**Resilience code documentation**

*For the analysis of networks for their robustness to failures as standalone networks and as systems where one network is dependent on another.*

Craig Robson

Newcastle University

March 2014

Contents

[1 Summary 2](#_Toc381014661)

[1.1 Utilisation once imported 2](#_Toc381014662)

[1.2 Dependencies 2](#_Toc381014663)

[2 Examples 4](#_Toc381014664)

[3 Module 5](#_Toc381014665)

[3.1 Classes – Error Handling 5](#_Toc381014666)

[3.2 Functions 5](#_Toc381014667)

[3.3 \_\_main\_\_ 9](#_Toc381014668)

[4 Key function diagrams 10](#_Toc381014669)

[4.1 main function 10](#_Toc381014670)

[4.2 step function 11](#_Toc381014671)

[4.3 analysis\_B function 12](#_Toc381014672)

# Summary

The code can be used standalone, or imported and as a module. This has been built with python 2.7 and utilises networkx, a python package the analysis and manipulation of complex networks. The module allows the behaviour of a network to be analysed against perturbations, performed through the removal of nodes from the network, for which there are multiple options available within this module. The module allows the analysis of a single network, or a pair of networks where one is dependent on the other, or where both are dependent on the each other. The behaviour of the network(s) can be mapped through the use of a multitude of metrics which can be selected by the user (though a default set will always be used on top of these as well). The module is also compatible with the nx\_pgnet module, another project also be undertaken. This allows the user to store networks in PostgreSQL/PostGIS database and analyse them as they wish, with the knowledge that the network will be stored in a safe and secure environment.

## Utilisation once imported

Once imported a failure simulation can be run through calling the main function. This will run the analysis from start to finish, saving the results to a text file upon completion.

Alternatively, the step function can be used. This removes the next node from the network and thus actually performs the analysis. A diagram of the process which this module handles can be found in section 4.2.

For more detail on these functions see 3.2.

## Dependencies

This module is reliant on a number of other modules, all of which are standard with most python installations, with the exception of networkx, which can be installed from here <http://networkx.github.io/download.html>. Extra options/connectivity is possible also through the use of the nx\_pgnet module (which also is dependent on a number of other modules/applications, see the documentation for further details).

Required:

* networkx
* sys
* random
* time
* datetime

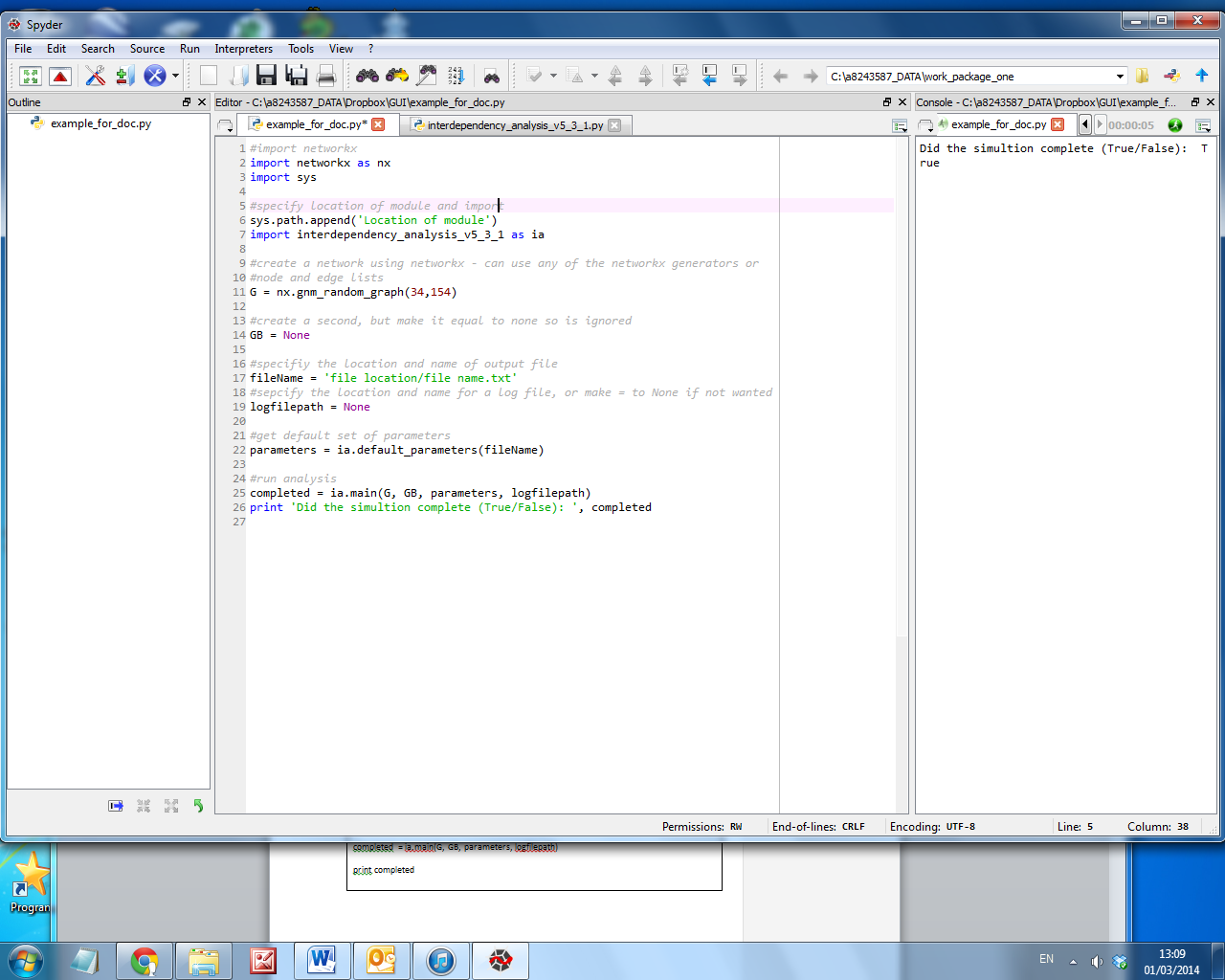
Optional:

* nx\_pgnet

# Examples

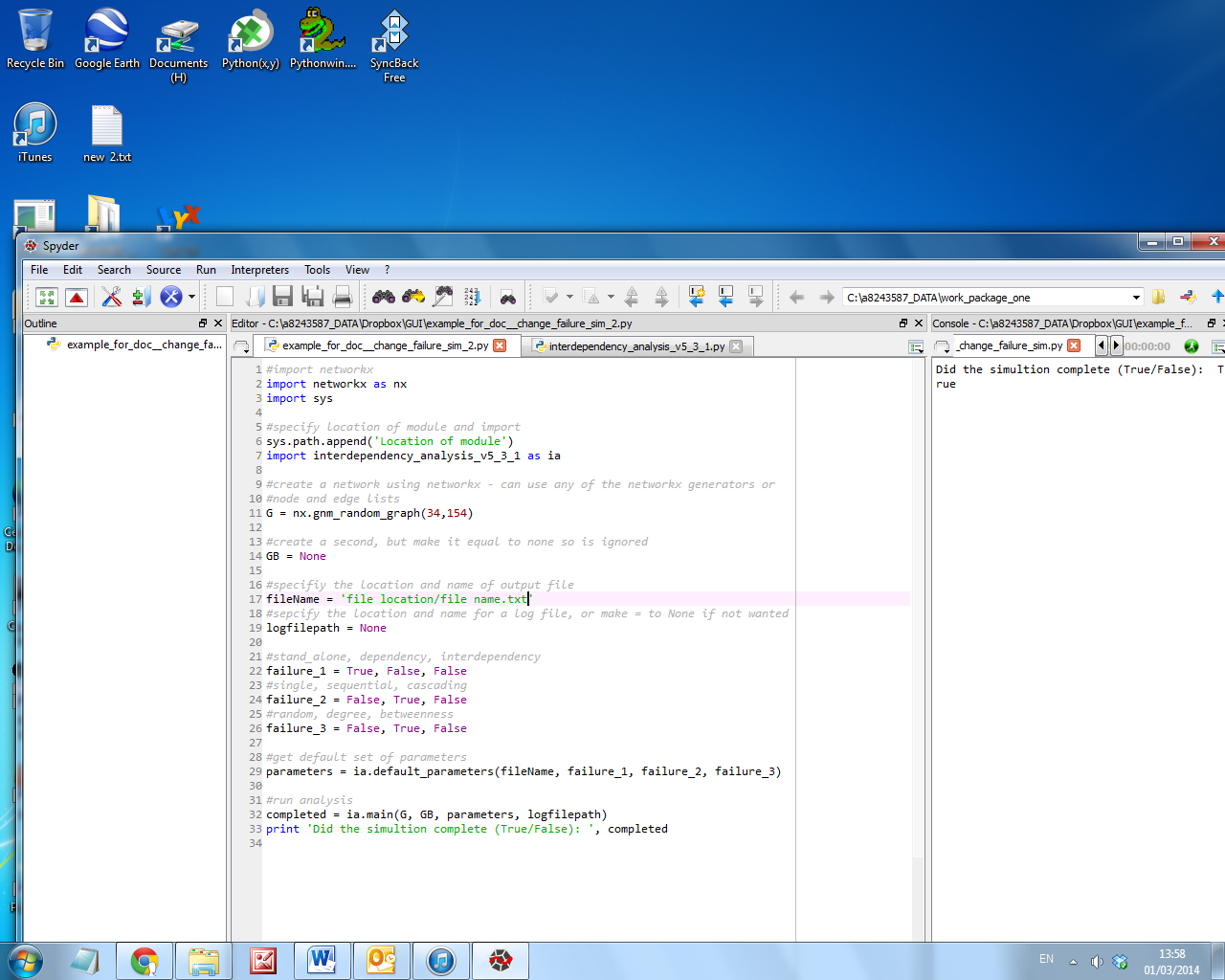
## Quick analysis of a network

The screen shot below shows a simple script which generates a network using networkx then using the default settings runs a failure simulation over the created network. The results are then output to a text file. (The default settings run a sequential random failure model on a single network with one simulation. It also uses only the basic set of metrics for recording the behaviour of the network.)



## Changing the simulation failure model

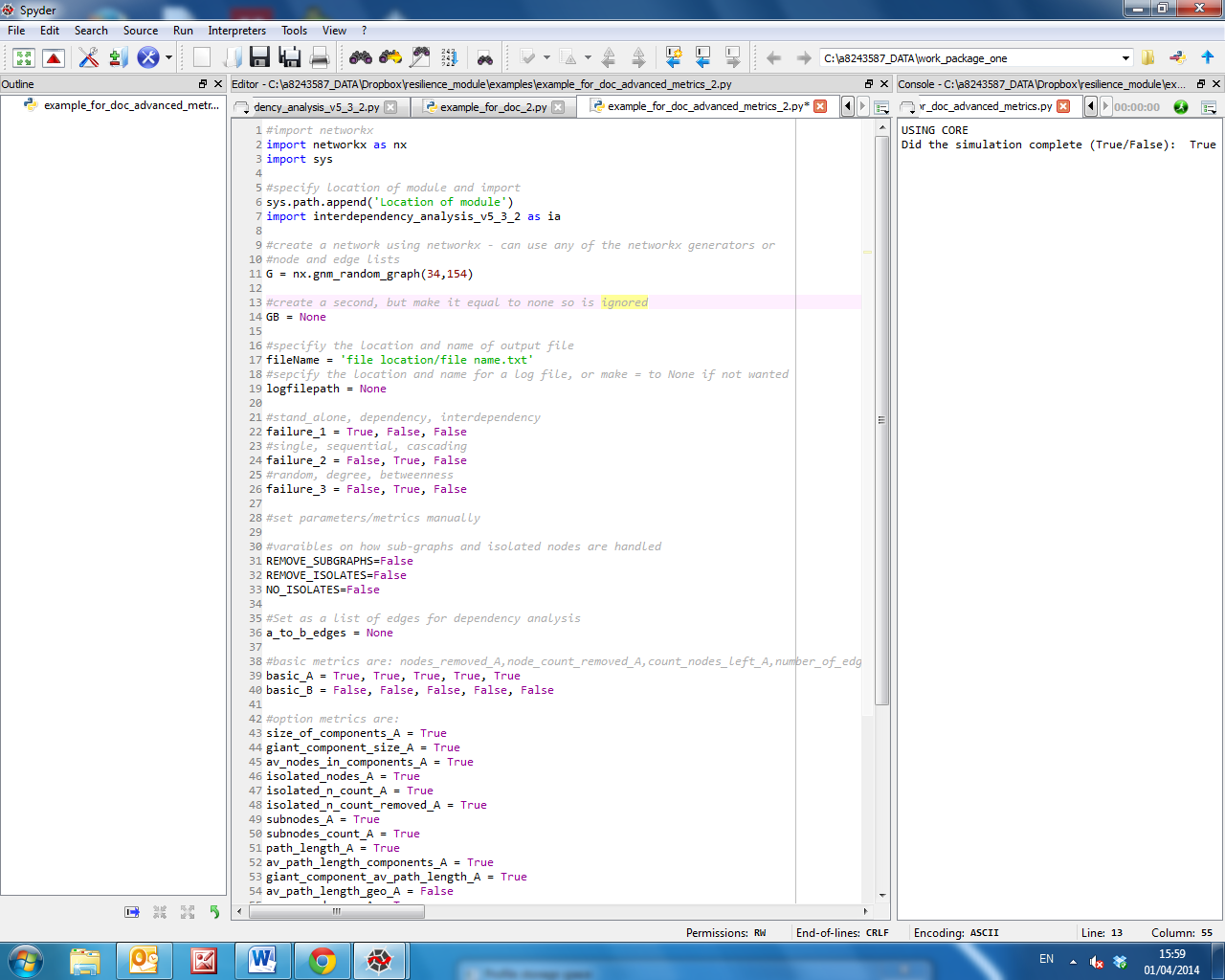
To change the type of failure model which is run over the network, a number of variables can be changed.

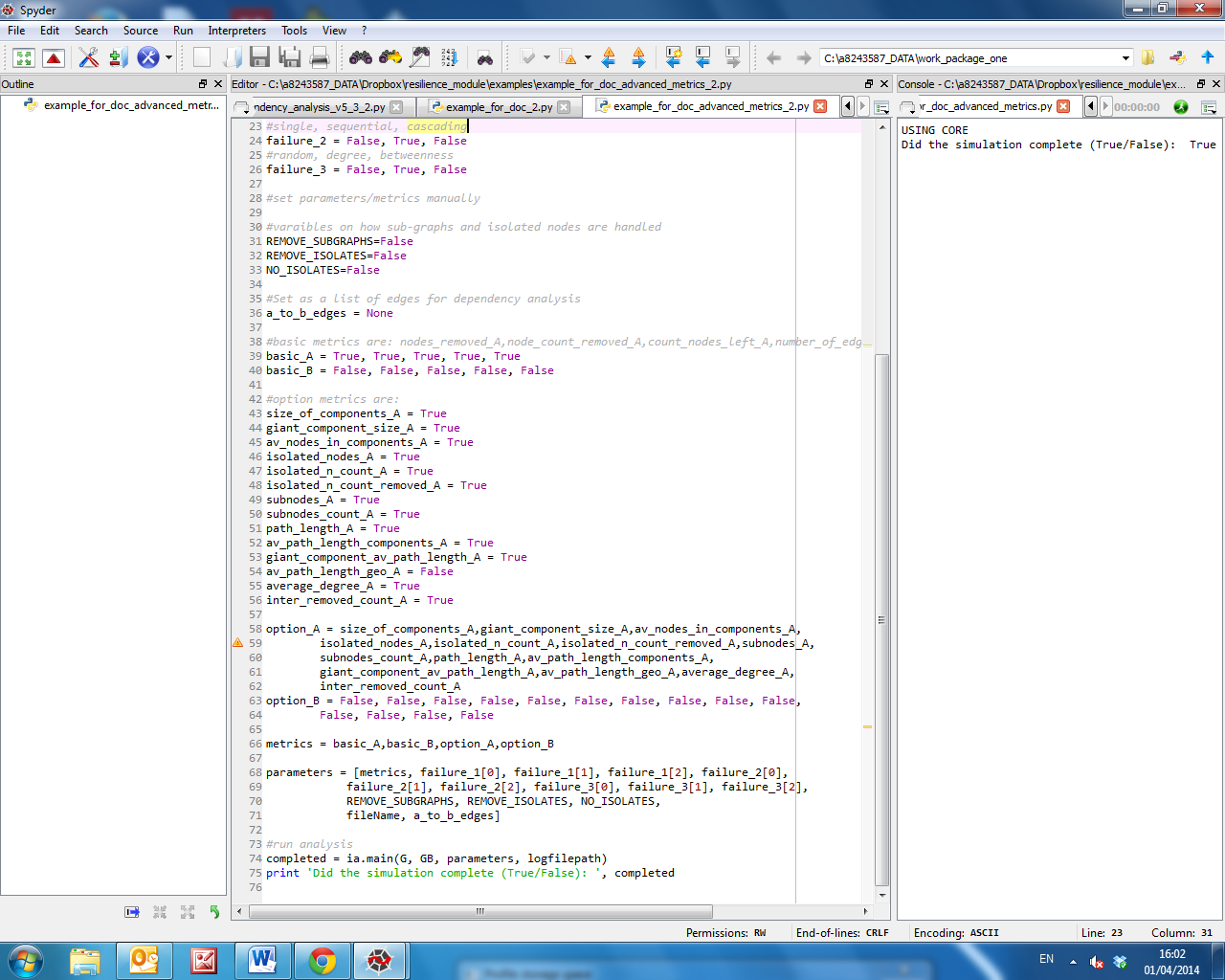


The introduction of the three variables, lines 21 to 26 in the above image, allows for any of the available failure models to be run. To change the type of analysis, simply set the appropriate item to True, making sure all others are set to False. For example, the above setting will run, for a single network (STAND ALONE = True (failure\_1)), a sequential failure model (SEQUENTIAL = True (failure\_2)), where the node to remove at each time step is selected using the degree based method (DEGREE = True (failure\_3)).

## Adding more metrics to the results

Again, this is dependent upon the setting of appropriate variables, as can be seen in the example below. The major changes are made from line 30 onwards, where all the parameters for the analysis need to be defined (previously done in the default function used in the earlier examples). The metrics are split into two categories; basic and option. The basic set are those which will always be computed as they are either needed for the analysis to work or are deemed fundamental to the interpretation of the results. The option set, which has 14 optional metrics, are those which add detail to the results, and thus can be set as either True or False (True meaning it will be computed at every time step). These need to be stated otherwise an error will be returned. The metrics can be set for each network when working with two, allowing for a flexible analysis approach.





# Module

Structure:

Main module -

## Classes – Error Handling

|  |
| --- |
| GeneralError(*Exception*)  Non-descript error handler. |

|  |
| --- |
| GraphError(*Exception*)  Error relating to a network not being suitable to compute the reselected function on. |

|  |
| --- |
| OutputError(*Exception*)  Error in the processes related to the outputting the results. |

|  |
| --- |
| CalculationError(*Exception*)  When a calculation cannot be performed for some reason. |

|  |
| --- |
| SearchError(*Exception*)  Handles errors associated to not being able to find the request node or edge in the network. |

|  |
| --- |
| WriteError(*Exception*)  Could not open or read or write to the text file. |

## Functions

|  |
| --- |
| write\_to\_log\_file(*logfileapth, text)*  Writes to a file given a file path and a string of text. Also adds the time to every entry. If the filepath is set to *None(default),* nothing will be written. |

|  |
| --- |
| max\_val\_random*(list)*  Returns the maximum value in the list and the position in the list (or the node if applicable). If the maximum value is shared between more than one node (list entry), one of those is picked at random. |

|  |
| --- |
| sequential\_degree(*G, INTERDEPENDENCY*)  Calculates the degree of all nodes in *G*, and then removes the node with the greatest value and its edges. Returns *G* and the node removed. |

|  |
| --- |
| sequential\_betweenness(*G, INTERDEPENDENCY*)  Calculates the betweenness centrality of all nodes, removes the one with the highest value and it’s the edges. Returns *G* and the node removed. |

|  |
| --- |
| sequential\_random(*G, NO\_ISOLATES, INTERDEPENDENCY)*  Picks a node at random from the network, removes it and its edges. Returns *G* and the node removed. |

|  |
| --- |
| cascading\_failure(*G, dlist, dead,k,subnodes\_A, isolated\_nodes\_A, removed\_nodes, INTERDEPENDENCY*)  Facilitates the running of the cascading failure model. Removes the nodes of those which were connected to the node(s) removed on the previous time step (*dlist*). Returns *G, dlist* (updates)*, removed\_nodes, node.* |

|  |
| --- |
| single\_random(*G,node\_list, INTERDEPENDENCY*)  Selects a node at random from *node\_list* and removes it from *G* along with its edges. Removes it from node\_list as well, which contains the nodes which have still not been removed. Returns an edited network and the node removed. |

|  |
| --- |
| handle\_isolates(*G*)  Removes any isolated nodes from the network (*G*). Returns the network (*G*) and a list of isolated nodes. |

|  |
| --- |
| remove\_isolates(*Gtemp,node\_list,isolated\_nodes,isolated\_n\_count\_removed,node\_count\_removed,to\_b\_nodes,from\_a\_nodes*)  Handles the removal of isolated nodes, adjusting the associated lists including the *node\_list*. Returns all those received with any edits required. |

|  |
| --- |
| remove\_edges(*G,nde,INTERDEPENDENCY*)  Remove the edges from the network (*G*) which have the node (*nde*) as the start or end point. Returns the network *G* |

|  |
| --- |
| clean\_node\_lists(*subn,node\_list, to\_b\_nodes, from\_a\_nodes)*  Clean the suite of node lists. Removes from the lists those nodes removed from the network due to being part of subgraphs. Returns the corrected lists. |

|  |
| --- |
| handle\_sub\_graphs(*nodelists, edgelists*)  Removes subgraphs form the network if required. The network is built from the node and edge lists before doing so. Returns the network, a list and count of the nodes removed and the new node and edge lists. |

|  |
| --- |
| check\_dependency\_edges(*networks,node,basic\_metrics\_A,basic\_metrics\_B,option\_metrics\_A,option\_metrics\_B,to\_b\_nodes,from\_a\_nodes,temp*)  Checks the dependency edges to see if any of them have been broken by the removal of nodes from the network. |

|  |
| --- |
| check\_node\_removed(*node, subnodes, isolated\_nodes*)  Checks if a node has been removed from the network before trying to remove it. Returns boolean variable. |

|  |
| --- |
| whole\_graph\_av\_path\_length(*Gtemp*)  Calculates the average path length over the entire network, including all subgraphs. Returns a single variable containing the result. |

|  |
| --- |
| calc\_initial\_values(*Gtemp, basic\_metrics, option\_metrics*)  Calculates the initial values for the metrics which are not set as False. Returns two containers with all the metrics in. |

|  |
| --- |
| core\_analysis(*GnetA, GnetB, parameters*)  Creates all the data containers for the metrics where the metric is not set as False. Returns a single variable containing all the metric containers. |

|  |
| --- |
| step(*graphparameters,parameters,iterate,logfilepath*)  Performs a single step of the analysis of the network(s), including the selection and removal of the nose(s) and associated edge(s) to the computation of the metrics. Returns two variables; the first containing the parameters and metrics, the second a boolean vale indicating the need to perform another iteration or not. |

|  |
| --- |
| analysis\_A(*networks, basic\_metrcs\_A, basic\_metrics\_B, optional\_metrics\_A, optional\_metrics\_B,i,node,to\_b\_nodes, from\_a\_nodes,temp*)  NOT USED |

|  |
| --- |
| analysis\_B(*parameters,iterate,Gtemp,i,to\_a\_nodes,from\_b\_nodes,node\_list,basic\_metrics,option\_metrics,to\_b\_nodes, from\_a\_nodes)*  Controls the removal of subnodes and isolated nodes (if either are set as True as parameters), and computes the required metric values. Returns all those received. |

|  |
| --- |
| outputresults(*graphparameters, parameters,logfilepath=None,multiiterations=None*)  Control function for the output of the results. Returns the values of those metrics written to the text file. |

|  |
| --- |
| average\_txtresults(*graphparameters, parameters,error*)  Reads in the text file with the results from the simulations and collates all the results together to produce an average for each metric at each time step. Returns the everages for each metric used. |

|  |
| --- |
| results(*basic\_metrics\_A, basic\_metrics\_B*)  USED IN TESTING ONLY. Prints the results into the console window. |

|  |
| --- |
| write\_text\_file(*outputfile,CASCADING,basic\_metrics,option\_metrics*)  Write the metric values to the results text file. |

|  |
| --- |
| txtout(*outputfile,graphparameters, parameters*)  Write the parameters for the simulation(s) to the results text file. |

|  |
| --- |
| replace\_all(*text, dic*)  Given a string and a dict of characters, will where found, replace them. Returns a string. |

|  |
| --- |
| main(*GA,GB,parameters,logfilepath,viewfailure = False*)  Controls the running of the analysis. Returns a single boolean variable on if the analysis was completed. |

|  |
| --- |
| get\_nodes\_edges\_csv(*location*)  Reads a csv file where the edges are listed on the top line and the nodes in the second line. Creates a list of nodes and edges which can be used to build a network from. |

|  |
| --- |
| failure\_type(*SINGLE, SEQUENTIAL, CASCADING, RANDOM, DEGREE, BETWEENNESS*)  Creates string which summarises the analysis being undertaken. |

|  |
| --- |
| analyse\_existing\_networks(*NETWORK\_NAME, conn, db, parameters, noioa, use\_db, use\_csv, logfilepath*)  For the analysis of a single network can get a network from a csv or from the database schema. For dependency analysis get only from a csv. |

## \_\_main\_\_

Where parameters are set when run independently (not as a module imported). All of the variables listed in 3.4 can be set here and facilitate a full analysis range. Of course this can also be achieved when it is used as a module.

## Variables

### Analysis

|  |  |  |
| --- | --- | --- |
| **Variable** |  | **Description** |
| Failure type  *(Only one should be set as True, all others should be set as False).* | STAND\_ALONE (*boolean*) | Analysis of a single network |
| DEPENDENCY (*boolean*) | Analysis of a pair of networks where B is dependent upon A. |
| INTERDEPENDENCY (*boolean*) | Analysis of a pair of networks where A is dependent upon B, and B is dependent upon A. |
| Failure method  *(Only one should be set as True, all others should be set as False).* | SINGLE (*boolean*) | Each node is removed to see how this affects the networks functionality. |
| SEQUENTIAL (*boolean*) | Each node is removed one after the next until no nodes are left. |
| CASCADING (*boolean*) | All nodes connected to the node(s) removed on the pervious time step are removed. This is started by the selection of a single node. |
| Node selection method  *(Only one should be set as True, all others should beset as False)* | RANDOM (*boolean*) | Selects the node to remove at random. Valid for all three failure methods. |
| DEGREE (*boolean*) | Picks the node with the highest degree to remove from the network. Valid for SEQUENTIAL and CASCADING failures. |
| BETWEENNESS (*boolean*) | Select the node with the highest betweenness centrality value to be removed. Valid for SEQUENTIAL and CASCADING. |
| Parameters | REMOVE\_SUBGRAPHS (*boolean*) | Remove all the nodes and edges associated with all subgraphs, leaving only the giant component. |
| REMOVE\_ISOLATES (*boolean*) | Remove all isolates node in the same time step as they appear. |
| NO\_ISOLATES (*boolean*) | Allow isolated node to be removed through being selected, e.g. through the random removal strategy. |

### Metrics

The metrics listed as being part of the basic metric list are compulsory when analysing a network and thus the status of these cannot be changed. For those listed under the optional category, these can be or not be used. Set as True to use. To use, all need to be listed, as either True of False.

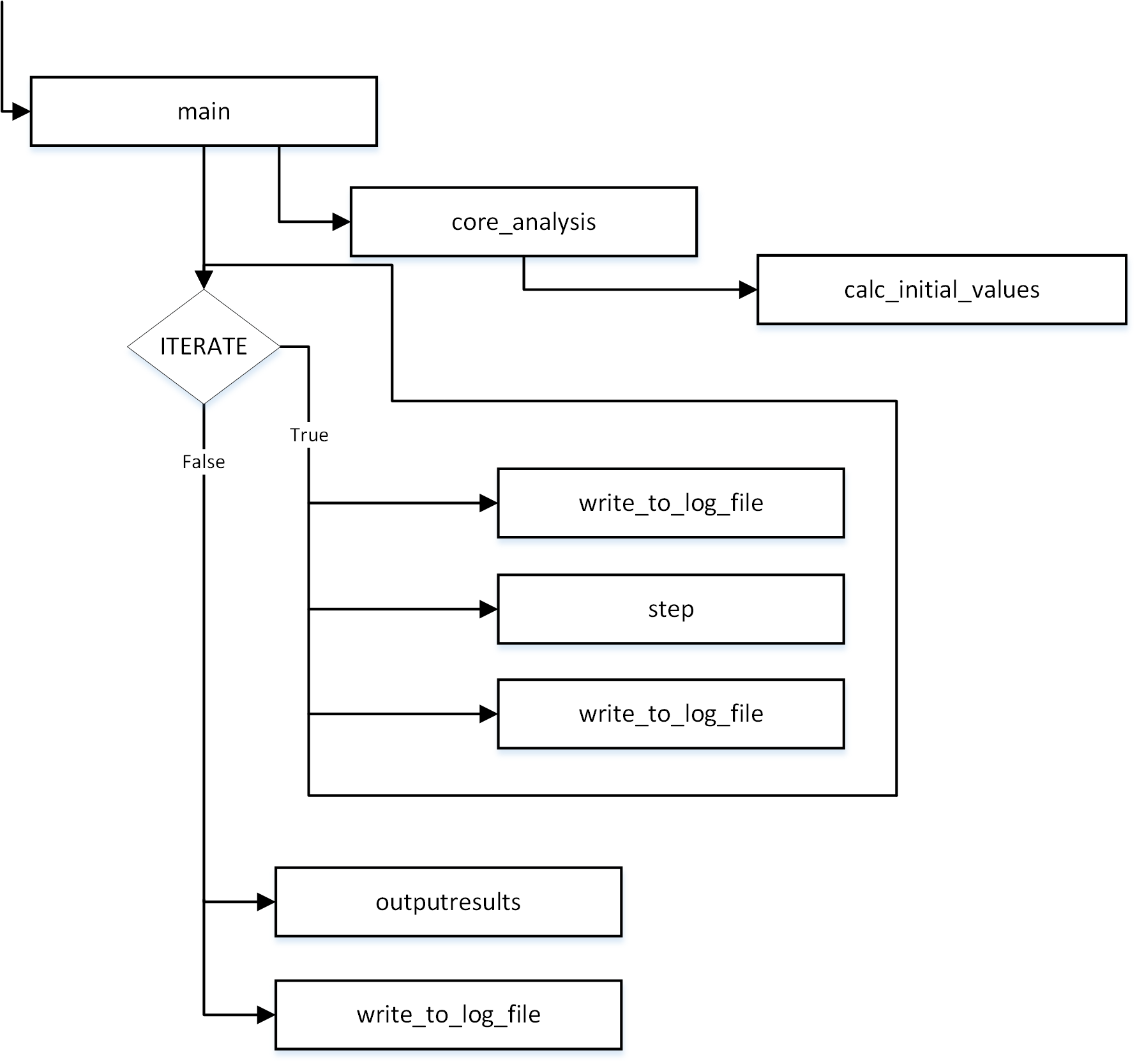
|  |  |
| --- | --- |
| **Basic metrics** | **Description** |
| nodes\_removed (*list*) | Lists the nodes removed. |
| node\_count\_removed (*list*) | Running total of the nodes removed. |
| count\_nodes\_left (*list*) | Running total of the node left in the network. |
| number\_of\_edges (*list*) | Running total of the nodes left in the network. |
| number\_of\_components (*list*) | The number of connected components in the network at each time step. |
| isolated\_n\_count (*list*) | The number of isolated nodes at each step. |
|  |  |
| **Optional metrics** |  |
| size\_of\_components (*list*) | The size in terms of the number of nodes of all the connected subgraphs in the network at each time step. |
| giant\_component\_size (*list*) | The size of the largest connected component at each time step. |
| av\_nodes\_in\_components (*list*) | The average number of nodes across all the components at each time step. |
| isolated\_nodes (*list*) | All isolated nodes at each time step. |
| isolated\_n\_count\_removed (*list*) | A running total of the number of nodes removed due to being isolated. Updated at every time step. |
| subnodes (*list*) | All those nodes which are not part of the giant component at every time step. |
| subnodes\_count (*list*) | A count of the nodes which are not part of the giant component. |
| path\_length (*list*) | The average path length of the network. |
| av\_path\_length\_components (*list*) | The average path length for each connected component. |
| giant\_component\_av\_path\_length (*list*) | The average path length of the giant component. |
| av\_path\_length\_geo (*list*) | The geographic average path length. |
| average\_degree (*list*) | The average degree of the nodes in the network. |
| inter\_removed\_count (*list*) | Those nodes which have been removed due to being part of a cross network link which has been removed. |
|  |  |

### Other

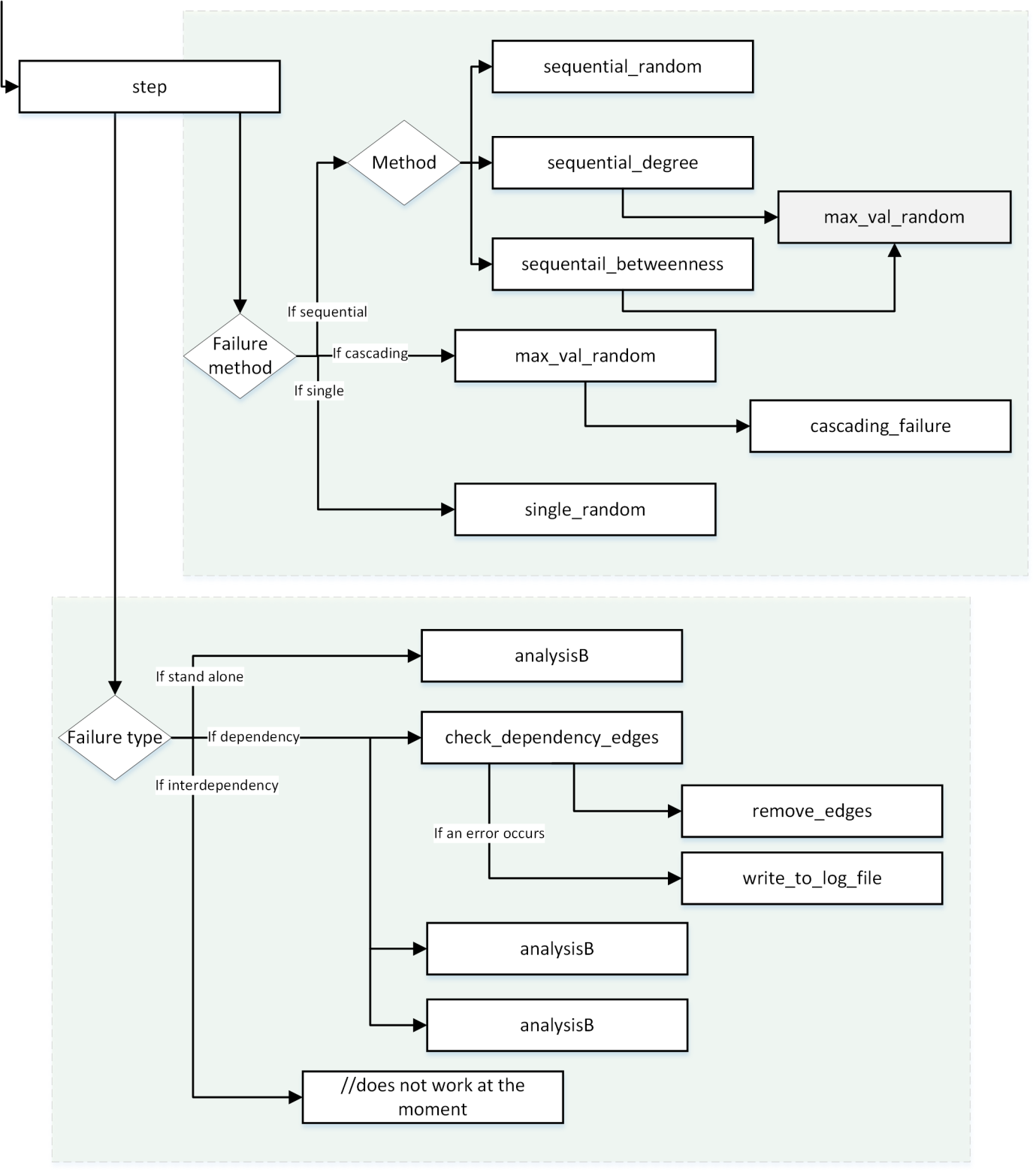
|  |  |
| --- | --- |
|  |  |
|  |  |

# Key function diagrams

## main function



## step function



## analysis\_B function

