The next code is divided into 2 parts – “sim functions” and “run sim”

Flow class, at the sim function section, is an object that run a simulation (X->mod->quantize->A->mod->inv(A) ) on given inputs

When generating new flow object, the output will be the simulations results

Example for generating flow object:

f=flow(init\_seed,init\_bin\_size,init\_samples,init\_modulo\_size,init\_quantizer\_offset,init\_cov,init\_A)

Flow gets the next inputs:

init\_bin\_size

the quantizer bin size  
init\_samples

number of samples to run at the simulation  
init\_modulo\_size

modulo size, edge to edge  
init\_quantizer\_offset

you can put here half of bin size so the quantizer will not have value at 0. Useful when even number of values at the quantizer  
init\_cov

data covariance matrix  
init\_A

A matrix

flow have 2 types of variables

Ones that start with init\_ which are for input variables

Ones that start with flow\_[a-g] that shows the simulation data flow

Run sim part is generating inputs for the flow class.

It has some initiating values:

#generating simulation seed to you can recover spesicif results:

init\_seed**=**np**.**random**.**randint**(**1**,**1000**)** **if** **True** **else** 473

np**.**random**.**seed**(**init\_seed**)**

#at 8G ram pc put max 1e6. you can run 1e7 but not collect more than 5 results.

init\_samples**=**1e4

#you can have here different bin size per data column:

init\_bin\_size**=[**0.05**,**0.00001**]**

init\_modulo\_size**=[**5**,**1200**]**

#running over last config and putting the same bin size and modulo for all data columns

init\_bin\_size**=[**0.15**]**

init\_modulo\_size**=[**5**]**

\_cov**=**9.99999

init\_quantizer\_offset**=[**0**]**

init\_cov**=[[**10**,**\_cov**],[**\_cov**,**10**]]**

init\_A**=[[**1**,-**1**],[**0**,**1**]]**

delta**=**pd**.**DataFrame**()**

then running simulations (generating flow class):

**for** \_cov **in** **[**5**,**8**,**8.5**,**9**,**9.5**,**9.7**,**9.9**,**9.95**,**9.9999**,**9.99999**,**9.999999**,**10**]:**#trying different cov and plotting them

init\_cov**=[[**10**,**\_cov**],[**\_cov**,**10**]]**

#running sim:

f**=**flow**(**init\_seed**,**init\_bin\_size**,**init\_samples**,**init\_modulo\_size**,**init\_quantizer\_offset**,**init\_cov**,**init\_A**)**

then gathering for each flow the outputs and plotting histogram:

delta**=**delta**.**append**(**pd**.**DataFrame**(**f**.**flow\_h\_output**[:,**0**]-**f**.**flow\_h\_output**[:,**1**],**columns**=[**f**.**init\_cov**[**1**,**0**]]))**

#if f.mse\_error!=0:

delta**.**hist**(**bins**=**1000**)**

see inline comments for more details:

**import** pandas **as** pd

**import** pylab **as** plt

**import** numpy **as** np

**import** inspect

#### # # # ###### # # # # #### ##### # #### # # ####

# # ## ## # # # ## # # # # # # # ## # #

#### # # ## # ##### # # # # # # # # # # # # # ####

# # # # # # # # # # # # # # # # # # #

# # # # # # # # # ## # # # # # # # ## # #

#### # # # # #### # # #### # # #### # # ####

**class** **flow():**

'''

this class is for running full system flow of:

X->mod->quantize->A->mod->inv(A)

inputs are covariance matrix, number of samples, A, mod and quantizer

'''

**def** \_\_init\_\_**(**self**,**init\_seed**,**init\_bin\_size**,**init\_samples**,**init\_modulo\_size**,**init\_quantizer\_offset**,**init\_cov**,**init\_A**):**

'''

this function is for taking the class arguments...

'''

**for** varname **in** inspect**.**currentframe**().**f\_code**.**co\_varnames**[**1**:]:**#[1:] to remove self

setattr**(**self**,** varname**,** locals**()[**varname**])**

self**.**init\_cov**=**np**.**mat**(**self**.**init\_cov**)**

self**.**init\_A**=**np**.**mat**(**self**.**init\_A**)**

self**.**init\_modulo\_size**=**np**.**mat**(**self**.**init\_modulo\_size**)**

self**.**run**()**

**def** generate\_data**(**self**,**cov**,**number\_of\_samples**):**

dim**=**cov**.**shape**[**1**]**

data**=**np**.**mat**(**np**.**random**.**multivariate\_normal**(**np**.**zeros**(**dim**),** cov**,** int**(**number\_of\_samples**)))**

**return** data

**def** mod**(**self**,**numbers**,**modulo\_size\_edge\_to\_edge**):**

'''

modulo\_size\_edge\_to\_edge should be array of size 1 or at numbers column size, for example:

mod(numbers,[6.0]

mod(numbers,[6.0,100])#for mod 6 for the first column and 100 for the second column

modulo\_size\_edge\_to\_edge should be numpy matrix

TODO add modulo size 0 for disabling modulo - just remove column and return it when finishing

'''

num\_resize**=(**numbers**+**modulo\_size\_edge\_to\_edge**/**2.0**)%**modulo\_size\_edge\_to\_edge

num\_resize**-=**modulo\_size\_edge\_to\_edge**/**2.0

**return** num\_resize

**def** quantize**(**self**,**numbers**,**offset**,**bin\_size**):**

'''

offset and bin\_size should be array of size 1 or at numbers column size, for example:

quantize(numbers,[0],[0.1]

quantize(numbers,[0.05,0.125],[0.1,0.25])

TODO add quantizer size 0 for disabling - just remove column and return it when finishing

'''

dim**=**numbers**.**shape**[**1**]**

quantizer**=**np**.**mat**(**np**.**identity**(**dim**)\***bin\_size**)**

**return** **((**numbers**+**offset**)\***quantizer**.**I**).**round**()\***quantizer**-**offset

**def** run**(**self**):**

#flow: X->mod->quantize->A->mod->inv(A)

self**.**flow\_a\_data**=**self**.**generate\_data**(**self**.**init\_cov**,**self**.**init\_samples**)**

self**.**flow\_b\_enc\_mod**=**self**.**mod**(**self**.**flow\_a\_data**,**self**.**init\_modulo\_size**)**

self**.**flow\_c\_enc\_quantized**=**self**.**quantize**(**self**.**flow\_b\_enc\_mod**,**self**.**init\_quantizer\_offset**,**self**.**init\_bin\_size**)**

self**.**flow\_d\_dec\_after\_a**=**self**.**flow\_c\_enc\_quantized**\***self**.**init\_A**.**T

self**.**flow\_e\_dec\_mod**=**self**.**mod**(**self**.**flow\_d\_dec\_after\_a**,**self**.**init\_modulo\_size**)**

self**.**flow\_f\_dec\_inv\_a**=**self**.**flow\_e\_dec\_mod**\***self**.**init\_A**.**I**.**T

self**.**flow\_h\_output**=**self**.**flow\_f\_dec\_inv\_a

self**.**flow\_g\_error**=(**self**.**flow\_h\_output**-**self**.**flow\_h\_output**).**round**(**2**)**

self**.**mse\_error**=**self**.**flow\_g\_error**.**flatten**()\***self**.**flow\_g\_error**.**flatten**().**T**/**self**.**flow\_g\_error**.**size

#next 2 functions are just for dumping all class element into excel

**def** \_\_iter\_\_**(**self**):**#just for dict function

**return** self**.**\_\_dict\_\_**.**iteritems**()**

**def** table**(**self**):**

data**=**pd**.**DataFrame**([**dict**(**self**)]).**transpose**()**

**return** data

##### # # # # #### # # #

# # # # ## # # # ## ##

# # # # # # # #### # # ## #

##### # # # # # # # # #

# # # # # ## # # # # #

# # #### # # #### # # #

#generating simulation seed to you can recover spesicif results:

init\_seed**=**np**.**random**.**randint**(**1**,**1000**)** **if** **True** **else** 473

np**.**random**.**seed**(**init\_seed**)**

#at 8G ram pc put max 1e6. you can run 1e7 but not collect more than 5 results.

init\_samples**=**1e4

#you can have here different bin size per data column:

init\_bin\_size**=[**0.05**,**0.00001**]**

init\_modulo\_size**=[**5**,**1200**]**

#running over last config and putting the same bin size and modulo for all data columns

init\_bin\_size**=[**0.15**]**

init\_modulo\_size**=[**5**]**

\_cov**=**9.99999

init\_quantizer\_offset**=[**0**]**

init\_cov**=[[**10**,**\_cov**],[**\_cov**,**10**]]**

init\_A**=[[**1**,-**1**],[**0**,**1**]]**

delta**=**pd**.**DataFrame**()**

**if** 1**:**

**for** \_cov **in** **[**5**,**8**,**8.5**,**9**,**9.5**,**9.7**,**9.9**,**9.95**,**9.9999**,**9.99999**,**9.999999**,**10**]:**#trying different cov and plotting them

init\_cov**=[[**10**,**\_cov**],[**\_cov**,**10**]]**

#running sim:

f**=**flow**(**init\_seed**,**init\_bin\_size**,**init\_samples**,**init\_modulo\_size**,**init\_quantizer\_offset**,**init\_cov**,**init\_A**)**

#taging only what is interesting from the last sim:

delta**=**delta**.**append**(**pd**.**DataFrame**(**f**.**flow\_h\_output**[:,**0**]-**f**.**flow\_h\_output**[:,**1**],**columns**=[**f**.**init\_cov**[**1**,**0**]]))**

#if f.mse\_error!=0:

delta**.**hist**(**bins**=**1000**)**

#plt.suptitle("var(x)=var(y)=10,y w/o quantizer and modulo, x with 0.05 qunatizer and 10 mod. hist of x-y, per cov:")

plt**.**suptitle**(**"var(x)=var(y)=10,x and y with 0.15 qunatizer and 10 mod. hist of x-y, per cov:"**)**

plt**.**show**()**