Example-gradient-less-optimization

November 23, 2019

[1]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
import examples as eg
import numpy as np
from numpy import *
import dionysus
```

The circular coordinates pipeline for examining different smoothness cost-functions:

```
Step 1. Getting the point cloud
Step 2. Computing the Vietoris-Rips filtration and its cohomology
Step 3. Selecting the Cocycle
Step 4. First smoothing using Least Squares (Optional)
Step 5. Second smoothing using a new cost function
```

0.1 Step 1 - Getting the point cloud

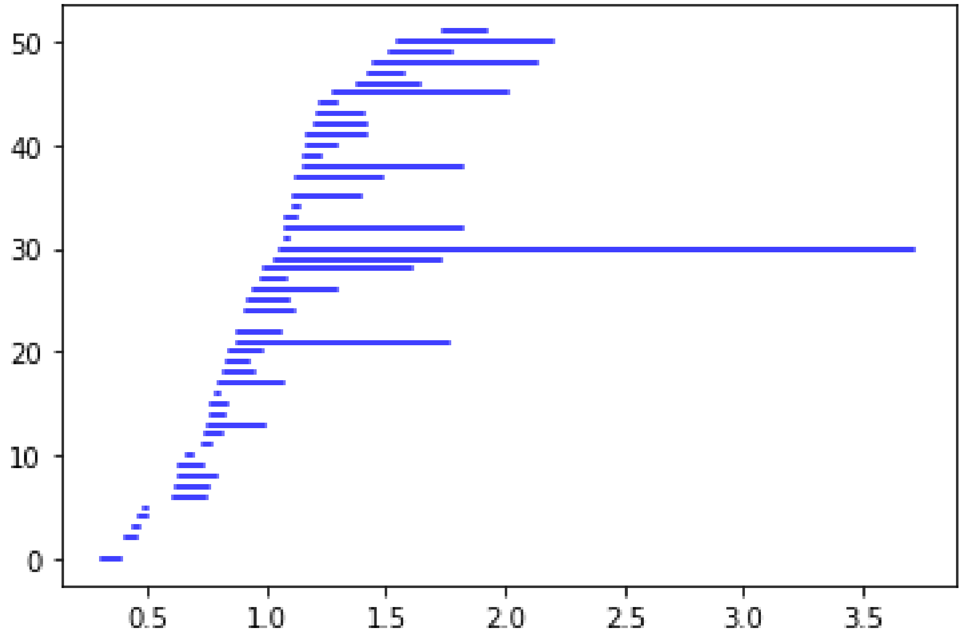
[3]:
```
annulus = eg.pinched_torus_example(n=200)
#The examples.py generates data points in form of point clouds that can be␣
↪analyzed using the imported dionysus module.
```

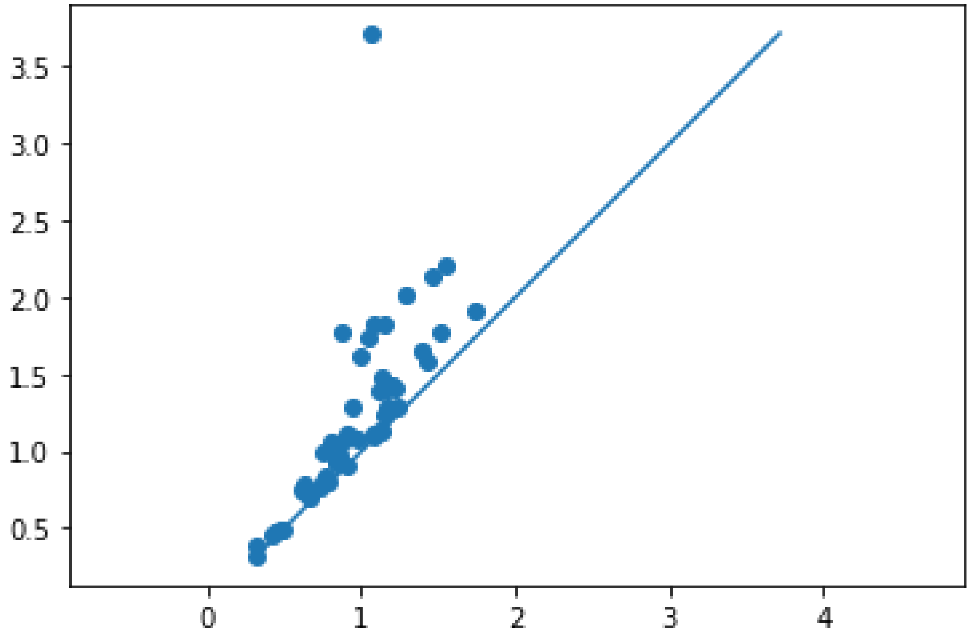
0.2 Step 2 - Computing Vietoris-Rips Complexes and Cohomology

[4]:
```
prime = 23 #choose the prime base for the coefficient field that we use to␣
↪construct the persistence cohomology.

vr = dionysus.fill_rips(annulus, 2, 4.) #Vietoris-Rips complex
cp = dionysus.cohomology_persistence(vr, prime, True) #Create the persistent␣
↪cohomology based on the chosen parameters.
dgms = dionysus.init_diagrams(cp, vr) #Calculate the persistent diagram using␣
↪the designated coefficient field and complex.
dionysus.plot.plot_bars(dgms[1], show=True)
dionysus.plot.plot_diagram(dgms[1], show=True)
#dionysus.plot.plot_diagram(dgms[0], show=True)
```

```
#Plot the barcode and diagrams using matplotlib incarnation within Dionysus2.
↪This mechanism is different in Dionysus.
```





0.3 Step 3 - Selecting the cocycle and visualization.

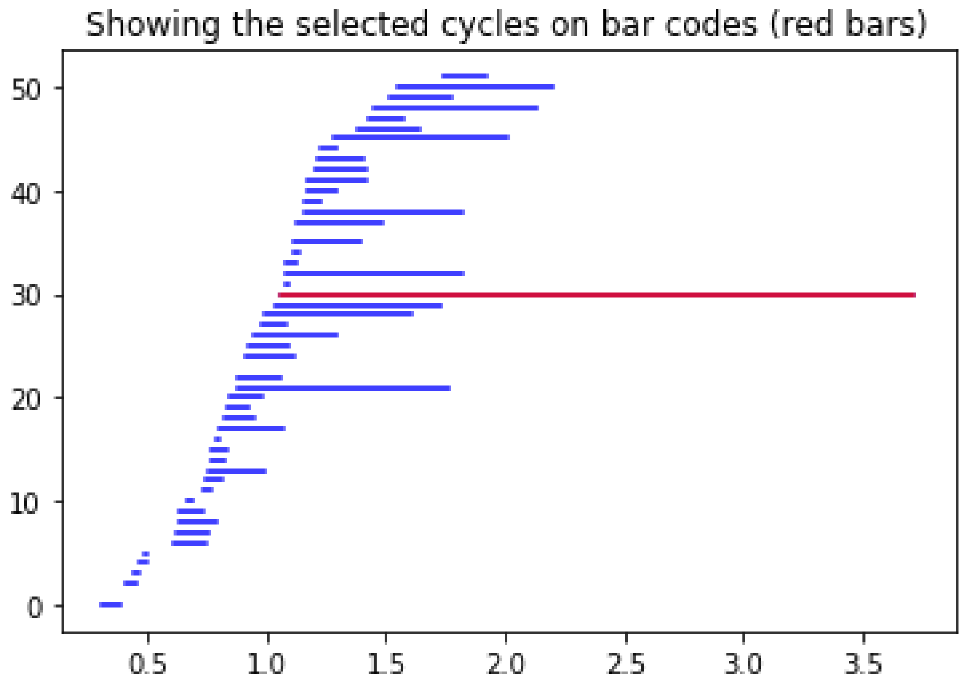
[5]:
```
type(plt)
```

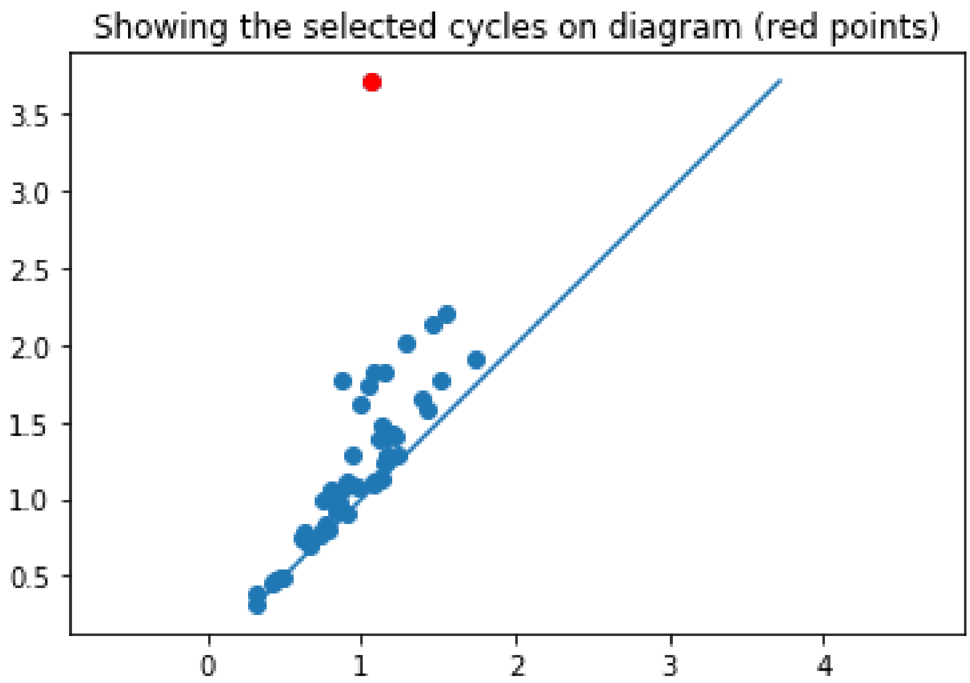
[5]: module

[5]:
```
threshold = 1
bars = [bar for bar in dgms[1] if bar.death-bar.birth > threshold] #choosing␣
↪cocycle that persist at least threshold=1.
cocycles = [cp.cocycle(bar.data) for bar in bars]
#plt is the matplotlib incarnation.

#Red highlight cocyles that persist more than threshold value on barcode, when␣
↪more than one cocyles have persisted over threshold values, this plots the␣
↪first one.
dionysus.plot.plot_bars(dgms[1], show=False)
plt.plot([[bar.birth,bar.death] for bar in dgms[1] if bar.death-bar.birth >␣
↪threshold][0],[[x,x] for x,bar in enumerate(dgms[1]) if bar.death-bar.birth␣
↪> threshold][0],'r')
plt.title('Showing the selected cycles on bar codes (red bars)')
plt.show()

#Red highlight ***ALL*** cocyles that persist more than threshold value on␣
↪diagram.
dionysus.plot.plot_diagram(dgms[1], show=False)
Lt1 = [[point.birth,point.death] for point in dgms[1] if point.death-point.
↪birth > threshold]
for Lt3 in Lt1:
    #print(Lt3)
    plt.plot(Lt3[0],Lt3[1],'ro')
plt.title('Showing the selected cycles on diagram (red points)')
plt.show()
```





[7]:
```
chosen_cocycle= cocycles[0]
chosen_bar= bars[0]
```

0.4 Step 4 - First smoothing using Least Squares (Optional)

If it is computed the smoothed coefficients can be used as initial condition for the optimization code

[8]:
```
vr_8 = dionysus.Filtration([s for s in vr if s.data <= max([bar.birth for bar␣
 ↪in bars])])
coords = dionysus.smooth(vr_8, chosen_cocycle, prime)
```

0.4.1 Visualization

[9]:
```
np.shape(annulus.T)
annulus.T[1,:]
```

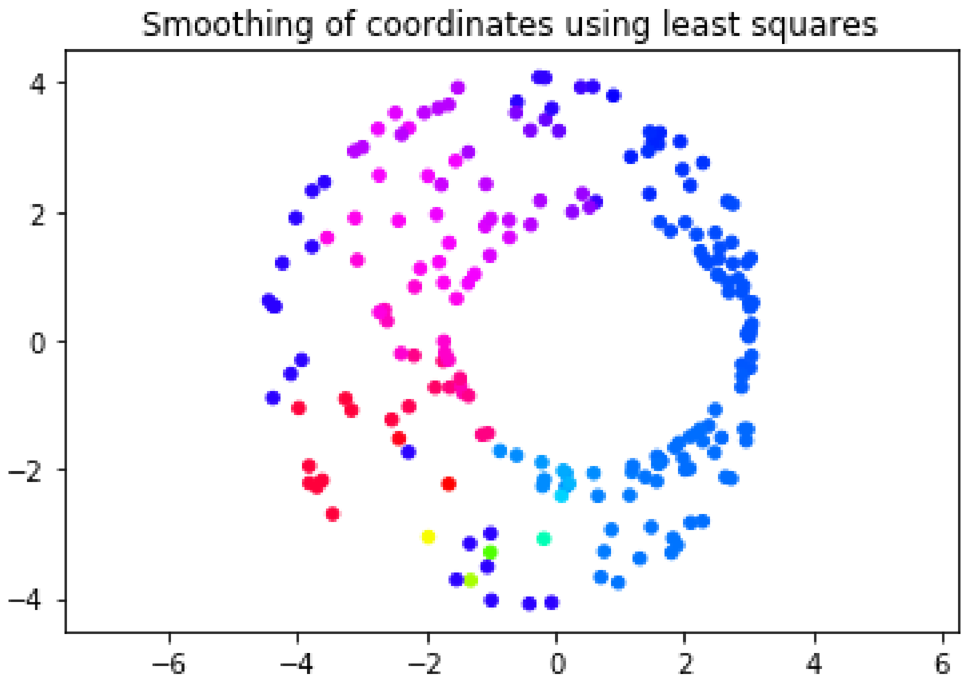
[9]:
```
array([-2.00508897,  1.20362447,  2.40415289,  3.04326264,  2.1568479 ,
        3.93622072,  1.27962633,  1.90079576,  3.92236424,  1.21402323,
        0.09961478, -1.43949787,  0.11915736, -0.90601696, -0.23756278,
       -2.16994545, -2.05953864, -2.0209653 , -1.98722423,  1.83351228,
       -0.23450408, -0.72958746, -1.07602218,  3.06613617,  3.52966921,
       -2.05974359, -2.25915   ,  1.19311805, -1.49262047,  1.85765028,
       -0.37499451, -0.81360144, -1.46045023,  2.15032601,  0.9576045 ,
        0.83230723, -1.0499545 , -2.26680817,  2.11450556,  0.30349523,
       -2.80436707,  3.65778482,  1.59307866, -3.72148371,  3.21935934,
       -3.14894429,  1.95540164, -0.51859973, -1.73901113,  1.52006897,
        3.92977568, -1.22762048,  0.89503248,  0.58962693,  2.84903371,
        1.11616751, -1.02539547,  0.05390425,  0.74388582,  1.66779298,
        0.64971302, -1.32349791,  0.51069434,  3.29352978, -2.40809596,
        3.79772269,  2.16727999,  3.28163315, -2.11894721, -4.03161433,
       -3.67115429, -1.86131828, -2.82578051,  0.7488461 ,  2.93668456,
       -1.79717501, -3.38042814, -0.20192614,  3.60356166,  1.26335588,
        0.52975111,  0.99859274, -0.31758155, -0.18784847, -2.22705332,
       -2.18491337, -2.15637596,  2.42586189,  3.25828906,  2.78350236,
       -0.86019629,  1.31740927,  3.42874808,  3.59946848, -3.7110652 ,
       -2.6932765 , -1.88043291,  3.23202772, -0.69327988,  2.06814286,
        0.18670734, -1.78272294,  2.56188977, -2.27540141, -1.5860532 ,
       -2.14224577, -3.06653512, -4.08530259,  1.77404725, -1.42331708,
        1.79586028,  2.3237186 ,  0.90302734,  1.24450941,  1.26208024,
       -1.82732811,  0.84230055,  0.53118069, -1.0815197 , -2.21365328,
       -3.29166122, -4.06465115, -3.04719054, -1.37715185, -2.41216368,
        2.4557141 ,  1.99461499, -3.07608331,  1.02806154, -0.01521507,
        4.08237977, -1.6617977 ,  0.88654172, -2.04420602,  2.27496134,
       -0.30460317, -0.89168836,  2.93311315,  3.5317377 , -0.58726041,
        0.58690553, -0.42582167, -2.93382506,  1.4355243 ,  1.89775831,
       -1.94499854,  1.64311387, -3.75642747, -1.71304345, -1.56227373,
       -0.73003982, -0.7242132 , -3.27368216,  1.8893643 , -1.52255221,
        0.61866326,  2.4179876 , -1.89013811,  1.19703588,  3.19322283,
```

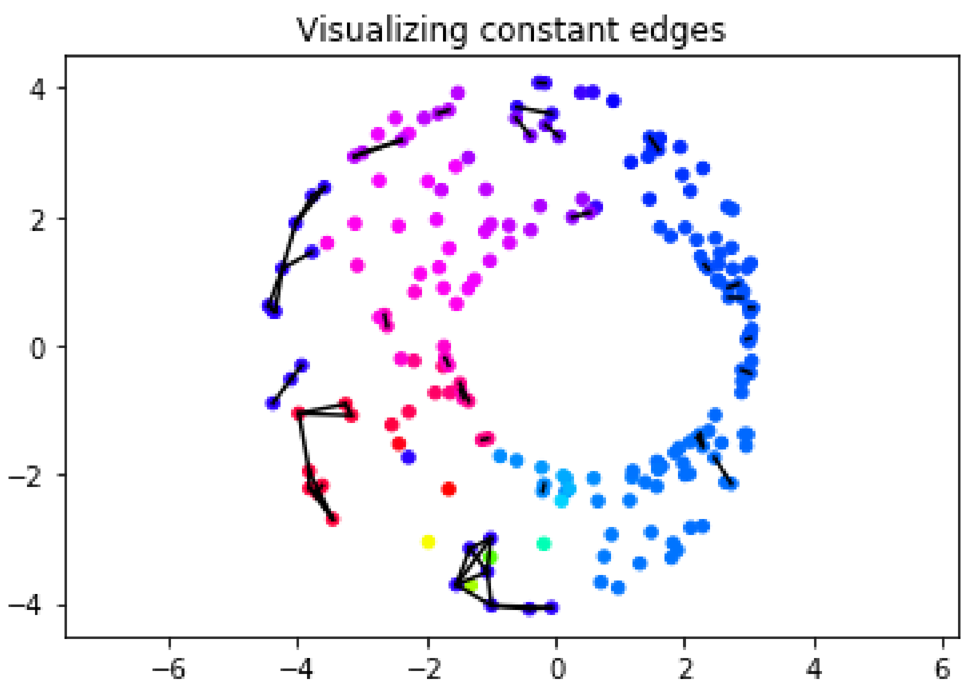
```
 2.99708416,  1.59782871,  2.65256463,  0.47483487,  1.45377251,
-1.94779554, -3.28603361,  1.01984503,  3.69970845, -1.50719167,
-2.89676935, -2.4032065 ,  3.24917405, -1.37622331, -0.54917669,
 1.82943857, -3.17478092,  1.19912508,  0.4392977 ,  2.27489728,
 3.52988491, -1.92106385, -1.37697533,  1.69870844, -2.11974327,
-1.73225712,  2.91900369,  1.38109927,  1.86577923,  3.0841793 ,
-0.30301528,  0.25470016,  1.51202881,  2.55250551,  2.75418444,
-2.22244898, -3.51102852,  4.07539552, -2.99252273, -1.55649298])
```

[10]:

```
#plt.rcParams['lines.markersize'] = 150
scatter(annulus.T[0,:],annulus.T[1,:],s=20, c=coords, cmap="hsv")
plt.axis('equal')
plt.title('Smoothing of coordinates using least squares')
plt.show()

toll = 1e-5
p,val = (chosen_bar,coords)
edges_costant = []
thr = p.birth # i want to check all edges that were there when the cycle was␣
↪created
for s in vr:
    if s.dimension() != 1:
        continue
    elif s.data > thr:
        break
    if abs(val[s[0]]-val[s[1]]) <= toll:
        edges_costant.append([annulus[s[0],:],annulus[s[1],:]])
edges_costant = np.array(edges_costant)
scatter(annulus.T[0,:],annulus.T[1,:],s=20, c=coords, cmap="hsv")
plot(edges_costant.T[0,:],edges_costant.T[1,:], c='k')
plt.axis('equal')
plt.title('Visualizing constant edges')
plt.show()
```


Smoothing of coordinates using least squares
4
2
0
−2
−4
−6
−4
−2
0
2
4
6


Visualizing constant edges
4
2
0
−2
−4
−6
−4
−2
0
2
4
6

0.5 Step 5 - Second smoothing using a new cost function

[11]:
```
import utils
l2_cocycle,f,bdry = utils.optimizer_inputs(vr, bars, chosen_cocycle, coords,␣
↪prime)
```

[62]:
```
#l2_cocycle.reshape(-1, 1)
l2_cocycle = l2_cocycle.reshape(-1, 1)
l2_cocycle.shape
f-bdry*l2_cocycle
```

[62]:
```
array([[ 1.34261787e-04],
       [ 0.00000000e+00],
       [ 7.21521040e-04],
       …,
       [-7.25702799e-01],
       [ 2.54886444e-02],
       [-3.46376230e-02]])
```

[13]:
```
##It does not seem to work to have double invokes here...
import scipy as scp
from scipy.optimize import minimize
#cost = lambda z: cost_functions.cost_Lpnorm_mvj(z, F= f, B= bdry, p= 20)
#grad = lambda z: cost_functions.grad_Lpnorm_mvj(z, F= f, B= bdry, p= 20)
def cost(z):
    import cost_functions
    return cost_functions.cost_Lpnorm_mvj(z, F= f, B= bdry, p= 20)

def grad(z):
    import cost_functions
    return cost_functions.grad_Lpnorm_mvj(z, F= f, B= bdry, p= 20)

#res = minimize(cost, l2_cocycle, method='L-BFGS-B', jac = grad)
res=scp.optimize.minimize(cost, l2_cocycle, method="Nelder-Mead")
res
```

[13]:
```
 final_simplex: (array([[-0.05451331,  0.28114358, -0.12064127, …,
0.05098397,
         -0.35296924,  0.04656912],
        [-0.05696441,  0.28471171, -0.12123665, …,  0.05021478,
         -0.35123827,  0.04911355],
        [-0.05586976,  0.28959666, -0.1201513 , …,  0.04395024,
         -0.3496279 ,  0.0462421 ],
        …,
        [-0.05578217,  0.3030439 , -0.12044484, …,  0.06175777,
         -0.36074508,  0.04456829],
        [-0.05392428,  0.29577467, -0.12064941, …,  0.07376492,
```

```
 -0.35477235,  0.04658347],
[-0.05531316,  0.29315442, -0.1205647 , …,  0.06823311,
 -0.35421269,  0.04844229]]), array([1.25956829, 1.25973874, 1.25979526,
1.25984778, 1.26002425,
       1.26016862, 1.26033665, 1.26035373, 1.26037607, 1.26040775,
       1.26047032, 1.26049098, 1.2605934 , 1.26059794, 1.26069819,
       1.26075562, 1.26076727, 1.26077282, 1.26093538, 1.26096887,
       1.2609698 , 1.26099686, 1.26100837, 1.2610201 , 1.26103036,
       1.26104095, 1.26106399, 1.26106715, 1.26108539, 1.26108758,
       1.26111996, 1.26116059, 1.26116662, 1.26116679, 1.26122926,
       1.26125171, 1.26126009, 1.26138357, 1.26140897, 1.26142743,
       1.26143208, 1.26143265, 1.26143433, 1.26143469, 1.26144025,
       1.26144321, 1.26146475, 1.26151626, 1.26152692, 1.26153774,
       1.26156518, 1.26158031, 1.26161764, 1.26163432, 1.26166357,
       1.26169775, 1.26172809, 1.26173529, 1.26177588, 1.2617763 ,
       1.26178843, 1.26181842, 1.26185993, 1.26187627, 1.26188992,
       1.26190712, 1.26191466, 1.26191792, 1.26194113, 1.26196473,
       1.26196515, 1.2619662 , 1.26196695, 1.26197776, 1.26199013,
       1.2620279 , 1.26203247, 1.26203365, 1.26208329, 1.26213151,
       1.26214313, 1.26218094, 1.26219364, 1.2621969 , 1.26219755,
       1.26220124, 1.26222865, 1.26223067, 1.26223605, 1.26224145,
       1.26227518, 1.26229821, 1.26235037, 1.26236873, 1.2623818 ,
       1.26239551, 1.26241953, 1.26243305, 1.26247871, 1.26248707,
       1.26249028, 1.26251119, 1.26254209, 1.26256433, 1.26259171,
       1.26263079, 1.26263765, 1.26264061, 1.26264958, 1.26266315,
       1.26267545, 1.26269331, 1.26270551, 1.2627651 , 1.26276814,
       1.26277286, 1.26281049, 1.26282568, 1.26284655, 1.26285448,
       1.26287195, 1.26287371, 1.26289175, 1.26290729, 1.2629234 ,
       1.2629274 , 1.26293125, 1.26293304, 1.26295298, 1.26296713,
       1.26298546, 1.26304422, 1.26307946, 1.26308951, 1.26309928,
       1.26315169, 1.26319948, 1.26320207, 1.2632279 , 1.26323262,
       1.26327723, 1.2633085 , 1.2633095 , 1.26332236, 1.26336195,
       1.26337316, 1.26340049, 1.26341088, 1.2634146 , 1.26343137,
       1.26343223, 1.26343641, 1.26344639, 1.26345213, 1.26347802,
       1.26348641, 1.26348684, 1.26350516, 1.26351443, 1.26352477,
       1.26352493, 1.26355632, 1.26355868, 1.26358308, 1.26362232,
       1.26366049, 1.26367223, 1.26368862, 1.26369232, 1.26369589,
       1.26373263, 1.26373604, 1.26374181, 1.26375046, 1.26376605,
       1.26378995, 1.26380019, 1.26381036, 1.26382274, 1.26382753,
       1.26386422, 1.26387224, 1.26387246, 1.26389985, 1.26390113,
       1.2639355 , 1.26395324, 1.26398531, 1.26406673, 1.26406976,
       1.26409664, 1.26410148, 1.26410676, 1.26412206, 1.26412329,
       1.26414855, 1.26417278, 1.2642054 , 1.26421945, 1.26422093,
       1.26422762]))
           fun: 1.2595682937309658
       message: 'Maximum number of function evaluations has been exceeded.'
          nfev: 40000
```

```
         nit: 39387
      status: 1
     success: False
           x: array([-5.45133120e-02,  2.81143583e-01, -1.20641269e-01,
1.32066783e-01,
       -5.24926468e-02, -2.48188521e-02, -1.63857403e-01,  1.40920426e-01,
        1.17554934e-02, -1.12864394e-02,  2.78806954e-01,  1.31464474e-04,
       -8.64968981e-02,  1.22675202e-01, -1.91937756e-02,  8.75710975e-03,
        8.80060689e-03,  4.98543238e-03, -1.37944549e-02,  1.98508080e-01,
        3.09655931e-01, -3.91594662e-02, -4.83917048e-03, -1.99463505e-03,
       -1.62441849e-02, -1.21083513e-02, -2.24772822e-04, -2.96680816e-01,
       -1.15546779e-01,  1.50693474e-02, -1.31031183e-02, -2.15524592e-01,
        1.28969121e-01,  7.88465743e-02,  4.04515059e-03,  8.07543256e-02,
        6.36145483e-02,  8.55514258e-02, -1.63712900e-01, -8.61171297e-02,
        2.68321000e-03,  7.78647340e-01, -1.79544377e-01, -1.31233936e-04,
       -8.02762275e-01,  1.60981258e-03,  2.55881242e-03,  8.18399814e-03,
       -1.80417746e-01, -4.97637470e-01,  7.40930279e-03, -2.44114648e-01,
       -1.01361674e-01,  1.02591057e-01, -3.51742551e-03,  7.02776922e-03,
        2.66108233e-03, -6.23332428e-02,  2.71324326e-01, -4.21141852e-02,
       -1.08173698e-01, -1.37024027e-03, -5.45996688e-02,  7.20319104e-01,
       -7.58701342e-02,  1.01357509e-02,  5.85964544e-02,  5.95829881e-01,
        9.48654092e-03,  6.63437237e-01,  5.36074569e-01, -1.62437077e-01,
        2.73710591e-03,  7.29320203e-02, -1.00645232e-03,  7.75637391e-02,
        3.13193902e-03,  9.68359832e-03, -1.92265196e-01,  4.40653940e-03,
       -1.28215330e-01,  1.13832278e-01,  2.54766611e-02, -6.67580185e-02,
        2.68207697e-03, -1.76666932e-01,  3.67097027e-02,  6.20554979e-01,
       -3.17989424e-01,  6.05238067e-03,  7.10083430e-01,  9.29155118e-02,
        1.81230492e-02, -3.73872629e-03,  3.45379566e-04, -7.89206733e-02,
        4.75675357e-01, -4.52737396e-02,  5.83830156e-01, -1.48261012e-03,
        4.01116416e-02, -7.84620422e-03, -2.21030318e-01,  2.89723545e-03,
        1.90226918e-01,  8.69960668e-03,  1.56836527e-02, -3.11904646e-03,
        5.55306465e-02,  4.01791277e-02,  7.03152376e-01, -1.85574678e-01,
        1.07982759e-01, -1.62207106e-01,  2.35751003e-03,  1.02865831e-01,
       -1.40515269e-01, -3.60901221e-01, -1.64037141e-01,  1.85124514e-01,
       -5.71738350e-02,  6.64370453e-03,  1.76227163e-01, -8.54809871e-02,
        3.48751566e-01,  1.05337777e-01, -1.39209222e-01, -3.47549807e-02,
       -5.01720539e-01,  1.34849827e-02,  1.34031907e-01, -4.92376028e-03,
        6.70637174e-01,  1.19287640e-02,  2.09579955e-01, -2.64210149e-01,
       -3.37364308e-01,  3.02646485e-01,  1.89037414e-02, -1.51195378e-01,
       -1.01843061e-01, -2.99442731e-01, -1.63939621e-01, -1.69784197e-01,
       -4.23980637e-01,  2.94414778e-01,  1.77179641e-01, -2.52933237e-02,
       -1.45231290e-02,  6.17100797e-02,  4.08289969e-01,  9.50129118e-02,
        1.62884010e-01,  6.96527720e-01, -1.32708705e-01, -5.00577229e-01,
        3.41912723e-03,  2.20164916e-04,  1.51485645e-03, -2.40423357e-01,
       -1.60101646e-02,  5.13620675e-03,  1.25511477e-01,  1.05104287e-01,
        4.33336445e-01, -2.80786766e-01,  4.88307021e-03,  7.12801091e-01,
        3.25356431e-01, -2.04920629e-02, -2.43416476e-02,  1.38167906e-02,
```

```
 5.36109026e-03,  5.11829922e-01, -8.36260896e-02,  2.13455107e-02,
-3.74698020e-02,  2.04416373e-02,  1.09837032e-01,  8.82086428e-02,
 7.46434580e-01, -2.97675058e-01,  1.84457051e-01,  1.66865505e-02,
-8.50934106e-02, -4.36920985e-02,  9.45159713e-02, -2.33925443e-01,
-2.22654498e-01,  1.61877186e-03, -2.87485112e-01, -3.69495245e-01,
 1.50987438e-01, -8.07331660e-02, -1.10013780e-01, -3.26917068e-01,
-6.29027151e-02,  5.09839667e-02, -3.52969239e-01,  4.65691232e-02])
```

[102]:

```
import tensorflow as tf
#import tensorflow_probability as tfp
'''Following seems deprecated in newer version of tfp
#pip install --upgrade tensorflow-probability==0.70
#alternatively, we can use tensorflow to minimize the cost function without␣
↪gradient information, here we can use multiple black-box functions like Adams
#For more: Check at https://www.tensorflow.org/probability/api_docs/python/tfp/
↪math/minimize
x = tf.Variable(0.)
cost_fun = lambda: cost_functions.cost_Lpnorm_mvj(x, F= f, B= bdry, p= 20)
res_tfp=tfp.math.minimize(
        cost_fun,
        num_steps=1000,
        optimizer=tfp.optimizers.Adam(learning_rate=0.1)
        )
'''
#Following seems working, c.f.
#https://stackoverflow.com/questions/55552715/
↪tensorflow-2-0-minimize-a-simple-function
def cost(z):
    import cost_functions
    return cost_functions.cost_Lpnorm_mvj(z, F= f, B= bdry, p= 20)
#type(bdry)
#scipy.sparse.csr.csr_matrix
B_mat = bdry.todense()
import tensorflow as tf
print(f.shape)
print((B_mat*l2_cocycle).shape)
z = tf.Variable(l2_cocycle, trainable=True)

#L1 in tensorflow language
cost_z = tf.reduce_sum( tf.abs(f - B_mat @ z) )
#L2 in tensorflow language
cost_z = tf.reduce_sum( tf.pow( tf.abs(f - B_mat @ z),2 ) )
#Lp+alpha*Lq norm in tensorflow language
lp=1
lq=4
alpha=0.8
```

```
cost_z = tf.pow( tf.reduce_sum( tf.pow( tf.abs(f - B_mat @ z),lp ) ), 1/lp) +␣
↪alpha* tf.pow( tf.reduce_sum( tf.pow( tf.abs(f - B_mat @ z),lq ) ), 1/lq)

#Gradient Descedent Optimizer
opt_gd = tf.train.GradientDescentOptimizer(0.1).minimize(cost_z)
#Adams Optimizer
opt_adams = tf.train.AdamOptimizer(1e-4).minimize(cost_z)
#The latter is much better in terms of result

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(1000):#How many iterations you want to run?
        #print(sess.run([x,loss]))
        sess.run(opt_adams)
    res_tf=sess.run([z,cost_z])
type(res_tf)
print(res_tf)
res_tf=res_tf[0]
res_tf
```

```
(8058, 1)
(8058, 1)
[array([[-0.07115968],
       [-0.07114568],
       [-0.07113611],
       [-0.07112636],
       [-0.07114245],
       [ 0.06375783],
       [-0.07115078],
       [ 0.06379926],
       [ 0.06378252],
       [ 0.06379566],
       [-0.0711468 ],
       [ 0.24585197],
       [-0.07114524],
       [ 0.1502833 ],
       [-0.07114713],
       [ 0.15029387],
       [-0.07116794],
       [-0.07116559],
       [-0.07116094],
       [-0.07112804],
       [ 0.09663156],
       [ 0.15028713],
       [-0.07115521],
       [-0.07113091],
       [ 0.06379791],
```

```
[-0.07116104],
[-0.07116707],
[-0.07115104],
[-0.07115912],
[ 0.06379298],
[-0.0711546 ],
[ 0.15029966],
[ 0.24360921],
[ 0.06374056],
[-0.07113759],
[ 0.06379808],
[ 0.15028529],
[-0.07117085],
[-0.07114127],
[ 0.06503996],
[-0.07115722],
[ 0.06380095],
[ 0.06380059],
[-0.30423522],
[-0.07113001],
[-0.07118606],
[ 0.06379416],
[ 0.06825693],
[-0.07116085],
[-0.07114706],
[ 0.06376837],
[ 0.15029268],
[ 0.06380657],
[-0.07115102],
[-0.07113026],
[ 0.06379302],
[ 0.15029337],
[-0.0711483 ],
[-0.07115229],
[-0.07114254],
[ 0.06380534],
[-0.0711506 ],
[-0.0711351 ],
[ 0.06378811],
[-0.07117208],
[ 0.06371252],
[ 0.06377282],
[ 0.06379638],
[-0.07116369],
[-0.07118046],
[-0.07116759],
[-0.07115972],
[-0.07115605],
```

```
[-0.07115171],
[ 0.06379579],
[-0.07115413],
[-0.0711613 ],
[ 0.06827538],
[ 0.06378875],
[-0.07114144],
[ 0.06380504],
[-0.07112845],
[ 0.117114  ],
[ 0.06825785],
[-0.50091244],
[-0.07115822],
[-0.07117216],
[ 0.06376634],
[ 0.06376374],
[ 0.06379307],
[ 0.15029881],
[ 0.06379185],
[ 0.06376451],
[ 0.06378328],
[-0.07117486],
[ 0.15030232],
[-0.07116215],
[-0.07113439],
[ 0.15029286],
[ 0.06375275],
[-0.07115511],
[-0.07127173],
[ 0.06380055],
[ 0.15029544],
[-0.07115914],
[-0.07115905],
[-0.07116091],
[-0.07117802],
[ 0.06379469],
[-0.07115746],
[ 0.0637872 ],
[ 0.06379708],
[-0.07114936],
[ 0.06379335],
[-0.0711475 ],
[-0.07115716],
[-0.07114699],
[ 0.06380313],
[ 0.15028193],
[ 0.15029468],
[-0.07116082],
```

```
[-0.07117351],
[-0.35357719],
[-0.07116921],
[-0.07116502],
[ 0.06379577],
[ 0.06377791],
[-0.10789312],
[ 0.063793  ],
[ 0.06576154],
[ 0.06378074],
[-0.07115502],
[ 0.06380364],
[-0.07117102],
[-0.07112503],
[ 0.06825815],
[ 0.07021414],
[-0.07110958],
[ 0.06379774],
[ 0.15029206],
[-0.07115251],
[-0.07115176],
[-0.0711665 ],
[-0.071142  ],
[ 0.06379682],
[-0.07116663],
[-0.07114235],
[-0.07116923],
[-0.07132554],
[-0.07114319],
[ 0.15027406],
[-0.07114733],
[-0.07116121],
[ 0.06380328],
[ 0.15783334],
[ 0.063805  ],
[ 0.0637919 ],
[-0.07117443],
[-0.07114663],
[ 0.06379339],
[ 0.06379579],
[ 0.06378303],
[-0.07112926],
[ 0.06496667],
[ 0.0637982 ],
[ 0.15029532],
[-0.25515962],
[-0.07113868],
[ 0.06379109],
```

```
       [-0.07115645],
       [-0.07116641],
       [-0.07115578],
       [ 0.06376832],
       [-0.07116086],
       [-0.07115393],
       [-0.07113704],
       [-0.07116506],
       [ 0.06379995],
       [ 0.06381305],
       [ 0.063746  ],
       [ 0.06378065],
       [-0.07116615],
       [-0.07115806],
       [-0.07113473],
       [-0.07115945],
       [ 0.08595695],
       [ 0.06379079],
       [-0.07114676],
       [ 0.06379002],
       [-0.07111208],
       [ 0.07807967],
       [-0.0711504 ],
       [ 0.0637938 ],
       [ 0.06379984],
       [-0.07113288],
       [-0.07117751],
       [-0.07117599],
       [ 0.06377672],
       [-0.07117685],
       [-0.07115548]]), 749.5293820986601]
```

```
[102]: array([[-0.07115968],
       [-0.07114568],
       [-0.07113611],
       [-0.07112636],
       [-0.07114245],
       [ 0.06375783],
       [-0.07115078],
       [ 0.06379926],
       [ 0.06378252],
       [ 0.06379566],
       [-0.0711468 ],
       [ 0.24585197],
       [-0.07114524],
       [ 0.1502833 ],
       [-0.07114713],
```

```
[ 0.15029387],
[-0.07116794],
[-0.07116559],
[-0.07116094],
[-0.07112804],
[ 0.09663156],
[ 0.15028713],
[-0.07115521],
[-0.07113091],
[ 0.06379791],
[-0.07116104],
[-0.07116707],
[-0.07115104],
[-0.07115912],
[ 0.06379298],
[-0.0711546 ],
[ 0.15029966],
[ 0.24360921],
[ 0.06374056],
[-0.07113759],
[ 0.06379808],
[ 0.15028529],
[-0.07117085],
[-0.07114127],
[ 0.06503996],
[-0.07115722],
[ 0.06380095],
[ 0.06380059],
[-0.30423522],
[-0.07113001],
[-0.07118606],
[ 0.06379416],
[ 0.06825693],
[-0.07116085],
[-0.07114706],
[ 0.06376837],
[ 0.15029268],
[ 0.06380657],
[-0.07115102],
[-0.07113026],
[ 0.06379302],
[ 0.15029337],
[-0.0711483 ],
[-0.07115229],
[-0.07114254],
[ 0.06380534],
[-0.0711506 ],
```

```
[-0.0711351 ],
[ 0.06378811],
[-0.07117208],
[ 0.06371252],
[ 0.06377282],
[ 0.06379638],
[-0.07116369],
[-0.07118046],
[-0.07116759],
[-0.07115972],
[-0.07115605],
[-0.07115171],
[ 0.06379579],
[-0.07115413],
[-0.0711613 ],
[ 0.06827538],
[ 0.06378875],
[-0.07114144],
[ 0.06380504],
[-0.07112845],
[ 0.117114  ],
[ 0.06825785],
[-0.50091244],
[-0.07115822],
[-0.07117216],
[ 0.06376634],
[ 0.06376374],
[ 0.06379307],
[ 0.15029881],
[ 0.06379185],
[ 0.06376451],
[ 0.06378328],
[-0.07117486],
[ 0.15030232],
[-0.07116215],
[-0.07113439],
[ 0.15029286],
[ 0.06375275],
[-0.07115511],
[-0.07127173],
[ 0.06380055],
[ 0.15029544],
[-0.07115914],
[-0.07115905],
[-0.07116091],
[-0.07117802],
[ 0.06379469],
```

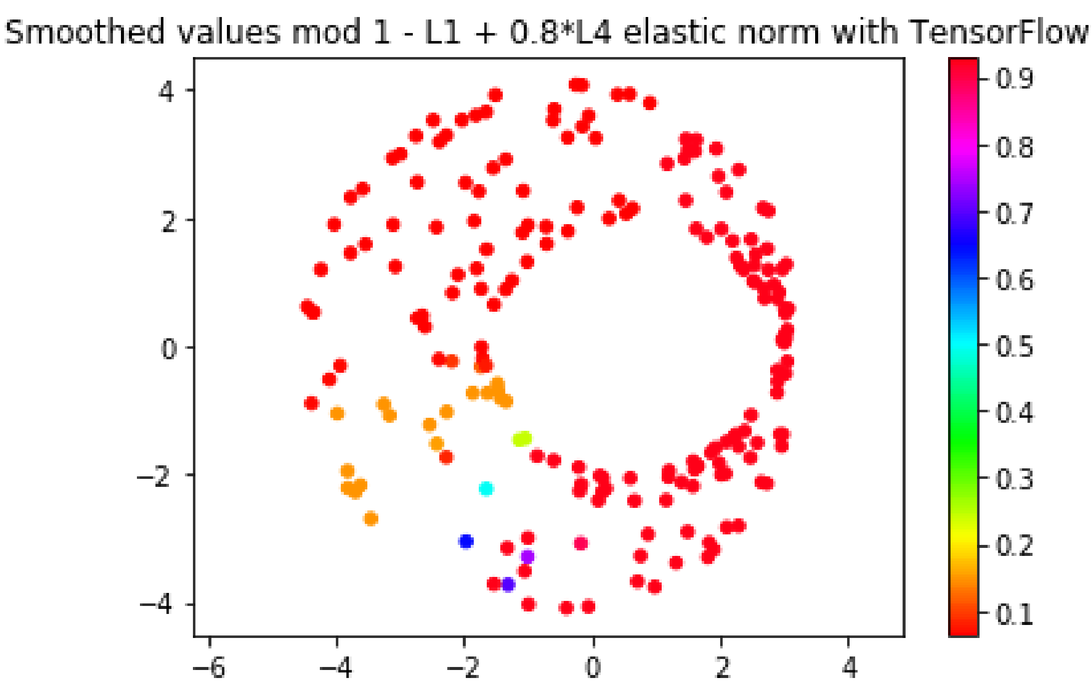
```
[-0.07115746],
[ 0.0637872 ],
[ 0.06379708],
[-0.07114936],
[ 0.06379335],
[-0.0711475 ],
[-0.07115716],
[-0.07114699],
[ 0.06380313],
[ 0.15028193],
[ 0.15029468],
[-0.07116082],
[-0.07117351],
[-0.35357719],
[-0.07116921],
[-0.07116502],
[ 0.06379577],
[ 0.06377791],
[-0.10789312],
[ 0.063793  ],
[ 0.06576154],
[ 0.06378074],
[-0.07115502],
[ 0.06380364],
[-0.07117102],
[-0.07112503],
[ 0.06825815],
[ 0.07021414],
[-0.07110958],
[ 0.06379774],
[ 0.15029206],
[-0.07115251],
[-0.07115176],
[-0.0711665 ],
[-0.071142  ],
[ 0.06379682],
[-0.07116663],
[-0.07114235],
[-0.07116923],
[-0.07132554],
[-0.07114319],
[ 0.15027406],
[-0.07114733],
[-0.07116121],
[ 0.06380328],
[ 0.15783334],
[ 0.063805  ],
```

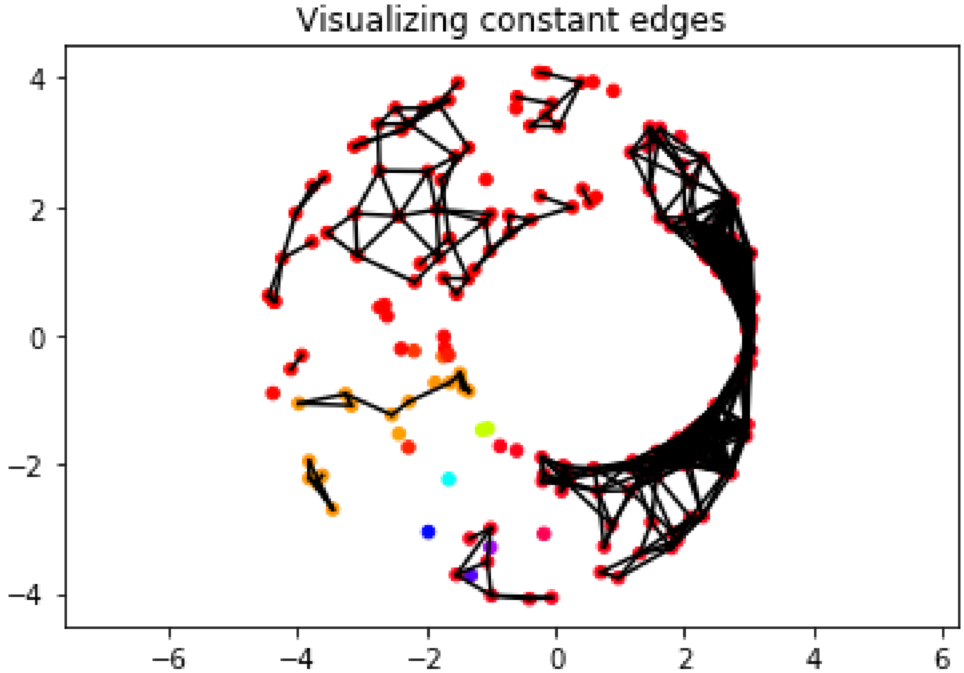
```
[ 0.0637919 ],
[-0.07117443],
[-0.07114663],
[ 0.06379339],
[ 0.06379579],
[ 0.06378303],
[-0.07112926],
[ 0.06496667],
[ 0.0637982 ],
[ 0.15029532],
[-0.25515962],
[-0.07113868],
[ 0.06379109],
[-0.07115645],
[-0.07116641],
[-0.07115578],
[ 0.06376832],
[-0.07116086],
[-0.07115393],
[-0.07113704],
[-0.07116506],
[ 0.06379995],
[ 0.06381305],
[ 0.063746  ],
[ 0.06378065],
[-0.07116615],
[-0.07115806],
[-0.07113473],
[-0.07115945],
[ 0.08595695],
[ 0.06379079],
[-0.07114676],
[ 0.06379002],
[-0.07111208],
[ 0.07807967],
[-0.0711504 ],
[ 0.0637938 ],
[ 0.06379984],
[-0.07113288],
[-0.07117751],
[-0.07117599],
[ 0.06377672],
[-0.07117685],
[-0.07115548]])
```

[103]:

```
color = np.mod(res_tf.T[0,:],1)
scatter(annulus.T[0,:],annulus.T[1,:],s=20, c=color, cmap="hsv")
```

```
#scatter(*annulus.T, c= color, cmap="hsv")
plt.colorbar()
plt.axis('equal')
plt.title('Smoothed values mod 1 - L{} + {}*L{} elastic norm with TensorFlow'.
↪format(lp,alpha,lq))
plt.show()
toll = 1e-5
edges_constant = []
thr = chosen_bar.birth # i want to check constant edges in all edges that were␣
↪there when the cycle was created
for s in vr:
    if s.dimension() != 1:
        continue
    elif s.data > thr:
        break
    if abs(color[s[0]]-color[s[1]]) <= toll:
        edges_constant.append([annulus[s[0],:],annulus[s[1],:]])
edges_constant = np.array(edges_constant)
#scatter(*annulus.T, c=color, cmap="hsv", alpha=.5)
scatter(annulus.T[0,:],annulus.T[1,:],s=20, c=color, cmap="hsv")
#plot(*edges_constant.T, c='k')
plot(edges_constant.T[0,:],edges_constant.T[1,:], c='k')
edges_constant.shape
plt.axis('equal')
plt.title('Visualizing constant edges')
plt.show()
```





Visualizing constant edges
4
2
0
−2
−4
−6
−4
−2
0
2
4
6

[ ]: