```
# ***************************************************************************
# *                                                                         *
# *   Copyright (c) 2019 - Harry van Langen <hvlanalysis@icloud.com>        *
# *                                                                         *
# *   This program is free software; you can redistribute it and/or modify  *
# *   it under the terms of the GNU Lesser General Public License (LGPL)    *
# *   as published by the Free Software Foundation; either version 2 of     *
# *   the License, or (at your option) any later version.                   *
# *   for detail see the LICENCE text file.                                 *
# *                                                                         *
# *   This program is distributed in the hope that it will be useful,       *
# *   but WITHOUT ANY WARRANTY; without even the implied warranty of        *
# *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         *
# *   GNU Library General Public License for more details.                  *
# *                                                                         *
# *   You should have received a copy of the GNU Library General Public     *
# *   License along with this program; if not, write to the Free Software   *
# *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  *
# *   USA                                                                   *
# *                                                                         *
# ***************************************************************************

import sys
import time
import FreeCADGui as Gui
import cProfile
import pstats
from pstats import SortKey

import matplotlib.pyplot as plt

from matplotlib.widgets import Button
import numpy as np


def prn_upd(*args):
    for obj in args:
        print(str(obj), end='')
    print('\n')
    Gui.updateGui()
```

```
prn_upd("fcFEM.FCMacro started")

if 'femTools' in sys.modules.keys():
    del (sys.modules['femTools'])
    prn_upd("femTools in sys.modules.keys")
else:
    sys.path.append("/fcFEM/fcFEM-source")  # put here the path to the fcFEM macros, if not the FreeCAD default
    prn_upd("femTools path added")

import femTools as ft

fmt = "{0:10.3f}"

timer = [["extract information from FreeCAD objects..... " + fmt + " seconds", []],
         ["prepare finite element input..... " + fmt + " seconds", []],
         ["calculate the global stiffness matrix and global load vector... " + fmt + " seconds", []],
         ["solve the global siffness matrix equation..... " + fmt + " seconds", []],
         ["map stresses to nodal points..... " + fmt + " seconds", []],
         ["paste results in the FEM result object..... " + fmt + " seconds", []],
         ["calculate internal load vector..... " + fmt + " seconds", []]]

# material input values - TODO: merge with FEM WB input dialogues
gravity = - 0.0e-06  # specific gravity acting in z-direction [N/mm3] = [1.0e-6 kN/m3]
sig_yield = 10000000.  # yield stress [MPa] for volume elements (von Mises material)
shr_yield = 1.  # yield stress [MPa] for interface elements (Coulomb material)
kn = 1.0   # interface elastic normal stiffness factor (0.0 < kn <1.0)
ks = 0.0   # interface elastic shear stiffness factor (0.0 < ks <1.0)

# control input values - TODO: merge with FEM WB input dialogues
out_disp = -100000  # +n: output total discplacment at step n ; -n: output incremental discplacement at step n (if n>last
step then n=last step)
nstep = 10  # number of load steps per run (default = 10). nstep == 1: elastic analysis
iterat_max = 20  # max number of iterations per step - this triggers a scale-down and restart
error_max = 1.0e-03  # convergence tolerance (default = 1.0e-03)
relax = 1.2  # numerical over-relaxation (1.0 < relax < 1.5; default = 1.2)
scale_re = 2.0  # scale factor for re-start (default = 2.0)
scale_up = 1.2  # scale up for fast convergence (default = 1.2)
scale_dn = 1.2  # scale down for slow convergence (default = 1.2)

# extract information from FreeCAD objects
Gui.updateGui()
t0 = time.time()
```

```
doc, mesh, analysis = ft.setUpAnalysis()
t1 = time.time()
timer[0][1] = t1 - t0
prn_upd("extract information from FreeCAD objects - time taken:  ", t1 - t0, "  seconds\n")

# cProfile.runctx('ft.setUpAnalysis()', globals(), locals())

# prepare finite element input
prn_upd("prepare finite element input\n")
t0 = time.time()
elNodes, noCoord, dispFaces, loadFaces, elMat, interface_elements, noce, pressure, link0, link1, ks_red=
ft.setUpInput(doc, mesh, analysis)
# cProfile.runctx('elNodes, noCoord, dispFaces, loadFaces, elMat, interface_elements, noce, pressure = \
#                     ft.setUpInput(doc, mesh, analysis)', globals(), locals())

t1 = time.time()
timer[1][1] = t1 - t0
prn_upd("prepare finite element input - time taken:  ", t1 - t0, "  seconds\n")

# print("elements containing nodes [13,14,15,16] at entry calcGSM")
# for index, el in enumerate(elNodes):
#     if any(elem in el for elem in [13, 14, 15, 16]):
#         print("element ", index, ":  ",el)

# calculate the global stiffness matrix and global load vector
prn_upd("extract information from FreeCAD objects\n")
t0 = time.time()
globalStiffnessMatrix, globalLoadVector, kmax = ft.calcGSM(elNodes, noCoord, elMat, loadFaces, interface_elements,
                                                           gravity, kn, ks, pressure, link0, link1, ks_red)
# cProfile.runctx('globalStiffnessMatrix, globalLoadVector, kmax = ft.calcGSM(elNodes, noCoord, elMat, loadFaces,
interface_elements, gravity, kn, ks)', globals(), locals())

t1 = time.time()
timer[2][1] = t1 - t0
prn_upd("extract information from FreeCAD objects - time taken:  ", t1 - t0, "  seconds\n")

# solve the global stiffness matrix equation
prn_upd("solve the global stiffness matrix equation\n")
Gui.updateGui()
t0 = time.time()

displacements, stresses, tractions = ft.calcDisp(elNodes, noCoord, dispFaces, elMat, interface_elements, kmax,
```

```
                                                globalStiffnessMatrix, globalLoadVector, nstep, iterat_max,
                                                error_max, relax, scale_re, scale_up, scale_dn, sig_yield, shr_yield,
                                                kn, ks, out_disp, link0, link1,
                                                elMat, loadFaces, gravity, pressure, ks_red)

# cProfile.runctx('displacements, stresses, tractions = ft.calcDisp (elNodes, noCoord, dispFaces, elMat,\
#                                                    interface_elements, kmax,\
#                                                    globalStiffnessMatrix, globalLoadVector, nstep, iterat_max,\
#                                                    error_max, relax, scale_re, scale_up, scale_dn, sig_yield,
shr_yield,\
#                                                    kn, ks, out_disp)', globals(), locals())

t1 = time.time()
timer[3][1] = t1 - t0
prn_upd("solve the global stiffness matrix equation - time taken:  ", t1 - t0, "  seconds\n")

# map stresses to nodal points
prn_upd("map stresses to nodal points\n")
t0 = time.time()
tet10stress, contactpressurevector, contactpressurevalue, contactshearvector = ft.mapStresses(elNodes, noCoord,
                                                                                              interface_elements,
                                                                                              displacements, stresses,
                                                                                              tractions, noce)
t1 = time.time()
timer[4][1] = t1 - t0
prn_upd("map stresses to nodal points - time taken:  ", t1 - t0, "  seconds\n")

# paste results in the FEM result object
prn_upd("paste results in the FEM result object\n")
t0 = time.time()
resInt, resVol = ft.pasteResults(doc, elNodes, noCoord, interface_elements, displacements, tet10stress,
                                 contactpressurevector, contactpressurevalue, contactshearvector)
t1 = time.time()
timer[5][1] = t1 - t0
prn_upd("paste results in the FEM result object - time taken:  ", t1 - t0, "  seconds\n")

# Calculate internal loads and residual loads
prn_upd("Calculate internal loads and residual loads\n")
t0 = time.time()
t1 = time.time()
timer[6][1] = t1 - t0
```

```
prn_upd("------------------------------ SUMMARY ------------------------------")
for entry in timer:
    prn_upd(entry[0].format(entry[1]))
```