```
#-----------------------------------------------------------------------------
#
# Signed Distances Function Interpolator
# **************************************
#
# This SGeMS plugin interpolates (OK) the signed distance function calculated
# for each data and rock type, and creates a geologic model based on the minimum
# estimated distance.
#
# AUTHOR: Roberto Mentzingen Rolo
#
#-----------------------------------------------------------------------------

#!/bin/python
import sgems
import math
import numpy as np
import random
import copy

#Creates a randon path given the size of the grid
def random_path(prop):
    nodes_not_nan = []
    for i in range(len(prop)):
        if not math.isnan(prop[i]):
            nodes_not_nan.append(i)
    random.shuffle(nodes_not_nan)
    return nodes_not_nan

#Calculates the proportions of variables on a list
def proportion(var, RT):
    rock_types =[]
    target_prop = []
    for k in range(len(RT)):
        target_prop.append(0)
        rock_types.append(int(RT[k][-1]))
    rock_types.sort()
    var_not_nan = []
    for i in var:
        if not math.isnan(i):
            var_not_nan.append(i)
    for i in range(len(rock_types)):
        target_prop[i] = float(var.count(rock_types[i]))/len(var_not_nan)
    return target_prop

#Transform i,j,k in n
def ijk_in_n(grid, i, j, k):
    dims = sgems.get_dims(grid)
    n = k*dims[0]*dims[1]+j*dims[0]+i
    return n

#Crestes a list with indices of the neighbors valid blocks
def neighb(grid, indice):
        ijk = sgems.get_ijk(grid, indice)
        neighborhood = []
        for i in range(ijk[0]-1,ijk[0]+2):
            for j in range(ijk[1]-1,ijk[1]+2):
                for k in range(ijk[2]-1,ijk[2]+2):
                    ijk_blk = [i,j,k]
                    neighborhood.append(ijk_blk)
        dims = sgems.get_dims(grid)
        neighborhood_cp = copy.copy(neighborhood)
        for i in neighborhood_cp:
            if dims[2] == 1:
                if i[0] < 0 or i[1] < 0:
                    neighborhood.remove(i)
```

```
                elif i[0] > (dims[0] - 1) or i[1] > (dims[1] - 1):
                    neighborhood.remove(i)
                elif i[2] != 0:
                    neighborhood.remove(i)
                elif i == sgems.get_ijk(grid, indice):
                    neighborhood.remove(i)
            else:
                if i[0] < 0 or i[1] < 0 or i[2] < 0:
                    neighborhood.remove(i)
                elif i[0] > (dims[0] - 1) or i[1] > (dims[1] - 1) or i[2] > (dims[2] - 1):

                    neighborhood.remove(i)
                elif i == sgems.get_ijk(grid, indice):
                    neighborhood.remove(i)
        neighborhood_n = []
        for i in neighborhood:
            neighborhood_n.append(ijk_in_n(grid,i[0],i[1],i[2]))
        return neighborhood_n


# Shows every parameter of the plugin in the command pannel
def read_params(a, j=''):
    for i in a:
        if (type(a[i]) != type({'a': 1})):
            print j + "['" + str(i) + "']=" + str(a[i])
        else:
            read_params(a[i], j + "['" + str(i) + "']")

class interpolator:
    def __init__(self):
        pass

    def initialize(self, params):
        self.params = params
               return True

           def execute(self):

               '''''# Execute the funtion read_params
               read_params(self.params)
               print self.params'''

               #Get the grid and rock type propery
               grid = self.params['propertyselectornoregion']['grid']
               prop = self.params['propertyselectornoregion']['property']

               #Get the X, Y and Z coordinates and RT property
               X = sgems.get_property(grid, '_X_')
               Y = sgems.get_property(grid, '_Y_')
               Z = sgems.get_property(grid, '_Z_')
               RT_data = sgems.get_property(grid, prop)

               # Getting properties
               grid_krig = self.params['gridselectorbasic_2']['value']
               grid_var = self.params['gridselectorbasic']['value']
               props = (self.params['orderedpropertyselector']['value']).split(';')
               n_var = int(self.params['indicator_regionalization_input']['number_of_indic
   ator_group'])
               n_prop = int(self.params['orderedpropertyselector']['count'])
               min_cond = self.params['spinBox_2']['value']
               max_cond = self.params['spinBox']['value']

               # Error messages
               if len(grid_var) == 0 or len(grid_krig) == 0:
                   print 'Select the variables'
                   return False

               if n_var != n_prop:
```

```
133.                     print 'Number of variables and number of variograms models are diferent
    .'
134.                     return False
135.
136.                 #Creating an empty list to store the interpolated distances
137.                 SG_OK_list = []
138.
139.                 # Loop in every variable
140.                 for i in xrange(0, n_var):
141.
142.                     # Getting variables
143.                     prop_HD = props[i]
144.                     prop_name = "Interpolated_" + str(prop_HD)
145.                     prop_name_var = "Interpolated_" + str(prop_HD) + ' krig_var'
146.                     var_str = ''
147.                     indicator_group = "Indicator_group_" + str(i + 1)
148.                     elipsoide = self.params['ellipsoidinput']['value']
149.                     n_struct = int(self.params['indicator_regionalization_input'][indicator
    _group]['Covariance_input']['structures_count'])
150.
151.                     # Error message
152.                     if n_struct == 0:
153.                         print 'Variogram have no structures'
154.                         return False
155.
156.                     # Loop in every variogram structure
157.                     for j in xrange(0, n_struct):
158.                         # Getting variogram parameters
159.                         Structure = "Structure_" + str(j + 1)
160.
161.                         cov_type = self.params['indicator_regionalization_input'][indicator
    _group]['Covariance_input'][Structure]['Two_point_model']['type']
162.
163.                         cont = self.params['indicator_regionalization_input'][indicator_gro
    up]['Covariance_input'][Structure]['Two_point_model']['contribution']
164.
165.                         if cov_type == 'Nugget Covariance':
166.                             #Writing variogram parameters on a variable in nugget effect ca
    se
167.                             var_str = var_str + '<{} type="{}">  <Two_point_model  contribu
    tion="{}"  type="{}"   >    </Two_point_model>    </Structure_1> '.format(Structure, 'Cova
    riance', cont, cov_type, Structure)
168.
169.                         else:
170.                             range1 = self.params['indicator_regionalization_input'][indicat
    or_group]['Covariance_input'][Structure]['Two_point_model']['ranges']['range1']
171.                             range2 = self.params['indicator_regionalization_input'][indicat
    or_group]['Covariance_input'][Structure]['Two_point_model']['ranges']['range2']
172.                             range3 = self.params['indicator_regionalization_input'][indicat
    or_group]['Covariance_input'][Structure]['Two_point_model']['ranges']['range3']
173.
174.                             rake = self.params['indicator_regionalization_input'][indicator
    _group]['Covariance_input'][Structure]['Two_point_model']['angles']['rake']
175.                             dip = self.params['indicator_regionalization_input'][indicator_
    group]['Covariance_input'][Structure]['Two_point_model']['angles']['dip']
176.                             azimuth = self.params['indicator_regionalization_input'][indica
    tor_group]['Covariance_input'][Structure]['Two_point_model']['angles']['azimuth']
177.
178.                             # Writing variogram parameters on a variable in other cases
179.                             var_str = var_str + '<{} type="{}">  <Two_point_model  contribu
    tion="{}"  type="{}"   >      <ranges range1="{}"  range2="{}"  range3="{}"   />      <ang
    les azimuth="{}"  dip="{}"  rake="{}"   />    </Two_point_model>    </{}> '.format(Structu
    re, 'Covariance', cont, cov_type, range1, range2, range3, azimuth, dip, rake, Structure)
180.
181.                     # Calling ordinary kriging for each variable, using the variograms para
    meters above
182.                     sgems.execute('RunGeostatAlgorithm  kriging::/GeostatParamUtils/XML::<p
    arameters>  <algorithm name="kriging" />     <Variogram  structures_count="{}" >    {}  </
    Variogram>    <ouput_kriging_variance  value="1"  />     <output_n_samples_  value="0"  />
```

```
        <output_average_distance value="0" />     <output_sum_weights value="0" />     <o
    utput_sum_positive_weights value="0" />     <output_lagrangian value="0" />     <Nb_pr
    ocessors value="-
    2" />    <Grid_Name value="{}" region="" />     <Property_Name value="{}" />     <Hard_
    Data grid="{}"   property="{}"   region="" />     <Kriging_Type type="Ordinary Kriging
    (OK)" >    <parameters /> </Kriging_Type>    <do_block_kriging value="1" />     <npoint
    s_x value="5" />     <npoints_y value="5" />     <npoints_z value="5" />     <Min_Condi
    tioning_Data value="{}" />     <Max_Conditioning_Data value="{}" />     <Search_Ellipsoi
    d value="{}" />    <AdvancedSearch use_advanced_search="0"></AdvancedSearch> </paramete
    rs>'.format(n_struct, var_str, grid_krig, prop_name, grid_var, prop_HD, min_cond, max_cond
    , elipsoide))
183.
184.                     SG_OK_list.append(sgems.get_property(grid_krig, prop_name))
185.
186.                     #Deleting kriged distances
187.                     sgems.execute('DeleteObjectProperties  {}::{}'.format(grid_krig, prop_n
    ame))
188.                     sgems.execute('DeleteObjectProperties  {}::{}'.format(grid_krig, prop_n
    ame_var))
189.
190.                 RT = (self.params['orderedpropertyselector']['value']).split(';')
191.
192.                 #Determinig geomodel based on minimum estimed signed distance function
193.                 GeoModel = SG_OK_list[0][:]
194.
195.                 t = 0
196.                 for i in range(len(SG_OK_list[0])):
197.                     sgmin = 10e21
198.                     for j in range(len(SG_OK_list)):
199.                         if SG_OK_list[j][i] < sgmin:
200.                             sgmin = SG_OK_list[j][i]
201.                             t = j
202.                     if math.isnan(SG_OK_list[j][i]):
203.                         GeoModel[i] = float('nan')
204.                     else:
205.                         GeoModel[i] = (int(RT[t][-1]))
206.
207.                 #Creating GeoModel property
208.                 lst_props_grid=sgems.get_property_list(grid_krig)
209.                 prop_final_data_name = 'Geologic_Model'
210.
211.                 if (prop_final_data_name in lst_props_grid):
212.                     flag=0
213.                     i=1
214.                     while (flag==0):
215.                         test_name=prop_final_data_name+'-'+str(i)
216.                         if (test_name not in lst_props_grid):
217.                             flag=1
218.                             prop_final_data_name=test_name
219.                         i=i+1
220.
221.                 #Assign conditioning data to grid node
222.                 for i in range(len(RT_data)):
223.                     if not math.isnan(RT_data[i]):
224.                         closest_node = sgems.get_closest_nodeid(grid_krig, X[i],Y[i],Z[i])
225.                         GeoModel[closest_node] = RT_data[i]
226.
227.                 sgems.set_property(grid_krig, prop_final_data_name, GeoModel)
228.
229.                 #Operating softmax transformation
230.                 if self.params['softmax_check']['value']=='1':
231.
232.                     gamma =float( self.params['Gamma']['value'])
233.                     Prob_list = SG_OK_list[:]
234.
235.                     for i in range(len(SG_OK_list[0])):
236.                         soma = 0
237.                         for j in range(len(SG_OK_list)):
```

```
                            soma = soma + math.exp(-SG_OK_list[j][i]/gamma)
                        for j in range(len(SG_OK_list)):
                            Prob_list[j][i] = math.exp(-SG_OK_list[j][i]/gamma)/soma

                #Creating probabilities propreties
                for k in range(len(Prob_list)):
                    prop_final_data_name = 'Probability_RT'+str(RT[k][-1])

                    if (prop_final_data_name in lst_props_grid):
                        flag=0
                        i=1
                        while (flag==0):
                            test_name=prop_final_data_name+'-'+str(i)
                            if (test_name not in lst_props_grid):
                                flag=1
                                prop_final_data_name=test_name
                            i=i+1

                    sgems.set_property(grid_krig, prop_final_data_name, Prob_list[k])

                #Operating servo-system
                if self.params['servo_check']['value'] == '1':
                    var_rt_grid = self.params['targe_prop']['grid']
                    var_rt_st = self.params['targe_prop']['property']
                    var_rt_region = self.params['targe_prop']['region']
                    if len(var_rt_grid) == 0 or len(var_rt_st) == 0:
                        print 'Select the target proportion property'
                        return False

                    #Getting variables
                    var_rt = sgems.get_property(var_rt_grid, var_rt_st)

                    #Getting parameters
                    lambda1 = float(self.params['Lambda']['value'])
                    mi = lambda1/(1-lambda1)

                    #Checking if a region exist
                    if len(var_rt_region) == 0:
                        #Variable without a region
                        var_region = var_rt

                    else:
                        region_rt = sgems.get_region(var_rt_grid, var_rt_region)
                        #Geting the variable inside the region
                        var_region = []
                        for i in range(len(var_rt)):
                            if region_rt[i] == 1:
                                var_region.append(var_rt[i])

                    #Getting the target proportion
                    target_prop = proportion(var_region, RT)

                    #Getting the random path
                    ran_path = random_path(Prob_list[0])

                    #Removing the blocks outside the region from randon path
                    if len(var_rt_region) != 0:
                        for i in range(len(region_rt)):
                            if region_rt[i] == 0:
                                ran_path.remove(i)

                    #servo system
                    p = 0
                    GeoModel_corrected = GeoModel[:]

                    visited_rts = []
                    for j in ran_path:
                        visited_rts.append(GeoModel[j])
                        instant_proportions = proportion(visited_rts,RT)
```

```
307.
308.                             sgmax = 10e-21
309.                             for i in range(len(Prob_list)):
310.                                 Prob_list[i][j] = Prob_list[i][j] + (mi * (target_prop[i] -
     instant_proportions[i]))
311.                                 if Prob_list[i][j] > sgmax:
312.                                     sgmax = Prob_list[i][j]
313.                                     p = i
314.
315.                             GeoModel_corrected[j] = int(RT[p][-1])
316.                             visited_rts[-1] = int(RT[p][-1])
317.
318.                         #Correcting servo servo-
    system by the biggest proportion on a neighborhood
319.                         GeoModel_corrected_servo_prop = GeoModel_corrected[:]
320.                         ran_path_servo_correction = random_path(GeoModel_corrected_servo_pr
    op)
321.                         for i in ran_path_servo_correction:
322.                             vizinhanca = neighb(grid_krig,i)
323.
324.                             blk_geo_model_corrected_servo = []
325.                             for j in vizinhanca:
326.                                 blk_geo_model_corrected_servo.append(GeoModel_corrected_ser
    vo_prop[j])
327.
328.                             proportions_servo = proportion(blk_geo_model_corrected_servo, R
    T)
329.                             indice_max_prop = proportions_servo.index(max(proportions_servo
    ))
330.
331.                             GeoModel_corrected_servo_prop[i] = int(RT[indice_max_prop][-
    1])
332.
333.                         #Creating Geologic_Model_Servo_System property
334.                         prop_final_data_name = 'Geologic_Model_Servo_System'
335.
336.                         if (prop_final_data_name in lst_props_grid):
337.                             flag=0
338.                             i=1
339.                             while (flag==0):
340.                                 test_name=prop_final_data_name+'-'+str(i)
341.                                 if (test_name not in lst_props_grid):
342.                                     flag=1
343.                                     prop_final_data_name=test_name
344.                                 i=i+1
345.
346.                         #Creating Geologic_Model_Corrected property
347.                         prop_final_data_name1 = 'Geologic_Model_Corrected'
348.
349.                         if (prop_final_data_name1 in lst_props_grid):
350.                             flag=0
351.                             i=1
352.                             while (flag==0):
353.                                 test_name1=prop_final_data_name1+'-'+str(i)
354.                                 if (test_name1 not in lst_props_grid):
355.                                     flag=1
356.                                     prop_final_data_name1=test_name1
357.                                 i=i+1
358.
359.                         #Assign conditioning data to grid node
360.                         for i in range(len(RT_data)):
361.                             if not math.isnan(RT_data[i]):
362.                                 closest_node = sgems.get_closest_nodeid(grid_krig, X[i],Y[i
    ],Z[i])
363.                                 GeoModel_corrected[closest_node] = RT_data[i]
364.                                 GeoModel_corrected_servo_prop[closest_node] = RT_data[i]
365.
366.                         #Setting properties
```

```
                    sgems.set_property(grid_krig, prop_final_data_name, GeoModel_corrected)
                    sgems.set_property(grid_krig, prop_final_data_name1, GeoModel_corrected_servo_prop)

                return True

            def finalize(self):

                return True

            def name(self):

                return "interpolator"

        ##############################################################################
        def get_plugins():
            return ["interpolator"]
```