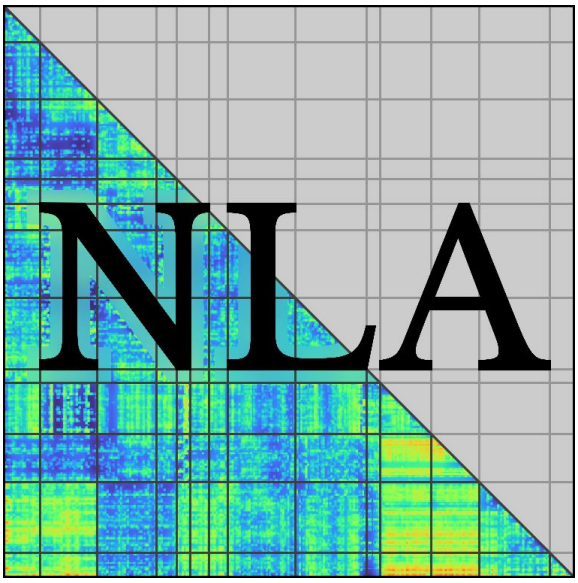


NLA Toolbox Manual

Version 1.0.0

Table of Contents

Preface

This is the reference manual for the Network Level Analysis (NLA) Toolbox. NLA is an extensible MATLAB-based software package for the analysis of behavioral associations with brain connectivity data. NLA utilizes a model-based statistical approach known variously as 'pathway analysis', 'over-representation analysis', or 'enrichment analysis', which was first used to describe behavioral or clinical associations in genome-wide association studies.[1–4]

Enrichment is a model-based data reduction approach to elucidate statistically significant network-features. The suite developed here includes data-driven permutation-based false-positive-rate procedures that manage multiple comparisons corrections for one or two independent groups.

Software and Hardware Requirements

NLA has been tested on MATLAB 2020b on Ubuntu 20.04. Current release of the GUI is not supported for Windows. NLA requires the Parallel Processing and Statistics and Machine Learning Toolboxes. Best performance will be achieved on a server setup with multiple cores to support parallel processing (particularly for the permutation testing portion of the toolbox).

NLA Overview

The connectome and network structure

The term connectome essentially describes any network description of whole brain connectivity, from the microscale of single neurons and synapses up to the macroscale of entire brain regions and pathways.[5] Connectomics is an ever-advancing field, and large-scale scientific endeavors such as the NIH's Human Connectome Project have made significant progress in mapping, analyzing, and understanding the human connectome. Contemporary connectome research views the brain as an extensive, complex network of non-adjacent, yet functionally and structurally connected brain regions.[6,7] The connectome can be utilized to assess whole-brain associations between behavior and spatially distinct neural networks.

MRI has traditionally been viewed as the gold standard for mapping the connectome and has been used to demonstrate consistencies between the spatial topology of task-based activation studies and the brain networks derived from task-free functional connectivity.[7,8] Contemporary cluster correction approaches do not utilize the spatial topology of brain networks when estimating cluster size significance.[9–11] Therefore, there is an urgent need for standardized tools that address the robust hierarchical network structure of the brain and the limitations of contemporary neuroimaging analysis approaches by utilizing this biologically informed network structure to increase reproducibility and biological interpretation of neuroscience results.

Why use this toolbox?

The NLA toolbox is designed to address the multiple comparisons problem that occurs within connectome research, wherein studies use hundreds of regions of interest (ROI) to create connectomes with thousands of potential connections, yet they lack the tools to establish statistical significance when analyzing associations between connectome and behavior. For example, previous research failed to find any significant differences in brain connectivity that passed a connectome-wise false discovery rate (FDR) correction between individuals with a neurological disorder and healthy controls—a finding which contradicts the recognized role of the brain in neurological functioning.[12] Other studies have found connectome-behavior associations that pass the FDR correction, but lack the statistical tools necessary to definitively establish these observations.[13] NLA, therefore, serves as a valuable tool for the statistical quantification of network-level associations with behavior. The toolbox relies on cross-disciplinary biostatistical approaches to evaluate brain-behavior relationships within the connectome and allows for control of FDR at the network level. In this way, NLA diverges from most contemporary tools with a focus on single connection associations, in that it is not dependent on edgewise false positive rate (FPR) or spatially contiguous brain regions. By organizing connectivity-behavior associations according to an a-priori model of underlying neurobiology (i.e., networks), NLA leverages the structure of the human connectome and provides a framework for rational interpretation and replication of findings across research methodologies. Finally, the integration of connectome analysis and visualization techniques within a single, extensible MATLAB-based pipeline makes NLA an expedient tool for statistical testing and production of publication quality images all in one package.

Introduction to NLA and enrichment

Network Level Analysis uses enrichment to evaluate whether pairs of networks demonstrate significant clustering of strong brain-behavior correlations. Enrichment applies common statistical tests to measure the clustering of associations within a given network pair and reduces the number of comparisons to those performed at the network level.[4] Network level statistics such as the Chi-Square test, Hypergeometric test, and Kolmogorov-Smirnov test have been used in numerous network-level investigations including joint attention and motor function in infants and toddlers, maternal inflammation during gestation, motor and attention development in very preterm children, sex differences during fetal brain development, and autism in adults.[14–24]

Edge-level Statistic

First, correlations are calculated between behavioral scores and Fisher z-transformed functional connectivity correlation measures for each pair of ROI. For behavioral scores that are normally distributed, Pearson r correlations are used to calculate the associations. Non-parametric Spearman rank correlations are used to assess the relationship between functional connectivity and behavioral scores that are not normally distributed. Other tests of correlation such as Kendall's tau and 2-sample Welch's t can also be used. Network pairs are then tested for enrichment of strong correlation values, defined as only those values that remain after being nominally thresholded. An uncorrected p-threshold (e.g., 0.05 or 0.01) is applied and the remaining correlations are binarized.

Network Level Statistics

After the edge-level statistic matrix has been calculated, it is given as input to a variety of network-level tests. First, it is input directly to the tests, and the resulting statistic is called the "non-permuted network level statistic" (for every given network-level test). Then, permuted edge-level statistics are calculated via the same method as described previously, but with the behavioral scores permuted across subjects. The network-level test is performed on this as well, and the significance of permuted network-level statistics ranked against the non-permuted, to calculate the permuted experiment-wide p-value (an empirical p-value produced from this ranking). Additionally, "single-sample within-net-pair" statistics are calculated for each test, which, rather than comparing a given network to the connectome over a number of permutations (as in the permuted network-level test), performs a single-sample test on the network alone, which is then ranked against permutations of said network similarly to the permuted network-level test.

A number of statistic tests are utilized at the network level. The 1-degree-of-freedom χ^2 test is used to compare the observed number of strong (thresholded and binarized) brain-behavior correlations within one pair of functional networks to the number of strong brain-behavior correlations that would be expected if strong correlations were uniformly distributed across all possible network pairs. A large resulting test statistic can indicate that the number of strong correlations within a specific network pair is enriched. The hypergeometric test aims to assess the likelihood of observing a given number of strong correlations within a pair of networks, given (1) the total number of strong correlations observed over the entire connectome and (2) the total number of possible hits for that network pair (i.e., the total number or ROI-pairs within a given network pair). Other tests such as Kolmogorov-Smirnov, Wilcoxon rank-sum, Welch's t can be used, as well as Cohen's d to measure effect sizes.

As described, significance for all statistical tests is determined using permutation testing. Behavioral labels are randomly permuted and correlated with the connectome data (typically 10k times) to create null brain-behavior correlation matrices. Tests are calculated on these permuted brain-behavior correlation matrices generating a null distribution of network level statistics. The measured (real) test statistics are compared to this null distribution to establish network-level significance.

NLA Alternatives / Comparison to other analysis methods

The NLA toolbox's use of a novel enrichment approach makes it a transformative tool in connectome-wide association studies, given that all current enrichment analysis methods are built for use with genome data and NLA is the first enrichment tool designed to analyze the connectome. Many alternative methods for connectome analysis rely on spatial extent cluster correction in order to control voxel-wise whole brain connectome FPR.[25,26] Despite mounting evidence that spatially non-contiguous brain regions are strongly correlated and often co-activate to the same stimuli, cluster extent correction is often regarded as the ideal thresholding approach in human connectome literature. By basing statistical significance on contiguous voxels, however, cluster extent correction methods fail to account for this covariance structure. Therefore, brain regions that are known to be highly correlated and part of the same network—such as the anterior cingulate and posterior cingulate—may be thresholded separately, resulting in one or both separate regions not meeting statistical thresholds.[27] NLA is distinguished from the cluster extent correction methodology in that it groups highly correlated, non-contiguous brain regions based on pre-defined network modules prior to estimating network-level significance.

Network Based Statistic (NBS)

Given this deviation from the popular extent cluster correction thresholding method, the most conceptually similar existing connectome analysis approach to NLA is the Network Based Statistic (NBS) toolbox.[28] NBS was the first tool control the edgewise FPR by leveraging graph-based estimates of modularity. Still, several crucial differences exist between NLA and NBS: (a) the results from NBS focus on edgewise significance as opposed to network-level significance, (b) NBS does not have a built-in visualization functionality, and (c) NBS allows for different module sizes, number of network modules, and configurations of edges assigned to network modules across various clinical populations, but draws no conclusions regarding the biological relevance of identified networks. The NLA pipeline addresses this issue by presenting a vast array of analysis and visualization options that utilize biologically informed hierarchical organization models of the brain.

Graph Theoretical Toolboxes

Graph Theoretical Toolboxes are another comparable approach to NLA, offering an analysis methodology to quantify network characteristics such as integration, segregation, resilience, and relative contribution of individual network nodes to overall information flow within the network.[29] Various other toolboxes have been created to address network thresholding, graph metric calculation, and graph visualization—such as GRETNA, GEPHI, and BrainNet Viewer. Additional methodologies aim to determine network topology differences by leveraging generalized estimating equations and generalized linear and nonlinear mixed models.[30–32] Each of these tools has helped to advance the application of graph theory approaches to connectome analysis. The NLA toolbox estimates statistical associations edgewise, rather than on network topology features, thereby providing a crucial and complementary approach to the existing collection of brain network analysis tools.

Statistical Inference and the use of liberal primary thresholds

NLA establishes statistical significance in the weak sense similar to traditional voxelwise cluster-level inference.[33] In voxelwise cluster correction, a liberal primary threshold is employed in addition to a cluster-extent threshold (determined by e.g., random field theory or Monte Carlo simulations). The resulting clusters are significant but inferences cannot be made about any particular sub-regions or voxels within a cluster. Similarly, NLA employs a liberal primary threshold in order to calculate the network-level statistic and significance is established with permutation testing, but claims cannot be made about the significance of any given ROI-pair within the network. One could apply an FDR correction within each network pair similar to the statistics outlined in the Network Based Statistics toolbox though this would still only control the false positive rate in the weak sense. The motivation of all of these approaches (cluster-level inference, network-level enrichment, network-based statistic) is to control the false positive rate when a massive number of tests are performed. Controlling the false positive rate in the strong sense with several thousand functional connections (e.g., 30k) will often result in no single ROI-pair surviving OR a few scattered ROI-pairs surviving with no clear biological pattern.[12]

NLA Methodology

Brain Network Map Selection

NLA requires the user to specify the network map that will be used to depict the known architecture of the human connectome, which is crucial given that the network map selection affects both statistical significance testing and interpretation.[34] The current pipeline uses network maps that are generated with Infomap, due to its greater congruence with networks derived from task-activation and seed-based connectivity studies than alternative modularity algorithms.[7,35] Network maps can be generated using one's preferred algorithm or one of several published ROI and corresponding network map options that will be included in the NLA toolbox.[6,7,36–39] The use of standardized ROI and network maps creates a common, reproducible framework for testing brain-behavior associations across connectome research.

General Linear Model / Edge-wise Statistical Model Selection

NLA also requires the user to specify the desired statistical model for testing associations between behavioral data and edge-wise—or ROI-pair connectivity—connectome data. The analysis pipeline within the NLA toolbox offers both parametric and non-parametric correlation.

Connectivity Matrices

Other software packages are used to create the connectivity matrices that are provided as input into the NLA toolbox. One useful option for mapping functional connectivity matrices is CONN—a MATLAB-based software with the ability to compute, display, and analyze functional connectivity in fMRI.

The NLA Method

First, connectome-wide associations are calculated between ROI-pair connectivity and behavioral data, resulting in a set of standardized regression coefficients that specify the brain-behavior association at each ROI-pair of the connectome matrix. Next, network level analysis—consisting of transformation of the edge-wise test statistics and enrichment statistic calculation[40] - is done to determine which networks are strongly associated with the behavior of interest.

Both p-value and test-statistic binarization are offered in the current NLA pipeline.[14,15] Prior research has supported the incorporation of a proportional edge density threshold, given that uneven edge density thresholds have been shown to unfairly bias results.[41]

For enrichment statistic calculation, NLA offers a number of statistical tests. Prior research has relied on chi-square and Fisher's Exact test, as well as a Kolmogorov-Smirnov (KS) test and non-parametric tests based on ranks, which compare the distribution of test values within a region to other regions.[15,17,42,43] In addition, KS alternatives such as averaging or minmax have also shown promise in connectome applications.[44–47]

NLA then conducts data-driven permutation testing to establish significance. In the NLA toolbox, network level significance is determined by comparing each measured enrichment statistic to permuted enrichment p-values which are calculated by randomly shuffling behavior vector labels and computing the enrichment statistic many times to produce a null distribution for each network. The FPR is controlled at the network level using Bonferroni correction. Therefore, NLA is able to retain edge-wise correlations within each network module, but network communities are used to reduce the number of comparisons

and control the FPR at the network level. After significance is determined, the pipeline allows users to create publication quality images to visualize network level findings both in connectome format and on the surface of the brain.

Note: While the behavior vector labels are shuffled to conduct permutations in the enrichment pipeline, functional connectivity data are not shuffled in order to preserve the inherent covariant structure of the data across permutations.

How should the test statistic threshold be chosen?

A nominal threshold is used for the thresholding and binarization step of the edge-level tests. The nominal threshold is uncorrected and is typically set at 0.05 or 0.01 in the edge-level prob_max field. In contrast, a network-level corrected threshold using the Bonferroni method is used in the net-level statistics, where the nominal threshold is divided by the number of tests being done to correct for multiple comparisons.

How should the networks be chosen?

There are many canonical ROI sets and there are many network definitions. Some of these network definitions include ROI that are not consistently assigned to any network. These ROI are typically removed prior to network level analysis, as is the case in the Seitzman_15nets_288ROI_on_TT and the Gordon_12nets_286parcels_on_MNI network atlases included in this version of the toolbox. Network atlases that are not included in this package may also be used, but they must first be formatted into the correct structure.

Setup

Add NLA Folders to MATLAB Path

In order to for any NLA functions to work, MATLAB must be able to find them on the path. To do this, in the MATLAB file explorer, navigate to where you have downloaded or cloned the NetworkLevelAnalysis folder to. Right click the folder, hover over 'Add to Path' in the context menu, and click the 'Selected Folders and Subfolders' option.

NOTE: If you only add the base 'NetworkLevelAnalysis' folder to the path the code will not work, you must pick the 'Selected Folders and Subfolders' option.

Running the GUI

To open the GUI, navigate to the root directory of the NetworkLevelAnalysis package in MATLAB and run the command `NLA_GUI` via the MATLAB command line.

Note: Running the GUI through an X11-based remote connection (eg: MobaXTerm or similar) can be very laggy in some cases. It is strongly recommended to use the GUI through a more modern remote protocol such as VNC instead.

Running as a pipeline script

To run NLA via a script instead, open the file main_pipeline.m (located in the root directory of the NetworkLevelAnalysis package) in MATLAB, and proceed through the stages of the pipeline. There is also a pipeline for precalculated data located in precalculated_pipeline.m

Note: The pipeline scripts are more complex and easy-to-mess-up than the GUI, and should only be used if you have a good reason to do so.

Using individual NLA functions

To use NLA functions within your own code or scripts, add the `NetworkLevelAnalysis` folder to your path. Most NLA functions are contained within the `+nla` namespace and its sub-namespaces: for example, if you wanted to use the `drawMatrixOrg` function you would call it as `nla.gfx.drawMatrixOrg()`. You can also use the command `import nla.*` to import all NLA functions and classes, allowing you to reference them as eg: `gfx.drawMatrixOrg()`, to save some typing.

Getting Started

Running with example data

First, open the NLA software (as described in Setup). Select "Pearson's r" as the edge-level test from the edge-level test dropdown.

Click 'Select' to choose the network atlas via the GUI, navigating to the **`support_files`** folder within your NetworkLevelAnalysis installation and selecting **`Wheelock_2020_CerebralCortex_15nets_288ROI_on_MNI.mat`**. This file is used to parcellate the data.

Then, select the functional connectivity, located in the **`examples/fc_and_behavior`** folder under the name **`sample_func_conn.mat`** - click "yes" to Fisher z-transform the data. Take a moment to visualize the FC average by clicking "View" and note that the FC appears to match the parcellation, (effects generally line up with network boundaries) – this can be a useful diagnostic tool if you are having issues with parcellations not matching data.

Finally, load the behavior **`sample_behavior.mat`** from the **`examples/`** folder – you will need to switch the "file type" from "Text" to "MATLAB table" in the file browser. Set the behavioral variable to 'Flanker_AgeAdj' by clicking that column within the table and then the 'Set Behavior' button.

Having finished our edge-level inputs, we now move over to the network-level test pane. Select all tests by clicking the top one, and then shift-clicking the bottom one.

Now we can run the tests via the bottom-right Run button. This will pop up a result window which will, at first, run the edge-level test. You can then visualize the edge-level result with 'View', and continue running the net-level tests if desired with 'Run' (in the results window). This will take longer, and when completed the network-level results can be visualized by selecting any particular one from the list, and then 'View figures'. Other visualizations such as chord plots and convergence maps can also be produced, following the same procedure of selecting one or more tests (Ctrl or Shift-click) and then the button. You can also save results (File > Save), which can be opened in the software from the main NLA window (File > Open previous result), or if you wish to access the raw numbers the results can also be loaded into MATLAB, as they are just structured .mat files (The structure of these results should be fairly intuitive to a user with some MATLAB experience, but it may be useful to consult the Variable name glossary to find exactly what you are looking for.

Running with example pre-calculated data

Similarly to the previous example, open the GUI and load the **`Wheelock_2020_CerebralCortex_15nets_288ROI_on_MNI.mat`** parcellation. However, this time you should select the "Precalculated data" edge-level test. Then, load the four input matrices from their corresponding files in the **`examples/precalculated`** folder – 1 file each for:

- Observed coefficients
- Observed, thresholded p-values (logical values, **`true`** meaning significance, **`false`** meaning otherwise)
- Permuted coefficients

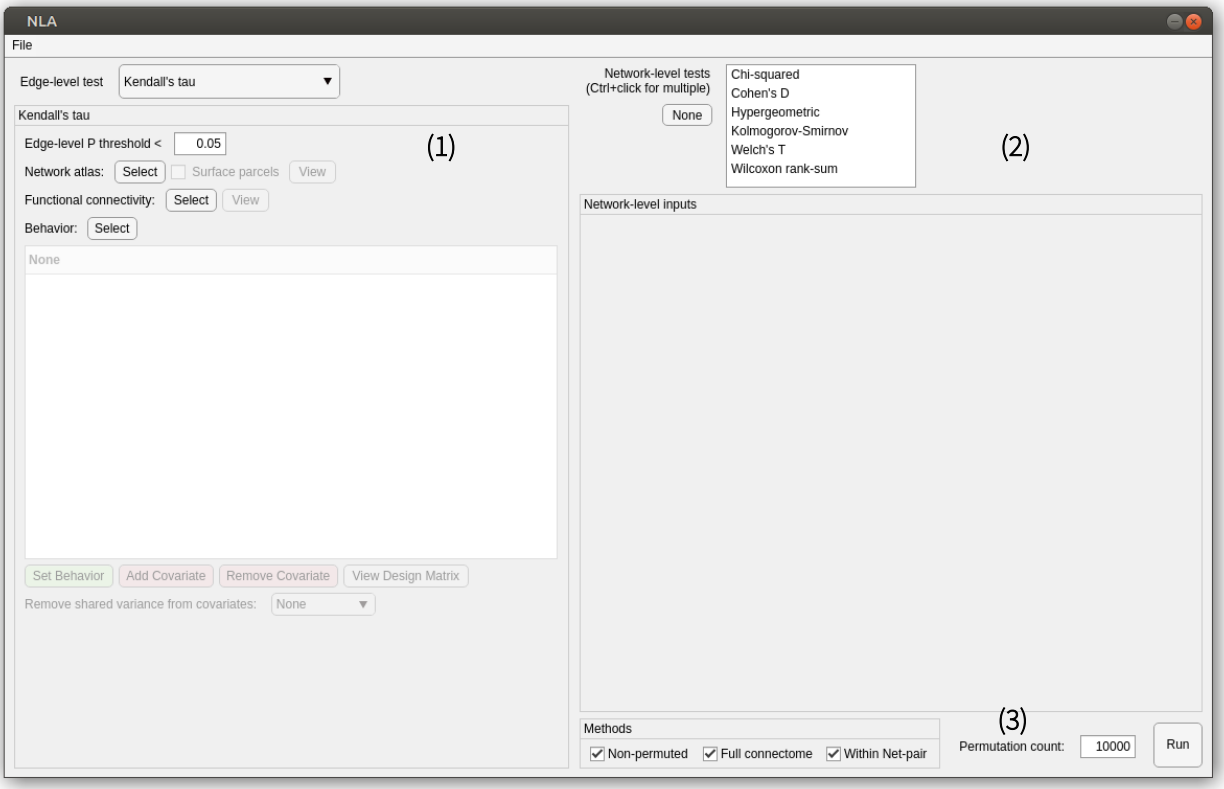
- Permuted, thresholded p-values

Set the lower and upper coefficient bounds to the range of your coefficient, in the example case, we are using beta weights with a range of -2 to 2, and you should set the bound variables accordingly. In the bottom right corner, set' perm count' to correspond with the number of permutations provided in the precalculated permuted matrices - in the example case, 600 permutations were provided. Then, run the analysis as described previously. *Further information on the precalculated data loader.*

A brief introduction to the GUI

Main window

After starting the NLA GUI, the main window will appear with a number of options.



1. Edge-level statistical test pane

The edge-level test pane on the left is populated with all input fields necessary to run the current edge-level test (visible in the 'Edge-level test' selector in the top left). Many fields are populated with reasonable default values, and if you switch between tests values are preserved for ease of use. For more information on edge-level tests, see the Edge-level statistical tests section of this documentation.

2. Network-level statistical tests pane

The net-level tests pane on the right behaves similarly to the edge-level tests pane, however, you can select multiple network-level tests via Ctrl + clicking them in the selector box. You can also click one test and Shift + click another to select the range of tests between them. For more information on network-level tests, see the Network-level statistical tests section of this documentation.

3. Run options

Method: Non-permuted, Full connectome, Within Net-pair

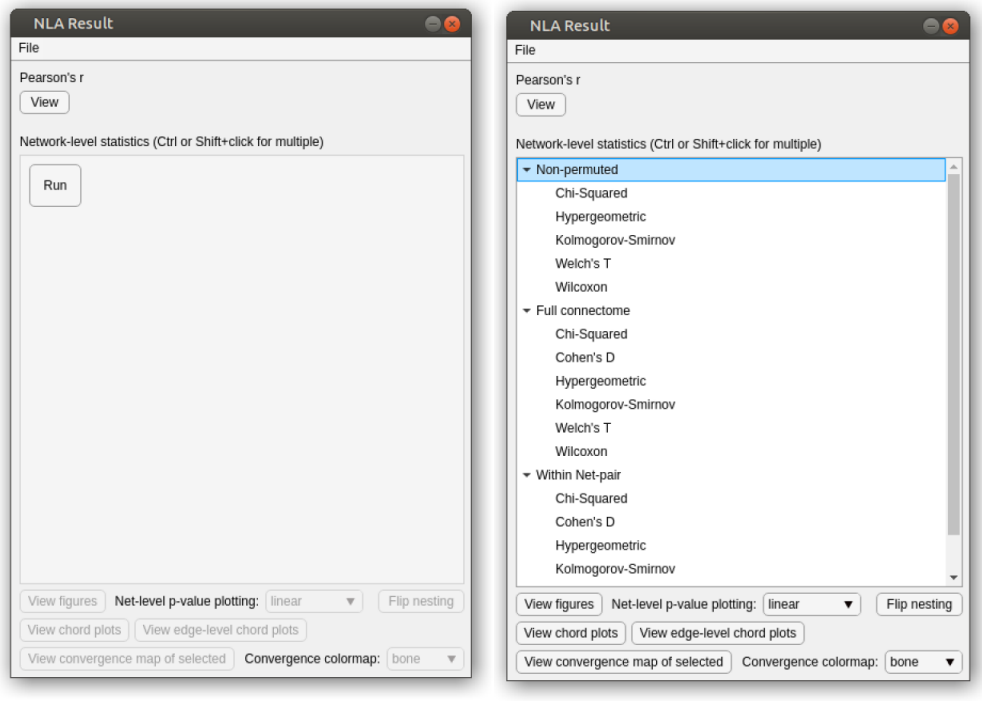
Each method necessarily runs all previous methods, eg: the full connectome method requires that the non-permuted method is also run, and the within net-pair method requires that the full connectome method is run. Each method is detailed further in the Network-level statistical tests > Methods section of this documentation.

Permutation count: 0 – Infinity
Permutation count determines the number of times to permute the data for the full connectome and within net-pair methods. More permutations will provide more precise results but will take longer to run. If you select the non-permuted method, permutation count will be set to zero as no permutation will be performed.

Run: Run the selected tests with the given parameters. If you don't fill out all necessary parameters, you will be prompted to complete them.

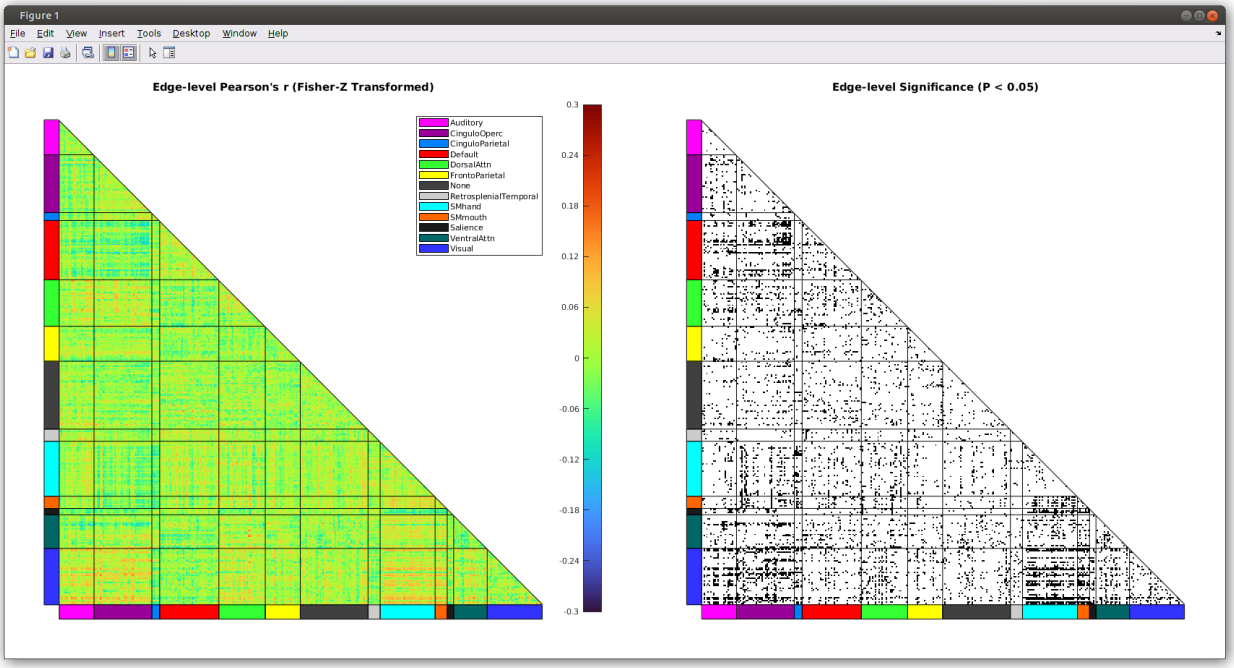
Load results
To load previous result files, click File > Load previous results, and select your result file.

Results window
After running the tests or loading a previous result, the results window will appear with a number of options.



Left: Results window after clicking 'Run' from the main NLA window.
Right: Results window after all tests are completed, or the user loaded a previous result.

View non-permuted edge-level results
Once the non-permuted edge-level test has finished running, you can view its results immediately via the 'View' button. This is important for some edge-level tests such as the Sandwich Estimator which require contrasts to be set at this point before continuing. It can also warn you if something is immediately wrong with your input data format.



A non-permuted edge-level result figure.

Run tests

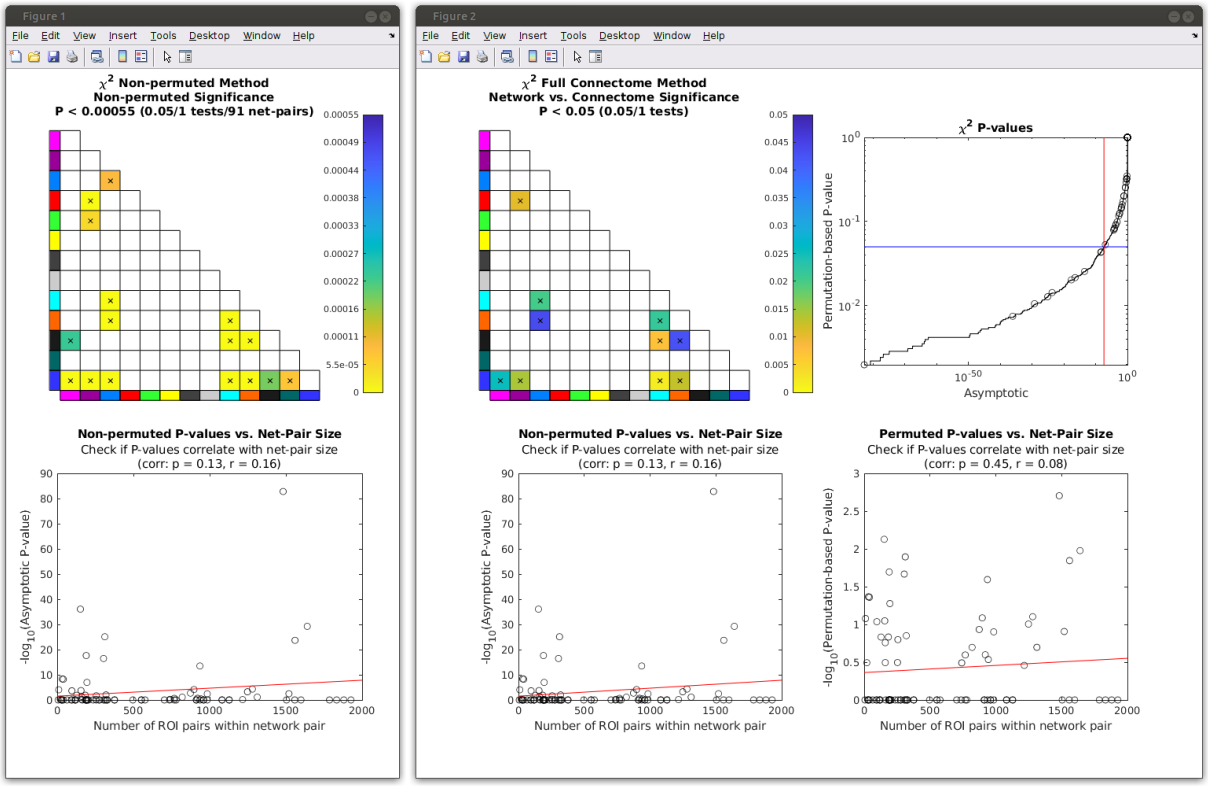
You can then run non-permuted net-level tests and permuted methods by clicking the 'Run' button.

Viewing results

To view network-level and permuted results, select them in the 'Network-level statistics' pane and click one of the 'View' buttons below. It can be useful to select multiple test or method results to view at once via either Ctrl + clicking to select multiple results, or Shift + clicking to select a range of results.

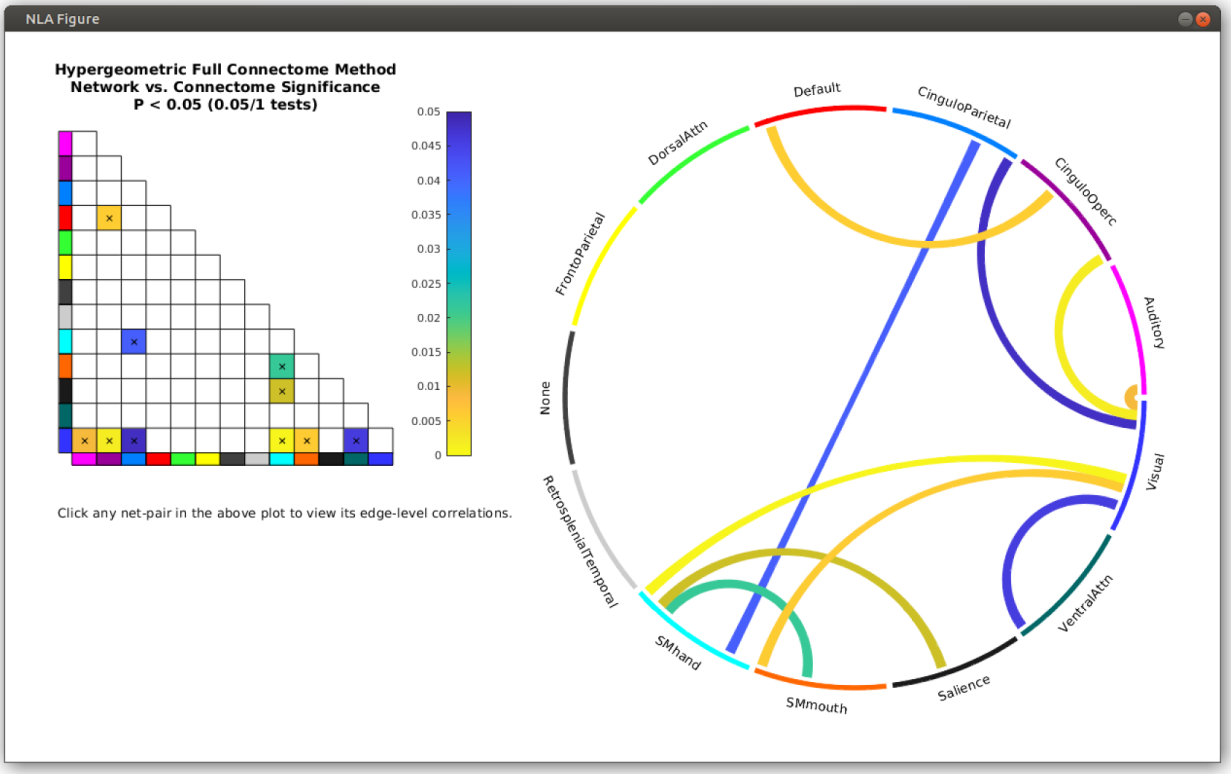
View figures

Display a triangular matrix plot of net-level P-values of the selected test(s), and some diagnostic plots.



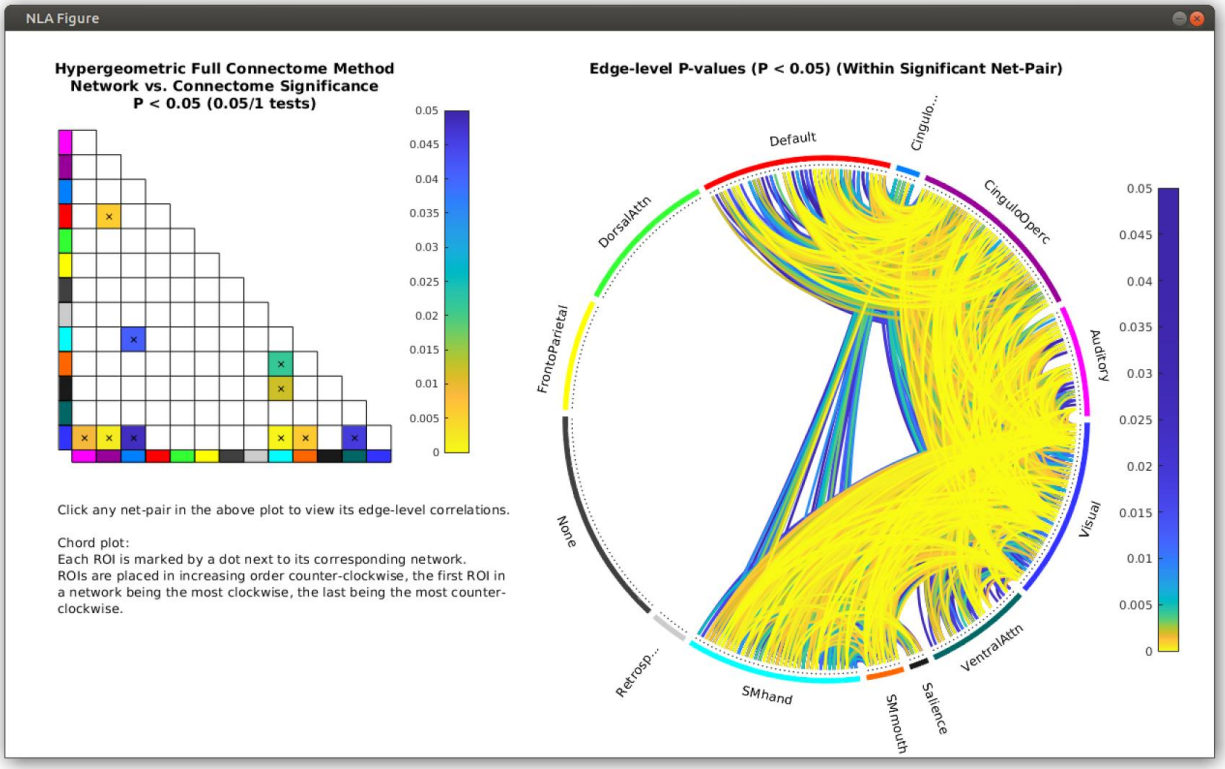
View chord plots

Display a triangular matrix of net-level P-values of the selected test(s), as well as a chord plot displaying the same information.



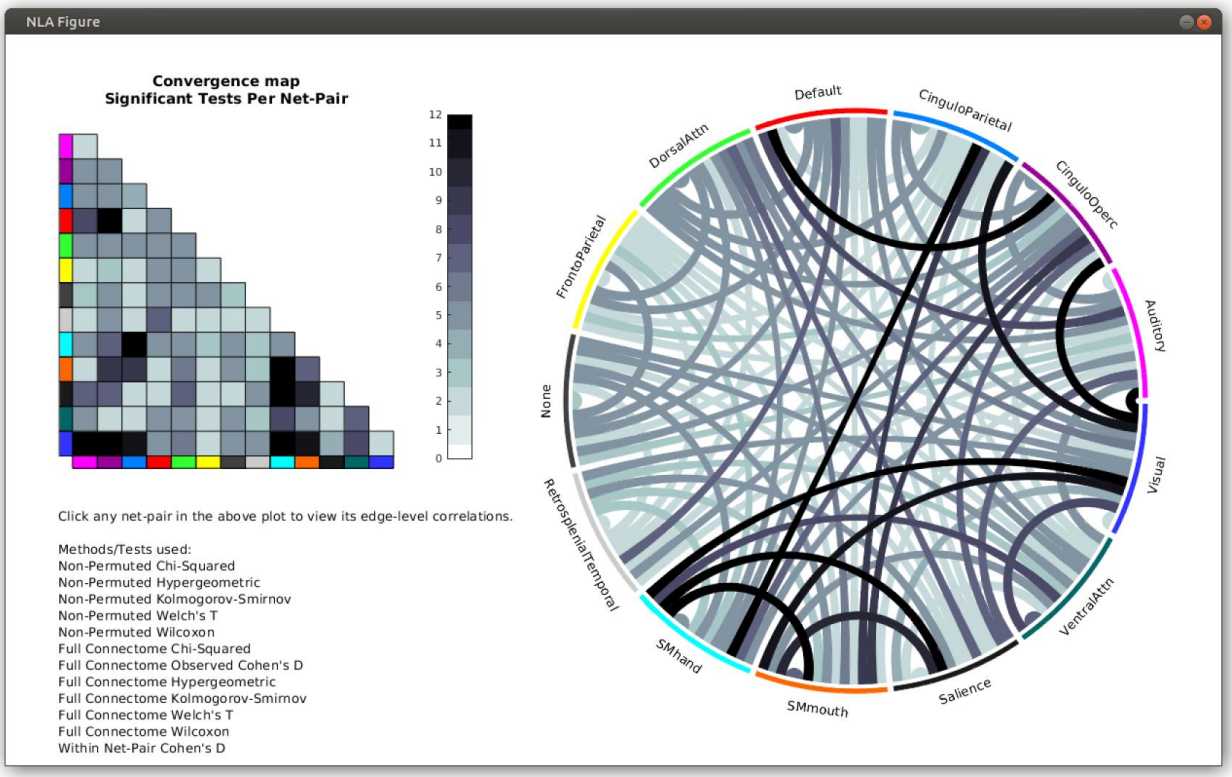
View edge-level chord plots

Display a triangular matrix of net-level P-values of the selected test(s), as well as a chord plot displaying edge-level P-values within significant networks (as determined by the displayed net-level P-value matrix).



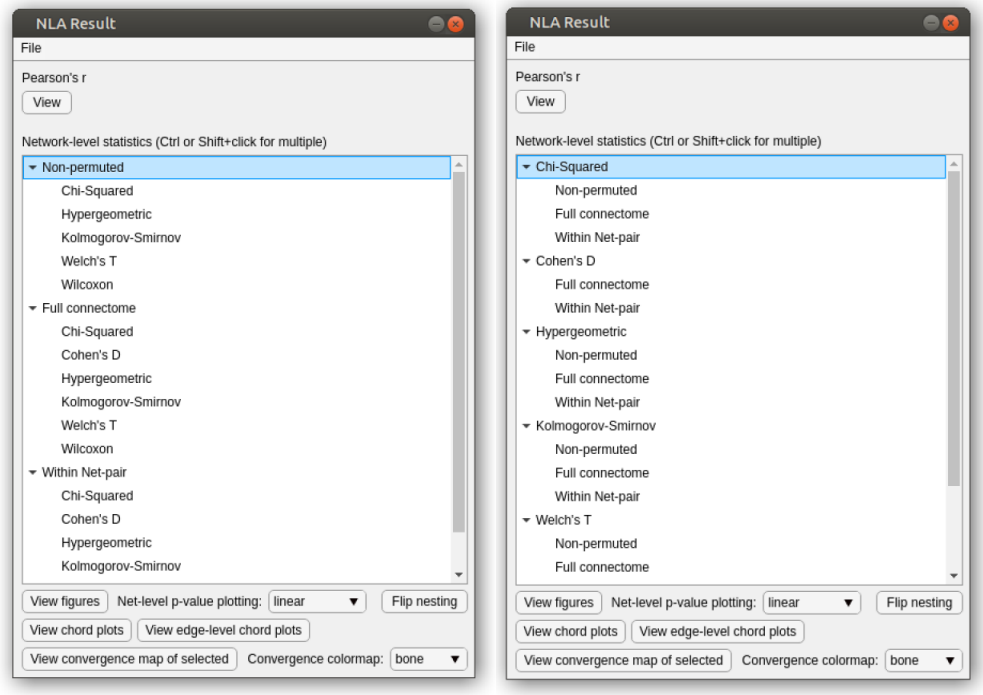
View convergence map

Display a triangular matrix and chord plot of convergence of the selected test(s). Convergence is calculated by counting the number of tests where each net-pair produced a significant P-value.



Flip nesting

To flip the nesting order from Method > Test to Test > Method (or vice versa), click the 'Flip nesting' button. This can make it easier to select results, depending on which you wish to view. For example, if you wish to view the results of all methods of the Chi-Squared test, flipping nesting and using Shift-click to select all methods under the Chi-squared test might be faster than Ctrl-clicking to the Chi-Squared result under each method.



Before and after flipping the nesting order

Save results

To save your result data, click the File > Save button. The resulting .mat file can either be loaded back in via NLA to view figures at a later date, or loaded into MATLAB proper via the **`load`** command to access exact data values. It can even be used by users who don't have the NLA toolbox themselves, although they will only be able to view the results as a MATLAB struct, and some data fields may be unintuitive. For example, data originally represented in a 'lower triangle matrix' class will be stored as a struct containing the compact representation of the data, and you will not be able to use the **`.asMatrix()`** function to view said struct as a matrix as you would with the original class. Still, the option is there.

Network Atlases

Overview

A network atlas is a data file describing networks of the brain, each containing a number of related regions of interest. It also contains metadata such as network colors and names, ROI spatial coordinates (with associated mesh/space), and optionally, a surface parcellation.

Structure

Network atlases are .mat files containing the following fields:

`ROI_key`	N_{ROIs} x 2 matrix, left column is ROI indexes and right column is the network they belong to.
`ROI_order`	N_{ROIs} x 1 vector, functional connectivity data indexes corresponding to each ROI.
`ROI_pos`	N_{ROIs} x 3 matrix, centroid positions of each ROI
`name`	The name of the atlas.
`net_colors`	N_{nets} x 3 matrix containing the color of each network.
`net_names`	N_{nets} x 3 matrix containing the name of each network.
`parcels`	Optional struct field for surface parcellations, containing two sub-fields, `ctx_l` and `ctx_r`, both N_{verts} x 1 vectors. Each element of the vectors corresponds to a vertex within the space mesh and contains the index of the ROI said vertex belongs to.
`space`	Which mesh the atlas' ROI locations (and parcels) are in. Currently the two options available are `'TT'` or `'MNI'`.

Network atlases can also contain other fields which aren't accessed programmatically, for example, some have a source field containing a link to the paper which they are derived from.

Provided network atlases

A number of network atlases are provided with the NLA software package in the `support_files` directory. Only NLA-specific details will be provided about them, if you wish to go into more depth on a particular atlas you should follow the link provided in its `source` field.

Gordon_13nets_333parcels_on_MNI

In surface space. Consists of 333 parcels and corresponding 13 networks.[6] Contains both MNI centroids and surface parcels on the provided `MNI_32k` mesh.

Gordon_12nets_286parcels_on_MNI

Same parcellation as before, with the 'None' network (and its constituent ROIs) removed.[6]

Seitzman_17nets_300ROI_on_TT

In volume space. Consists of 300 ROI and corresponding 17 networks.[48] Contains TT centroids.

Seitzman_15nets_288ROI_on_TT

In volume space. Same as before, with 12 ROI and 2 networks removed due to inconsistent placement in a network.[48]

Meshes/Cortex Anatomy

Overview

A mesh/Cortex Anatomy file describes the space ROIs are positioned in (or on, in the case of surface parcels), as a 3d mesh. Three meshes are provided by default in the NLA `support_files/meshes/` subfolder. By default, the GUI selects the relevant one based on the 'space' field of the selected network atlas – either `MNI_32k` for atlases with `space = 'MNI'`, or `Conte69_32k_on_TT` for atlases with `space = 'TT'`. In the future, the option to select other meshes within the GUI may be added. Within the pipeline script, the user can load any mesh file they prefer.

Structure

Meshes are `.mat` files containing the following fields:

`ctx_l`	The left hemisphere of the brain – a struct containing the following fields:
`nodes`	N_{verts} x 3 - X, Y, and Z coordinates of each surface vertex of this hemisphere of the brain.
`elements`	N_{faces} x 3 - indexes of the three nodes making up each surface face of this hemisphere of the brain. It can additionally contain other `*nodes` fields, for example, `Inodes`, `VInodes`, etc., to denote inflated or very inflated versions of the mesh. These are formatted similarly to nodes.
`ctx_r`	The right hemisphere of the brain, in the same format as the left one.
`space`	Unused now, but in future versions of NLA may be used within the GUI to automatically detect which of a set of meshes are compatible with a selected network atlas. Should be set to `'TT'`, `'MNI'`, or perhaps something else (to correspond to a user-provided network atlas in some other space)

Provided meshes

MNI_32k
MNI mesh with 32k vertices.

Conte69_32k_on_TT
Conte69 mesh in TT space, with 32k vertices.

Colin27_on_MNI
Colin27 mesh in MNI space.

Edge-level statistical tests

Methods

The non-permuted method calculates the correlation of each ROI to all other ROIs (all edges) via the given test, and stores this as a correlation coefficient (`coeff`), a p-value (`prob`), and a thresholded p-value (`prob_sig`).

The permuted method is identical except it also stores the correlation coefficients and thresholded p-values in `coeff_perm` and `prob_perm`.

Common inputs

P — Edge-level p-value threshold (aka. primary threshold)

Network atlas — See here

Functional connectivity — Initial correlation matrix of size N_{ROIs} x N_{ROIs} x $N_{subjects/scans}$ – r values or Fisher z-transformed r values.

Behavior format: Tab-separated text file (or MATLAB table, spreadsheet, etc) in the following

VarName	AnotherVarName	FurtherVarNames	…
1	5	1.4	
0	4	1.5	
1	3	6.5	
…($N_{subjects/scans}$)			

where each column header is the name of a variable or score, and each column contains $N_{subjects}$ entries. After loading this file into the GUI, the user marks one column as a 'Behavior' score of interest, and has the option to mark other column(s) as 'Covariates' which are partialled prior to running statistics.

Note: NLA cannot handle behavior scores or covariates with missing values. If a given column in your behavior file includes NaN's, 9999's, etc. as a substitute for missing values, said column should either not be selected as an behavior/covariate, or the subject/scan containing the missing value should be removed from the file prior to inputting to NLA.

Provided tests

Pearson's r

Uses the built-in MATLAB corr function with parameters `'type', 'Pearson'`.

Spearman's rho

Uses the built-in MATLAB corr function with parameters `'type', 'Spearman'`.

Spearman's rho estimator

Implements a faster approximation of the Spearman's rho function at the cost of producing slightly inaccurate output, as compared to the built-in Spearman's rho: Based upon our testing, edge-level rho values may differ by at most 1^{-4} and p-values by at most 1^{-5}. This error is passed on to network-level tests and can vary their resulting p-values by 1^{-4} (with 10,000 permutations). Lower permutation counts result in higher error being passed on, and in a less evenly distributed fashion. For example, running only 100 permutations can result in network-level p-value errors of up to 1^{-3}.

Based on these results, the Spearman's rho estimator is recommended for exploratory research due to its much better performance, and the Spearman's rho built-in is recommended for publication.

Kendall's tau-b

Implements the Kendall tau-b (allows tied values) function using C in a MEX file (`+mex/+src/kendallTauB.c`). This faster implementation produces identical tau and p-values to the MATLAB built-in Kendall's tau function while simplifying run-time complexity from the $O(n^2)$ to $O(n \log(n))$. The complexity reduction is enabled by using sorting and a balanced tree data structure (in this implementation, a red-black tree). A number of p-value equations from various sources were considered, and eventually one producing identical results to the MATLAB built-in (within double-precision rounding errors) was selected. This was done primarily for compatibility purposes, as it was somewhat less efficient than potential alternatives, and not necessarily any more valid.

Welch's t

Implements an optimized Welch's t-test comparing the functional connectivity of two groups. Has a few extra required inputs:

Group name(s): The names to associate with each group in a grouping variable (to be compared by the Welch's t-test). For example, 'Female' and 'Male' might be the names of the two groups.

Group val(s): The behavioral value associated with each group. For example, if our selected behavioral variable was 'F0M1', representing 'F' as a 0 and 'M' as a 1, we would set the grouping values accordingly to 0 and 1.

Pre-calculated data loader

Chosen by selecting 'Precalculated data' from the Edge-level test dropdown in the top left of the screen. Allows the user to load any observed and permuted edge-level data they have pre-calculated themselves. Instead of requiring functional connectivity and a behavioral vector, the precalculated data loader requires four .mat files containing matrices, which are input directly into the net-level tests. Note that the p-values should be thresholded, while coefficients can be of any type (specify the range of the coefficient you are using with the two coefficient range inputs).

Observed p — .mat file containing an N_{ROI_pairs} x 1 matrix of logical values (ones and zeros) - observed, thresholded edge-level p-values, where N_{ROI_pairs} are the lower triangle values of a N_{ROIs} x N_{ROIs} matrix.

Observed coeff — .mat file containing an N_{ROI_pairs} x 1 matrix of observed edge-level coefficients, where N_{ROI_pairs} are the lower triangle values of a N_{ROIs} x N_{ROIs} matrix.

Permuted p .mat file containing a N_{ROI_pairs} x N_{perms} matrix of logical values (ones and zeros) - permuted, thresholded edge-level p-values, where N_{ROI_pairs} is the lower triangle values of a N_{ROIs} x N_{ROIs} matrix, and N_{perms} is the number of permutations produced.

Permuted coeff .mat file containing a N_{ROI_pairs} x N_{perms} matrix of permuted edge-level coefficients, where N_{ROI_pairs} is the lower triangle values of a N_{ROIs} x N_{ROIs} matrix, and N_{perms} is the number of permutations produced.

Creating your own edge-level tests

To create your own edge-level tests, you must write a test class, as detailed below. It is also advised to look at how existing edge-level tests are implemented, as it may help you simplify your code or avoid pitfalls. Pay particular attention to the `updateResult()` function within `nla.edge.test.Base`, which you may find useful in simplifying the definition of your own custom edge-level tests.

Test objects

Your test object must inherit from `nla.edge.test.Base`, be located within the +`nla/+edge/+test` directory, and define certain members and methods.

Members and methods your object must define

Your test object must define the constant member `name` in order to be compatible with the test-running architecture. An example is given below:

```
properties (Constant)
    name = "Kendall's tau"
end
```

It must also implement the method:

```
result = run(obj, input_struct, previous_result)
```

where the inputs are defined as:

`input_struct` input structure, containing values provided from input fields (as detailed below in `requiredInputs()`)

`previous_result` either a logical false, in the non-permuted case (your run function should return a new test-specific `Result` object, with its non-permuted data members filled out), or a previous `Result` object, in the permuted case (your run function should modify the permuted data members of the `previous_result` object and return it as the result).

Your test object may also need to implement the static method:

```
inputs = requiredInputs()
```

which returns a cell array of `nla.inputField.*` objects which are required to run your test. For example, `nla.edge.test.Base` defines `requiredInputs` as follows:

```
methods (Static)
```

```
        function inputs = requiredInputs()
            import nla.* % required due to MATLAB package system quirks
            inputs = {inputField.Number('prob_max', 'P <', 0, 0.05, 1),
inputField.NetworkAtlas(), inputField.Behavior()};
        end
    end
```

which creates several input fields in the GUI –a floating-point number named `prob_max` ranging between 0 and 1, defaulting to 0.05, a network atlas input field, and a behavior input field. When the user selects a test derived from this within the GUI, they are prompted to fill out all input fields, and the values they provide will be accessible to your test's run function as members of the `input_struct` structure. For example, a test using the above definition of `requiredInputs` would be able to access the user-provided field `prob_max` within its `run()` function as `input_struct.prob_max`.

How to add your edge-level test to the GUI

In order to have your test appear within the GUI, you must place it within the `+nla/+edge/+test/` folder, and restart the GUI (The + sign folders are a MATLAB-ism which signifies that said folder is a namespace, for example, if your test was named MyTest, you would reference it as `nla.edge.test.MyTest`)

Result objects

In addition to implementing the test, you may also implement a result object to be returned by your test. If you choose to do so, your custom result object must inherit from `nla.edge.BaseResult`, as it contains a number of data members which must be filled by your code in order to provide network-level tests with necessary input data. It must also be placed within the `+nla/+edge/+result/` directory, and referenced within your code as a member of that namespace.

However, it may be sufficient to avoid creating a custom result object entirely and merely return an `nla.edge.result.Base` object if you don't need to save out custom data fields which aren't defined within that class.

Members and methods your object must define

Your result object may implement an output method, which produces test-specific figures when run:

```
    output(obj, net_atlas, flags)
```

`net_atlas` provided `nla.NetworkAtlas` object, defining networks, ROI locations, etc.

`flags:` Contains flags for the type of figures to output, currently the only one used is the Boolean value `display_sig`, to determine whether to threshold p-values for display.

It can also optionally implement a merge method, which merges a 'block' of result objects. This is necessary as, after permutation, we have a number of results (one per process), which we must somehow combine/reduce down to one per test. Typically this is an abstraction of the permutation process - for example, if your permutation code increments an integer by one for each permutation, the merge function might sum the values of that integer across all result objects, and if you append a value to the

end of a vector once per permutation, your merge function might concatenate said vectors together from all results, etc. You may also wish to run certain code in the merge function which is dependent on the entire distribution of result data, and is run only once – for example, this is where the within net-pair method is implemented in most tests.

```
merge(obj, results)
```

`results` Cell array of other result objects (from the same test) to merge with.

After calling the merge function on one result object, passing others in as the cell array results, the calling object should contain the reduction of itself and said other results. The contents of the other result objects can be ignored as only the calling object is preserved.

Network-level statistical tests

Methods

The non-permuted method measures how significant each network is compared to the entire connectome, via the given statistical test, and stores this as `prob`.

The full connectome method ranks the non-permuted significance of each network against the significance of the same network calculated over many permutations (using the same test), using this ranking to calculate the empirical probability of the network being as or more significant within the connectome than it was in the observed case, by chance (`perm_prob`). It also ranks each network's non-permuted significance against *all* network's permuted significance, and this is stored as `perm_prob_ew` (experiment-wide).

The within net-pair method measures how significant each network is compared to all permutations of *only* said network. It then Bonferroni corrects each network's significance by the number of networks in total, as by only comparing to one network's worth of permutations it is only sampling ~$1/N_{nets}$ of the total permuted data.

Common inputs

P	Network level p-value threshold
D	Network level Cohen's D threshold (for within net-pair method)
Behavior count	Number of different behavior vectors you intend to test. P-values are Bonferroni corrected by this number.

Provided tests

Hypergeometric

Implements a hypergeometric test.

Chi-squared

Implements a chi-squared test. Ranks networks by chi-squared values rather than probabilities. Outputs an additional `chi2` field containing non-permuted chi-squared values.

Kolmogorov-Smirnov

Uses the built-in MATLAB implementation of the Kolmogorov-Smirnov test, `kstest2`.

Wilcoxon

Uses the built-in MATLAB implementation of the Wilcoxon rank-sum test, `ranksum`.

Welch's t

Uses `welchT()`, an optimized implementation of `ttest2` with "`Vartype"` set to "`unequal"`.

Student's t

Uses the built-in MATLAB implementation of the Student's T-test, `ttest2`.

Cohen's d

Implements a Cohen's d test.

Creating your own network-level tests

To create your own network-level tests, you must write both test and result classes, as detailed below. It is also advised to look at how existing net-level tests are implemented, as it may help you simplify your code or avoid pitfalls.

Test objects

Your test object must inherit from `nla.net.test.Base`, be located within the `+nla/+net/+test` directory, and define certain members and methods.

Members and methods your object must define

Your test object must define the constant member `name` in order to be compatible with the test-running architecture. An example is given below:

```
properties (Constant)
    name = "Chi-squared"
end
```

It must also implement the method:

```
result = run(obj, input_struct, edge_result, net_atlas, previous_result)
```

where the inputs are defined as:

`input_struct`	input structure, containing values provided from input fields (as detailed below in `requiredInputs()`)
`edge_result`	the edge-level result we are running this network-level statistic on – at minimum you will be able to access the data members defined in `nla.edge.ResultBase`, but you might have access to additional test-specific data members.
`net_atlas`	provided `nla.NetworkAtlas` object, defining networks, ROI locations, etc.
`previous_result`	either a logical false, in the non-permuted case (your run function should return a new test-specific `Result` object, with its non-permuted data members filled out), or a previous `Result` object, in the permuted case (your run function should modify the permuted data members of the `previous_result` object and return it as the result).

Your test object may also need to implement the static method:

```
inputs = requiredInputs()
```

which returns a cell array of `nla.inputField.*` objects which are required to run your test.

For example, `nla.net.BaseTest` defines `requiredInputs` as follows:

```
   methods (Static)
       function inputs = requiredInputs()
           import nla.* % required due to MATLAB package system quirks
           inputs = {inputField.Integer('behavior_count', 'Behavior count:',
1, 1, Inf), inputField.Number('prob_max', 'P <', 0, 0.05, 1)};
       end
   end
```

which creates two input fields in the GUI – an integer named `behavior_count` ranging between 1 and infinity, defaulting to 1, and a floating-point number named `prob_max` ranging between 0 and 1, defaulting to 0.05. When the user selects the test within the GUI, they are prompted to fill out both input fields, and the values they provide will be accessible to your test's run function as members of the `input_struct` structure. For example, a test using the above definition of `requiredInputs` would be able to access the user-provided field `prob_max` within its `run()` function as `input_struct.prob_max`.

How to add your network-level test to the GUI

In order to have your test appear within the GUI, you must place it within the `+nla/+net/+test/` folder, and restart the GUI (The + sign folders are a MATLAB-ism which signifies that said folder is a namespace, for example, if your test was named MyTest, you would reference it as `nla.net.test.MyTest`)

Result objects

In addition to implementing the test, you must also implement a result object to be returned by your test. Your result object must inherit from `nla.TestResult` or one of its children such as `nla.net.BasePermResult` or `nla.net.BaseResult` - read the classes to see which fits your needs best, and note that inheriting from certain classes may require you to define additional member variables such as `name`, `name_formatted`, etc. Your result class must additionally define the following members and methods, and should be placed within the `+nla/+net/+result/` directory.

Members and methods your object must define

Your result object must implement an output method, which produces test-specific figures when run:

```
   output(obj, input_struct, net_atlas, flags)
```

`input_struct`	input structure, containing values provided from input fields.
`net_atlas`	provided `nla.NetworkAtlas` object, defining networks, ROI locations, etc.
`flags:`	contains flags for the type of figures to output, some typical ones are the Boolean values: `show_nonpermuted`, `show_full_conn`, and `show_within_net_pair`, to determine which figures to output

It must also implement a merge method, which merges a 'block' of result objects. This is necessary as, after permutation, we have a number of results (one per process), which we must somehow

combine/reduce down to one per test. Typically this is an abstraction of the permutation process - for example, if your permutation code increments an integer by one for each permutation, the merge function might sum the values of that integer across all result objects, and if you append a value to the end of a vector once per permutation, your merge function might concatenate said vectors together from all results, etc. You may also wish to run certain code in the merge function which is dependent on the entire distribution of result data, and is run only once – for example, this is where the within net-pair method is implemented in most tests.

```
   merge(obj, input_struct, edge_result_nonperm, edge_result, net_atlas,
results)
```

`input_struct`	input structure, containing values provided from input fields.
`edge_result_nonperm`	the non-permuted edge-level result.
`edge_result`	the permuted edge-level result.
`net_atlas`	provided `nla.NetworkAtlas` object, defining networks, ROI locations, etc.
`results`	cell array of other result objects to merge with.

After calling the merge function on one result object, passing others in as the cell array results, the calling object should contain the reduction of itself and said other results. The contents of the other result objects can be ignored as only the calling object is preserved.

Advanced Settings

Customizing Parallel Pool

For computationally expensive parallel jobs, you may want to maximize the number of workers and/or threads created in your worker pool created by MATLAB.

NOTE: The number of workers and threads you are able to include in a pool will be limited by the number of available threads / cores supported by your machine. The limit is that (NumWorkers * NumThreadsPerWorker) < (Threads available on your machine)

Option 1: Start a local pool with a custom number of workers

To generate a pool that has a custom number of workers, first shut down any currently running parallel pool. Do this by clicking the parallel icon at the bottom right of your MATLAB session (looks like 4 vertical bars that are blue or gray). If any parallel pool exists in your current session, the option 'Shut Down Parallel Pool' will be selectable. Do this to close any current pool you have.

Then, generate a new pool with a custom number of workers. Do this with the command

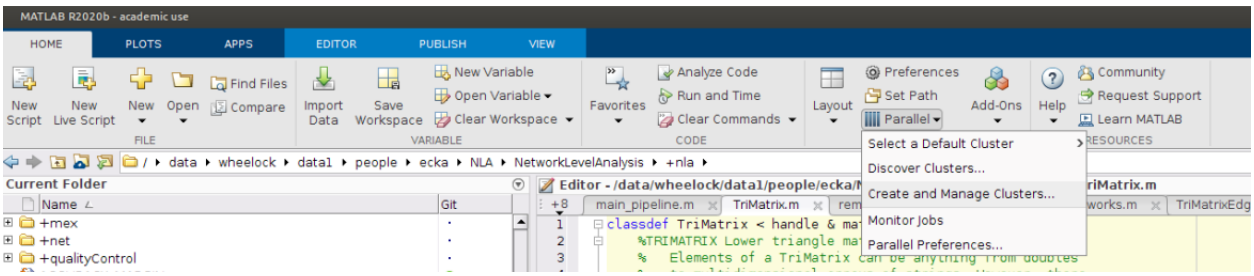
```
p = parpool('local', numWorkers);
```

where '*numWorkers*' is the number of workers you want in the new pool.

Now when you run the NLA GUI, it will use the newly created cluster to run permuted jobs

Option 2: Create a new cluster profile in MATLAB

In MATLAB, select the "Home" tab at the top of the application. In the 'Environment' section of the Home tab, select the 'Parallel' dropdown, and select 'Create and Manage Clusters…'



In the ‘Cluster Profile Manager’ that pops up, you can create a custom cluster profile that uses more than the default number of workers or threads. Click the ‘Add Cluster Profile’ dropdown, and select the “Local” option.

Configure customs settings for the new cluster profile

To name the new profile, select the newly created profile, right click, and rename it to enter an appropriate name.

To change the number of workers for the profile, right click the profile and select ‘Edit’. You can enter the desired number of workers for the pool to reserve in the textbox to the right of the label ‘NumWorkers’.

You can also change the number of threads per worker in the NumThreads textbox. NumThreads will improve the performance of individual workers by allowing them to use MATLAB’s built-in parallelism for its matrix operations. For many permutations, it is generally preferred to increase NumWorkers rather than NumThreads. The only likely reason one should consider increasing the number of threads per worker is if the size of your dataset means your machine will be memory-limited rather than core-limited (each worker uses its copy of the input data, so there is memory overhead, and processing overhead to copy the data to each worker).

Variable Name Glossary

When opening NLA result data structures in MATLAB, one will encounter a number of different data fields, the meaning of some which may not be immediately obvious. This glossary serves as a reference for many of the variable names/fields making up result objects.

`avg_prob_sig`	prob_sig averaged over all edges (see `prob_sig`)
`coeff`	Fisher transformed correlation coefficient (r-value. rho-value, etc.)
`coeff_name`	display name of correlation coefficient.
`empirical_fdr`	empirical false-discovery rate calculated from `perm_prob_hist`.
`perm_count`	number of permutations performed on the given result
`perm_prob`	empirical probability of the original network being more significant than permutations of said network.
`perm_prob_hist`	histogram of permuted probabilities.
`perm_prob_ew`	empirical probability of the original network being more significant than the permuted connectome.
`perm_rank`	the rank of the original network's p-values among permutations of said network.
`perm_rank_ew`	the rank of the original network's p-values among permutations of the entire connectome – used in full connectome method.
`prob`	non-permuted significance.
`prob_max`	p-value threshold.
`prob_max_ew`	experiment-wide p-value threshold.
`prob_sig`	p-values, thresholded at prob_max.
`ss_prob, ss_ks`	single-sample p-values and test statistics
`within_np_d`	within net-pair Cohen's D.
`within_np_prob`	within net-pair p-values.

References

1. Rivals, I., Personnaz, L., Taing, L. & Potier, M.-C. Enrichment or depletion of a GO category within a class of genes: which test? *Bioinformatics* **23**, 401–407 (2007).
2. Khatri, P., Sirota, M. & Butte, A. J. Ten Years of Pathway Analysis: Current Approaches and Outstanding Challenges. *PLoS Comput Biol* **8, e1002375**, (2012).
3. Backes, C. Systematic permutation testing in GWAS pathway analyses: identification of genetic networks in dilated cardiomyopathy and ulcerative colitis. *BMC Genomics* **15**, 622 (2014).
4. Subramanian, A. Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. *Proc Natl Acad Sci* **102**, 15545–15550 (2005).
5. Sporns, O., Tononi, G. & Kötter, R. The Human Connectome: A Structural Description of the Human Brain. *PLoS Comput Biol* **1**, 42 (2005).
6. Gordon, E. M. Generation and Evaluation of a Cortical Area Parcellation from Resting-State Correlations. *Cereb Cortex* **26**, 288–303 (2016).
7. Power, J. D. Functional Network Organization of the Human Brain. *Neuron* **72**, 665–678 (2011).
8. Gratton, C. Functional Brain Networks Are Dominated by Stable Group and Individual Factors, Not Cognitive or Daily Variation. *Neuron* **98**, 439-452 5 (2018).
9. Forman, S. D. Improved Assessment of Significant Activation in Functional Magnetic Resonance Imaging (fMRI): Use of a Cluster-Size Threshold. *Magn Reson Med* **33**, 636–647 (1995).
10. Friston, K. J., Worsley, K. J., Frackowiak, R. S. J., Mazziotta, J. C. & Evans, A. C. Assessing the significance of focal activations using their spatial extent: Assessing Focal Activations by Spatial Extent. *Hum Brain Mapp* **1**, 210–220 (1994).

11. Vieira, S., Pinaya, W. H. L. & Mechelli, A. Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications. *Neurosci Biobehav Rev* **74**, 58–75 (2017).

12. Greene, D. J. Multivariate pattern classification of pediatric Tourette syndrome using functional connectivity MRI. *Dev Sci* **19**, 581–598 (2016).

13. Shirer, W. R., Ryali, S., Rykhlevskaia, E., Menon, V. & Greicius, M. D. Decoding Subject-Driven Cognitive States with Whole-Brain Connectivity Patterns. *Cereb Cortex* **22**, 158–165 (2012).

14. Eggebrecht, A. T. Joint Attention and Brain Functional Connectivity in Infants and Toddlers. *Cereb Cortex* **27**, 1709–1720 (2017).

15. Wheelock, M. D. Altered functional network connectivity relates to motor development in children born very preterm. *NeuroImage* **183**, 574–583 (2018).

16. Wheelock, M. D. Sex differences in functional connectivity during fetal brain development. *Dev Cogn Neurosci* **36**, 100632 (2019).

17. Rudolph, M. D. Maternal IL-6 during pregnancy can be estimated from newborn brain connectivity and predicts future working memory in offspring. *Nat Neurosci* **21**, 765–772 (2018).

18. Wheelock, M. D. Altered brain-behavior relationships underlie attention impairments in very preterm children. *Cereb Cortex* **31**, 1383–1394 (2021).

19. Maron-Katz, A., Vaisvaser, S., Lin, T., Hendler, T. & Shamir, R. A large-scale perspective on stress-induced alterations in resting-state networks. *Sci Rep* **6**, 21503 (2016).

20. Marrus, N. Walking, Gross Motor Development, and Brain Functional Connectivity in Infants and Toddlers. *Cereb Cortex* **28**, 750–763 (2018).

21. Feczko, E. Subtyping cognitive profiles in Autism Spectrum Disorder using a Functional Random Forest algorithm. *NeuroImage* **172**, 674–688 (2018).

22. Thomason, M. E. Prenatal lead exposure impacts cross-hemispheric and long-range connectivity in the human fetal brain. *NeuroImage* **191**, 186–192 (2019).

23. McKinnon, C. J. Restricted and Repetitive Behavior and Brain Functional Connectivity in Infants at Risk for Developing Autism Spectrum Disorder. *Biol Psychiatry Cogn Neurosci Neuroimaging* **4**, 50–61 (2019).

24. Marek, S. Identifying reproducible individual differences in childhood functional brain networks: An ABCD study. *Dev Cogn Neurosci* **40**, 100706 (2019).

25. Shehzad, Z. A multivariate distance-based analytic framework for connectome-wide association studies. *NeuroImage* **93**, 74–94 (2014).

26. Sharma, A. Common Dimensional Reward Deficits Across Mood and Psychotic Disorders: A Connectome-Wide Association Study. *Am J Psychiatry* **174**, 657–666 (2017).

27. Raichle, M. E. The Brain's Default Mode Network. *Annu Rev Neurosci* **38**, 433–447 (2015).

28. Zalesky, A., Fornito, A. & Bullmore, E. T. Network-based statistic: Identifying differences in brain networks. *NeuroImage* **53**, 1197–1207 (2010).

29. Rubinov, M. & Sporns, O. Complex network measures of brain connectivity: Uses and interpretations. *NeuroImage* **52**, 1059–1069 (2010).

30. Bahrami, M., Laurienti, P. J. & Simpson, S. L. A MATLAB toolbox for multivariate analysis of brain networks. *Hum Brain Mapp* **40**, 175–186 (2019).

31. Ginestet, C. E., Fournel, A. P. & Simmons, A. Statistical network analysis for functional MRI: summary networks and group comparisons. *Front Comput Neurosci* **8**, (2014).

32. Simpson, S. L., Lyday, R. G., Hayasaka, S., Marsh, A. P. & Laurienti, P. J. A permutation testing framework to compare groups of brain networks. *Front Comput Neurosci* **7**, (2013).

33. Nichols, T. & Hayasaka, S. Controlling the familywise error rate in functional neuroimaging: a comparative review. *Stat Methods Med Res* **12**, 419–446 (2003).

34. Bellec, P. Impact of the resolution of brain parcels on connectome-wide association studies in fMRI. *NeuroImage* **123**, 212–228 (2015).

35. Rosvall, M. & Bergstrom, C. T. Maps of random walks on complex networks reveal community structure. *Proc Natl Acad Sci* **105**, 1118–1123 (2008).

36. Thomas Yeo, B. T. The organization of the human cerebral cortex estimated by intrinsic functional connectivity. *J Neurophysiol* **106**, 1125–1165 (2011).

37. Glasser, M. F. A multi-modal parcellation of human cerebral cortex. *Nature* **536**, 171–178 (2016).

38. Shen, X., Tokoglu, F., Papademetris, X. & Constable, R. T. Groupwise whole-brain parcellation from resting-state fMRI data for network node identification. *NeuroImage* **82**, 403–415 (2013).

39. Craddock, R. C., James, G. A., Holtzheimer, P. E., Hu, X. P. & Mayberg, H. S. A whole brain fMRI atlas generated via spatially constrained spectral clustering. *Hum Brain Mapp* **33**, 1914–1928 (2012).

40. Ackermann, M. & Strimmer, K. A general modular framework for gene set enrichment analysis. *BMC Bioinformatics* **10**, 47 (2009).

41. van den Heuvel, M. P. Proportional thresholding in resting-state fMRI functional connectivity networks and consequences for patient-control connectome studies: Issues and recommendations. *NeuroImage* **152**, 437–449 (2017).

42. Mootha, V. K. PGC-1α-responsive genes involved in oxidative phosphorylation are coordinately downregulated in human diabetes. *Nat Genet* **34**, 267–273 (2003).

43. Zahn, J. M. Transcriptional Profiling of Aging in Human Muscle Reveals a Common Aging Signature. *PLoS Genet* **2**, 115 (2006).

44. Chen, J. J., Lee, T., Delongchamp, R. R., Chen, T. & Tsai, C.-A. Significance analysis of groups of genes in expression profiling studies. *Bioinformatics* **23**, 2104–2112 (2007).

45. Newton, M. A., Quintana, F. A., Boon, J. A., Sengupta, S. & Ahlquist, P. Random-set methods identify distinct aspects of the enrichment signal in gene-set analysis. *Ann Appl Stat* **1**, (2007).

46. Yaari, G., Bolen, C. R., Thakar, J. & Kleinstein, S. H. Quantitative set analysis for gene expression: a method to quantify gene set differential expression including gene-gene correlations. *Nucleic Acids Res* **41**, 170–170 (2013).

47. Efron, B. & Tibshirani, R. On testing the significance of sets of genes. *Ann Appl Stat* **1**, (2007).

48. Seitzman, B. A. A set of functionally-defined brain regions with improved representation of the subcortex and cerebellum. *NeuroImage* **206**, 116290 (2020).