

```

1 """
2     @Author: Margarita Kuvaldina
3     @https://github.com/margkuval
4     @date: May 2018
5 """
6
7 """SOLVER FOR ANY TRUSS IN GENETIC ALGORITHM - deflections, stress, weight"""
8
9 import numpy as np
10
11
12 def deflection(xcoord, ycoord, mem_begin, mem_end, numelem, E, A, F, dof):
13
14     "Link x and y coordinates with member's beginning and end"""
15     xi = xcoord[np.ix_(mem_begin)]
16     xj = xcoord[np.ix_(mem_end)] # take mem_end numbers and replace them with corresponding xcoord
17     yi = ycoord[np.ix_(mem_begin)]
18     yj = ycoord[np.ix_(mem_end)]
19
20     "Connectivity matrix"
21     ij = np.vstack([[2 * mem_begin, 2 * mem_begin + 1], [2 * mem_end, 2 * mem_end + 1]]).transpose()
22
23     "Other information"
24     numnode = xcoord.shape[0] # number of nodes, xcoord because all nodes are used
25     dof_tot = 2 * numnode # total degrees of freedom
26
27     """Global Stiffness Matrix"""
28     glob_stif = np.zeros((dof_tot, dof_tot)) # empty Global Stiffness Matrix
29     length = np.sqrt(pow((xj - xi), 2) + pow((yj - yi), 2)) # defines length of members
30     c = (xj - xi) / length # cos
31     s = (yj - yi) / length # sin
32
33     for p in range(numelem):
34         # takes p from the range of numelem 1 by 1 and creates multiple k1 (local) matrices
35         # at the end maps k1 matrices on right places in glob_stif matrix
36         n = ij[p]
37         cc = c[p] * c[p]
38         cs = c[p] * s[p]
39         ss = s[p] * s[p]
40         k1 = E[p] * A[p] / length[p] * np.array([[cc, cs, -cc, -cs],
41                                                 [cs, ss, -cs, -ss],
42                                                 [-cc, -cs, cc, cs],
43                                                 [-cs, -ss, cs, ss]])
44         glob_stif[np.ix_(n, n)] += k1
45
46     """Forces and deflections"""
47
48     "Fixed and active DOFs"
49     dof_a = np.array(np.where(dof == 0)) # node where dof = 0 is an active node
50     dof_active = dof_a[0]
51
52     "Solve deflections"
53     u = np.zeros((dof_tot, 1)) # empty deflections matrix; 1 = number of columns
54     u1 = np.linalg.solve(glob_stif[np.ix_(dof_active, dof_active)],
55                          F[np.ix_(dof_active)]) # solve equation glob_stif*u = F
56     u[np.ix_(dof_active)] = u1 # map back to the empty deflection matrix
57     deflection = np.round(u, 3)
58
59     return deflection
60
61
62 def stress(xcoord, ycoord, mem_begin, mem_end, E, A, F, dof, deflection):
63
64     "Link x and y coordinates with member's beginning and end"""
65     xi = xcoord[np.ix_(mem_begin)]
66     xj = xcoord[np.ix_(mem_end)]
67     yi = ycoord[np.ix_(mem_begin)]
68     yj = ycoord[np.ix_(mem_end)]
69
70     "Other information"
71     numnode = xcoord.shape[0]
72
73     "Reshaped outside forces and DOFs"
74     F_x2 = F.reshape(numnode, 2)
75     dof_x2 = dof.reshape(numnode, 2) # reshape for plotting
76
77     """Stress calculation"""
78
79     "Deflections in x, y directions"
80     # using mem_end and mem_begin to calculate new nodes location
81     length = np.sqrt(pow((xj - xi), 2) + pow((yj - yi), 2)) # members length
82     k = E * A / length
83
84     u = deflection
85     uxi = u[np.ix_(2 * mem_begin)].transpose()
86     uxj = u[np.ix_(2 * mem_end)].transpose()
87     uyi = u[np.ix_(2 * mem_begin + 1)].transpose()
88     uyj = u[np.ix_(2 * mem_end + 1)].transpose()
89
90     "Inner forces"

```

File - C:\Personal stuff\1. Skola\10. semestr - Spring 18\BP\repository\Thesis\python_projects\solver_univ.py

```
91     c = (xj - xi) / length # cos
92     s = (yj - yi) / length # sin
93
94     Flocal = k * ((uxj - uxi) * c + (uyj - uyi) * s)
95
96     "Stress (kPa)"
97     stress = Flocal[0] / A
98     stress_normed = [i / sum(abs(stress)) for i in abs(stress)]
99
100    xinew = xi + uxi[0] # [[ in u array, now solved by taking "list 0" from the MAT
101    xjnew = xj + uxj[0]
102    yinew = yi + uyi[0]
103    yjnew = yj + uyj[0]
104
105    return stress, stress_normed, xi, xj, yi, yj, xinew, xjnew, yinew, yjnew, F_x2, numnode, dof_x2, length
106
107
108 def weight(A, length, ro):
109
110     "Density of each element (1000 kg/m3)"
111     # reinforced concrete = 2500 kg/m3, steel = 7700 kg/m3
112     # defined in GA
113
114     "Weigth calculation"
115     weight = length * A * ro
116
117     return weight
```

```

1 """
2     @Author: Margarita Kuvaldina
3     @https://github.com/margkuval
4     @date: May 2018
5 """
6
7 import numpy as np
8 import random as rnd
9 import matplotlib.pyplot as plt
10 import solver_univ as slv
11 from matplotlib.gridspec import GridSpec
12 import datetime
13
14 """GENETIC ALGORITHM 1x4 TRUSS"""
15
16 # CH = change if implementing on a new structure
17
18
19 class Individual:
20     def __init__(self):
21         "Structural dimentions (m)"""
22         # CH = change if implementing on a new structure
23         a = 2 # CH
24         h = a # triangle height # CH
25
26         "Original coordinates"
27         xcoord = np.array([0, a, 2.5 * a, 4 * a, 5 * a, a, 2.5 * a, 4 * a]) # CH
28         ycoord = np.array([0, 0, 0, 0, 0, h, h, h]) # CH
29
30         "Choose a random number in range +- (m) from the original coordinate"
31         x2GA = rnd.randrange(np.round((xcoord[2] - 0.7) * 100), np.round((xcoord[2] + 0.7) * 100)) / 100 # CH
32         x1GA = rnd.randrange(np.round((xcoord[1] - 0.7) * 100), np.round((xcoord[1] + 0.7) * 100)) / 100
33         x3GA = rnd.randrange(np.round((xcoord[3] - 0.7) * 100), np.round((xcoord[3] + 0.7) * 100)) / 100
34
35         "New coordinates"
36         xcoord = np.array([0, x1GA, x2GA, x3GA, 5 * a, a, 2.5 * a, 4 * a]) # CH
37         ycoord = np.array([0, 0, 0, 0, 0, h, h, h]) # can use np.ix_? # CH
38
39         "Cross-section area (m)"
40         self.A = np.random.uniform(low=0.0144, high=0.0529, size=(13,)) # area between 12x12 and 23x23cm # CH
41         self.A[0] = rnd.randrange(0.0004 * 10000, 0.0064 * 10000) / 10000 # special condition for steel elements # CH
42         self.A[1] = rnd.randrange(0.0004 * 10000, 0.0064 * 10000) / 10000
43         self.A[2] = rnd.randrange(0.0004 * 10000, 0.0064 * 10000) / 10000
44         self.A[3] = rnd.randrange(0.0004 * 10000, 0.0064 * 10000) / 10000
45         self.A[11] = rnd.randrange(0.0004 * 10000, 0.0064 * 10000) / 10000
46
47         "Material characteristic E=(MPa), ro=kg/m3" # CH
48         # modulus of elasticity for each member, E_reinforced_concrete C30/37 = 33 000 MPa, E_steel S235 = 210 000 MPa
49         self.E = np.array([33000, 33000, 33000, 33000, 33000, 33000, 33000, 33000, 33000, 33000, 33000, 33000, 33000])
50         self.E[0] = 210000
51         self.E[1] = 210000
52         self.E[2] = 210000
53         self.E[3] = 210000
54         self.E[11] = 210000
55         self.E[12] = 210000
56
57         # density for each member, ro_reinforced_concrete C30/37 = 2 600 kg/m3, ro_steel S235= 7850 kg/m3
58         self.ro = np.array([2600, 2600, 2600, 2600, 2600, 2600, 2600, 2600, 2600, 2600, 2600, 2600, 2600])/1000
59         self.ro[0] = 7850
60         self.ro[1] = 7850
61         self.ro[2] = 7850
62         self.ro[3] = 7850
63         self.ro[11] = 7850
64         self.ro[12] = 7850
65
66         self._plot_dict = None
67         self._nodes = np.array([xcoord, ycoord])
68         self._deflection = 0
69         self._stress = 0
70         self._weight = 0
71         self._fitness = 0
72         self._probability = 0
73
74     @property
75     def deflection(self):
76         return self._deflection
77
78     @deflection.setter
79     def deflection(self, new):
80         self._deflection = new
81
82     @property
83     def stress(self):
84         return self._stress
85
86     @stress.setter
87     def stress(self, new):
88         self._stress = new
89
90     @property

```

```

91     def weight(self):
92         return self._weight
93
94     @weight.setter
95     def weight(self, new):
96         self._weight = new
97
98     @property
99     def fitness(self):
100         return self._fitness
101
102     @fitness.setter
103     def fitness(self, new):
104         self._fitness = new
105
106     @property
107     def probability(self):
108         return self._probability
109
110     @probability.setter
111     def probability(self, new):
112         self._probability = new
113
114
115 class GA:
116     def __init__(self, pop):
117         self.mem_begin = np.array([0, 1, 2, 3, 4, 7, 6, 5, 1, 5, 6, 2, 7]) # beginning of an edge # CH
118         self.mem_end = np.array([1, 2, 3, 4, 7, 6, 5, 0, 5, 2, 2, 7, 3]) # end of an edge # CH
119
120         self._pool = list()
121         self._popsize = pop
122
123     def initial(self):
124         for i in range(self._popsize):
125             self._pool.append(Individual())
126             print("node_1:{} node_2:{} node_3:{}".format(
127                 np.round([self._pool[i]._nodes[0, 1], self._pool[i]._nodes[1, 1]], 3),
128                 np.round([self._pool[i]._nodes[0, 2], self._pool[i]._nodes[1, 2]], 3),
129                 np.round([self._pool[i]._nodes[0, 3], self._pool[i]._nodes[1, 3]], 3)
130             ))
131
132     def calculation(self):
133         numelem = len(self.mem_begin) # count number of beginnings
134
135         """Structural characteristics"""
136
137         "Fixed Degrees of Freedom (DOF)" # CH
138         dof = np.zeros((2 * len(np.unique(self.mem_begin)), 1)) # dof vector # counts unique values in mem_begin
139         dof[0] = 1 # 1 = fixed
140         dof[1] = 1
141         dof[9] = 1
142
143         "Outside Forces [kN]" # CH
144         F = np.zeros((2 * len(np.unique(self.mem_begin)), 1)) # forces vector # counts unique values in mem_begin
145         F[10] = 10
146         F[11] = -15
147         F[13] = -5
148         F[14] = 10
149         F[15] = -15
150
151         print("calculation ")
152
153         """Access solver"""
154         for i in range(self._popsize):
155
156             "DEFLECTION"
157             pool = self._pool[i]
158             res = slv.deflection(pool._nodes[0], pool._nodes[1], self.mem_begin, self.mem_end, numelem,
159                               pool.E, pool.A, F, dof)
160
161             deflection = res
162             pool._deflection = deflection
163             pool._probability = 0
164
165             "STRESS"
166             # globbing, to "res" save everything that slv.stress returns (tuple)
167             res = slv.stress(pool._nodes[0], pool._nodes[1],
168                             self.mem_begin, self.mem_end,
169                             pool.E, pool.A, F, dof, deflection)
170             stress, stress_normed, xi, xj, yi, yj, xinew, xjnew, yinew, yjnew, F_x2, numnode, dof_x2, length = res
171             pool._stress = stress
172             pool._stress_normed = stress_normed
173             pool._stress_max = np.round(np.max(pool._stress), 3)
174
175             plot_dict = {"xi": xi, "xj": xj, "yi": yi, "yj": yj, "xinew": xinew, "xjnew": xjnew, "yinew": yinew,
176                         "yjnew": yjnew, "F_x2": F_x2, "dof_x2": dof_x2, "stress_normed": stress_normed,
177                         "numnode": numnode, "numelem": numelem, "A": pool.A}
178             pool._plot_dict = plot_dict
179
180             "WEIGHT"

```

```

180         pool._weight = slv.weight(pool.A, length, pool.ro)
181
182     print("node_1:{} node_2:{} node_3:{} |def| sum:{} |stress| sum:{} |weight| sum:{}".format(
183         np.round([pool._nodes[0, 1], pool._nodes[1, 1]], 3),
184         np.round([pool._nodes[0, 2], pool._nodes[1, 2]], 3),
185         np.round([pool._nodes[0, 3], pool._nodes[1, 3]], 3),
186         np.round(abs(pool._deflection).sum(), 3),
187         np.round(abs(pool._stress).sum()),
188         np.round(pool._weight.sum()))
189     print(".....")
190
191
192 def fitness(self):
193     fitnesses = []
194
195     "Condition to disadvantage members with stress > E"
196     # if the inner force is higher than member's strength, make its fitness worse
197     for x in self._pool:
198         for i in range(len(self.mem_begin)):
199             for strength in self._pool[i].E:
200                 if strength < abs(x._stress[i]):
201                     x._stress[i] = x._stress[i] * 200
202
203     """Rate / give fitness to each member"""
204     print("fitness") # higher fitness = better fitness
205
206     "Conditions to find the best candidate"
207     deflections = [(abs(x._deflection)).sum() for x in self._pool] # sum of absolute deflections
208     stresses = [sum(abs(x._stress)).sum() for x in self._pool] # sum of absolute stresses
209     weights = [sum(x._weight).sum() for x in self._pool] # sum of weights
210
211     "Importance coefficients for stated conditions"
212     deflection_coef = 0.40
213     stress_coef = 0.50
214     weight_coef = 0.10
215
216     "Fill the cell based on conditions and importance coefficients"
217     for deflection, stress, weight in zip(deflections, stresses, weights):
218         if weight.sum() < 0:
219             print("Weight is negative!")
220         else:
221             fitnesses.append(1/(deflection_coef * deflection +
222                 stress_coef * stress * (min(deflections)/min(stresses)) +
223                 weight_coef * weight * (min(deflections)/min(weights))))
224     sum_fit = sum(fitnesses)
225
226     "Fitness of each candidate"
227     len_sf = len(self._pool)
228     for i in range(len_sf):
229         self._pool[i]._fitness = fitnesses[i]
230
231     "Probability record"
232     # member with higher fit (= better) has a higher probability to be chosen for the crossover
233     self._pool[i]._probability = fitnesses[i] / sum_fit
234
235     "Sort members based on probability"
236     # sorting in Python is ascending (if "-x._fitness", is descending)
237     self._pool.sort(key=lambda x: -x._fitness)
238
239     "Print results"
240     for i in range(self._popsize):
241         pool = self._pool[i]
242         print("node_1:{} node_2:{} node_3:{} fit:{} prob:{}".format(
243             "|def(mm)| sum:{} |stress(kPa)| sum:{} |weight(t)| sum:{}".format(
244                 np.round([pool._nodes[0, 1], pool._nodes[1, 1]], 3),
245                 np.round([pool._nodes[0, 2], pool._nodes[1, 2]], 3),
246                 np.round([pool._nodes[0, 3], pool._nodes[1, 3]], 3),
247                 np.round(pool._fitness, 3),
248                 np.round(pool._probability, 3),
249                 np.round(abs(pool._deflection).sum(), 3),
250                 np.round(abs(pool._stress).sum()),
251                 np.round(pool._weight.sum()))
252
253     print("_____")
254
255 def get_best_fit(self):
256     "Best fitness"
257     best_obj= max(self._pool, key=lambda x: x._fitness)
258     return np.round(best_obj.fitness, 3)
259
260 def get_best_weight(self):
261     "Best fitness - weight"
262     best_obj= max(self._pool, key=lambda x: x._fitness)
263     return np.round(sum(best_obj.weight), 3)
264
265 def get_best_stress(self):
266     "Best fitness - sum stress"
267     best_obj= max(self._pool, key=lambda x: x._fitness)
268     return np.round(sum(abs(best_obj.stress)), 3)
269

```

```

270 def get_best_stress_negative(self):
271     "Best fitness - negative stress"
272     best_obj = max(self._pool, key=lambda x: x._fitness)
273     best_stress_negative_sum = 0
274     for num in best_obj.stress:
275         if num < 0:
276             best_stress_negative_sum += num
277     return np.round(best_stress_negative_sum, 3)
278
279 def get_best_stress_positive(self):
280     "Best fitness - positive stress"
281     best_obj = max(self._pool, key=lambda x: x._fitness)
282     best_stress_positive_sum = 0
283     for num in best_obj.stress:
284         if num > 0:
285             best_stress_positive_sum += num
286     return np.round(best_stress_positive_sum, 3)
287
288 def get_best_defl(self):
289     "Best fitness - deflections"
290     best_obj = max(self._pool, key=lambda x: x._fitness)
291     return best_obj.deflection
292
293 """def get_avg_fit(self):
294     "Best fitness"
295     best_obj = np.average(self._pool, key=lambda x: x._fitness)
296     return np.round(best_obj.fitness, 3)
297
298 def get_best_weight(self):
299     "Best fitness - weight"
300     best_obj = max(self._pool, key=lambda x: x._fitness)
301     return np.round(sum(best_obj.weight), 3)
302
303 def get_best_stress(self):
304     "Best fitness - stress"
305     best_obj = max(self._pool, key=lambda x: x._fitness)
306     return np.round(sum(abs(best_obj.stress)), 3)
307
308 def get_best_defl(self):
309     "Best fitness - deflections"
310     best_obj = max(self._pool, key=lambda x: x._fitness)
311     return best_obj.deflection"""
312
313 def _switch1(self, individual_pair, axis=0):
314     # set switch values between 2 individuals (node 1)
315     first = individual_pair[0]
316     second = individual_pair[1]
317     tmp = first._nodes[axis, 1] # = temporary
318     first._nodes[axis, 1] = second._nodes[axis, 1]
319     second._nodes[axis, 1] = tmp
320
321 def _switch2(self, individual_pair, axis=0):
322     # set switch values between 2 individuals (node 2)
323     first = individual_pair[0]
324     second = individual_pair[1]
325     tmp = first._nodes[axis, 2] # temporary
326     first._nodes[axis, 2] = second._nodes[axis, 2]
327     second._nodes[axis, 2] = tmp
328
329 def _switch3(self, individual_pair, axis=0):
330     # set switch values between 2 individuals (node 3)
331     first = individual_pair[0]
332     second = individual_pair[1]
333     tmp = first._nodes[axis, 3] # temporary
334     first._nodes[axis, 3] = second._nodes[axis, 3]
335     second._nodes[axis, 3] = tmp
336
337 def _switch_A(self, individual_pair, axis=0):
338     # set switch values between 2 individuals (node 3)
339     first = individual_pair[0]
340     second = individual_pair[1]
341     tmp = first._nodes[axis, 0] # temporary
342     first._nodes[axis, 0] = second._nodes[axis, 0]
343     second._nodes[axis, 0] = tmp
344
345 def crossover(self):
346     probs = ([x._probability for x in self._pool])
347
348     "Select the best and worst member"
349     best = max(self._pool, key=lambda x: x._fitness)
350     worst = min(self._pool, key=lambda x: x._probability)
351
352     "Nodes crossover"
353     # choose 2 individuals that will crossover
354     switch_x0 = np.random.choice(self._pool, 2, replace=False, p=probs)
355     switch_x1 = np.random.choice(self._pool, 2, replace=False, p=probs)
356     switch_x2 = np.random.choice(self._pool, 2, replace=False, p=probs)
357     #switch_y = np.random.choice(self._pool, 2, replace=False, p=probs)
358
359     "Areas Crossover"

```

```

360     switch_a = np.random.choice(self._pool, 2, replace=False, p=probs)
361
362     self._switch1(switch_x0, 0)
363     self._switch2(switch_x1, 0)
364     self._switch3(switch_x2, 0)
365     self._switch_A(switch_a, 0)
366     #self._switch2(switch_y, 1)
367
368     self._pool.remove(worst)
369     self._pool.append(best)
370
371 def mutation(self, mutation_type):
372     probs = [x._probability for x in self._pool]
373
374     fits = np.array([x._fitness for x in self._pool])
375     sum_f = sum(x._fitness for x in self._pool)
376     if sum(probs) != 1:
377         probs = abs(np.array(fits))/sum_f
378
379     "Pick a mutation candidate"
380     mutation_candidate = np.random.choice(self._pool, 1, p=probs)[0]
381
382     possible_coefficients = [0.9, 0.9, 0.9, 1.1, 1.1, 1.1, 0.8, 0.85, 0.75, 1.3, 1.2, 1.2]
383     coef = np.random.choice(possible_coefficients, 1) # choose one from the list above
384
385     "Best member included in population"
386     best = max(self._pool, key=lambda x: x._fitness)
387     worst = min(self._pool, key=lambda x: x._fitness)
388
389     "Mutate"
390     for i in range(rnd.randrange(1, 2, 3)): # CH - 3 nodes for 1x4
391         if mutation_type == "x":
392             mutation_candidate._nodes[0, i] = mutation_candidate._nodes[0, i] * coef
393         if mutation_type == "y":
394             mutation_candidate._nodes[1, i] = mutation_candidate._nodes[1, i] * coef
395         break
396
397     if mutation_type == "a":
398         for i in range(rnd.randrange(len(self.mem_begin))):
399             cur_candidate = self._pool[i]
400             if cur_candidate == best:
401                 self._pool[-1] = best
402                 self._pool[-1].probability = 1 - sum(x._probability for x in self._pool[:len(self._pool)])
403                 se = np.argmin(self._pool[i]._stress)
404                 if cur_candidate.A[se] > 0.0001:
405                     continue
406                 cur_candidate.A[se] = cur_candidate.A[se] * coef
407             break
408
409     self._pool.remove(worst)
410     self._pool.append(best)
411
412 def plot_stress(self):
413     num_to_plot = 4
414
415     gs = GridSpec(1, 4)
416     gs.update(left=0.05, right=0.95, wspace=0.2)
417     fig = plt.figure(figsize=(18, 5))
418     fig.suptitle("Best members in generation - stress")
419
420     for index in range(num_to_plot):
421         # take num_to_plot best candidates, load data from saved dict
422         pool = self._pool[index]
423         plot_dict = pool._plot_dict
424         stress = pool._stress
425         xi = plot_dict['xi']
426         xj = plot_dict['xj']
427         yi = plot_dict['yi']
428         yj = plot_dict['yj']
429         xinew = plot_dict['xinew']
430         xjnew = plot_dict['xjnew']
431         yinew = plot_dict['yinew']
432         yjnew = plot_dict['yjnew']
433         stress_normed = plot_dict['stress_normed']
434         F_x2 = plot_dict['F_x2']
435         dof_x2 = plot_dict['dof_x2']
436         numnode = plot_dict['numnode']
437         numelem = plot_dict['numelem']
438
439         ax = fig.add_subplot(gs[0, index], aspect="equal")
440
441         ax.grid(True)
442         ax.set_xlim(-2, 11) # CH
443         ax.set_ylim(-2.5, 5) # CH
444         ax.set_title("Candidate {}".format(index + 1))
445
446         for r in range(numelem):
447             x = (xi[r], xj[r])
448             y = (yi[r], yj[r])
449             line = ax.plot(x, y)

```

```

450         plt.setp(line, ls='-', c='black', lw='1', label='orig')
451         xnew = (xnew[r], xjnew[r])
452         ynew = (ynew[r], yjnew[r])
453         linenew = ax.plot(xnew, ynew)
454
455         plt.setp(linenew, ls='-',
456                 c='c' if stress[r] > 0.000001 else ('red' if stress[r] < -0.000001 else 'black'),
457                 lw=(1 + 20 * stress_normed[r]), label='strain' if stress[r] > 0 else 'stress')
458         ax.plot()
459
460     "Annotate outside forces"
461     for r in range(numnode):
462         plt.annotate(F_x2[r],
463                    xy=(xi[r], yi[r]), xycoords='data', xytext=np.sign(F_x2[r]) * -35,
464                    textcoords='offset pixels',
465                    arrowprops=dict(facecolor='black', shrink=0, width=1.3, headwidth=5),
466                    horizontalalignment='right', verticalalignment='bottom')
467
468     "Annotate fixed DOFs"
469     for r in range(numnode):
470         if np.array_equal(dof_x2[r], np.array([0, 1])):
471             plt.plot([xi[r]], [yi[r] - 0.2], 'o', c='k', markersize=8)
472         if np.array_equal(dof_x2[r], np.array([1, 0])):
473             plt.plot([xi[r] - 0.2], [yi[r]], 'o', c='k', markersize=8)
474         if np.array_equal(dof_x2[r], np.array([1, 1])):
475             plt.plot([xi[r]], [yi[r] - 0.2], '^', c='k', markersize=8)
476
477     plt.savefig(datetime.datetime.now().
478                strftime('stress_1x4_%Y%m%d_%H%M%S_pop300_cyc200_mx50_myA_45') + ".pdf")
479
480     #plt.show()
481
482     def plot_A(self):
483         num_to_plot = 4
484         gs = GridSpec(1, 4) # 1 column, 4 in row
485         gs.update(left=0.05, right=0.95, wspace=0.1)
486
487         fig = plt.figure(figsize=(18, 5))
488         fig.suptitle("Best members in generation - cross section")
489
490         for index in range(num_to_plot):
491             # take num_to_plot best candidates, load data from saved dictionary
492             pool = self._pool[index]
493             plot_dict = pool._plot_dict
494
495             xi = plot_dict['xi']
496             xj = plot_dict['xj']
497             yi = plot_dict['yi']
498             yj = plot_dict['yj']
499             xnew = plot_dict['xnew']
500             xjnew = plot_dict['xjnew']
501             ynew = plot_dict['ynew']
502             yjnew = plot_dict['yjnew']
503             stress_normed = plot_dict['stress_normed']
504             F_x2 = plot_dict['F_x2']
505             dof_x2 = plot_dict['dof_x2']
506             numnode = plot_dict['numnode']
507             numelem = plot_dict['numelem']
508
509             ax = fig.add_subplot(gs[0, index], aspect="equal")
510             ax.grid(True)
511             ax.set_xlim(-2, 11) # CH
512             ax.set_ylim(-2.5, 5) # CH
513             ax.set_title("Candidate {}".format(index + 1))
514
515             for r in range(numelem):
516                 x = (xi[r], xj[r])
517                 y = (yi[r], yj[r])
518                 line = ax.plot(x, y)
519                 plt.setp(line, ls='-', c='black', lw='1', label='orig')
520
521                 xnew = (xnew[r], xjnew[r])
522                 ynew = (ynew[r], yjnew[r])
523                 linenewA = ax.plot(xnew, ynew)
524
525                 plt.setp(linenewA, ls='-', c='green', lw=(1 + 70 * pool.A[r]))
526             ax.plot()
527
528     "Annotate outside forces"
529     for r in range(numnode):
530         plt.annotate(F_x2[r],
531                    xy=(xi[r], yi[r]), xycoords='data', xytext=np.sign(F_x2[r]) * -35,
532                    textcoords='offset pixels',
533                    arrowprops=dict(facecolor='black', shrink=0, width=1.5, headwidth=8),
534                    horizontalalignment='right', verticalalignment='bottom')
535
536     "Annotate fixed DOFs"
537     for r in range(numnode):
538         if np.array_equal(dof_x2[r], np.array([0, 1])):
539             plt.plot([xi[r]], [yi[r] - 0.2], 'o', c='k', markersize=8)

```


File - C:\Personal stuff\1. Skola\10. semestr - Spring 18\BP\repository\Thesis\python_projects\fc_1x4_GA.py

```
540         if np.array_equal(dof_x2[r], np.array([1, 0])):
541             plt.plot([xi[r] - 0.2], [yi[r]], 'o', c='k', markersize=8)
542         if np.array_equal(dof_x2[r], np.array([1, 1])):
543             plt.plot([xi[r]], [yi[r] - 0.2], '^', c='k', markersize=8)
544
545     plt.savefig(datetime.datetime.now().
546                 strftime('cross section_1x4_%Y%m%d_%H%M%S_pop300_cyc200_mx50_myA_45') + ".pdf")
547
548     #plt.show()
```

```

1 """
2     @Author: Margarita Kuvaldina
3     @https://github.com/margkuval
4     @date: May 2018
5 """
6
7 import fc_1x4_GA as GA
8 import plots_univ as plt_uni
9 import numpy as np
10
11 """BRAIN FOR GENETIC ALGORITHM 1x4 TRUSS"""
12
13 "Initial values - population, number of iterations, mutation 1, mutation 2, iteration of plotting"
14 inp_task_1 = np.array([40, 50, 10, 8, 25])
15
16 population_1 = inp_task_1[0] # population size
17 num_cycles_1 = inp_task_1[1] # number of computation cycles
18 mut_x_1 = inp_task_1[2]
19 mut_yA_1 = inp_task_1[3]
20 plt_s_A_1 = inp_task_1[4]
21
22 inp_task_2 = inp_task_1 #np.array([20, 20, 10, 9, 25])
23
24 population_2 = inp_task_2[0] # population size
25 num_cycles_2 = inp_task_2[1] # number of computation cycles
26 mut_x_2 = inp_task_2[2]
27 mut_yA_2 = inp_task_2[3]
28 plt_s_A_2 = inp_task_2[4]
29
30 inp_task_3 = inp_task_1 #np.array([20, 20, 10, 9, 25])
31
32 population_3 = inp_task_3[0] # population size
33 num_cycles_3 = inp_task_3[1] # number of computation cycles
34 mut_x_3 = inp_task_3[2]
35 mut_yA_3 = inp_task_3[3]
36 plt_s_A_3 = inp_task_3[4]
37
38 "Task number 1"
39 task = GA.GA(population_1)
40
41 list_iter = []
42 list_fit = []
43 list_weight = []
44 list_stress = []
45 list_stress_positive = []
46 list_stress_negative = []
47 list_defl = []
48
49 task.initial()
50 print("New task 1")
51 for i in range(num_cycles_1):
52     task.calculation()
53     task.fitness()
54     if i % plt_s_A_1 == 0:
55         task.plot_stress()
56         task.plot_A()
57     task.crossover()
58     if i % mut_x_1 == 0:
59         task.mutation(mutation_type="x")
60     if i % mut_yA_1 == 0:
61         task.mutation(mutation_type="y")
62         task.mutation(mutation_type="a")
63
64     list_iter.append(i)
65
66     task.get_best_fit()
67     task.get_best_weight()
68     task.get_best_stress()
69     task.get_best_stress_positive()
70     task.get_best_stress_negative()
71     task.get_best_defl()
72
73     list_fit.append(task.get_best_fit())
74     list_weight.append(task.get_best_weight())
75     list_stress.append(task.get_best_stress())
76     list_stress_positive.append(task.get_best_stress_positive())
77     list_stress_negative.append(task.get_best_stress_negative())
78     list_defl.append(task.get_best_defl())
79
80 plt_best_1 = plt_uni.plot_best_1(list_iter, list_fit, list_stress_positive, list_stress_negative, list_weight, list_defl)
81
82 "Task number 2"
83 task_2 = GA.GA(population_2) # population size
84
85 list_iter_2 = []
86 list_fit_2 = []
87 list_weight_2 = []
88 list_stress_2 = []
89 list_stress_positive_2 = []
90 list_stress_negative_2 = []

```

```

91 list_defl_2 = []
92
93 task_2.initial()
94 print("New task 2")
95 for r in range(num_cycles_2): # number of computation cycles
96     task_2.calculation()
97     task_2.fitness()
98     if r % plt_s_A_2 == 0:
99         task_2.plot_stress()
100        task_2.plot_A()
101        task_2.crossover()
102        if r % mut_x_2 == 0:
103            task_2.mutation(mutation_type="x")
104        if r % mut_yA_2 == 0:
105            task_2.mutation(mutation_type="y")
106            task_2.mutation(mutation_type="a")
107
108        list_iter_2.append(r)
109
110        task_2.get_best_fit()
111        task_2.get_best_weight()
112        task_2.get_best_stress()
113        task_2.get_best_stress_positive()
114        task_2.get_best_stress_negative()
115        task_2.get_best_defl()
116
117        list_fit_2.append(task_2.get_best_fit())
118        list_weight_2.append(task_2.get_best_weight())
119        list_stress_2.append(task_2.get_best_stress())
120        list_stress_positive_2.append(task_2.get_best_stress_positive())
121        list_stress_negative_2.append(task_2.get_best_stress_negative())
122        list_defl_2.append(task_2.get_best_defl())
123
124 plt_uni.plot_best_2(list_iter_2, list_fit_2, list_stress_positive_2, list_stress_negative_2, list_weight_2, list_defl_2
    )
125
126 "Task number 3"
127 task_3 = GA.GA(population_3) # population size
128
129 list_iter_3 = []
130 list_fit_3 = []
131 list_weight_3 = []
132 list_stress_3 = []
133 list_stress_positive_3 = []
134 list_stress_negative_3 = []
135 list_defl_3 = []
136
137 task_3.initial()
138 print("New task 3")
139 for k in range(num_cycles_3): # number of computation cycles
140     task_3.calculation()
141     task_3.fitness()
142     if k % plt_s_A_3 == 0:
143         task_3.plot_stress()
144         task_3.plot_A()
145     task_3.crossover()
146     if k % mut_x_3 == 0:
147         task_3.mutation(mutation_type="x")
148     if k % mut_yA_3 == 0:
149         task_3.mutation(mutation_type="y")
150         task_3.mutation(mutation_type="a")
151
152     list_iter_3.append(k)
153
154     task_3.get_best_fit()
155     task_3.get_best_weight()
156     task_3.get_best_stress()
157     task_3.get_best_stress_positive()
158     task_3.get_best_stress_negative()
159     task_3.get_best_defl()
160
161     list_fit_3.append(task_3.get_best_fit())
162     list_weight_3.append(task_3.get_best_weight())
163     list_stress_3.append(task_3.get_best_stress())
164     list_stress_positive_3.append(task_3.get_best_stress_positive())
165     list_stress_negative_3.append(task_3.get_best_stress_negative())
166     list_defl_3.append(task_3.get_best_defl())
167
168 plt_uni.plot_best_3(list_iter_3, list_fit_3, list_stress_positive_3, list_stress_negative_3, list_weight_3, list_defl_3
    )
169
170 plt_fits_3 = plt_uni.plot_fits_3(list_iter, list_iter_2, list_iter_3,
    list_fit, list_fit_2, list_fit_3,
    population_1, population_2, population_3,
    mut_x_1, mut_x_2, mut_x_3, mut_yA_1, mut_yA_2, mut_yA_3)

```

```

1  """
2      @Author: Margarita Kuvaldina
3      @https://github.com/margkuval
4      @date: May 2018
5  """
6
7  """PLOT FOR ANY TRUSS IN BRAIN"""
8
9  import matplotlib.pyplot as plt
10 import numpy as np
11 import datetime
12
13
14 def plot_best_1(list_iter, list_fit, list_stress_positive, list_stress_negative, list_weight, list_defl):
15     fig = plt.figure(figsize=(10, 8))
16     fig.suptitle('Individual 1: the fittest individual development in time',
17                 horizontalalignment='center', verticalalignment='center')
18
19     "Fitness plot"
20     list_fit = np.array(list_fit).transpose()
21     x_fit = list_iter
22     y_fit = list_fit
23
24     ax1 = fig.add_subplot(2, 2, 1)
25     ax1.plot(x_fit, y_fit, c='r')
26     ax1.set_title('Fitness evolution')
27     ax1.set_xlabel('Iterations')
28     ax1.set_ylabel('Fitness')
29     plt.grid(b=True, which='both', axis='both')
30
31     "Stress plot"
32     list_stress_positive = np.array(list_stress_positive)
33     x_stress_positive = list_iter
34     y_stress_positive = list_stress_positive
35     stress_positive = (x_stress_positive, y_stress_positive)
36
37     list_stress_negative = np.array(list_stress_negative)
38     x_stress_negative = list_iter
39     y_stress_negative = list_stress_negative
40
41     stress_negative = (x_stress_negative, y_stress_negative)
42
43     ax2 = fig.add_subplot(2, 2, 2)
44     ax2.set_title('Stress evolution')
45     ax2.set_xlabel('Iterations')
46     ax2.plot(x_stress_negative, y_stress_negative, c='coral')
47     ax2.set_ylabel('Negative stress sum (MPa)')
48     ax2.yaxis.label.set_color('coral')
49
50     ax2_t = ax2.twinx()
51     ax2_t.plot(x_stress_positive, y_stress_positive, c='darkslateblue')
52     ax2_t.set_ylabel('Positive stress sum (MPa)')
53     plt.grid(b=True, which='both', axis='both')
54     ax2_t.yaxis.label.set_color('darkslateblue')
55
56     "Weight plot"
57     list_weight = np.array(list_weight).transpose()
58     x_weight = list_iter
59     y_weight = list_weight
60
61     ax3 = fig.add_subplot(2, 2, 3)
62     ax3.plot(x_weight, y_weight, c='firebrick')
63     ax3.set_title('Weight evolution')
64     ax3.set_xlabel('Iterations')
65     ax3.set_ylabel('Construction weight (1000kg)')
66     plt.grid(b=True, which='both', axis='both')
67
68     "Deflection plot"
69     list_defl = np.array(list_defl).transpose()
70     x_defl = list_iter
71     y_defl = np.round(sum(abs(list_defl[0])), 3)
72
73     ax4 = fig.add_subplot(2, 2, 4)
74     ax4.plot(x_defl, y_defl, c='lightcoral')
75     ax4.set_title('Deflection evolution')
76     ax4.set_xlabel('Iterations')
77     ax4.set_ylabel('Abs deflection sum (m)')
78     plt.grid(b=True, which='both', axis='both')
79
80     # plt.legend(bbox_to_anchor=(0., 1.007, 1., .101), loc=3, ncol=1, mode="expand", borderaxespad=0.)
81
82     plt.subplots_adjust(wspace=0.5, hspace=0.5) # keep top
83     plt.savefig(datetime.datetime.now().strftime('F_snp_w_d_I1_%Y%m%d_%H%M%S_') + ".pdf")
84
85
86 def plot_best_2(list_iter_2, list_fit_2, list_stress_positive_2, list_stress_negative_2, list_weight_2, list_defl_2):
87     fig = plt.figure(figsize=(10, 8))
88     fig.suptitle('Individual 2: the fittest individual development in time',
89                 horizontalalignment='center', verticalalignment='center')
90

```

```

91     "Fitness plot"
92     list_fit_2 = np.array(list_fit_2).transpose()
93     x_fit = list_iter_2
94     y_fit = list_fit_2
95
96     ax1 = fig.add_subplot(2, 2, 1)
97     ax1.plot(x_fit, y_fit, c='navy')
98     ax1.set_title('Fitness evolution')
99     ax1.set_xlabel('Iterations')
100    ax1.set_ylabel('Fitness')
101    plt.grid(b=True, which='both', axis='both')
102
103    "Stress plot"
104    list_stress_positive_2 = np.array(list_stress_positive_2)
105    x_stress_positive_2 = list_iter_2
106    y_stress_positive_2 = list_stress_positive_2
107    stress_positive = (x_stress_positive_2, y_stress_positive_2)
108
109    x_stress_negative_2 = list_iter_2
110    y_stress_negative_2 = np.array(list_stress_negative_2)
111    stress_negative = (x_stress_negative_2, y_stress_negative_2)
112
113    ax2 = fig.add_subplot(2, 2, 2)
114    ax2.set_title('Stress evolution')
115    ax2.set_xlabel('Iterations')
116    ax2.plot(x_stress_negative_2, y_stress_negative_2, c='coral')
117    ax2.set_ylabel('Negative stress sum (MPa)')
118    ax2.yaxis.label.set_color('coral')
119
120    ax2_t = ax2.twinx()
121    ax2_t.plot(x_stress_positive_2, y_stress_positive_2, c='darkslateblue')
122    ax2_t.set_ylabel('Positive stress sum (MPa)')
123    plt.grid(b=True, which='both', axis='both')
124    ax2_t.yaxis.label.set_color('darkslateblue')
125
126    "Weight plot"
127    list_weight = np.array(list_weight_2).transpose()
128    x_weight = list_iter_2
129    y_weight = list_weight
130
131    ax3 = fig.add_subplot(2, 2, 3)
132    ax3.plot(x_weight, y_weight, c='cornflowerblue')
133    ax3.set_title('Weight evolution')
134    ax3.set_xlabel('Iterations')
135    ax3.set_ylabel('Construction weight (1000kg)')
136    plt.grid(b=True, which='both', axis='both')
137
138    "Deflection plot"
139    list_defl = np.array(list_defl_2).transpose()
140    x_defl = list_iter_2
141    y_defl = np.round(sum(abs(list_defl[0])), 3)
142
143    ax4 = fig.add_subplot(2, 2, 4)
144    ax4.plot(x_defl, y_defl, c='mediumblue')
145    ax4.set_title('Deflection evolution')
146    ax4.set_xlabel('Iterations')
147    ax4.set_ylabel('Abs deflection sum (m)')
148    plt.grid(b=True, which='both', axis='both')
149
150    # plt.legend(bbox_to_anchor=(0., 1.007, 1., .101), loc=3, ncol=1, mode="expand", borderaxespad=0.)
151
152    plt.subplots_adjust(wspace=0.5, hspace=0.5) # keep top
153    plt.savefig(datetime.datetime.now().strftime('F_snp_w_d_I2_%Y%m%d%H%M%S') + ".pdf")
154
155
156 def plot_best_3(list_iter_3, list_fit_3, list_stress_positive_3, list_stress_negative_3, list_weight_3, list_defl_3):
157     fig = plt.figure(figsize=(10, 8))
158     fig.suptitle('Individual 3: the fittest individual development in time',
159                 horizontalalignment='center', verticalalignment='center')
160
161     "Fitness plot"
162     list_fit_3 = np.array(list_fit_3).transpose()
163     x_fit = list_iter_3
164     y_fit = list_fit_3
165
166     ax1 = fig.add_subplot(2, 2, 1)
167     ax1.plot(x_fit, y_fit, c='darkgreen')
168     ax1.set_title('Fitness evolution')
169     ax1.set_xlabel('Iterations')
170     ax1.set_ylabel('Fitness')
171     plt.grid(b=True, which='both', axis='both')
172
173     "Stress plot"
174     list_stress_positive = np.array(list_stress_positive_3)
175     x_stress_positive = list_iter_3
176     y_stress_positive = list_stress_positive
177     stress_positive = (x_stress_positive, y_stress_positive)
178
179     list_stress_negative = np.array(list_stress_negative_3)
180     x_stress_negative = list_iter_3

```

```

181 y_stress_negative = list_stress_negative
182 stress_negative = (x_stress_negative, y_stress_negative)
183
184 ax2 = fig.add_subplot(2, 2, 2)
185 ax2.set_title('Stress evolution')
186 ax2.set_xlabel('Iterations')
187 ax2.plot(x_stress_negative, y_stress_negative, c='coral')
188 ax2.set_ylabel('Negative stress sum (MPa)')
189 ax2.yaxis.label.set_color('coral')
190
191 ax2_t = ax2.twinx()
192 ax2_t.plot(x_stress_positive, y_stress_positive, c='darkslateblue')
193 ax2_t.set_ylabel('Positive stress sum (MPa)')
194 plt.grid(b=True, which='both', axis='both')
195 ax2_t.yaxis.label.set_color('darkslateblue')
196
197 "Weight plot"
198 list_weight = np.array(list_weight_3).transpose()
199 x_weight = list_iter_3
200 y_weight = list_weight
201
202 ax3 = fig.add_subplot(2, 2, 3)
203 ax3.plot(x_weight, y_weight, c='olivedrab')
204 ax3.set_title('Weight evolution')
205 ax3.set_xlabel('Iterations')
206 ax3.set_ylabel('Construction weight (1000kg)')
207 plt.grid(b=True, which='both', axis='both')
208
209 "Deflection plot"
210 list_defl = np.array(list_defl_3).transpose()
211 x_defl = list_iter_3
212 y_defl = np.round(sum(abs(list_defl[0])), 3)
213
214 ax4 = fig.add_subplot(2, 2, 4)
215 ax4.plot(x_defl, y_defl, c='mediumseagreen')
216 ax4.set_title('Deflection evolution')
217 ax4.set_xlabel('Iterations')
218 ax4.set_ylabel('Abs deflection sum (m)')
219 plt.grid(b=True, which='both', axis='both')
220
221 # plt.legend(bbox_to_anchor=(0., 1.007, 1., .101), loc=3, ncol=1, mode="expand", borderaxespad=0.)
222
223 plt.subplots_adjust(wspace=0.5, hspace=0.5) # keep top
224 plt.savefig(datetime.datetime.now().strftime('F_snp_w_d_I3_%Y%m%d_%H%M%S_') + ".pdf")
225
226 def plot_fits_3(list_iter, list_iter_2, list_iter_3,
227               list_fit, list_fit_2, list_fit_3,
228               population_1, population_2, population_3,
229               mut_x_1, mut_x_2, mut_x_3, mut_yA_1, mut_yA_2, mut_yA_3):
230
231     list_fit = np.array(list_fit).transpose()
232     list_fit_2 = np.array(list_fit_2).transpose()
233     list_fit_3 = np.array(list_fit_3).transpose()
234
235     fig = plt.figure(figsize=(10, 8))
236
237     fig.suptitle('Fitness evolution', horizontalalignment='center', verticalalignment='center')
238     fig.subplots_adjust(wspace=0.5, hspace=0.6)
239
240     "Fitness plot"
241     x_fit_1 = list_iter
242     y_fit_1 = list_fit
243     x_fit_2 = list_iter_2
244     y_fit_2 = list_fit_2
245     x_fit_3 = list_iter_3
246     y_fit_3 = list_fit_3
247
248     ax1 = fig.add_subplot(1, 1, 1)
249
250     ax1.plot(x_fit_1, y_fit_1, 'r', label=('Population size: %s' % population_1,
251                                         'Mutation x: 2 ind every %s iter' % mut_x_1, 'Mutation y, A: every %s' %
mut_yA_1))
252     ax1.plot(x_fit_2, y_fit_2, 'navy', label=('Population size: %s' % population_2,
253                                             'Mutation x: 1 ind every %s iter' % mut_x_2, 'Mutation y, A: every %s' %
mut_yA_2))
254     ax1.plot(x_fit_3, y_fit_3, 'darkgreen', label=('Population size: %s' % population_3,
255                                                  'Mutation x: 1 ind every %s iter' % mut_x_3, 'Mutation y, A: every %s' %
mut_yA_3))
256
257     #ax1.plot(x_fit_1, y_fit_1, 'r', label='Mutation %s' % )
258     #ax1.plot(x_fit_2, y_fit_2, 'navy', label='Pop %s' % population_2)
259     #ax1.plot(x_fit_3, y_fit_3, 'gold', label='Pop %s' % population_3)
260     ax1.set_xlabel('Iterations')
261     ax1.set_ylabel('Fitness')
262     plt.grid(b=True, which='both', axis='both')
263     plt.legend(bbox_to_anchor=(0., 1.007, 1., .101), loc=3,
264              ncol=1, mode="expand", borderaxespad=0.)
265
266     plt.savefig(datetime.datetime.now().strftime('Fit3_%Y%m%d_%H%M%S_') + ".pdf")
267

```