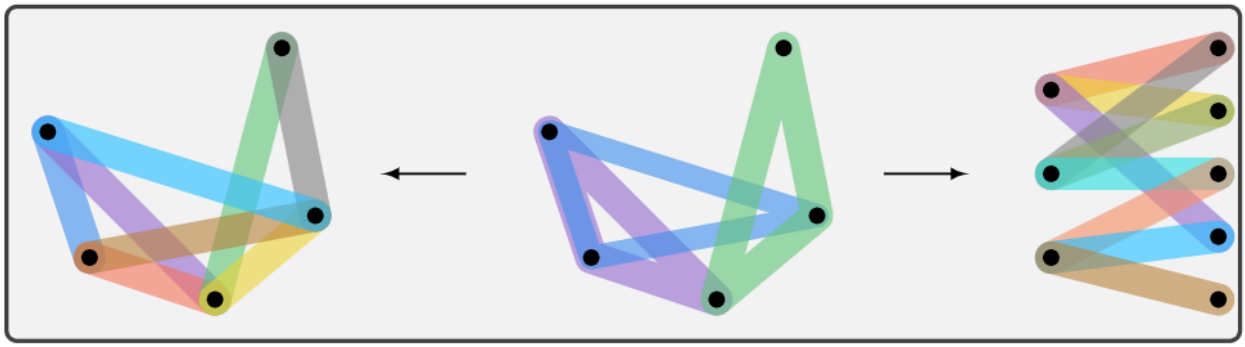
# Hypergraph Analysis Toolbox

*Release 0.0.1*

**Joshua Pickard**

**Sep 24, 2023**

# CONTENTS:



CHAPTER

ONE

# INTRODUCTION

Hypergraph Analysis Toolbox (HAT) is a software suite for the analysis and visualization of hypergraphs and higher order structures. Motivated to investigate Pore-C data, HAT is intended as a general prupose, versatile software for hypergraph construction, visualization, and analysis. HAT addresses the following hypergraph problems:

1. Construction
2. Visualization
3. Expansion and numeric representation
4. Structral Properties
5. Controllability
6. Similarity Measures

The capabilities and use cases of HAT are outlined in this notice.

## 1.1 Contributors

Joshua Pickard, Can Chen, Rahmy Salman, Cooper Stansbury, Sion Kim, Amit Surana, Anthony Bloch, and Indika Rajapakse

### 1.1.1 Bug Reporting

Please report all bugs or defects in HAT to this page.

#### Installation

An instillation guide for the MATLAB, Python, and Development versions of HAT is available here.

### Python Distribution

The Python distribution of HAT may be installed through pip:

```
>> pip install HypergraphAnalysisToolbox
```

Once installed, HAT may be imported into the Python environment with the command:

```
import HAT
```

The Python distribution has the following dependencies:

1. numpy
2. scipy
3. matplotlib
4. itertools
5. networkx

### MATLAB Distribution

The MATLAB distribution of HAT can be installed through either the MATLAB Central. A MathWorks `.mltbx` file can be downloaded from the site, and installed through the add on manager in the MATLAB Home environment. Once installed as a toolbox, you will have access to all HAT functionality.

The MATLAB distribution has the following dependencies which need to be installed separately:

1. TenEig —Tensor Eigenpairs Solver

### Development Distribution

All implementations of HAT are managed through a common git repository. This is public, so it may be cloned and modified. If interested in modifying or contributing to HAT, please see information on the Development page and contact Joshua Pickard at jpic@umich.edu.

### Bug Reporting

Please report all bugs or defects in HAT to this page.

### Tutorials

This page contains a series of tutorials for using HAT. Every tutorial is available in both Python and MATLAB.

### Python

Each Python tutorial open as a Google CoLab notebook and can be run online. You need to be logged into a google account in order to access the notebooks. The links below may open to seemingly large .txt documents, in which case there will be a button near the top to open the links in CoLab. Additionally, the tutorials can be downloaded from the links and run locally as a jupyter notebook as well.

1. Introduction to HAT in Python
2. Constructing Hypergraphs from Multicorrealtions

### MATLAB

Each MATLAB tutorial opens to MATLAB Online. If you have a MATLAB account you can run it online. Otherwise, you can download the tutorial file from MATLAB Online and run it as a live script locally.

1. Introduction to HAT in MATLAB
2. Constructing Hypergraphs from Multicorrealtions

### Bug Reporting

Please report all bugs or defects in HAT to this page.

### HAT Documentation

### Submodules

### HAT.Hypergraph module

**class** HAT.Hypergraph.**Hypergraph**(*im*, *ew=None*, *nw=None*)

Bases: `object`

This is the base class representing a Hypergraph object. It is the primary entry point and provides an interface to functions implemented in HAT's other modules. The underlying data structure of this class is an incidence matrix, but many methods exploit tensor representation of uniform hypergraphs.

Formally, a Hypergraph $H = (V, E)$ is a set of vertices $V$ and a set of edges $E$ where each edge $e \in E$ is defined $e \subseteq V$. In contrast to a graph, a hypergraph edge $e$ can contain any number of vertices, which allows for efficient representation of multi-way relationships.

In a uniform Hypergraph, all edges contain the same number of vertices. Uniform hypergraphs are represnted as tensors, which precisely model multi-way interactions.

**Parameters**

- **im** – Incidence matrix
- **ew** – Edge weight vector
- **nw** – Node weight vector

**draw**(*shadeRows=True*, *connectNodes=True*, *dpi=200*, *edgeColors=None*)

This function draws the incidence matrix of the hypergraph object. It calls the function `HAT.draw.incidencePlot`, but is provided to generate the plot directly from the object.

**Parameters**

- **shadeRows** – shade rows (bool)
- **connectNodes** – connect nodes in each hyperedge (bool)
- **dpi** – the resolution of the image (int)
- **edgeColors** – The colors of edges represented in the incidence matrix. This is random by default

**Returns**
*matplotlib* axes with figure drawn on to it

`dual()`

The dual hypergraph is constructed.

**Returns**
Hypergraph object

**Return type**
*Hypergraph*

Let $H = (V, E)$ be a hypergraph. In the dual hypergraph each original edge $e \in E$ is represented as a vertex and each original vertex $v \in E$ is represented as an edge. Numerically, the transpose of the incidence matrix of a hypergraph is the incidence matrix of the dual hypergraph.

### References

`cliqueGraph()`

The clique expansion graph is constructed.

**Returns**
Clique expanded graph

**Return type**
*networkx.graph*

The clique expansion algorithm constructs a *graph* on the same set of vertices as the hypergraph by defining an edge set where every pair of vertices contained within the same edge in the hypergraph have an edge between them in the graph. Given a hypergraph $H = (V, E_h)$, then the corresponding clique graph is $C = (V, E_c)$ where $E_c$ is defined

$$E_c = \{(v_i, v_j) | \exists\, e \in E_h \text{ where } v_i, v_j \in e\}.$$

This is called clique expansion because the vertices contained in each $h \in E_h$ forms a clique in $C$. While the map from $H$ to $C$ is well-defined, the transformation to a clique graph is a lossy process, so the hypergraph structure of $H$ cannot be uniquely recovered from the clique graph $C$ alone [1].

### References

`lineGraph()`

The line graph, which is the clique expansion of the dual graph, is constructed.

**Returns**
Line graph

**Return type**
*networkx.graph*

### References

**starGraph**()

The star graph representation is constructed.

**Returns**
Star graph

**Return type**
*networkx.graph*

The star expansion of $H = (V, E_h)$ constructs a bipartite graph $S = \{V_s, E_s\}$ by introducing a new set of vertices $V_s = V \cup E_h$ where some vertices in the star graph represent hyperedges of the original hypergraph. There exists an edge between each vertex $v, e \in V_s$ when $v \in V$, $e \in E_h$, and $v \in e$. Each hyperedge in $E_h$ induces a star in $S$. This is a lossless process, so the hypergraph structure of $H$ is well-defined] given a star graph $S$.

### References

**laplacianMatrix**(*type='Bolla'*)

This function returns a version of the higher order Laplacian matrix of the hypergraph.

**Parameters**
**type** (*str, optional*) – Indicates which version of the Laplacin matrix to return. It defaults to `Bolla` [1], but `Rodriguez` [2,3] and `Zhou` [4] are valid arguments as well.

**Returns**
Laplacian matrix

**Return type**
*ndarray*

Several version of the hypergraph Laplacian are defined in [1-4]. These aim to capture the higher order structure as a matrix. This function serves as a wrapper to call functions that generate different specific Laplacians (See `bollaLaplacian()`, `rodriguezLaplacian()`, and `zhouLaplacian()`).

### References

**bollaLaplacian**()

This function constructs the hypergraph Laplacian according to [1].

**Returns**
Bolla Laplacian matrix

**Return type**
*ndarray*

### References

**rodriguezLaplacian()**

This function constructs the hypergraph Laplacian according to [1, 2].

**Returns**
Rodriguez Laplacian matrix

**Return type**
*ndarray*

### References

**zhouLaplacian()**

This function constructs the hypergraph Laplacian according to [1].

**Returns**
Zhou Laplacian matrix

**Return type**
*ndarray*

### References

**adjTensor()**

This constructs the adjacency tensor for uniform hypergraphs.

**Returns**
Adjacency Tensor

**Return type**
*ndarray*

The adjacency tensor $A$ of a $k-$ '$orderhypergraph : math : 'H$ is the multi-way, hypergraph analog of the pairwise, graph adjacency matrix. It is defined as a $k-$ mode tensor ( $k-$ dimensional matrix):

$$A \in \mathbf{R}^{\overbrace{n \times \cdots \times n}^{k \text{ times}}} \text{ where } A_{j_1 \ldots j_k} = \begin{cases} \frac{1}{(k-1)!} & \text{if } (j_1, \ldots, j_k) \in E_h \\ 0 & \text{otherwise} \end{cases},$$

as found in equation 8 of [1].

### References

**degreeTensor()**

This constructs the degree tensor for uniform hypergraphs.

**Returns**
Degree Tensor

**Return type**
*ndarray*

The degree tensor $D$ is the hypergraph analog of the degree matrix. For a $k-$ order hypergraph $H = (V, E)$ the degree tensor $D$ is a diagonal supersymmetric tensor defined

$$D \in \mathbf{R}^{\overbrace{n \times \cdots \times n}^{k \text{ times}}} \text{ where } D_{i \ldots i} = degree(v_i) \text{ for all } v_i \in V$$

### References

`laplacianTensor()`

This constructs the Laplacian tensor for uniform hypergraphs.

**Returns**
Laplcian Tensor

**Return type**
*ndarray*

The Laplacian tensor is the tensor analog of the Laplacian matrix for graphs, and it is defined equivalently. For a hypergraph $H = (V, E)$ with an adjacency tensor $A$ and degree tensor $D$, the Laplacian tensor is

$$L = D - A$$

### References

`tensorEntropy()`

Computes hypergraph entropy based on the singular values of the Laplacian tensor.

**Returns**
tensor entropy

**Return type**
*float*

Uniform hypergraph entropy is defined as the entropy of the higher order singular values of the Laplacian matrix [1].

### References

`matrixEntropy(type='Rodriguez')`

Computes hypergraph entropy based on the eigenvalues values of the Laplacian matrix.

**Parameters**
**type** (*str, optional*) – Type of hypergraph Laplacian matrix. This defaults to 'Rodriguez' and other choices inclue `Bolla` and `Zhou` (See: `laplacianMatrix()`).

**Returns**
Matrix based hypergraph entropy

**Return type**
*float*

Matrix entropy of a hypergraph is defined as the entropy of the eigenvalues of the hypergraph Laplacian matrix [1]. This may be applied to any version of the Laplacian matrix.

### References

**avgDistance**()

Computes the average pairwise distance between any 2 vertices in the hypergraph.

**Returns**
avgDist

**Return type**
*float*

The hypergraph is clique expanded to a graph object, and the average shortest path on the clique expanded graph is returned.

**ctrbk**(*inputVxc*)

Compute the reduced controllability matrix for $k-$ uniform hypergraphs.

**Parameters**
**inputVxc** (*ndarray*) – List of vertices that may be controlled

**Returns**
Controllability matrix

**Return type**
*ndarray*

### References

**bMatrix**(*inputVxc*)

Constructs controllability $B$ matrix commonly used in the linear control system

$$\frac{dx}{dt} = Ax + Bu$$

**Parameters**
**inputVxc** (*ndarray*) – a list of input control nodes

**Returns**
control matrix

**Return type**
*ndarray*

### References

**clusteringCoef**()

Computes clustering average clustering coefficient of the hypergraph.

**Returns**
average clustering coefficient

**Return type**
*float*

For a uniform hypergraph, the clustering coefficient of a vertex $v_i$ is defined as the number of edges the vertex participates in (i.e. $deg(v_i)$) divided by the number of $k - `wayedgesthatcouldexistamongvertex : math : `v_i$ and its neighbors (See equation 31 in [1]). This is written

$$C_i = \frac{deg(v_i)}{\binom{|N_i|}{k}}$$

where $N_i$ is the set of neighbors or vertices adjacent to $v_i$. The hypergraph clustering coefficient computed here is the average clustering coefficient for all vertices, written

$$C = \sum_{i=1}^{n} C_i$$

### References

**centrality**(*tol=0.0001*, *maxIter=3000*, *model='LogExp'*, *alpha=10*)

Computes node and edge centralities.

**Parameters**

- **tol** (*int*, optional) – threshold tolerance for the convergence of the centrality measures, defaults to 1e-4
- **maxIter** (*int, optional*) – maximum number of iterations for the centrality measures to converge in, defaults to 3000
- **model** (*str, optional*) – the set of functions used to compute centrality. This defaults to 'LogExp', and other choices include 'Linear', 'Max' or a list of 4 custom function handles (See [1]).
- **alpha** (*int, optional*) – Hyperparameter used for computing centrality (See [1]), defaults to 10

**Returns**
vxcCentrality

**Return type**
*ndarray* containing centrality scores for each vertex in the hypergraph

**Returns**
edgeCentrality

**Return type**
*ndarray* containing centrality scores for each edge in the hypergraph

### References

## HAT.HAT module

HAT.HAT.**directSimilarity**(*HG1*, *HG2*, *measure='Hamming'*)

This function computes the direct similarity between two uniform hypergraphs.

**Parameters**

- **HG1** (*Hypergraph*) – Hypergraph 1
- **HG2** (*Hypergraph*) – Hypergraph 2
- **measure** (*str, optional*) – This sepcifies which similarity measure to apply. It defaults to `Hamming`, and `Spectral-S` and `Centrality` are available as other options as well.

**Returns**
Hypergraph similarity

**Return type**
*float*

### References

HAT.HAT.**indirectSimilarity**(*G1*, *G2*, *measure='Hamming'*, *eps=0.01*)

This function computes the indirect similarity between two hypergraphs.

**Parameters**

- **G1** (*nx.Graph* or *ndarray*) – Hypergraph 1 expansion
- **G2** (*nx.Graph* or *ndarray*) – Hypergraph 2 expansion
- **measure** (*str*, optional) – This specifies which similarity measure to apply. It defaults to `Hamming` , and `Jaccard` , `deltaCon` , `Spectral` , and `Centrality` are provided as well. When `Centrality` is used as the similarity measure, `G1` and `G2` should *ndarray* s of centrality values; Otherwise `G1` and `G2` are *nx.Graph*s or *ndarray** s as adjacency matrices.
- **eps** (*float*, optional) – a hyperparameter required for deltaCon similarity, defaults to 10e-3

**Returns**
similarity measure

**Return type**
*float*

### References

HAT.HAT.**multicorrelations**(*D*, *order*, *mtype='Drezner'*, *idxs=None*)

This function computes the multicorrelation among pairwise or 2D data.

**Parameters**

- **D** (*ndarray*) – 2D or pairwise data
- **order** (*int*) – order of the multi-way interactions
- **mtype** (*str*) – This specifies which multicorrelation measure to use. It defaults to `Drezner` [1], but `Wang` [2] and `Taylor` [3] are options as well.
- **idxs** (*ndarray*, optional) – specify which indices of `D` to compute multicorrelations of. The default is `None`, in which case all combinations of `order` indices are computed.

**Returns**
A vector of the multicorrelation scores computed and a vector of the column indices of `D` used to compute each multicorrelation.

**Return type**
*(ndarray, ndarray)*

### References

HAT.HAT.**uniformErdosRenyi**(*v*, *e*, *k*)

This function generates a uniform, random hypergraph.

**Parameters**

- **v** (*int*) – number of vertices
- **e** (*int*) – number of edges
- **k** (*int*) – order of hypergraph

**Returns**
Hypergraph

**Return type**
*Hypergraph*

HAT.HAT.**load**(*dataset='Karate'*)

This function loads built-in datasets. Currently only one dataset is available and we are working to expand this.

**Parameters**
**dataset** (*str, optional*) – sets which dataset to load in, defaults to 'Karate'

**Returns**
incidence matrix or graph object

**Return type**
*ndarray* or *nx.Graph*

HAT.HAT.**hyperedges2IM**(*edgeSet*)

This function constructs an incidence matrix from an edge set.

**Parameters**
**edgeSet** (*ndarray*) – a $e \times k$ matrix where each row contains $k$ integers that are contained within the same hyperedge

**Returns**
a $nimese$ incidence matrix where each row of the edge set corresponds to a column of the incidence matrix. $n$ is the number of nodes contained in the edgeset.

**Return type**
*ndarray*

HAT.HAT.**hyperedgeHomophily**(*H*, *HG=None*, *G=None*, *method='CN'*)

This function computes the hyperedge homophily score according to the below methods. The homophily score is the average score based on structural similarity of the vertices in hypredge *H* in the clique expanded graph *G*. This function is an interface from *HAT* to *networkx* link prediction algorithms.

**Parameters**

- **G** (*networkx.Graph*) – a pairwise hypergraph expansion
- **H** (*ndarray*) – hyperedge containing individual vertices within the edge
- **method** – specifies which structural similarity method to use. This defaults to *CN* common neighbors.

HAT.HAT.**edgeRemoval**(*HG*, *p*, *method='Random'*)

**This function randomly removes edges from a hypergraph. In [1], four primary reasons are given for data missing in pairwise networks:**

1. random edge removal
2. right censoring
3. snowball effect
4. cold-ends

This method removes edes from hypergraphs according to the multi-way analogue of these.

### References

HAT.HAT.**randomRemoval**(*HG*, *p*)

HAT.HAT.**rightCensorRemoval**(*HG*, *p*)

HAT.HAT.**coldEndsRemoval**(*HG*, *p*)

HAT.HAT.**snowBallRemoval**(*HG*, *p*)

## HAT.draw module

HAT.draw.**incidencePlot**(*H*, *shadeRows=True*, *connectNodes=True*, *dpi=200*, *edgeColors=None*)

Plot the incidence matrix of a hypergraph.

**Parameters**

- **H** – a HAT.hypergraph object
- **shadeRows** – shade rows (bool)
- **connectNodes** – connect nodes in each hyperedge (bool)
- **dpi** – the resolution of the image (int)
- **edgeColors** – The colors of edges represented in the incidence matrix. This is random by default

**Returns**

matplotlib axes with figure drawn on to it

## HAT.multilinalg module

HAT.multilinalg.**hosvd**(*T*, *M=True*, *uniform=False*, *sym=False*)

Higher Order Singular Value Decomposition

**Parameters**

- **uniform** – Indicates if T is a uniform tensor
- **sym** – Indicates if T is a super symmetric tensor
- **M** – Indicates if the factor matrices are required as well as the core tensor

**Returns**

The singular values of the core diagonal tensor and the factor matrices.

HAT.multilinalg.**supersymHosvd**(*T*)

Computes the singular values of a uniform, symetric tensor. See Algorithm 1 in [1].

**Parameters**

**T** – A uniform, symmetric multidimensional array

**Returns**

The singular values that compose the core tensor of the HOSVD on T.

### References

HAT.multilinalg.**HammingSimilarity**(*A1*, *A2*)

Computes the Spectral-S similarity of 2 Adjacency tensors [1].

**Parameters**

- **A1** (*ndarray*) – adjacency tensor 1
- **A2** (*ndarray*) – adjacency tensor 2

**Returns**
Hamming similarity measure

**Return type**
*float*

### References

HAT.multilinalg.**SpectralHSimilarity**(*L1*, *L2*)

Computes the Spectral-S similarity of 2 Laplacian tensors [1].

**Parameters**

- **L1** (*ndarray*) – Laplacian tensor 1
- **L2** (*ndarray*) – Laplacian tensor 2

**Returns**
Spectral-S similarity measure

**Return type**
*float*

### References

HAT.multilinalg.**kronExponentiation**(*M*, *x*)

Kronecker Product Exponential.

**Parameters**

- **M** (*ndarray*) – a matrix
- **x** (*int*) – power of exponentiation

**Returns**
Krnoecker Product exponentiation of $\mathbf{M}$ a total of $\mathbf{x}$ times

**Return type**
*ndarray*

This function performs the Kronecker Product on a matrix $M$ a total of $x$ times. The Kronecker product is defined for two matrices $A \in \mathbf{R}^{l \times m}$, $B \in \mathbf{R}^{m \times n}$ as the matrix

$$A \bigotimes B = \begin{pmatrix} A_{1,1}B & A_{1,2}B & \dots & A_{1,m}B \\ A_{2,1}B & A_{2,2}B & \dots & A_{2,m}B \\ \vdots & \vdots & \ddots & \vdots \\ A_{l,1}B & A_{l,2}B & \dots & A_{l,n}B \end{pmatrix}$$

### Bug Reporting

Please report all bugs or defects in HAT to this page.

### Hypergraph References

### Bug Reporting

Please report all bugs or defects in HAT to this page.

### Development

This page contains the goals for the next version of HAT. To contribute, for information on the below, or to request specific features in HAT, please contact us at jpic@umich.edu

### Software Development

1. Automated testing of software
2. Increased number of tutorial
3. Increased number of hypergraph visualization methods
4. Add hyperlink prediction module
5. Include built-in hypergraphs

### Hypergraph Methods Development

1. Directed hypergraphs
2. Nonuniform hypergraphs
3. Large hypergraph constructions

### Bug Reporting

Please report all bugs or defects in HAT to this page.

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## h

# INDEX

## T



## U



## Z