Description

Objectif

L'intérêt du cas test est de vérifier le calcul TRUST pour la conduction de chaleur anisotrope dans une géométrie assimilé de la couche de diffusion de gaz de PEMFC.

Physique

La conduction de la chaleur dans GDL (pemfc) est anisotrope. Le coefficient de conductivité dans le plan (parallèle) de GDL est beaucoup plus élevé que celui perpendiculaire au plan (dans l'ordre de grandeur d'une centaine). De plus, la conduction est réduit lors de l'écrasement du GDL: zone non écrasé est plus conductrice que la zone écrasée. La conductivité dans la zone transitoire est interpolé linéairement.

Géométrie La géométrie utilisée est modélisé d'une partie de GDL écrasé par la plaque bipolaire (en métal) comme montrant la Figure 1.

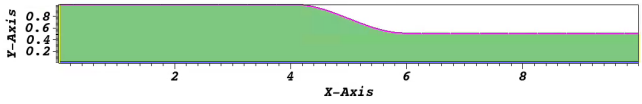

Figure 1: Géometrie de GDL (2D)

La dimension est la suivante:

- largeur (horizontale, Ox) 10
- épaisseur (vertical, Oy):
  - zone non écrasé: épaisseur à 1
  - zone écrasé: épaisseur à 0.5

Quatre bords sont définie: Gauche et Droit (jaune), Haut (violet) et Bas (bleue)

Modèle mathématique On considère la Conduction de la Chaleur dont l'équation comprend une opérateur de diffusion avec le coefficient anisotrope matriciel:

$$div(-Dgrad(T)) = q \tag{1}$$

avec:
T température K
q source par e.x. densité du flux de chaleur W/m^2

D coefficient de conductivité, $W/m/K$. Dans le cas d'anisotropie, il s'agit d'une matrice de taille 3x3 en tridimensionnel et 2x2 en bidimensionnel. La forme de matrice D dépend de la zone sur la géométrie:

- zone sur laquelle le bord est droit, non incliné: D est diagnonale
  $D = \begin{pmatrix} D_{xx} & 0 & 0 \\ 0 & D_{yy} & 0 \\ 0 & 0 & D_{zz} \end{pmatrix}$ avec $D_{xx} = D_{zz} >> D_{yy}$
- zone dont le bord est courbé et incliné: D est pleine symétrique
  $D = \begin{pmatrix} D_{xx} & D_{xy} & D_{xz} \\ - & Dyy & D_{yz} \\ - & - & D_{zz} \end{pmatrix}$ avec $D_{diagonal} >> D_{extra-diagnonal}$
  La matrix D dans cette zone est le produit matriciel $D = M.D_{diagonal}.M^{-1}$ avec M la matrice de rotation (unitaire) en fonction de l'angle d'inclination.
  - en 2D:
    $M = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$
  - en 3D
    $M = \begin{bmatrix} M_{xx} & M_{xy} & M_{xz} \\ M_{yx} & M_{yy} & M_{yz} \\ M_{zx} & M_{zy} & M_{zz} \end{bmatrix} = [\vec{u}, \vec{v}, \vec{w}]$
    avec $\vec{v} = \nabla T(\theta)$ vecteur normal du plan de bord d'écrasement GDL,
    $\vec{u} = \vec{v} \times \vec{OY}$,
    $\vec{w} = \vec{v} \times \vec{u}$
    La propriétaire de la matrice est $M^{-1} = M^T$ et $\det M = 1$

- zone d'écrasement: la magnitude de D dépend linéairement aux valeurs de conductivité de la zone non écrasement et écrasement qui correspondent respectivement à l'épaisseur maximal et minimal de GDL.
  $\alpha = \frac{ep - ep_{ec}}{ep_{non\_ec} - ep_{ec}}$

  $D = \alpha D_{non\_ec} + (1 - \alpha) D_{ec}$

Condition limite

- adiabatique sur les bords Gauche et Haut
- dirichlet $T = 0$ sur le bord Bas
- température externe imposé $T = 1$ avec coefficient d'échange imposé mix

– zone non écrasement ($x = [0, 4]$) h_imp égale à 0.

– zone écrasement ($x = [6, 10]$) h_imp égale à 10.

– zone transitoire ($x = (4, 6)$) h_imp est en fonction de x (continue) $10.0, 5(1. + \sin(0, 5\Pi(x - 5)))$

Maillage La discrétisation schéma utilise un non structuré maillage (VEF)

Sonde on crée deux sondes

- le champ température sur la ligne diagonale de la géométrie P1(0,1) et P2(10,0)
- et le flux sur le bord Haut

MEDCoupling script

Fichier MEDCoupling_matrix_aniso.py

```
import MEDLoader as ml
import medcoupling as mc

# INPUT
medfilename = "Mesh_1.med"
crushedGDLgroupname = "Haut"
ymin = 0.
ymax = 1.

# INPUT med file, read mesh of GDL
meshMEDFileRead = ml.MEDFileMesh.New(medfilename) # MEDFileUMesh
mesh2d = meshMEDFileRead.getMeshAtLevel(0)  # MEDCouplingUMesh
mesh2d.setName("mesh2D")
mesh2d.getCoords().setInfoOnComponents(["X [m]","Y [m]"])
#print "MESH2D ",mesh2d

# read mesh of the boundary of crushed GDL
mesh1d = meshMEDFileRead.getGroup(-1, crushedGDLgroupname)  #
    MEDFileUMesh
mesh1d.zipCoords() # important to remove inutil points
mesh = mesh1d.deepCopy()
mesh.setName("mesh1D")
mesh.changeSpaceDimension(1) # magic function!
mesh.checkConsistencyLight() # check
#print "MESH1D", mesh

# thickness of GDL non crushed, assuming that GDL is crushed in
    vertical axe, e.g Oy
Cxx1 = 100  # in-plane, not crushed
Cxx2 = 200 # in-plane, crushed
Cyy1 = 1 # through-plane, not crushed
Cyy2 = 2 # through-plane, crushed
```

```
ep1 = 1. # max thickness = ymax - ymin , not crushed
ep2 = 0.5 # min thickness, crushed

# creer un champ sur la maille surfacique
time = 4.22 # random ms
epArr = mesh1d.getCoords()[:,1]-ymin # epaisseur calculated from the curve bord y-coordinate
#print "MIN MAX VALEUR D'EPAISSEUR EN METRE ", epArr.getMinMaxPerComponent()

# creer un champ sur le maillage du bord
f = ml.MEDCouplingFieldDouble(ml.ON_NODES, ml.ONE_TIME) # impossible pour un champ no_time ???
f.setTimeUnit("ms") # Time unit is ms.
f.setTime(time,1,-1)   # Time attached is 4.22 ms, iteration id is 2 and order id (or sub iteration id) is -1
f.setArray(epArr)
f.setMesh(mesh)
f.setName("champ_epaisseur_1D")
#print "CHAMP DEFINED ON THE PLATE MESH FOR INTERPOLATION", f
ml.WriteField("ep1d.med",f, True)

# JUST FOR TEST
bary = mesh.computeCellCenterOfMass()
valArray = f.getValueOnMulti(bary)
#print "GETTING VALUES (INTERPOLATED) IN BARYCENTRE OF CELLS OF MESH ", valArray

# creer un champ sur le maillage volumique
coo2Dx = mesh2d.computeCellCenterOfMass()[:, 0]
val2dArr = f.getValueOnMulti(coo2Dx)
f2 = ml.MEDCouplingFieldDouble(ml.ON_CELLS, ml.ONE_TIME)  # impossible pour un champ no_time ???
f2.setTimeUnit("ms") # Time unit is ms.
f2.setTime(time,1,-1)  # Time attached is 4.22 ms, iteration id is 2 and order id (or sub iteration id) is -1
f2.setArray(val2dArr)
f2.setMesh(mesh2d)
f2.setName("champ_epaisseur_2D")
#print "CHAMP DEFINED IN THE MESH2D ", f2
ml.WriteField("ep2d.med",f2, True)

# ortho field interpolated
n = mesh2d.getNumberOfCells()
gradTField = ml.MEDCouplingFieldDouble(ml.ON_CELLS, ml.ONE_TIME)  # impossible pour un champ no_time ???
gradTData = mc.DataArrayDouble(n, 2) # nb_cells x 2 comp
gradTData[:,0] = [0.]
gradTData[:,1] = [1.]
gradTField.setTimeUnit("ms") # Time unit is ms.
gradTField.setTime(time,1,-1) # Time attached is 4.22 ms, iteration id is 2 and order id (or sub iteration id) is -1
gradTField.setArray(gradTData)
gradTField.setMesh(mesh2d)
gradTField.setName("champ_normal_2D")
# create the normal vector on the boundary 'haut'
normaField = mesh1d.buildOrthogonalField()
```

```
normaField.setMesh(mesh)
normaData = normaField.getArray() # ATTENTION: ortho field not
    correct!!! vector [-0 -1] for elem #0 -> #79
for i in range(len(normaData)):
  if(normaData[i,1] < 0):
    normaData[i,0] = -normaData[i, 0]
    normaData[i,1] = -normaData[i, 1]

#print "normaField", normaField
#print "normaData", normaData

bary = gradTField.getMesh().computeCellCenterOfMass() # [x, y]
norm = normaField.getValueOnMulti(bary[:,0])
epai = f.getValueOnMulti(bary[:,0])
uy = mc.DataArrayDouble(n, 2)
uy[:,0] = 0.
uy[:,1] = 1.
dist = norm - uy
eps = 1e-8
for i in range(n):
  # get epaisseur
  epi = epai[i]
  # get y
  yi = bary[i,1]
  # alpha
  alpha = (yi - ymin) / (epi - ymin)
  #if(dist[i,0] > eps):
  if(dist.magnitude()[i] > eps):
    gradTData[i] = alpha*norm[i] + (1. - alpha)*uy[i] # ATTENTION:
    gradTData not identity

gradTData /= gradTData.magnitude()

#print "gradTData" , gradTData
ml.WriteField("ortho.med",gradTField, True)

# number of Cells
n = gradTField.getMesh().getNumberOfCells()
#print "number of Cells", n

# coefficient de conductivite dans le plan
conduc_xx = mc.DataArrayDouble(n)
conduc_xx[:] = Cxx1

# coefficient de conductivite hors le plan
conduc_yy = mc.DataArrayDouble(n)
conduc_yy[:] = Cyy1

for i in range(n):
    alpha = (val2dArr[i] - ep2)/(ep1 - ep2)
    conduc_xx[i] = alpha * Cxx1 + (1-alpha)*Cxx2
    conduc_yy[i] = alpha * Cyy1 + (1-alpha)*Cyy2

#print "min max conductivity in-plan xx ", conduc_xx.
    getMinMaxPerComponent()
#print "min max conductivity through-plan yy ", conduc_yy.
    getMinMaxPerComponent()
```

```
132
133 # matrix inverse of rotation M^-1
134 matM = mc.DataArrayDouble(n,4)  # 4 comp or 2x2 comp
135 matM[:, 1] = gradTData[:, 0]
136 matM[:, 3] = gradTData[:, 1]
137 matM[:, 0] = gradTData[:,1]
138 matM[:, 2] = -gradTData[:,0]
139
140 # calcul du coefficient de conductivite
141 conduc = mc.DataArrayDouble(n,4)     # 4 components
142 conduc[:,:] = 0.
143 #print "CONDUCTIVITY INITIAL", conduc
144 conduc[:,0] += matM[:,0]*matM[:,0]*conduc_xx
145 conduc[:,0] += matM[:,1]*matM[:,1]*conduc_yy
146
147 conduc[:,1] += matM[:,0]*matM[:,2]*conduc_xx
148 conduc[:,1] += matM[:,1]*matM[:,2]*conduc_yy
149
150 conduc[:,2] = conduc[:,1]
151
152 conduc[:,3] += matM[:,2]*matM[:,2]*conduc_xx
153 conduc[:,3] += matM[:,3]*matM[:,3]*conduc_yy
154
155 #print "MIN MAX CONDUCTIVITY PER COMPONENT", conduc.
    getMinMaxPerComponent()
156
157 # export champ to a .med file
158 fconduc = ml.MEDCouplingFieldDouble(ml.ON_CELLS, ml.ONE_TIME) #
    impossible pour un champ no_time ???
159 fconduc.setTimeUnit("s") # Time unit is second
160 fconduc.setTime(time,1,-1) # Time attached is 4.22 ms, iteration id
     is 2 and order id (or sub iteration id) is -1
161 fconduc.setArray(conduc)
162 fconduc.setMesh(mesh2d)
163 fconduc.setName("CONDUCTIVITY_ELEM_dom")
164 #print "CHAMP CONDUCTIVITY DEFINED IN THE MESH2D ", fconduc
165 ml.WriteField("conduc2d.med",fconduc, True)
```

Data fichier

Fichier PEMFC_2D_aniso_withMEDCouplingFull.data

```
1 # PARALLEL OK #
2 # Heat Conduction 2D anisotrope #
3 dimension 2
4
5 # domain name #
6 domaine dom
7
8 #  physic problem: heat conduction  #
9 pb_conduction pb
10
11 # BEGIN MESH #
12 Lire_med family_names_from_group_names dom Mesh_1 Mesh_1.med
13 # END MESH #
14
```

```
# BEGIN PARTITION
Partition dom
{
    /* Choose Nb_parts so to have ~ 25000 cells per processor */
  Partition_tool metis { nb_parts 2 }
  Larg_joint 2
  zones_name dom
}
End
END PARTITION #

# BEGIN SCATTER
Scatter dom.Zones dom
END SCATTER #

# discretisation VDF for the domain 2D #
vefprep1b dis

# time scheme #
Scheme_euler_implicit sch
Read sch
{
  # Time step #
  # Initial time [s] #
  tinit 0
  # Min time step #
  # dt_min 1e-9 #
  # Max time step #
  # dt_max 1. #
  # facsec maximal possible value for the heat transfer #
  facsec 1000
  facsec_max 5000
  # .out files printing period #
  dt_impr 1e-3 # Note: small value to print at each time step #
  # .sauv files printing period #
  dt_sauv 1e-3
  # Stop if one of the following criteria is checked: #
  # End time [s] #
  tmax 10.0
  # Max number of time steps #
  # nb_pas_dt_max 10 #
  # max number of implicit solver #
  max_iter_implicite 50
  # Convergence threshold (see .dt_ev file) #
  seuil_statio_relatif_deconseille 1
  seuil_statio 1e-5
  Solveur Implicite
  {
      Solveur petsc cholesky { quiet }
  }
}

# physics properties of medium  #
solide sol
read sol
{
  rho champ_uniforme 4 1. 1. 1. 1.
```

```
  lambda champ_fonc_med last_time conduc2d.med Mesh_1
    CONDUCTIVITY_ELEM_dom elem 0
  Cp champ_uniforme 4 1. 1. 1. 1.
}


# associate the physic problem with the geometry, the time sch and
    the physic properties #
associate pb dom
associate pb sch
associate pb sol

# meshing the geometry with the given discretisation method #
discretize pb dis

# definition of the physic problem: heat conduction #
read pb
{
  # Resolution of the laplacien equation modifying the heat
    equation #
  conduction
  {
    diffusion { Tensor }
    initial_conditions {
      temperature champ_uniforme 1 0.0
      # temperature champ_fonc_med last_time prepare_0000.med dom
    temperature elem 0 #
    }
    boundary_conditions {
      Gauche paroi_flux_impose champ_front_uniforme 1 0.
      Droit paroi_flux_impose champ_front_uniforme 1 0.
      Haut paroi_echange_externe_impose h_imp champ_front_fonc_xyz
    1 (x[4.)*0.+(x>4.)*(x<6.)*10.*0.5*(1.+SIN(0.5*Pi*(x-5.)))+(x
    ]6.)*(10.) T_ext champ_front_uniforme 1 1.
      Bas paroi_temperature_imposee champ_front_uniforme 1 0.
    }
  }

  # post traitement  #
  post_processing
  {
    definition_champs
    {
      # gradient of temperature grad(u) = [ux uy]^T #
      gradT gradient {
        source refchamp { Pb_champ pb temperature }
      }
    }
    probes {
      sonde_temperature temperature periode 0.01 segment 501 0. 1.
    10. 0.
    }
    format lata     # lata for VisIt tool #
    fields dt_post 0.1 # warning small value #
    {
      temperature elem
      temperature som
```

```
      gradT som
      diffusivite_thermique elem
      capacite_calorifique elem
      masse_volumique elem
    }
  }
}

imprimer_flux dom { Haut Bas }

# solve the problem #
solve pb

end
```