# Отчет.

## MrMatiuz, 521 группа

## Математическая постановка задачи:

## Численное решение:

* Строим сетки со всеми входными параметрами в Fidesys. Сеток будет несколько, для дальнейшей проверки сходимости.

Количество элементов для разных сеток:

316

628

1410

2464

4026

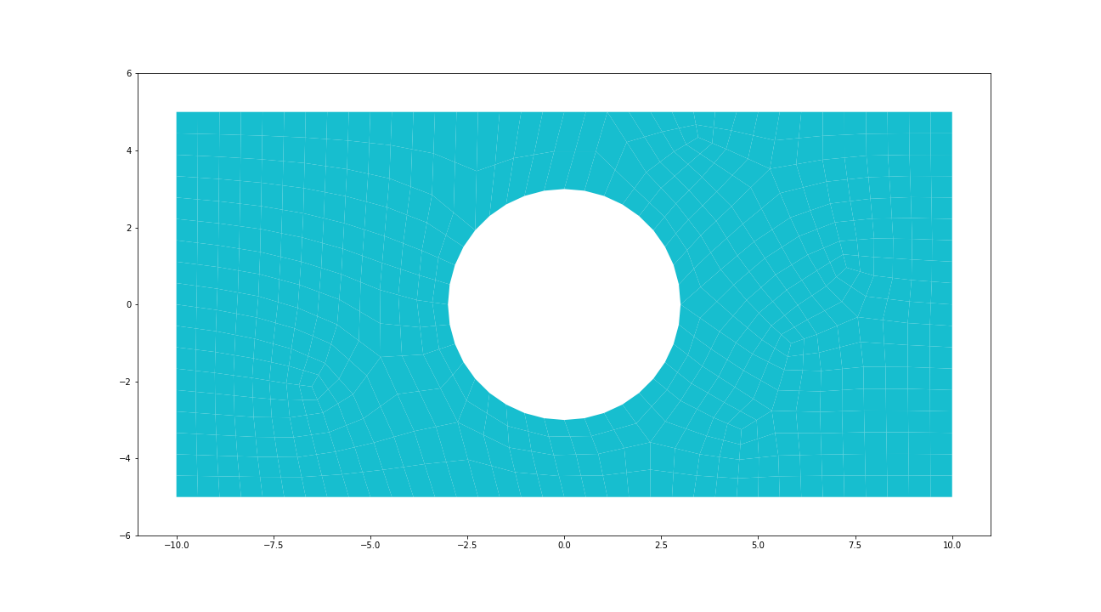
4544

* Считываем данные из файлов в функции **read\_nodes\_file**.

Составляем:

1) двумерный массив, где номер строки – индекс фигуры, а строка – индексы узлов, формирующие данную фигуру;

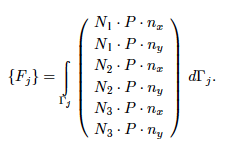
2) словарь, где ключом является индекс узла, а значением – координаты.

* Выводим саму сетку для подтверждения правильности действий через функцию **draw\_mesh**. Данная функция получает на вход параметр value, благодаря которому можно получить окрашенную сетку в зависимости от значения в каждой фигуре сетки. Сетка строится благодаря отрисовке каждой отдельной фигуры (в нашем случае треугольники).  
  
* В функции K\_global считаем глобальную матрицу жесткости [K].  
  Считается она следующим образом: , где локальная матрица равна интегралу . Для расчета уже этого интеграла нам надо посчитать частные производные функции форм. Это реализовано в функции get\_dN. Функции форм задаются на единичном квадрате.  
  Далее проводим замену переменных: , где определяем по: ,   
  Запишем матрицу Якоби:  
  И получаем:Тогда:  
     
  Для вычисления используем скобки Пуассона, которые имеет следующий вид:

Матрица В – матрица функций форм, рассчитывается по формуле:

Матрица D – матрица упругости, рассчитывается по формуле:

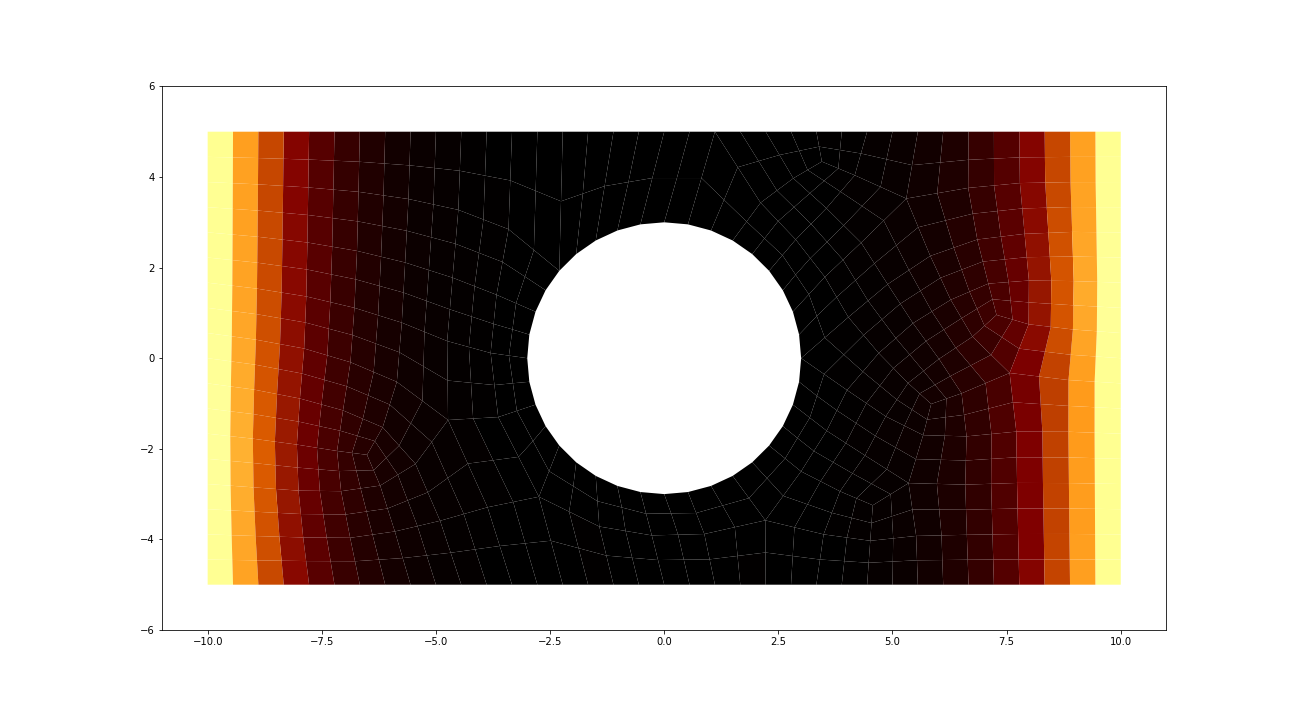
, где E – модуль Юнга, – коэффициент Пуассона.

* В функции M\_global строим глобальную матрицу масс. , где локальная матрица равна интегралу
* Cоставляем правую часть F\_global на основе локальных векторов силы для отдельных треугольников:  
  
* В функции **main\_solver** выполняется запуск решения динамической задчи с использованием схемы Ньюморка:  
  Решаем систему .   
  Перемещения в узлах от времени считаем следующим образом. На каждой итерации считаем перемещения:  
  Для нахождения поправок решаем систему Для нахождения следующего значения скорости считаем:  
  И заново повторяем итеративный подход.
* Через функции **get\_deformation и get\_stress** считаем деформации и напряжения через такой же подсчет локально для каждого треугольника по формулам:

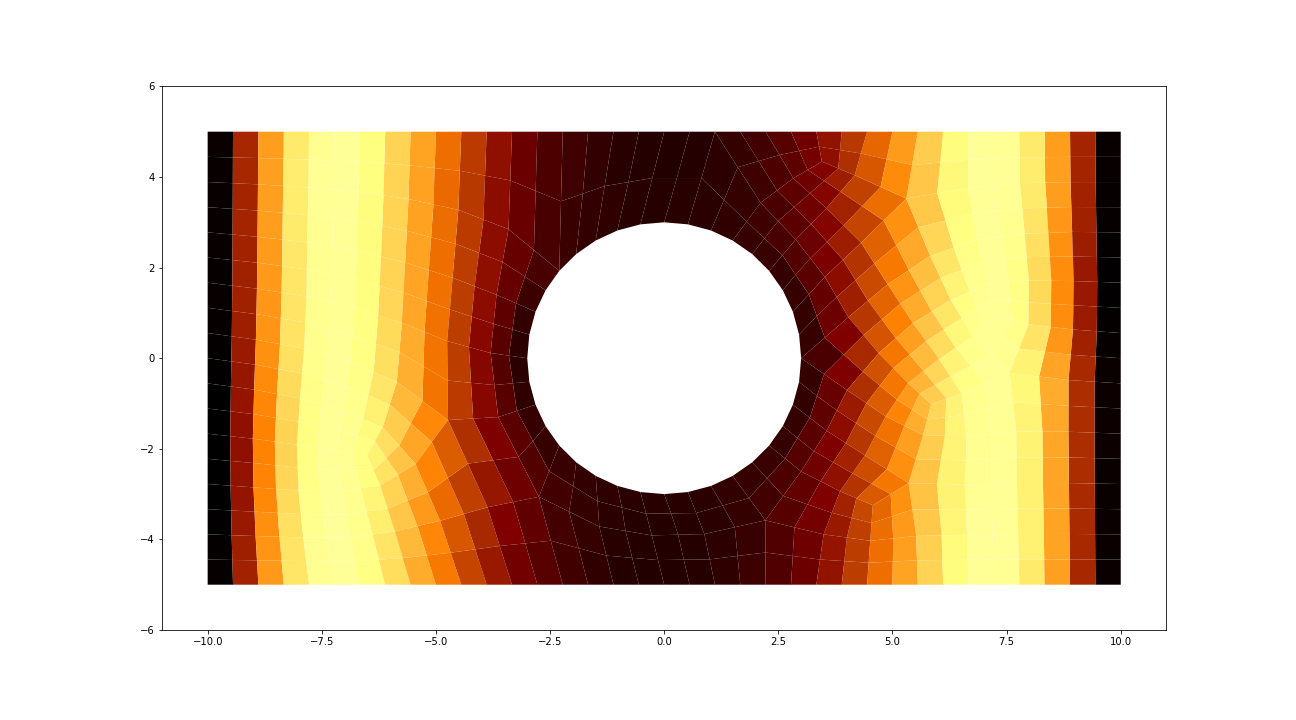
## Результаты:

Рассмотрим случай с приложением одинаковой силы к обеим боковым сторонам. Запустим код и взглянем на напряжение вдоль оси Х в динамике.

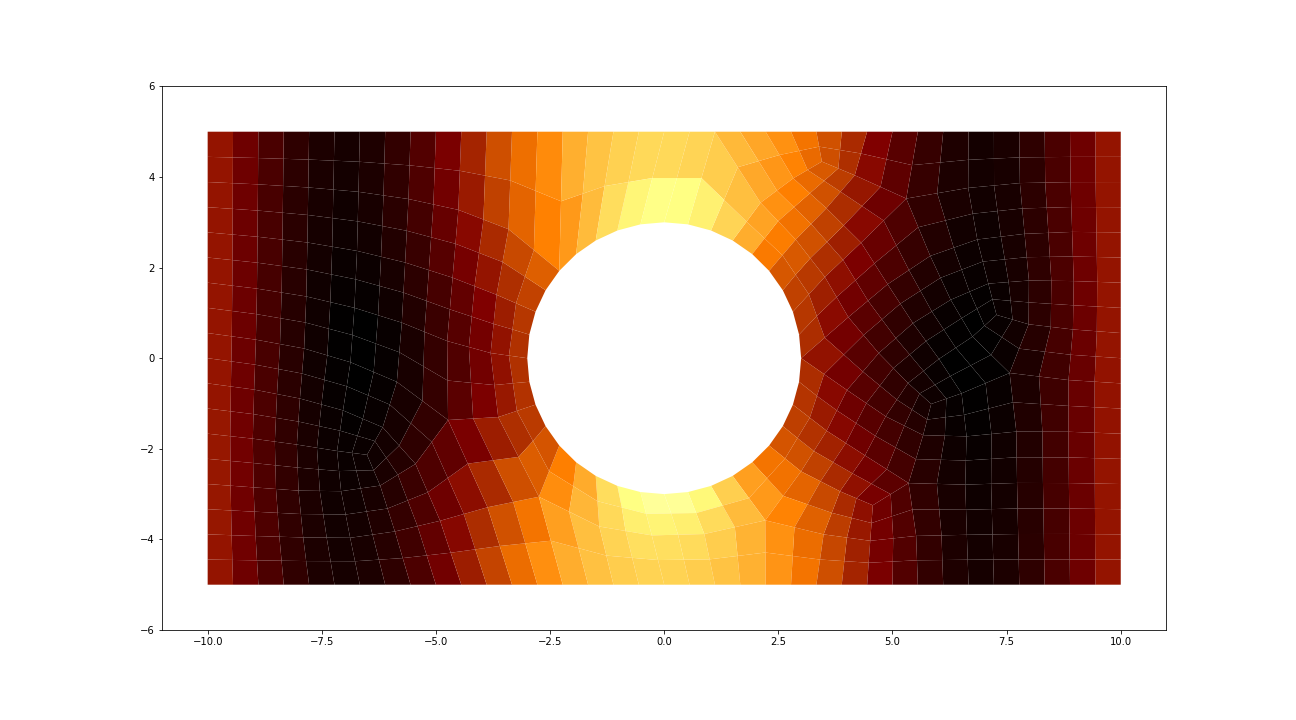
T=0.4



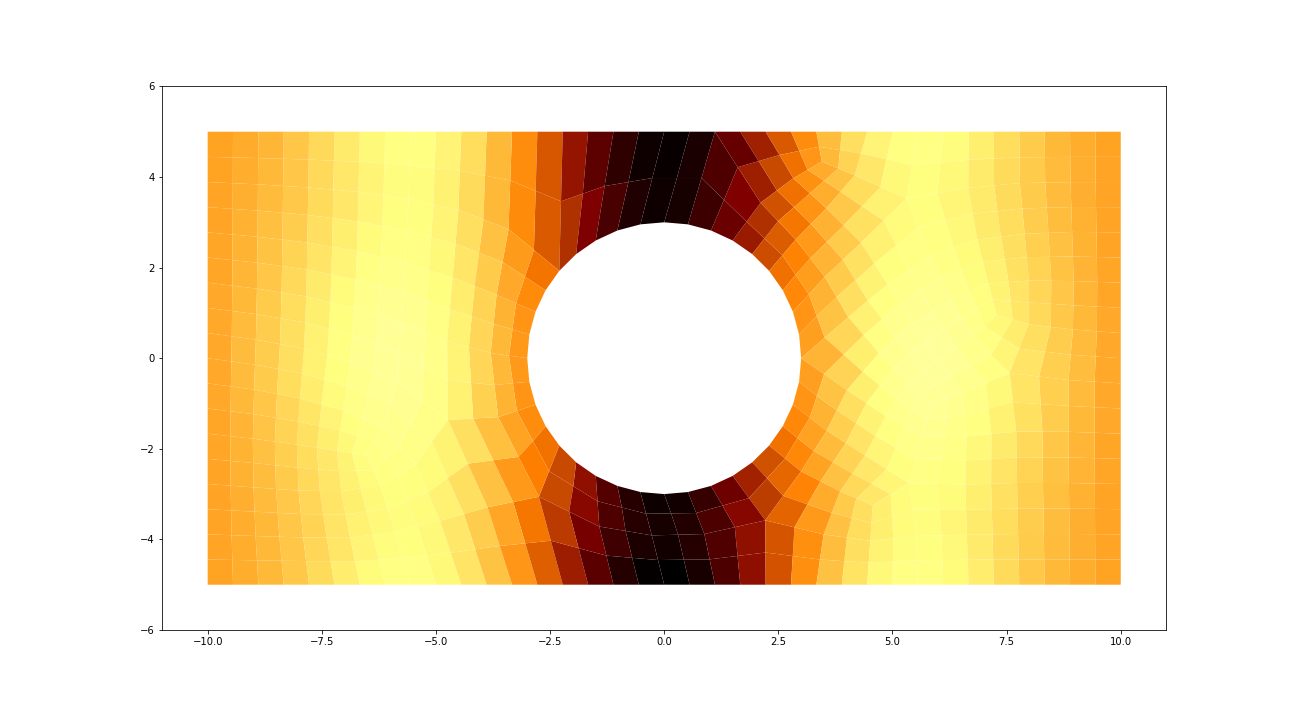
T=0.8



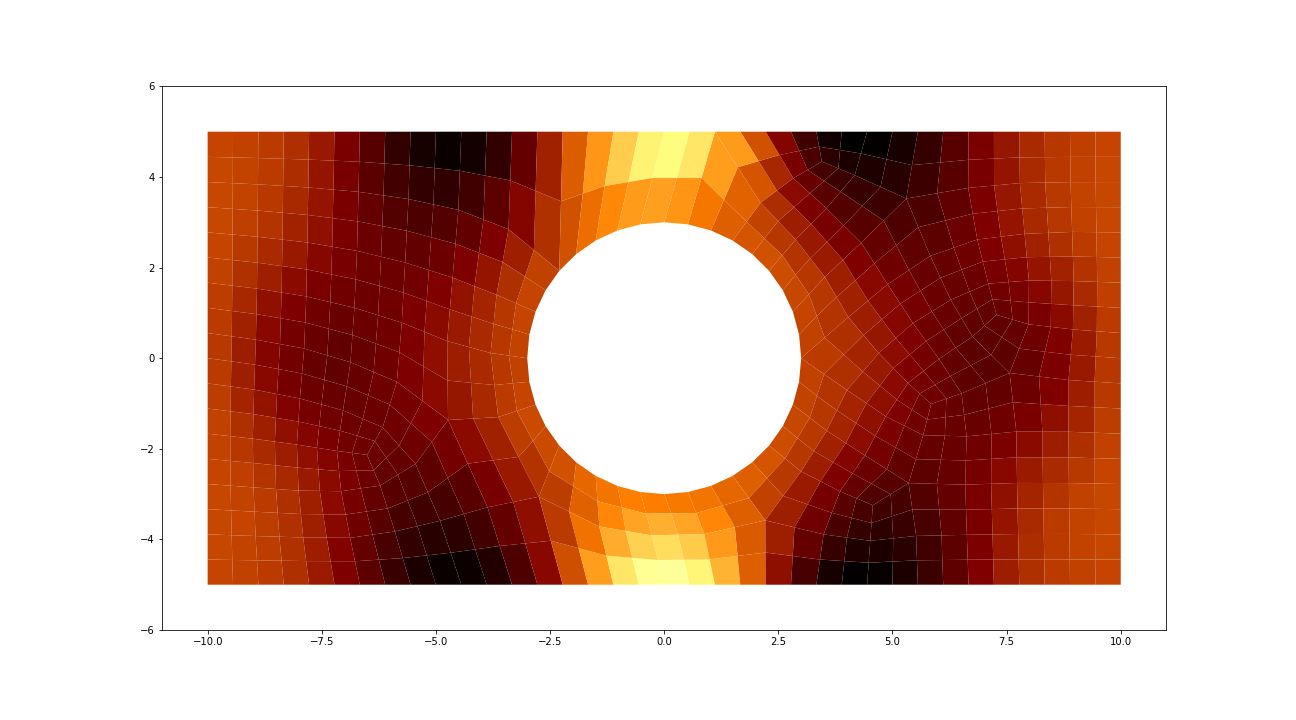
T=1.2

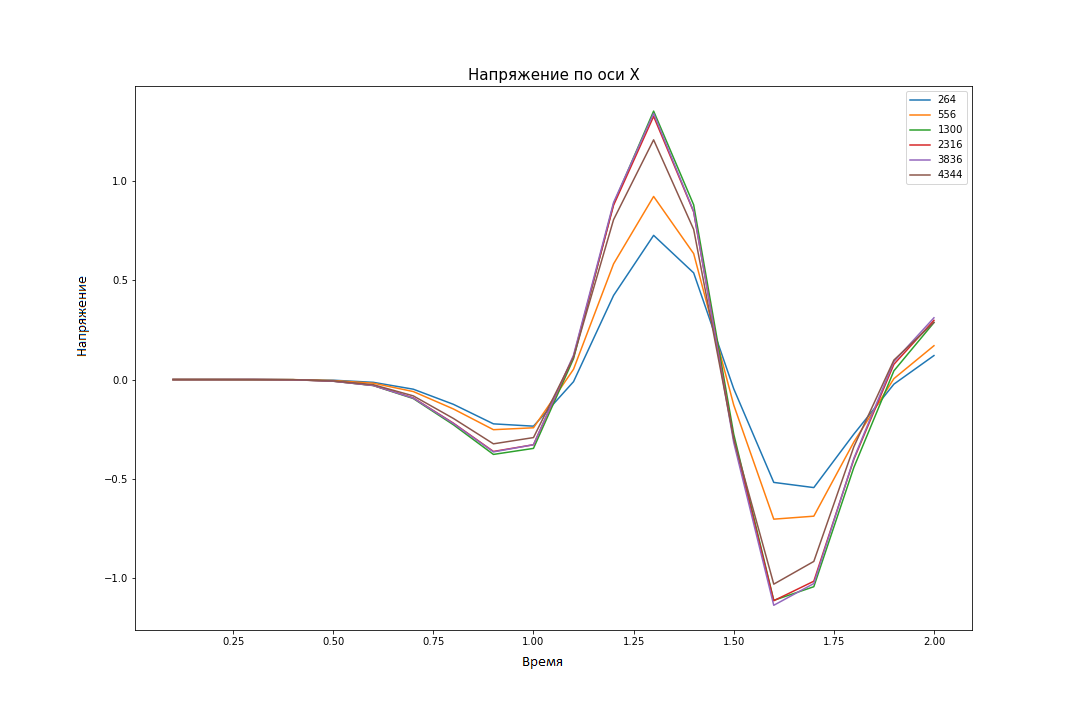


T=1.6



T=2

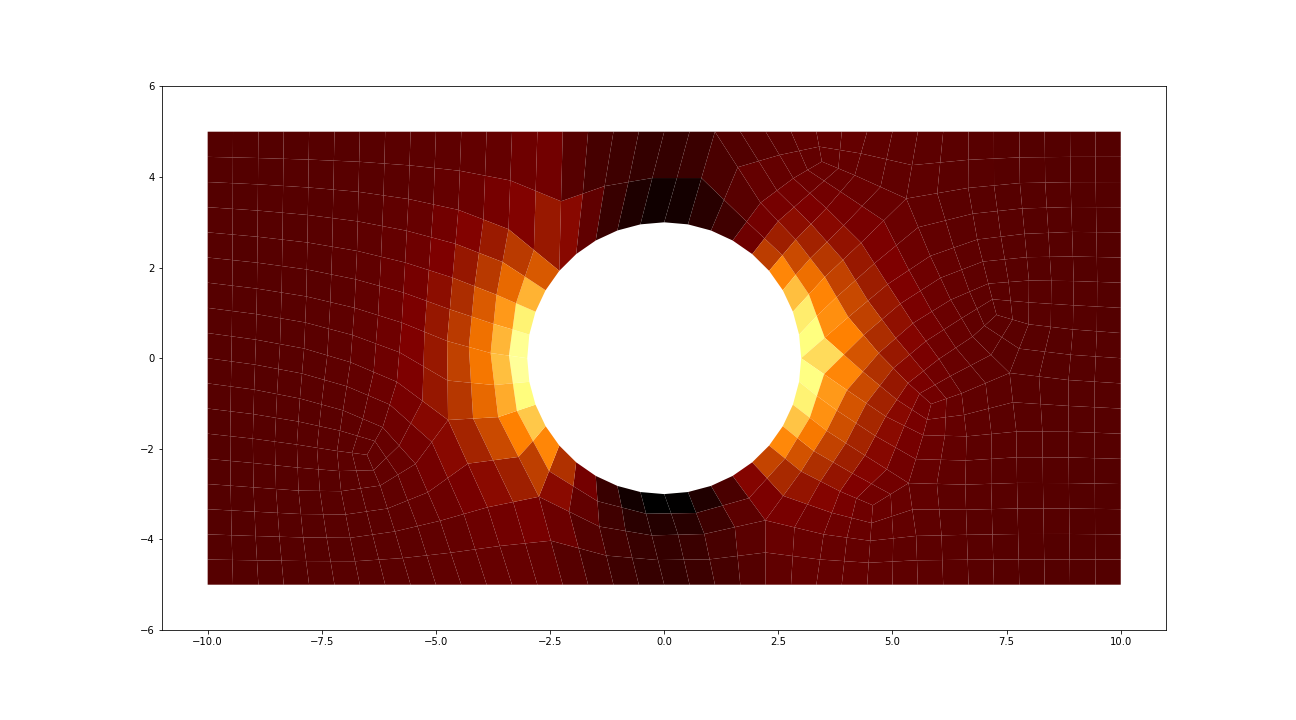


Посмотрим теперь на сеточную сходимость. Можно видеть, что при последних трех разбиениях сеток графики практически совпадают, следовательно наше решение хорошо сходится при измельчении сетки.

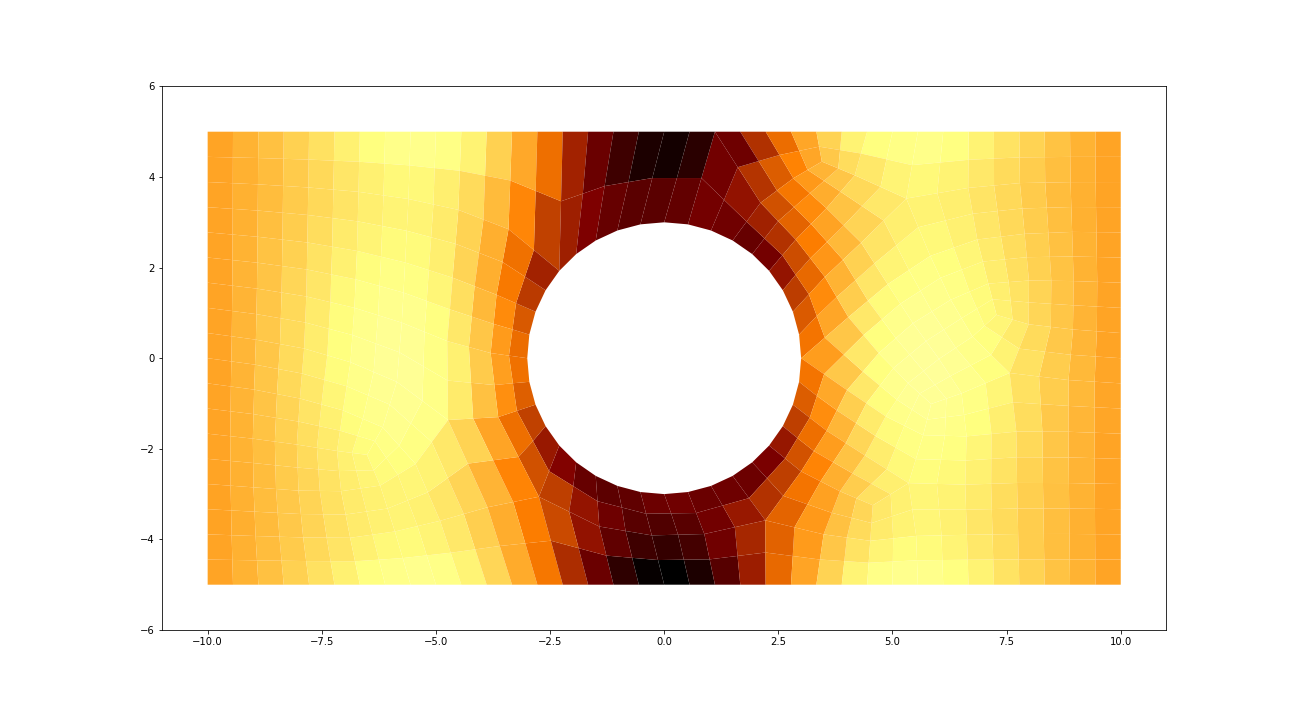
## Дополнительное задание:

Рассмотрим случай, когда сила действует на границе круговой полости. Получаем следующее распределение напряжения вдоль оси Х в динамике.

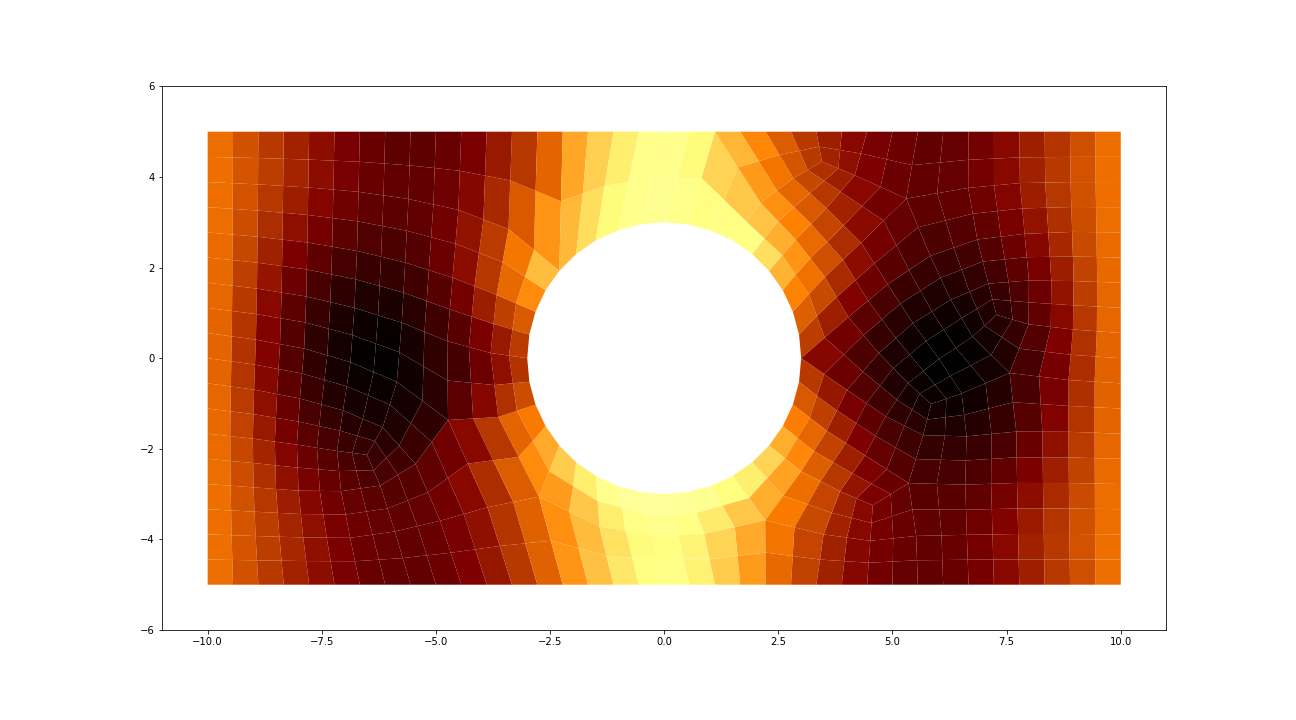
T=0.4



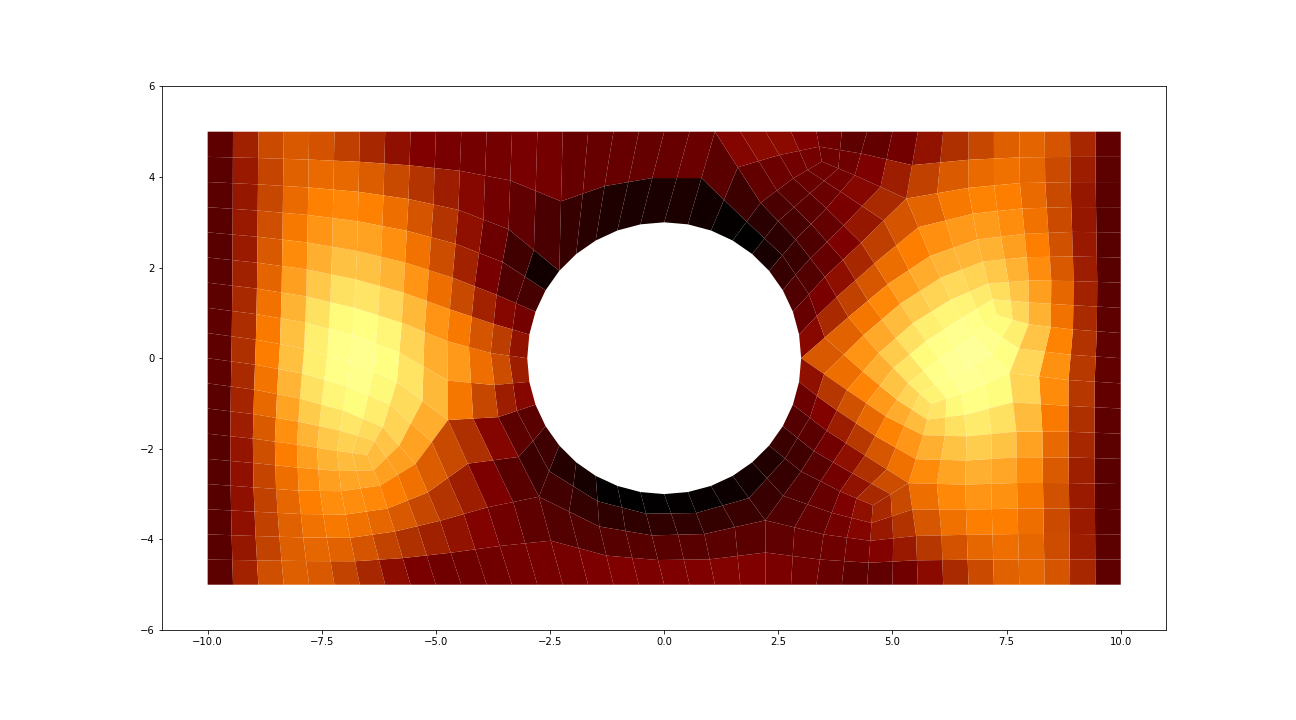
T=0.8



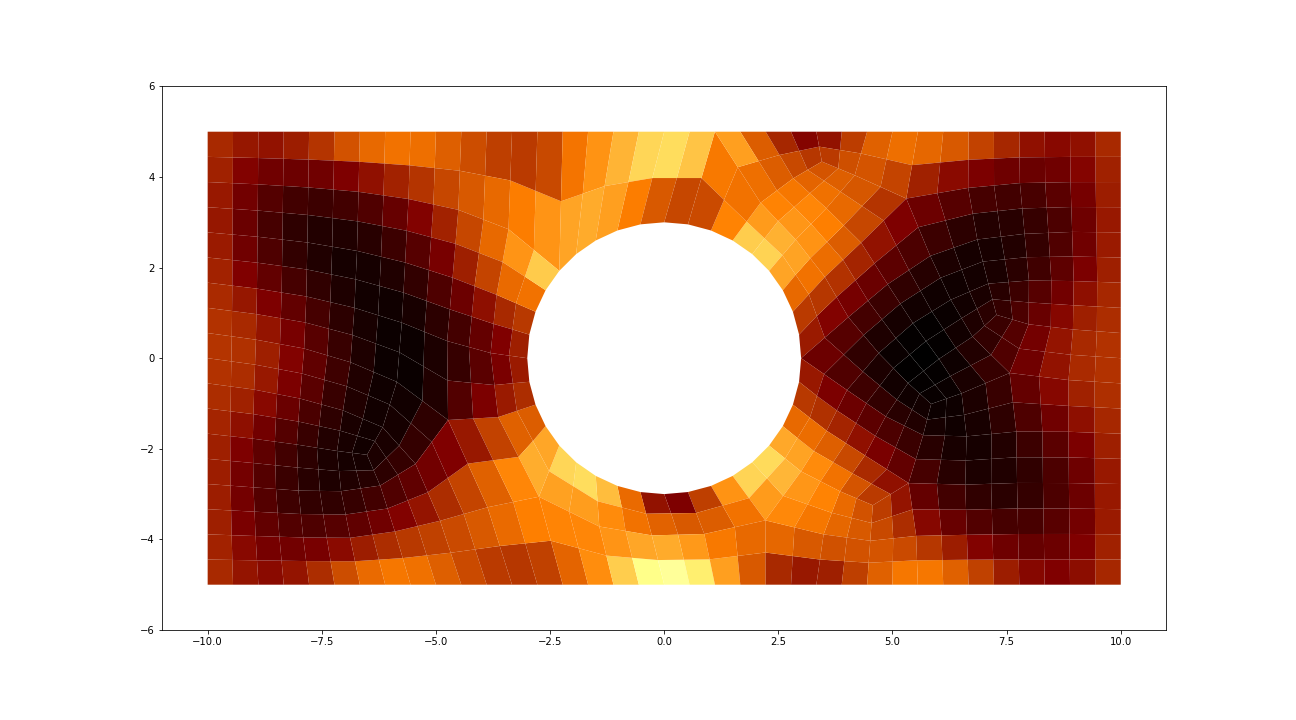
T=1.2



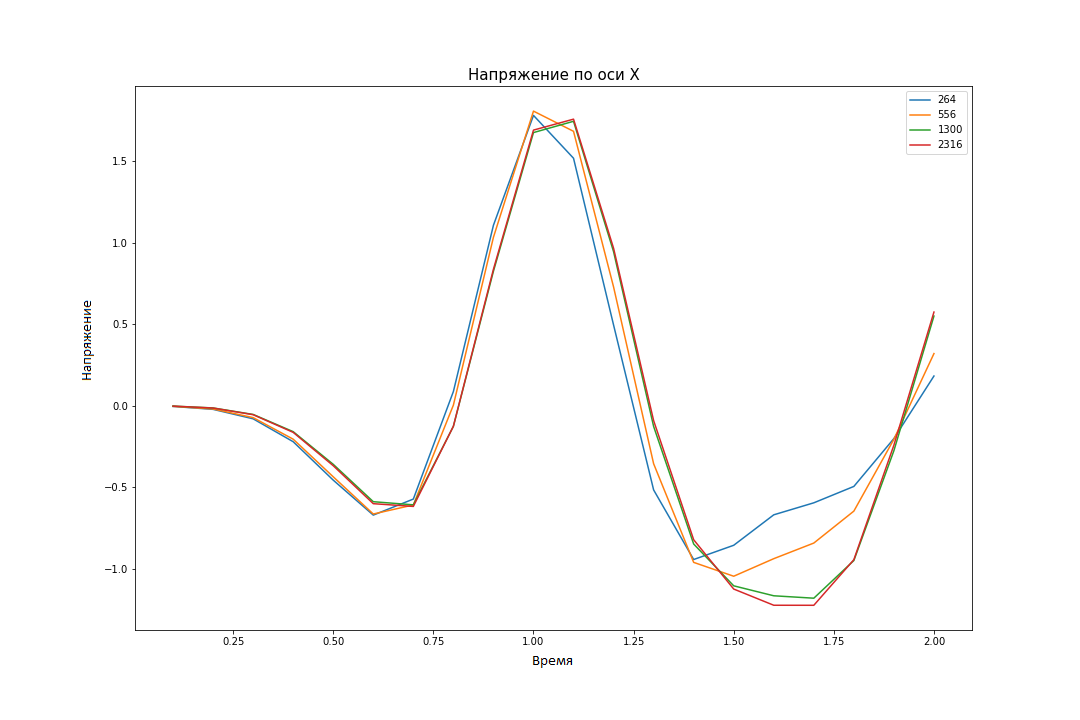
T=1.6



T=2



Посмотрим теперь на сеточную сходимость. Рассматривалось напряжение в точке (-3,0)



## Листинг программы:

1. **import** numpy **as** np
2. **import** matplotlib.pyplot **as** plt
3. **from** itertools **import** combinations
4. **import** math
5. **from itertools import product**
6. **import** matplotlib.pyplot **as** plt
7. **import** matplotlib.tri **as** mtri
8. **from** matplotlib **import** cm
9. **from** matplotlib.patches **import** Polygon
10. **import glob**

13. *# In[2]:*

16. **from** IPython.core.display **import** display, HTML
17. display(HTML("<style>.container { width:97% !important; }</style>"))

20. ***# In[3]:***

23. **def** read\_nodes\_file(file):
24. '''
25. **Функция считывает сетку (пока только для плоского объекта), построенную в фидесисе из файла формата '.k'.**
26. На вход принимает:
27. file - Файл формата '.k'.
28. Возвращает:
29. x\_nodes - x-координата узла;
30. **y\_nodes - y-координата узла;**
31. figs\_nodes - последовательность индексов узлов, которые образуют фигуру;
32. dict\_of\_nodes - словарь, ключом является индекс узла, а значвением является координаты этого узла.
33. '''
34. mesh\_file = open(file, 'r', encoding='utf-8')
35. **mesh = mesh\_file.read()**
36. start, mid, end = str(mesh).find('NODE'), str(mesh).find('ELEMENT\_SHELL'), str(mesh).find('END')
37. list\_of\_nodes = mesh[start:mid].split('**\n**')[2:-4]
38. list\_figs\_nodes = mesh[mid:end].split('**\n**')[1:-1]
40. **x\_nodes = []**
41. y\_nodes = []
42. figs\_nodes = []
43. dict\_of\_nodes = {}
45. **for node in list\_of\_nodes:**
46. lst = node.split()
47. dict\_of\_nodes[int(lst[0])-1] = (float(lst[1]), float(lst[2]))
48. x\_nodes += [float(lst[1])]
49. y\_nodes += [float(lst[2])]
51. **for** node **in** list\_figs\_nodes:
52. lst = node.split()
53. figs\_nodes += [[int(lst[2])-1, int(lst[3])-1, int(lst[4])-1, int(lst[5])-1]] *#Случай для четырехугольнов*
54. *# figs\_nodes += [[int(lst[2])-1, int(lst[3])-1, int(lst[4])-1]]*
56. *# count\_of\_nodes = len(dict\_of\_nodes)*
58. **return** x\_nodes, y\_nodes, figs\_nodes, dict\_of\_nodes

61. *# In[16]:*

64. **def** draw\_mesh(figs\_nodes, dict\_of\_nodes, value=None, size=(18,10)):
65. **'''**
66. Функция рисует сетку. Так же может выводить с градиентом цвета при наличии значений value.
67. На вход принимает:
68. figs\_nodes - последовательность индексов узлов, которые образуют фигуру;
69. dict\_of\_nodes - словарь, ключом является индекс узла, а значвением является координаты этого узла;
70. **value - list со значениями величины (например, напряжения). По умолчанию value = None;**
71. size - размер рисунка.
72. '''
73. fig, ax = plt.subplots(1, figsize=size)
74. ax.set\_xlim([-11, 11])
75. **ax.set\_ylim([-6, 6])**
77. **if**(value **is** **not** None):
78. colored\_tr = {}
79. max\_v = max(value)
80. **min\_v = min(value)**
82. **for** i, v **in** enumerate(value):
83. colored\_tr[i] = cm.get\_cmap('afmhot')(0.8\*(max\_v-v) / (max\_v-min\_v))
85. **for num, nodes in enumerate(figs\_nodes):**
86. ax.add\_patch(Polygon([dict\_of\_nodes[x] **for** x **in** nodes],
87. facecolor=colored\_tr[num]))
89. **else**:
90. **for num, nodes in enumerate(figs\_nodes):**
91. ax.add\_patch(Polygon([dict\_of\_nodes[x] **for** x **in** nodes],
92. facecolor='tab:cyan'))

95. ***# In[5]:***

98. get\_ipython().run\_cell\_magic('time', '', "x, y, figs\_nodes, dict\_of\_nodes = read\_nodes\_file('meshes/mesh\_v2\_628nodes.k')**\n**draw\_mesh(figs\_nodes, dict\_of\_nodes)**\n**plt.show()**\n**# plt.savefig('fig1')")

101. *# In[6]:*

104. **def** Geron(x1, y1, x2, y2, x3, y3):
105. **'''**
106. Функция для расчета площади треугольника.
107. На вход принимает:
108. координаты трех точек треугольника.
109. Возвращает:
110. **S - площадь.**
111. '''
112. A = np.array([[x1 - x3, y1 - y3],
113. [x2 - x3, y2 - y3]])
114. S = abs(np.linalg.det(A))
116. **return** S\*1/2
118. **def** get\_centre\_os\_sqr(x1, y1, x2, y2, x3, y3, x4, y4):
119. '''
120. **Функция для расчета центра четырехугольника.**
121. На вход принимает:
122. координаты четырех точек четырехугольника.
123. Возвращает:
124. x5, y5 - координаты центра.
125. **'''**
126. x5 = (x1 + x2 + x3 + x4)/4
127. y5 = (y1 + y2 + y3 + y4)/4
129. **return** x5, y5
131. **def** get\_area\_of\_sqr(x1, y1, x2, y2, x3, y3, x4, y4):
132. '''
133. Функция для расчета площади четырехугольника.
134. На вход принимает:
135. **координаты четырех точек четырехугольника.**
136. Возвращает:
137. S - площадь.
138. '''
139. S1 = Geron(x1, y1, x2, y2, x4, y4)
140. **S2 = Geron(x1, y1, x2, y2, x3, y3)**
141. S3 = Geron(x4, y4, x2, y2, x3, y3)
142. S4 = Geron(x1, y1, x4, y4, x3, y3)
144. **return** np.array([S1, S2, S3, S4])
146. *# Geron(x1=0, y1=0, x2=-1, y2=0, x3=0, y3=1)*
148. **def** get\_D():
149. '''
150. **Функция создает матрицу D.**
151. На вход принимает:
152. Возвращает:
153. D - матрица.
154. '''
155. **D = np.array([[1, mu , 0],**
156. [mu, 1, 0],
157. [0, 0, (1-mu)/2]])
159. **return** E \* D / (1-mu\*\*2)

162. **def** get\_B(x1, y1, x2, y2, x3, y3, x4, y4):
163. '''
164. Функция создает матрицу B.
165. **На вход принимает:**
166. координаты четырех точек четырехугольника.
167. Возвращает:
168. B - матрица.
169. '''
170. **B = np.array([[y2 - y4, y3 - y1],**
171. [x4 - x2, x1 - x3]])
172. s\_i = get\_area\_of\_sqr(x1, y1, x2, y2, x3, y3, x4, y4)
174. **return** B / (2 \* sum(s\_i) \* 0.5)
176. *# get\_B(x1=0, y1=0, x2=-1, y2=0, x3=0, y3=1)*

179. *# In[7]:*

182. **def** get\_dN(figs\_nodes, dict\_of\_nodes):
183. '''
184. '''
185. **N = np.shape(figs\_nodes)[0]**
186. M = len(dict\_of\_nodes)
188. brackets = **lambda** k, e: np.array([[0 , (1 - e), (e - k), (k - 1)],
189. [(e - 1), 0 , (1 + k), -(e + k)],
190. **[(k - e), -(1 + k), 0 , (1 + e)],**
191. [(1 - k), (k + e), -(1 + e), 0 ]]) / 8
193. coord = [[-1, -1], [1, -1], [1, 1], [-1, 1]]
194. brackets\_matrix\_all = [brackets(k, e) **for** (k, e) **in** coord]
196. XX = np.zeros((M, M))
197. YY = np.zeros((M, M))
198. XY = np.zeros((M, M))
200. **for fig\_id in range(N):**
201. curr\_fig=figs\_nodes[fig\_id]
203. coords=np.array([dict\_of\_nodes[curr\_fig[0]],
204. dict\_of\_nodes[curr\_fig[1]],
205. **dict\_of\_nodes[curr\_fig[2]],**
206. dict\_of\_nodes[curr\_fig[3]]])
208. X, Y = coords.T
209. XX\_local = np.zeros((4, 4))
210. **YY\_local = np.zeros((4, 4))**
211. XY\_local = np.zeros((4, 4))
213. **for** brackets\_matrix **in** brackets\_matrix\_all:
214. dN\_dX\_local = brackets\_matrix @ Y
215. **dN\_dY\_local = X @ brackets\_matrix**
216. J = np.abs(X @ dN\_dX\_local)
217. XX\_local += dN\_dX\_local[:, np.newaxis] @ dN\_dX\_local[np.newaxis, :] / J
218. YY\_local += dN\_dY\_local[:, np.newaxis] @ dN\_dY\_local[np.newaxis, :] / J
219. XY\_local += dN\_dX\_local[:, np.newaxis] @ dN\_dY\_local[np.newaxis, :] / J
221. **for** i **in** range(4):
222. XX[curr\_fig[i], curr\_fig] += XX\_local[i]
223. YY[curr\_fig[i], curr\_fig] += YY\_local[i]
224. XY[curr\_fig[i], curr\_fig] += XY\_local[i]
226. **return** XX, YY, XY
228. **def** get\_K\_global(figs\_nodes, dict\_of\_nodes):
229. '''
230. **'''**
231. M = len(dict\_of\_nodes)
232. K\_global = np.zeros((2\*M, 2\*M))
234. D = get\_D()
235. **XX, YY, XY = get\_dN(figs\_nodes, dict\_of\_nodes)**
237. K\_global[0::2, 0::2] = XX \* D[0, 0] + YY \* D[2, 2]
238. K\_global[1::2, 1::2] = YY \* D[1, 1] + XX \* D[2, 2]
239. K\_global[0::2, 1::2] = XY \* D[0, 1] + XY.T \* D[2, 2]
240. **K\_global[1::2, 0::2] = XY.T \* D[0, 1] + XY \* D[2, 2]**
242. **return** K\_global
244. **def** get\_F\_global(figs\_nodes, dict\_of\_nodes, F=20):
245. **'''**
246. '''
247. F\_global = np.zeros((2\*len(dict\_of\_nodes)))
249. **for** tr **in** figs\_nodes:
250. **x1, y1 = dict\_of\_nodes[tr[0]]**
251. x2, y2 = dict\_of\_nodes[tr[1]]
252. x3, y3 = dict\_of\_nodes[tr[2]]
253. x4, y4 = dict\_of\_nodes[tr[3]]
255. **original\_id = {0:tr[0], 1:tr[1], 2:tr[2], 3:tr[3]}**
257. x = [x1, x2, x3, x4]
258. y = [y1, y2, y3, y4]
260. **for i, j in zip([0, 1, 2, 3], [3, 0, 1, 2]):**
261. **if** (abs(x[i]) == l) & (abs(x[j]) == l):
262. delta = np.array([y[i]-y[j], x[j]-x[i]]).T
263. F\_global[2 \* original\_id[j]:2 \* original\_id[j] + 2] += F \* delta / (4 \* h \* t)
264. F\_global[2 \* original\_id[i]:2 \* original\_id[i] + 2] += F \* delta / (4 \* h \* t)
266. **return** F\_global
268. *# def main\_solver\_2(K\_global, F\_global, figs\_nodes, dict\_of\_nodes):*
269. *# '''*
270. ***# Вектор узловых перемещений находится как решение СЛАУ.***
271. *# На вход принимает:*
272. *# Возвращает:*
273. *# U - вектор узлового перемещения.*
274. *# '''*
275. ***# U = []***
276. *# U\_row = np.linalg.solve(K\_global, F\_global)*
278. *# for tr in figs\_nodes:*
279. *# x1, y1 = dict\_of\_nodes[tr[0]]*
280. ***# x2, y2 = dict\_of\_nodes[tr[1]]***
281. *# x3, y3 = dict\_of\_nodes[tr[2]]*
282. *# x4, y4 = dict\_of\_nodes[tr[3]]*
284. *# tmp = np.zeros((2, 4))*
285. ***# tmp[0][0] = U\_row[2 \* tr[0]]***
286. *# tmp[0][1] = U\_row[2 \* tr[1]]*
287. *# tmp[0][2] = U\_row[2 \* tr[2]]*
288. *# tmp[0][3] = U\_row[2 \* tr[3]]*
289. *# tmp[1][0] = U\_row[2 \* tr[0] + 1]*
290. ***# tmp[1][1] = U\_row[2 \* tr[1] + 1]***
291. *# tmp[1][2] = U\_row[2 \* tr[2] + 1]*
292. *# tmp[1][3] = U\_row[2 \* tr[3] + 1]*
294. *# U += [tmp.T]*
296. *# return U*
298. **def** reshape\_U(figs\_nodes, dict\_of\_nodes, U\_row):
299. '''
300. **'''**
301. U = []
303. **for** tr **in** figs\_nodes:
304. x1, y1 = dict\_of\_nodes[tr[0]]
305. **x2, y2 = dict\_of\_nodes[tr[1]]**
306. x3, y3 = dict\_of\_nodes[tr[2]]
307. x4, y4 = dict\_of\_nodes[tr[3]]
309. tmp = np.zeros((2, 4))
310. **tmp[0][0] = U\_row[2 \* tr[0]]**
311. tmp[0][1] = U\_row[2 \* tr[1]]
312. tmp[0][2] = U\_row[2 \* tr[2]]
313. tmp[0][3] = U\_row[2 \* tr[3]]
314. tmp[1][0] = U\_row[2 \* tr[0] + 1]
315. **tmp[1][1] = U\_row[2 \* tr[1] + 1]**
316. tmp[1][2] = U\_row[2 \* tr[2] + 1]
317. tmp[1][3] = U\_row[2 \* tr[3] + 1]
319. U += [tmp.T]
321. **return** U

324. **def** get\_M\_global(figs\_nodes, dict\_of\_nodes, ro, t=1):
325. **'''**
326. '''
327. N = len(dict\_of\_nodes)
328. M\_local = np.zeros((N, N))
329. M\_global = np.zeros((2\*N, 2\*N))
330. **simple = np.array([[4, 2, 1, 2],**
331. [2, 4, 2, 1],
332. [1, 2, 4, 2],
333. [2, 1, 2, 4]])/12
335. **for num, tr in enumerate(figs\_nodes):**
336. x1, y1 = dict\_of\_nodes[tr[0]]
337. x2, y2 = dict\_of\_nodes[tr[1]]
338. x3, y3 = dict\_of\_nodes[tr[2]]
339. x4, y4 = dict\_of\_nodes[tr[3]]
341. s\_i = get\_area\_of\_sqr(x1, y1, x2, y2, x3, y3, x4, y4)
342. S = sum(s\_i) \* 0.5
343. S\_local = simple\*(S + s\_i[:, np.newaxis] + s\_i[np.newaxis, :])
345. **M\_local[tr, tr[0]] += S\_local[0]**
346. M\_local[tr, tr[1]] += S\_local[1]
347. M\_local[tr, tr[2]] += S\_local[2]
348. M\_local[tr, tr[3]] += S\_local[3]
350. **M\_global[0::2, 0::2] = M\_local \* ro \* t / 6**
351. M\_global[1::2, 1::2] = M\_local \* ro \* t / 6
353. **return** M\_global

356. *# In[8]:*

359. **def** get\_stress(U, deformation, figs\_nodes, dict\_of\_nodes):
360. **'''**
361. Функция строит вектор напряжений.
362. На вход принимает:
363. U - вектор узловых перемещений;
364. deformation - вектор деформаций;
365. **figs\_nodes - последовательность индексов узлов, которые образуют фигуру;**
366. dict\_of\_nodes - словарь, ключом является индекс узла, а значвением является координаты этого узла;
367. Возвращает:
368. stress - вектор напряжений.
369. '''
370. **stress = [0]\*(len(figs\_nodes))**
372. **for** num, tr **in** enumerate(figs\_nodes):
373. D = get\_D()
374. stress[num] = D @ deformation[num]
376. **return** np.array(stress)

379. **def** get\_deformation(U, figs\_nodes, dict\_of\_nodes):
380. **'''**
381. Функция строит вектор деформаций.
382. На вход принимает:
383. U - вектор узловых перемещений;
384. figs\_nodes - последовательность индексов узлов, которые образуют фигуру;
385. **dict\_of\_nodes - словарь, ключом является индекс узла, а значвением является координаты этого узла.**
386. Возвращает:
387. deformation - вектор деформаций.
388. '''
389. deformation = [0]\*(len(figs\_nodes))
391. **for** num, tr **in** enumerate(figs\_nodes):
392. x1, y1 = dict\_of\_nodes[tr[0]]
393. x2, y2 = dict\_of\_nodes[tr[1]]
394. x3, y3 = dict\_of\_nodes[tr[2]]
395. **x4, y4 = dict\_of\_nodes[tr[3]]**
396. B = get\_B(x1, y1, x2, y2, x3, y3, x4, y4)
398. tmp = B @ (U[num][0:2] - U[num][2:4])
399. deformation[num] = [tmp[0][0], tmp[1][1], tmp[1][0] + tmp[0][1]]
401. **return** deformation

404. *# In[9]:*

407. **def** get\_R\_func(F, A, step\_i, t\_j):
408. '''
409. '''
410. **t = (np.pi \* (step\_i \* t\_j - 1))\*\*2**
411. func = F \* A \* (1 - 2 \* t) \* (np.e\*\*(-t))
412. **return** func
414. **def** main\_solver\_1(dict\_of\_nodes, M\_global, K\_global, F\_global, steps=1000, t=0.002):
415. **'''**
416. '''
417. N = len(dict\_of\_nodes)
418. U = np.zeros((2\*N, steps))
419. V = np.zeros((2\*N, steps))
420. **alpha = np.zeros((2\*N, steps))**
422. **for** i **in** range(steps-1):
423. f = get\_R\_func(F\_global, 1, i, t)
424. U[:,i + 1] = U[:,i] + V[:,i] \* t + alpha[:,i] \* t\*\*2 / 2.
425. **b = f - K\_global @ U[:,i+1]**
426. alpha[:, i+1] = np.linalg.solve(M\_global, b)
427. V[:, i+1] = V[:, i] + t \* (alpha[:, i] + alpha[:, i+1]) / 2.
429. **return** U, V

432. *# In[10]:*

435. **def use\_CP(value, figs\_nodes, dict\_of\_nodes):**
436. '''
437. Метод согласованных результантов.
438. Принимает на вход:
439. value - list со значениями величины (например, напряжения);
440. **figs\_nodes - последовательность индексов узлов, которые образуют фигуру;**
441. dict\_of\_nodes - словарь, ключом является индекс узла, а значвением является координаты этого узла.
442. Возвращает:
443. result - вектор со значениями результантов элементов
444. '''
445. **N = len(dict\_of\_nodes)**
446. R = np.zeros(N)
447. M\_local = np.zeros((N, N))
448. simple = np.array([[4, 2, 1, 2],
449. [2, 4, 2, 1],
450. **[1, 2, 4, 2],**
451. [2, 1, 2, 4]])/12
453. **for** num, tr **in** enumerate(figs\_nodes):
454. x1, y1 = dict\_of\_nodes[tr[0]]
455. **x2, y2 = dict\_of\_nodes[tr[1]]**
456. x3, y3 = dict\_of\_nodes[tr[2]]
457. x4, y4 = dict\_of\_nodes[tr[3]]
459. s\_i = get\_area\_of\_sqr(x1, y1, x2, y2, x3, y3, x4, y4)
460. **S = sum(s\_i) \* 0.5**
461. S\_local = simple\*(S + s\_i[:, np.newaxis] + s\_i[np.newaxis, :])
463. M\_local[tr, tr[0]] += S\_local[0]
464. M\_local[tr, tr[1]] += S\_local[1]
465. **M\_local[tr, tr[2]] += S\_local[2]**
466. M\_local[tr, tr[3]] += S\_local[3]
468. R[tr[0]] += (S + s\_i[0])\*value[num]
469. R[tr[1]] += (S + s\_i[1])\*value[num]
470. **R[tr[2]] += (S + s\_i[2])\*value[num]**
471. R[tr[3]] += (S + s\_i[3])\*value[num]
473. result = np.linalg.solve(M\_local, R)
475. **return result**

478. *# In[11]:*

481. **def** check\_tr(x1, y1, x2, y2, x3, y3, x0, y0):
482. '''
483. Функция проверяет, лежит ли точка в треугольнике.
484. На вход получает:
485. **координаты трех точек треугольника;**
486. коориднаты точки, которую проверяем.
487. Возвращает:
488. res - ответ типа bool.
489. '''
490. **Q = lambda x1, y1, x2, y2, x0, y0 : x0\*(y2-y1) + y0\*(x1-x2) + y1\*x2 - x1\*y2**
492. q1 = Q(x1, y1, x2, y2, x0, y0)
493. q2 = Q(x2, y2, x3, y3, x0, y0)
494. q3 = Q(x3, y3, x1, y1, x0, y0)
496. res = (((q1 > 0) **and** (q2 > 0) **and** (q3 > 0)) **or**
497. ((q1 <= 0) **and** (q2 <= 0) **and** (q3 <= 0)))
499. **return** res
501. **def** check\_sqr(x1, y1, x2, y2, x3, y3, x4, y4, x0, y0):
502. '''
503. '''
504. ch1 = (check\_tr(x1, y1, x2, y2, x4, y4, x0, y0) == True)
505. **ch2 = (check\_tr(x1, y1, x2, y2, x3, y3, x0, y0) == True)**
506. ch3 = (check\_tr(x4, y4, x2, y2, x3, y3, x0, y0) == True)
507. ch4 = (check\_tr(x1, y1, x4, y4, x3, y3, x0, y0) == True)
509. **if** sum([ch1, ch2, ch3, ch4]) > 0:
510. **return True**
511. **else**:
512. **return** False

515. **def find\_tr\_num(line, figs\_nodes, dict\_of\_nodes):**
516. '''
517. Функция ищет номера треугольников, которые содержат линию.
518. На вход получает:
519. line - линия;
520. **figs\_nodes - последовательность индексов узлов, которые образуют фигуру;**
521. dict\_of\_nodes - словарь, ключом является индекс узла, а значвением является координаты этого узла.
522. Возвращает:
523. line\_of\_num - словарь, ключом является индекс треугольника, а значением явзяется координаты лежащей в нем точки.
524. '''
525. **line\_of\_num = {}**
527. **for** x **in** line:
528. x0, y0 = x[0], x[1]
529. **for** num, tr **in** enumerate(figs\_nodes):
530. **x1, y1 = dict\_of\_nodes[tr[0]]**
531. x2, y2 = dict\_of\_nodes[tr[1]]
532. x3, y3 = dict\_of\_nodes[tr[2]]
533. x4, y4 = dict\_of\_nodes[tr[3]]

536. **if** check\_sqr(x1, y1, x2, y2, x3, y3, x4, y4, x0, y0) == True:
537. line\_of\_num[num] = x
539. **return** line\_of\_num

542. *# In[18]:*

545. ***# Запуск программы***
546. **global** E, mu, F, l, h, r, t, ro
548. E = 200
549. mu = 0.1
551. F = 20 *# Воздействие на боковые стороны*
552. l = 10 *# Расстояние от центра отверстия до одной из боковых сторон*
553. h = 5 *# Расстояние от центра отверстия до верхней и нижней сторон*
554. r = 3 *# Радиус отверстия*
555. **t = 1**
556. ro = 0.9
558. K\_global = get\_K\_global(figs\_nodes, dict\_of\_nodes)
559. F\_global = get\_F\_global(figs\_nodes, dict\_of\_nodes)
560. **M\_global = get\_M\_global(figs\_nodes, dict\_of\_nodes, ro)**
562. U, V = main\_solver\_1(dict\_of\_nodes, M\_global, K\_global, F\_global)
564. u\_i = reshape\_U(figs\_nodes, dict\_of\_nodes, U[:, int(0.8/0.002)-1])
565. **deformation = get\_deformation(u\_i, figs\_nodes, dict\_of\_nodes)**
566. stress = get\_stress(u\_i, deformation, figs\_nodes, dict\_of\_nodes)
568. *# U\_ = main\_solver\_2(K\_global, F\_global.T, figs\_nodes, dict\_of\_nodes)*
569. *# deformation = get\_deformation(U, figs\_nodes, dict\_of\_nodes)*
570. ***# stress = get\_stress(U, deformation, figs\_nodes, dict\_of\_nodes)***

573. *# In[20]:*

576. new\_dict\_of\_nodes = {}
578. **print**('Напряжение по оси X со смещением')
579. **for** i **in** [199, 399, 599, 799, 999]:
580. **u\_i = reshape\_U(figs\_nodes, dict\_of\_nodes, U[:, i])**
581. deformation = get\_deformation(u\_i, figs\_nodes, dict\_of\_nodes)
582. stress = get\_stress(u\_i, deformation, figs\_nodes, dict\_of\_nodes)
584. draw\_mesh(figs\_nodes,
585. **dict\_of\_nodes,**
586. stress[:,0])
587. *# plt.savefig('fig'+str(i))*

590. ***# In[ ]:***

593. *# x\_line = np.linspace([-r+0.00002, 0], [-l-0.00002, 0], 100)*
594. *# tr\_num = find\_tr\_num(x\_line, figs\_nodes, dict\_of\_nodes)*
595. ***# stress\_CP = use\_CP(stress[:,0], figs\_nodes, dict\_of\_nodes)***
597. *# plt.figure(figsize=(15, 10))*
598. *# plt.title('Напряжение по оси X', fontsize=15)*
599. *# plt.plot(np.array(list(tr\_num.values()))[:, 0],*
600. ***# stress[list(tr\_num.keys()), 0])***
601. *# plt.plot(np.array(list(tr\_num.values()))[:, 0],*
602. *# [sum(stress\_CP[i])/4 for i in np.array(figs\_nodes)[list(tr\_num.keys())]])*
603. *# plt.legend(['негладкое решение', 'решение методом СР'])*
604. *# # plt.savefig('fig4')*
605. ***# plt.show()***

608. *# In[22]:*

611. plt.figure(figsize=(15, 10))
612. plt.title('Напряжение по оси X', fontsize=15)
614. i = 0
615. **files\_list = sorted([filename for filename in glob.glob(r'meshes/\*.k')],**
616. reverse=False)
618. **for** filename **in** files\_list:
619. x, y, figs\_nodes, dict\_of\_nodes = read\_nodes\_file(filename)
621. K\_global = get\_K\_global(figs\_nodes, dict\_of\_nodes)
622. F\_global = get\_F\_global(figs\_nodes, dict\_of\_nodes)
623. M\_global = get\_M\_global(figs\_nodes, dict\_of\_nodes, ro)
625. **U, V = main\_solver\_1(dict\_of\_nodes, M\_global, K\_global, F\_global)**
627. motion\_steps = np.arange(0.1,2.1,0.1)
628. u\_t = {t: U[:, int(t / 0.002) - 1] **for** t **in** motion\_steps}
629. u\_in\_motion = {t: reshape\_U(figs\_nodes, dict\_of\_nodes, u\_t[t]) **for** t **in** motion\_steps}
630. **deformation\_in\_motion = {t:get\_deformation(u\_in\_motion[t], figs\_nodes, dict\_of\_nodes) for t in motion\_steps}**
631. stress\_in\_motion = {t:get\_stress(u\_in\_motion[t], deformation\_in\_motion[t], figs\_nodes, dict\_of\_nodes) **for** t **in** motion\_steps}
633. x\_line = np.linspace([-r+0.00002, 0], [-l-0.00002, 0], 10)[1]
634. tr\_num = find\_tr\_num([x\_line], figs\_nodes, dict\_of\_nodes)
635. **stress\_CP\_in\_motion = {t:use\_CP(stress\_in\_motion[t][:,0], figs\_nodes, dict\_of\_nodes) for t in motion\_steps}**
636. stress = 0
638. plt.plot(motion\_steps, [sum(stress\_CP\_in\_motion[t][figs\_nodes[list(tr\_num.keys())[0]]])/4 **for** t **in** motion\_steps], label = len(figs\_nodes))
639. plt.legend()
640. **print('finished for ', len(dict\_of\_nodes))**
641. plt.savefig('fig2')