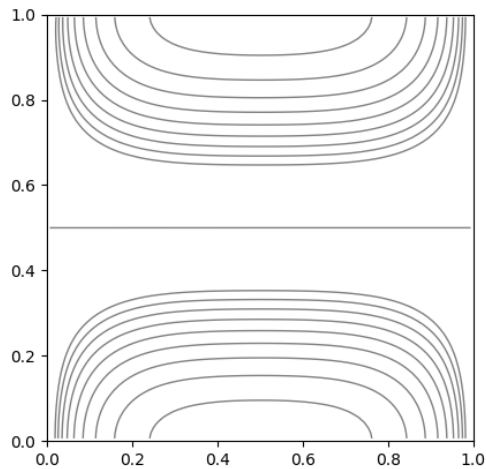


# תרגיל נומרי

208504050

23 ביוני 2019

## 1 מימוש וקוד.



מאחר, והממוצע הוא אופרטור לניארי, ניתן להסתכל על המרחב כאל וקטור

$$n^2 \times n^2 : \begin{bmatrix} \phi_{1,1} \\ \phi_{1,2} \\ \vdots \\ \phi_{n,n} \end{bmatrix} = \phi$$

$$n^2 \times \frac{1}{4} \begin{bmatrix} 1 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\ 0 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & 0 & \cdot & 1 & 0 & 1 & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} = M$$

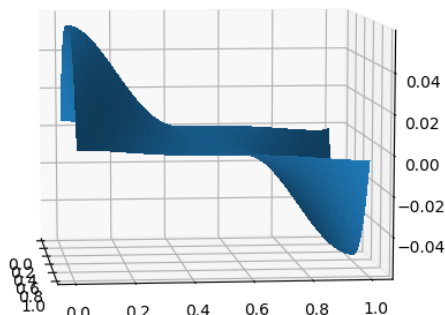
. כאשר מכפלת השורה ה- $i$  ב- $\phi$ . צריכה למעשה להחזיר לנו את

$$[M \cdot \phi]_i = \frac{\phi_{x-1,y} + \phi_{x+1,y} + \phi_{x,y-1} + \phi_{x,y+1}}{4}$$

. ( פרט ל  $n$  השורות הראשונות והאחרונות המוודאות שהפוטנציאל על השפה ישאר זהה ). היופי הוא שבקוד הדבר נראה כך (השטרודל בפיתרון מייצג כפל מטריצות):

```
62
63 def calc_iteration(laplace_vec, laplace_matrix):
64     ret = 0.25 * laplace_matrix @ laplace_vec
65     diff = abs( ret - laplace_vec )
66     maxdiff = np.max( diff )
67     return ret , maxdiff
68
69 def calc( laplace_vec, eps, harmonic=False):
70     laplace_matrix = init_operator(harmonic)
71
72     laplace_vec, maxdiff = calc_iteration(laplace_vec, laplace_matrix)
73     fn = [ maxdiff ]
74     while maxdiff > eps :
75         laplace_vec, maxdiff = calc_iteration(laplace_vec, laplace_matrix)
76         fn.append( maxdiff )
77     return laplace_vec , fn
```

וכפי שההינו מצפים, הפוט' בקרבת הלוחות עולה באופן שמזכיר לניאריות. זאת מאחר והשדה שם, בהתאם לציפיה שואף להיות קבוע (בקרבת הלוחות, הלוחות הם ביקרוב אין-סופים). להלן תמונה בתלת מימד :



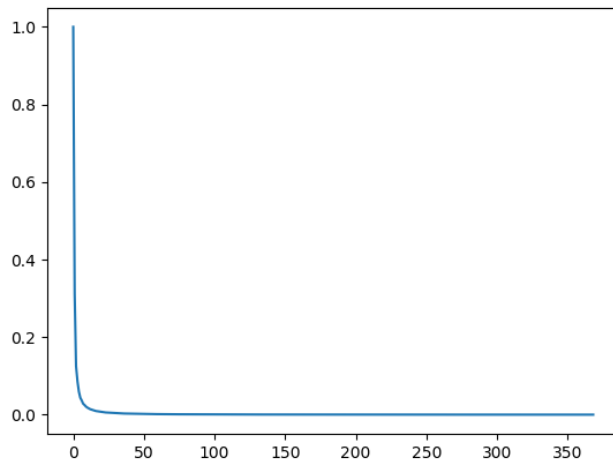
## 2 חישוב פוט', שדה, רזולציה.

נשים לב כי המשוואה -

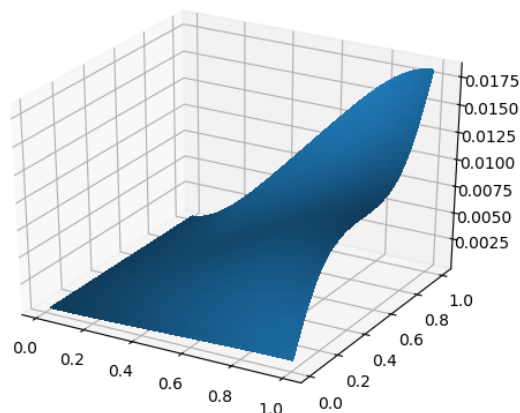
$$\phi_{i,j}^{n+1} = \frac{1}{2} \cdot \frac{1}{\Delta x^2 + \Delta y^2} \cdot [\Delta y^2(\phi_{i-1,j}^n + \phi_{i+1,j}^n) + \Delta x^2(\phi_{i,j-1}^n + \phi_{i,j+1}^n)]$$

היא למעשה לקיחת הממוצע של  $\phi^n$  בסביבת  $i, j$ . כלומר, למעשה, אנחנו מציבים ב  $\phi_{i,j}$  את ממוצע "השפה העוטפת" את נק'  $i, j$ . להלן, גרפים של הפוטנציאל, לאחר חלוקת המקטע ל-80 חלקים :

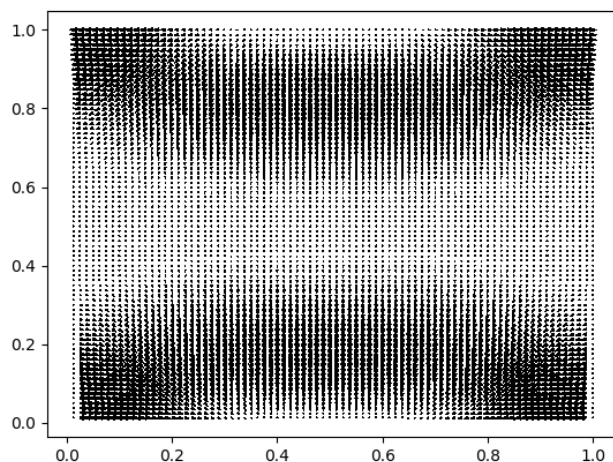
גרף ההתכנסות,  $f(t)$  המציג את,  $\max_{i,j} |\phi_{i,j}^{t+1} - \phi_{i,j}^t|$ , כלומר את גודל התיקון המקסימלי, ניתן לראות כיצד האיטרציות הראשונות מתקנות באופן יחסתי משמעותי, עד אשר אנו מגיעים לנקודה קריטית מימנה, השיפור מאיטרציה לאיטרציה כמעט ולא קיים:



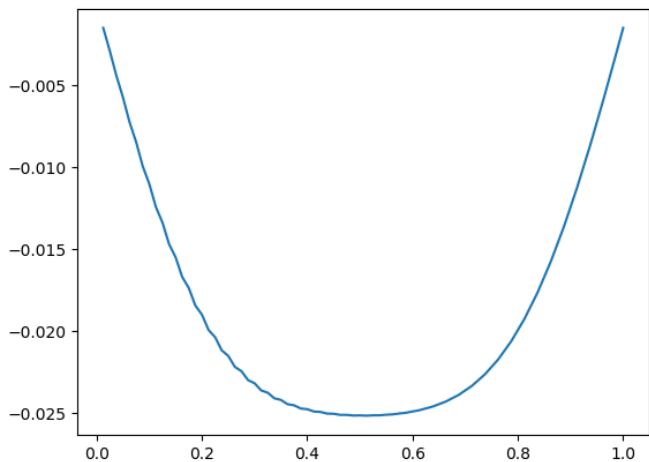
גרף הרזולוציה  $g(n)$ , המציג את הערכים  $|g(n_x, n_y) - g(\frac{n_x}{2}, \frac{n_y}{2})|$ :



השדה, החשמלי. ניתן לראות כי השדה בהחלט שואף להיות קבוע. כפי שההינו מצפים מהתאוריה.



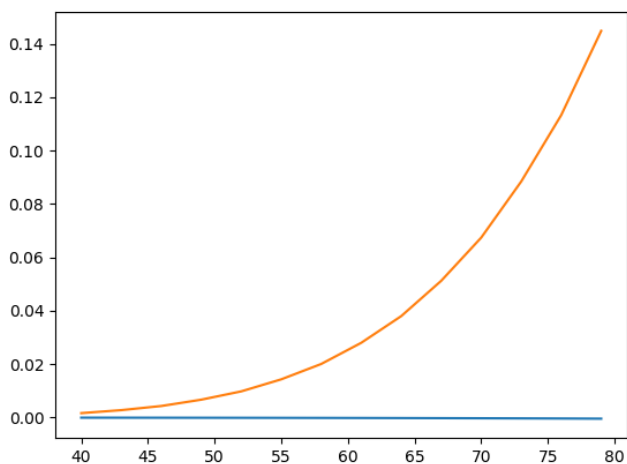
צפיפות, המטען על הלוח התחתון (בעל פוטנציאל חיובי), ניתן לראות כי האלקטרונים במרכזו. בדומה לתוצאה שראינו בהרצאה בעת חישוב צפיפות



### 3 האנרגיה במערכת

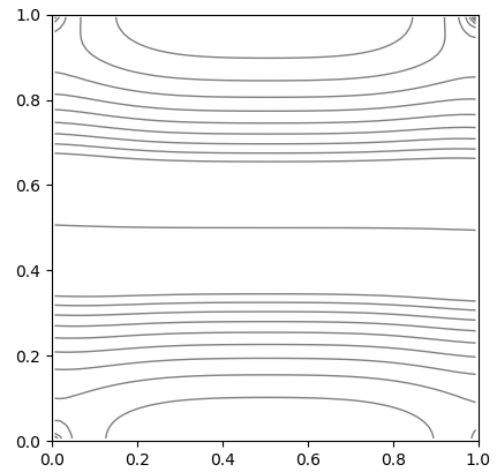
$$\frac{1}{2} \iint_{\partial V} \Delta \phi \cdot \phi = \frac{1}{2} \iiint_V \Delta \phi \phi = \frac{\epsilon_0}{2} \iint_{\partial V} E \cdot \phi - \frac{\epsilon_0}{2} \iiint_V E^2$$

המעבר הראשון נובע מאחר ו  $\Delta \phi = 0$  לכל נק' בתוך התיבה שלא על השפה. נשים לב כי מסימטריה, נקבל כי השדות על השפה, מצביעים מצד אחד אלואותו כיוון מצד שני,  $\phi$  על השפות שונה בסימנו ולכן  $\frac{\epsilon_0}{2} \iint_{\partial V} E = 0$ . נקבל בסך הכל כי:  $\frac{1}{2} \iint_{\partial V} \Delta \phi \cdot \phi = -\frac{\epsilon_0}{2} \iiint_V E^2$ . לא הצלחנו להגיע לשיוון מוחלט, יחד עם זאת ניתן בהחלט לראות כי בין שתי צורות החישוב מבדיל פקטור קבוע. בציר ה  $y$  אנו רואים סכימה על הנפח, בציר  $x$  סכימה על השפה.

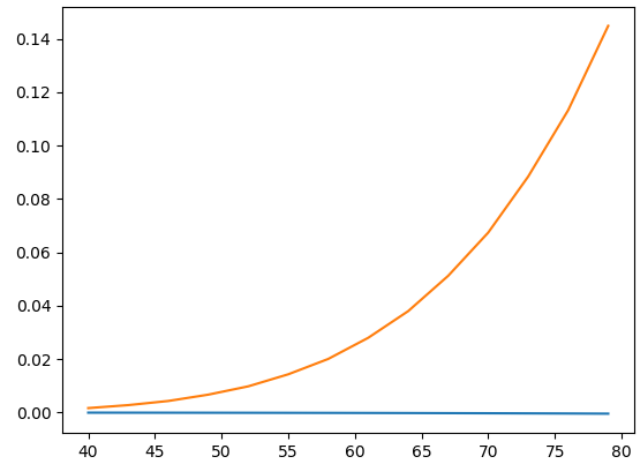


### 4 מישור אינסופי.

הפוטנציאל, תנאים מחזוריים, ניתן בהחלט לראות כי הפתרון מתכנס ללוח אינסופי.



בכל אופן לא הצלחנו להתכנס לערך זהה בפוטנציאלים. **הפשלה העיקרית** **נובעת** ככל הנראה מיכך שהשתמשנו בכפל מטריצה שמימדיה  $n^2 \times n^2$ . עבור  $n > 80$ , משתגרת חריגה של ניצול זיכרון. בניראה מאחר והזיכרון שהמטריצה דורשת חורגת ממגבלת הזיכרון של ה-*RAM*. אז בניסיון שלנו לצאת עצלנים ולכתוב כמה שפחות קוד, הבנו רק לקראת הסוף שאין ארוחות חינם, שזה מוסר ההשכל האמיתי מכל הסיפור והוא שווה הרבה יותר מפונק' פוטנציאל. בכל זאת מצורף פה הגרף של חישוב הפוטנציאל בשתי דרכים.



```

1 import numpy as np
2 from scipy.optimize import curve_fit
3 from mpl_toolkits.mplot3d import Axes3D
4 from matplotlib import pyplot as plt
5 from matplotlib.backends.backend_pdf import PdfPages
6 import matplotlib.cm as cm
7 import matplotlib.mlab as mlab
8 from math import pi
9
10
11 size = 80
12
13 def init_operator( harmonic = False ):
14     ret = np.zeros( shape = ( size*size , size*size ))
15
16     for raw in range( size*size ):
17
18         if raw % size != 0 or harmonic:
19             ret[raw][raw-1] = 1
20         if raw % size != size - 1 or harmonic:
21             ret[raw][(raw+1) % (size*size)] = 1
22         if raw >= size :
23             ret[raw][raw - size] = 1
24         if raw < size * (size - 1):
25             ret[raw][raw + size] = 1
26         if raw < size:
27             ret[raw][raw] = 1.
28         if raw > size * (size - 1):
29             ret[raw][raw] = 1.
30
31     return ret
32
33 def init_E_fieldX_oprator( harmonic = False ):
34     ret = np.zeros( shape = ( size*size , size*size ))
35
36     for raw in range( size*size ):
37         if not harmonic :
38             if raw % size != 0 and raw % size != size - 1:
39                 ret[raw][raw] = -1
40                 ret[raw][raw+1] = 1
41             else:
42                 ret[raw][raw] = -1
43                 ret[raw][(raw+1) % size*size] = 1
44
45
46     return ret
47
48 def init_E_fieldY_oprator():
49     ret = np.zeros( shape = ( size*size , size*size ))
50
51     for raw in range( size*size ):
52         if raw < size * (size - 1):
53             ret[raw][raw] = -1
54             ret[raw][raw + size] = 1
55
56     return ret
57
58 laplace_matrix = []
59 EX_operator = [] #init_E_fieldX_oprator()
60 EY_operator = [] #init_E_fieldY_oprator()
61
62
63 def calc_iteration(laplace_vec, laplace_matrix):
64     ret = 0.25 * laplace_matrix @ laplace_vec
65     diff = abs( ret - laplace_vec )
66     maxdiff = np.max( diff )
67     return ret , maxdiff
68
69 def calc( laplace_vec, eps, harmonic=False):
70     laplace_matrix = init_operator(harmonic)
71
72     laplace_vec, maxdiff = calc_iteration(laplace_vec, laplace_matrix)

```

```

73     fn = [ maxdiff ]
74     while maxdiff > eps :
75         laplace_vec, maxdiff = calc_iteration(laplace_vec, laplace_matrix)
76         fn.append( maxdiff )
77     return laplace_vec , fn
78
79 def reso_matrix (matrix, nx, ny):
80
81     reso = np.zeros( shape = (nx, ny) )
82
83     reso[0,0] = abs(matrix[0,0])
84
85     for j in range(1 , ny ):
86         reso[0 , j] = reso[0 , j-1] + abs(matrix[0, j])
87
88     for i in range(1 , nx ):
89         reso[i , 0] = reso[i-1 , 0] + abs(matrix[i, 0])
90
91
92     for i in range(1 , nx):
93         for j in range(1 , ny):
94             reso[i , j] = abs(matrix[i, j]) + reso[i, j - 1] + \
95                 reso[i-1, j] - reso[i-1, j-1]
96
97     reso = reso / ( nx * ny )
98
99     plotreso = np.zeros( shape = (nx, ny) )
100
101     eps = 10
102
103     for i in range(1 , nx):
104         for j in range(1 , ny):
105             if abs(reso[i , j] - reso[i // 2 , j // 2]) < eps:
106                 plotreso[i,j] = reso[i,j]
107
108     return reso
109
110 def electricfield(laplace_vec, harmonic = False) :
111
112     global laplace_matrix
113     laplace_matrix = None
114
115     EX_operator = init_E_fieldX_oprator(harmonic)
116     EY_operator = init_E_fieldY_oprator()
117
118     Ey = EY_operator @ laplace_vec * size
119     Ex = EX_operator @ laplace_vec * size
120
121     U , V = ( np.zeros( shape=(size,size) ) for _ in range(2) )
122
123     for i in range(size):
124         for j in range(size):
125             V[i,j] = Ex[i*size + j%size]
126             U[i,j] = Ey[i*size + j%size]
127
128     EX_operator = None
129     EY_operator = None
130
131     return V, U
132
133 def chargeDist(laplace_vec):
134     global laplace_matrix
135     laplace_matrix = None
136
137     V = np.zeros( size )
138     U = np.zeros( size )
139
140     EY_operator = init_E_fieldY_oprator()
141     Ey = EY_operator @ laplace_vec * size
142
143     for i in range(size):
144         V[i] = Ey[i]
145         U[i] = Ey[-i]
146     return V , U

```

```

147
148 def calculateEnergySurface(laplace_vec, charge1 , charge2):
149     ret = 0.0
150
151     for i in range(10 ,size-10):
152         ret += laplace_vec[i]* charge1[i]
153         ret += laplace_vec[-i]*charge2[i]
154
155     return ret / 2
156
157 def calculateEnergyVolume(Ex ,Ey):
158     ret = 0.0
159
160     for i in range(10 , size-10):
161         for j in range(size):
162             ret += Ex[i,j]**2 + Ey[i,j]**2
163
164     return ret / (8 * pi)
165
166 if __name__ == '__main__':
167
168     #print(laplace_matrix)
169     # X = np.arange(-5, 5, 0.25)
170     # Y = np.arange(-5, 5, 0.25)
171     # X, Y = np.meshgrid(X, Y)
172     # R = np.sqrt(X**2 + Y**2)
173     # print(R)
174
175     section = [ False, False, False, False , True, True, True, True ]
176
177     state = np.zeros(size*size)
178
179     for i in range(size):
180         state[i] = 1.
181         state[-i] = -1.
182
183     state , fn = calc( state, 0.0001 )
184
185     V = np.zeros(shape =(size,size))
186
187     for i in range(size):
188         for j in range(size):
189             V[i,j] = state[i*size + j*size]
190             print( "{0: f}".format(state[i*size + j*size]) , end = " ")
191         print()
192
193
194     x_range = 1
195     delta_x = 1.0 / size
196     delta_y = delta_x
197     y_range = x_range
198     xvec = np.arange(delta_x ,x_range+delta_x,delta_x)
199     yvec = np.arange(delta_y ,y_range+delta_y,delta_y)
200     Xgrid, Ygrid = np.meshgrid(xvec,yvec)
201
202     if section[0] :
203         plt.figure()
204         xp = np.exp(np.arange(-5., 0.5, 0.25))
205         xm = [-x for x in xp[::-1]]
206         zz = [0.]
207         levels = np.concatenate((xm,zz,xp))
208         # plt.streamplot(Xgrid, Ygrid, Ex, Ey,
209         #                 color='r',linewidth=1.5,density=[0.75,1.]
210         #                 ,maxLength=10.)
211         CS = plt.contour(V, levels,
212                         origin='lower',colors='grey',
213                         linestyles='solid',
214                         linewidths=1.,
215                         extent=(0, x_range, 0, y_range))
216         plt.axis([0,x_range,0,y_range])
217         plt.axes().set_aspect('equal','box')
218         plt.show()
219
220     if section[1]:

```

```

221     fig = plt.figure()
222     ax = fig.gca(projection='3d')
223     print(V)
224     print(Xgrid)
225     surf = ax.plot_surface(Xgrid, Ygrid, V,
226                           linewidth=0, antialiased=False)
227     plt.show()
228
229 if section[2]:
230     fig = plt.figure()
231     plt.plot( [ i for i in range( len(fn) ) ] , fn )
232     plt.show()
233
234 if section[3]:
235     fig = plt.figure()
236     ax = fig.gca(projection='3d')
237     surf = ax.plot_surface(Xgrid, Ygrid, reso_matrix(V , size, size),
238                           linewidth=0, antialiased=False)
239     plt.show()
240
241 if section[4]:
242     Ex, Ey = electricfield(state)
243
244     fig, ax = plt.subplots()
245     # q = ax.quiver(X, Y, U, V)
246     # ax = fig.gca(projection='2d')
247     q = ax.quiver(Xgrid, Ygrid, Ex, Ey,
248                  linewidth=0, antialiased=False)
249     plt.show()
250
251
252 if section[5]:
253     V , U = chargeDist(state)
254     plt.plot( xvec , V )
255     plt.show()
256
257
258 if section[6]:
259
260     XX = []
261     YY = []
262     for i in range(40 , 80 , 3):
263         size = i
264         state = np.zeros(size*size)
265         for j in range(size):
266             state[j] = 1.
267             state[-j] = -1.
268         state , fn = calc( state, 0.00001 )
269         Ex, Ey = electricfield(state)
270         V , U = chargeDist(state)
271
272
273         YY.append(calculateEnergyVolume(Ex, Ey))
274         XX.append(calculateEnergySurface(state, V , U ))
275     fig = plt.figure()
276     print(XX)
277     print(YY)
278     #plt.plot( [ i for i in range(len(XX)) ] , [ x / y for x , y in zip(XX, YY) ] )
279     ZZ = [ i for i in range(40 , 80 , 3) ]
280     plt.plot(ZZ, XX)
281     plt.plot(ZZ, YY)
282     plt.show()
283
284 if section[7]:
285     size = 80
286     state = np.zeros(size*size)
287     for i in range(size):
288         state[i] = 1.
289         state[-i] = -1.
290
291     state , fn = calc( state, 0.00001, harmonic=True )
292     Ex, Ey = electricfield(state, harmonic=True)
293
294     V = np.zeros(shape =(size,size))

```

```

295
296     for i in range(size):
297         for j in range(size):
298             V[i,j] = state[i*size + j*size]
299             print( "{0: f}".format(state[i*size + j*size]) , end = " ")
300         print()
301
302     plt.figure()
303     xp = np.exp(np.arange(-5., 0.5, 0.25))
304     xm = [-x for x in xp[::-1]]
305     zz = [0.]
306     levels = np.concatenate((xm,zz,xp))
307     # plt.streamplot(Xgrid, Ygrid, Ex, Ey,
308     #               color='r',linewidth=1.5,density=[0.75,1.]
309     #               ,maxLength=10.)
310     CS = plt.contour(V, levels,
311                     origin='lower',colors='grey',
312                     linestyle='solid',
313                     linewidths=1.,
314                     extent=(0, x_range, 0, y_range))
315     plt.axis([0,x_range,0,y_range])
316     plt.axes().set_aspect('equal','box')
317     plt.show()
318
319     XX = []
320     YY = []
321     for i in range(40 , 80 , 3):
322         size = i
323         state = np.zeros(size*size)
324         for j in range(size):
325             state[j] = 1.
326             state[-j] = -1.
327         state , fn = calc( state, 0.00001, harmonic=True)
328         Ex, Ey = electricfield(state, harmonic=True)
329         V , U = chargeDist(state)
330
331
332         YY.append(calculateEnergyVolume(Ex, Ey))
333         XX.append(calculateEnergySurface(state, V , U ))
334     fig = plt.figure()
335     print(XX)
336     print(YY)
337     #plt.plot( [ i for i in range(len(XX)) ] , [ x / y for x , y in zip(XX, YY) ] )
338     ZZ = [ i for i in range(40 , 80 , 3) ]
339     plt.plot(ZZ, XX)
340     plt.plot(ZZ, YY)
341     plt.show()
342

```