Demo-gradient-less-optimization

February 7, 2020

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

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

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

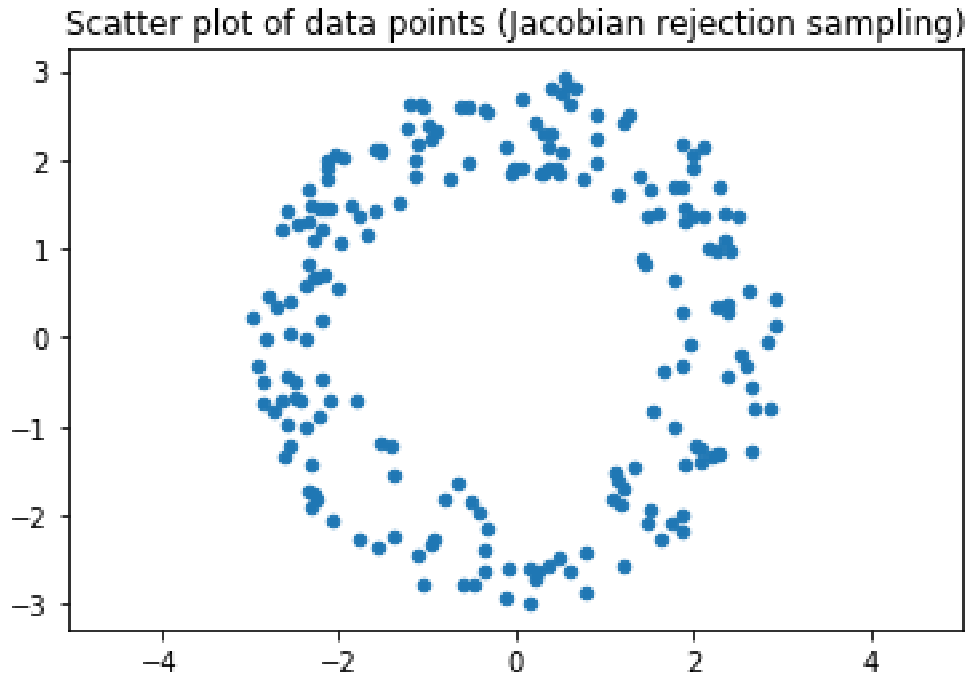
The circular coordinates pipeline for examining different smoothness cost-functions:

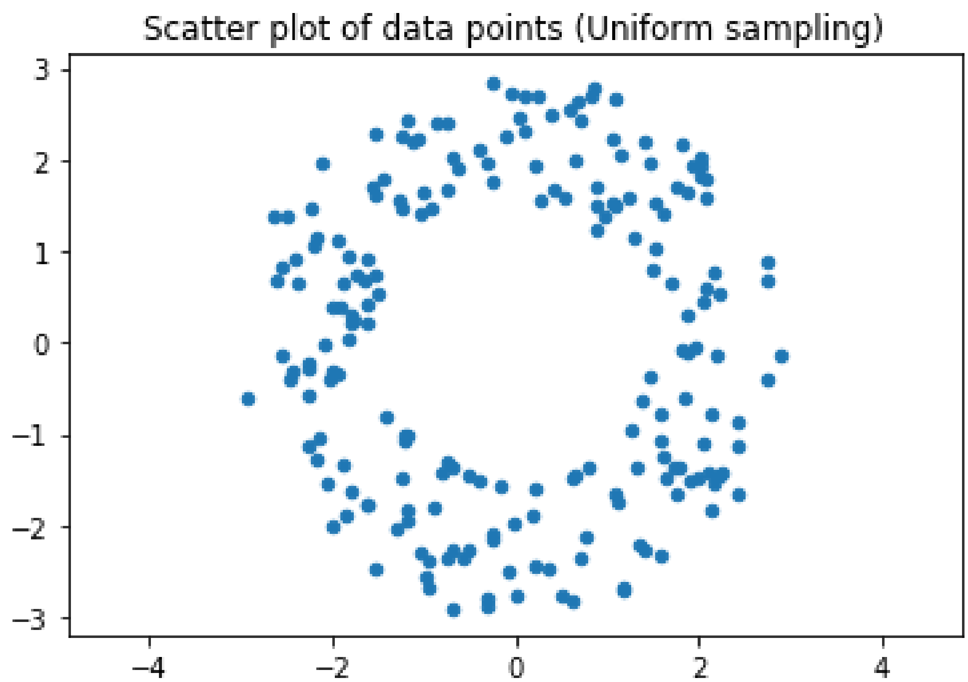
```
Step 1. Getting the point cloud
Step 2. Computing the Vietoris-Rips filtration and its cohomology
Step 3. Selecting the Cocycle
Step 4. First smoothing using Least Squares (Optional)
Step 5. Second smoothing using a new cost function
```

0.1 Step 1 - Getting the point cloud

[3]:
```
annulusJ = eg.annulus_example(d=1.5,n=200,Jacobian=True)
#The examples.py generates data points in form of point clouds that can be␣
↪analyzed using the imported dionysus module.
#plt.rcParams['lines.markersize'] = 150
#annulusJ=np.transpose(annulusJ)
scatter(annulusJ.T[0,:],annulusJ.T[1,:],s=20)
plt.axis('equal')
plt.title('Scatter plot of data points (Jacobian rejection sampling)')
plt.show()

annulus = eg.annulus_example(d=1.5,n=200,Jacobian=False)
#The examples.py generates data points in form of point clouds that can be␣
↪analyzed using the imported dionysus module.
#plt.rcParams['lines.markersize'] = 150
#annulus=np.transpose(annulus)
scatter(annulus.T[0,:],annulus.T[1,:],s=20)
plt.axis('equal')
plt.title('Scatter plot of data points (Uniform sampling)')
```

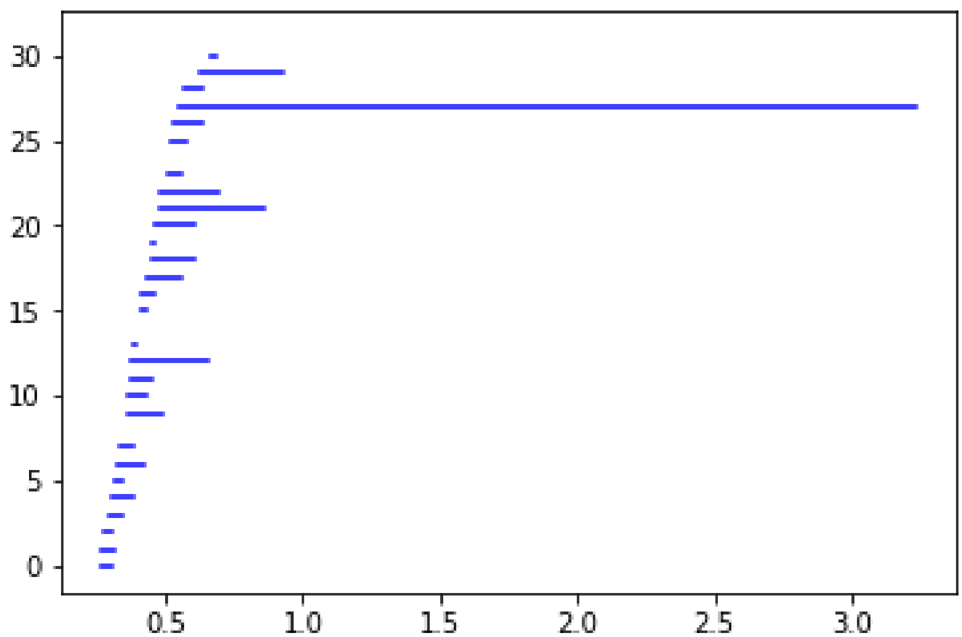
```
plt.show()

annulus = annulusJ
```

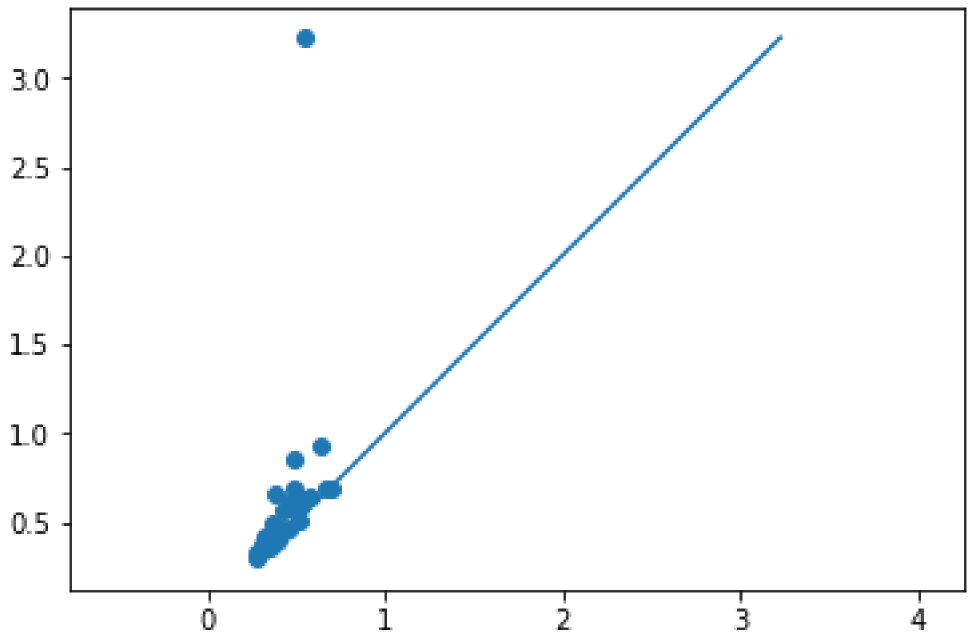




0.2 Step 2 - Computing Vietoris-Rips Complexes and Cohomology

[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.
```

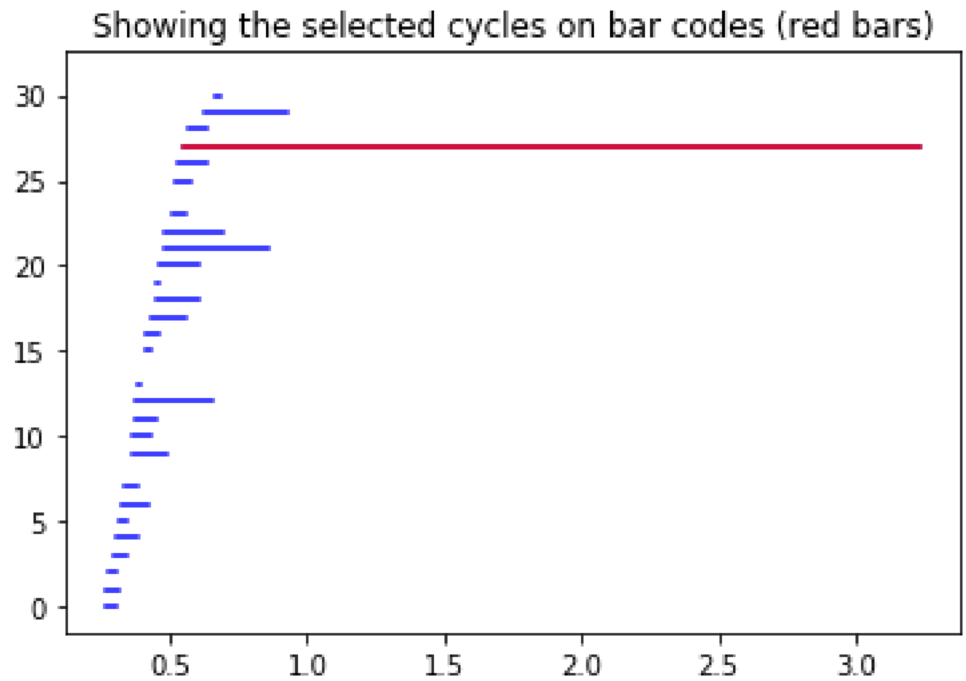


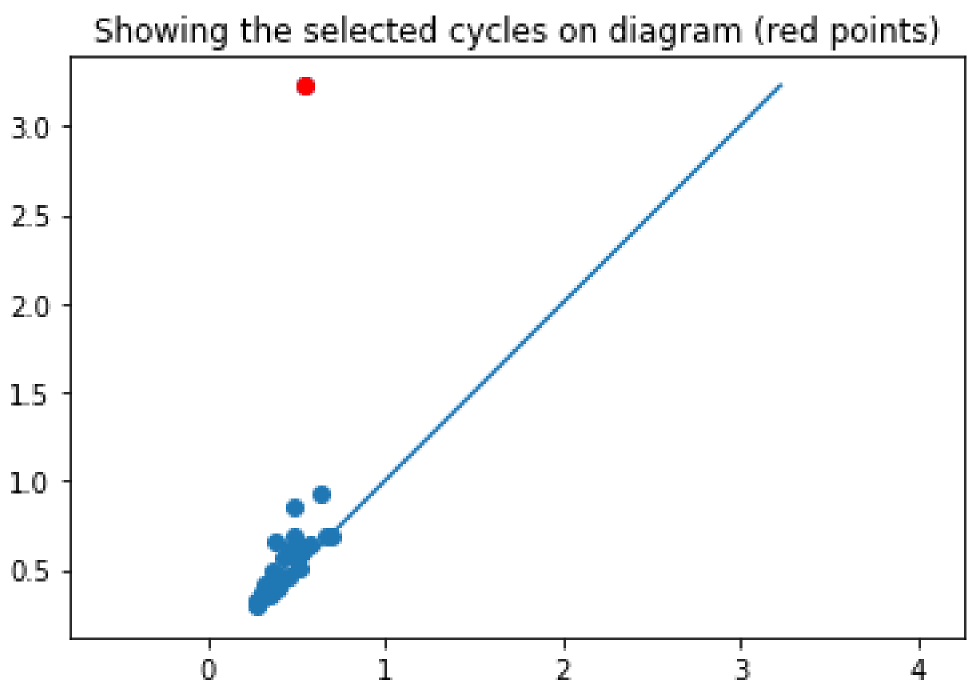


0.3 Step 3 - Selecting the cocycle and visualization.

[5]:
```
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*** cocycles 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()
```





[6]:
```
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

[7]:
```
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)
```

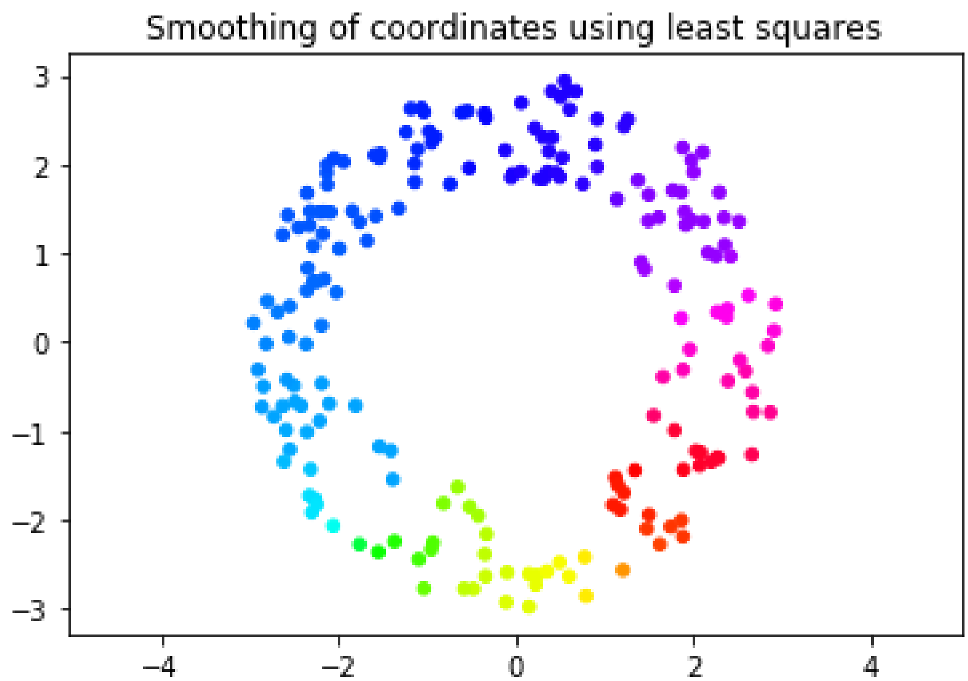
0.4.1 Visualization

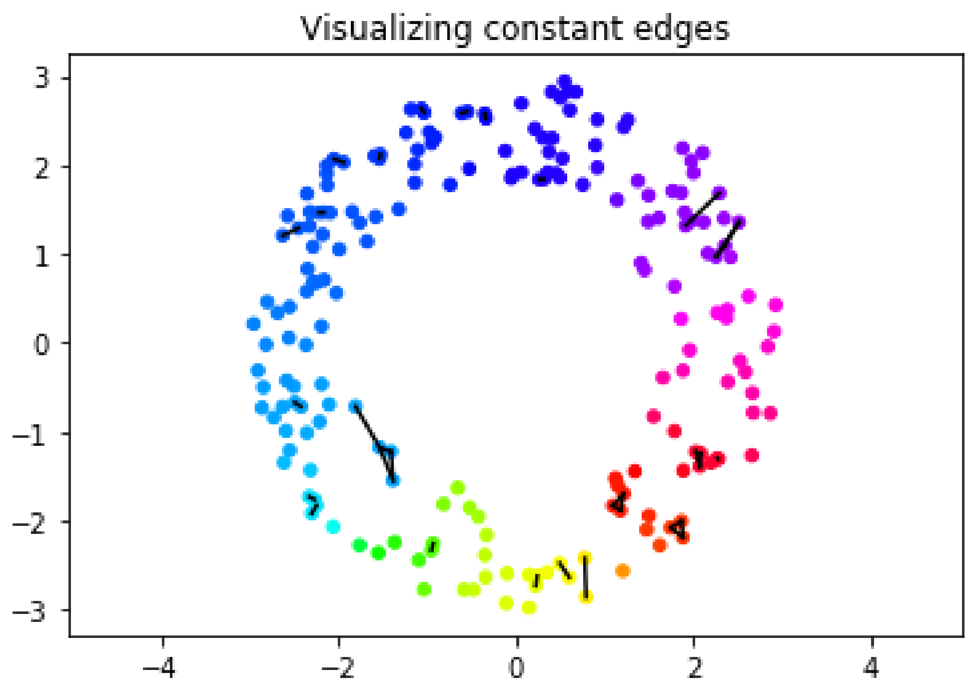
[8]:
```
np.shape(annulus.T)
#annulus.T[1,:]
np.shape(annulusJ.T)
#annulusJ.T[1,:]
```

[8]: (2, 200)

[9]:
```
#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()
```





0.5 Step 5 - Second smoothing using a new cost function

[10]:
```
from Python_code import utils
l2_cocycle,f,bdry = utils.optimizer_inputs(vr, bars, chosen_cocycle, coords,␣
↪prime)
```

[11]:
```
#l2_cocycle.reshape(-1, 1)
l2_cocycle = l2_cocycle.reshape(-1, 1)
l2_cocycle.shape
#f-bdry*l2_cocycle
```

[11]: (200, 1)

[12]:
```
##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):
    from Python_code import cost_functions
    return cost_functions.cost_Lpnorm_mvj(z, F= f, B= bdry, p= 20)

def grad(z):
    from Python_code 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
```

[15]:
```
import tensorflow as tf
#To make this cell run, you need tensorflow 1.2.0
#pip uninstall tensorflow
#pip install tensorflow==1.2.0 --ignore-installed
#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(
```

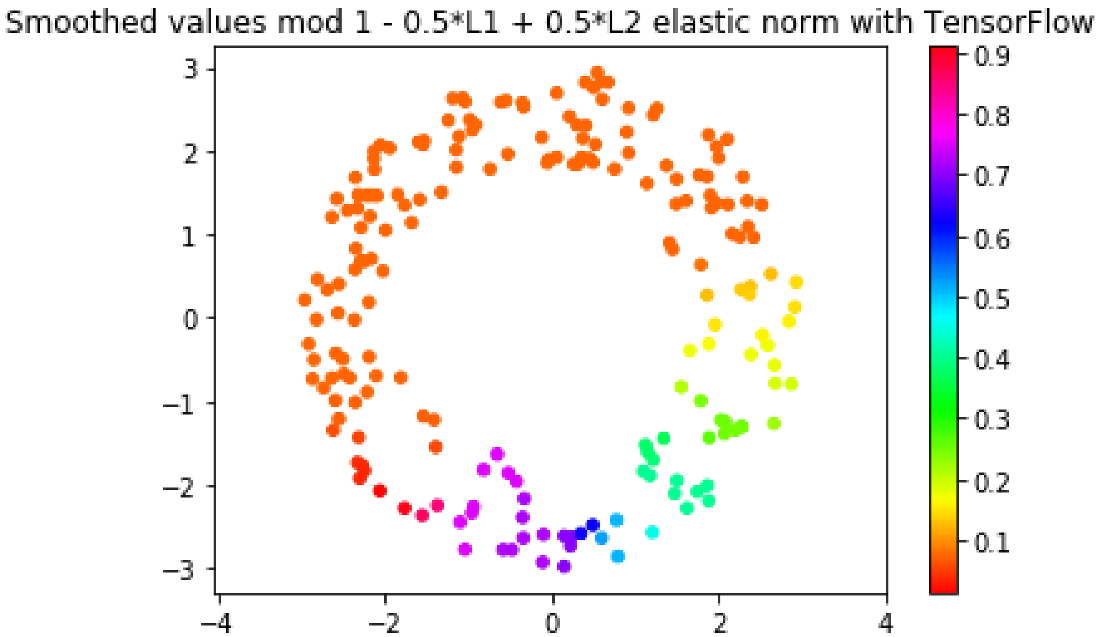
```
        cost_fun,
        num_steps=1000,
        optimizer=tfp.optimizers.Adam(learning_rate=0.1)
        )
'''
#Following seems working, c.f.
#https://stackoverflow.com/questions/55552715/
↪tensorflow-2-0-minimize-a-simple-function
def cost(z):
    import cost_functions
    return cost_functions.cost_Lpnorm_mvj(z, 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 - tf.matmul(B_mat,z) ) )
#L2 in tensorflow language
cost_z = tf.reduce_sum( tf.pow( tf.abs(f - tf.matmul(B_mat,z) ),2 ) )
#Lp+alpha*Lq norm in tensorflow language
#2020-02-07: I am not sure why @ operator is no longer a valid syntax, but I␣
↪replace them with tf.matmul.
lp=1
lq=2
alpha=.5
cost_z = (1-alpha)*tf.pow( tf.reduce_sum( tf.pow( tf.abs(f - tf.matmul(B_mat,z)␣
↪),lp ) ), 1/lp) + alpha* tf.pow( tf.reduce_sum( tf.pow( tf.abs(f - tf.
↪matmul(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)
```

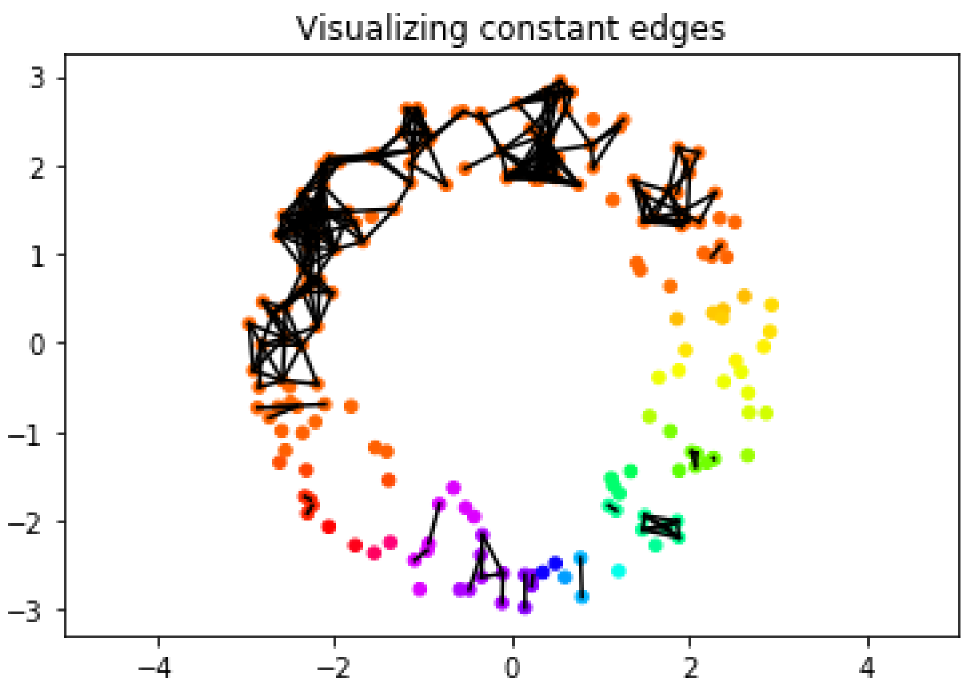
```
res_tf=res_tf[0]
#res_tf
```

```
(9135, 1)
(9135, 1)
```

[14]:

```
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(1-alpha,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()
```


Smoothed values mod 1 - 0.5*L1 + 0.5*L2 elastic norm with TensorFlow
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Visualizing constant edges
3
2
1
0
−1
−2
−3
−4
−2
0
2
4

[ ]: