

---

# Cutnorm Documentation

*Release 0.1.5*

**Ping-Ko Chiu**

**Jan 24, 2018**



# CONTENTS

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Cutnorm . . . . .	3
<b>2</b>	<b>cutnorm</b>	<b>5</b>
2.1	cutnorm package . . . . .	5
<b>3</b>	<b>Indices and tables</b>	<b>11</b>
	<b>Bibliography</b>	<b>13</b>
	<b>Python Module Index</b>	<b>15</b>
	<b>Index</b>	<b>17</b>



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 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 Example Usage

Below is an example of using the cutnorm package and tools. Given two graphs A and B, we wish to compute a norm for the difference matrix (A - B) between the two graphs. An obvious example to represent the advantage of using a cutnorm over l1 norm is to consider A and B as [Erdos-Renyi random graphs](#). Under a fixed vertex set, an Erdos-Renyi random graph is one where a fixed probability determines the presence of an edge.

Given two Erdos-Renyi random graphs with fix n and p=0.5, the l1 norm of the difference (after normalization) has an expectation of 0.5. However, we know that these two graphs should not have any correlation between each other. A reasonable norm of the difference should have an expectation of 0. In fact, the cutnorm of the difference approaches 0 as n grows.

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

# Generate Erdos Renyi Random Graph
n = 100
p = 0.5
erdos_renyi_a = tools.sbm.erdos_renyi(n, p)
erdos_renyi_b = tools.sbm.erdos_renyi(n, p)

# 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 = 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
```





## 2.1 cutnorm package

### 2.1.1 Subpackages

cutnorm.tools package

Submodules

cutnorm.tools.sbm module

cutnorm.tools.sbm.**erdos\_renyi** (*n*, *p*)

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

**Parameters**

- **n** – int, size of the output matrix
- **p** – float, edge probability

**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*)

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

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

cutnorm.tools.sbm.**sbm\_autoregressive** (*community\_sizes*, *prob\_list*)

Generates an autoregressive SBM

An autoregressive SBM has edge probability according to the `prob_list` on the diagonal but  $(\text{prob\_list}[i] * \text{prob\_list}[j])^{*(\text{abs}(i - j))}$  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

**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

## Module contents

### 2.1.2 Submodules

#### 2.1.3 cutnorm.OptManiMulitBallGBB module

`cutnorm.OptManiMulitBallGBB.cutnorm_quad (V: numpy.ndarray, C: numpy.ndarray) -> (<class 'numpy.float64'>, <class 'numpy.ndarray'>)`

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: numpy.ndarray, C: numpy.ndarray) -> (<class 'numpy.float64'>, <class 'numpy.ndarray'>)`

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_mani_mulit_ball_gbb(x: numpy.ndarray, 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  $\|x\|_2 = 1$
- **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

```
cutnorm.compute.compute_cutnorm(A: numpy.ndarray, B: numpy.ndarray, w1=None, w2=None,
                                   max_round_iter=100, logn_lowrank=False, extra_info=False)
-> (<class 'numpy.float64'>, <class 'numpy.float64'>, <class 'dict'>)
```

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  $\log_2(n)$  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

S: Cutnorm set axis = 0

T: Cutnorm set axis = 1

**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\_nrmG: information from OptManiMulitBallGBB

**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

**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: numpy.ndarray, vjs: numpy.ndarray) -> (<class 'numpy.ndarray'>, <class 'numpy.ndarray'>)
```

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: numpy.ndarray, V: numpy.ndarray, C: numpy.ndarray, max_round_iter: int, logn_lowrank=False, extra_info=False) -> (<class 'numpy.float64'>, <class 'numpy.ndarray'>, <class 'numpy.ndarray'>, <class 'dict'>)
```

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

vis\_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>



## PYTHON MODULE INDEX

### C

`cutnorm`, 9  
`cutnorm.compute`, 7  
`cutnorm.OptManiMulitBallGBB`, 6  
`cutnorm.tools`, 6  
`cutnorm.tools.sbm`, 5



## C

compute\_cutnorm() (in module cutnorm.compute), 7  
 cutnorm (module), 9  
 cutnorm.compute (module), 7  
 cutnorm.OptManiMulitBallGBB (module), 6  
 cutnorm.tools (module), 6  
 cutnorm.tools.sbm (module), 5  
 cutnorm\_quad() (in module cutnorm.OptManiMulitBallGBB), 6  
 cutnorm\_sets() (in module cutnorm.compute), 8

## E

erdos\_renyi() (in module cutnorm.tools.sbm), 5

## G

gaussian\_round() (in module cutnorm.compute), 8

## M

make\_symmetric\_triu() (in module cutnorm.tools.sbm), 5  
 maxcut\_quad() (in module cutnorm.OptManiMulitBallGBB), 6

## O

opt\_man\_i\_mulit\_ball\_gbb() (in module cutnorm.OptManiMulitBallGBB), 7

## S

sbm() (in module cutnorm.tools.sbm), 5  
 sbm\_autoregressive() (in module cutnorm.tools.sbm), 5  
 sbm\_autoregressive\_prob() (in module cutnorm.tools.sbm), 6  
 sbm\_prob() (in module cutnorm.tools.sbm), 6