Convergence_Diffusion_FV5_SQUARE

January 26, 2019

```
In [15]: from IPython.display import display, Markdown
         with open('DiffusionProblemOnSquare.md', 'r') as file1:
             DiffusionProblemOnSquare = file1.read()
         with open('DescriptionFV5DiffusionProblem.md', 'r') as file2:
             DescriptionFV5DiffusionProblem = file2.read()
         with open('CodeFV5DiffusionProblem.md', 'r') as file3:
             CodeFV5DiffusionProblem = file3.read()
```

1 FV5 scheme for a Diffusion equation

```
In [16]: display(Markdown(DiffusionProblemOnSquare))
```

1.1 The Diffusion problem on the square

We consider the following diffusion problem with Dirichlet boundary conditions

$$\begin{cases} -\left(\partial_{xx}u + K\partial_{yy}u\right) = f \text{ on } \Omega \\ u = 0 \text{ on } \partial\Omega \end{cases}$$

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

$$f = (1+K)\pi^2 sin(\pi x) sin(\pi y).$$

The unique solution of the problem is

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

The Diffusion equation can be written in a matrix form

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

and the associated diffusion matrix is

$$D = \begin{pmatrix} 1 & 0 \\ 0 & K \end{pmatrix}$$

We are interested in case where $K \gg 1$. In the following numerical results we take the value $K = 10^4$.

```
In [17]: display(Markdown(DescriptionFV5DiffusionProblem))
```

1.2 The FV5 scheme for the Diffusion equation

The domain Ω 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.

$\vec{n}_{ij}$ is the normal vector to 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 Diffusion 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

$$F_{ij} = \frac{u_j - u_i}{d_{ij}}{}^t\vec{n}_{ij}D\vec{n}_{ij},$$

for two cells i and j inside the domain,

and

$$F_{boundary} = \frac{u(x_f) - u_i}{d_{if}}{}^t\vec{n}_{if}D\vec{n}_{if},$$

for a boundary face with center x_f, inner cell i, outer normal vector $\vec{n}_{ij}$ and distance between face and cell centers d_{if}

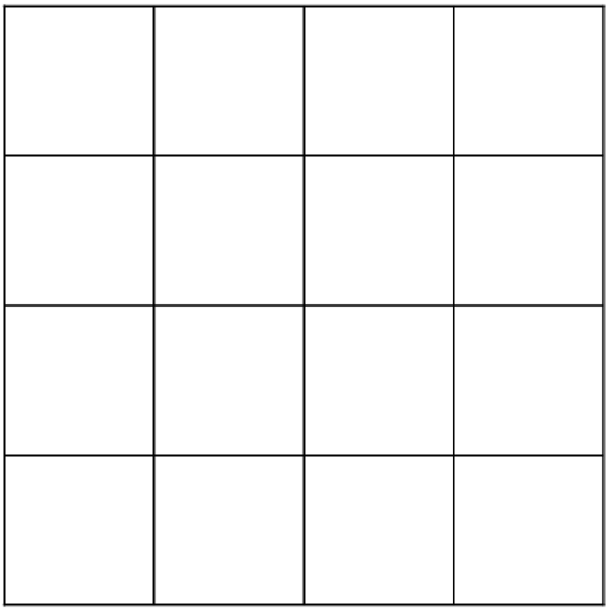
```
In [18]: display(Markdown(CodeFV5PoissonProblem))
```

1.3 The script

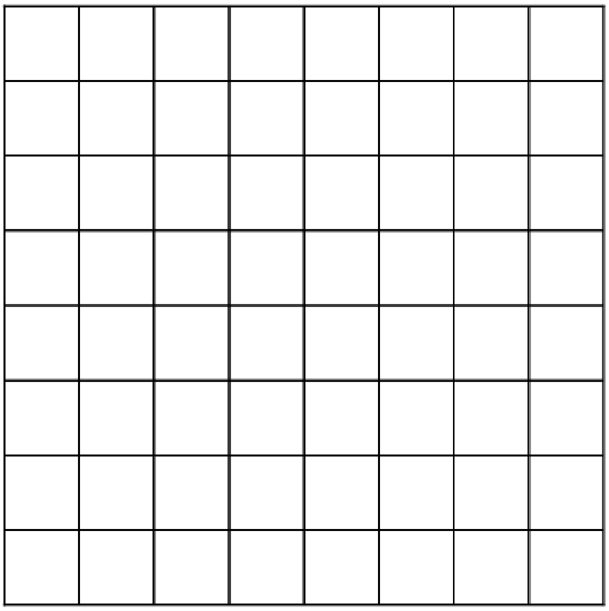
```
#Discrétisation du second membre et extraction du nb max de voisins d'une cellule
#================================================================================
my_RHSfield = cdmath.Field("RHS_field", cdmath.CELLS, my_mesh, 1)
maxNbNeighbours=0#This is to determine the number of non zero coefficients in the 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)
normal=cdmath.Vector(dim)
#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))
                    for idim in range(dim) :
                        normal[idim] = Ci.getNormalVector(j, idim);#normale sortante
        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*(normal[0]*normal[0] + K*normal[1]*no
            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")
```

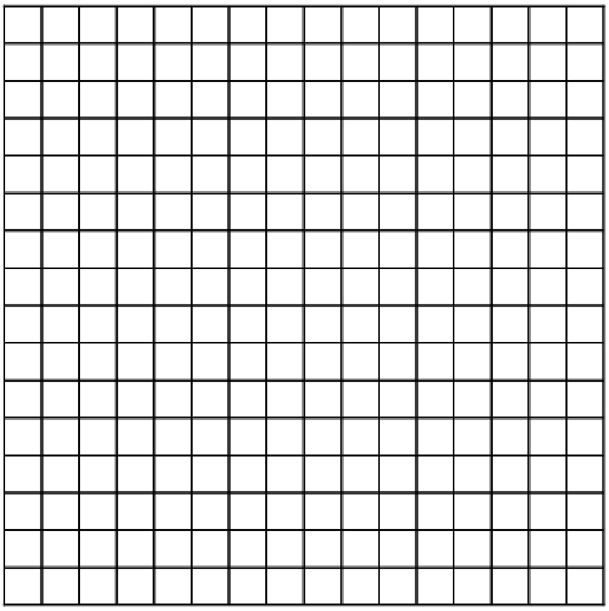
1.4 Regular grid

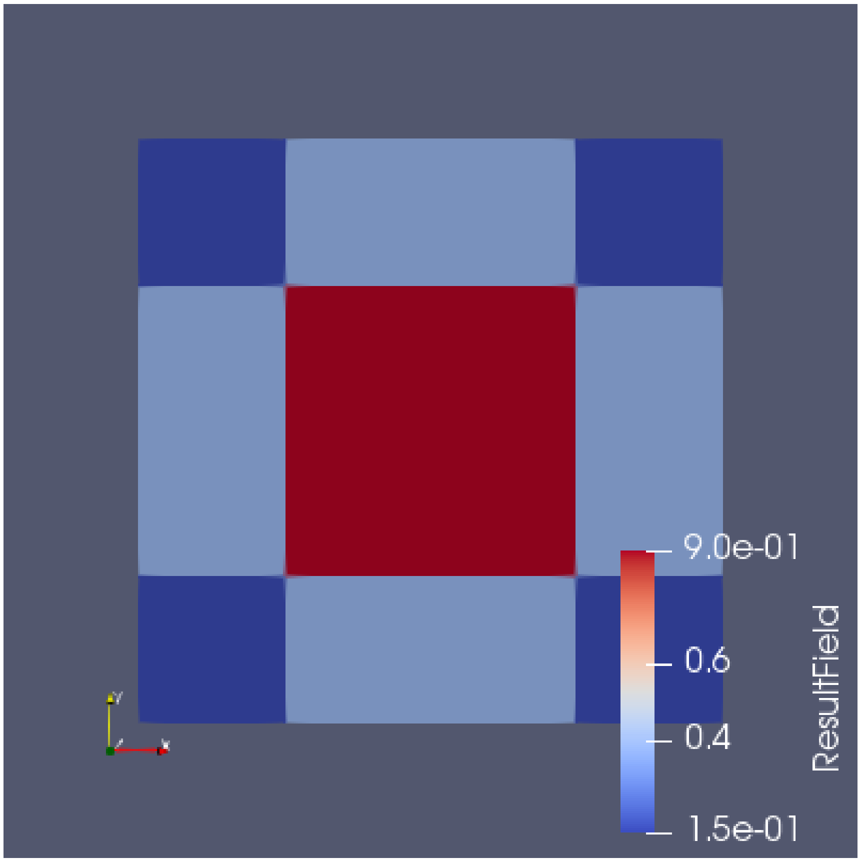


mesh 1 | mesh 2 | mesh 3 - | - - | -

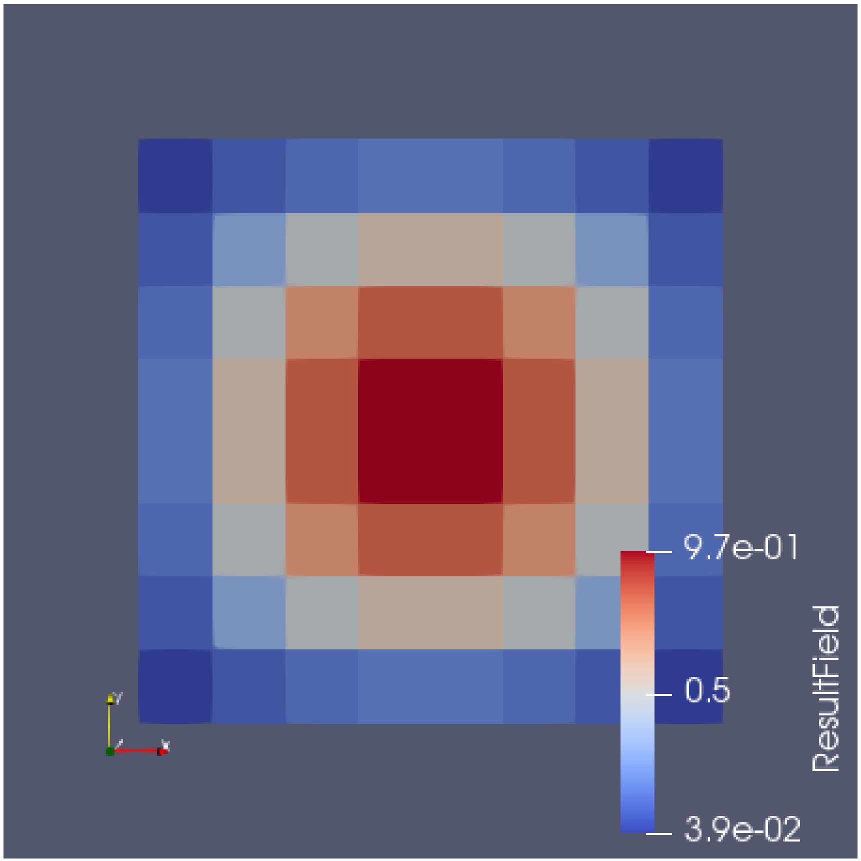


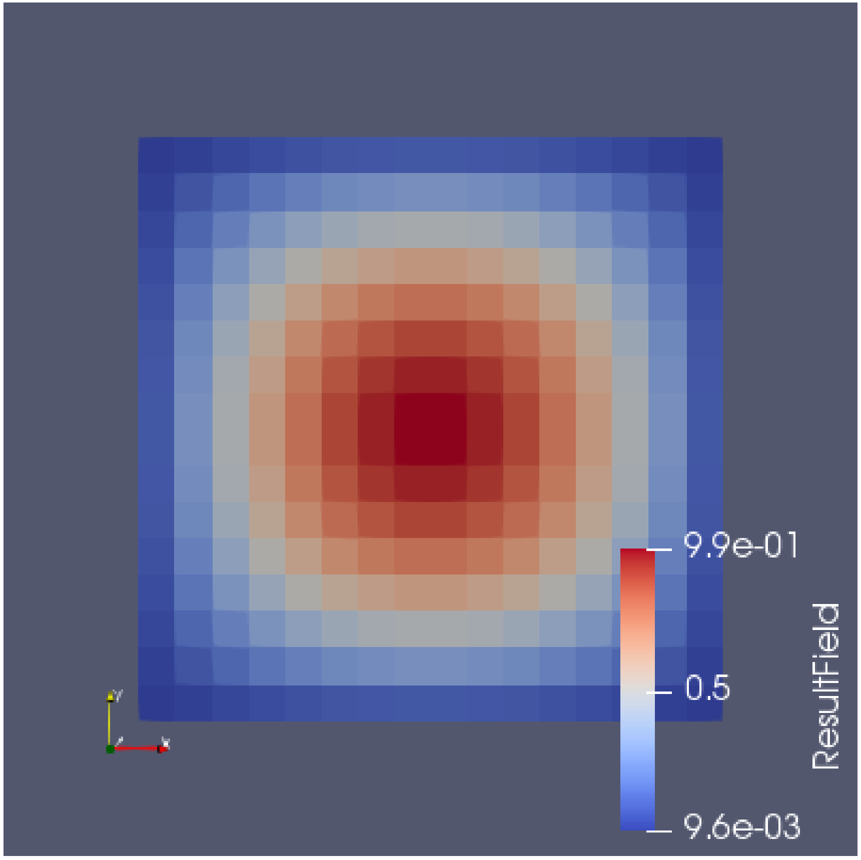
| |

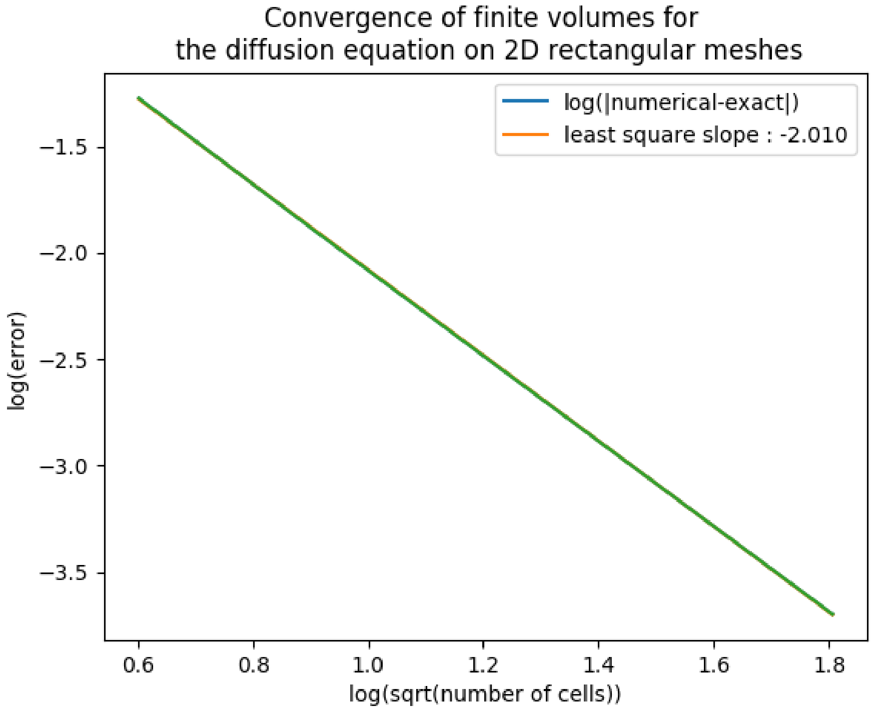




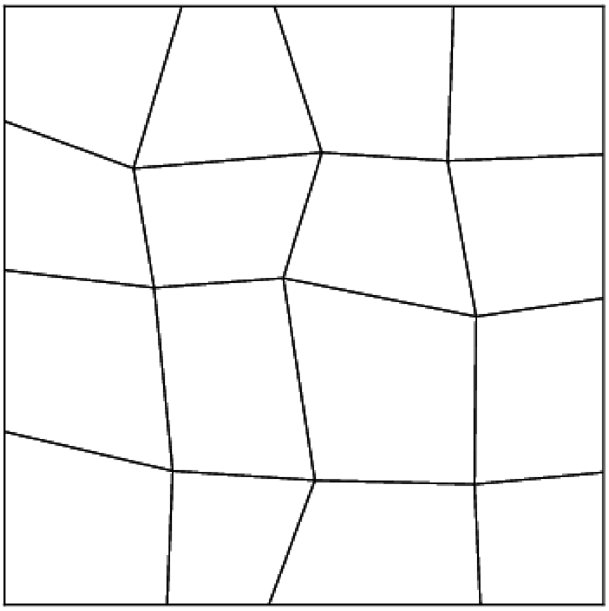
result 1 | result 2 | result 3 - | -- | -


9.7e-01
0.5
3.9e-02
ResultField
Y
X

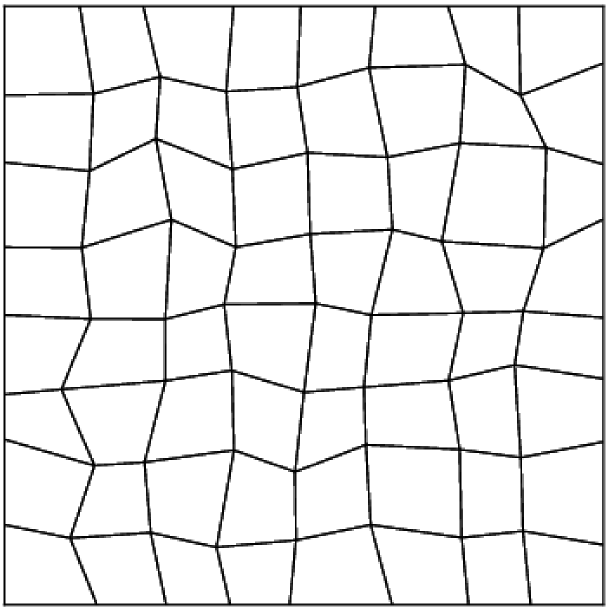

9.9e-01
0.5
9.6e-03
ResultField
Y
X


Convergence of finite volumes for
the diffusion equation 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))

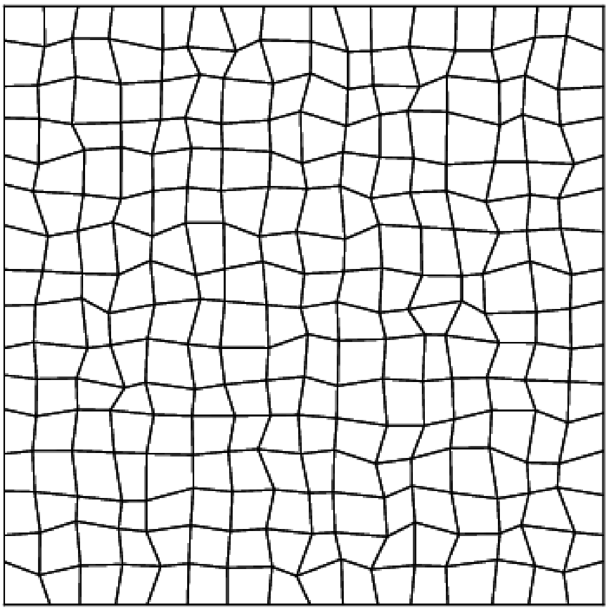
1.5 Deformed quadrangles



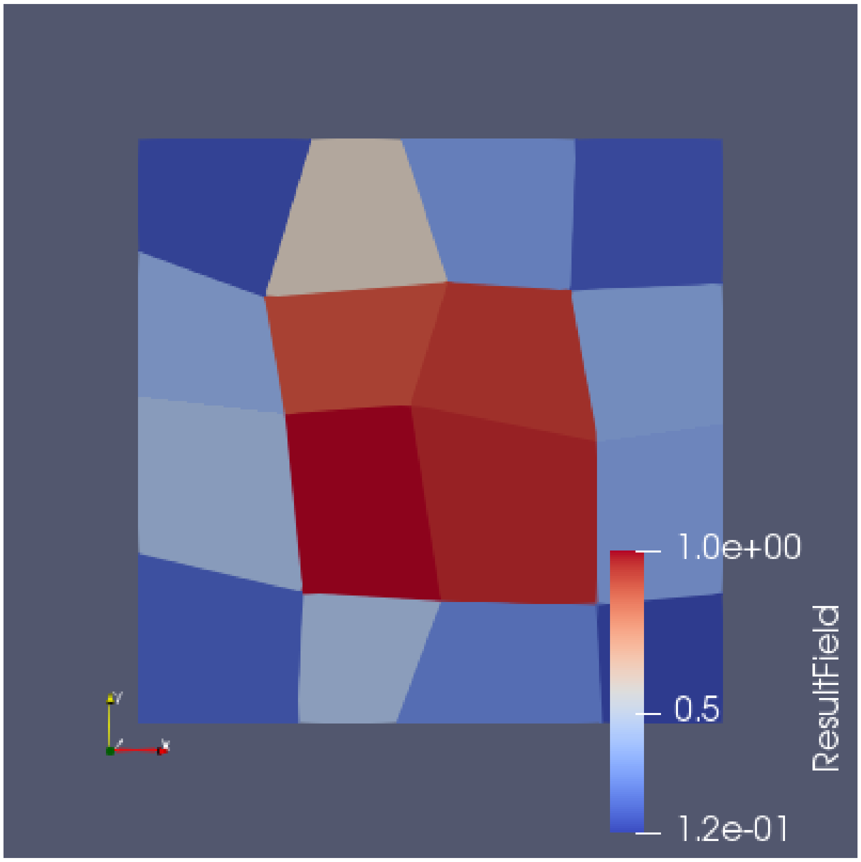
mesh 1 | mesh 2 | mesh 3 - | - - | -

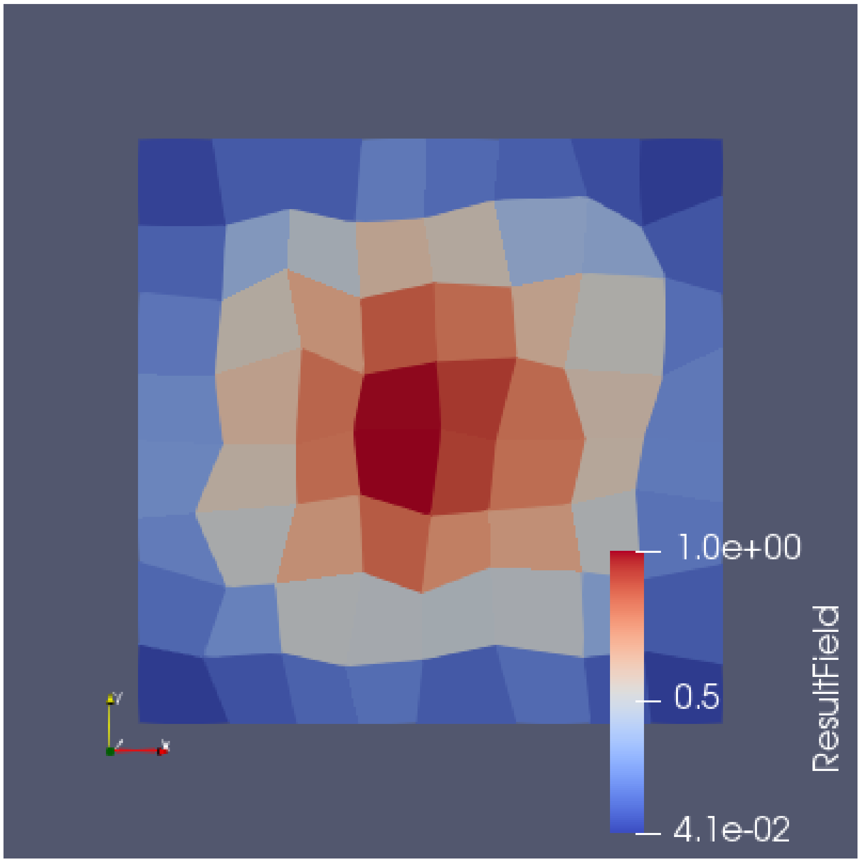


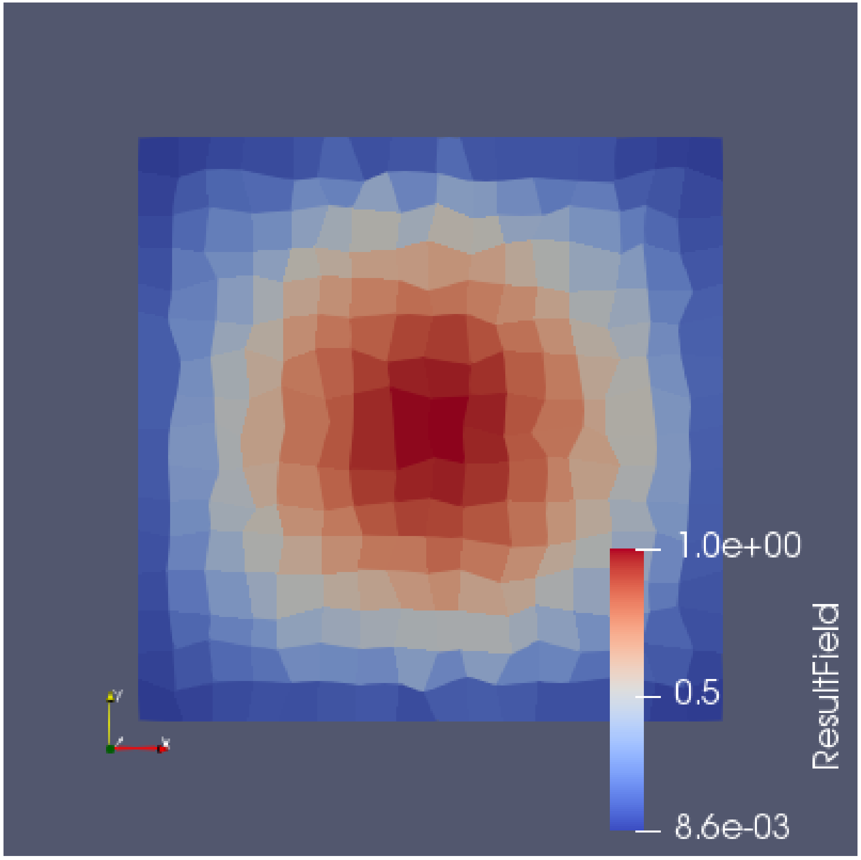
| |

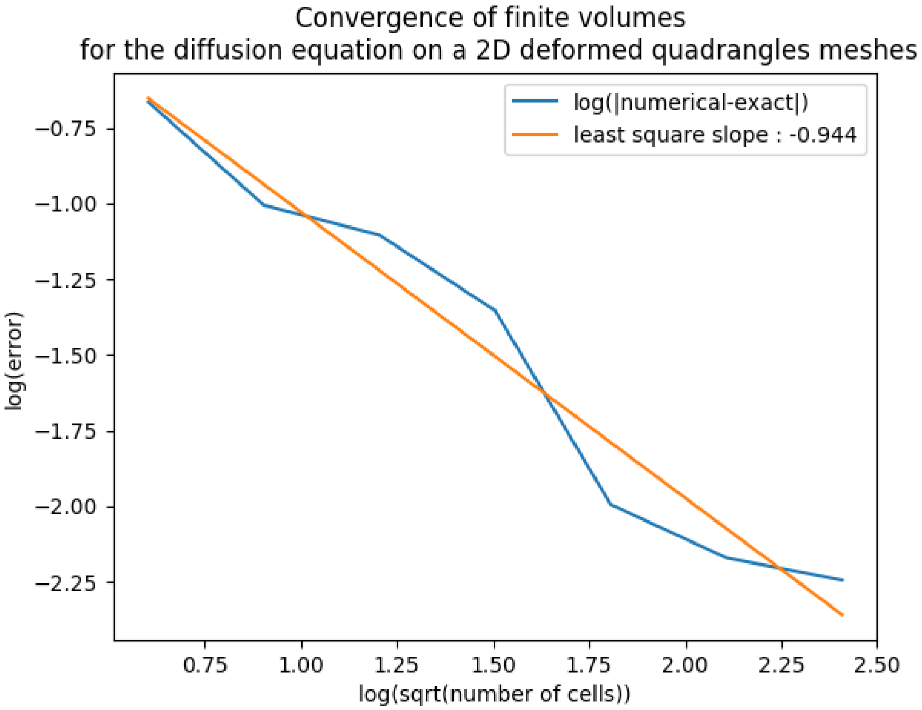


result 1 | result 2 | result 3 - | -- | -

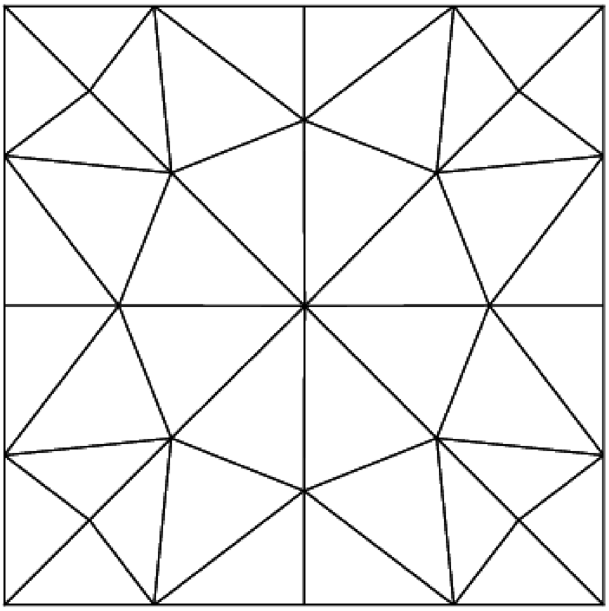



1.0e+00
0.5
4.1e-02
ResultField
Y
X

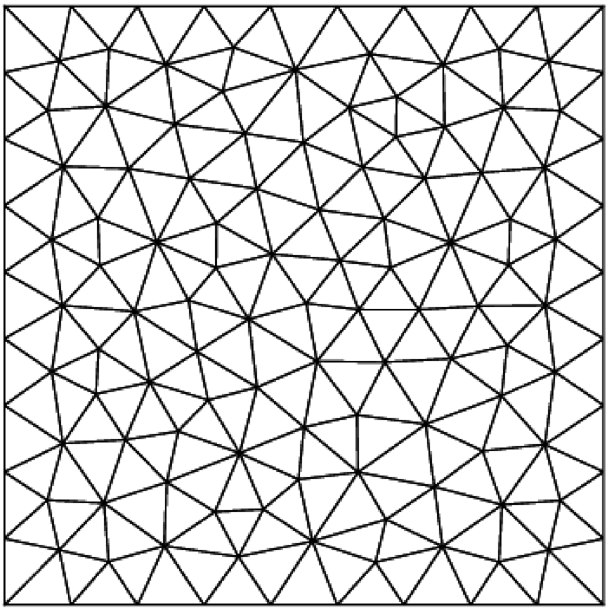

1.0e+00
0.5
8.6e-03
ResultField
Y
X


Convergence of finite volumes
for the diffusion equation on a 2D deformed quadrangles meshes
log(|numerical-exact|)
least square slope : -0.944
log(error)
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
−2.25
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50
log(sqrt(number of cells))

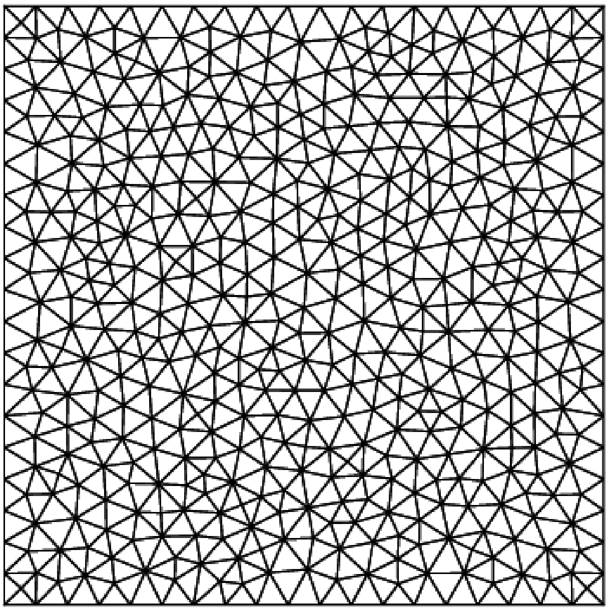
1.6 Delaunay triangular meshes

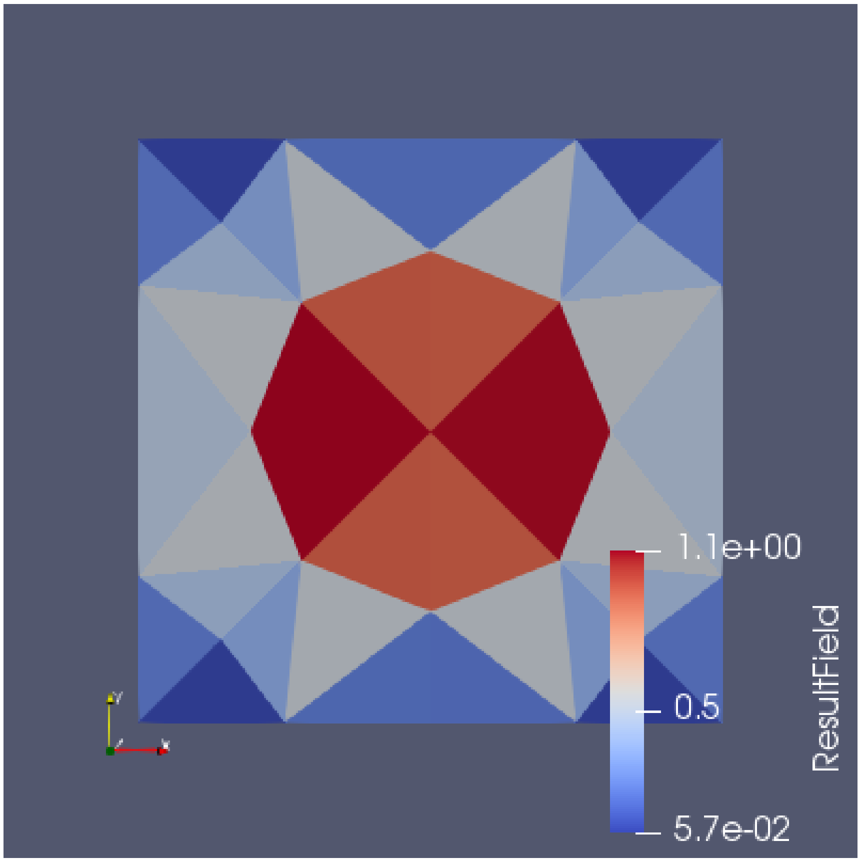


mesh 1 | mesh 2 | mesh 3 - | - - | -

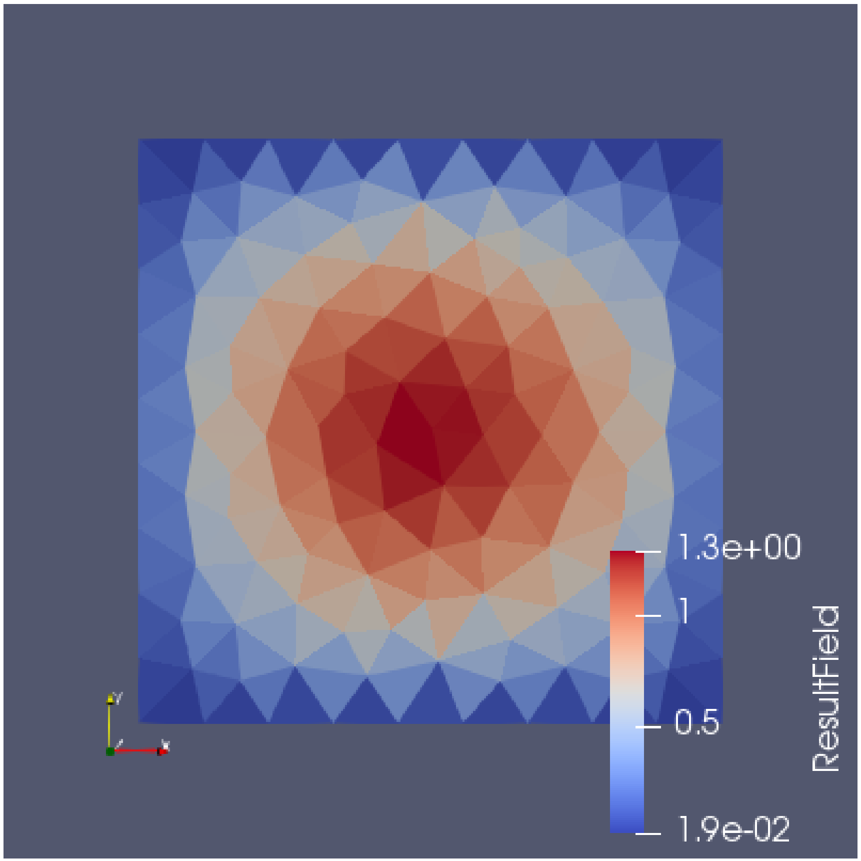


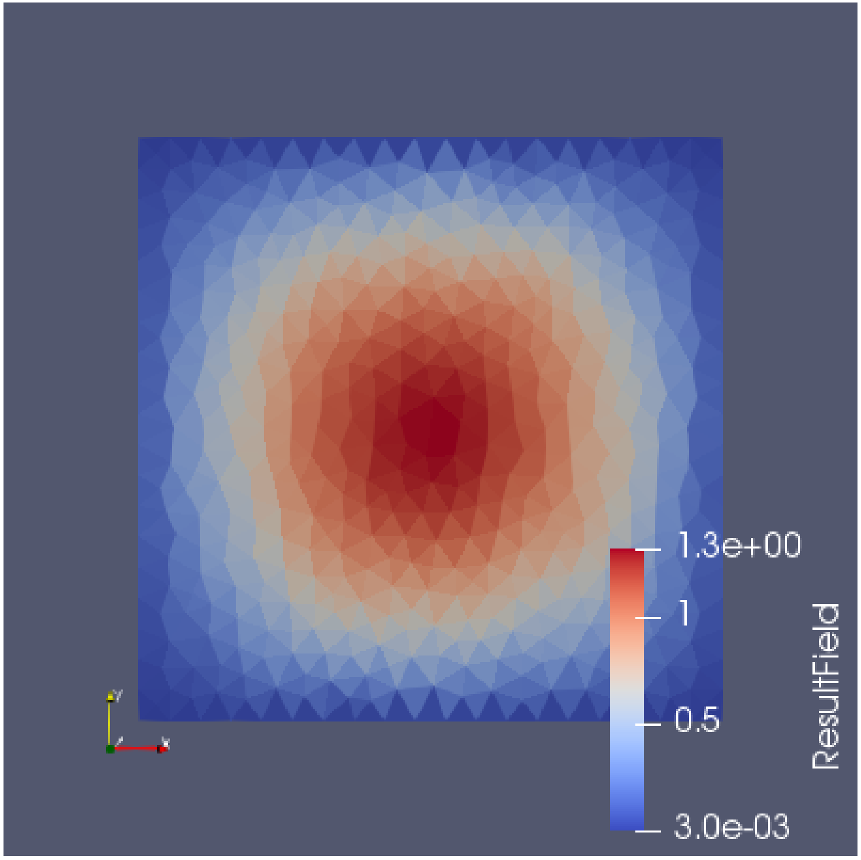
| |

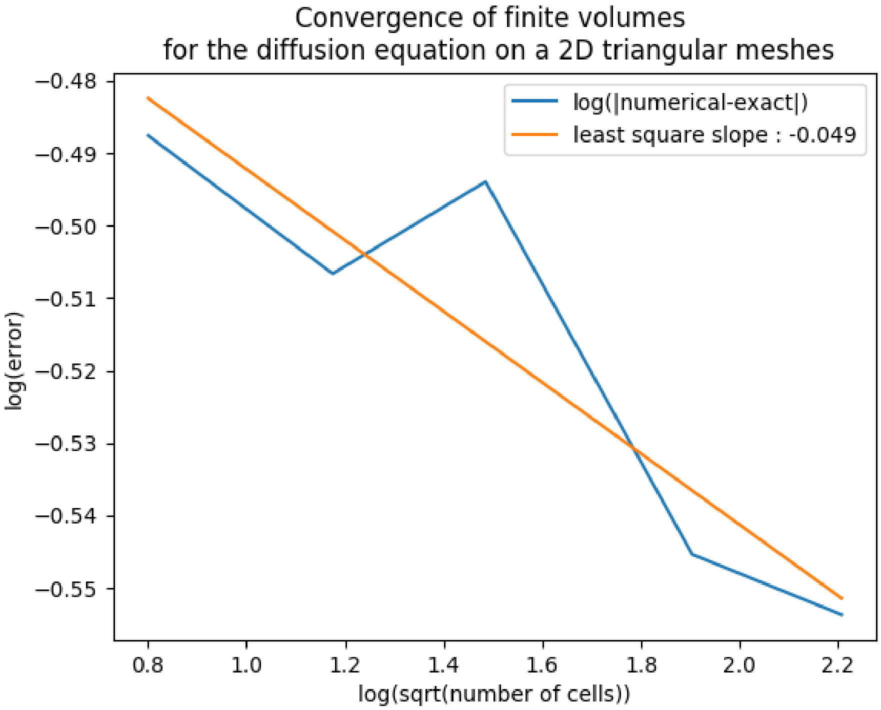




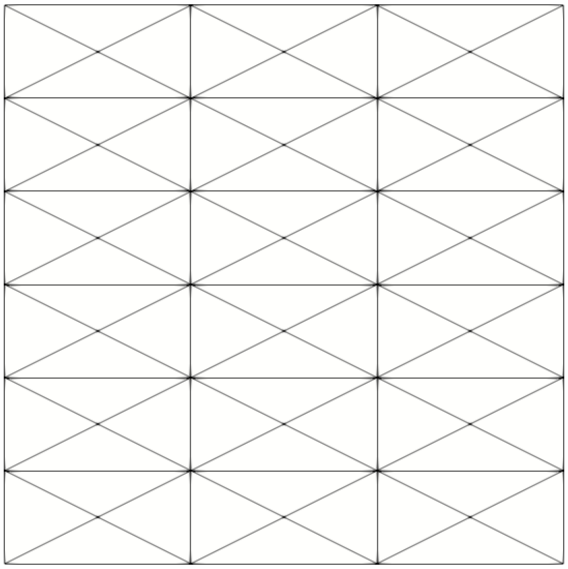
result 1 | result 2 | result 3 - | - - | -


1.3e+00
1
0.5
1.9e-02
ResultField
y
x

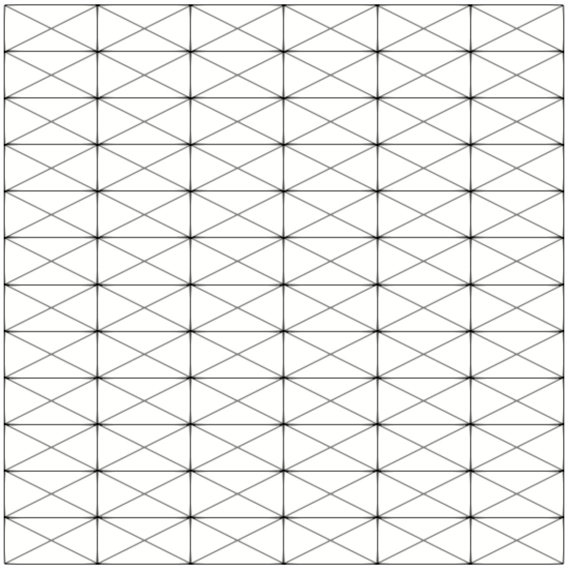

1.3e+00
1
0.5
3.0e-03
ResultField
y
x


Convergence of finite volumes
for the diffusion equation on a 2D triangular meshes
log(|numerical-exact|)
least square slope : -0.049
log(error)
−0.48
−0.49
−0.50
−0.51
−0.52
−0.53
−0.54
−0.55
0.8
1.0
1.2
1.4
1.6
1.8
2.0
2.2
log(sqrt(number of cells))

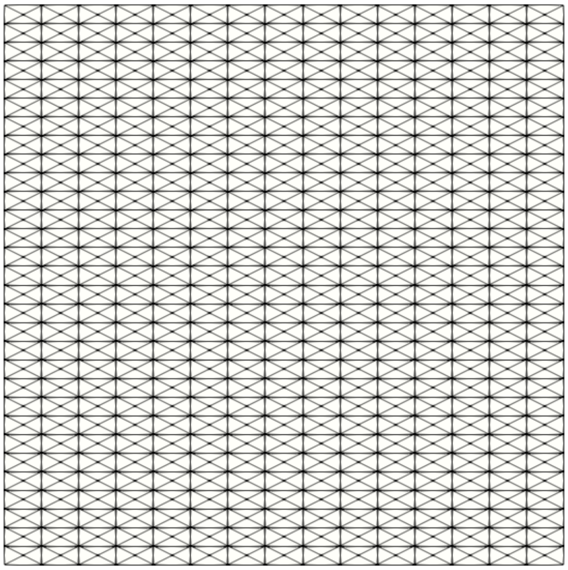
1.7 Cross triangle meshes (from a $(n, 2n)$ rectangular grid)



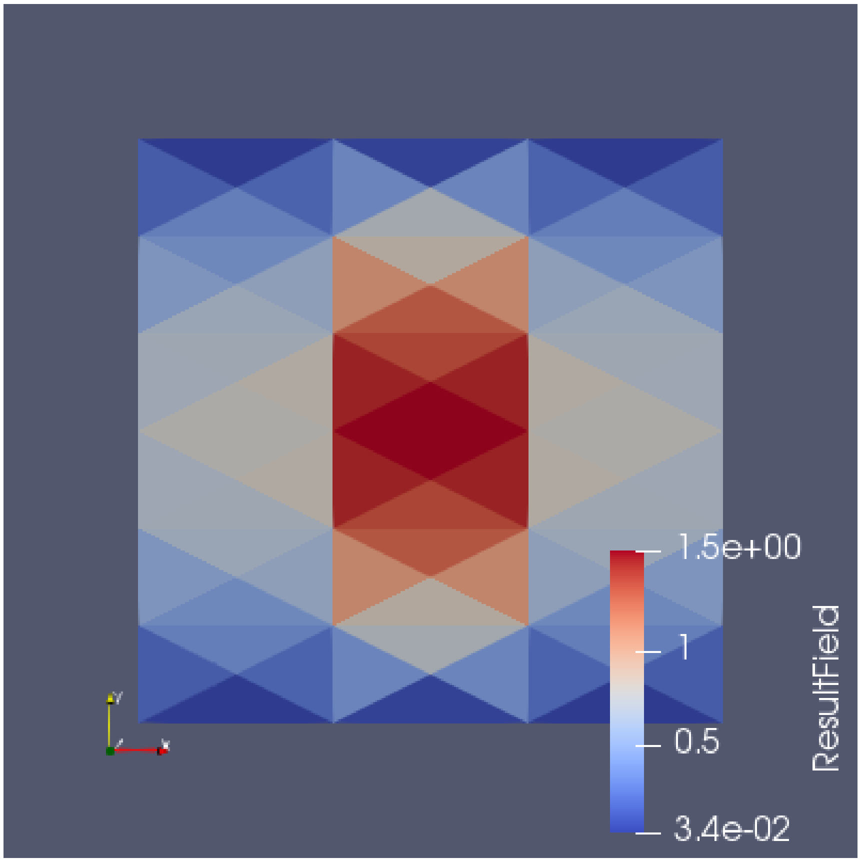
mesh 1 | mesh 2 | mesh 3 - | - - | -

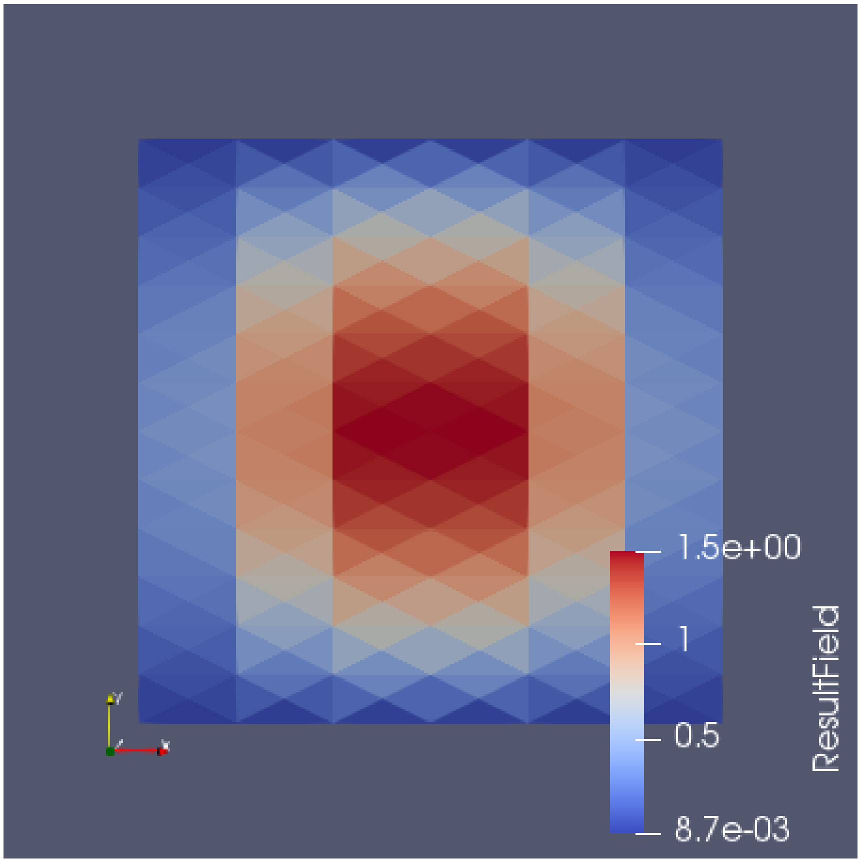


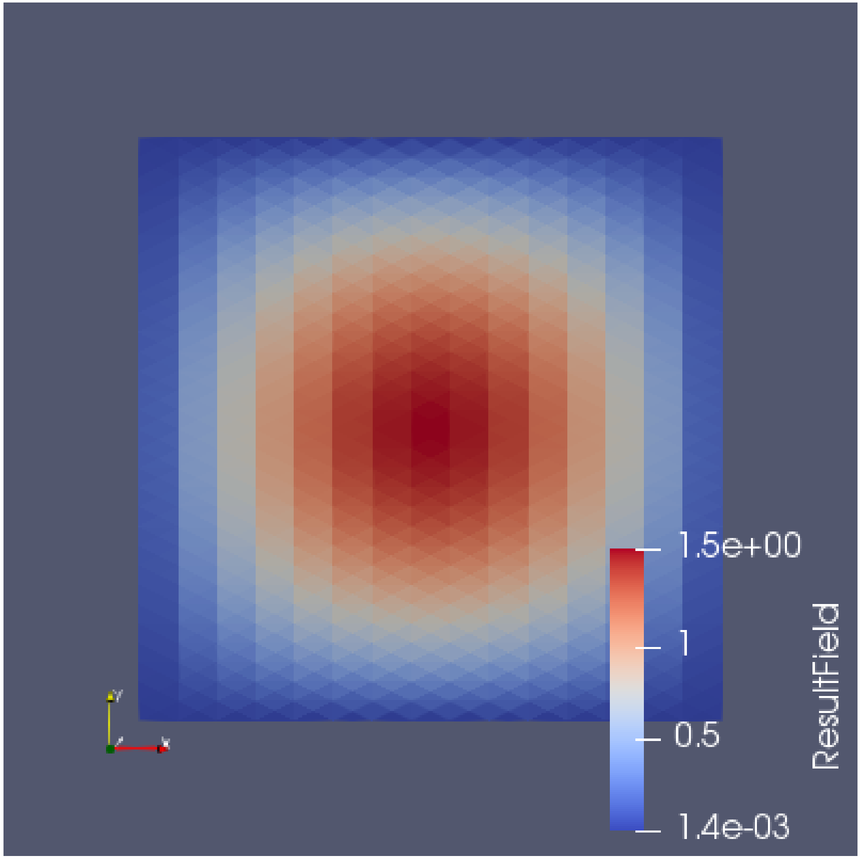
| |

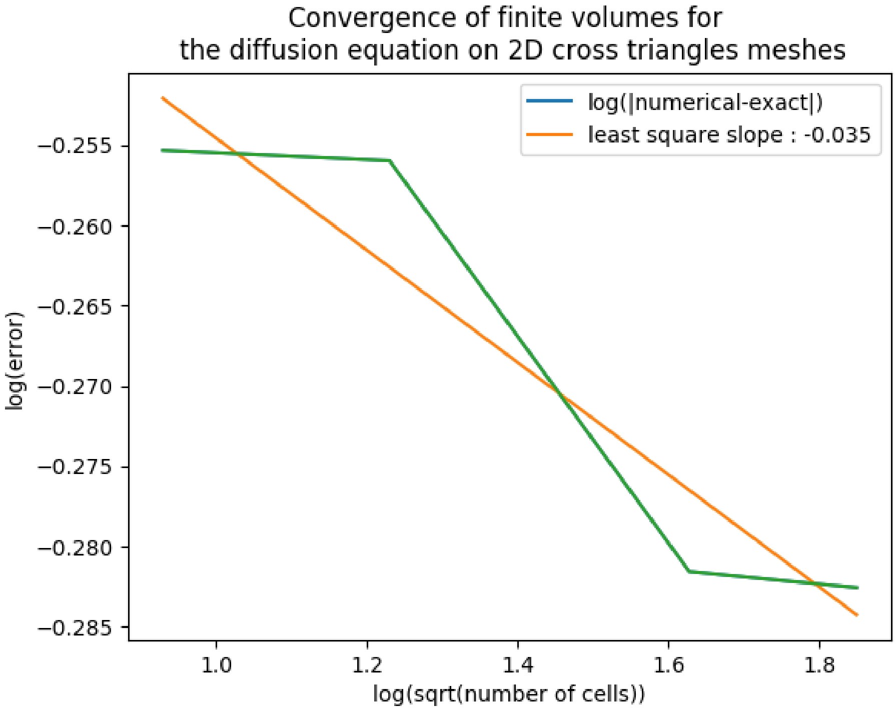


result 1 | result 2 | result 3 - | -- | -

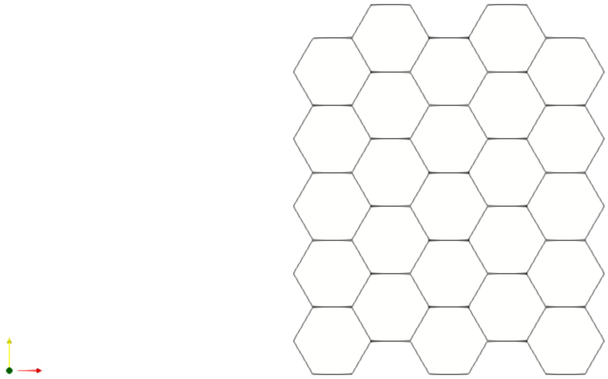



1.5e+00
1
0.5
8.7e-03
ResultField
Y
X

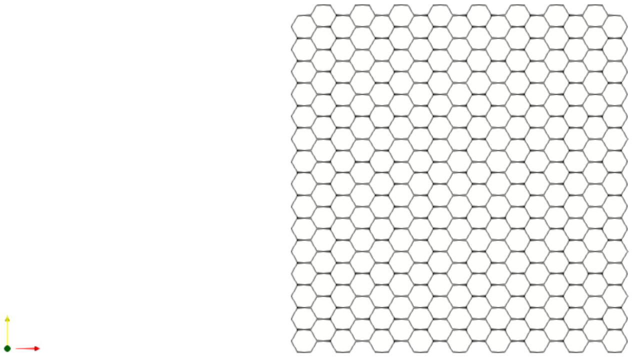

1.5e+00
1
0.5
1.4e-03
ResultField
Y
X


Convergence of finite volumes for
the diffusion equation on 2D cross triangles meshes
log(|numerical-exact|)
least square slope : -0.035
log(error)
−0.255
−0.260
−0.265
−0.270
−0.275
−0.280
−0.285
1.0
1.2
1.4
1.6
1.8
log(sqrt(number of cells))

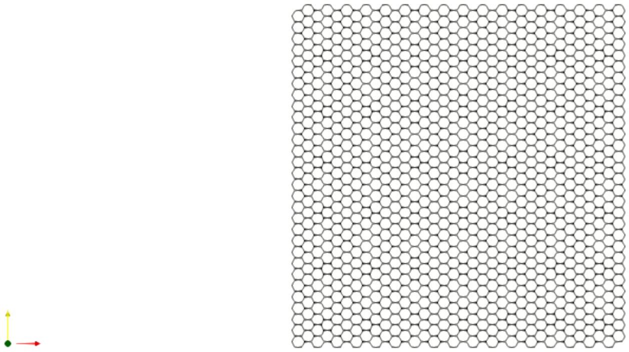
1.8 Hexagonal meshes

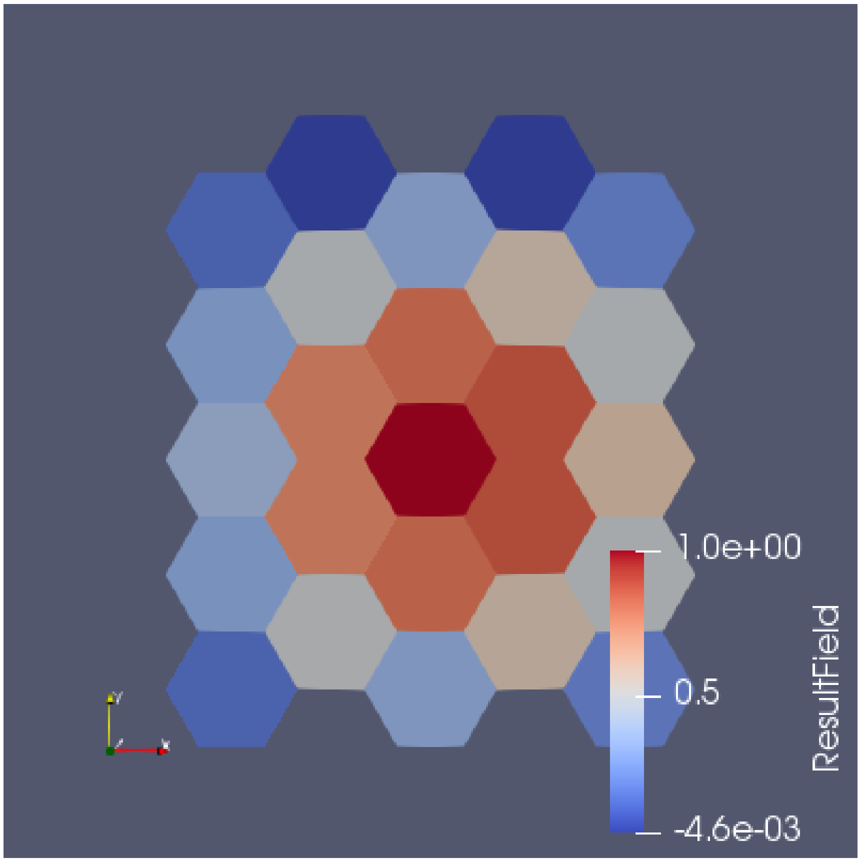


mesh 1 | mesh 2 | mesh 3 - | - - | -

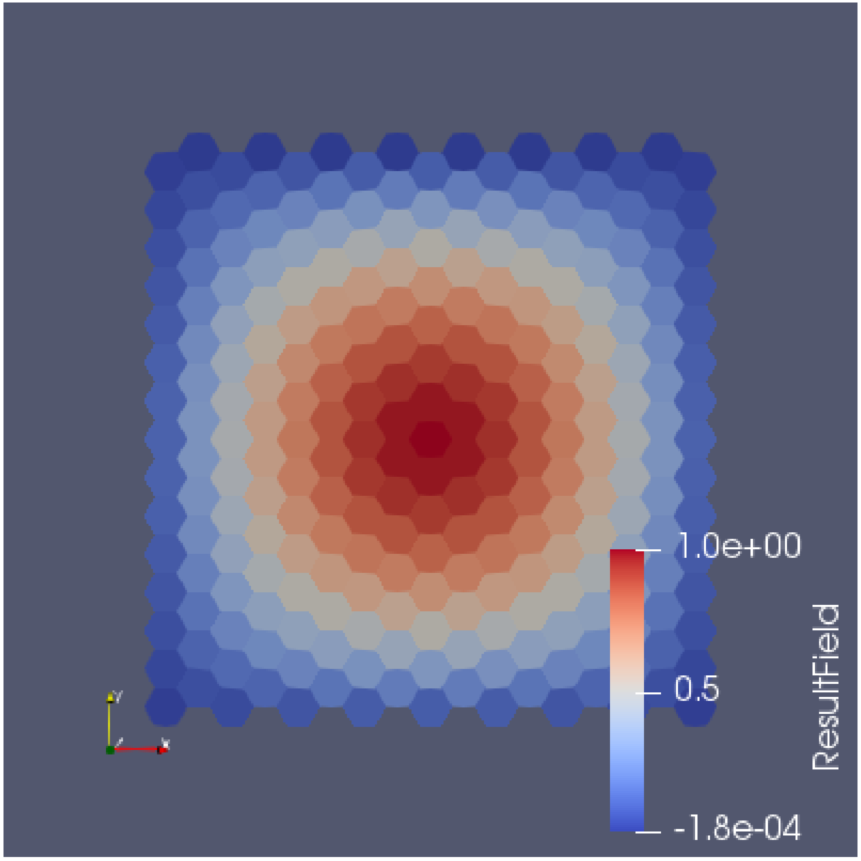


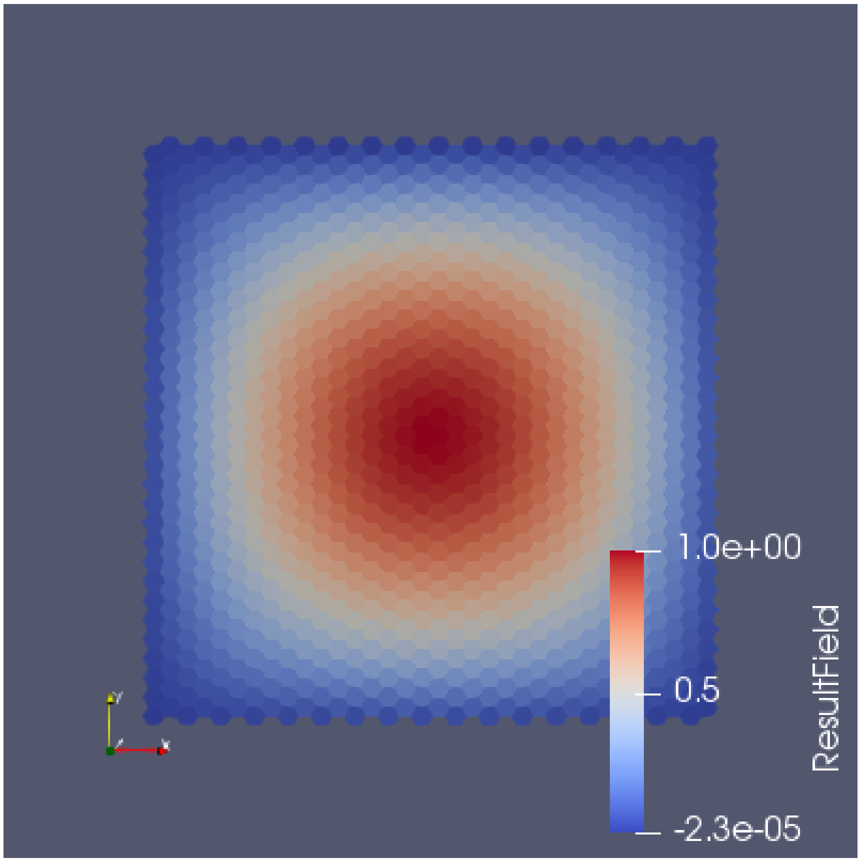
| |

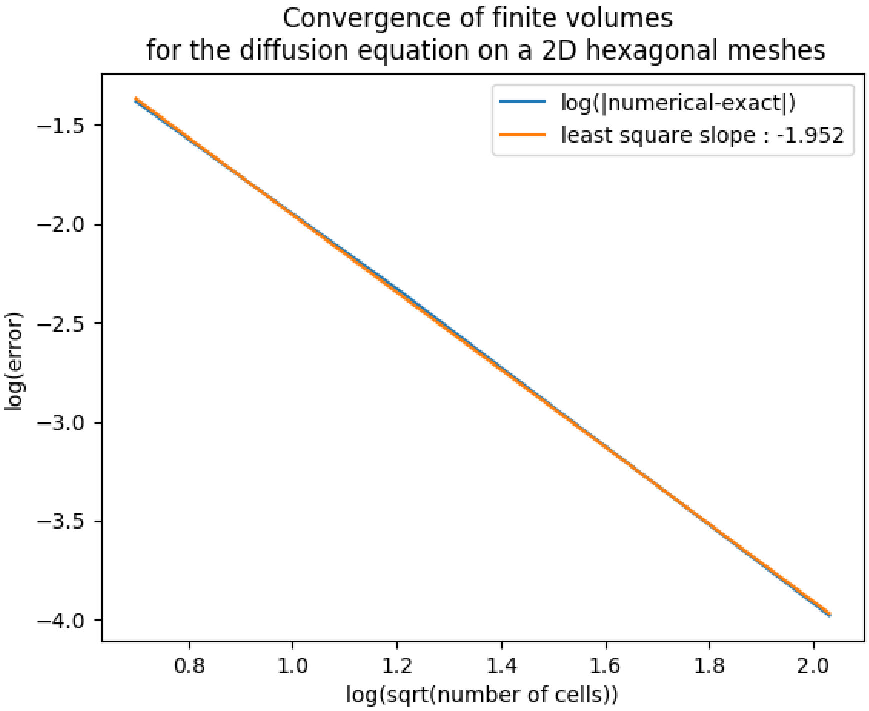




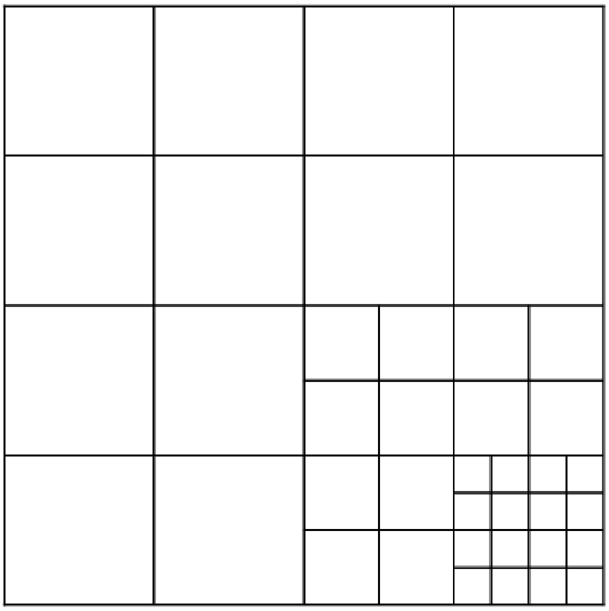
result 1 | result 2 | result 3 - | - - | -

|  |

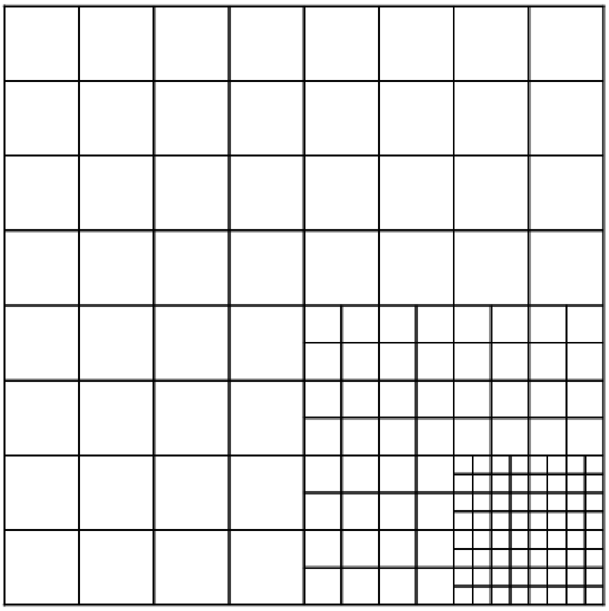

1.0e+00
0.5
-2.3e-05
ResultField
Y
X


Convergence of finite volumes
for the diffusion equation on a 2D hexagonal meshes
log(|numerical-exact|)
least square slope : -1.952
−1.5
−2.0
−2.5
−3.0
−3.5
−4.0
log(error)
0.8
1.0
1.2
1.4
1.6
1.8
2.0
log(sqrt(number of cells))

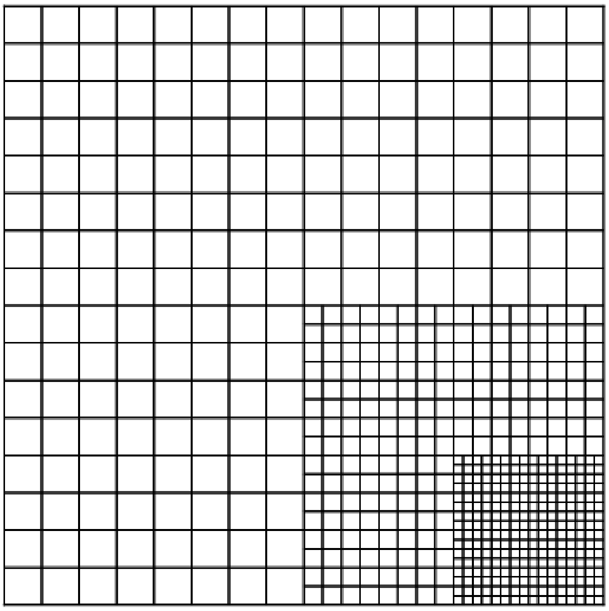
1.9 Locally refined meshes

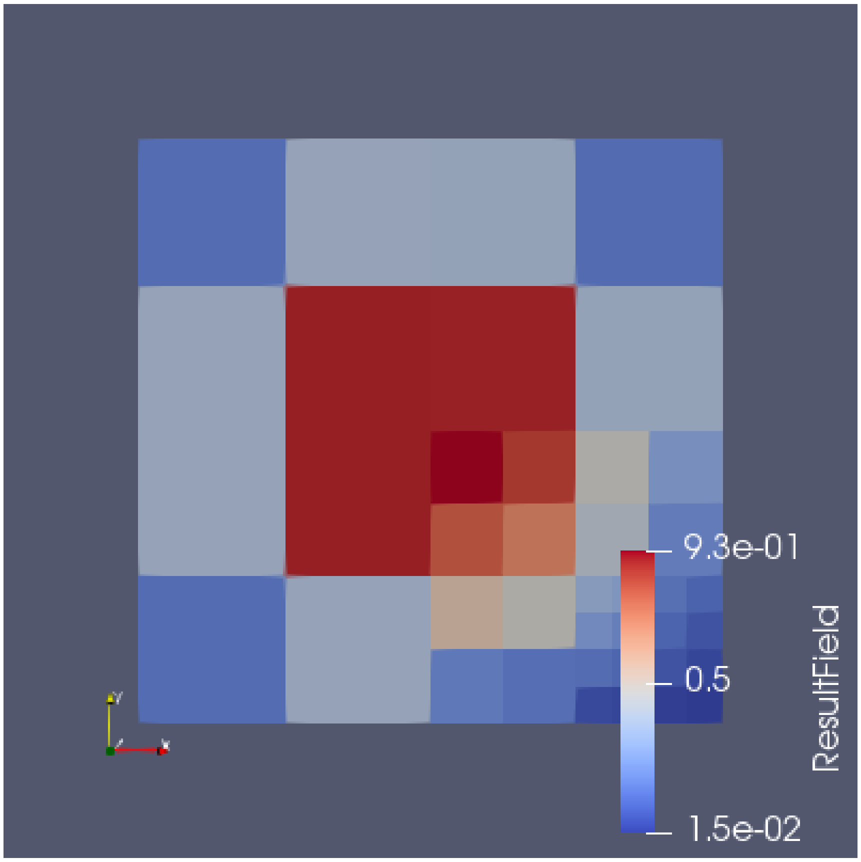


mesh 1 | mesh 2 | mesh 3 - | - - | -

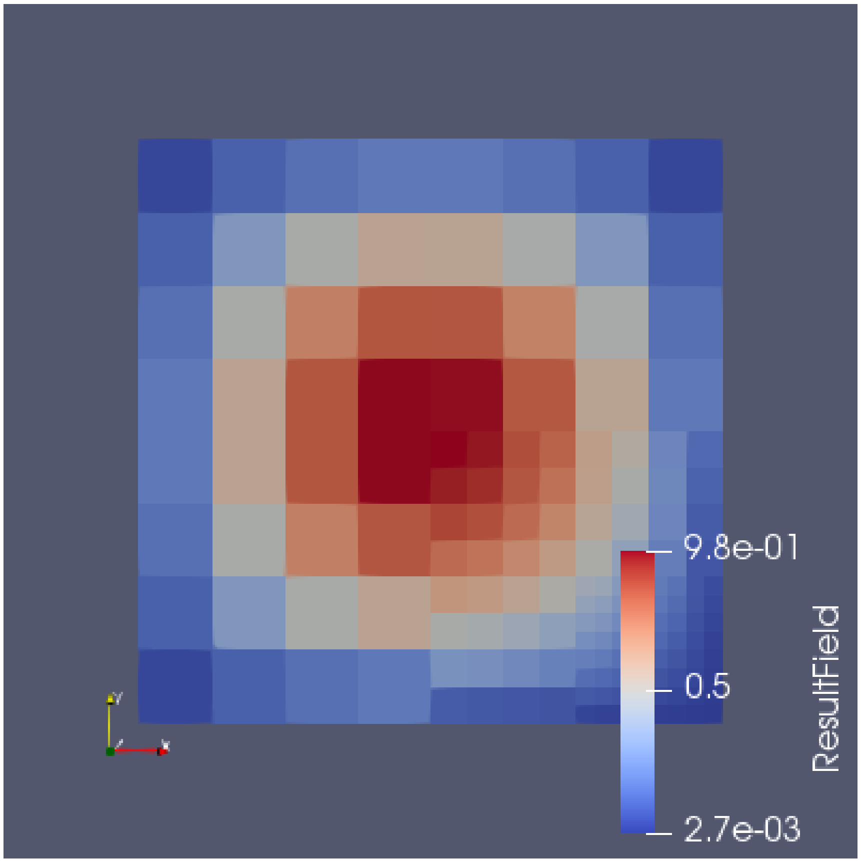


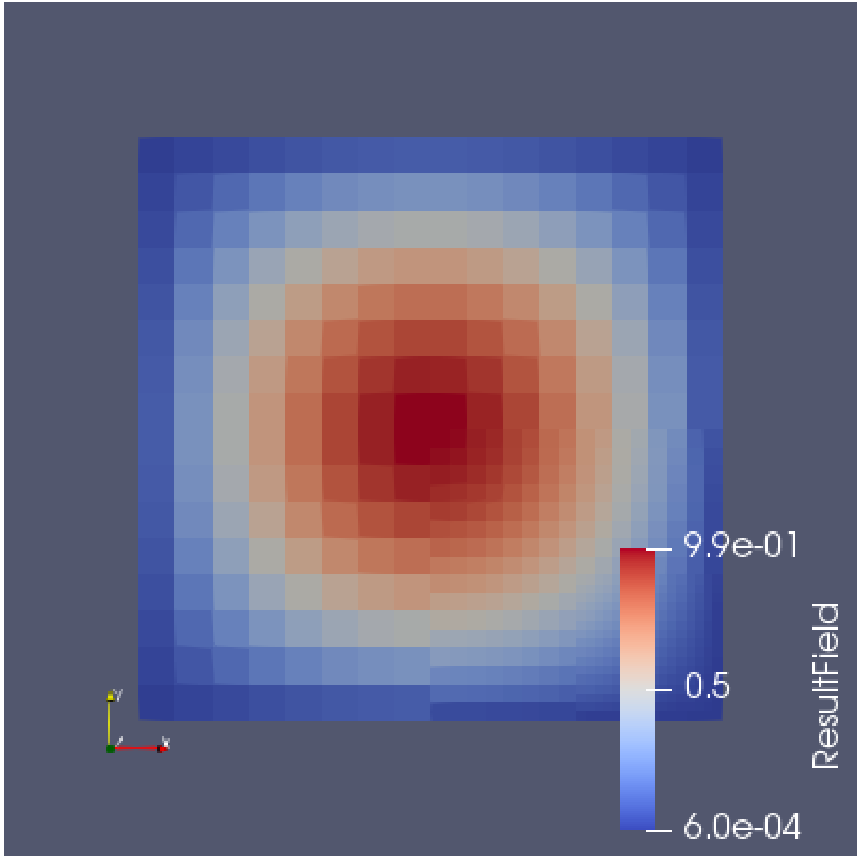
| |

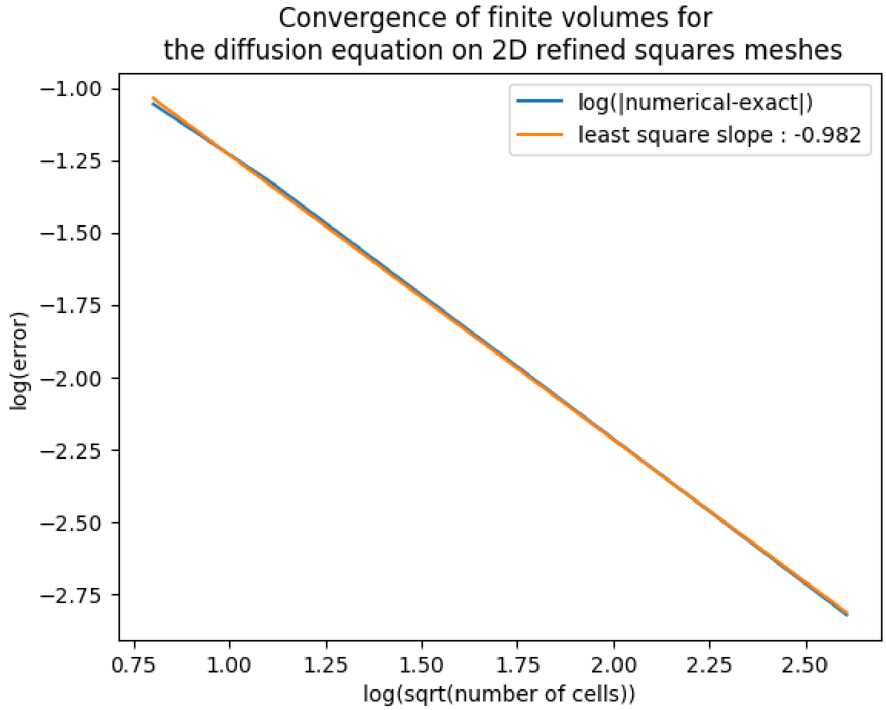




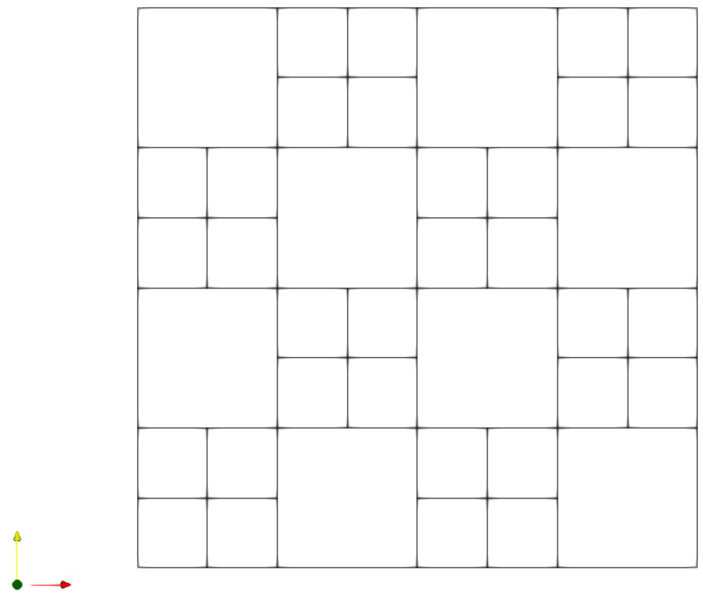
result 1 | result 2 | result 3 - | -- | -


9.8e-01
0.5
2.7e-03
ResultField
Y
X

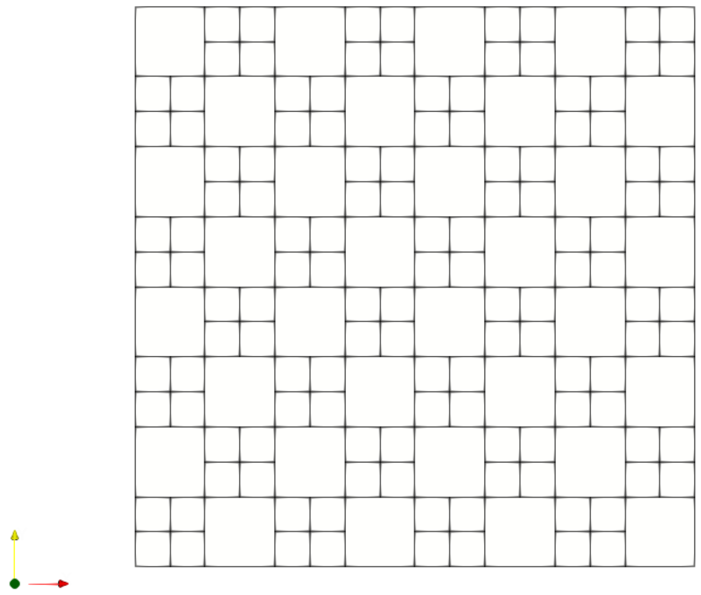

9.9e-01
0.5
6.0e-04
ResultField
Y
X


Convergence of finite volumes for
the diffusion equation on 2D refined squares meshes
log(|numerical-exact|)
least square slope : -0.982
log(error)
−1.00
−1.25
−1.50
−1.75
−2.00
−2.25
−2.50
−2.75
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50
log(sqrt(number of cells))

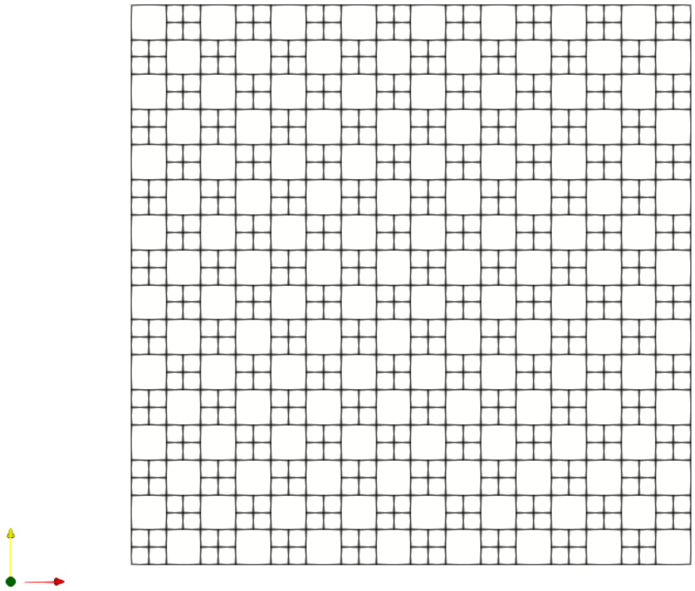
1.10 Checkerboard meshes

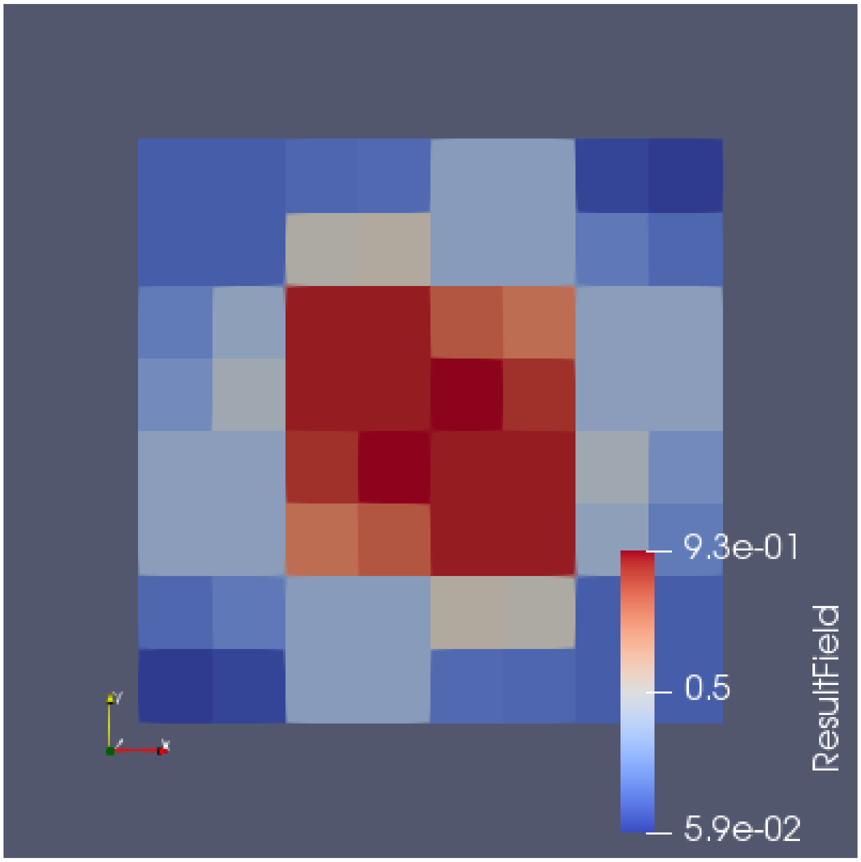


mesh 1 | mesh 2 | mesh 3 - | - - | -

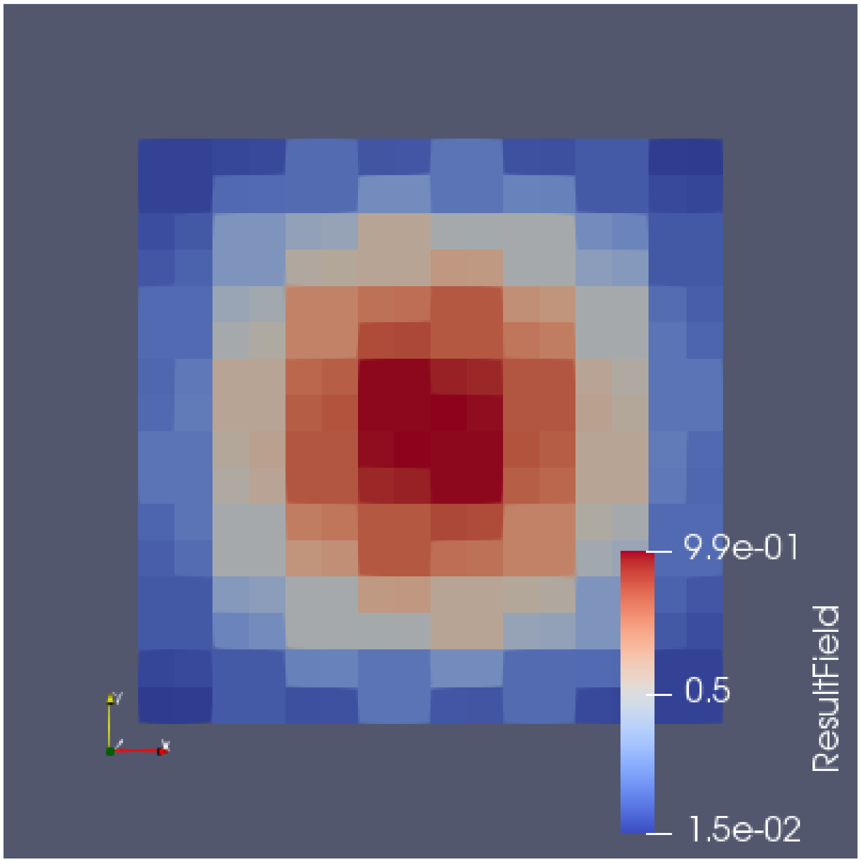


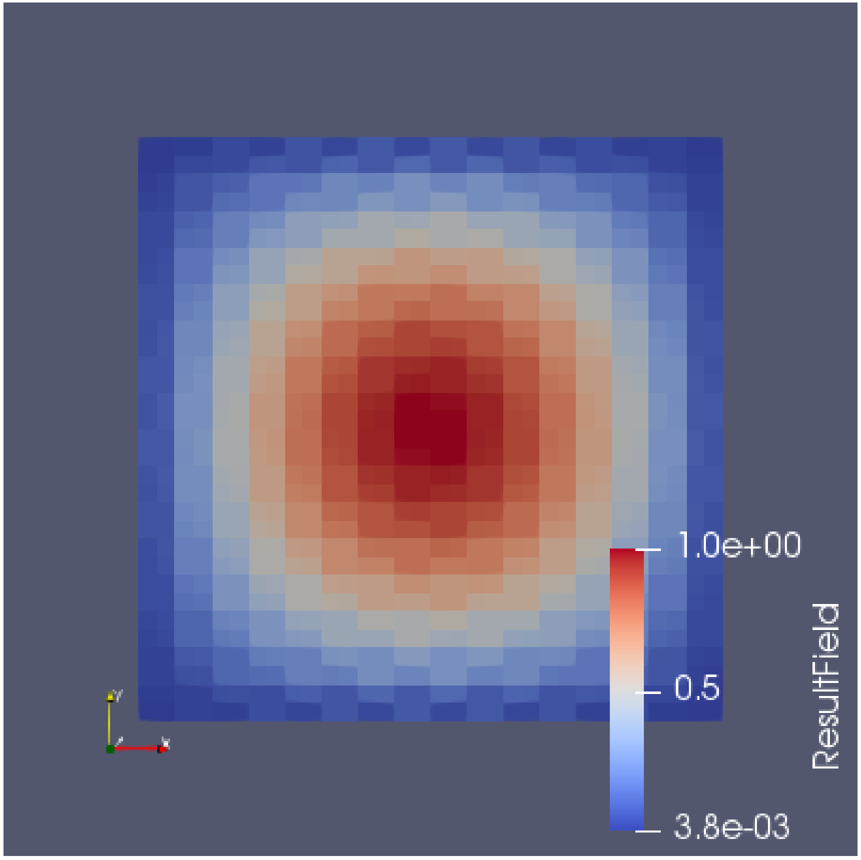
| |

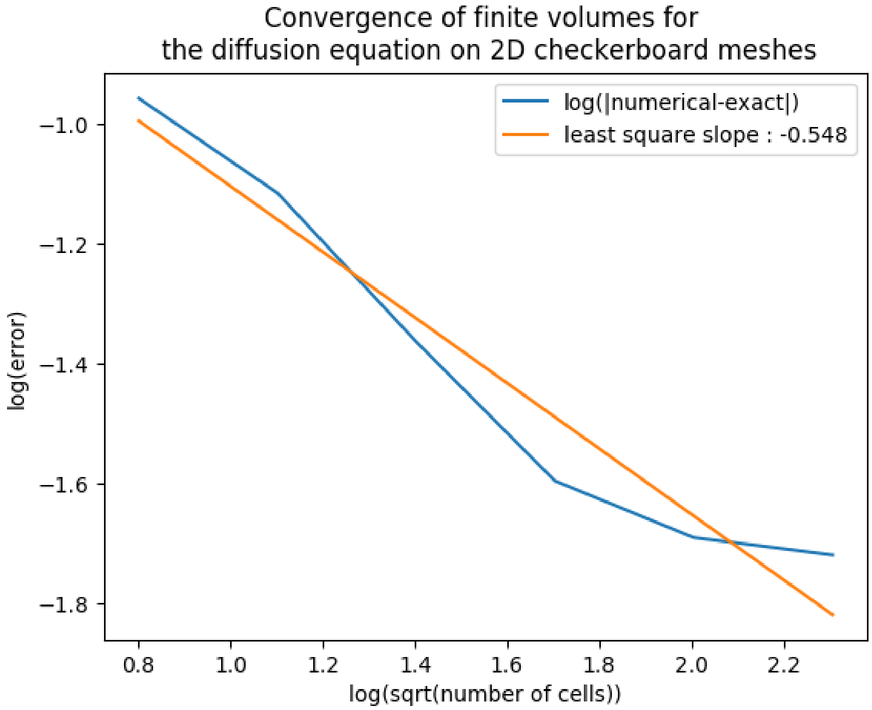




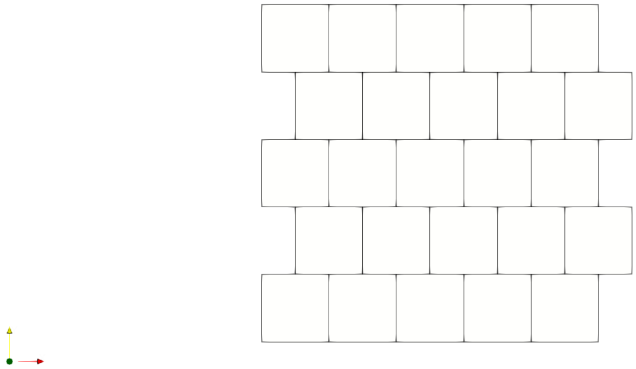
result 1 | result 2 | result 3 - | - - | -


y
x
9.9e-01
0.5
1.5e-02
ResultField

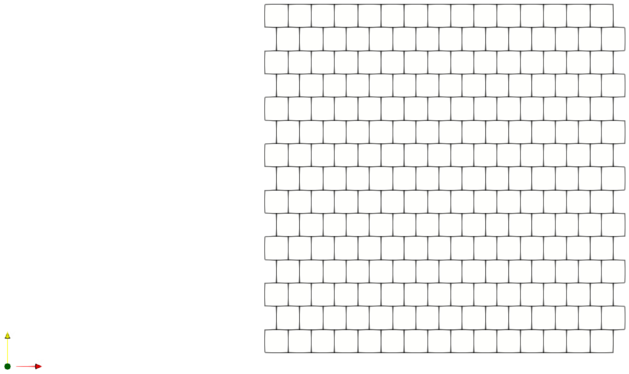

y
x
1.0e+00
0.5
3.8e-03
ResultField


Convergence of finite volumes for
the diffusion equation on 2D checkerboard meshes
log(|numerical-exact|)
least square slope : -0.548
−1.0
−1.2
−1.4
−1.6
−1.8
log(error)
0.8
1.0
1.2
1.4
1.6
1.8
2.0
2.2
log(sqrt(number of cells))

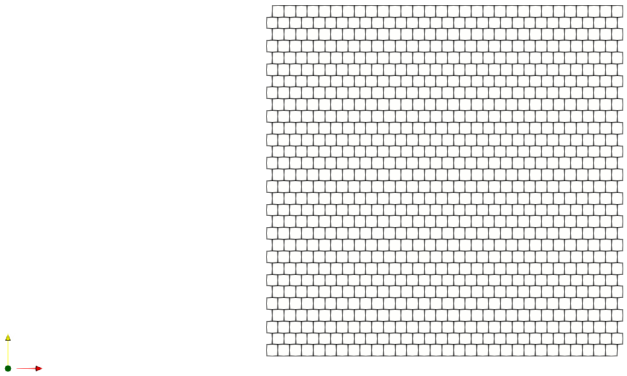
1.11 Brick wall meshes

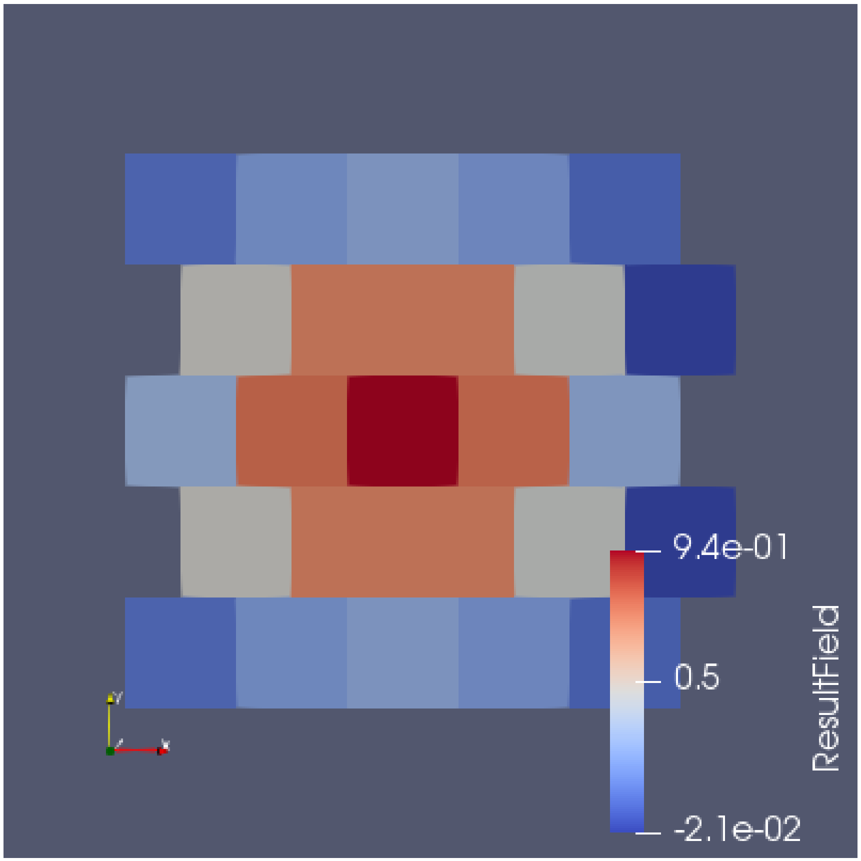


mesh 1 | mesh 2 | mesh 3 - | - - | -

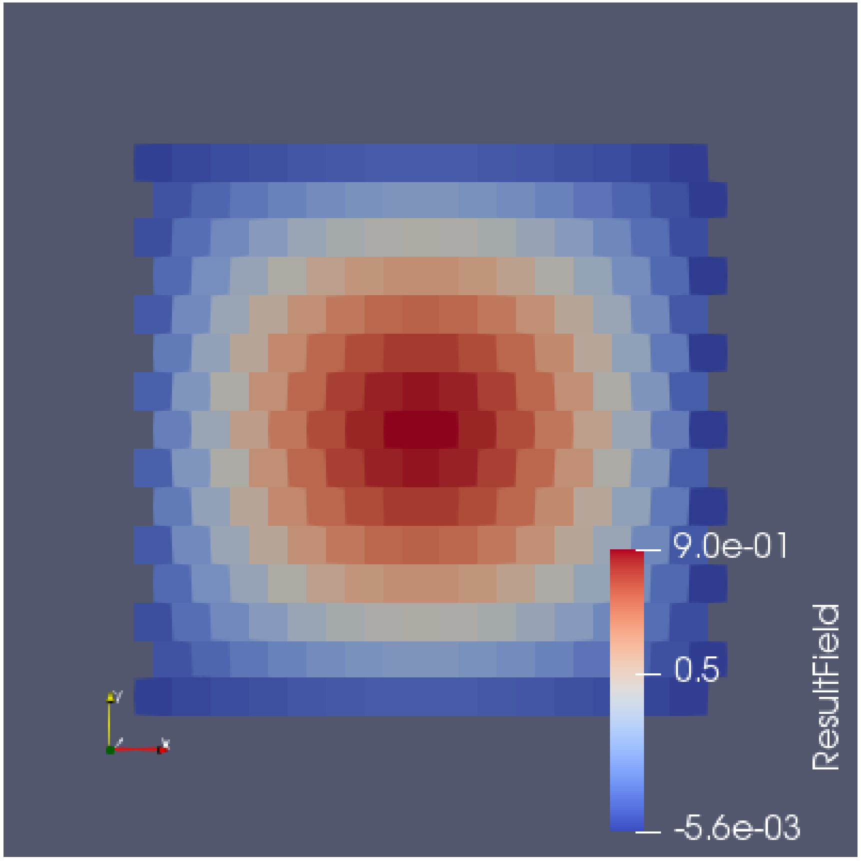


| |

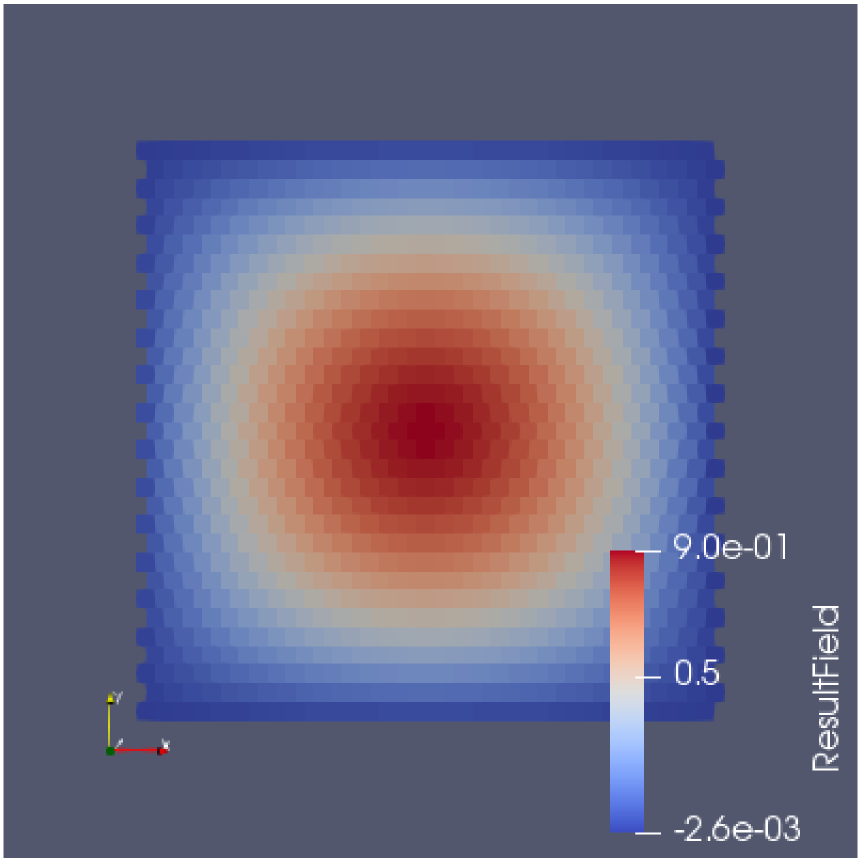




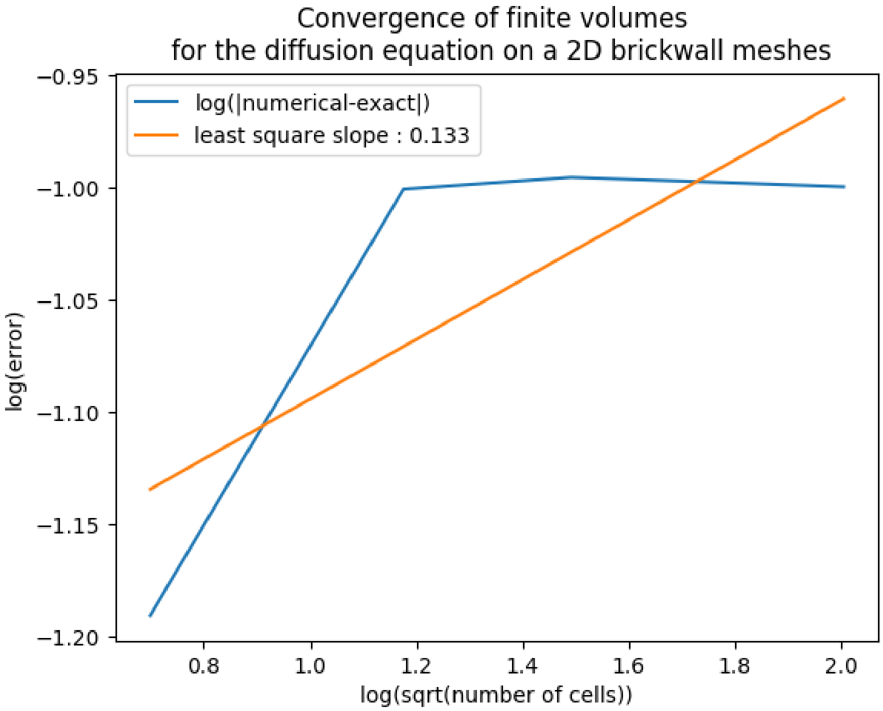
result 1 | result 2 | result 3 - | -- | -



| |

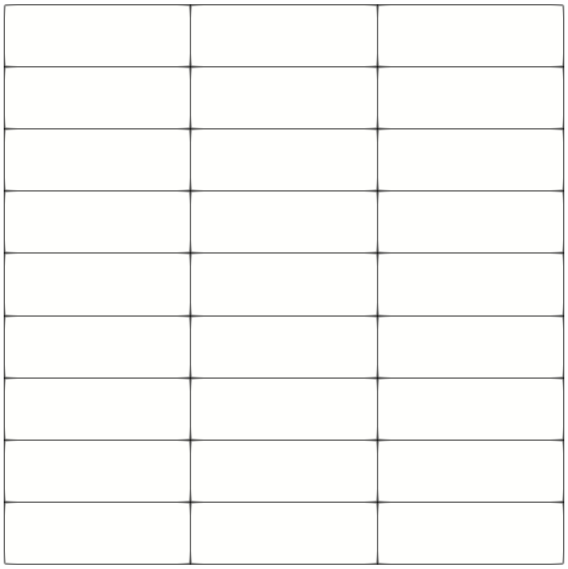


9.0e-01
0.5
-2.6e-03
ResultField

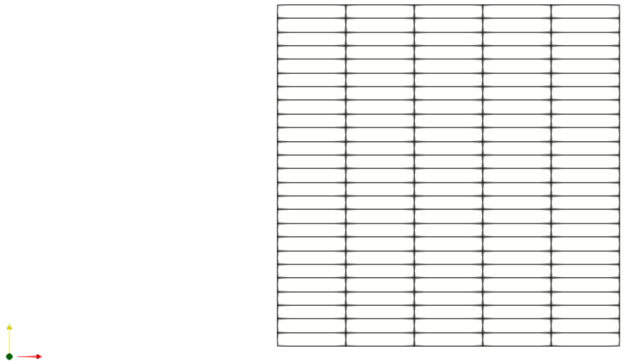


Convergence of finite volumes
for the diffusion equation on a 2D brickwall meshes
log(|numerical-exact|)
least square slope : 0.133
−0.95
−1.00
−1.05
−1.10
−1.15
−1.20
log(error)
0.8
1.0
1.2
1.4
1.6
1.8
2.0
log(sqrt(number of cells))

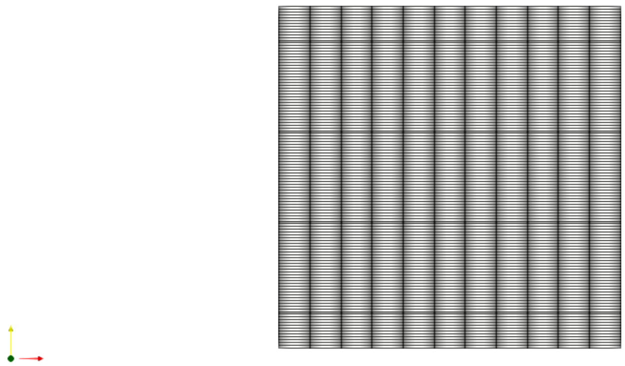
1.12 Long rectangle meshes ((n, n^2) rectangular grid)

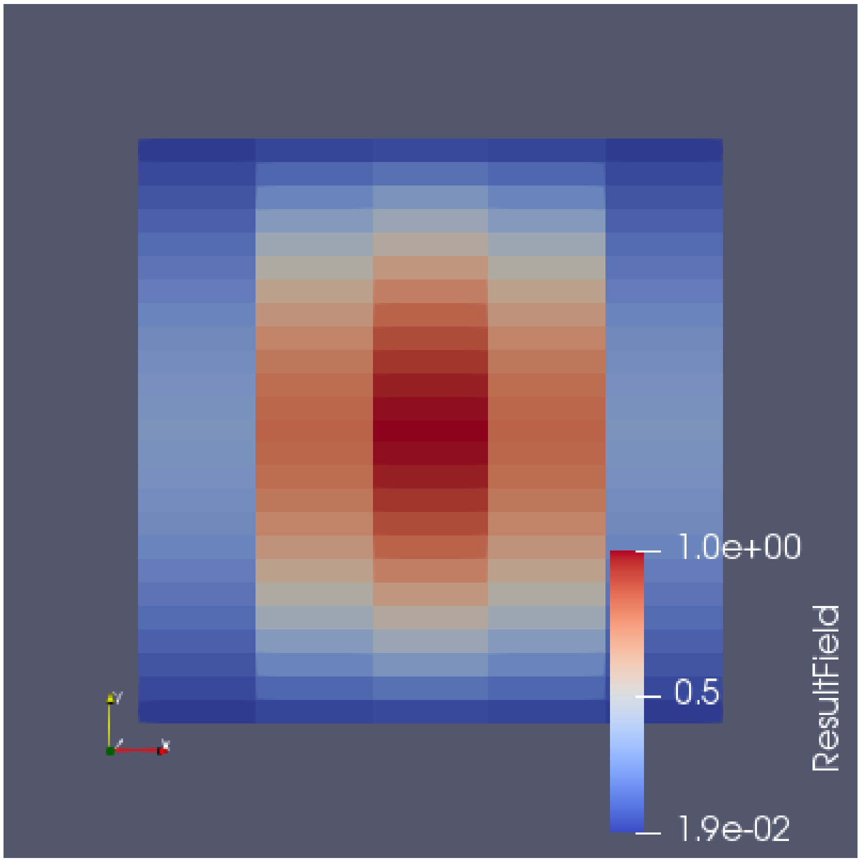


mesh 1 | mesh 2 - | - -

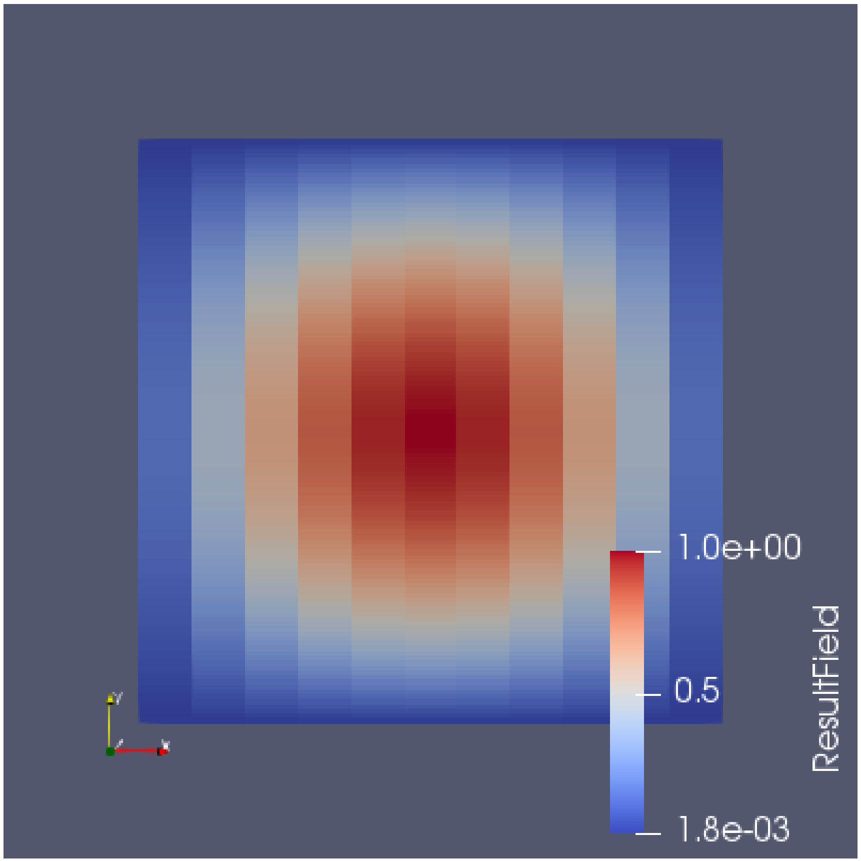


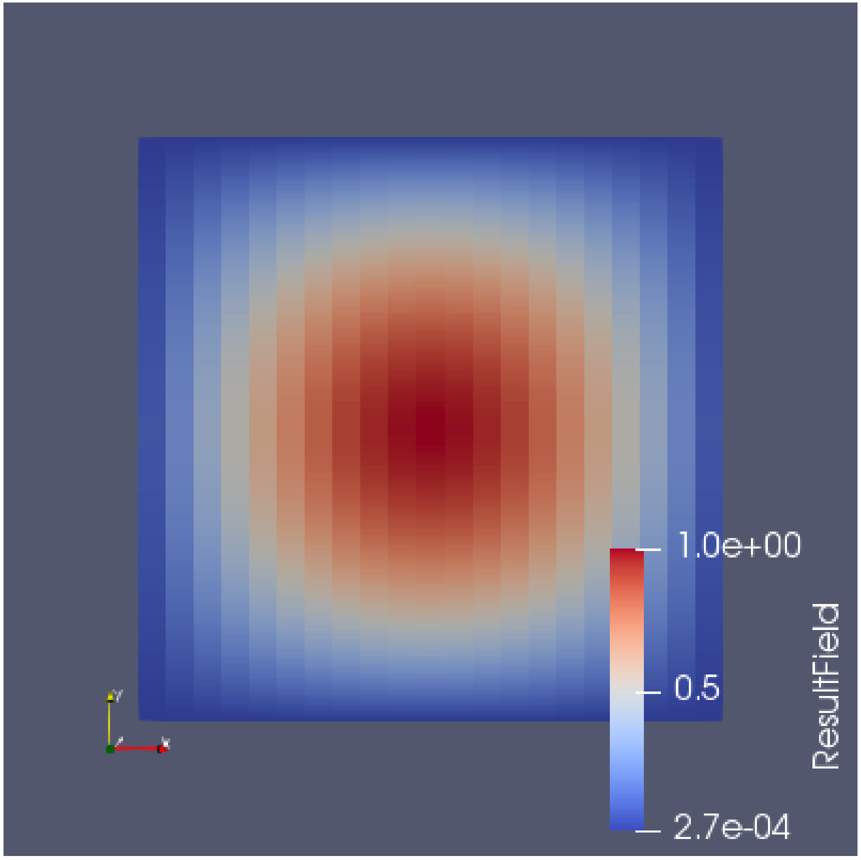
| |

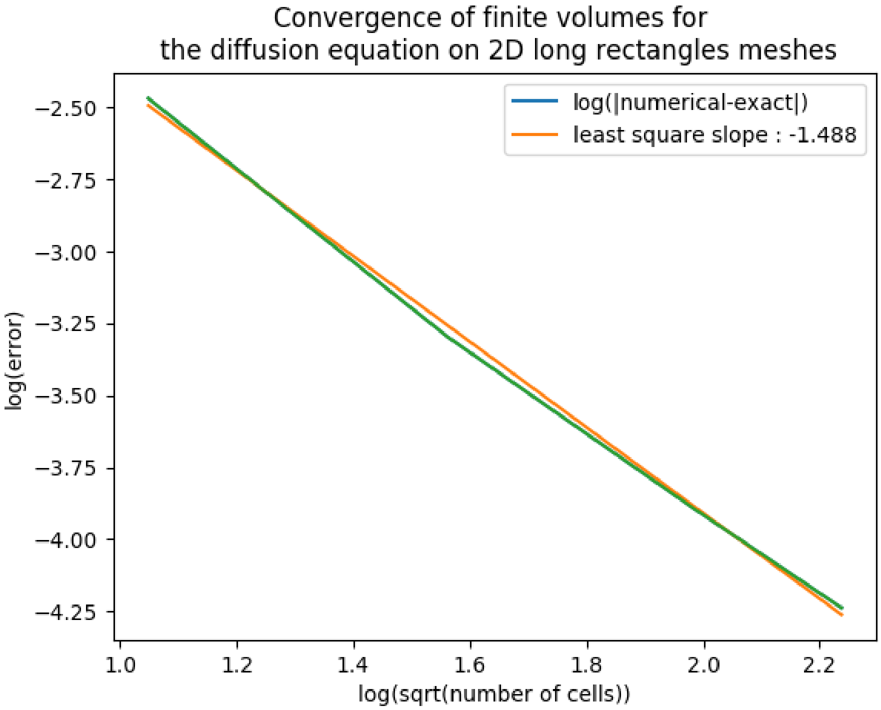




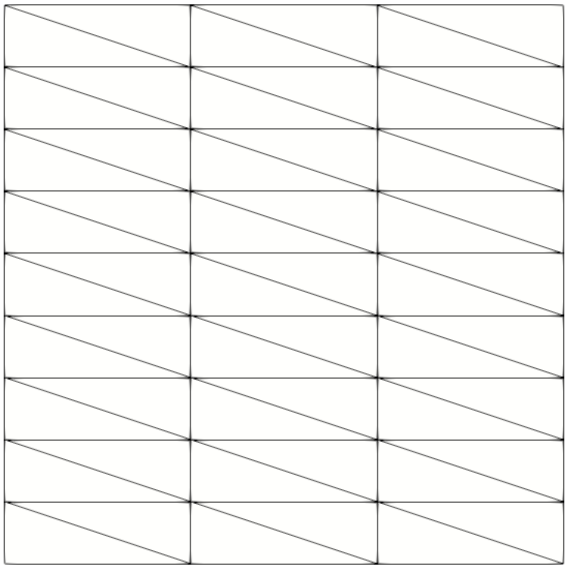
result 1 | result 2 | result 3 - | - - | -


1.0e+00
0.5
1.8e-03
ResultField
Y
X

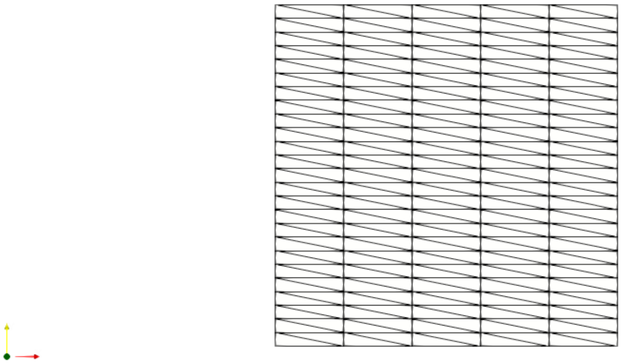

1.0e+00
0.5
2.7e-04
ResultField
Y
X


Convergence of finite volumes for
the diffusion equation on 2D long rectangles meshes
log(|numerical-exact|)
least square slope : -1.488
log(error)
−2.50
−2.75
−3.00
−3.25
−3.50
−3.75
−4.00
−4.25
1.0
1.2
1.4
1.6
1.8
2.0
2.2
log(sqrt(number of cells))

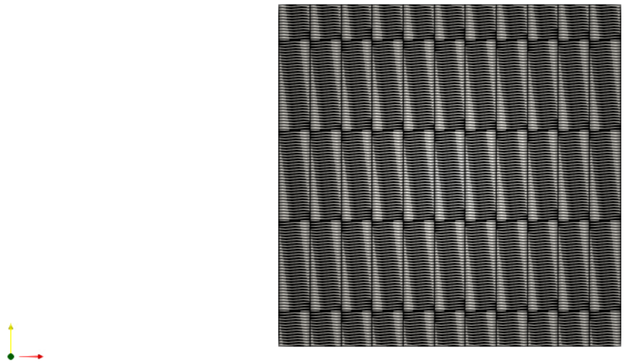
1.13 Skinny right triangle meshes (from a (n, n^2) rectangular grid)



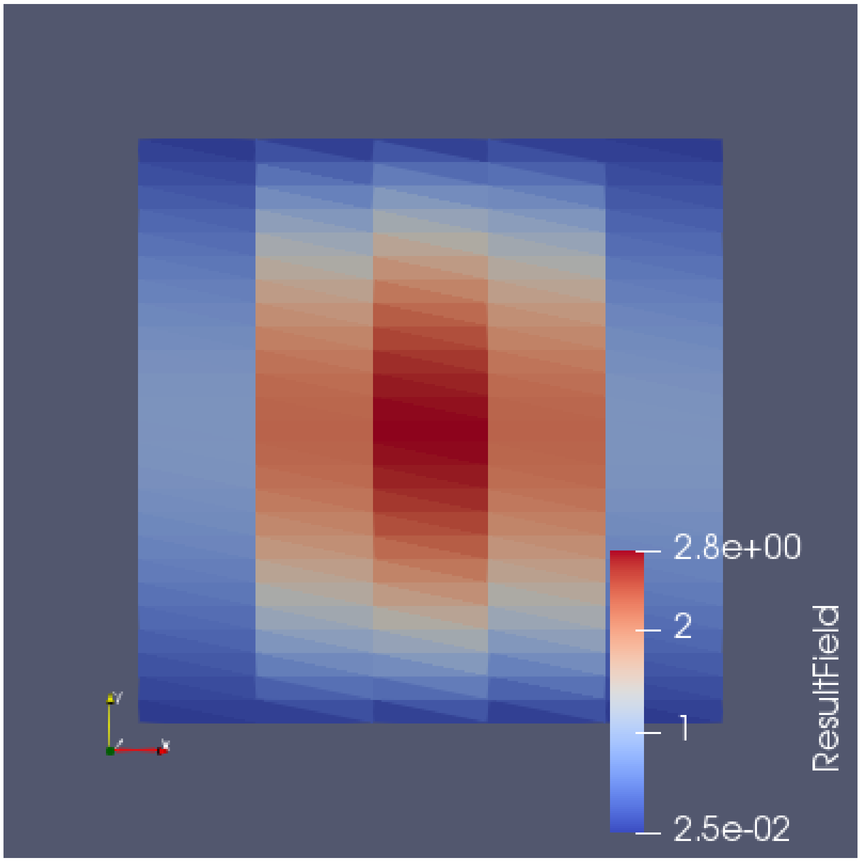
mesh 1 | mesh 2 | mesh 3 - | - - | - -

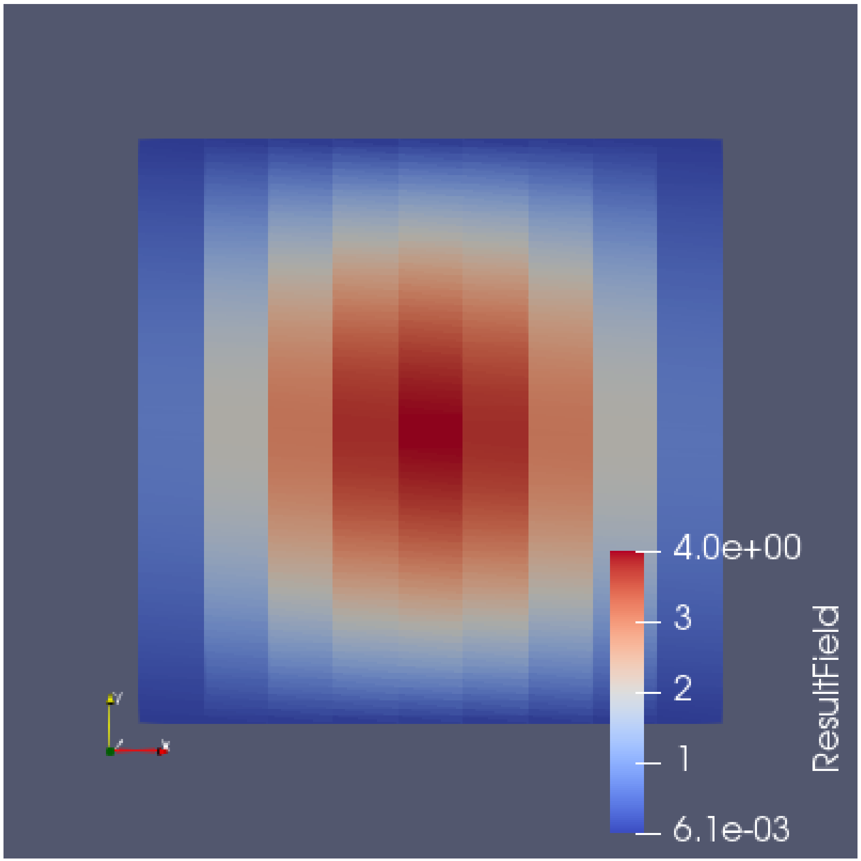


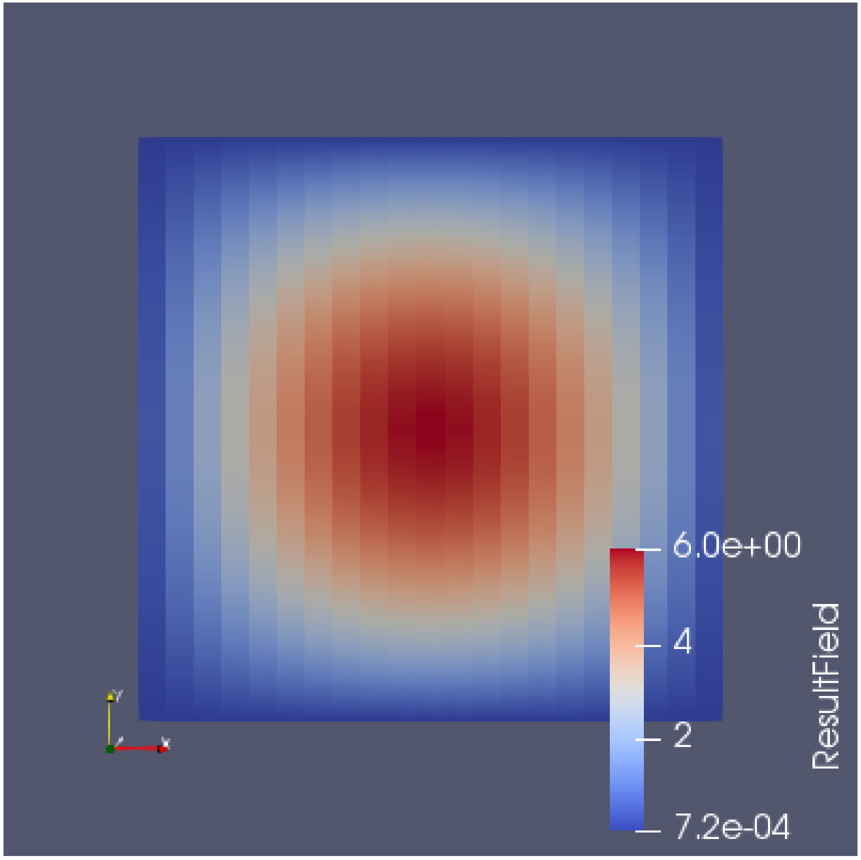
| |

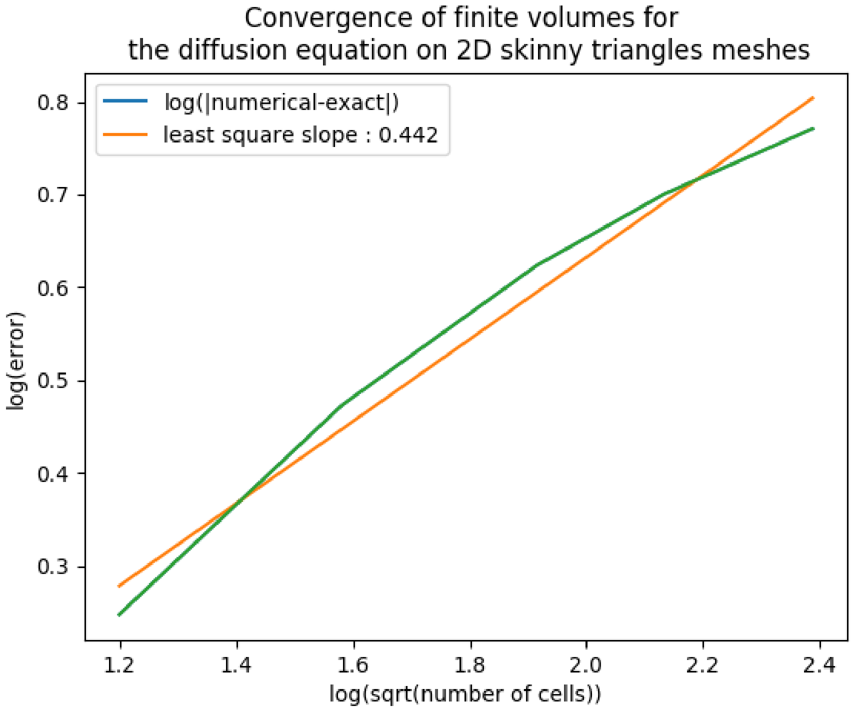


result 1 | result 2 | result 3 - | - - | -

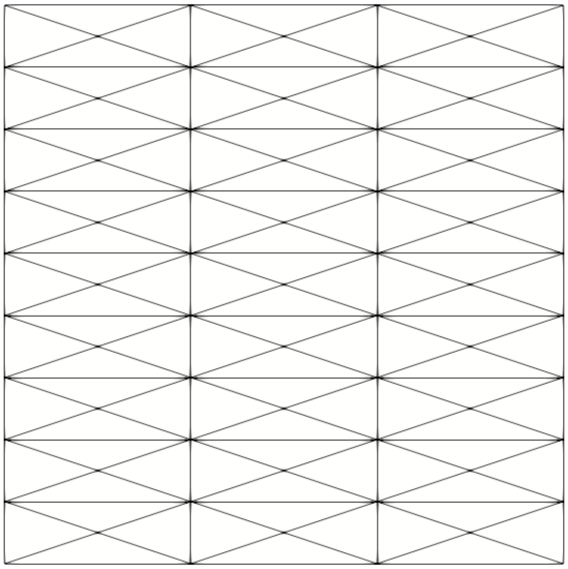



4.0e+00
3
2
1
6.1e-03
ResultField
Y
X

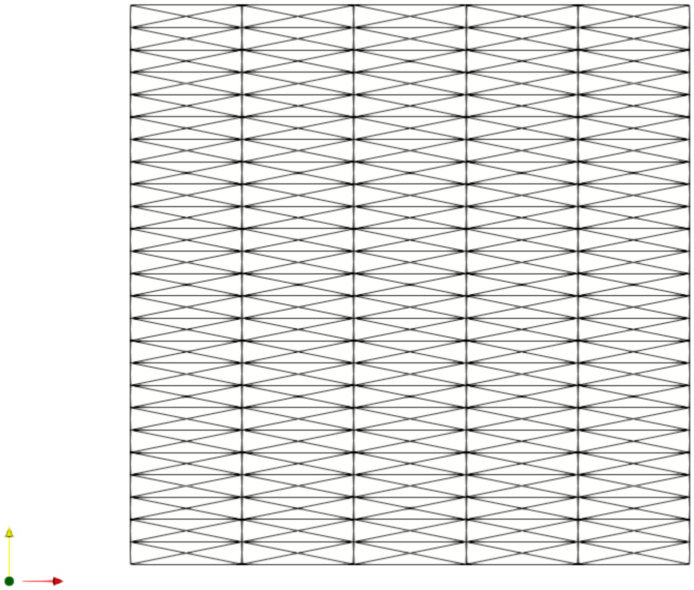

6.0e+00
4
2
7.2e-04
ResultField
Y
X


Convergence of finite volumes for
the diffusion equation on 2D skinny triangles meshes
log(|numerical-exact|)
least square slope : 0.442
log(error)
0.8
0.7
0.6
0.5
0.4
0.3
1.2
1.4
1.6
1.8
2.0
2.2
2.4
log(sqrt(number of cells))

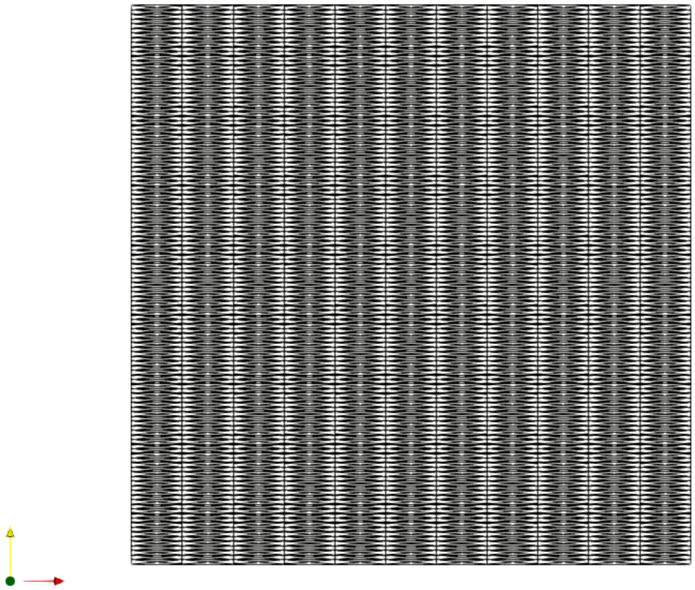
1.14 Flat cross triangle meshes (from a (n, n^2) rectangular grid)



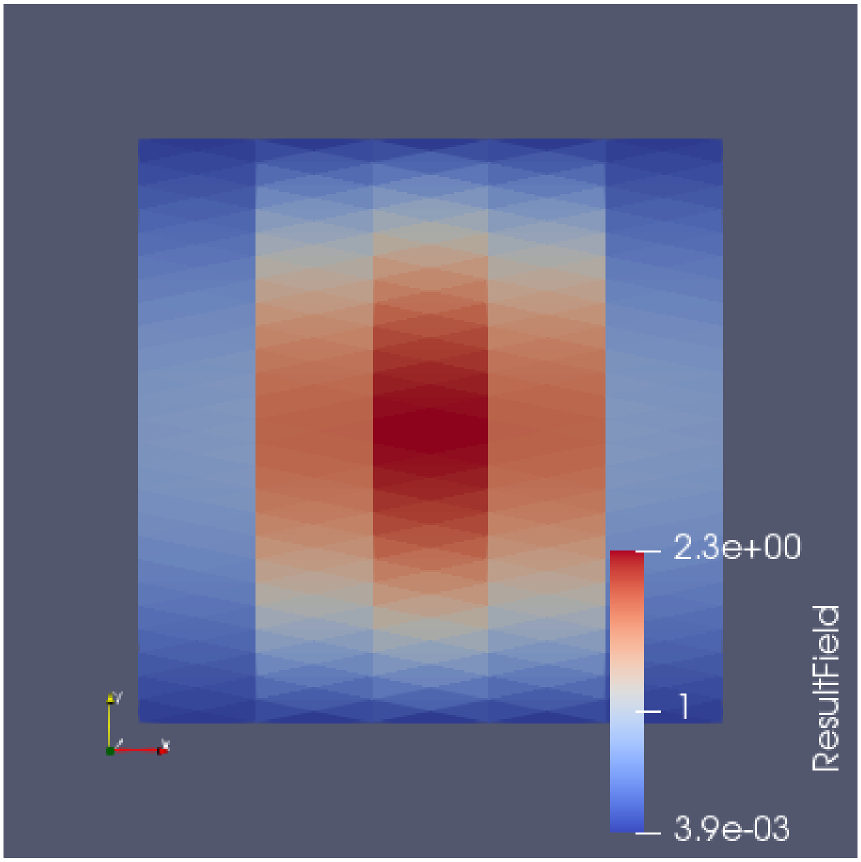
mesh 1 | mesh 2 | mesh 3 - | - - | - -

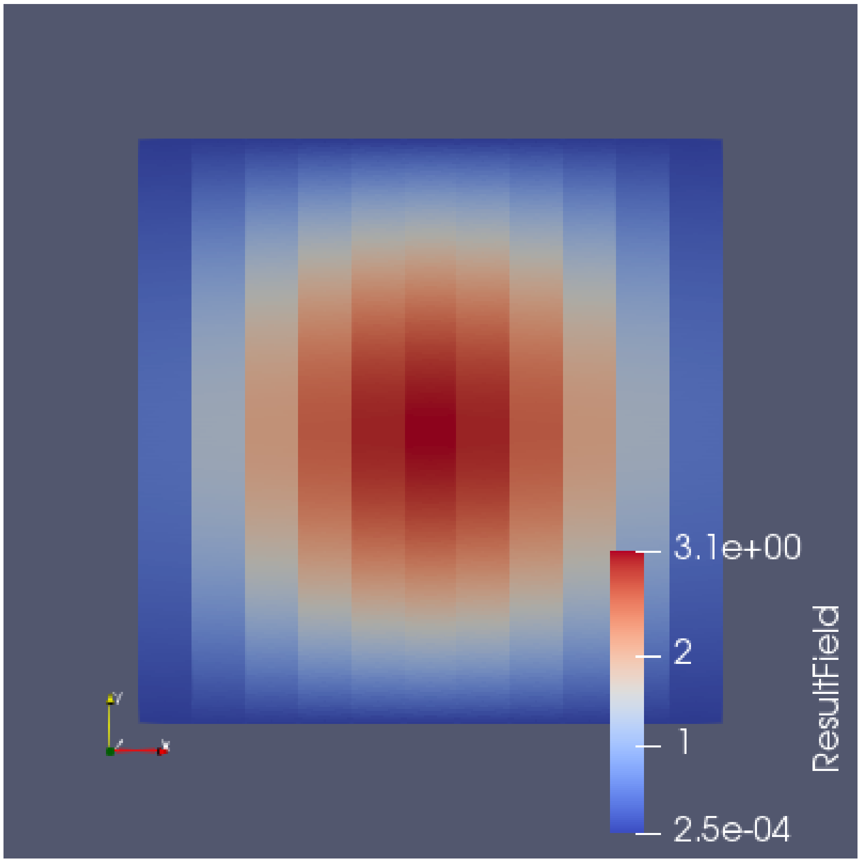


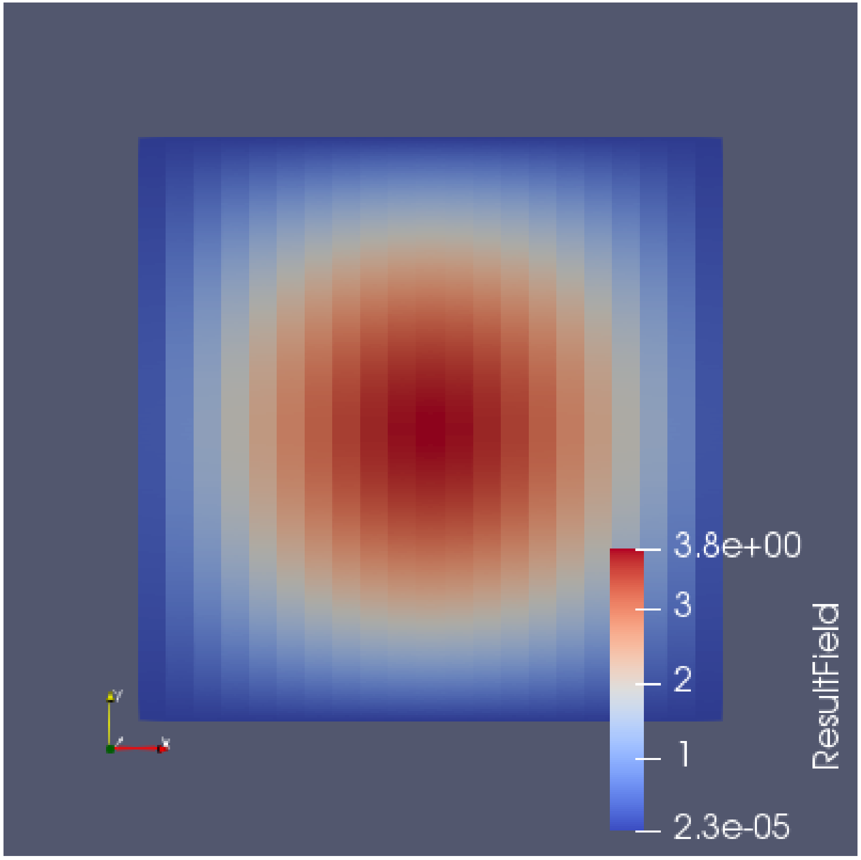
| |

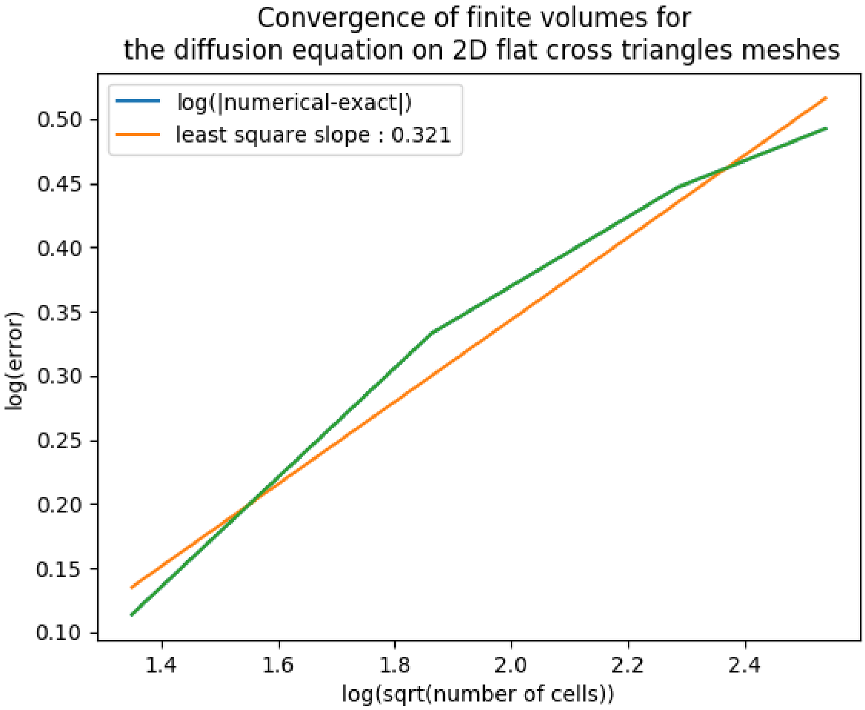


result 1 | result 2 | result 3 - | - - | -




3.1e+00
2
1
2.5e-04
ResultField
y
x


3.8e+00
3
2
1
2.3e-05
ResultField
y
x


Convergence of finite volumes for
the diffusion equation on 2D flat cross triangles meshes
log(|numerical-exact|)
least square slope : 0.321
log(error)
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
1.4
1.6
1.8
2.0
2.2
2.4
log(sqrt(number of cells))