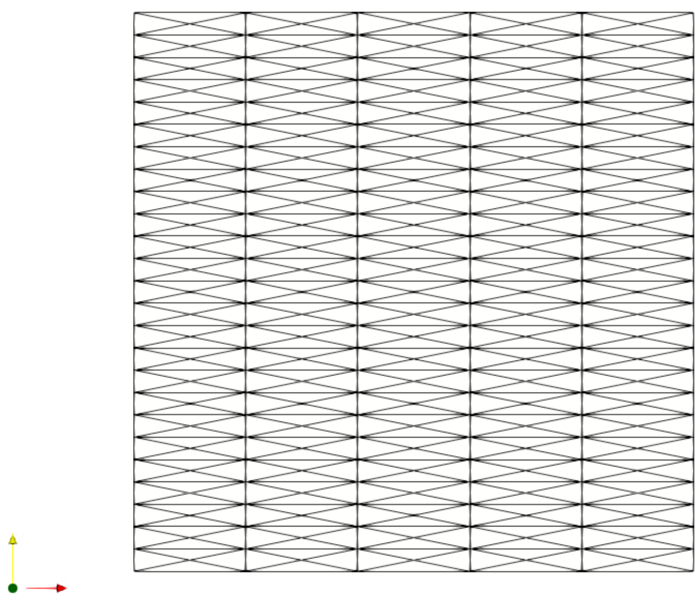


Convergence of finite volumes for
Laplace operator on 2D long triangles meshes
log(|numerical-exact|)
least square slope : 0.126
−0.50
−0.55
−0.60
−0.65
log(error)
1.0
1.2
1.4
1.6
1.8
2.0
2.2
2.4
log(sqrt(number of cells))

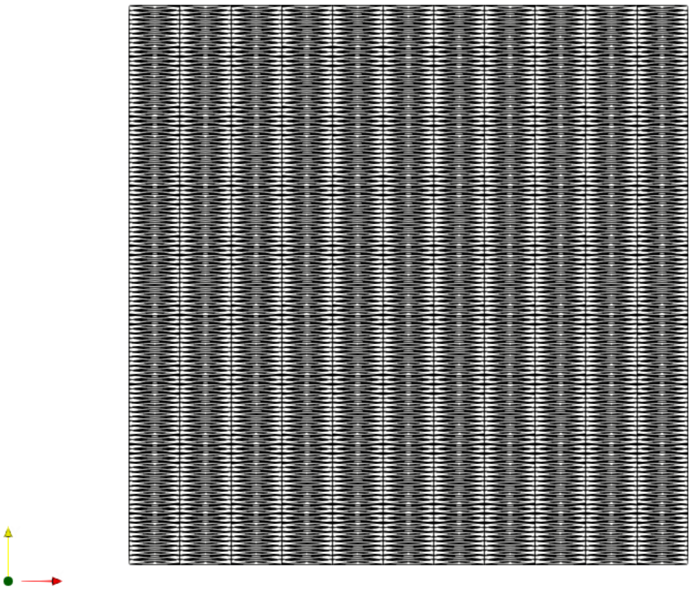
### 1.14 Flat cross triangle meshes (from a $(n, n^2)$ rectangular grid)

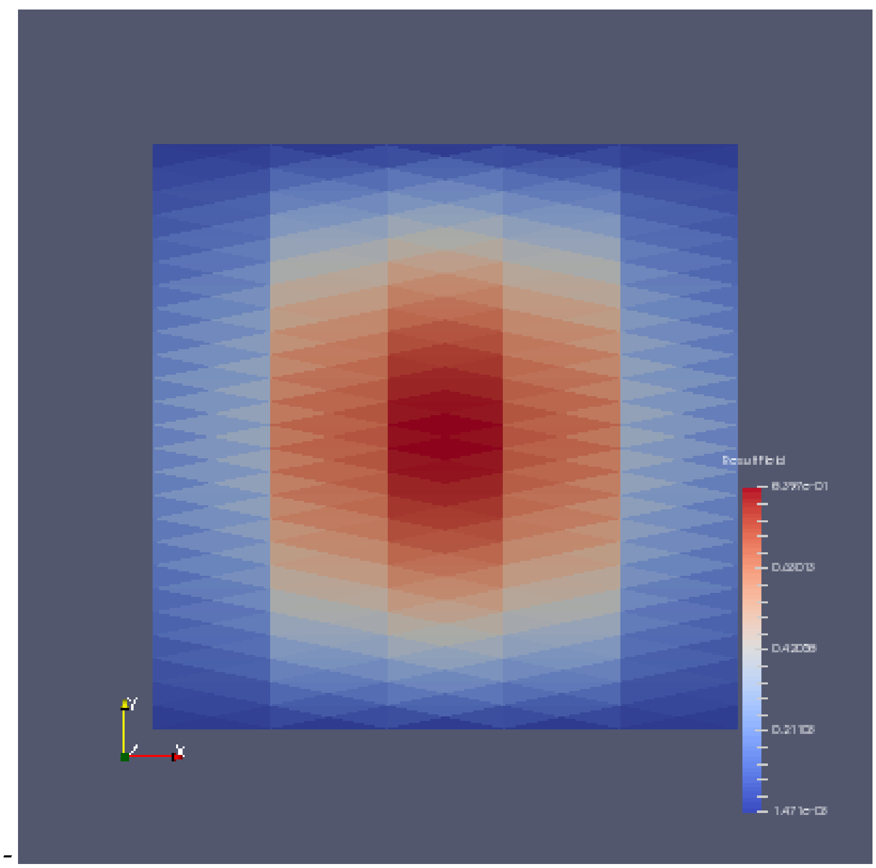


mesh 1 | mesh 2 | mesh 3 - | - - | - -

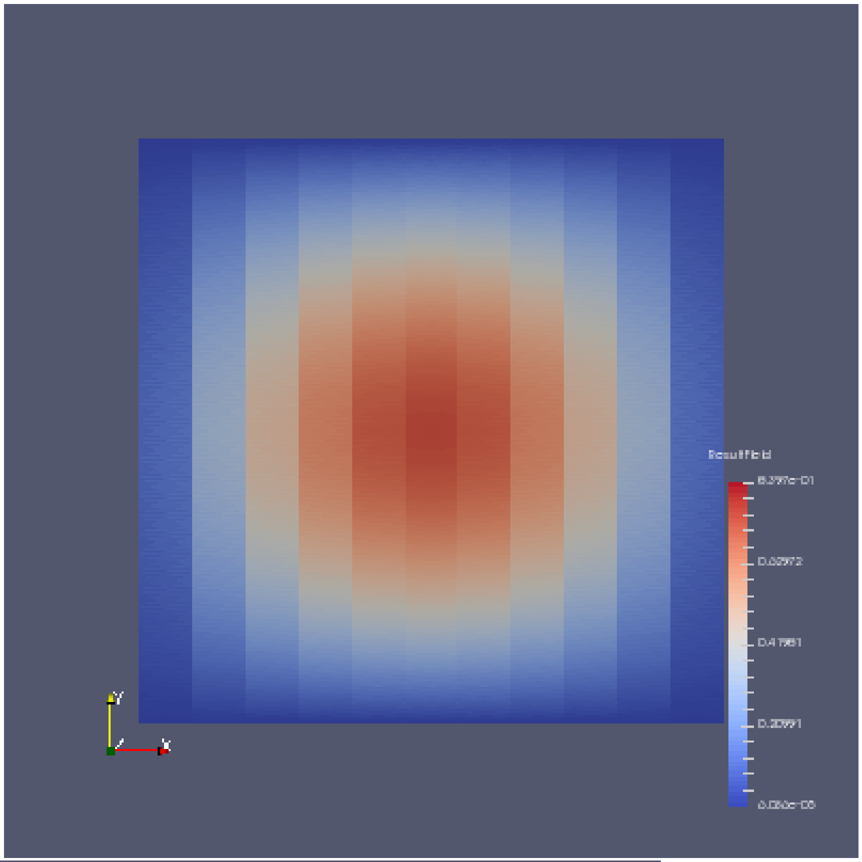


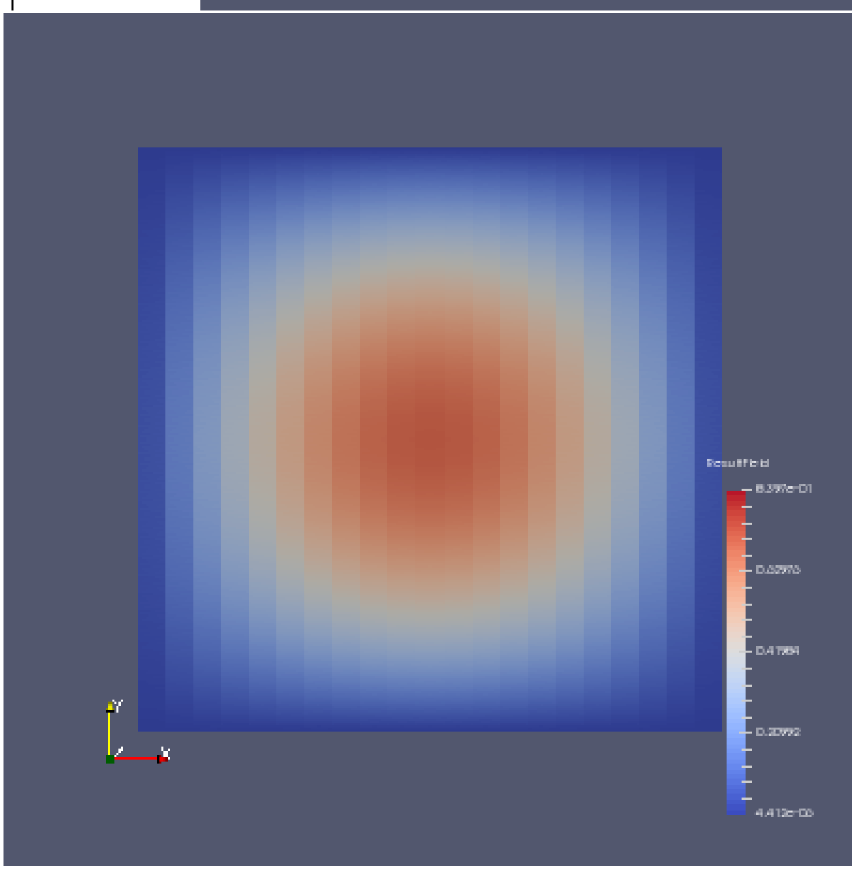
| |

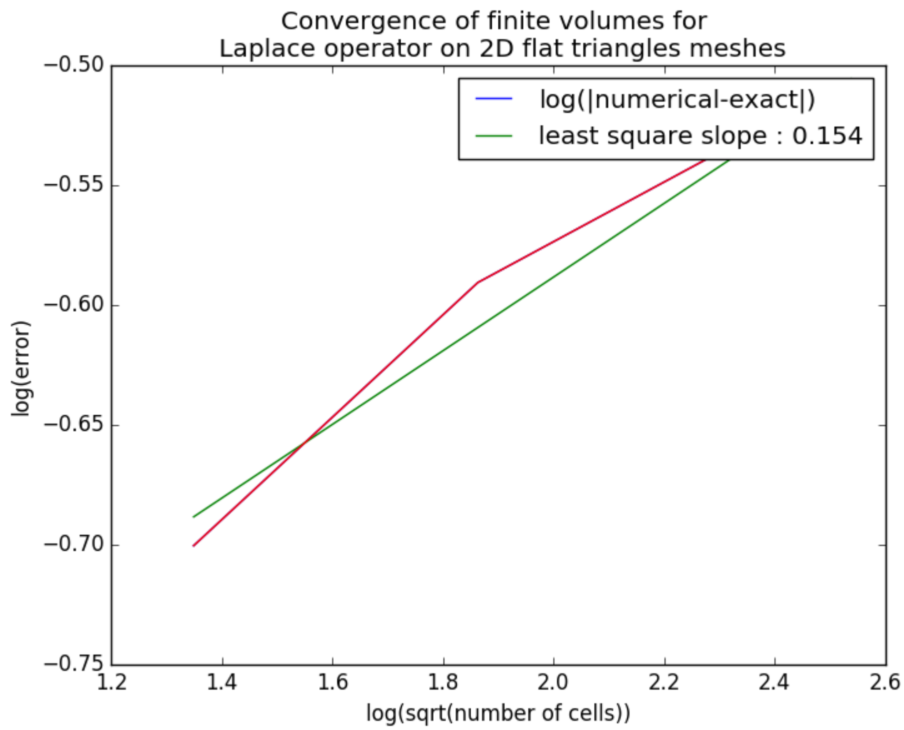




result 1 | result 2 | result 3 - | -- | -








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

## 1.15 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 voisinnes
        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 Diri
        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_Result
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")
```