
Cutnorm Documentation

Release 0.1.9

Ping-Ko Chiu

May 07, 2018

CONTENTS

1	Introduction	3
1.1	Cutnorm	3
2	cutnorm	5
2.1	cutnorm package	5
3	Indices and tables	13
	Bibliography	15
	Python Module Index	17
	Index	19

Welcome to the Cutnorm package documentation. Please read the introduction and checkout the documentation.

INTRODUCTION

1.1 Cutnorm

1.1.1 Approximation via Gaussian Rounding and Optimization with Orthogonality Constraints

This package computes the approximations to the cutnorm of matrices using some of the techniques detailed by Alon and Noar [ALON2004] and a fast optimization algorithm by Wen and Yin [WEN2013].

Read the [documentation](#).

1.1.2 Installation

Use `pip` to install the package. Install from terminal as follows:

```
$ pip install cutnorm
```

1.1.3 Example Usage

Given the adjacency matrices of two simple graphs A and B, we wish to compute a norm for the difference matrix (A - B) between the two graphs. An obvious display of the advantages of using a cutnorm over l1 norm is to consider the value of the norms on [Erdos-Renyi random graphs](#).

Given two Erdos-Renyi random graphs with constant n and p=0.5, the edit distance (l1 norm) of the difference (after normalization) is 0.5 with large probability. An l1 norm of 1 implies the two matrices are completely different, 0 implies identity, and 0.5 is somewhere in between. However, these two graphs have the same global structure. As n approaches infinity, A and B converges to the same graphon object that is 0.5 everywhere. The edit distance fails as a notion of ‘distance’ between the two graphs in the perspective of global structural similarity as discussed by Lovasz [LOVASZ2009]. The cutnorm is a measure of distance that reflects global structural similarity. In fact, the cutnorm of the difference for this example approaches 0 as n grows.

Below is an example of using the cutnorm package and tools.

```
import numpy as np
from cutnorm import compute_cutnorm, tools

# Generate Erdos Renyi Random Graph (Simple/Undirected)
n = 100
p = 0.5
erdos_renyi_a = tools.sbm.erdos_renyi(n, p, symmetric=True)
erdos_renyi_b = tools.sbm.erdos_renyi(n, p, symmetric=True)
```

```
# Compute l1 norm
normalized_diff = (erdos_renyi_a - erdos_renyi_b) / n**2
l1 = np.linalg.norm(normalized_diff.flatten(), ord=1)

# Compute cutnorm
cutn_round, cutn_sdp, info = compute_cutnorm(erdos_renyi_a, erdos_renyi_b)

print("l1 norm: ", l1) # prints l1 norm value near ~0.5
print("cutnorm rounded: ",
      cutn_round) # prints cutnorm rounded solution near ~0
print("cutnorm sdp: ", cutn_sdp) # prints cutnorm sdp solution near ~0
```

CUTNORM

2.1 cutnorm package

2.1.1 Subpackages

cutnorm.tools package

Submodules

cutnorm.tools.sbm module

This package generates stochastic matrices.

Currently supported models are general Stochastic Block Models, Erdos Renyi, and Autoregressive Models.

`cutnorm.tools.sbm.erdos_renyi` (*n*, *p*, *symmetric=True*)

Generates Erdos Renyi random graph size *n* with probability *p*

Parameters

- **n** – int, size of the output matrix
- **p** – float, edge probability
- **symmetric** – boolean, if true, the function will output a symmetric matrix

Returns Erdos Renyi random graph matrix 2d array, shape (n,n)

`cutnorm.tools.sbm.make_symmetric_triu` (*mat*)

Makes the matrix symmetric upper triangular

Parameters **mat** – 2d array, shape (n,n)

Returns upper triangular symmetric matrix of the input 2d array, shape (n,n)

`cutnorm.tools.sbm.sbm` (*community_sizes*, *prob_mat*, *symmetric=True*)

Generates a stochastic block matrix

Community_sizes indicate the size of each community and the probability matrix indicate the probability that a 1 will be generated for each element within the community.

Parameters

- **community_sizes** – 1d array, shape (n) sizes of community
- **prob_mat** – 2d array, shape (n,n) probability of edges for each community
- **symmetric** – boolean, if true, the function will output a symmetric matrix

Returns stochastic block matrix, 2d array, shape depending on community sizes

`cutnorm.tools.sbm.sbm_autoregressive (community_sizes, prob_list, symmetric=True)`

Generates an autoregressive SBM

An autoregressive SBM has edge probability according to the `prob_list` on the diagonal but $(\text{prob_list}[i] * \text{prob_list}[j])^{(|i-j|)}$ for the off-diagonal blocks entries.

This idea is similar to the autoregressive models

Parameters

- **community_sizes** – 1d array, shape (n) sizes of community
- **prob_list** – 1d array, shape (n), where n is the number of diagonal blocks
- **symmetric** – boolean, if true, the function will output a symmetric matrix

Returns An autoregressive SBM, 2d array, shape depending on community sizes

`cutnorm.tools.sbm.sbm_autoregressive_prob (community_sizes, prob_list)`

Generates the underlying probability matrix that gives rise to the autoregressive SBM

Parameters

- **community_sizes** – 1d array, shape (n) sizes of community
- **prob_list** – 1d array, shape (n), where n is the number of diagonal blocks

Returns A probability matrix for an autoregressive SBM, 2d array, shape depending on community sizes

`cutnorm.tools.sbm.sbm_prob (community_sizes, prob_mat)`

Generates a matrix indicating the underlying probability that gives rise to a stochastic block matrix

Parameters

- **community_sizes** – 1d array, shape (n) sizes of community
- **prob_mat** – 2d array, shape (n,n) probability of edges for each community

Returns probabilities of a stochastic block matrix, 2d array, shape depending on community sizes

cutnorm.tools.distort module

This package adds distortion to matrices.

`cutnorm.tools.distort.add_gaussian_noise (mat, mean, std)`

Adds gaussian noise to the matrix

Parameters

- **mat** – 2d array, shape (n,n)
- **mean** – gaussian mean
- **std** – gaussian std

Returns Processed matrix

`cutnorm.tools.distort.shift (mat, n_shift)`

Shifts the matrix by rolling it along the diagonal

Parameters

- **mat** – 2d array, shape (n,n)

- **n_shift** – number to roll

Returns Shifted matrix

cutnorm.tools.dbf_testing module

Implementation of DBF statistic

from “Distance-based analysis of variance: approximate inference and an application to genome-wide association studies”, Christopher Minas and Giovanni Montana

`cutnorm.tools.dbf_testing.dbf_pvalue(mc, vc, gc, fstat, trW, trB)`

pval = pearson_three(mc, vc, gc, fstat) compute one sided p value from standardized pearson three distribution

Parameters

- **mc** – mean
- **vc** – variance
- **gc** – skewness
- **fstat** – DBF distance based F statistic, between vs. within group variability
- **trW** – within group variability for DBF statistic
- **trB** – between group variability for DBF statistic

Returns pval = one sided p value

`cutnorm.tools.dbf_testing.dbf_test(dmatrix, labels)`

pval, fstat, Bvar, Wvar = dbf_test(dmatrix, labels) run dbf test

Parameters

- **dmatrix** – N by N array of distances
- **Labels** – N array group membership

Returns one sided p value fstat: DBF distance based F statistic, between vs. within group variability
Bvar: between vs overall group variability Wvar: within vs overall group variability

Return type pval

`cutnorm.tools.dbf_testing.distance_variability(dmatrix, Ic)`

fstat, trW, trB = distance_variability(dmatrix, Ic)

compute (i) within group variability, (ii) between group variability (iii) total variability (iv) distance based F statistic (DBF)

Parameters

- **dmatrix** – N by N array of distances
- **Ic** – N by G array group membership

Returns DBF distance based F statistic, between vs. within group variability trW: Within group variability trB: Between group variability

Return type fstat

`cutnorm.tools.dbf_testing.distribution_parameters(dmatrix, Ic)`

mc, vc, gc = distribution_parameters(dmatrix, Ic) compute (pearson III) null distribution parameters

Parameters

- **dmatrix** – N by N array of distances
- **Ic** – N by G array group membership

Returns mean vc: variance gc: skewness

Return type mc

`cutnorm.tools.dbf_testing.inv_fn`(*mc, vc, gc, fstat, trT*)
inverse fn

Parameters

- **mc** – mean
- **vc** – variance
- **gc** – skewness
- **fstat** – DBF distance based F statistic, between vs. within group variability
- **trT** – total group variability for DBF statistic

Returns inv fn val

cutnorm.tools.lpdist module

This package computes the lp distances given two matrices.

Matrices of different dimension and weights will be rescaled and reweighted using the same method in Cutnorm package.

`cutnorm.tools.lpdist.compute_lp_distance`(*p, A, B, w1=None, w2=None*)

Module contents

2.1.2 Submodules

2.1.3 cutnorm.OptManiMulitBallGBB module

This package contains the algorithm in ‘A feasible method for optimization with orthogonality constraints’ by Zaiwen Wen and Wotao Yin.

We have reinterpreted the algorithm from Matlab by Wen & Yin to Python.

`cutnorm.OptManiMulitBallGBB.cutnorm_quad`(*V, C*)
Cutnorm function to compute objective function value and gradient

Parameters

- **V** – ndarray, Low rank model $X = V^* * V$;
- **C** – ndarray, Objective matrix to compute maxcut

Returns

(f, g)

f: float, objective function value

g: ndarray, gradient

`cutnorm.OptManiMulitBallGBB.maxcut_quad(V, C)`

Maxcut function to compute objective function value and gradient

maxcut SDP: X is n by n matrix $\max \text{Tr}(C^*X)$, s.t., $X_{ii} = 1$, X psd

Parameters

- **V** – ndarray, Low rank model $X = V^* * V$;
- **C** – ndarray, Objective matrix to compute maxcut

Returns

(f, g)

f: float, objective function value

g: ndarray, gradient

`cutnorm.OptManiMulitBallGBB.opt_man_i_mulit_ball_gbb(x, fun, args, xtol=1e-06, ftol=1e-12, gtol=1e-06, rho=0.0001, eta=0.1, gamma=0.85, tau=0.001, nt=5, mxitr=1000, record=0)`

Line search algorithm for optimization on manifold Reinterpreted directly from Zaiwen Wen and Wotao Yin's Matlab implementation of their paper on 'A feasible method for optimization with orthogonality constraints'

Parameters

- **x** – Numpy array where each column lies on the unit sphere $\|x_{:i}\|_2 = 1$
- **fun** – Function that returns the objective function value and its gradient. Params: [x, args]
Returns: [f, g]
- **args** – args to be used in fun
- **kwargs** – Options `record = 0`, no print out `mxitr` max number of iterations `xtol` stop control for $\|X_k - X_{k-1}\|$ `gtol` stop control for the projected gradient `ftol` stop control for $\frac{\|F_k - F_{k-1}\|}{(1 + \|F_{k-1}\|)}$ usually, $\max\{xtol, gtol\} > ftol$

Returns

(x, g, out)

x: solution

g: gradient of x

Out: output information

2.1.4 cutnorm.compute module

This package computes the Cutnorm.

`cutnorm.compute.compute_cutnorm(A, B, w1=None, w2=None, max_round_iter=100, logn_lowrank=False, extra_info=False)`

Computes the cutnorm of the differences between the two matrices

Parameters

- **A** – ndarray, (n, n) matrix
- **B** – ndarray, (m, m) matrix
- **w1** – ndarray, (n, 1) array of weights for A

- **w2** – ndarray, (m, 1) array of weights for B
- **max_round_iter** – int, number of iterations for gaussian rounding
- **logn_lowrank** – boolean to toggle log2(n) low rank approximation
- **extra_info** – boolean, generate extra computational information

Returns

(cutnorm_round, cutnorm_sdp, info)

cutnorm_round: objective function value from gaussian rounding

cutnorm_sdp: objective function value from sdp solution

info: dictionary containing computational information

Computational information from OptManiMulitBallGBB: sdp_augm_n: dimension of augmented matrix sdp_relax_rank_p: rank sdp_solve: computation time sdp_itr, sdp_nfe, sdp_feasi, sdp_fval, sdp_g, sdp_nrmG: information from OptManiMulitBall-GBB

Computational information from gaussian rounding: round_solve: computation time for rounding round_approx_list: list of rounded objf values round_uis_list: list of uis round_vjs_list: list of vjs round_uis_opt: optimum uis round_vjs_opt: optimum vjs

Computational information from processing the difference: weight_of_C: weight vector of C, the difference matrix

Cutnorm information: cutnorm_sets (S,T): vectors of cutnorm

Raises `ValueError` – if A and B are of wrong dimension, or if weight vectors does not match the corresponding A and B matrices

`cutnorm.compute.cutnorm_sets(uis, vjs)`

Generates the cutnorm sets from the rounded SDP solutions

Parameters

- **uis** – ndarray, (n+1,) shaped array of rounded +- 1 solution
- **vjs** – ndarray, (n+1,) shaped array of rounded +- 1 solution

Returns

(S, T) Reconstructed S and T sets that are $\{1, 0\}^n$

S: Cutnorm set axis = 0

T: Cutnorm set axis = 1

`cutnorm.compute.gaussian_round(U, V, C, max_round_iter, logn_lowrank=False, extra_info=False)`

Gaussian Rounding for Cutnorm

The algorithm picks a random standard multivariate gaussian vector w in R^p and computes the rounded solution based on $\text{sgn}(w \cdot u_i)$.

Adopted from David Koslicki's cutnorm rounding code <https://github.com/dkoslicki/CutNorm> and Peter Diao's modifications

Parameters

- **U** – ndarray, (p, n) shaped matrices of relaxed solutions
- **V** – ndarray, (p, n) shaped matrices of relaxed solutions

- **C** – ndarray, original (n, n) shaped matrix to compute cutnorm
- **max_round_iter** – maximum number of rounding operations
- **logn_lowrank** – boolean to toggle $\log_2(n)$ low rank approximation
- **extra_info** – boolean, generate extra computational information

Returns

(approx_opt, uis_opt, vjs_opt, round_info)

approx_opt: approximated objective function value

uis_opt: rounded u vector

vjs_opt: rounded v vector

round_info: information for rounding operation

2.1.5 Module contents

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

BIBLIOGRAPHY

- [ALON2004] Noga Alon and Assaf Naor. 2004. Approximating the cut-norm via Grothendieck's inequality. In Proceedings of the thirty-sixth annual ACM symposium on Theory of computing (STOC '04). ACM, New York, NY, USA, 72-80. DOI: <http://dx.doi.org/10.1145/1007352.1007371>
- [WEN2013] Zaiwen Wen and Wotao Yin. 2013. A feasible method for optimization with orthogonality constraints. Math. Program. 142, 1-2 (December 2013), 397-434. DOI: <https://doi.org/10.1007/s10107-012-0584-1>
- [LOVASZ2009] Lovasz, L. 2009. Very large graphs. ArXiv:0902.0132 [Math]. Retrieved from <http://arxiv.org/abs/0902.0132>

PYTHON MODULE INDEX

C

- `cutnorm`, 11
- `cutnorm.compute`, 9
- `cutnorm.OptManiMulitBallGBB`, 8
- `cutnorm.tools`, 8
 - `cutnorm.tools.dbf_testing`, 7
 - `cutnorm.tools.distort`, 6
 - `cutnorm.tools.lpdist`, 8
 - `cutnorm.tools.sbm`, 5

A

`add_gaussian_noise()` (in module `cutnorm.tools.distort`), 6

C

`compute_cutnorm()` (in module `cutnorm.compute`), 9

`compute_lp_distance()` (in module `cutnorm.tools.lpdist`), 8

`cutnorm` (module), 11

`cutnorm.compute` (module), 9

`cutnorm.OptManiMulitBallGBB` (module), 8

`cutnorm.tools` (module), 8

`cutnorm.tools.dbf_testing` (module), 7

`cutnorm.tools.distort` (module), 6

`cutnorm.tools.lpdist` (module), 8

`cutnorm.tools.sbm` (module), 5

`cutnorm_quad()` (in module `cutnorm.OptManiMulitBallGBB`), 8

`cutnorm_sets()` (in module `cutnorm.compute`), 10

D

`dbf_pvalue()` (in module `cutnorm.tools.dbf_testing`), 7

`dbf_test()` (in module `cutnorm.tools.dbf_testing`), 7

`distance_variability()` (in module `cutnorm.tools.dbf_testing`), 7

`distribution_parameters()` (in module `cutnorm.tools.dbf_testing`), 7

E

`erdos_renyi()` (in module `cutnorm.tools.sbm`), 5

G

`gaussian_round()` (in module `cutnorm.compute`), 10

I

`inv_f_fn()` (in module `cutnorm.tools.dbf_testing`), 8

M

`make_symmetric_triu()` (in module `cutnorm.tools.sbm`), 5

`maxcut_quad()` (in module `cutnorm.OptManiMulitBallGBB`), 8

O

`opt_man_i_mulit_ball_gbb()` (in module `cutnorm.OptManiMulitBallGBB`), 9

S

`sbm()` (in module `cutnorm.tools.sbm`), 5

`sbm_autoregressive()` (in module `cutnorm.tools.sbm`), 6

`sbm_autoregressive_prob()` (in module `cutnorm.tools.sbm`), 6

`sbm_prob()` (in module `cutnorm.tools.sbm`), 6

`shift()` (in module `cutnorm.tools.distort`), 6