#!/usr/bin/env python

# 12.01.2007, c

"""

Solve partial differential equations given in a SfePy problem definition file.

Example problem definition files can be found in ``examples/`` directory of the

SfePy top-level directory. This script works with all the examples except those

in ``examples/standalone/``.

Both normal and parametric study runs are supported. A parametric study allows

repeated runs for varying some of the simulation parameters - see

``examples/diffusion/poisson\_parametric\_study.py`` file.

"""

from \_\_future\_\_ import print\_function

from \_\_future\_\_ import absolute\_import

from argparse import ArgumentParser, RawDescriptionHelpFormatter

# import sfepy

import meshio

from sfepy.mechanics.extmods import \*

from sfepy.discrete.common.extmods import \*

from sfepy.discrete.fem.extmods import \*

from sfepy.base.base import output

from sfepy.base.conf import ProblemConf, get\_standard\_keywords

from sfepy.applications import PDESolverApp, EVPSolverApp

from sfepy.mechanics.matcoefs import stiffness\_from\_youngpoisson

from sfepy.discrete.fem.extmods.bases import CLagrangeContext

from sfepy.discrete.fem.utils import refine\_mesh

from sfepy.discrete.common.poly\_spaces import PolySpace

from sfepy.discrete.dg.poly\_spaces import \*

from sfepy.discrete.fem import \*

from sfepy.discrete.fem.poly\_spaces import \*

from sfepy.mechanics import \*

from sfepy.terms import \*

from sfepy.solvers import \*

from sfepy.solvers.ls import \*

from sfepy.solvers.nls import \*

from sfepy.solvers.ts\_solvers import \*

from sfepy.terms.terms\_elastic import \*

from sfepy.discrete.fem.meshio import \*

import distutils

from distutils.version import LooseVersion

from meshio import \*

from sfepy import data\_dir

from sfepy.base.base import IndexedStruct

from sfepy.discrete import (FieldVariable, Material, Integral, Function, Equation, Equations, Problem)

from sfepy.discrete.fem import Mesh, FEDomain, Field

from sfepy.terms import Term

from sfepy.discrete.conditions import Conditions, EssentialBC

from sfepy.solvers.ls import ScipyDirect

from sfepy.solvers.nls import Newton

from sfepy.mechanics.tensors import get\_von\_mises\_stress

import numpy as np

import string

from sfepy.base.base import output

# from sfepy.postprocess.viewer import Viewer

import os

import json

def print\_terms():

import sfepy.terms as t

tt = t.term\_table

print('Terms: %d available:' % len(tt))

print(sorted(tt.keys()))

def print\_solvers():

from sfepy.solvers import solver\_table

print('Solvers: %d available:' % len(solver\_table))

print(sorted(solver\_table.keys()))

helps = {

'debug':

'automatically start debugger when an exception is raised',

'conf' :

'override problem description file items, written as python'

' dictionary without surrounding braces',

'options' : 'override options item of problem description,'

' written as python dictionary without surrounding braces',

'define' : 'pass given arguments written as python dictionary'

' without surrounding braces to define() function of problem description'

' file',

'filename' :

'basename of output file(s) [default: <basename of input file>]',

'output\_format' :

'output file format, one of: {vtk, h5} [default: vtk]',

'save\_restart' :

'if given, save restart files according to the given mode.',

'load\_restart' :

'if given, load the given restart file',

'log' :

'log all messages to specified file (existing file will be overwritten!)',

'quiet' :

'do not print any messages to screen',

'save\_ebc' :

'save a zero solution with applied EBCs (Dirichlet boundary conditions)',

'save\_ebc\_nodes' :

'save a zero solution with added non-zeros in EBC (Dirichlet boundary'

' conditions) nodes - scalar variables are shown using colors,'

' vector variables using arrows with non-zero components corresponding'

' to constrained components',

'save\_regions' :

'save problem regions as meshes',

'save\_regions\_as\_groups' :

'save problem regions in a single mesh but mark them by using different'

' element/node group numbers',

'save\_field\_meshes' :

'save meshes of problem fields (with extra DOF nodes)',

'solve\_not' :

'do not solve (use in connection with --save-\*)',

'list' :

'list data, what can be one of: {terms, solvers}',

}

def run\_sfepy():

parser = ArgumentParser(description=\_\_doc\_\_,

formatter\_class=RawDescriptionHelpFormatter)

parser.add\_argument('--debug',

action='store\_true', dest='debug',

default=False, help=helps['debug'])

parser.add\_argument('-c', '--conf', metavar='"key : value, ..."',

action='store', dest='conf', type=str,

default=None, help= helps['conf'])

parser.add\_argument('-O', '--options', metavar='"key : value, ..."',

action='store', dest='app\_options', type=str,

default=None, help=helps['options'])

parser.add\_argument('-d', '--define', metavar='"key : value, ..."',

action='store', dest='define\_args', type=str,

default=None, help=helps['define'])

parser.add\_argument('-o', metavar='filename',

action='store', dest='output\_filename\_trunk',

default=None, help=helps['filename'])

parser.add\_argument('--format', metavar='format',

action='store', dest='output\_format',

default=None, help=helps['output\_format'])

parser.add\_argument('--save-restart', metavar='mode', type=int,

action='store', dest='save\_restart',

default=None, help=helps['save\_restart'])

parser.add\_argument('--load-restart', metavar='filename',

action='store', dest='load\_restart',

default=None, help=helps['load\_restart'])

parser.add\_argument('--log', metavar='file',

action='store', dest='log',

default=None, help=helps['log'])

parser.add\_argument('-q', '--quiet',

action='store\_true', dest='quiet',

default=False, help=helps['quiet'])

parser.add\_argument('--save-ebc',

action='store\_true', dest='save\_ebc',

default=False, help=helps['save\_ebc'])

parser.add\_argument('--save-ebc-nodes',

action='store\_true', dest='save\_ebc\_nodes',

default=False, help=helps['save\_ebc\_nodes'])

parser.add\_argument('--save-regions',

action='store\_true', dest='save\_regions',

default=False, help=helps['save\_regions'])

parser.add\_argument('--save-regions-as-groups',

action='store\_true', dest='save\_regions\_as\_groups',

default=False, help=helps['save\_regions\_as\_groups'])

parser.add\_argument('--save-field-meshes',

action='store\_true', dest='save\_field\_meshes',

default=False, help=helps['save\_field\_meshes'])

parser.add\_argument('--solve-not',

action='store\_true', dest='solve\_not',

default=False, help=helps['solve\_not'])

group = parser.add\_mutually\_exclusive\_group(required=False)

group.add\_argument('--list', metavar='what',

action='store', dest='\_list',

default=None, help=helps['list'])

options, petsc\_opts = parser.parse\_known\_args()

if options.debug:

from sfepy.base.base import debug\_on\_error; debug\_on\_error()

filename\_in = 'VB\_problem\_discription\_lame.py'

output.set\_output(filename=options.log,

quiet=options.quiet,

combined=options.log is not None)

required, other = get\_standard\_keywords()

if options.solve\_not:

required.remove('equations')

required.remove('solver\_[0-9]+|solvers')

other.extend(['equations'])

conf = ProblemConf.from\_file\_and\_options(filename\_in, options,

required, other,

define\_args=options.define\_args)

opts = conf.options

output\_prefix = opts.get('output\_prefix', 'sfepy:')

opts.save\_restart = options.save\_restart

opts.load\_restart = options.load\_restart

if conf.options.get('evps') is None:

app = PDESolverApp(conf, options, output\_prefix)

else:

app = EVPSolverApp(conf, options, output\_prefix)

if hasattr(opts, 'parametric\_hook'): # Parametric study.

parametric\_hook = conf.get\_function(opts.parametric\_hook)

app.parametrize(parametric\_hook)

app()

if \_\_name\_\_ == '\_\_main\_\_':

run\_sfepy()

from \_\_future\_\_ import absolute\_import

from sfepy.mechanics.matcoefs import stiffness\_from\_lame

from sfepy.mechanics.matcoefs import stiffness\_from\_youngpoisson

from sfepy.discrete.fem.utils import refine\_mesh

from sfepy import data\_dir

from sfepy.base.base import IndexedStruct

from sfepy.discrete import (FieldVariable, Material, Integral, Function, Equation, Equations, Problem)

from sfepy.discrete.fem import Mesh, FEDomain, Field

from sfepy.terms import Term

from sfepy.discrete.conditions import Conditions, EssentialBC

from sfepy.solvers.ls import ScipyDirect

from sfepy.solvers.nls import Newton

from sfepy.mechanics.tensors import get\_von\_mises\_stress

import numpy as np

import meshio

import string

from sfepy.base.base import output

# from sfepy.postprocess.viewer import Viewer

import os

import json

'''

In this standard problem discription file we define PMMA and Bone separately and we trying to define material/region for

each voxel separately.

'''

#filename\_mesh = './data/result/anonymous\_L3\_FEA\_DispLoading2000um.inp'

#material\_file = './data/result/anonymous\_L3\_Material.txt'

filename\_mesh = './data/result/FEA\_DispLoading.inp'

material\_file = './data/result/Material.txt'

#h = 31.4

with open('data/result/height.json', 'r') as f:

data = json.load(f)

h = data['height']

"""

Please help with import the bone-specific height

"""

def vertebral\_bone\_strength(out, pb, state, extend=False):

"""

Calculate and output strain and stress for given displacements.

"""

from sfepy.base.base import Struct

from sfepy.base.base import output

from sfepy.mechanics.tensors import get\_full\_indices

from sfepy.discrete import Material, Function

import numpy as np

def eval\_force(region\_name):

strain = pb.evaluate(

'ev\_cauchy\_strain\_s.3.%s(u)' % region\_name, mode='qp',

verbose=False,

)

D = pb.evaluate(

'ev\_surface\_integrate\_mat.3.%s(m.D, u)' % region\_name,

mode='qp',

verbose=False,

)

normal = np.array([1, 0, 0], dtype=np.float64)

s2f = get\_full\_indices(len(normal))

stress = np.einsum('cqij,cqjk->cqik', D, strain)

# Full (matrix) form of stress.

mstress = stress[..., s2f, 0]

# Force in normal direction.

force = np.einsum('cqij,i,j->cq', mstress, normal, normal)

def get\_force(ts, coors, mode=None, \*\*kwargs):

if mode == 'qp':

return {'force': force.reshape(coors.shape[0], 1, 1)}

aux = Material('aux', function=Function('get\_force', get\_force))

middle\_force = - pb.evaluate(

'ev\_surface\_integrate\_mat.3.%s(aux.force, u)' % region\_name,

aux=aux,

verbose=False,

)

stiffness = middle\_force / 0.69

factor = 0.0068

height = h

strength = stiffness \* factor \* height

'''

with open('strength.txt', 'w') as file:

begin = 'Vertebral bone strength: ' + str(strength) + ' Newton'

file.write('%s\n' % begin)

'''

strenthDict = {'strength':str(round(strength, 2))}

with open(os.path.join('report', 'fea.json'), 'w') as fp:

json.dump(strenthDict, fp)

'''

import sys

from PyQt5.QtWidgets import QApplication, QLabel, QMainWindow

from PyQt5.QtCore import Qt

class MainWindow(QMainWindow):

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(MainWindow, self).\_\_init\_\_(\*args, \*\*kwargs)

self.setWindowTitle("Bone strength result")

file\_in = open("strength.txt", 'r')

for y in file\_in.readlines():

label = QLabel(y)

label.setAlignment(Qt.AlignCenter)

self.setCentralWidget(label)

app = QApplication(sys.argv)

window = MainWindow()

window.show()

app.exec\_()

"""

you have to write code for further calculating bone height for different

here 32.6 mm is the height of one patient named as CUIMINGYI

"""

'''

output('Vertebral bone strength: ', strength, ' Newton')

eval\_force('inferior')

ev = pb.evaluate

stress = ev('ev\_cauchy\_stress.3.Omega(m.D, u)', mode='el\_avg',

copy\_materials=False)

vms = get\_von\_mises\_stress(stress.squeeze())

vms.shape = (vms.shape[0], 1, 1, 1)

out['von\_mises\_stress'] = Struct(name='output\_data', mode='cell', data=vms, dofs=None)

return out

'''

Now we finished the region definition for bone voxel region definition. Then we will move to main function

'''

'''

remember change the input file for FEA simulation

'''

refinement\_level = 0

filename\_mesh = refine\_mesh(filename\_mesh, refinement\_level)

output\_dir = './data/result/' # set this to a valid directory you have write access to

options = {'ls': 'ls.scipy\_iterative',

'output\_dir': output\_dir,

'post\_process\_hook': 'vertebral\_bone\_strength'

}

region = meshio.read(filename\_mesh)

Top\_cell\_index = []

Bottom\_cell\_index = []

for each\_array in region.cell\_sets['SET-TOP']:

Top\_cell\_index.append(each\_array)

for each\_array in region.cell\_sets['SET-BOTTOM']:

Bottom\_cell\_index.append(each\_array)

Top\_cell = []

Bottom\_cell = []

for each\_index in Top\_cell\_index[0]:

single\_index = str(each\_index)

cell = single\_index

Top\_cell.append(cell)

for each\_index in Bottom\_cell\_index[0]:

single\_index = str(each\_index)

cell = single\_index

Bottom\_cell.append(cell)

Top\_cell[0] = 'cell ' + str(Top\_cell[0])

Bottom\_cell[0] = 'cell ' + str(Bottom\_cell[0])

Inferior\_PMMA = []

Superior\_PMMA = []

separator = ', '

Superior\_PMMA = separator.join(Top\_cell)

Inferior\_PMMA = separator.join(Bottom\_cell)

Bone\_cell\_index = {}

del region.cell\_sets['SET-BOTTOM']

del region.cell\_sets['SET-TOP']

del region.cell\_sets['\_Inferior\_S4']

Bone\_cell\_index = region.cell\_sets

'''

For extracting bone sets only we firstly remove other sets not below to bone

'''

Bone\_element = []

for each\_index in Bone\_cell\_index:

array = Bone\_cell\_index[each\_index]

for each\_element in array:

for each\_array in each\_element:

Bone\_element.append(each\_array)

Bone\_element\_cell = []

for each\_element in Bone\_element:

name = 'cell ' + str(each\_element)

Bone\_element\_cell.append(name)

Bone\_region = []

for each\_element in Bone\_element:

Bone\_region.append(str(each\_element))

Bone\_voxel\_name = []

for each\_element in Bone\_element:

name = 'Bone' + str(each\_element)

Bone\_voxel\_name.append(name)

mesh = Mesh.from\_file(filename\_mesh)

domain = FEDomain('domain', mesh)

min\_x, max\_x = domain.get\_mesh\_bounding\_box()[:, 0]

eps = 1e-8 \* (max\_x - min\_x)

with open('regions.txt', 'w') as file:

begin = 'regions = {'

file.write('%s\n' % begin)

omega = '"Omega"' + ': ' + '"all"' + ','

file.write('%s\n' % omega)

inferior = '"inferior"' + ': (' + '"vertices in x < %.10f" % (min\_x + eps)' + ', ' + '"facet"' + ')' + ','

file.write('%s\n' % inferior)

superior = '"superior"' + ': (' + '"vertices in x > %.10f" % (max\_x - eps)' + ', ' + '"facet"' + ')' + ','

file.write('%s\n' % superior)

Top\_PMMA = '"Top\_PMMA"' + ': (' + 'Superior\_PMMA' + ', ' + '"cell"' + ')' + ','

file.write('%s\n' % Top\_PMMA)

Bottom\_PMMA = '"Bottom\_PMMA"' + ': (' + 'Inferior\_PMMA' + ', ' + '"cell"' + ')' + ','

file.write('%s\n' % Bottom\_PMMA)

PMMA = '"PMMA"' + ': (' + '"""r.Top\_PMMA +c r.Bottom\_PMMA"""' + ', ' + '"cell"' + ')' + ','

file.write('%s\n' % PMMA)

for each\_element in Bone\_element:

name = 'Bone' + str(each\_element)

cell = 'cell'

location = 'cell ' + str(each\_element)

string = '"%s"' % name + ': (' + '"%s"' % location + ',' + '"%s"' % cell + ')' + ','

file.write('%s\n' % string)

end = '}'

file.write('%s\n' % end)

Mod = []

file\_in = open(material\_file, 'r')

for y in file\_in.readlines():

Mod.append(float(y))

with open('materials.txt', 'w') as file:

begin = 'materials = {'

file.write('%s\n' % begin)

md = '"m": ({"D": {'

file.write('%s\n' % md)

pmma = '"PMMA": stiffness\_from\_lame(3, 1442, 962),'

file.write('%s\n' % pmma)

difference = Bone\_element[0]

i = 0

for each\_element in Bone\_element:

name = 'Bone' + str(each\_element)

lam = str(0.3 \* Mod[i] / (1.3 \* 0.4))

mu = str(Mod[i] / 2.6)

string = '"%s"' % name + ': ' + 'stiffness\_from\_lame(3, ' + lam + ', ' + mu + '),'

file.write('%s\n' % string)

i = i + 1

end = ' }},),}'

file.write('%s\n' % end)

exec(open("regions.txt").read())

exec(open("materials.txt").read())

os.remove("regions.txt")

os.remove("materials.txt")

fields = {

'displacement': ('real', 'vector', 'Omega', 2),

}

variables = {'u': ('unknown field', 'displacement', 0),

'v': ('test field', 'displacement', 'u'),

}

integrals = {'i': 3,

}

equations = {'balance\_of\_forces':

"""dw\_lin\_elastic.i.Omega(m.D,v,u)

= 0""",

}

ebcs = {'fix\_u': ('inferior', {'u.all': 0.0}),

'shift\_u': ('superior', {'u.0': -0.69})

}

solvers = {'ls': ('ls.scipy\_iterative', {

'i\_max': 200,

'method': 'cg'}),

'nls': ('nls.newton', {}), }

"""

solvers = {'ls': ('ls.scipy\_iterative', {'i\_max': 5,

'eps\_a': 1e-10,

'eps\_r': 1e-10}),

'nls': ('nls.newton', {

'i\_max': 5,

'eps\_a': 1e-10,

'eps\_r': 1e-10}), }

there are different types of linear solvers include

ls.auto\_direct

ls.scipy\_direct

or you may try only nls

'ls': ('ls.scipy\_direct', {}),

"""