A University of Sussex PhD thesis

Available online via Sussex Research Online:

http://sro.sussex.ac.uk/

This thesis is protected by copyright which belongs to the author.

This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the Author.

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the Author.

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given.

Please visit Sussex Research Online for more information and further details.
The Mathematics of Human Contact

Developing Stochastic Algorithms for the Generation of
Time-Varying Dynamic Human Contact Networks

Stephen Ashton

A thesis presented for the degree of
Doctor of Philosophy in Mathematics

Department of Mathematics
School of Mathematical and Physical Sciences
University of Sussex

December 4, 2019
Summary

UNIVERSITY OF SUSSEX
STEPHEN ASHTON
DOCTOR OF PHILOSOPHY IN MATHEMATICS
THE MATHEMATICS OF HUMAN CONTACT

In this thesis, I provide a statistical analysis of high-resolution contact pattern data within primary and secondary schools as collected by the SocioPatterns collaboration. Students are graphically represented as nodes in a temporally evolving network, in which links represent proximity or interaction between students. I focus on link- and node-level statistics, such as the on- and off-durations of links as well as the activity potential of nodes and links. Parametric models are fitted to the on- and off-durations of links, interevent times and node activity potentials and, based on these, I propose a number of theoretical models that are able to reproduce the collected data within varying levels of accuracy. By doing so, I aim to identify the minimal network-level properties that are needed to closely match the real-world data, with the aim of combining this contact pattern model with epidemic models in future work.

I also provide Bayesian methods for parameter estimation using exact Bayesian and Markov Chain Monte Carlo methods, applying these in the case of Mittag-Leffler distributed data to artificially generated data and real-world examples. Additionally, I present probabilistic methods for model selection - namely the Akaike and Bayesian Information Criteria and apply them to the data and examples in the previous section.
Dedication

I would like to dedicate my thesis to everyone who made this possible:

To my supervisor, Enrico Scalas, and to my colleagues in my research group, Nicos Georgiou and István Zoltán Kiss, for guiding me through this process;

To my many lecturers and teachers, who gave me the skills to reach this stage;

To Tommy Nash and Paul Belcher, for inspiring my love of the subject and pushing me to fulfil my potential;

To my friends, who I undoubtedly frustrated with my ramblings about problems in my programming;

To my parents, for everything;

And to you, the reader.
Declaration

I declare that this thesis has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree. Except where stated otherwise by reference or acknowledgement, the work presented is entirely my own.

Parts of this work have been published in Acta Physica Polonica A [16] and are being prepared for submission to Physica A: Statistical Mechanics and its Applications. Additionally, material has been presented in various formats at the following external conferences:

• Séminaire de Mathématiques Supérieures: Dynamics of Biological Systems - Edmonton, Canada - 2016 - Poster presentation

• 13th Econophysics Colloquium & 9th Polish Symposium on Physics in Economy and Social Sciences - Warsaw, Poland - 2017 - Poster presentation

• 20th European Conference on Mathematics for Industry - Budapest, Hungary - 2018 - Poster presentation and talk

• 23rd Annual Workshop on Economic Science with Heterogeneous Interacting Agents - Tokyo, Japan - 2018 - Poster presentation and talk

Stephen Ashton
December 4, 2019
Acknowledgements

Data used for the basis of my network models was previously collected by the SocioPatterns collaboration (http://www.sociopatterns.org/) - I focus on that presented by Gemmetto, Barrat, Cattuto, Mastrandrea and Fournet [74, 188].

Earthquake data used for Bayesian analysis was provided by the United States Geological Society [196]. Data for the recidivism of drug ex-prisoners and data for the duration of human residence was extracted from work presented by Stage and Fedotov [186].


Additionally, I would like to thank an anonymous referee for a thorough reading of my paper with Scalas, Georgiou and Kiss [16] and for the useful suggestions that improved the clarity and presentation of that work that have subsequently improved this thesis.

This research was funded by an Engineering and Physical Sciences Research Council (EPSRC) DTP grant with reference number 1652194.
Notation

Acronyms

ABC Approximate Bayesian computation
AIC Akaike information criterion
ARP Alternating renewal process
BIC Bayesian information criterion
CCDF Complementary cumulative distribution function
CDF Cumulative distribution function
ECCDF Empirical complementary cumulative distribution function
ECDF Empirical cumulative distribution function
EDF Empirical distribution function
IID Independent and identically distributed
GCC Global clustering coefficient
MCMC Markov-chain Monte Carlo
MGF Moment generating function
MLE Maximum likelihood estimator
PDF Probability density function (for continuous random variables)
Probability distribution function (for discrete random variables)
RLAD Random link activation and deletion

Symbols

\( f \star g \) The convolution of continuous density functions \( f \) and \( g \) representing the density of the sum of the two corresponding random variables:

\[
(f \star g) (z) = \int_{-\infty}^{\infty} f(z-t)g(t)dt
\]

\( f_X \) The PDF of the random variable \( X \)
\( F_X(x) \) The CDF of the random variable \( X \):

\[
F_X(x) = \int_{-\infty}^{x} f_X(t)dt
\]

\( \mathbb{I}_A \) The indicator function for \( A \)
Contents

Summary i
Dedication ii
Declaration iii
Acknowledgements iv
Notation v

1 Introduction 1
   1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
   1.2 Literature Review . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
      1.2.1 Epidemiological Models . . . . . . . . . . . . . . . . . . . . 1
      1.2.2 Network Models . . . . . . . . . . . . . . . . . . . . . . . . . 3
   1.3 Network Theory . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
   1.4 Distributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2 Data Analysis 17
   2.1 Data Collection and Description . . . . . . . . . . . . . . . . . . . . . 17
      2.1.1 Analysis of Original Data . . . . . . . . . . . . . . . . . . . . 18
      2.1.2 Methods of Comparison . . . . . . . . . . . . . . . . . . . . . 19
   2.2 Results of Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

3 Model Development 24
   3.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
      3.1.1 Sample Comparison . . . . . . . . . . . . . . . . . . . . . . . . 24
   3.2 Model Creation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
      3.2.1 Model 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
      3.2.2 Model 2a . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
      3.2.3 Model 2b . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
      3.2.4 Model 2c . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
      3.2.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
   3.3 Model Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
      3.3.1 Model 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
      3.3.2 Model 2a . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
      3.3.3 Model 2b . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
      3.3.4 Model 2c . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
   3.4 Model Comparison . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
# Contents

3.5 Model Validation .................................................. 44

4 Bayesian Estimation ................................................. 46
   4.1 Introduction .................................................. 46
   4.2 Bayesian Estimates ............................................ 46
      4.2.1 Exact Bayesian Estimate ................................ 46
      4.2.2 Markov-Chain Monte Carlo Method ...................... 47
      4.2.3 Approximate Bayesian Computation ..................... 49
   4.3 Examples ...................................................... 50
      4.3.1 Testing of Exact and MCMC Methods on Generated Data 50
      4.3.2 Earthquake Recurrence .................................. 58
      4.3.3 Testing of ABC on Generated Data ....................... 61
      4.3.4 Social Data .............................................. 64

5 Model Selection ...................................................... 66
   5.1 Introduction .................................................. 66
   5.2 Methods ....................................................... 66
      5.2.1 Akaike Information Criterion ............................. 66
      5.2.2 Bayesian Information Criterion .......................... 67
      5.2.3 Comparisons Between Methods ............................ 67
   5.3 Results for Mittag-Leffler Distributed Data .................. 68
   5.4 Applications to Network Generation Models ................... 68

6 Conclusions ........................................................ 70

Bibliography .......................................................... 71

A Code Manuals ....................................................... 87
   A.1 ABC_ml2.m ..................................................... 87
   A.2 aicbic_ML.m ................................................... 88
   A.3 analyse.m ...................................................... 88
   A.4 analyse_interevents.m ......................................... 90
   A.5 compareData.m ................................................ 90
   A.6 compareOriginal.m ............................................ 91
   A.7 comparison.m .................................................. 92
   A.8 echoes.m ....................................................... 92
   A.9 explognconvhist.m ............................................. 93
   A.10 MCMCP_mlf2.m ............................................... 94
   A.11 triangleClosed.m ............................................ 94
   A.12 validation.m .................................................. 95

B Code Listing .......................................................... 97

Lists & Indices ........................................................ 300
   List of Figures .................................................... 300
   List of Tables ..................................................... 303
   Index of Names .................................................... 304
   Index of Subjects ................................................ 306
Chapter 1

Introduction

1.1 Motivation

The use of networks to model contact patterns or interactions between individuals has proved to be a step change in how epidemics and other spreading processes are modelled [48, 105, 114, 138, 205, 212]. The basic ingredient of such models is to represent individuals by nodes and contacts between these as links between nodes. The use of graph-theoretical methods have helped to reveal and understand the role of contact heterogeneity, preferential mixing and clustering in how diseases invade and spread [21, 102]. Having good network models is crucial. Simple mechanistic models that capture and preserve key properties of empirical networks are often employed as they offer greater flexibility in changing and tuning various network properties. While such models and theory are well developed for static networks, it is only recently that real-world time varying forms have been empirically measured [21, 48, 62, 64, 102, 104, 162, 166, 180, 187, 189].

As a starting point, I utilise the paper by Georgiou, Kiss and Scalas [75], which uses a non-Markovian dynamic network with random link activation and deletion (RLAD) and a heavy-tailed Mittag-Leffler distribution for the interevent times.

1.2 Literature Review

1.2.1 Epidemiological Models

1.2.1.1 Deterministic Categorical Models

The most basic deterministic categorical epidemiological models are where the population is split into categories (such as susceptible to a disease, or infected with a disease etc.) and movement between categories happens with given rates. Examples of the setup of this type of model can be seen in Figures 1.1 and 1.2. The dynamics in these models is fairly simple, but can be easily changed to account for further categories (such as those recovered from a disease), population processes (by adding appropriate birth and death rates into the model) and strategies such as vaccination (represented by movement from the susceptible category straight to the recovered category with a given rate). Models of this type such as susceptible-infectious-susceptible (SIS), susceptible-infectious-recovered (SIR) and so forth reference a random variable, but only use its expected value during calculations in the
1.2.1.2 Stochastic Categorical Models

Simple stochastic forms of these models exist, which can introduce a noise component or use random processes for time period - such as replacing the recovery time with an independent and identically distributed (IID) random variable for each member with mean chosen to align with the deterministic model. The most common stochastic variations are the general stochastic epidemic model and the Reed-Frost model. In the general stochastic epidemic model, a fixed population SIR model may be used with the transition probabilities in (1.1), where $S_t$ and $I_t$ are the numbers of susceptible and infected individuals at time $t$, $N$ is the total population size, and $\beta$ and $\gamma$ are the infection and recovery rates:

- $P((S_{t+\Delta t}, I_{t+\Delta t}) - (S_t, I_t) = (-1, 1)) = \beta \frac{S_t I_t}{N} \Delta t + o(\Delta t)$,
- $P((S_{t+\Delta t}, I_{t+\Delta t}) - (S_t, I_t) = (0, -1)) = \gamma I_t \Delta t + o(\Delta t)$,
- $P((S_{t+\Delta t}, I_{t+\Delta t}) - (S_t, I_t) = (0, 0)) = 1 - \left(\beta \frac{S_t}{N} + \gamma\right) I_t \Delta t + o(\Delta t)$.

In a simple Reed-Frost model, a fixed population categorical model with susceptible and infected categories is created and the number of new cases $C_{t+1}$ given by (1.2) is examined:

$$C_{t+1} = S_t \left(1 - (1 - p)^{I_t}\right),$$

where $C_{t+1}$ is the number of new infections in the interval $(t, t+1]$, $S_t$ and $I_t$ are the numbers of susceptible and infected individuals at time $t$ and $p$ is the effective contact rate - this method assumes guaranteed infection if a susceptible individual is in contact with an infected individual. Other models may use Markovian
switching, where the infection and recovery rates have multiple states with transi-
tional probabilities [82], or jump perturbations [222], where some movement between
states occurs stochastically [96], in order to introduce an element of stochastic be-
haviour into the system.

1.2.1.3 Network-Driven Models

For network-driven models, there are two broad areas where focus can be made -
the model itself, and the underlying network (which will be discussed in subsection
1.2.2). Keeling and Eames provide a good overview of many of the types of network-
driven models that exist [102].

The most simple network-driven model replaces the rate at which a susceptible
person is infected with a rate proportional to the number of infections within a
neighbourhood of it [60, 65, 127, 155, 204, 210]. A generalisation of this is the
independent cascade model (ICM), where the infection rate can vary between edges
[78, 103, 173], and is one of the most basic and widely-studied diffusion models.

In this model, the network starts with an initial set of active (infected) nodes and
proceeds in discrete time steps. When a node \( v \) first becomes active at time \( t \), it will
attempt to activate each inactive neighbour \( w \) with probability \( p_{v,w} \), which may vary
for different choices of \( v \) and \( w \). If this activation occurs then \( w \) will become active at
\( t + 1 \), if it does not, then \( v \) cannot make any further activation attempts on \( w \) [103].

Another model of this type is the linear threshold model (LTM) [103, 173, 226] where
each node, \( v \), is influenced by its neighbour, \( w \), with a weight \( b_{v,w} \), such that (1.3)
holds where \( A \) is the adjacency matrix for the network (discussed later in section
1.3):

\[
\sum_w A_{v,w} b_{v,w} \leq 1. \tag{1.3}
\]

If the sum of these weights around \( v \) exceed some random threshold \( \theta_v \), chosen for
each \( v \) uniformly at random from the interval \([0, 1]\), then \( v \) becomes active in the
network. As with the ICM, this starts with an initial set of active nodes and proceeds
in discrete time steps. At time \( t \), any nodes active at \( t - 1 \) remain active and the
model actives any node such that (1.4) holds:

\[
\sum_w A_{v,w} b_{v,w} \geq \theta_v. \tag{1.4}
\]

These thresholds \( \theta_v \) represent the different tendencies of nodes to copy their neigh-
bours - in effect their level of immunity to a disease - whilst the weightings \( b_{v,w} \)
represent pressure to become active - a weighted measure of the infectivity of the
neighbours of \( v \) [103]. This model, in itself, is a generalisation of models based on
the concept of node-specific thresholds that were first proposed by authors such as
Granovetter and Schelling [80, 170] and investigated by many other authors
[29, 121, 122, 133, 147, 199, 200, 208, 218, 219] where these thresholds \( \theta_v \) are dis-
tributed according to some given distribution (or may in fact all be equal and result
in a degenerate distribution).

1.2.2 Network Models

A larger problem involves the dynamics of the underlying network. Some mod-
els use empirical data collected from sensors or other methods (such as I will be
using later on) and directly applying an appropriate disease model on top of this [21] [162] [187]. Others use this data, but simplify it for modelling purposes - such as using links as in gathered data, but drawing link lifespans from a uniform distribution [162], giving links simple weights [166] [187] or converting fully-connected networks into sparse ones [166]. However, having an exact contact network, such as that given by empirical data drawn from sensors, would typically be infeasible, even for small populations. Some researchers therefore prefer to work with approximate contact networks, either by gathering information regarding the contacts with infected individuals [109], surveying individuals [65] or using census [65] [127], social characteristic [87], or other such data [65] [126], and creating an appropriate network through strategies such as random walks [109] or other similar methods that restrict movements based on infrastructure [65]. As an example, a random walk method of constructing a contact network involves selecting an individual at random from a population and interviewing them to gather the identities and locations of those they are linked with. This method then selects one of these linked individuals at random to be the next step of the walk and repeats for the desired number of steps and can result in a network such as that in Figure 1.3 [109].

1.2.2.1 Idealised and Regular Random Networks

Another approach to the network dynamics is to generate an idealised network for usage in the model. Random-mixing networks are the most simple variation of these [102] - the most common algorithm for which is the Erdős-Rényi model which has two closely related but distinct variants. The first, \( G(n, M) \), chooses a network uniformly from the space of all networks with \( n \) nodes and \( M \) edges, whilst the second, \( G(n, p) \), connects \( n \) nodes randomly, with each link being included in the graph with probability \( p \) independent from every other edge [63] [77], and leads to a network with a degree distribution that is approximately Poisson [102]. This gives rise to the alternative name for these random networks - Poisson random networks [21]. In regular random networks, the most analytically simple version of a random network, each individual will have a fixed number of contact links [21] [32], leading to networks of this type exhibiting a lack of clustering and displaying homogeneity of node-specific network properties. In both forms of random network, the spatial position of individuals is not considered.

1.2.2.2 Lattice and Small-World Networks

Lattice networks are also used [102], in which individuals are positioned on a regular grid of points and adjacent nodes are connected by a link, thereby localising contacts in space and resulting in a high degree of clustering and once more homogeneity of node-specific network properties - an example of this type of network can be seen in Figure 1.4.

The contact process [88] [116] and forest-fire model [20] are the better known uses of lattice networks, and have parallels with the SIS and susceptible-infectious-recovered-susceptible (SIRS) models respectively. In the contact process, an infected
Figure 1.3: Example contact network generated by a random walk study in Canberra, Australia, with nodes placed in relation to the locations within the city where participants were resident (adapted from Figure 2 of [109]).
node becomes healthy and susceptible again at rate 1 independent of other nodes, and a susceptible node becomes infected at a rate of $\lambda$ times the number of infected neighbours [59]. In the forest-fire model, nodes are updated at discrete time steps using the following rules [57]:

(i) A burning tree becomes an empty site.

(ii) A green tree becomes a burning tree if at least one neighbour is burning.

(iii) An empty site grows a green tree with probability $p$.

This has clear parallels with an epidemic model with green trees representing susceptible nodes, burning trees representing infected nodes and empty sites representing recovered nodes. Due to the form of their construction, lattice-based models lead to the spread of infection in roughly circular wavefronts around the initially infected individuals, with the collision of wavefronts leading to a much more rapid infection of the entire system. Due to this behaviour, models of this type are best suited for spatially extended regions with highly localised transmission, where this wave-like propagation is most likely to occur [84, 132]. Like all network-driven epidemiological models, lattice-based models show a reduced initial growth as compared to random-mixing models, although this effect is more pronounced due to the spatial layout resulting in a much faster consumption of susceptible individuals within the local environments of initially infected sites. Small-world networks are a modification of lattice networks and introduce random and rare long-range links, which reduces the distance between any two randomly selected individuals and leads to a more rapid infection spread. These often have highly connected sets of nodes, with sparse interconnections between these sets, and can be seen to represent such things as road infrastructure - where there will be many short local roads within a town, with a few longer roads between towns. A simple example can be seen in Figure 1.5 where each node is connected to its neighbours, but some long-range links also exist.

One proposed method for constructing a small world network is to first create a random network using a method such as the Erdős-Rényi model. Some nodes are then replaced by a densely connected set of nodes of a random size. Links in the original random network are then placed between these sets, choosing end points at
Small-world networks as described by Watts and Strogatz [209, 210] provide a useful intermediary stage between the rigidity of lattices and the unstructured nature of random networks. Small-world networks are thought to be the basis of many human-related networks [195], being observed in the collaboration networks of scientific authors [136], co-star networks of film networks [210] and gene and neural networks [210].

1.2.2.3 Spatial Networks

Spatial networks are highly important in many forms from epidemiology to urbanism as they put weight on space as well as network topology. Here, nodes are placed
in space according to the natural geometry of the system - the locations of road intersections in a town, for example - and links are inserted with a probability that depends on their separation distance, defined by a connection kernel - most simply, links that are longer in space have higher cost and lower probability of being inserted during the generation of these networks \[23, 102\]. This cost association with distance reflects in situations such as neurology \[39\], technological infrastructure and quantitative geography \[86\], where spatially longer links have a clear material or temporal cost associated with them. More generally, this cost can be based on other parameters, such as socio-economic distance. This cost involved in the generation of the network has an important effect on the topological properties and processes that will occur, and requires that longer links are compensated by some advantage, such as being connected to a ‘hub’ node - one with a high degree \[23\]. By changing the spatial layout of nodes or the connection kernel, it is possible to generate a large variety of networks, from lattices and small-world networks to completely random and highly-connected networks \[60, 100, 155\]. Spatial networks show a relatively high amount of heterogeneity, with a degree distribution that is approximately Poisson \[102\].

1.2.2.4 Scale-Free Networks

One of the largest issues with the network types discussed so far is the properties of the degree distributions. Small-world models, lattice networks and regular random networks often display little variation in the size of neighbourhoods, whilst spatial networks and those produced by methods such as the Erdős-Rényi model display degree distributions that are approximately Poisson. However, this is often not the case in many observed networks - it is often the case that many nodes in a network have a very small degree, whilst a few have much larger neighbourhoods \[9, 11, 21, 22, 97, 117\]. Since this degree distribution is likely to be highly important in modelling disease transmission - with highly connected individuals having a much higher impact on the dynamics than those with smaller neighbourhoods - reflecting this in models is necessary in order to capture intricacies in the behaviour within the network \[12, 91, 174\]. So-called scale-free networks attempt to address this by having a degree distribution that follows a power law, at least asymptotically. The property is independent of the size of the network, and thus networks that have this feature are known as scale-free \[216\]. These networks can be constructed by adding nodes and links in a manner that mimics natural mechanics \[10, 22, 141\], with each new individual having a preference for connecting to nodes that already have a large neighbourhood - reflecting the real-world behaviour of the desire to be friends with people who are already popular. This power-law property has been observed in internet networking \[9\], collaborations \[22\] and sexual contacts \[117\]. An example of a scale-free network can be seen in Figure 1.7.

In scale-free networks, the few high degree nodes and large number of more regular nodes greatly increases the individual risk of infection as once one of these hub nodes has been infected, it can infect a large number of other individuals in the network very quickly or maintain the disease within the network as can be seen in many studies \[91, 137, 141, 156\]. An example of a scale-free network is the preferential
Figure 1.7: Examples of a scale-free network, showing the preservation of the underlying structure between various sizes. (From [197])

attachment model [22], also known as the Barabási-Albert model which follows the algorithm:

1. **Growth**: Starting with a small number \( (m_0) \) of nodes, at each time step, add a new node with \( m \ll m_0 \) links to other nodes already in the system.

2. **Preferential attachment**: When choosing nodes to which the new node connects, assume that the probability, \( p_i \), of the new node connecting to node \( i \) depends on the degree \( k_i \) of node \( i \) such that (1.5) holds:

\[
 p_i = \frac{k_i}{\sum_j k_j}.
\]

(1.5)

After \( t \) time steps, this results in a network with \( t + m_0 \) nodes and \( mt \) edges [8]. This model is related to Yule’s process and Simon’s process as discussed by Simkin and Roychowdhury [170]. Simulations and studies of scale-free network, reveal that the efficiency of random vaccination strategies is very limited [12, 119, 141, 143], although for limitations on node degree and attachments, such strategies can be more effective [165, 206]. The best method of vaccination on scale-free networks is a targeted approach, with the inclusion of a limited number of highly connected nodes in the vaccination strategy being enough to prevent an epidemic [9, 119, 141, 142]. This strategy relies on knowledge of the network dynamics, although similar effective strategies without such knowledge do also exist [45].

### 1.2.2.5 ERGM Networks

Exponential random graph models (ERGMs), commonly called the \( p^* \) class of models [70, 144, 160, 207], permit the construction of models that reflect the structural foundations of social behaviour and are useful for examining multilevel hypotheses in social networks [47]. One approach to generating networks of this type is a probability model, where the probability of a network \( G \) taking the value \( g \) from \( G_n \), the set of all networks on \( n \) nodes, is given by (1.6):

\[
P(G = g) = \exp \left( \sum_i \theta_i T_i(g) - \psi(\theta) \right).
\]

(1.6)

Here \( \theta = \{\theta_i\}_{i=1}^m \) are \( m \) real valued parameters, \( T_i \) are real valued statistics defined on \( G_n \), and \( \psi \) is a normalising function. Put simply, a researcher specifies a ERGM
by choosing a set of network configurations of theoretical interest. Then, by applying this particular model to an observed network, parameters $\theta$ are estimated, permitting inferences about the network patterns in the data, and in turn inferences about the type of processes that are important in creating and sustaining the network [159]. Compared to other methods of network generation, ERGMs focus on the interaction between the structures of a relationship network and individual attributes [98] and can represent complicated dependency patterns that are not easily captured by more basic probability models [182].

### 1.2.2.6 Temporal Networks

For networks that vary in time, there is less established theory. One approach is to use RLAD as mentioned earlier, with absent links being activated independently at some rate $\alpha$, while existing links are independently deactivated at some other rate $\omega$ [104, 193]. Additionally constraints can be placed on this globally such as reducing $\alpha$ as the total number of active links in the network increase. Whilst a useful approach, this random choice of links may not fully reflect real-world dynamics. Another approach is to focus on small subgraph patterns in networks, called network motifs (such as those shown in Figure 1.8), evolving the network by creating new motifs by the addition or reuse of nodes and links [154]. The distinction between a network motif and any other subgraph is a statistical question, with motifs being recognised as subgraphs that are statically over-represented in observed networks [214].

Similarly, algorithms using a temporal hyperedge replacement grammar (tHRG) - a listing of graphical rewriting rules extracted from tree decompositions, where network building-blocks are identified and changed - can also be used, although it is acknowledged that models of this type make improper assumptions and fail to model the growth of a temporal graph [148]. This is an extension to algorithms that use
hyperedge replacement grammars (HRGs), where the HRG is learned from a given graph $H$ by first computing a clique tree\(^1\), then using this HRG to add and remove nodes and links to create a new network $H^*$. Put simply, a HRG gives the rules for replacing labelled edges in a network. For example in Figure 1.9 we apply a rule, $S ::= G'$ that states that all edges labelled $S$ in the original network $G$ (Subfigure 1.9a) are to be replaced with $G'$ (Subfigure 1.9b), which results in a new network (Subfigure 1.9c) \([4, 13, 43, 56]\).

Various algorithms exist, which can either create an isomorphic copy\(^2\) of the original networks such that $H \simeq H^*$, or use stochastic methods to generate random networks that have similar characteristics to the original, but allow for broader generalisations in the exact networks created \([4]\).

It is therefore in the area of temporal networks that this thesis will direct its focus.

### 1.3 Network Theory

Whilst I shall not present an entire summary of graph theory here, a brief summary including key definitions will be given for the purposes of clarity.

A network $G = (V, E)$ is a collection of nodes $V$ and links $E$ defined between pairs of nodes - alternative language refers to $G$ as a graph, with vertices $V = V(G)$ and edges $E = E(G) \subseteq V \times V$ \([37]\). Networks can have various property such as being directed (where a link from $u$ to $v$ does not necessarily imply a link from $v$ to $u$) or undirected, weighted (where different links have different weights assigned to them - this can be the cost of traversing such a link, or the distance that a link represents in an observed system) or unweighted and so forth \([37]\). Unless specified otherwise, in this thesis, I shall be using unweighted, undirected networks without self-loops (links that connect nodes to themselves).

When dealing with a network, it is often useful to consider the adjacency matrix of $G$, $A = A(G)$. This matrix (for undirected and unweighted networks) is a logical symmetric matrix defined by the rules in \((1.7)\):

$$A_{i,j} = \begin{cases} 
1, & \text{if there is a link between } i \text{ and } j, \\
0, & \text{otherwise.} 
\end{cases} \quad (1.7)$$

A path of length $n - 1$ between two nodes $v_1$ and $v_n$ exists if there is a sequence of nodes $(v_1, v_2, \ldots, v_n)$ such that $A_{i_{i+1}} = 1, \forall i \in \{1, 2, \ldots, n - 1\}$. The $(i, j)$-th

\(^1\)This definition is given in section 1.3
\(^2\)This definition is given in section 1.3
entry of \( A^k \) counts the number of paths of length \( k \) between \( v_i \) and \( v_n \) [37]. A triangle is a path of length 3 from a node to itself - the number of unique triangles involving a node \( v \) is equal to \( \frac{1}{2} (A_3)_{ii} \), with the total number of triangles in the network equal to \( \frac{1}{2} \text{Tr} (A^3) \) [37]. If every pair of nodes in a network is connected by exactly one path, then the network is called a tree [28]. A tree is given an arbitrary root node to assign a direction in the tree - if two nodes, \( v_i \) and \( v_j \), have an edge \((v_i, v_j)\) in the tree and there is a path of length \( n \) from \( v_i \) to the root node, and a path of length \( n + 1 \) from \( v_j \) to the root node, then \( v_i \) is said to be a parent of \( v_j \) and \( v_j \) is said to be a child of \( v_i \). A leaf node is one that has no children [167].

Two undirected, unweighted networks \( \mathcal{G} \) and \( \mathcal{H} \) are said to be isomorphic if there exists a bijection \( f \) in (1.8):

\[
f : V(\mathcal{G}) \rightarrow V(\mathcal{H})
\]

such that a link between vertices \( u \) and \( v \) in \( \mathcal{G} \) exists if and only if a link between vertices \( f(u) \) and \( f(v) \) in \( \mathcal{H} \) exists. If this isomorphism exists, then \( \mathcal{G} \simeq \mathcal{H} \). Network isomorphism is an equivalence relation on networks and thus partitions the set of all networks into equivalence classes [110].

The degree of a node \( v \) is the number of links involving it [51], and is equal to \( \sum_{u} A_{u,v} \). With directed networks, this should be extended to the notions of in-degree (the number of active links to a specific node) and out-degree (the number of active links from a specific node). The degree distribution of the network is the probability distribution of these degrees over the whole network such that (1.9):

\[
p_k = \text{probability that a node has degree } k.
\]

A complete network is one such that an edge exists between every pair of nodes \( u, v \in V \), that is \( V = E \times E \). A subgraph \( \mathcal{H} \) of a network \( \mathcal{G} \) is formed from a subset of the nodes and links, such that \( V(\mathcal{H}) \subseteq V(\mathcal{G}) \) and \( E(\mathcal{H}) \subseteq E(\mathcal{G}) \) and that all endpoints of links in \( E(\mathcal{H}) \) are contained within \( V(\mathcal{H}) \). A clique of a network \( \mathcal{G} \) is a complete subgraph of \( \mathcal{G} \) [66]. A maximal clique \( U_1 \) of \( \mathcal{G} \) is a clique with \( n \) nodes, such that no other clique \( U_2 \) of \( \mathcal{G} \) exists such that \( V(U_1) \subset V(U_2) \) - that is there is no additional set of nodes that can be added to \( U_1 \) such that it remains a clique. This should not be confused with a maximum clique of \( \mathcal{G} \) - that is a clique of \( \mathcal{G} \) with \( n \) nodes, such that no clique of \( \mathcal{G} \) with a larger number of nodes exists [26].

Let \( F = \{F_1, F_2, \ldots \} \) be a family of subsets of some arbitrary set of elements, then the intersection graph of \( F \) is a graph whose nodes correspond to the subsets of \( F \), with a link existing between nodes \( v_i \) and \( v_j \) if \( F_i \cap F_j \neq \emptyset \). A clique graph \( K(\mathcal{G}) \) of some network \( \mathcal{G} \) is the intersection graph of the maximal cliques of \( \mathcal{G} \) - that is the intersection graph on the family \( F \) where, for all \( i, F_i \in F \) is a maximal clique of \( \mathcal{G} \), and for every maximal clique \( U \) of \( \mathcal{G} \), there is some \( j \) such that \( U = F_j \in F \) [191]. A clique tree \( T(\mathcal{G}) \) (which is not necessarily unique) of \( \mathcal{G} \) is a subgraph of the clique graph of \( \mathcal{G} \) such that:

- \( T(\mathcal{G}) \) is a tree,
- \( V(T(\mathcal{G})) = V(K(\mathcal{G})) \),
- For every pair of cliques \( U_1, U_2 \) such that \( U_1 \cap U_2 = S \), all cliques on the path between \( U_1 \) and \( U_2 \) in \( T(\mathcal{G}) \) contain \( S \) [31].

An example of a simple network and its corresponding clique tree is given in Figure 11.10.
1.4 Distributions

I shall use a range of test distributions in this thesis when examining behaviours, including heavy-tailed and power-law distributions. A heavy-tailed distribution is one that has a tail heavier than an exponential distribution - more formally, a distribution is said to have a heavy tail if the moment generating function (MGF) of the random variable $X$, $M_X(t)$ is infinite for all $t > 0$ [38, 163]. The inclusion of power-law distributions is important as it is known that several properties linked to human-related activities show power-law decaying distributions [198]. Many of these are common, although I shall also be using the Mittag-Leffler distribution as used by Georgiou, Kiss and Scalas [75].

**Exponential Distribution**  The exponential distribution has the probability density function (PDF) as given in (1.10) with mean $\lambda^{-1}$:

$$f(x; \lambda) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0, \\ 0, & x < 0. \end{cases} \quad (1.10)$$

Here the parameter $\lambda \in (0, \infty)$ represents rate, and the distribution has support $x \in [0, \infty)$ [69]. The exponential distribution describes the time between events in a homogeneous Poisson point process and is a member of the exponential family of distributions [69, 139, 168]. This distribution features heavily in queuing theory and due to the memoryless property of the exponential distribution is well-suited to model such things as hazard rates [52, 69].

**Gamma Distribution**  The gamma distribution is also a member of the exponential family of distributions and has the PDF as given in (1.11) - where $\Gamma(z)$ is the gamma function as given in (1.12) - with mean $k\theta$:

$$f(x; k, \theta) = \begin{cases} \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-\frac{x}{\theta}}, & x \geq 0, \\ 0, & x < 0. \end{cases} \quad (1.11)$$

$$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx. \quad (1.12)$$

Here, the parameters $k \in (0, \infty)$ and $\theta \in (0, \infty)$ represent shape and scale respectively, and the distribution has support $x \in [0, \infty)$ [69]. The gamma distribution
can be modified to the exponential distribution with $\lambda = \theta^{-1}$ ($k = 1$), the Erlang distribution ($k \in \mathbb{Z}$) or the chi-squared distribution with $v$ degrees of freedom ($k = \frac{v}{2}$ and $\theta = 2$) \cite{183}. This distribution is one of the most common, seeing uses such as describing inter-event intervals in neuroscience\cite{161, 215} and cancer incidence in oncology\cite{27}. For independent gamma distributions with identical scale parameters, the additive property is satisfied - that is, for $N$ IID $X_i \sim \Gamma (k_i, \theta)$, (1.13) holds:

$$\sum_{i=1}^{N} X_i = \Gamma \left( \sum_{i=1}^{N} k_i, \theta \right) \quad (1.13)$$

**Rayleigh Distribution**  The Rayleigh distribution has the PDF as given in (1.14) with mean $\sigma \sqrt{\frac{\pi}{2}}$ :

$$f(x; \sigma) = \begin{cases} \frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}, & x \geq 0, \\ 0, & x < 0. \end{cases} \quad (1.14)$$

Here the parameter $\sigma \in (0, \infty)$ represents scale, and the distribution has support $x \in [0, \infty)$ \cite{69}. It can be generalised into the Rice distribution, being equal to a Rice distribution with distance parameter, $\nu$, equal to 0 \cite{6}. It is naturally observed when dealing with the magnitude of a vector with random IID Gaussian components and is often linked to dealing with such events, for example, estimating the noise variance in magnetic resonance imaging (MRI) and radar data \cite{7, 51, 175} and for wind energy analysis \cite{99, 172}.

**Log-Normal Distribution**  The log-normal distribution is the distribution of a random variable whose logarithm is normally distributed. It has the PDF as given in (1.15) with mean $\exp \left( \mu + \frac{\sigma^2}{2} \right)$ :

$$f(x; \mu, \sigma^2) = \begin{cases} \frac{1}{x\sqrt{2\pi} \sigma} \exp \left( -\frac{(\log x - \mu)^2}{2\sigma^2} \right), & x \geq 0, \\ 0, & x < 0. \end{cases} \quad (1.15)$$

Here, the parameters $\mu \in (-\infty, \infty)$ and $\sigma \in (0, \infty)$ are the mean and standard deviation respectively of the underlying normal distribution, and the log-normal distribution has support $x \in (0, \infty)$ \cite{69}. It is a heavy-tailed distribution for all parameters \cite{135}. It appears in many natural phenomena and growth processes, such as forum post lengths \cite{183} and dwell times \cite{217}. It also appears in such things as citation counts \cite{194} and disease incubation rates \cite{115}, alongside many other measurements in biology, medicine and social sciences.

**Generalised Pareto Distribution**  The generalised Pareto distribution has the PDF as given in (1.16) with mean $\theta + \frac{\sigma}{1-k}$ for $k < 1$ :

$$f(x; k, \sigma, \theta) = \begin{cases} \frac{1}{\sigma} \left( 1 + k \frac{x - \theta}{\sigma} \right)^{-\frac{1}{k}}, & k > 0, x > \theta, \\ \frac{1}{\sigma} e^{-\frac{x - \theta}{\sigma}}, & k = 0, x > \theta, \\ \frac{1}{\sigma} \left( 1 + k \frac{\theta - x}{\sigma} \right)^{-\frac{1}{k}}, & k < 0, \theta - \frac{\sigma}{k} > x > \theta, \\ 0, & \text{otherwise}. \end{cases} \quad (1.16)$$
Here, the parameters are $k \in (-\infty, \infty)$, $\sigma \in (0, \infty)$ and $\theta \in (-\infty, \infty)$, representing shape, scale and location respectively and the distribution has support $x \in [\theta, \infty)$ for $k \geq 0$ and $x \in [\theta, \theta - \frac{\sigma}{k}]$ for $k < 0$ [16, 49, 178, 221]. For $k = 0$, this becomes an exponential distribution and for $k > 0$, this becomes a heavy-tailed distribution - a Pareto (Type I) distribution with tail index $\alpha = \frac{1}{k}$ [33, 135]. It has uses in extreme value theory [34, 92, 95, 201] when applied to phenomena such as extreme wind speeds or earthquake ground movements.

**Weibull Distribution** The Weibull distribution has the PDF as given in (1.17) with mean $\lambda \Gamma \left(1 + \frac{1}{k}\right)$:

$$f(x; \lambda, k) = \begin{cases} \frac{k}{\lambda} \left(\frac{x}{\lambda}\right)^{k-1} e^{-\left(\frac{x}{\lambda}\right)^k}, & x \geq 0, \\ 0, & x < 0. \end{cases}$$

(1.17)

Here, the parameters $\lambda \in (0, \infty)$ and $k \in (0, \infty)$ represent scale and shape respectively, and the distribution has support $x \in [0, \infty)$ [69]. For $k \in (0, 1)$, this is a heavy-tailed distribution [135]. The Weibull distribution interpolates between the exponential ($k = 1$) and Rayleigh distributions ($k = 2$ and $\lambda = \sqrt{2\sigma}$) [184]. Additionally, as $k$ tends to $\infty$, the Weibull distribution converges to a Dirac delta distribution centred at $x = \lambda$ [140]. This distribution is used in a wide range of applications including survival analysis and extreme value theory [41, 223].

**The Mittag-Leffler Distribution** The Mittag-Leffler distribution is a useful distribution in terms of analysing stochastic processes with positive values containing rare events, as it is a heavy-tailed power-law distribution with infinite mean [94, 150]. Additionally, its usage leads to simpler analytical treatment of non-Markovianity in the presence of extreme power-law tails than a Pareto distribution with similar behaviours as discussed by Georgiou, Kiss and Scalas [75].

There are different families of Mittag-Leffler distributions, relating the Mittag-Leffler function to either the cumulative distribution function (CDF) or to the MGF. I shall use the single-parameter version of the former, and throughout this thesis I shall define the Mittag-Leffler distribution as the continuous distribution with survival function of the form given in Equation (1.18) below:

$$\Psi(t) = P(T > t) = E_{\mu} \left( - \left( \frac{t}{\tau_0} \right)^{\mu} \right), \quad t \geq 0, \quad \tau_0 > 0, \quad 0 < \mu \leq 1,$$

(1.18)

where $E_{\mu}(z)$ is the one-parameter Mittag-Leffler function, given as

$$E_{\mu}(z) = \sum_{n=0}^{\infty} \frac{z^n}{\Gamma(\mu n + 1)}, \quad \mu > 0, \quad z \in \mathbb{C},$$

(1.19)

where $\Gamma(z)$ is the gamma function defined in (1.12) [68].

For $\mu = 1$, the Mittag-Leffler distribution reduces to the exponential distribution, and as such for $0 < \mu < 1$ is viewed as a fat-tailed generalisation of the exponential [118].

To illustrate the heavy-tailed nature of the Mittag-Leffler distribution and the effect that the parameter $\mu$ has on this, the complementary cumulative distribution functions (CCDFs) for a Mittag-Leffler distribution between $t = 1$ and $t = 100$ for $\tau_0 = 1$ and various values of $\mu$ are presented in Figure 1.11.
Figure 1.11: Example CCDFs for a Mittag-Leffler distribution
Chapter 2

Data Analysis

2.1 Data Collection and Description

In the original data - both for the primary and high school students - the participants were equipped with sensors that deemed them ‘in contact’ if they were within 1 to 1.5m of each other (an interaction), chosen by the researchers of the original study. This was to act as a proxy of a close-range encounter during which a communicable disease infection can be transmitted, for example, either by cough or sneeze, or directly by hand contact [125]. Every 20s, a radio packet would be exchanged between the sensors, and all packets transferred would be relayed to a central system to be recorded. This timescale was deemed to allow an adequate description of person-to-person interactions that includes brief encounters [125].

In both cases, this central system saved the data in a CSV file, with each row containing the timestamp (in 20s intervals), the IDs of the two sensors in contact, and some additional data about the two participants (such as their class). I modify the original data slightly before my initial analysis. Firstly, I remove any participants marked as staff from the data as their behaviour could be potentially anomalous when compared to that of the school children. Whilst I acknowledge that this could remove any potential impact of staff on the behaviour of pupils, I feel justified in this as staff only account for 11 participants and approximately 5% of the originally recorded links, which may prove problematic in terms of drawing any statistically significant conclusions about potential behaviour. In future work, including this additional layer may lead to an improved model - however, I feel that more data describing these interactions would be needed before I could confidently add this to a model. I also split the students into their separate classes. Whilst this results in the discarding of approximately 20% of the originally recorded links, this gives me more samples to analyse; moreover, it allows for a statistical comparison between the dynamics of different classes. From a more practical perspective, this restriction to classes has a considerable impact on the runtime of the model simulations (reducing this size from around 500 students to around 25).

The choice to restrict to classes is also justified from a modelling perspective as it is realistic to assume (at least as an initial hypothesis) that contacts outside of the classroom (during break/lunch) would follow substantially different behaviour.

I also split the data into individual days - similarly to splitting by class, this helped reduce the runtime of the simulation as well as increasing the number of samples I could analyse. Again, this is not unrealistic, as the interactions between
students in the same class can reasonably be assumed to be similar from one day to another.

2.1.1 Analysis of Original Data

A series of MATLAB functions\(^1\) were written to take these (separated) CSV files and perform an analysis of a variety of network and temporal features, and attempt to do best-fit analysis on all appropriate results - a full list of these features below. Animations showing the network evolution over time were also produced.\(^2\)

I identified a variety of key features for analysis. As usual, many more features can be observed from the data, and indeed, in order to approximate a completely realistic model, many of these should be analysed and incorporated into more detailed models. The models I shall produce are just an initial step into understanding these social-interaction temporal networks, and I am only focusing on aspects that categorise and describe both the topology of the network and several temporal properties of the system. These features are presented in subsubsection 2.1.1.1 below, along with brief definitions of these terms.

2.1.1.1 Features Examined

**Active Nodes** The measure of active nodes at a given time \(t\) is defined as the number of pupils involved in at least one interaction at time \(t\), as a fraction of all pupils active during that day.

**Active Links** The measure of active links at a given time \(t\) is defined as the number of unique (undirected) pupil-pupil interactions at time \(t\), as a fraction of all possible links for that day, equal to \(\ell_{\text{max}} = N(N-1)/2\), where \(N\) is the number of pupils active during that day in the class under consideration.

**Node & Link Activity Potential** The activity potential of a node is defined as the number of activations involving that node, as a fraction of all node activations across the day \(^1\). I also define an analogue for links, defined as the number of activations of that link, as a fraction of all link activations across the day.

**Global Clustering Coefficient** The global clustering coefficient (GCC) at a given time \(t\) is defined as the ratio between the number of closed triplets and the number of connected triplets in the network \(^1\). That is, the ratio between the number of triangles in the network and the number of paths of length 2, that do not have a third edge connecting the end points.

**Node Degree** As defined previously, the degree of node \(n\) is the number of active links involving it \(^1\).

---

\(^1\)See appendices A.3 and A.4 for details.
\(^2\)See appendix A.8 for details.
Component Features  Defining a component as a maximal subset of nodes that are fully connected [61] (that is that a path can be found between any two nodes in the subset), I can also examine properties such as component count and nodes and links per component at a given time \( t \).

Initialisation Time  For each link, an initialisation time is measured - defined as the period of time it takes for that specific link to be activated for the first time.

On-Duration  For each link, on-durations are measured - defined as the period of time between the activation and deactivation of that link.

Off-Duration  For each link, off-durations are measured - defined as the period of time between the deactivation and reactivation of that link.

Interevent Time  The time between the activation of one link in the network and the activation between the next (possibly different) link within the network.

2.1.2 Methods of Comparison

As I do not have any explicit theories for the dynamics of any of the chosen properties, I shall test the data against a series of appropriate common probability distributions [134, pp. 899–917] and variations on these, representing a range of behaviours defined on the semi-infinite interval \([0, \infty)\). I will be using exponential, gamma, Rayleigh, log-normal, Mittag-Leffler, generalised Pareto and Weibull distributions. All of these will have best fit parameters chosen using three different methods - method of moments [33], maximum likelihood estimators (MLEs) [171], and the curve fitting tool in MATLAB (non-linear least squares) - and then compared to the empirical complementary cumulative distribution functions (ECCDFs) of the original data to determine which one is most optimal.

This comparison will be achieved by looking at a variety of empirical distribution function (EDF) statistics - Kolmogorov-D, Cramér-von-Mises, Kuiper, Watson, Anderson-Darling and modified versions of the Kullback-Leibler and Jensen-Shannon [190] - details of which are given below in subsubsection 2.1.2.1. These statistics and comparisons were chosen as they emphasise a wide varying range of properties of the distributions to be compared - with, for example, some being more sensitive to changes in the head and tail of the ECCDF, whilst others are more sensitive to changes in the middle. Finding a distribution that had ‘good’ values for all of these distances would indicate that it was a good fit across the entirety of the compared ECCDF.

2.1.2.1 EDF Statistics

Below, I briefly summarise the EDF statistics that I will be using in my comparisons. Here, I assume that I am given \( n \) values of data in non-descending order, such as in (2.1), and I am attempting to find the distance between this data and a known distribution with CDF \( F(x) \). Additionally I define the value \( z_i = F(x_i) \):

\[
x_1 \leq x_2 \leq \ldots \leq x_n.
\]  (2.1)
Kolmogorov-D  The Kolmogorov statistics as given in (2.2) \[190\], with the value of \( D \in [0, 1] \) in (2.2c) (known as the Kolmogorov-D or Kolmogorov-Smirnov distance) being used in the Kolmogorov-Smirnov test, capture the maximum distance between a given CDF and the empirical cumulative distribution function (ECDF) given by data and can be applied to both discrete and continuous variables:

\[
D^+ = \max_{1 \leq i \leq n} \left[ \frac{i}{n} - z_i \right].
\] (2.2a)

\[
D^- = \max_{1 \leq i \leq n} \left[ z_i - \frac{i - 1}{n} \right].
\] (2.2b)

\[
D = \max \left[ D^+, D^- \right].
\] (2.2c)

If the test data \( \{x_i\} \) is IID and drawn from the distribution being compared, then it would be expected that \( D \approx 0 \). The Kolmogorov statistics are known to be fairly conservative when used for comparing two distributions, with the related test often being seen as weaker than similar tests as it is devised to be sensitive against all possible types of differences between two distribution functions. This sensitivity and conservativeness is known to decrease as the sample size increases \[123\]. This noted however, the test actually exhibits poor sensitivity to deviations when these occur in the tails of the distribution \[124\]. As some of the distributions I will be testing against have heavy tails, this statistic will struggle to capture significant variations in the behaviours in this region - for example, it may struggle to distinguish between Mittag-Leffler and exponential data, despite clear differences in properties such as their MGFs.

A two-sample version of this test uses the two-sample Kolmogorov-Smirnov statistic, \( D_{n,m} \). This is given in (2.3) and measures the distance between two EDFs, \( F_{1,n} \) and \( F_{2,m} \), the first and second sample respectively \[153, 181\]:

\[
D_{n,m} = \sup_x \left| F_{1,n}(x) - F_{2,m}(x) \right|.
\] (2.3)

Cramér-von-Mises  The Cramér-von-Mises statistic, \( \omega^2 \), as given in (2.4) \[190\], is an alternative to the Kolmogorov-D statistic, although it does share some of the same faults:

\[
\omega^2 = \frac{1}{12n} + \sum_{i=1}^{n} \left[ z_i - \frac{2i - 1}{2n} \right]^2.
\] (2.4)

This statistic belongs to the class of quadratic EDF statistics, which measure the distance between a hypothesized distribution \( F \) and an ECDF \( F_n \) by (2.5), where \( \Psi(x) \) is a weighting function. For the Cramér-von-Mises statistic, this weighting function is \( \Psi(x) = 1 \) \[146\]:

\[
n \int_{-\infty}^{\infty} (F_n(x) - F(x))^2 \Psi(x) dF(x).
\] (2.5)

The Cramér-von-Mises test is still relatively conservative, although less so than the Kolmogorov-Smirnov test, and performs relatively well in normal and light-tailed distributions \[123\]. However, comparisons in the case of heavy-tailed data is still problematic as this test weights all data equally \[18, 53\].
The Kuiper statistic, $V$, as given in (2.6) [190], is defined based on the Kolmogorov statistics $D^+$ and $D^-$ as given by (2.2a) and (2.2b) respectively:

$$V = D^+ + D^-.$$  (2.6)

This slight modification makes this test more sensitive to the tails of distributions [202], important here as some of the distributions will be heavy-tailed. Additionally, this statistic is invariant to changes of scale and cyclic transformations [202]. I will not be relying on this invariant property in this case, as I have split the data into relatively small blocks - however, this property may be important in future work when multiple days of data are considered together as it can test the fit of polar distributions so may reveal variations and similarities in behaviour that occur on a day-to-day basis.

The Watson statistic, $U^2$, as given in (2.7) [190], modifies the Cramér-von-Mises statistic, $\omega^2$:

$$U^2 = \omega^2 - n \left( \sum_{i=1}^{n} \frac{z_i}{n} - \frac{1}{2} \right)^2.$$  (2.7)

This has many similar properties to the statistic on which it is based, but introduces similar invariants to those in the Kuiper statistic [15].

The Anderson-Darling statistic, $A^2$, as given in (2.8) [190], is another member of the family of quadratic EDF statistics. In this case, the weighting function in (2.5) is given by (2.9):

$$A^2 = -n \left( \sum_{i=1}^{n} (2i - 1) \ln \frac{z_i}{n} + \ln (1 - z_{n+1-i}) \right) - n.$$  (2.8)

$$\Psi(x) = \frac{1}{F(x) (1 - F(x))}.$$  (2.9)

This difference in the weighting function makes the Anderson-Darling statistic much more sensitive to data in the tail regions of the distributions (that is where $F(x) \approx 0$ or 1) than the Cramér-von-Mises test where the weighting function $\Psi(x) = 1$ results in all data being weighted equally.

The Kullback-Leibler divergence (also known as the relative entropy), $D_{KL}$, between two distributions $P$ and $Q$, as given in (2.10) [120], is an asymmetric measurement between two distributions. It measures the information ‘lost’ when using $Q$ instead of $P$, or in terms of Bayesian analysis, is a measure of the information gained when revising from a prior distribution $Q$ to a posterior $P$:

$$D_{KL} (P||Q) = \sum_x P(x) \log \left( \frac{P(x)}{Q(x)} \right).$$  (2.10)
Jensen-Shannon The Jensen-Shannon divergence, $JSD$, between two distributions $P$ and $Q$, as given in (2.11) [71], is a symmetric, bounded and smoothed version of the Kullback-Leiber divergence. It has been applied in many fields, including bioinformatics [177], social sciences [50] and machine learning [79]. The square root of this divergence is a metric, referred to as the Jensen-Shannon distance [2]:

$$JSD(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M), \quad (2.11)$$

where

$$M = \frac{1}{2}(P + Q). \quad (2.12)$$

2.2 Results of Analysis

Due to their expansive nature, tables of results for this section are presented in supplemental materials. For the models introduced in the next chapter, there are certain key measurements I will be using - the initialisation time, $X_{ij}^{Init}$, the on-duration, $X_{ij,n}^{On}$, and the off-duration, $X_{ij,n}^{Off}$ - here, $i$ and $j$ are the endpoints of that specific link and $n$ refers to the $n$-th activation or deactivation of said link. Additionally, I shall be using the interevent times as defined in subsection 2.1.1.1.

From my data analysis, it was decided that the on-durations would be exponentially distributed and the off-durations would be log-normally distributed. From further data analysis, it was decided that the exponential distribution from which the on-durations were taken would have itself a random parameter. The parameter for the exponential distribution from which the on-durations were taken was chosen to come from a log-normal distribution. Additionally, an exponential distribution was chosen for the initialisation phase. Analysis also lead to the decision to take interevent times from a log-normal distribution.

There is a potential explanation for the uniformity of the off-duration parameters against the variability of the on-duration parameters. For the off-durations my analysis indicates the standard assumption of uniform mixing holds - this would result in the expected time between any two individuals breaking and re-establishing contact being independent of the individuals in question. However, a degree of preferential attachment is implied - if an individual comes into contact with an individual they have a preference for, they will seek to extend their contact as long as possible.

Blanking all parameters, these measurements have the following chosen distributions:

- Initialisation Time: $X_{ij}^{Init} \sim \text{Exp}(\lambda)$
- On-Duration: $X_{ij,n}^{On} \sim \text{Exp}(Y_{ij})$
- On-Duration Parameter: $Y_{ij} \sim \text{LogNormal}(\mu_1, \sigma_1)$

3Available from https://drive.google.com/open?id=13eUXp3xGph1B5T7d1IPvEVPZcHyACmom
4See appendix A.3 for details.
5See appendix A.4 for details.
\begin{table}
\centering
\begin{tabular}{|l|c|c|}
\hline
Parameter & Mean & Variance \\
\hline
$X_{ij}^{init}$ & $6.2780 \times 10^{3}$ & $4.4029 \times 10^{7}$ \\
$X_{ij,n}^{On}$ & 33.2949 & $1.2254 \times 10^{3}$ \\
$Y_{ij}$ & 35.6676 & 104.3023 \\
$X_{ij,n}^{Off}$ & $1.4626 \times 10^{3}$ & $1.1791 \times 10^{7}$ \\
t & 2.4557 & 30.2956 \\
\hline
\end{tabular}
\caption{Mean and variance of observed parameters}
\end{table}

\begin{table}
\centering
\begin{tabular}{|l|c|}
\hline
Parameter & Value \\
\hline
$\lambda$ & $6.2780 \times 10^{3}$ \\
$\mu_1$ & 3.5348 \\
$\sigma_1$ & 0.2807 \\
$\mu_2$ & 6.3512 \\
$\sigma_2$ & 1.3688 \\
$\mu_3$ & $5.6901 \times 10^{-4}$ \\
$\sigma_3$ & 1.7957 \\
\hline
\end{tabular}
\caption{Chosen parameters}
\end{table}

- Off-Duration: $X_{ij,n}^{Off} \sim \text{LogNormal} (\mu_2, \sigma_2)$
- Interevent Time: $t_i \sim \text{LogNormal} (\mu_3, \sigma_3)$

Here, values are IID as appropriate - that is $X_{ij}^{init}$ (similarly $Y_{ij}$) are IID for all $i, j$, and $X_{ij,n}^{On}$ (similarly $X_{ij,n}^{Off}$) are IID for each $(i, j)$ pair and all $n$.

Examining data from primary schools, I obtain the summary statistical results presented in Table 2.1.

Using the formula for the parameters of a log-normal distribution from its mean and variance, and the properties of the exponential distribution, I therefore settle upon the choices for the parameters given in Table 2.2.

This gives the final distributions for these parameters as follows:

- Initialisation Time: $X_{ij}^{init} \sim \text{Exp}(6278.0)$
- On-Duration: $X_{ij,n}^{On} \sim \text{Exp} (Y_{ij})$
- On-Duration Parameter: $Y_{ij} \sim \text{LogNormal}(3.5348, 0.2807)$
- Off-Duration: $X_{ij,n}^{Off} \sim \text{LogNormal}(6.3512, 1.3688)$
- Interevent Time: $t_i \sim \text{LogNormal}(5.6901 \times 10^{-4}, 1.7957)$
Chapter 3
Model Development

3.1 Introduction

The aim of my model is to recreate the dynamics seen in the original data with as few properties and parameters taken from the original data as possible. In more precise terms, I wish to test if the mechanism of interactions within the original data can be explained by a small number of key factors and identify and refine those parameters. As with any model, I doubt that I will be able to replicate every property in the original data, but it is important to examine the differences between my model and the original data, and to quantitatively evaluate the distance between the two. Whilst there will be some properties that I will be controlling, there will be several network and temporal properties that emerge from my model that I can compare to the original data, hence giving a measure of the distance between the two.

3.1.1 Sample Comparison

When creating models, I aim to have little dependence on the original data - varying parameters only between differing settings (primary school vs. high school), rather

<table>
<thead>
<tr>
<th></th>
<th>1%</th>
<th>5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Links</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>Active Nodes</td>
<td>139</td>
<td>125</td>
</tr>
<tr>
<td>Node Activity Potential</td>
<td>190</td>
<td>185</td>
</tr>
<tr>
<td>Global Clustering Coefficient</td>
<td>67</td>
<td>46</td>
</tr>
<tr>
<td>Interaction Time</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>Time Between Contacts</td>
<td>30</td>
<td>19</td>
</tr>
<tr>
<td>Component Count</td>
<td>113</td>
<td>94</td>
</tr>
<tr>
<td>Links per Component</td>
<td>62</td>
<td>50</td>
</tr>
<tr>
<td>Nodes per Component</td>
<td>63</td>
<td>54</td>
</tr>
<tr>
<td>Triangle Count</td>
<td>114</td>
<td>99</td>
</tr>
</tbody>
</table>

Table 3.1: Acceptances of $\mathcal{H}_0$ (see equation 3.1) at 1% and 5% Levels (max: 190) (see subsection 3.1.1 for full explanation)
than within these settings. For example, I would aim to have the parameters for the random variable generation for the model for class 5A in the primary school to be the same as those in the model for class 1B of the primary school. Therefore, the first statistical test will be to test the validity of this statement. The $H_0$ is:

$$H_0 : \text{The two observed samples come from a common distribution.}$$ (3.1)

I compute two-sample Kolmogorov-Smirnov distances [153, 181] between each of the data sets within each setting.

I present the number of acceptances of this hypothesis (out of 190) for the primary school data samples at the 1 and 5 percent levels in Table 3.1. Examining these results, I conclude that while there is not a unanimous degree of acceptances for $H_0$, I have a substantial number in some metrics and a notable level in others. Other metrics have a very low degree of matching - most noticeably in terms of active links and interaction times. Whilst this is not ideal for my aim to only vary parameters between scenarios, for brevity I shall still proceed under this assumption - although it should be noted that when I present the models I do not actually fix the parameter in the distribution for the interaction times. Instead I draw this parameter from a random distribution itself, which reflects the behaviour in the data originally collected by the SocioPatterns Collaboration.

### 3.2 Model Creation

Two different model types will be proposed. The first model assigns on-off durations to each link from an appropriate probability distribution. The second model triggers activations at appropriate times (with inter-event times being drawn from an appropriate distribution), before selecting the link to be activated (using a probability matrix drawn from the original data) and assigning an on-duration to that link from an appropriate probability distribution. Even if these models do not capture all the important features of the real-world network, they still provide a useful first approximation. This is highly important due to the limited number of approaches to generating temporally-varying random networks as discussed in section 1.2 - although the methods I present here may not be a full solution to his problem, they represent a more organic form of generating these networks and may provide a basis for future research in this direction. Whilst I focus on school classrooms, this approach can be adapted to modelling other types of social interactions.

#### 3.2.1 Model 1

For this stage-0 model I look at each (potential) link individually and model its behaviour as an alternating renewal process (ARP) [25]. I also include an initialization phase for each link that models the time (in seconds) until the first activation of that link. This can be seen as the following process for each link where $X_{ij,n}$ represents duration of the $n$-th on (or off) phase for the link $(i,j)$, with the distributions chosen using an empirical analysis of the data. Algorithmically, I present this as:

1. **Initialization Phase:** Generate the initialisation time for this link with

$$X_{ij}^{\text{Init}} \sim \text{Exp}(6278.0)$$

1See appendix A.6 for details.
2. **ARP On-Phase:** Assign the link the on-duration

\[ X_{ij,n}^{\text{On}} \sim \text{Exp} (Y_{ij}) \]

with parameter fixed for each \((i, j)\) to

\[ Y_{ij} \sim \text{LogNormal}(3.5348, 0.2807). \]

3. **ARP Off-Phase:** Assign the link the off-duration as

\[ X_{ij,n}^{\text{Off}} \sim \text{LogNormal}(6.3512, 1.3688). \]

4. **Repeating Process:** Repeat Stages 2 and 3 until the total time has reached or exceeded the simulation time.

The MATLAB code for this model and the method of link selection can be found in the appendices\(^2\).

### 3.2.1.1 The Convolution Problem

In terms of components and failure-times, an ARP will have two components with PDFs \(f(x)\) and \(g(x)\), respectively. The system starts with one of these components (say Component I) and any component is immediately replaced on failure with a component of the opposite type. Assume the failure times for Component I are given by \(\{X_1, X_2, \ldots\}\) and the failure times for Component II are given by \(\{Y_1, Y_2, \ldots\}\).

In the model above, Component I can be seen as the period in which two designated students are in contact, and Component II can be seen as the period that these students are not in contact. By combining the two components together (with failure times \(Z_i = X_i + Y_i\), and PDF \(h(x)\)), this will collapse to a normal renewal process, and will allow us to calculate standard properties associated with renewal processes.

There are also certain properties of an ARP that can be directly calculated using convolutions of the on- and off- distributions (in this case, an exponential and log-normal distribution). The one of these that I am most interested in is the number of new contacts between two students within a given time - in effect the number of Type II failures in the ARP. If I start with this link being active, this shall be defined as \(N_t\), and if I start with the link being inactive, this shall be defined as \(N'_t\).

I shall focus on the first of these, as the second requires nothing more than a simple modification [164].

\[
\begin{align*}
\Pr(N_t = n) &= \Pr\left(\sum_{m=1}^{\infty} \sum_{i=1}^{m} Z_i \leq t = n\right) \\
&= \Pr\left(\left\{\sum_{i=1}^{n} Z_i \leq t\right\} \cap \left\{\sum_{i=1}^{n+1} Z_i > t\right\}\right) \\
&= \Pr\left(\left\{\sum_{i=1}^{n} Z_i \leq t\right\} \cap \left\{Z_{n+1} > t - \sum_{i=1}^{n} Z_i\right\}\right) \\
&= \int_{0}^{t} f_{\sum_{i=1}^{n} Z_i}(w) \left(\int_{t - w}^{\infty} f_{Z_{n+1}}(u)\,du\right)\,dw
\end{align*}
\]

---

\(^2\)See appendices B.55 and B.22 for details.
Figure 3.1: PDF produced using Convolve (bar) vs. PDF from Monte-Carlo Simulation (line) for $N_{2000}$

$$
P(N_{2000} = n) = \int_{0}^{t} f_{\sum_{i=1}^{n} Z_{i}(w)} \left( 1 - F_{Z_{n+1}}(t - w) \right) dw
= \int_{0}^{t} \left( h \ast h \ast \ldots \ast h \right)_{n-times}(w) (1 - H(t - w)) dw
$$

Here, $h(x) = (f \ast g)(x)$ is the PDF of $X + Y$ and $H(x)$ is its associated CDF. For $P(N'_{t} = n)$, it should be clear that this only needs to be slightly modified to ‘add in’ the extra distribution, giving (3.2):

$$
P(N'_{t} = n) = \int_{0}^{t} \left( g \ast h \ast h \ast \ldots \ast h \right)_{n-times}(w) (1 - H(t - w)) dw. \quad (3.2)
$$

As part of creating the distributions involved in the ARP, I need to sum the relevant on- and off- distributions, and hence I need to convolve an exponential and a log-normal distribution. It is possible to either do this directly, using (3.3) or using the Laplace transformations and (3.4):

$$
h(z) = (f \ast g)(z) = \int_{-\infty}^{\infty} f(z - t)g(t)dt = \int_{-\infty}^{\infty} f(t)g(z - t)dt / (3.3)
$$

$$
\tilde{h}(s) = (f \ast g)(s) = \tilde{f}(s)\tilde{g}(s). \quad (3.4)
$$

However, no closed form solution for the Laplace transformation of the log-normal distribution (or the equivalent integral in the first method) exists - although several approximations do [17, 129]. Instead, numerical simulations using the convolve function in MATLAB shall be used in any calculations of $N(t)$ and $N'(t)$. Whilst this method is not perfect, it does produce fairly accurate results when tested against sufficiently large Monte Carlo simulations as can be seen in Figure 3.1

---

3See appendix A.9 for details.
3.2.2 Model 2a

In this stage-0 model, I will be dealing with the system on a macroscopic basis. I am drawing times between activations from an appropriate distribution, then at each of these activations, a link is chosen at random from a custom distribution constructed from the link activity potentials extracted from the data and represented by a symmetric weighting matrix $M$. This $M$ is constructed such that $M_{ij}$ is equal to the activations of the link from $i$ to $j$ as a proportion of the total number of link activations in the network across the entire time period. If the chosen link is already active in the network, this selection is discarded, and another link is chosen for that activation time. Once a link has been activated, it is given a lifespan from an appropriate distribution. This can be seen as the following process, with the distributions chosen using an empirical analysis of the data. Algorithmically, I present this as:

1. **Time between Activations**: Generate
   \[ t_i \sim \text{LogNormal}(5.6901 \times 10^{-4}, 1.7957). \]

2. **Link Activation**: At each activation time $T_k$, defined as
   \[ T_k = \sum_{i=0}^{k} t_i, \]
   a link $(n_1, n_2)$ is chosen using the relative weights in the matrix $M$. If $(n_1, n_2)$ is already active at time $T_k$, choose another link for this time $(n_1', n_2')$ by the same method.

3. **Assign On-Durations**: This link is given the duration
   \[ X^{k}_{n_1 n_2} \sim \text{Exp}(Y_{n_1 n_2}) \]
   as before with parameter fixed for each $(n_1, n_2)$ to
   \[ Y_{n_1 n_2} \sim \text{LogNormal}(3.5348, 0.2807). \]

The MATLAB code for this model can be found in the appendices\(^4\).

3.2.3 Model 2b

In this model, I modify the Model 2a and attempt to improve triangle count and clustering. Most of the method is similar to the earlier model, but I force chosen links to close a pair of links into a triangle at a fixed rate, reweighting the selection matrix to only account for these links (if no such links exist, I use the original selection matrix), before proceeding as before with this link selected. This can be seen as the following algorithm, with the distributions always chosen using an empirical analysis of the data:

1. **Time between Activations**: Generate
   \[ t_i \sim \text{LogNormal}(5.6901 \times 10^{-4}, 1.7957). \]

---

\(^4\)See appendix B.59 for details.
2. **Triangulation Bias:** Generate a random number $u$ such that

$$u \sim \text{Unif}[0,1].$$

If $u \geq 0.0640$ (the ‘forcing’ rate, calculated from the data\footnote{See appendix A.11 for details.}), proceed to Stage 3a, else proceed to Stage 3b.

3. **Link Activation:**

   (a) **Standard Activation:** At each activation time $T_k$, defined as

   $$T_k = \sum_{i=0}^{k} t_i,$$

   a link $(n_1, n_2)$ is chosen using the relative weights in the matrix $M$. If $(n_1, n_2)$ is already active at time $T_k$, choose another link for this time $(n'_1, n'_2)$ by the same method. Proceed to Stage 4.

   (b) **Triangle-Biased Activation:**

   i. **Matrix Reweighting:** Generate the (symmetric logical) matrix $C$ of links that will complete triangles. That is $C_{ij} = 1$ if there is some node $k$, such that links from $i$ to $k$ and $k$ to $j$ exist. If no such node $k$, then $C_{ij} = 0$. If this matrix is 0, set $C = I$. Create the adjusted weighted matrix $M'$ where $M'_{ij} = C_{ij} M_{ij}$.

   ii. **Link Activation:** At each activation time $T_k$, defined as

   $$T_k = \sum_{i=0}^{k} t_i,$$

   a link $(n_1, n_2)$ is chosen using the relative weights in the adjusted matrix $M'$. If $(n_1, n_2)$ is already active at time $T_k$, choose another link for this time $(n'_1, n'_2)$. Proceed to Stage 4.

4. **Assign On-Durations:** This link is given the duration

   $$X_{n_1n_2}^k \sim \text{Exp}(Y_{n_1n_2})$$

   as usual with parameter fixed for each $(n_1, n_2)$ to

   $$Y_{n_1n_2} \sim \text{LogNormal}(3.5348, 0.2807).$$

The MATLAB code for this model and the methods for extracting the matrix $M$ and selecting a link can be found in the appendices\footnote{See appendices B.37, B.34 and B.22 for details.}.
3.2.4 Model 2c

I shall again build upon the previous model - Model 2b - this time changing the matrix $M$. Previously, this has been a fixed matrix extracted from the data, but I wish to move to a randomly generated one to reduce this strict dependency on the original data. Based on an examination of these (symmetric) matrices, I will attempt to generate such a matrix using the row (or column) sums - which is equivalent to the node activity potentials - which I will attempt to find a distribution for. This approach supposes that the popularity of a link in the matrix is dependent upon the popularity of the individuals involved in such a link, which I believe to be a reasonable assumption.

From an analysis of the data\textsuperscript{7} available in the supplemental materials\textsuperscript{8} I choose an appropriate distribution for these sums - I shall use row sums

$$M_{i\Sigma} = \sum_{j=1}^{n} M_{ij} \sim \Gamma(12.3109, 0.0037).$$

For this attempt at generating an appropriate random matrix $M$, I shall assume that each term is taken from a gamma distribution with

$$M_{ij} \sim \Gamma(k_{ij}, 0.0037) + \Gamma(k_{ji}, 0.0037)$$

for $i < j$, $M_{ij} = 0$ for $i = j$ and $M_{ij} = M_{ji}$ for $i = j$. Due to the additive properties of the gamma distribution, this is equivalent to the distribution

$$M_{ij} \sim \Gamma(k_{ij} + k_{ji}, 0.0037)$$

for $i < j$, $M_{ij} = 0$ for $i = j$ and $M_{ij} = M_{ji}$ for $i > j$. All entries where $i \neq j$, must therefore take the form

$$M_{ij} \sim \Gamma(k_{ij}, 0.0037),$$

for some $k_{ij}$. This choice of distribution is fairly elementary, as it assumes the scale is consistent across all entries. However, it allows a simple dependency on both individuals involved in the link as well as allowing for more detailed results due to the additive nature of the distribution. I also construct this in such a way that the choice of a self-loop is impossible, whilst also ensuring symmetry (which is to be expected as the network is undirected). No dependency is assumed between the values of $k_i$ at this stage.

As I wish to specify the $k_i$, I shall use the additive property of the gamma distribution and the required symmetry of the matrix to form a system of linear equations that may be solved for these values. From the choice of distribution for row sums and the additive property of the distribution, (3.5) must hold:

$$M_{i\Sigma} = \sum_{j=1}^{n} M_{ij} \sim \sum_{j=1}^{n} \Gamma(k_{ij}, 0.0037) = \Gamma \left( \sum_{j=1}^{n} k_{ij}, 0.0037 \right)$$

$$= \Gamma(k_{i\Sigma}, 0.0037) = \Gamma(12.3109, 0.0037).$$

\textsuperscript{7}See appendix A.3 for details.

\textsuperscript{8}Available from https://drive.google.com/open?id=13eUXp3xGph1B5T7dIIPvEWPZcHyACmom.
Thus to identify the individual parameters, I wish to solve the system (3.6):

$$\begin{align*}
k_i \Sigma &= \sum_{j=1, j \neq i}^{n} k_{ij} = 12.3109, \\
i &= 1, 2, \ldots, n.
\end{align*}$$

(3.6)

To account for symmetry restrictions governing $k_{ij}$, I will approach it as follows:

To sum across a row, I first add the entries to the right of the diagonal, which is equal to

$$(n - i)k_i + \sum_{j=i+1}^{n} k_j.$$  

I then notice that the entries to the left of the diagonal, are equal to the column sum to the diagonal, equal to

$$\sum_{j=1}^{i-1} k_j + (i - 1)k_i,$$

giving the total sum to be

$$k_i \Sigma = (n - 1)k_i + \sum_{j \neq i} k_j = (n - 1)k_i + \sum_{j} k_j - k_i = (n - 2)k_i + \sum_{j} k_j.$$  

To match the distributions for the row sums, I require that:

$$(n - 2)k_1 + \sum_{j} k_j = (n - 2)k_2 + \sum_{j} k_j = (n - 2)k_3 + \sum_{j} k_j = \ldots$$

$$= (n - 2)k_{n-1} + \sum_{j} k_j$$

$$= (n - 2)k_n + \sum_{j} k_j = 12.3109.$$  

This system gives that $k_i = k^* \ \forall i \in \{1, 2, \ldots, n\}$. Returning to the system of equations, the sum of these can be examined, giving:

$$\sum_{i} k_i \Sigma = \sum_{i} \left( (n - 2)k_i + \sum_{j} k_j \right)$$

$$= \sum_{i} ((n - 2)k_i) + n \sum_{j} k_j$$

$$= (n - 2) \sum_{i} k_i + n \sum_{j} k_j$$

$$= 2(n - 1) \sum_{i} k_i$$

$$= 2(n - 1) \sum k^* = 2(n - 1)nk^* = n(12.3109).$$
Thus $k^* = 12.3109/2(n - 1)$, and the initial model for a randomly generated symmetric $M$ shall be with

$$M_{ij} \sim \Gamma \left( \frac{12.3109}{2(n - 1)}, 0.0037 \right)$$

for $i < j$, $M_{ij} = 0$ for $i = j$ and $M_{ij} = M_{ji}$ for $i > j$.

Whilst the approach I have taken is somewhat simplistic, I believe that the inclusion of this method is an important step in reducing the level of dependency on the original data and allows me to examine behaviours and test mechanics when using random matrices. In future work, non-linear symmetric combinations of $k_i$ and $k_j$ with non-trivial solutions, such as $k_i + k_i k_j + k_j$, may be studied - this example gives the system (3.7):

$$\begin{align*}
(k_i \Sigma = (n - 2)k_i - k_i^2 + \sum_j (1 + k_i) k_j = 12.3109)_{i=1,2,\ldots,n}.
\end{align*}$$

(3.7)

However, such combinations shall not be considered here due to the complexity of identifying physically appropriate formulae and solutions to the corresponding equation systems. The MATLAB code for this model, the generation of the matrix $M$ and the method of link selection can be found in the appendices.9

---

9See appendices B.57, B.74 and B.22 for details.
### Table 3.2: Summary of model dependencies (see subsection 3.2.5 for full explanation, subsection 3.2.2 for the definitions of $M$ and subsection 3.2.3 for the definitions of $u_f$)

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameters</th>
<th>Para. Values</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$X_{ij}^{\text{init}} \sim \text{Exp}(\lambda)$</td>
<td>6278.0</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>$Y_{ij} \sim \text{LogNormal}(\mu_1, \sigma_1^2)$</td>
<td>(3.5348, 0.2807)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$X_{ij,n}^{\text{init}} \sim \text{LogNormal}(\mu_2, \sigma_2^2)$</td>
<td>(6.3512, 1.3688)</td>
<td></td>
</tr>
<tr>
<td>2a</td>
<td>$t_i \sim \text{LogNormal}(\mu_1^2, \sigma_1^2)$</td>
<td>$(5.6901 \times 10^{-4}, 1.7957)$</td>
<td>4 $+ \frac{n(n-1)}{2}$</td>
</tr>
<tr>
<td></td>
<td>$Y_{n_1n_2} \sim \text{LogNormal}(\mu_2, \sigma_2^2)$</td>
<td>(3.5348, 0.2807)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$M \sim n \times n$ symmetric matrix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2b</td>
<td>$t_i \sim \text{LogNormal}(\mu_1^2, \sigma_1^2)$</td>
<td>$(5.6901 \times 10^{-4}, 1.7957)$</td>
<td>5 $+ \frac{n(n-1)}{2}$</td>
</tr>
<tr>
<td></td>
<td>$Y_{n_1n_2} \sim \text{LogNormal}(\mu_2, \sigma_2^2)$</td>
<td>(3.5348, 0.2807)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$u \geq u_f$ (the ‘forcing’ rate)</td>
<td>0.0640</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$M \sim n \times n$ symmetric matrix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2c</td>
<td>$t_i \sim \text{LogNormal}(\mu_1^2, \sigma_1^2)$</td>
<td>$(5.6901 \times 10^{-4}, 1.7957)$</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>$Y_{n_1n_2} \sim \text{LogNormal}(\mu_2, \sigma_2^2)$</td>
<td>(3.5348, 0.2807)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$u \geq u_f$ (the ‘forcing’ rate)</td>
<td>0.0640</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$M_{ij} \sim \Gamma(k, \theta)$</td>
<td>$(\frac{12.3109}{2(n-1)}, 0.0037)$</td>
<td></td>
</tr>
</tbody>
</table>

### 3.2.5 Summary

In Table 3.2 I present a concise comparative summary of the data dependencies of each of the 4 model variants. With all models, a balance must be established between the number of inputs and the accuracy of outputs - a model that describes behaviour extremely accurately, but requires a significant amount of input, is equally as poor as one that requires little input, but returns inaccurate results. For most of these models, I feel as though the parameter count is acceptable considering the complexities of the behaviours I am attempting to capture. In Models 2a and 2b, the parameter count is much higher than reasonable due to the explicit dependence on the original data, suggesting that these would not be ideal models to fully implement - however they are included in the analysis in order to allow me to observe the accuracy of Model 2c.

### 3.3 Model Analysis

Please note, in the figures highlighting key results, simulated data is represented by crosses whereas observed data is represented by dotted lines, with the data displayed as an ECCDF with log-log axes (with scaling preserved between models). Each colour represents a different simulation or data set. In order, the four EC-CDFs shown represent active nodes, node activity potentials, component counts and the GCCs. I chose these metrics to illustrate as they represent both promising behaviours and less-optimal ones, thereby giving a representative snapshot of my results. Additionally, these ECCDFs are some of the clearer and easier ones to read,
<table>
<thead>
<tr>
<th></th>
<th>Node Activity Potential</th>
<th>Global Clustering Coefficient</th>
<th>Component Count</th>
<th>Links per Component</th>
<th>Nodes per Component</th>
<th>Triangle Count</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong></td>
<td>min 0.1304</td>
<td>0.02517</td>
<td>0.01813</td>
<td>0.009406</td>
<td>0.006316</td>
<td>0.02394</td>
</tr>
<tr>
<td></td>
<td>max 0.5769</td>
<td>0.2876</td>
<td>0.6083</td>
<td>0.07935</td>
<td>0.07935</td>
<td>0.2876</td>
</tr>
<tr>
<td></td>
<td>mean 0.3365</td>
<td>0.1191</td>
<td>0.2895</td>
<td>0.03934</td>
<td>0.03856</td>
<td>0.1189</td>
</tr>
<tr>
<td></td>
<td>mode 0.3043</td>
<td>0.03524</td>
<td>0.01813</td>
<td>0.009406</td>
<td>0.006316</td>
<td>0.03524</td>
</tr>
<tr>
<td><strong>2a</strong></td>
<td>min 0.08696</td>
<td>0.01835</td>
<td>0.03448</td>
<td>0.004412</td>
<td>0.00467</td>
<td>0.002869</td>
</tr>
<tr>
<td></td>
<td>max 0.4249</td>
<td>0.2677</td>
<td>0.6067</td>
<td>0.09015</td>
<td>0.09015</td>
<td>0.2641</td>
</tr>
<tr>
<td></td>
<td>mean 0.2212</td>
<td>0.09876</td>
<td>0.308</td>
<td>0.04169</td>
<td>0.04195</td>
<td>0.06908</td>
</tr>
<tr>
<td></td>
<td>mode 0.1739</td>
<td>0.01835</td>
<td>0.03448</td>
<td>0.004412</td>
<td>0.00467</td>
<td>0.002869</td>
</tr>
<tr>
<td><strong>2b</strong></td>
<td>min 0.08</td>
<td>0.02513</td>
<td>0.03112</td>
<td>0.004466</td>
<td>0.003064</td>
<td>0.005029</td>
</tr>
<tr>
<td></td>
<td>max 0.4377</td>
<td>0.2065</td>
<td>0.5274</td>
<td>0.08203</td>
<td>0.08203</td>
<td>0.2011</td>
</tr>
<tr>
<td></td>
<td>mean 0.2241</td>
<td>0.08137</td>
<td>0.2838</td>
<td>0.0373</td>
<td>0.03754</td>
<td>0.05605</td>
</tr>
<tr>
<td></td>
<td>mode 0.2273</td>
<td>0.06793</td>
<td>0.03112</td>
<td>0.004466</td>
<td>0.003064</td>
<td>0.005029</td>
</tr>
<tr>
<td><strong>2c</strong></td>
<td>min 0.08696</td>
<td>0.03049</td>
<td>0.03836</td>
<td>0.005304</td>
<td>0.006754</td>
<td>0.003863</td>
</tr>
<tr>
<td></td>
<td>max 0.4945</td>
<td>0.2036</td>
<td>0.552</td>
<td>0.08004</td>
<td>0.08004</td>
<td>0.1842</td>
</tr>
<tr>
<td></td>
<td>mean 0.2485</td>
<td>0.07893</td>
<td>0.3087</td>
<td>0.04149</td>
<td>0.04166</td>
<td>0.04898</td>
</tr>
<tr>
<td></td>
<td>mode 0.2273</td>
<td>0.07942</td>
<td>0.03836</td>
<td>0.005304</td>
<td>0.006754</td>
<td>0.006151</td>
</tr>
</tbody>
</table>

Table 3.3: Selected two-sample Kolmogorov-Smirnov distances (see section 3.3 for full explanation)
allowing me to demonstrate a number of behaviours in a brief and compact manner. It should be noted that in some cases (most apparent in the case of the GCCs) some of these ECCDFs appear not to start at 1 as expected - this is a result of a high prevalence of the value 0 in the data, with a large jump between this and other values. For readability, this jump has been excluded from the graphics, with the images only showing the section of the graph where the majority of the values fall. Original graphics for all properties (included those omitted here) are available in the supplemental materials.\footnote{\url{https://drive.google.com/open?id=13eUXp3zGph1B5T7dILPvEVPZcHyACmom}}

I also present comparative data in two tables. In Table \ref{tab:comp}, I show a summary of the two-sample Kolmogorov-Smirnov distances - see \eqref{eq:KSD} in subsection 2.1.2.1 - between the collection of 20 empirical samples and 20 simulated data samples from each of the 4 models presented, showing the minimum, maximum, mean and mode of the distance between any of the 20 sets of real world data and any of the 20 sets of generated data. I also compare horizontally, comparing each empirical data set against 50 data sets generated using my chosen metrics\footnote{See appendix A.5 for details.}. I test the hypothesis $H_0$, in this case, this is given in \eqref{eq:h0} below:

$$H_0: \text{The chosen empirical and generated data samples come from a common distribution.} \quad \text{(3.8)}$$

In Table \ref{tab:accepts} I present the total number of acceptances (out of a possible 1000) at the 5%-level of this hypothesis when tested on a particular metric.

Additionally, I present Figure \ref{fig:activity} to highlight long-term behaviours in the models.\footnote{See appendix A.8 for details.} In this figure, the transparency of each link represents its relative activity in comparison to other links, and the size of each node represents the relative activity of each node. The 5 images in this figure represent this behaviour at $t = 15000$ seconds for an example of the original data, Model 1, Model 2a, Model 2b and Model 2c.

<table>
<thead>
<tr>
<th>Active Links</th>
<th>Model 1</th>
<th>Model 2a</th>
<th>Model 2b</th>
<th>Model 2c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Nodes</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Node Activity Potential</td>
<td>954</td>
<td>1000</td>
<td>1000</td>
<td>999</td>
</tr>
<tr>
<td>GCC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Interaction Time</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Time Between Contacts</td>
<td>24</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Component Count</td>
<td>151</td>
<td>264</td>
<td>240</td>
<td>306</td>
</tr>
<tr>
<td>Links per Component</td>
<td>62</td>
<td>35</td>
<td>56</td>
<td>42</td>
</tr>
<tr>
<td>Nodes per Component</td>
<td>33</td>
<td>68</td>
<td>79</td>
<td>63</td>
</tr>
</tbody>
</table>

Table 3.4: Acceptances of $H_0$ (see equation \ref{eq:h0}) at 5% Level (max: 1000) (see section 3.3 for full explanation)
Figure 3.2: Long term behaviours for original data and models - Size of nodes & transparency of links represent relative activities (see last paragraph of the opening of section 3.3 for full explanation and the relevant subsections of section 3.3 and section 3.4 for an analysis of these results). One immediate observation is that Model 1 homogenises much faster - note the limited number of darker links.
2c. Using this figure, systemic behaviours can be seen, such as possible grouping of nodes into friendship groups or similar metrics that would be more difficult to measure empirically. This also gives me an intrinsic definition for link spread.

**Link Spread**  Link spread is a visually evaluated measure that combines several long-term network properties and their relationships. Such properties include the distribution of activity potentials and the relationship between the placement of highly active links and nodes, as well as the occurrence of certain subgraphs and other structures within the larger network.

Figure 3.2b demonstrates a poor spread - the long-term behaviour is relatively homogeneous with fewer darker links. Similarly, a simulation that resulted in long-term behaviour that only had darker links limited to a very small number of nodes would also suffer from poor spread. Comparatively, Figure 3.2a has a better spread - there are a higher number of darker links spread among a larger number of nodes. More precisely, this is measuring a combination of factors - including activity potentials, component structures and other network features - but gives an impression of many of these features at a glance. It is not expected to get a perfect matching between the examples here due to the randomness of the data, but I am instead looking for system-wide similarity in behaviour. Differences are expected in the placement of stronger links and nodes (and indeed, do occur between simulation runs). However, I would expect a well-fitting model to exhibit similar numbers to those in the original data and with a similar relationship between them (for example, as Figure 3.2a has many nodes being involved in at least one stronger link, a well-fitting model would not be expected to have all of its strong links emanating from a common node).

### 3.3.1 Model 1

Looking at Figure 3.3, the appropriate sections of Tables 3.3 and 3.4 and other comparative and graphical results included in the appendices and supplemental materials\(^\text{13}\) as a first attempt at creating a model, promising results can be seen. The model produces acceptable fits for several of the examined features. Active links, active nodes and on-durations all produce graphically acceptable results, although using the Kolmogorov-Smirnov acceptances (shown in Table 3.4), there are improvements to be made in terms of these fits. For off-durations, when the ECCDFs are examined, I observe a reasonable fit in certain areas of the distribution although this fit deteriorates for extreme values and once again I notice that my acceptances indicate that the current construction of this model requires refinement to fully capture this behaviour. For the GCC (presented in Figure 3.3) and triangle count, there are poor fits where comparing the data sets graphically, although I am getting a small number of acceptances with the two-sample Kolmogorov-Smirnov tests - likely as a result of an extreme prevalence of certain values in these data sets. For nodes per component, links per component and the component count (partially presented in Figure 3.3), I observe acceptable fits graphically and are indeed accepting a small

\(^{13}\) Available from [https://drive.google.com/open?id=13eUXp3xGph1B5T7d1IPvEVPZcHyACsom](https://drive.google.com/open?id=13eUXp3xGph1B5T7d1IPvEVPZcHyACsom)
Figure 3.3: Selected results for Model 1. Simulated data is represented by crosses whereas observed data is represented by dotted lines. Each colour represents a different simulation or data set. See subsection 3.3.1 for full explanation. Additional figures are available in the supplemental materials.
number of these fits when calculating the statistical distances, as shown in Table 3.4. This also indicates that slight refinement to this fit may be possible. For node activity potential, there is a good fit, both graphically and when considering the number of Kolmogorov-Smirnov acceptances.

It is evident that this model does have noticeable differences to the observed data. There are a substantial number of small linear components in the model, which is impacting many of the features described above. Additionally there are problems with link selection spread (defined in section sec:modelanalysis) as can be seen when comparing the original behaviour displayed in Figure 3.2a with that in Figure 3.2b, resulting in very few popular links (reflecting strong friendships), which could also explain differences within the node activity potentials at the tail of the CCDFs.

### 3.3.2 Model 2a

Considering Figure 3.4, the appropriate sections of Tables 3.3 and 3.4 and other results measured, a substantially improved model can be seen. As with Model 1,
I have results that appear graphically similar across the entirety or key sections of the distribution for active links, active nodes, GCC, on-durations and off-durations, whilst the Kolmogorov-Smirnov distances for these indicate that there are still improvements to the fits to be made here. For the triangle count, reasonable fits can be seen graphically and a higher number of the statistical comparisons are being accepted. Again, for nodes per component, links per component and the component count, I observe acceptable fits graphically (partially presented in Figure 3.4) and are indeed accepting a small number of these fits when calculating the statistical distances - overall a slightly higher number than in Model 1, but with only small variations in each one. For node activity potential, there is a very good fit, both graphically and when considering Kolmogorov-Smirnov distances. As can be seen when I compare Figure 3.2a and Figure 3.2c, an acceptable link selection spread (defined in section 3.3) is being produced, which reflects the varying levels of friendships observed in the real world data.

However, this model is insufficient to capture the related component structure - with the generated data still having too many linear components in comparison to triangles. Although attempting to resolve this will increase the dependence on the data, it is believed to be significant enough to warrant this.

3.3.3 Model 2b

In Figure 3.5 the relevant sections of the tables and other results measured, I see similar results to Model 2a. Again, the fits have various levels of visual similarity to the observed data for active links, active nodes, on-durations and off-durations, whilst the Kolmogorov-Smirnov distances, as reported in Tables 3.3 and 3.4 for these indicate that there are issues with these. With the GCC there are reasonable fits graphically, but similar to Model 2a there are still issues with Kolmogorov-Smirnov acceptances. Again, for nodes per component, links per component and the component count, I observe acceptable fits graphically (partially presented in Figure 3.5) and note in Table 3.4 a slight increase or similar levels in count of acceptances. I have a similar result for the node activity potential, with a very good graphic fit and a very high number of Kolmogorov-Smirnov acceptances. For the triangle count in the network, I observe good fits graphically and in terms of the statistical tests, with a substantial improvement over the results obtained in Model 2a. I also observe varying levels of popularity in the links, reflecting the various levels of friendships that can be seen in the original data - as can be seen in comparing behaviours in Figure 3.2a and Figure 3.2d.
Figure 3.5: Selected results for Model 2b. Simulated data is represented by crosses whereas observed data is represented by dotted lines. Each colour represents a different simulation or data set. See subsection 3.3.3 for full explanation. Additional figures are available in the supplemental materials.
Figure 3.6: Selected results for Model 2c. Simulated data is represented by crosses whereas observed data is represented by dotted lines. Each colour represents a different simulation or data set. See subsection 3.3.4 for full explanation. Additional figures are available in the supplemental materials.

### 3.3.4 Model 2c

In most metrics, this model performs similarly to Model 2b, with little to no difference in all of the examined metrics. Whilst, as illustrated in Table 3.4, some see a slight drop in the number of acceptances of the null hypothesis for the two-sample Kolmogorov-Smirnov test, others see a slight increase and overall I see a very marginal increase in the total count. Overall behaviours and link selection weighting reflect the observed data with a reasonable degree of accuracy as can be seen in Figure 3.6 and a comparison between Figures 3.2a and 3.2c.

### 3.4 Model Comparison

Overall, there is a considerable improvement across most metrics between Model 1 and Model 2a. This can be seen empirically when examining the statistical distances between the observed data and the generated simulations and the count of
5% acceptances (as illustrated in Tables 3.3 and 3.4). Significant improvements are made to the node activity potential and triangle count, including a noticeable graphical improvement to the GCC, as can be seen in Figures 3.3 and 3.4. Whilst modifications could be made to Model 1 to improve its accuracy in some of these areas (such as including the link selection preference matrix), due to its improved performance with similar levels of dependence on the data, the second model will be the basis for all future work. I also notice a substantial drop in link selection spread (defined in section 3.3) as I move between these models, with Model 2a reflecting real world behaviours much closer in the observations, as displayed in Figure 3.2.

Animations showing comparisons of the network in time are available in the supplemental materials. Model 1 homogenises considerably quicker than either the original data or the other models, implying that the ARP does not fully capture the preferential dynamics observed in the collected data. Additionally, whilst these show that the long-term behaviour for Model 2a and Model 2b more closely reflect the original data, it is worth noting that they converge to this behaviour much faster than originally observed. Whilst reasons for this will not be examined here, this suggests that there may be some variability in the distributions over the course of the day - for example, at the beginning of school, interactions between pupils may differ due to the extended period of time since their contact with others in their class. It is felt however, that this does not have a significant impact on the usage of these models as the limiting behaviour is likely to have a heavier impact in transmission studies.

Between Model 2a and Model 2b, many metrics remain similar, although as expected from the modifications to the algorithm, a considerable improvement to the triangle count is noticeable, illustrated in both Table 3.4 and when observing the decrease in the maximum and mean statistical distance for this metric in Table 3.3. However, one of the larger problems with Model 2b is that the link selection preference matrix depends heavily on the original data, and it is noted that I could reduce this data draw considerably by generating this matrix rather than extracting it directly from the data. Model 2c attempts to do this, and can be considered successful as can be observed in Tables 3.3 and 3.4, although a deeper examination of the temporal and network properties indicates that further improvements are still to be made.

The significant improvement between Model 1, Model 2a and Model 2b is to be expected given the nature of their constructions. Model 1 focuses closely on the interaction between individuals in the network and is relatively elementary in the approach that it takes in view of the initial analysis and data. Model 2a focuses more broadly, keeping some of the interaction dynamics, but also adding a preferential attachment, thus expanding the focus to the network topology, but at a slight expense of the link dynamics. Model 2b increases this focus on network topology by forcing a higher amount of triangulation. This improvement as the focus of the models is expanded suggests that, in future work, it may be beneficial to have similar ‘forcing’ rates to that which appeared in Model 2b but for other network motifs and substructures (as discussed in section 1.2). However, it is important to ensure that, as with any model construction, any additional variables (i.e. ‘forcing’ rates) lead to a large enough improvement in the accuracy of the model to warrant their inclusion.

---

14 See appendix A.7 for details.
15 Available from https://drive.google.com/open?id=13eUXp3xGph1B577d1IPvEVPZcHyACwom
<table>
<thead>
<tr>
<th></th>
<th>Model 1</th>
<th>Model 2a</th>
<th>Model 2b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Links</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Active Nodes</td>
<td>56</td>
<td>389</td>
<td>362</td>
</tr>
<tr>
<td>Node Activity Potential</td>
<td>881</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>GCC</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Time Between Contacts</td>
<td>–</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Component Count</td>
<td>0</td>
<td>211</td>
<td>364</td>
</tr>
<tr>
<td>Links per Component</td>
<td>0</td>
<td>43</td>
<td>45</td>
</tr>
<tr>
<td>Nodes per Component</td>
<td>0</td>
<td>42</td>
<td>48</td>
</tr>
<tr>
<td>Triangle Count</td>
<td>0</td>
<td>367</td>
<td>771</td>
</tr>
</tbody>
</table>

Table 3.5: Validation Acceptances of $H_0$ (see equation 3.9) at 5% Level (max: 1000) (see subsection 3.3 for full explanation)

- methods that may be used to examine this balance are discussed in chapter 5

### 3.5 Model Validation

As indicated in Table 3.1, the approach to using the same distributions through all of the primary school models is not ideal. Therefore, I shall examine the methods in such a way to examine if the dynamics used in these models is a valid choice. To do this I shall draw temporal data directly from the appropriate ECCDFs - for Model 1, these are the on-, off- and activation times, whilst for Model 2, these are the on-times and interevent times. I shall then compare the data generated using this method to the real world sample that I draw the ECCDFs from - if I have a low statistical distance between these, I can conclude that the model dynamics have validity and that any issues identified in the examination above can be significantly addressed through parameter improvements and refinements to the choice of distributions for the random values.

In Table 3.5, I present the results of this validation. I take each of the 20 original data samples and input the appropriate ECCDFs in the place of the random generation outlined in Methods 1, 2a and 2b as described in section 3.2. I do not analyse Method 2c using this method of validation as if I were to draw the link preferential matrix in this method from the data, this would be functionally identical to Model 2b.

I then generate 50 samples for each and compare them to the original data (for a total of 1000 comparisons for each metric and model). Please note that interaction times for all models (and the time between contacts for Model 1) have been excluded from this table as they are being controlled directly from the data and thus, a validation using this metric would serve no purpose. In this table, $H_0$ is given as

$$H_0 : \text{The chosen empirical and generated data samples come from a common distribution.}$$

$^{16}$See appendix A.12 for details.
Using this data, it is clear to see that the variations of Model 2 have considerably improved dynamics over Model 1, although it can be seen that there are still improvements to be made. When I compare Tables 3.4 and 3.5 I observe whilst choosing the ‘right’ time structures does lead to some improvements - most notably in terms of active nodes - it is not enough to ensure a fit across all chosen metrics, and therefore changes to the overall dynamics should be considered. From an examination of these results, I conclude that efforts should be made to improve link dynamics and hypothesise that by modifying the code to change the number of links generated in the network should improve the dynamics - especially for active nodes, although I would expect to also see improvements in the global clustering coefficient, component features and active links. This should also improve the time between contacts as changing the number of link activations will have a direct impact on this metric.

However, despite these small improvements still to be made to the model, I conclude that my Model 2b (and therefore 2c) have justifiable dynamics and that an improvement to the random generations will lead to an improved model overall.
Chapter 4

Bayesian Estimation

4.1 Introduction

Given a set of data, it is often wished to identify the distribution from which it has emerged. This is useful in many areas of mathematics and statistics, most notably in data modelling. In chapters 2 and 3, I use the method of moments [33], MLEs [171] and the non-linear least squares method used by the curve-fitting tool in MATLAB to attempt to identify distributions. However, this problem can also be approached from a Bayesian perspective. Here, I use Bayes’ theorem (given in Equation (4.1) below) to identify likely parameter choices:

\[ p(A|B) = \frac{p(B|A)p(A)}{p(B)} \, d\theta. \]  

(4.1)

To this end, I wish to identify the probability of each choice of the parameters \( A \) given data \( B \) - known as the posterior distribution. Doing so will allow me to calculate an expected value for these parameters. I also assume some information regarding the probability of these parameters appearing, which shall be the prior distribution \( p(A) \), and the likelihood of the data occurring given this choice of parameters, \( p(B|A) \).

In this chapter, I shall apply Bayesian techniques to the Mittag-Leffler distribution, given the importance of the distribution in this thesis and the challenges that this distribution will pose - namely the bounded nature of \( \mu \) and semi-infinite nature of \( \tau_0 \), as well as the heavy tail and power-law properties. Whilst I shall be focusing on this distribution here, it should be noted that these techniques can be applied to many different distributions.

4.2 Bayesian Estimates

4.2.1 Exact Bayesian Estimate

This is the direct method. I shall be calculating the value of the posterior probability as given in Equation (4.2) below as presented by Bishop [30, p. 21-24], where I use the likelihood (the probability of observations \( x \) given the parameters \( \theta \)) and a prior probability \( p(\theta) \):

\[ p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}, \]  

(4.2)
where \( p(\mathbf{x}) \) can be seen as a normalisation factor given by

\[
p(\mathbf{x}) = \int_\theta p(\mathbf{x} | \theta)p(\theta).
\]  

(4.3)

In this chapter, I will always assume compact support for the prior \( p(\theta) \) which ensures that integral (4.3) is finite. The basic idea is to compute the likelihood based on a specific model, multiply it by the chosen prior and then divide by the normalisation factor to get the posterior. All these calculations are based on numerical approximations, so the term exact refers to the fact that I am explicitly computing the exact Bayesian formula.

### 4.2.2 Markov-Chain Monte Carlo Method

In the Markov-chain Monte Carlo (MCMC) method, I use a standard Metropolis-Hastings algorithm.

#### 4.2.2.1 Introduction to the Markov-Chain Monte Carlo Method

The MCMC algorithm generates a Markov chain \( \{X(t)\} \) according to a desired distribution \( P(\mathbf{x}) \) and a proposal transitional kernel \( J(\mathbf{x} | \mathbf{y}) \), which gives the probability of moving to state \( \mathbf{x} \) at \( t + 1 \) given state \( \mathbf{y} \) at \( t \). Asymptotically, this chain will reach a unique stationary distribution such \( \pi(\mathbf{x}) \) such that \( \pi(\mathbf{x}) = P(\mathbf{x}) \) \cite{157}. This algorithm is as follows:

1. Given \( X^{(t)} = x^{(t)} \),
   1. Generate \( Y_t \sim J(y | x^{(t)}) \).
   2. Take
      \[
      X^{(t+1)} = \begin{cases} 
      Y_t, & \text{with probability } A(x^{(t)}, Y_t), \\
      x^{(t)}, & \text{with probability } 1 - A(x^{(t)}, Y_t),
      \end{cases}
      \]
      \[
      A(x, y) = \min \left\{ \frac{P(y) J(x | y)}{P(x) J(y | x)}, 1 \right\}.
      \]
      \[
      \] 
      (4.4)

For long chains, the distribution of the Markov chain \( \{X^{(t)}\} \) produced by this algorithm will resemble that of the underlying distribution \( P(\mathbf{x}) \) and can be generated without complete knowledge of \( P(\mathbf{x}) \), only the ratio between two terms \( \frac{P(\mathbf{y})}{P(\mathbf{x})} \).

With the correct construction, this ratio can be formed of known terms, allowing for the estimation of \( P(\mathbf{x}) \).

In subsubsection 4.2.2.2 below, we wish to estimate an unknown posterior distribution \( P(\theta | \mathbf{x}) \). Through Bayes’ theorem, we know that \( P(\theta | \mathbf{x}) = \frac{P(\mathbf{x} | \theta)P(\theta)}{P(\mathbf{x})} \) - of which we have information on the prior, \( P(\theta) \), and the likelihood, \( P(\mathbf{x} | \theta) \). It is our lack of knowledge on the evidence, \( P(\mathbf{x}) \), that would prevent us from calculating the posterior directly. However, the ratio used in the MCMC algorithm removes the dependency upon this knowledge as shown in (4.6):

\[ \footnote{See appendix A.10 for details.} \]
Since we have knowledge on all of the terms in the right-hand side, we can therefore use the MCMC algorithm to estimate the distribution for $P(\theta|x)$.

In the case where the proposal kernel is symmetric, that is $J(x|y) = J(y|x)$, this is known as the Metropolis algorithm.

4.2.2.2 Markov-Chain Monte Carlo Method for the Posterior Distribution

If this were a single parameter distribution, I would start by first generating an initial set of parameters $\theta_0$ at $t = 0$. I then iterate, first generating a candidate state $\hat{\theta}$ equal to a random perturbation from the current state, with this random perturbation distributed according to some chosen distribution. I shall be using a truncated multivariate Gaussian $N_\sigma^\prime(0, \sigma)$ with appropriate choices for $\sigma$ - as discussed in subsubsection 4.2.2.3 below - and $a', b'$ to ensure that the proposed value fits within the range of allowable values.

I then calculate an acceptance probability $A(\hat{\theta}, \theta_t)$ as given in (4.7) below. As I wish to estimate the posterior probability, $P(\theta|x)$, I use this in the place of $P(\cdot)$ in (4.5) and apply Bayes’ theorem using the likelihood and a chosen prior. Thus in (4.7), $J(r|s)$ is the proposal kernel, $P(x|\theta)$ is the likelihood of observing the data $x$ given the parameters $\theta$, and $P(\theta)$ is the chosen prior. This acceptance probability is then compared to a uniform random number $u \in [0, 1]$:

$$A(\hat{\theta}, \theta_t) = \min\left(1, \frac{P(x|\hat{\theta}) P(\theta_t) J(\theta_t|\hat{\theta})}{P(x|\theta_t) P(\hat{\theta}) J(\hat{\theta}|\theta_t)}\right).$$  \hspace{1cm} (4.7)

If $u \leq A(\hat{\theta}, \theta_t)$, I accept this new state and set $\theta_{t+1} = \hat{\theta}$. Otherwise, I reject the trial state and set $\theta_{t+1} = \theta_t$. As this is a multi-parameter distribution, I follow a slightly modified method, running through this method in each parameter at the same time step before moving on. That is, at time $t$, I first propose a move from $\theta_t = (\mu_t, \tau_t)$ to $(\mu', \tau_t)$, setting the result of the acceptance or rejection as an intermediary state $\theta_t^+ = (\mu_{t+1}, \tau_t)$. I then propose a move from this state $\theta_t^+ = (\mu_{t+1}, \tau_t)$ to $(\mu_{t+1}, \tau')$, setting the result of the acceptance or rejection as $\theta_{t+1} = (\mu_{t+1}, \tau_{t+1})$.

When I reach the chosen end time of the chain, I discard an adequate equilibration period, thus focusing on limiting behaviours, before giving the final estimate for the parameter values as the mean of the remaining chain values. In order to make best use of computation resources, I shall run several chains in parallel with varying starting points, $s_1, s_2, \ldots, s_n$ such that $s_i$ are IID and $s_i \sim U(a', b')$, having the final estimate for the parameter values as the mean across all chains once I have discarded the equilibration periods. This parallelisation allows for the checking the equilibrium states are unique through a visual inspection of the long-term behaviour of the chains, since if different starting states led to different equilibrium states, this behaviour would likely be caught through the use of a range of starting points. Lambert presents an argument for using as many parallel chains as possible, in order to
Figure 4.1: Example MCMC chains for varying choices of $\sigma$ with same distribution of given data, initial point and scales showing parameter values against MCMC step limit the chances of missing equilibrium states between which transition is unlikely \cite{112}.

4.2.2.3 Choice of $\sigma$

The choice of $\sigma$ has a significant impact on the algorithm. If this variance is too large, the Markov chain will become stationary within different regions of the space, with abrupt jumps between them, whilst if it is too small, the chain moves very slowly and will not converge within a reasonable simulation time. One method of quickly assessing whether the choice of $\sigma$ is reasonable is to run a short simulation and calculate the acceptance rate for moves. If I am accepting few moves, then the chain is being stationary for long periods between jumps, indicating that the value of $\sigma$ is too large. Examples of chains with too large, small and ideal values of $\sigma$ can be seen in Figure 4.1. If I am accepting many moves, then the chain is moving constantly and is likely to be moving towards any convergent state very slowly, indicating that the value of $\sigma$ is too small. Literature points towards an acceptance rate of between 0.15 and 0.5 being ideal in terms of obtaining convergence within a reasonable chain length \cite{81,158,213}.

Due to this, in the MCMC method, I shall first run short chains with the same prior distribution and data, and varying values of $\sigma$ and choose a value that gives an acceptance rate within a small window around 0.45. This is chosen both to ensure that the acceptance rate of the longer chain will still fall within the ideal range, as well as to fit with the optimal scaling for a single dimension - as I am stepping dimension-by-dimension - as simulated by Roberts and Rosenthal \cite{158}.

The MATLAB code for selecting $\sigma$ can be found in the appendices \cite{3}.

4.2.3 Approximate Bayesian Computation

In this method, as explored by Beaumont \cite{24}, I use summary statistics rather than the full dataset. The aim is, as always, to create an estimate for the posterior distribution $p(\theta|x)$. However, sampling from the full dataset may be non-viable in practice and highly inefficient even for moderately sized datasets. This method also does not rely on knowledge of the likelihood, which in many cases is either unknown or computationally complex to calculate. A typical approximate Bayesian

\footnote{See appendix A.10 for details.}

\footnote{See appendix B.47 for details.}
computation (ABC) analysis specifies a vector of summary statistics \( s = S(\mathbf{x}) \), where \( \dim(s) \ll \dim(\mathbf{x}) \), and then outputs a posterior distribution for \( p(\theta|s) \) [179]. I sample parameter points \( \hat{\theta} \) from chosen priors, then for each set of points, I generate a summary data set \( \hat{\mathbf{x}} \) under the model with these parameters. I compare this summary data set \( \hat{D} \) with the original summary data set \( D \) using a chosen distance \( \rho \), accepting this set of parameter points if this value is below a defined \( \varepsilon \), and rejecting this set of parameter points if it is above (see Equation (4.8) below)

\[
\rho(D, \hat{D}) \leq \varepsilon.
\] (4.8)

This will produce, as in the other methods, a posterior distribution, and thus, the final estimate for the parameter values is given as the mean of all accepted sets of parameter points. However, the method requires care with the summary statistic to ensure that it contains sufficient information about the distribution. This sufficiency in summary statistics for an ABC analysis is a critical decision and as such the dimension of \( s \) should be large enough that it contains as much information about the observed data as possible, whilst also low enough that the method does not run into issues with the number of acceptances using the distance \( \rho \).

If this vector \( s = S(\mathbf{x}) \) is sufficient for the model parameters, then \( p(\theta|s) \equiv p(\theta|x) \), and thus the posterior produced is equivalent to the true posterior. If this vector \( s \) is not sufficient, then ABC analysis shall produce the posterior \( p(\theta|s) \), which may be significantly different to the true posterior [179].

### 4.3 Examples

#### 4.3.1 Testing of Exact and MCMC Methods on Generated Data

As a means of verification for the methods presented, I shall first test them on data generated from a Mittag-Leffler distribution (available in the supplemental materials\(^5\)). I shall generate 9 data sets representing combinations of \( \mu \) and \( \tau_0 \) across the space of parameter values - I choose to look at \( \mu = [0.1, 0.5, 0.9] \) and \( \tau_0 = [10^{-2}, 1, 10^2] \) - and generate 500 data samples (justified below in subsubsection 4.3.1.1).

I shall assume parameters are independent of each other, and for ease of computation estimate a window in which I assume each falls using a brief examination of the data. I shall make no further assumptions on the values of the parameters, and use the simplest non-informative prior - the uniform distribution - over the chosen windows. For \( \mu \), I assume that the actual value lies anywhere within the range of acceptable values, \( 0 < \mu < 1 \). However due to programming constraints, I shall reduce this slightly and examine the range \([0.01, 0.99] \). For \( \tau_0 \), I estimate a range using the given data (discussed in subsubsection 4.3.1.2) and assume that the true value lies within this. The priors for \( \mu \) and \( \tau_0 \) used were \( U(0.01, 0.99) \) and \( U(a,b) \) - where \( a \) is equal to 0.9 times the lower bound given by my estimation method, and \( b \) is equal to 1.1 times the upper bound. Running the data and chosen priors through the MCMC method with 8 parallel chains with a chain length of 3000 and discarding

\(^8\)See appendix A.1 for details.

\(^5\)Available from [https://drive.google.com/open?id=13eUXp3xGph1BST7dIPvEVPZcHyACmom](https://drive.google.com/open?id=13eUXp3xGph1BST7dIPvEVPZcHyACmom)
Table 4.1: Mean and standard deviation estimates for the MCMC method

<table>
<thead>
<tr>
<th>Actual Parameters</th>
<th>Estimated Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu$</td>
<td>$\tau_0$</td>
</tr>
<tr>
<td>Mean</td>
<td>Std.</td>
</tr>
<tr>
<td>0.1</td>
<td>0.01</td>
</tr>
<tr>
<td>0.1</td>
<td>1.00</td>
</tr>
<tr>
<td>0.1</td>
<td>100</td>
</tr>
<tr>
<td>0.5</td>
<td>0.01</td>
</tr>
<tr>
<td>0.5</td>
<td>1.00</td>
</tr>
<tr>
<td>0.5</td>
<td>100</td>
</tr>
<tr>
<td>0.9</td>
<td>0.01</td>
</tr>
<tr>
<td>0.9</td>
<td>1.00</td>
</tr>
<tr>
<td>0.9</td>
<td>100</td>
</tr>
</tbody>
</table>

In both methods, I observe that the estimated parameters do differ somewhat from the actual parameters, especially in regards to $\tau_0$. However, it is important to note that in most cases, the true parameters lie within a window of 1-2 standard deviations of the true parameters. Additionally, when examining the CCDFs (such as those in Figure 4.5), the generated data does not always follow the curve using the true or estimated parameters too closely, although again, looking at a window of 1-2 standard deviations either side of the estimated parameters, the generated data is captured well.
Figure 4.2: Sample MCMC chains for $\mu = 0.5$ and $\tau_0 = 1$. Varying colours in each image represent each of the parallel chains. Highlighted region shows equilibration period. Solid red lines represent estimated parameters with red dashed lines showing one standard deviation either side of this. Solid black lines represent true parameters.
Figure 4.3: Sample multidimensional posterior PDF plots for $\mu = 0.5$ and $\tau_0 = 1$ using exact and MCMC methods. Red and blue lines represent estimated parameters. Black lines represent true parameters.
Figure 4.4: Sample marginal posterior PDF plots and histograms of MCMC chains for $\mu = 0.5$ and $\tau_0 = 1$. Prior densities are shown as yellow horizontal lines. Red vertical line shows estimated mean using MCMC method (with red shaded region being one standard deviation either side of this). Blue vertical line shows estimated mean using exact method (with blue shaded region being one standard deviation either side of this). Black vertical line shows true parameters.
Figure 4.5: Sample CCDF plots showing generated data and distributions using true and estimated parameters for \( \mu = 0.5 \) and \( \tau_0 = 1 \). Black line shows true parameters, red lines show parameters estimated using MCMC method, blue lines show parameters estimated using exact method. Solid lines represent estimates, dashed lines represent one standard deviation from estimates, dotted lines represent two standard deviations from estimates.
### Actual Parameters | Estimated Parameters
| $\mu$ | $\tau_0$ | $\mu$ | $\tau_0$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.01</td>
<td>0.0976</td>
<td>0.0036</td>
</tr>
<tr>
<td>0.1</td>
<td>1.00</td>
<td>0.0998</td>
<td>0.0037</td>
</tr>
<tr>
<td>0.1</td>
<td>100</td>
<td>0.0999</td>
<td>0.0037</td>
</tr>
<tr>
<td>0.5</td>
<td>0.01</td>
<td>0.5389</td>
<td>0.0173</td>
</tr>
<tr>
<td>0.5</td>
<td>1.00</td>
<td>0.5016</td>
<td>0.0167</td>
</tr>
<tr>
<td>0.5</td>
<td>100</td>
<td>0.5307</td>
<td>0.0173</td>
</tr>
<tr>
<td>0.9</td>
<td>0.01</td>
<td>0.8950</td>
<td>0.0181</td>
</tr>
<tr>
<td>0.9</td>
<td>1.00</td>
<td>0.8905</td>
<td>0.0183</td>
</tr>
<tr>
<td>0.9</td>
<td>100</td>
<td>0.8840</td>
<td>0.0183</td>
</tr>
</tbody>
</table>

Table 4.2: Mean and standard deviation estimates for the exact method

<table>
<thead>
<tr>
<th>Sample Size</th>
<th>Estimated Parameters</th>
</tr>
</thead>
</table>
|             | $\mu$ | $\tau_0$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Std.</td>
</tr>
<tr>
<td>100</td>
<td>0.5427</td>
<td>0.0386</td>
</tr>
<tr>
<td>500</td>
<td>0.5018</td>
<td>0.0161</td>
</tr>
<tr>
<td>1000</td>
<td>0.5140</td>
<td>0.0117</td>
</tr>
</tbody>
</table>

Table 4.3: Mean and standard deviation estimates for the MCMC method for various data sample sizes

#### 4.3.1.1 Comments on Data Size

For the tabulated results above, I have used generated data consisting of 500 points, which was considered reasonable - to justify this, I additionally ran the code with data sizes of 100 and 1000 with $\mu = 0.5$ and $\tau_0 = 1$ and present the results in Tables 4.3 and 4.4. These are the variations presented in Figures 4.2, 4.3, 4.4 and 4.5.

It can be seen that although for small sample sizes the estimates for $\mu$ and $\tau_0$ deviate further from the true values (as they also do, in this case for large sample sizes), there are several things to note. Firstly, that the mean and standard deviation for each sample size is comparable between methods, indicating that the MCMC method I am using offers a similar degree of accuracy to the more computationally complex exact method. Secondly, despite the changes in deviation from the true values, the standard deviation of the estimated parameters increases as the sample size is reduced - thus despite the size of the sample, the estimated parameter values are likely to lie within approximately $\sigma$ of the true values.

#### 4.3.1.2 Prior Information on the Range of $\tau_0$

For $\tau_0$, I studied the survival function for the Mittag-Leffler distribution given in (4.9):

$$\Psi(t) = E_\mu (- (t/\tau_0)^\mu).$$

(4.9)
Table 4.4: Mean and standard deviation estimates for the exact method for various data sample sizes

<table>
<thead>
<tr>
<th>Sample Size</th>
<th>Estimated Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>μ</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
</tr>
<tr>
<td>100</td>
<td>0.5444</td>
</tr>
<tr>
<td>500</td>
<td>0.5016</td>
</tr>
<tr>
<td>1000</td>
<td>0.5142</td>
</tr>
</tbody>
</table>

Figure 4.6: $E_\mu(z)$ plotted again $\mu$ showing relationship. Different curves show this relationship for various values between $z = -1.1$ (lowest curve, blue) and $z = -0.9$ (highest curve, red).

When $t = \tau_0$, I have that $\Psi(\tau_0) = E_\mu(-1)$. Examining the single-parameter Mittag-Leffler function for $0 < \mu < 1$ in a neighbourhood around $z = -1$ through simulations (see Figure 4.6), it emerges that $E_\mu(z)$ decreases as $\mu$ increases and thus in this neighbourhood, Equation (4.10) below must hold:

$$
\sum_{k=0}^{\infty} z^k = E_0(z) > E_\mu(z) > E_1(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(k+1)} = \sum_{k=0}^{\infty} \frac{z^k}{k!} = \exp(z).
$$

If I let $t_* = \tau_0 - \varepsilon$ for some sufficiently small $0 < \varepsilon \ll 1$, I can approximate $-(t/\tau_0)^\mu$ by $z_* = -(1 + \delta)$ for some $0 < \delta \ll 1$, such that Equation (4.10) holds for $z_*$. Thus, I have Equation (4.11) below, since $|z_*| < 1$:

$$
\frac{1}{2} > \frac{1}{1 - z_*} = \sum_{k=0}^{\infty} z_*^k > E_\mu(z) = \Psi(t_*) > \exp(z_*) > \exp(-1).
$$

Therefore, if I use the ECCDF of the data to calculate $\mathcal{T}$ given in Equation (4.12) below, it must hold that $t_* \in \mathcal{T}$ and thus that $\tau_0$ belongs to $\mathcal{T}_\varepsilon$ given in Equation (4.13):

$$
\mathcal{T} = \{ t \in \mathbb{R} : \Psi(t) \in [\exp(-1), 0.5] \}.
$$
Figure 4.7: Method for approximating $\tau_0$, showing ECCDF (blue), $\exp(-1)$ and 0.5 (orange), first values outside of this region (yellow) and the resulting search range (purple) - here $\tau_0 = 1$ which is captured within the search range

$$T_\varepsilon = \left[ \inf(T) - \varepsilon, \sup(T) + \varepsilon \right]. \quad (4.13)$$

Algorithmically, I shall return all data points $t$ such that $\Psi(t) \in [\exp(-1), 0.5]$, then extend this range slightly to account for the $\varepsilon$-window and the fact that this is an estimation based on ideal behaviour of the data. As I have no further information about the values of the parameters except that they fall within these windows, I shall use the simplest non-informative prior - the uniform distribution. A rough visualisation of this method can be seen in Figure 4.7.

If the information on the ECCDF is incomplete (such as in the case of limited summary data), I perform a rough approximation, using 0 as the lower bound of the search range. For the upper bound, I first use a least squares method to return a best fit exponential curve of the form $y = a \exp(bx)$ through any given data. I then extrapolate this curve to find $x_*$ such that $a \exp(bx_*) = \exp(-1)$ with these best fit parameters. As this is a rough approximation, the upper bound that I use for the $\tau_0$ search range is equal to one order of magnitude greater than that of $x_*$. For example, if the limited information given gave rise to a best-curve $y = a' \exp(b'x)$, and for $y = \exp(-1)$, the solution to this equation was $4.1 \times 10^3$, I would take my window to be $[0, 10^4]$.

### 4.3.2 Earthquake Recurrence

I shall now examine waiting times between localised earthquakes as gathered by the United States Geological Survey and presented on their website [196] on the assumption that their unconditional distribution is the Mittag-Leffler distribution. The data chosen was earthquakes of magnitude greater than 2.5, occurring in a 100km radius around the point (44°N,8°E) between 2000-08-23 00:00:00UTC and 2010-08-23 23:59:59UTC. The data used and geology of the selected region can be visualised in Figure 4.8.
As I have all the waiting times for this example, I can use my method for exact Bayesian estimation (as given in subsection 4.2.1) as well as MCMC methods (for instance, as given in subsection 4.2.2). I shall also assume that the waiting times are identically distributed, and independent of each other. This means that I can use the following formula for the likelihood $p(x|\theta)$:

$$p(x|\theta) = \prod_i p(x_i|\theta), \quad (4.14)$$

where $x_i$ are the waiting times. Note that this assumption is indeed wrong, as earthquake point processes seem to be better represented by self-exciting processes such as Hawkes processes \[89\]. However, for the sake of simplicity in this methodological example, I shall assume the IID hypothesis in the same spirit as in earlier work by Scalas, Gorenflo and Mainardi \[169\].

The parameters for both the exact method and MCMC method when applied to this earthquake data under these assumptions and with uniform priors $\mathcal{U}(0.01, 0.99)$ and $\mathcal{U}(a,b)$ (as discussed in subsection 4.3.1) are presented in Table 4.5, with chains, multidimensional posterior PDF, marginal posterior PDF and ECCDF plots presented in Figures 4.9, 4.10, 4.11 and 4.12.
Figure 4.9: MCMC chains for earthquake data. Varying colours in each image represent each of the parallel chains. Highlighted region shows equilibration period. Solid red lines represent estimated parameters with red dashed lines showing one standard deviation either side of this.

Figure 4.10: Multidimensional posterior PDF plots for earthquake data using exact and MCMC methods. Red and blue lines represent estimated parameters.

Figure 4.11: Marginal posterior PDF plots and histograms of MCMC chains for earthquake data. Prior densities are shown as yellow horizontal lines. Red vertical line shows estimated mean using MCMC method (with red shaded region being one standard deviation either side of this). Blue vertical line shows estimated mean using exact method (with blue shaded region being one standard deviation either side of this).
Figure 4.12: CCDF plot showing generated data and distributions using estimated parameters for earthquake data. Red lines show parameters estimated using MCMC method, blue lines show parameters estimated using exact method. Solid lines represent estimates, dashed lines represent one standard deviation from estimates, dotted lines represent two standard deviations from estimates.

It is worth noting that when examining the CCDF shown in Figure 4.12, the 2σ-window is only a good fit up until approximately 2 × 10⁶s (∼ 3 weeks) between events. However, as discussed earlier, the assumption that the data is Mittag-Leffler distributed is incorrect, thus this conclusion was to be expected.

Additionally, it is worth noting that as the Mittag-Leffler distribution provides this partial fit, this earthquake data may come from a truncated power-law distribution, a distribution type often found in natural phenomena [3, 44, 85, 108, 130, 192]. The PDF of a truncated power-law is of the form (4.15a), where the PDF of the power-law is of the form (4.15b):

\[ P(x) \propto x^{\alpha-1} e^{\frac{x}{x_c}} \]  
\[ P(x) \propto x^{-\gamma} \]  

where \( \alpha \) is a constant value, \( x_c \) is the truncation or cut-off value and \( \gamma \) is the scaling constant for the power-law [3, 192]. Although I shall not examine truncated variations of the Mittag-Leffler distribution here, fits could be examined using similar methods with minimal changes to the given algorithms.

It is worth noting though that, similar to the results I observed on synthetically generated data, there is a great deal of similarity between the parameter estimates generated by the MCMC and exact methods, further supporting the conclusion that the MCMC algorithm presented provides a very accurate estimation, whilst being much less computationally intensive for large data sizes.

4.3.3 Testing of ABC on Generated Data

Similar to how I tested the methods for exact Bayesian estimation and MCMC methods in subsection 4.3.1, I shall test the method of ABC on summary results of data generated from a Mittag-Leffler distribution (as discussed in section 1.4). I take the data previously generated in subsection 4.3.1 and generate appropriate summary data - namely the value for the CCDF at 20 points equally spaced along this function (see subsubsection 4.3.3.1 for justification). Summary data is available
Table 4.6: Mean and standard deviation estimates for the ABC method

<table>
<thead>
<tr>
<th>Actual Parameters</th>
<th>Estimated Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\mu$</td>
</tr>
<tr>
<td>$\mu$</td>
<td>$\tau_0$</td>
</tr>
<tr>
<td>0.1</td>
<td>0.01</td>
</tr>
<tr>
<td>0.1</td>
<td>1.00</td>
</tr>
<tr>
<td>0.1</td>
<td>100</td>
</tr>
<tr>
<td>0.5</td>
<td>0.01</td>
</tr>
<tr>
<td>0.5</td>
<td>1.00</td>
</tr>
<tr>
<td>0.5</td>
<td>100</td>
</tr>
<tr>
<td>0.9</td>
<td>0.01</td>
</tr>
<tr>
<td>0.9</td>
<td>1.00</td>
</tr>
<tr>
<td>0.9</td>
<td>100</td>
</tr>
</tbody>
</table>

in the supplemental materials. The ABC algorithm uses a uniform prior for $\mu$ and $\tau_0$, with the same supports as discussed in subsection 4.3.1, and is looped until it returns at least 100 values with a maximum distance of 0.05 from the data given, then returns the mean of these values. I present the estimated parameters in table 4.6 and sample CCDFs in Figure 4.13.

Similar to the results on generated data using the exact and MCMC methods, the ABC method returns good approximations to the true values, with most estimates being within 1-2 standard deviations of these. As is expected, the ABC method does not return estimates as accurately as these other methods (as can be seen when comparing Tables 4.1, 4.2 and 4.6), although when considering that this method only requires a limited amount of summary data, the approximations it returns are comparable. Additionally, when examining the CCDFs (such as those in Figure 4.13), the generated data does not always follow the curve using the true or estimated parameters too closely, although again, looking at a window of 1-2 standard deviations either side of the estimated parameters, the generated data is captured well.

It is worth noting that in all three Bayesian methods I have presented on generated data, the estimated parameters lead to a curve that often has a slight repeated bias - in Figures 4.5 and 4.13, the estimated parameters lead to a curve that is consistently lower than one using the true parameters. Whilst I will not investigate the cause of this here nor perform a systematic study of this bias, this could possibly be indicative of a slight bias in the algorithm that I am using for the generation of Mittag-Leffler data that results in an improperly weighted tail.

4.3.3.1 Comments on Data Size

For the tabulated results above, I have used generated data consisting of 20 summary points, which was considered reasonable - to justify this, I additionally ran the code

Available from [https://drive.google.com/open?id=13eUXp3xGph1BST7d1IPvEVPZcHyACmom](https://drive.google.com/open?id=13eUXp3xGph1BST7d1IPvEVPZcHyACmom)
Figure 4.13: Sample CCDF plots showing generated data and distributions using true and estimated parameters for $\mu = 0.5$ and $\tau_0 = 1$. Black line shows true parameters, red lines show parameters estimated using ABC method. Solid lines represent estimate, dashed lines represent one standard deviation from estimate, dotted lines represent two standard deviations from estimate.
Table 4.7: Mean and standard deviation estimates for the ABC method for various data sample sizes

<table>
<thead>
<tr>
<th>Sample Size</th>
<th>Estimated Parameters</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>µ Mean</td>
<td>Std.</td>
<td>τ Mean</td>
</tr>
<tr>
<td>10</td>
<td>0.5186</td>
<td>0.0428</td>
<td>0.8415</td>
</tr>
<tr>
<td>20</td>
<td>0.5190</td>
<td>0.0416</td>
<td>0.8390</td>
</tr>
<tr>
<td>50</td>
<td>0.5118</td>
<td>0.0358</td>
<td>0.8353</td>
</tr>
</tbody>
</table>

Table 4.8: Mean and standard deviation estimates for recidivism of drug ex-prisoners (RDEP) and duration of human residence (DHR) with summary data sizes of 10 and 50 with \( \mu = 0.5 \) and \( \tau_0 = 1 \) and present the results in Table 4.7. These are the variations presented in Figure 4.13.

Similarly to the results for the exact and MCMC methods, the ABC method does show improvement as it is given more data, although the degree of this improvement is fairly small. However, based on these results, I believe that using 20 pieces of summary data is justified as the algorithm produces fairly accurate results with this number of data points.

### 4.3.4 Social Data

I shall examine two data sets on the recidivism rates of ex-prisoners convicted of drug felonies during 1993 in Jackson County, Missouri[185] and the duration of human residence in the United States from 1985 to 1993[14], estimated from given curves displaying the survival probability \( \Psi(\tau) = E_{\mu}(- (\tau/\tau_0)^\mu) \) (as discussed in section 1.4) as presented by Stage and Fedotov[186]. Since I am using summary data, namely the survivor function, as the source for the analysis, I shall be using the approximate Bayesian computation as given in subsection 4.2.3. I present the parameter estimates and their standard deviations for both these data sets, with times measured in months, against those presented by Stage and Fedotov in Table 4.8. It should be noted that whilst the Stage values for \( \mu \) are taken from[186], the Stage values for \( \tau_0 \) have had to be approximated using the graphical results presented in the paper. The ABC algorithm uses a uniform prior for \( \mu \) and \( \tau_0 \) and is looped until it returns a minimum 100 values with a maximum distance of 0.05 from the data given, then returns the mean of these 100 values.
From this table, it is noticeable that the estimates for $\tau_0$ generated from the Stage graphical results are very close to those estimated by the ABC, although the estimates for $\mu$ do differ between the two methods. However, when I present these fits as CCDFs as in Figure 4.14, it appears that the ABC results produce a much closer fit than those presented by Stage and Fedotov.
Chapter 5
Model Selection

5.1 Introduction
When different approaches to data produce different parameters, it is often essential to compare these to find a preferred set of parameters. I have briefly touched upon this with my use of EDF statistics as introduced in subsection 2.1.2.1, but other methods do exist including Bayesian-based calculations.

5.2 Methods
I shall use three methods of model selection - the $L^2$ distance (also known as the Euclidean distance), given by Equation (5.1), Akaike information criteria (AIC), given by Equation (5.2), and Bayesian information criteria (BIC), given by Equation (5.3) [67, p. 399-403]:

$$d(p, q) = \sqrt{\sum_i (p_i - q_i)^2}.$$  \hspace{1cm} (5.1)

$$AIC(x|\theta) = 2k - 2 \log(P(x|\theta)).$$  \hspace{1cm} (5.2)

$$BIC(x|\theta) = \log(n)k - 2 \log(P(x|\theta)).$$  \hspace{1cm} (5.3)

In the definition of the $L^2$ distance, I assume a ECCDF for the data evaluated at discrete points, which has values $p$, and a CCDF of the chosen distribution and parameters evaluated at the points, which has values $q$. In the definition of AIC and BIC, $\theta$ refers to the chosen parameters to be examined, $k$ refers to the number of parameters being estimated and $n$ to the number of observations. $P(x|\theta)$ is the likelihood as discussed in chapter 4.

5.2.1 Akaike Information Criterion
AIC is a technique based on in-sample fit to estimate the likelihood of a model to predict values. A good model is one that has minimum AIC amongst competing models on the same data set [131]. The AIC may be used to judge which of two sets of parameters is better, or which of two functions with best-fitting parameters is a better descriptor of a given data set [107]. It should be noted that the numerical value of the criterion is not a goodness of fit measure - it is not possible to compare fits between data sets, only the difference between two values for different models.
of a single data set can be interpreted. In (5.2), the second term measures the model fit, and the negative factor of this indicates why smaller AIC values indicate better fits. The first term penalises over-fitting - the addition of a new parameter to a model must have a substantial impact on the size of $\log(P(x|\theta))$, else AIC will indicate a poorer model. The combination of these terms results in a trade-off between under-fitting and over-fitting that is core to developing a good model.

It should be noted that AIC is useful in selecting the best model out of those given - if all analysed models are poor, AIC will still select the one estimated to be best, even if that model is still poor in the absolute sense. Thus care must be taken to ensure that the set of models is well founded.

AIC requires the full data set as this is necessary in the calculation of the log-likelihood $\log(P(x|\theta))$.

### 5.2.2 Bayesian Information Criterion

BIC is very similar to AIC in its method for comparing models. Like AIC, this is not a goodness-of-fit measure and can only compare models on the same data set and a model with minimum BIC amongst competing models on the same data set will be selected as the best. The computational difference is in the weighting of the penalty of additional parameters - in BIC this is $\log(n)k$ compared to the value of $2k$ in AIC, and thus BIC will penalise large parameter models more than AIC.

It should be clear that BIC also requires the full data set, both as the formula depends on the size of this data set, as well as for the same reasons as AIC.

### 5.2.3 Comparisons Between Methods

Whilst the two formulae look similar, there are several differences between AIC and BIC and thus on where it is appropriate to use each one. BIC is argued to be appropriate selecting the ‘true’ model out of a set of candidates, whilst AIC is not appropriate - this is as if such a model belongs to the candidate set, then the probability of BIC selecting this model approaches 1 as $n \to \infty$, whilst for AIC, this probability can still be less than 1. Simulation studies have been done, and whilst BIC will select this model asymptotically, AIC can select a significantly better model than BIC, even when such a ‘true’ model is in the candidate set - this can even hold for substantially large values of $n$. BIC can have a risk of selecting a poor model from the given set, whilst with AIC this risk is minimised. Additional simulations suggest that predictions based on an AIC-selected model are much closer to being unbiased than a prediction based on a BIC-selected model.

It is possible to consider two simulations to view the situations in which AIC and BIC are appropriate.

In the first, a very complex model produces the data and, as such, it would be expected that the parameter space would be much larger than the size of the data and it would not be expected that the true process would appear in a set of candidate models - indeed candidate models may not include the complete pool of true parameters and could include additional parameters. In this case, it is not possible to find the true model for this process, and maximising predictive accuracy is highly desirable.

In the second, a relatively simple model produces the data and, as such, it would
be expected that the size of the data would be much larger than the parameter space and, with a sufficiently large set of candidates, the true process would appear. Here, predictive accuracy is less important as actually identifying the true process.

AIC would be appropriate when dealing with situations analogous to the first case, whilst BIC would be appropriate when dealing with situations analogous to the second - indeed, AIC will almost always perform better than BIC in the first case, and BIC will almost always perform better than AIC in the second [5].

5.3 Results for Mittag-Leffler Distributed Data

Results of AIC, BIC and $L^2$ distances for the generated Mittag-Leffler data and Earthquake data can be found in Table 5.1.

Using this, it is possible to compare the fit for the model generated using the parameters from the MCMC method to that generated using the parameters from the exact Bayesian method. In many cases, the MCMC parameter model returns similar, if not better, results to the information criteria than the exact method, supporting the conclusion that it is a very good approach to estimating parameters. Additionally, as several of these data sets are generated using known parameters, I can compare the fit of my models to the fit of a model with these known parameters (for example, in $G_{500}(0.1, 0.01)$, I can compare to a model with $\mu = 0.5$ and $\tau_0 = 0.1$). In a large number of cases, the information criteria are comparable across all three models. Indeed, in some cases, the models produced using the MCMC or exact method have better criteria than this ‘true’ fit (where available), suggesting once again the potential for bias in the methodology for the generation of data.

5.4 Applications to Network Generation Models

Originally, when choosing my distributions and parameters for use in the models introduced in chapter 3, I used classical Fisherian methods. Further analysis using Bayesian methods to select parameters as discussed in chapter 4 and a comparison of distributions with these parameters using AIC and BIC could lead to desired improvements to these models. Whilst in these chapters, these methods are discussed in regards to the Mittag-Leffler distribution, the algorithms presented are fairly easy to adapt to other distributions, or indeed to include any prior knowledge. Additionally, whilst the Mittag-Leffler distribution was originally discounted as a potential distribution for use in the models proposed in chapter 3, a truncated form of this distribution, as briefly discussed in subsection 4.3.2, may be considered as a candidate in an examination of a broader range of distributions.

However, this shall not be presented here, but is instead left alongside the tools and methodologies required as a basis for future work.

---

1See appendix A.2 for details.
<table>
<thead>
<tr>
<th>Data Set</th>
<th>Paras.</th>
<th>AIC Value</th>
<th>BIC Value</th>
<th>$L^2$ distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_{500}(0.1, 0.01)$</td>
<td>MCMC</td>
<td>$-1.7787 \times 10^4$</td>
<td>$-1.7702 \times 10^4$</td>
<td>13.2424</td>
</tr>
<tr>
<td></td>
<td>Exact</td>
<td>$-1.7787 \times 10^4$</td>
<td>$-1.7703 \times 10^4$</td>
<td>13.2427</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>$-1.7787 \times 10^4$</td>
<td>$-1.7694 \times 10^4$</td>
<td>13.2456</td>
</tr>
<tr>
<td>$G_{500}(0.1, 1)$</td>
<td>MCMC</td>
<td>$2.5892 \times 10^4$</td>
<td>$2.5977 \times 10^4$</td>
<td>13.2447</td>
</tr>
<tr>
<td></td>
<td>Exact</td>
<td>$2.5891 \times 10^4$</td>
<td>$2.5975 \times 10^4$</td>
<td>13.2447</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>$2.5892 \times 10^4$</td>
<td>$2.5977 \times 10^4$</td>
<td>13.2446</td>
</tr>
<tr>
<td>$G_{500}(0.1, 100)$</td>
<td>MCMC</td>
<td>$8.5121 \times 10^4$</td>
<td>$8.5205 \times 10^4$</td>
<td>13.2491</td>
</tr>
<tr>
<td></td>
<td>Exact</td>
<td>$8.5121 \times 10^4$</td>
<td>$8.5205 \times 10^4$</td>
<td>13.2491</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>$8.5109 \times 10^4$</td>
<td>$8.5194 \times 10^4$</td>
<td>13.2532</td>
</tr>
<tr>
<td>$G_{500}(0.5, 0.01)$</td>
<td>MCMC</td>
<td>$-2.4187 \times 10^3$</td>
<td>$-2.4103 \times 10^3$</td>
<td>14.8897</td>
</tr>
<tr>
<td></td>
<td>Exact</td>
<td>$-2.4187 \times 10^3$</td>
<td>$-2.4103 \times 10^3$</td>
<td>14.8886</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>$-2.4106 \times 10^3$</td>
<td>$-2.4021 \times 10^3$</td>
<td>14.8421</td>
</tr>
<tr>
<td>$G_{100}(0.5, 1)$</td>
<td>MCMC</td>
<td>427.6142</td>
<td>432.8245</td>
<td>6.6687</td>
</tr>
<tr>
<td></td>
<td>Exact</td>
<td>427.5987</td>
<td>432.8090</td>
<td>6.6707</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>429.2691</td>
<td>434.4794</td>
<td>6.6528</td>
</tr>
<tr>
<td>$G_{500}(0.5, 1)$</td>
<td>MCMC</td>
<td>1.7983 \times 10^4</td>
<td>1.8068 \times 10^4</td>
<td>14.8897</td>
</tr>
<tr>
<td></td>
<td>Exact</td>
<td>1.7983 \times 10^4</td>
<td>1.8068 \times 10^4</td>
<td>14.7406</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>1.8004 \times 10^4</td>
<td>1.8088 \times 10^4</td>
<td>14.7069</td>
</tr>
<tr>
<td>$G_{1000}(0.5, 1)$</td>
<td>MCMC</td>
<td>3.8037 \times 10^3</td>
<td>3.8135 \times 10^3</td>
<td>20.8847</td>
</tr>
<tr>
<td></td>
<td>Exact</td>
<td>3.8037 \times 10^3</td>
<td>3.8135 \times 10^3</td>
<td>20.8858</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>3.8060 \times 10^3</td>
<td>3.8158 \times 10^3</td>
<td>20.8215</td>
</tr>
<tr>
<td>$G_{500}(0.5, 100)$</td>
<td>MCMC</td>
<td>6.4711 \times 10^4</td>
<td>6.4796 \times 10^4</td>
<td>14.8474</td>
</tr>
<tr>
<td></td>
<td>Exact</td>
<td>6.4711 \times 10^4</td>
<td>6.4796 \times 10^4</td>
<td>14.8477</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>6.4750 \times 10^4</td>
<td>6.4834 \times 10^4</td>
<td>14.7609</td>
</tr>
<tr>
<td>$G_{500}(0.9, 0.01)$</td>
<td>MCMC</td>
<td>$-3.3068 \times 10^3$</td>
<td>$-3.2983 \times 10^3$</td>
<td>16.9017</td>
</tr>
<tr>
<td></td>
<td>Exact</td>
<td>$-3.3067 \times 10^3$</td>
<td>$-3.2983 \times 10^3$</td>
<td>16.9000</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>$-3.3059 \times 10^3$</td>
<td>$-3.2975 \times 10^3$</td>
<td>16.9629</td>
</tr>
<tr>
<td>$G_{500}(0.9, 1)$</td>
<td>MCMC</td>
<td>1.2130 \times 10^4</td>
<td>1.2214 \times 10^4</td>
<td>16.8376</td>
</tr>
<tr>
<td></td>
<td>Exact</td>
<td>1.2130 \times 10^4</td>
<td>1.2214 \times 10^4</td>
<td>16.8380</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>1.2135 \times 10^4</td>
<td>1.2219 \times 10^4</td>
<td>16.8503</td>
</tr>
<tr>
<td>$G_{500}(0.9, 100)$</td>
<td>MCMC</td>
<td>5.9382 \times 10^4</td>
<td>5.9466 \times 10^4</td>
<td>16.8427</td>
</tr>
<tr>
<td></td>
<td>Exact</td>
<td>5.9382 \times 10^4</td>
<td>5.9466 \times 10^4</td>
<td>16.9406</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>5.9411 \times 10^4</td>
<td>5.9495 \times 10^4</td>
<td>16.9640</td>
</tr>
<tr>
<td>Earthquake</td>
<td>MCMC</td>
<td>1.4814 \times 10^4</td>
<td>1.4823 \times 10^4</td>
<td>16.8305</td>
</tr>
<tr>
<td></td>
<td>Exact</td>
<td>1.4814 \times 10^4</td>
<td>1.4823 \times 10^4</td>
<td>16.8299</td>
</tr>
</tbody>
</table>

Table 5.1: AIC, BIC and $L^2$ analysis for Mittag-Leffler data sets. For simplicity, the set of generated data consisting of $n$ points with true parameters $\mu^\ast$ and $\tau_0^\ast$ is represented as $G_n (\mu^\ast, \tau_0^\ast)$. 
Chapter 6

Conclusions

As discussed in section 1.2, having good network models is crucial in fields such as mathematical epidemiology. Current methods for underlying network dynamics are fairly limited in capturing the complexity of real-world behaviour without being fully dependent upon it.

In this thesis, I have presented an alternative option to these. Whilst on several key network properties this option cannot fully recreate behaviours seen in real-world social interaction networks, other properties and behaviours emerge from my models that do reflect real-world properties (such as node activity potential, component count and triangle count) as can be seen in Table 3.4. Additionally, results such as those in Table 3.5 indicate that the models I have presented could capture more properties with a greater deal of accuracy given adjustments to the parameters and distributions that I have used within my models.

Whilst I have not examined improvements explicitly, I have discussed in chapters 4 and 5 the methods by which this process could be achieved - in particular looking at the case of Mittag-Leffler data. I present a MCMC method that returns very similar parameter estimates to that of the exact Bayesian estimate (with much fewer calculations) in addition to a method for estimating the range for a uniform prior for the $\tau_0$ parameter. I have also briefly looked at information criteria methods, which can be used when refining the parameters and distributions within the models to select one that produces both a good fit and ensures a satisfactory balance between under-fitting and over-fitting the data I have. These methods for parameter estimation and model selection can also be applied to other modelling methods, such as that presented in subsection 4.3.2 where I was using ABC to estimate parameters for data regarding earthquake occurrence.

The hope is that whilst my models are not the solution to having an accurate description for the underlying network dynamics to be used in fields such as mathematical epidemiology, this thesis provides the foundations and tools for future research in approaching this problem from a new perspective.
Bibliography


Appendix A
Code Manuals

Here I shall present descriptions and dependencies of top-level functions, as well as working examples. All functions are working as of MATLAB v9.5 (R2018b) and toolboxes released for this version.

A.1 ABC_ml2.m


Key Uses  ABC calculations in subsections 4.3.3 and 4.3.4.

Dependencies  ml.m, mlrnd.m

MathWorks Products Required  MATLAB, Statistics and Machine Learning Toolbox, Curve Fitting Toolbox, Parallel Computing Toolbox, MATLAB Distributed Computing Server

Inputs

- data - 2 × n vector with the first row consisting of points x and the second row consisting of the equivalent values of the CCDF at x
- dataname - string giving a reference name to the data
- oF - string containing the relative path for the output directory
- vargin - if testing against known parameters, μ and τ₀ can be given here as 2 doubles.

Outputs

- soln_ABC - structure containing the means and standard deviations of the estimates for μ and τ₀

1See appendices B.50 and B.52 for code.
Description  Returns means and standard deviations for parameter estimates of a Mittag-Leffler distribution using ABC. Additionally, will produce a figure containing the ECCDF of given data, the CCDF with given known parameters (if given), and CCDF curves using the estimates and both 1 and 2 standard deviations away. Will save input data, results and figure to a given output directory.

Working Example  soln_ABC = ABC_ml2(testdata,'example','output',0.5,1)

A.2  aicbic_ML.m

Summary  Returns AIC, BIC and $L^2$ values for given Mittag-Leffler data and estimated parameters. Code is given in appendix B.3.

Key Uses  AIC and BIC calculations in section 5.3.

Dependencies  ml.m


Inputs

- data - 1 x m vector of data
- m1 - 1 x n vector of $\beta$ parameters for Mittag-Leffler fits
- m2 - 1 x n vector of $\tau_0$ parameters for Mittag-Leffler fits

Outputs

- AIC - 1 x n vector of AIC values of the given Mittag-Leffler fits
- BIC - 1 x n vector of BIC values of the given Mittag-Leffler fits
- L2 - 1 x n vector of $L^2$ values of the given Mittag-Leffler fits

Description  This gives the values for AIC, BIC and $L^2$ values for n Mittag-Leffler fits when tested against a 1 x m vector of data.

Working Example  [aic,bic,L2] = aicbic_ML(data,0.5,1)

A.3  analyse.m

Summary  Analyses a given CSV file returning best fits for chosen properties, graphical representations of data and statistical distances. Code is given in appendix B.4.  

2See appendix B.50 for code.
**Key Uses**  Data analysis performed to obtain key distributions in section 2.2 and row sums in subsection 3.2.4.

**Dependencies**  analyse_ActiveEdges.m, analyse_ActiveNodes.m, analyse_ActivityPotential.m, analyse_ComponentEdges.m, analyse_ComponentNodes.m, analyse_GlobalClusteringCoeff.m, analyse_interactionTimes.m, analyse_NumberComponents.m, analyse_T imeBetweenContacts.m, buildStruc_ExpGamRayLN_FitTool.m, buildStruc_ExpGamRayLN_MLE.m, buildStruc_ExpGamRayLN_Moments.m, buildStruc_ExpMLGPWei_FitTool.m, buildStruc_ExpMLGPWei_MLE.m, buildStruc_ExpMLGPWei_Moments.m, dataMinMax.m, gpSolve.m, JSDiv.m, KLDiv.m, la2latex.m, mlf.m, mlrnd.m, mm_ExpGamRayLN.m, mm_ExpMLGPWei.m, networkComponents.m, num2matlabstr.m, pvals_ex.m, pvals_gm.m, pvals_gp.m, pvals_ln.m, pvals_ml.m, pvals_rl.m, pvals_wb.m, testStatistics.m


**Inputs**

- **input_folder** - string pointing to location of CSV file
- **input_filename** - string naming the file to be analysed
- **structure** - string labelling the format of the CSV file
- **timestamp** - string containing date/time information

**Outputs**

- **data2global** - structure containing vectors for data of each of the tested properties
- **mins** - structure containing minimum values for each parameter for each of the tested properties and methods
- **maxs** - structure containing maximum values for each parameter for each of the tested properties and methods

**Description**  Analyses a given CSV file returning structures containing best-fit parameters and related statistics for multiple properties using the MATLAB fit tool, method of moments and MLE. It will save this information to disk, along with outputting it into MATLAB. It will also produce CCDFs showing extracted data and the best-fit curves and files containing this information ready for compiling with \LaTeX.  

Working Example  
[analysis, mins, maxs] = analyse('inputfolder',  
'testdata.csv', '%f %f %f *s %s',  
'20150917T133000')

A.4  analyse_interevents.m

Summary  
Analyses a given CSV file returning best fits for interevent times,  
graphical representations of data and statistical distances. Code is given in appendix B.12

Key Uses  
Data analysis performed to obtain interevent distribution in section 2.2

Dependencies  
gpSolve.m, JSDiv.m, KLDiv.m, mlf.m, testStatistics.m

MathWorks Products Required  

Inputs
- input_folder - string pointing to location of CSV file
- input_filename - string naming the file to be analysed

Outputs
- dataout - structure containing best fit parameters and statistics for all tested distributions

Description  
Analyses a given CSV file returning structures containing best-fit parameters and related statistics for interevent times using the MATLAB fit tool. It will save this information to disk, along with outputting it into MATLAB. It will also produce CCDFs showing extracted data and the best-fit curves and files containing this information ready for compiling with \LaTeX.

Working Example  
analysis = analyse_interevents('inputfolder',  
'testdata.csv')

A.5  compareData.m

Summary  
Compares two sets of data calculating statistical distances and creating plots representing data. Code is given in appendix B.23

Key Uses  
Model analysis in section 3.3 including Table 3.4

Dependencies  
JSDiv.m, KLDiv.m, makeGraphs.m, networkComponents.m,  
pullData.m, stats_KLJS.m

4See appendices B.39, B.41, B.42, B.51 and B.81 for code.

Inputs

- genFolder - string pointing to first directory
- realFolder - string pointing to second directory

Outputs  None

Description  Compares two folders containing CSV files, plotting the CCDF of each set separately on the same axis, plotting two CCDFs on the same axis representing the ‘average’ behaviour of each folder, and calculating the Kolmogorov-Smirnov distances and the Kullback-Leibler and Jensen-Shannon divergences between these two folders of data.

Working Example  compareData(‘inputfolder1’,‘inputfolder2’)

A.6  compareOriginal.m

Summary  Examine a folder of data sets and tests the Kolmogorov-Smirnov hypothesis that this data all comes from the same underlying distribution. Code is given in appendix B.24.

Key Uses  Analysis of original data to examine if all data sets can be assumed to be generated identically, discussed in subsection 3.1.1 and presented in Table 3.1

Dependencies  acceptances.m, networkComponents.m, pullData.m


Inputs

- folder - string pointing to directory

Outputs  None

Description  Runs through a folder of \( n \) files, comparing them pairwise on multiple properties using the Kolmogorov-Smirnov test that they both come from the same underlying distribution. For each property, it will produce a \( \LaTeX \)-ready file containing the number of acceptances (out of a maximum of \( \frac{n(n-1)}{2} \)) for each property at the 1%, 5% and 10% levels.

Working Example  compareOriginal(‘inputfolder’)

\textsuperscript{6}See appendices B.2, B.62 and B.66 for code.
A.7 comparison.m

Summary Produces simulated data for each of the 4 model variations, then compares the data produced to a folder of observed data, testing the Kolmogorov-Smirnov hypothesis that these data sets come from the same underlying distribution. Code is given in appendix [B.25]

Key Uses Comparison of model variations as used in section [3.4] and presented in Table [3.4]

Dependencies calculateDistance.m, chooselink.m, distanceStruct.c.m, EAP_matrix.m, extractTriangles.m, model.m, modelv2.1.m, modelv2a.1.m, networkComponents.m, pullData.m, rndLPM.m, sampleCSV.m, sampleCSV2.m, sigma_for_mu_and_mean.m


Inputs
- **folder** - string pointing to directory containing observed data
- **count** - double containing number of samples for each model variation to generate ($n$)
- **timelength** - double containing simulation time for each model variation in seconds

Outputs None

Description Produces $n$ simulations using each model variation, then pairwise compares generated data to $m$ sets of observed data, using the Kolmogorov-Smirnov test that they both come from the same underlying distribution. For each property and variation, it will produce a \texttt{LaTeX}-ready file containing the number of acceptances (out of a maximum of $mn$) for each property at the 1%, 5% and 10% levels.

Working Example \texttt{comparison('inputfolder',20,20000)}

A.8 echoes.m

Summary Shows an animation for a CSV file showing network evolution in time. Code is given in appendix [B.35]

Key Uses Production of animations included in supplemental materials and used for Figure [3.2]

Dependencies None

\textsuperscript{7}See appendices [B.21, B.22, B.30, B.34, B.38, B.55, B.57, B.59, B.62, B.66, B.74, B.75, B.76] for code.
\textsuperscript{8}Available from [https://drive.google.com/open?id=13eUXp3xGph1BST7dLPvEVPZcHyACmom](https://drive.google.com/open?id=13eUXp3xGph1BST7dLPvEVPZcHyACmom).
MathWorks Products Required  MATLAB, Parallel Computing Toolbox, MATLAB Distributed Computing Server

Inputs

- **datapath** - string pointing to the CSV file
- **sampletime** - maximum time to sample from

Outputs  None

Description  This shows an animation showing the evolution in time of data extracted from a CSV file showing the link spread (see section 3.3) up to the current point in time.

Working Example  `echoes('dataset.csv',20000)`

A.9  `explognconvhist.m`

Summary  Compares in-built convolve function against MCMC simulations for a combination of exponential and log-normal distributions for $t$ seconds. Code is given in appendix B.37.

Key Uses  Used for the generation of Figure 3.1

Dependencies  None


Inputs

- **$t$** - double representing length of simulation time

Outputs

- **pdf** - vector of PDF values from $n = 0$ up to a set maximum

Description  Calculates and plots PDF values for the number of renewals in a process with interevent times consisting of the result of convolving exponential and log-normal distributions using a MCMC method and the built-in MATLAB `convolve` function.

Working Example  `pdf = explognconvhist(2000)`
A.10 MCMCSP_mlf2.m

Summary  Produces several graphical visualisations and Bayesian estimates for Mittag-Leffler data using MCMC estimation and exact methods. Code is given in appendix B.49.

Key Uses  Analysis of data using exact Bayesian estimate and MCMC methods throughout chapter 4.

Dependencies  MCMCestimatesig_mlf2.m, MCMCmultiCW_mlf2.m, ml.m, scaledmarginals.m, scaledposterior2_mlf2.m


Inputs
- **data** - $1 \times n$ vector of data
- **dataname** - string giving a reference name to the data
- **of** - string containing the relative path for the output directory
- **vargin** - if testing against known parameters, $\mu$ and $\tau_0$ can be given here as 2 doubles.

Outputs
- **soln_MCMC** - structure containing the means and standard deviations of the estimates for $\mu$ and $\tau_0$ using MCMC method
- **soln_SP** - structure containing the means and standard deviations of the estimates for $\mu$ and $\tau_0$ using exact method

Description  Returns means and standard deviations for parameter estimates of a Mittag-Leffler distribution using MCMC and exact methods. Additionally, will produce a figure containing the ECCDF of given data, the CCDF with given known parameters (if given), and CCDF curves using the estimates and both 1 and 2 standard deviations away. Will save input data, results and figure to a given output directory.

Working Example  
```matlab
[soln_MCMC, soln_SP] = MCMCSP_mlf2(testdata, 'example', 'output', 0.5, 1)
```

A.11 triangleClosed.m

Summary  Calculates the fraction of triangle closures and times at which one is not possible, given a CSV data file. Code is given in appendix B.82.

See appendices B.47, B.48, B.50, B.77 and B.78 for code.
**Key Uses**  Analysis for choice of triangle ‘forcing’ rate used in Model 2b and 2c, discussed in subsection 3.2.3.

**Dependencies**  None

**MathWorks Products Required**  MATLAB, Parallel Computing Toolbox, MATLAB Distributed Computing Server

**Inputs**
- `input_folder` - string pointing to location of CSV file
- `input_filename` - string naming the file to be analysed

**Outputs**
- `frac` - $1 \times n$ vector containing triangle closures as a fraction of total new links for each of the $n$ time steps closures up to
- `FiT` - double containing triangle closures as a fraction of total new links for the entire data set
- `NTP` - double containing fraction of time that no triangle closure is possible for the entire data set

**Description**  Returns a vector showing the evolution through time of the number of new links that changed an open triple to a closed triple as a fraction of total new links. Also returns the fraction of time no closure was possible (e.g. no triples exist, or all triples are closed) and the fraction of closures over the entire time period.

**Working Example**  
```
[fracvec, closures, noclosures] = triangleClosed('inputfolder','testdata.csv')
```

## A.12 validation.m

**Summary**  Creates validation counts for each model variation and metric. Code is given in appendix B.86.

**Key Uses**  Validation of model variations as used in section 3.5 and presented in Tables 3.5.

**Dependencies**  calculateDistance.m, chooselink.m, distanceStruc.c.m, EAP_matrix.m, emprand.m, extractTriangles.m, networkComponents.m, pullData.m, sampleCSV.m, sampleCSV2.m, validate1.m, validate2a.m, validate2b.m

**MathWorks Products Required**  MATLAB, Statistics and Machine Learning Toolbox, Parallel Computing Toolbox, MATLAB Distributed Computing Server

---

10See appendices B.21, B.22, B.30, B.34, B.36, B.38, B.62, B.66, B.75, B.76, B.83, B.84 and B.85 for code.
Inputs

- **folder** - string pointing to location of observed data
- **count** - double representing the number of validation samples to generate
- **timelength** - double representing the simulation time of the validations

Outputs    None

**Description**  Performs validation test by generating samples using distributions extracted directly by data before performing pairwise Kolmogorov-Smirnov tests at the 5% level that validation data sets and observed data come from the same distribution. Returns these as a table of results in a \LaTeX-ready file.

**Working Example**  validation('datafolder',5,20000)
Appendix B

Code Listing

Below I shall present all custom MATLAB functions used in this thesis. Details on top-level functions can be found in appendix A and all functions below are available for download in the supplemental materials.

B.1 ABC_m12.m

```matlab
function soln_ABC = ABC_ml2(data, dataname, oF, varargin)
% Approximate Bayesian Computation for the Mittag-Leffler distribution

%=== SETTINGS ===%
rangeMu = [0.01, 0.99]; % Search range for mu
nloops = 100; % Minimum number of acceptances needed
eps = 0.05; % Distance for acceptance
nrand = 10000; % Number of random points
tauWindow = 0.1; % Extent tau0 from by this proportion
paracount = 2; % Parameter count
%=== SETTINGS ===%
parchains = max(2, maxNumCompThreads);
adjnloops = ceil(nloops/parchains);
oF_eps = [oF, '/eps'];
mkdir(oF);
mkdir(oF_eps);
save([oF, '/', dataname, '_data.mat'], 'data');
DX = data(1,:);
```

1Available from https://drive.google.com/open?id=13eUXp3xGph1B5T7dIPvEVPZcHyACmom
DY = data(2,:);

idx = find(exp(-1)<DY & DY<0.5);
if idx(1) == 1
  lb = 0;
else
  lb = DX(idx(1)-1);
end
if idx(end) == length(DX)
  f = fit(DY',DX', 'exp1');
  est = (f.a)*exp((f.b)*exp(-1));
  ub = 10^((ceil(log10(est))));
else
  ub = DX(idx(end)+1);
end
rangeTau = [(1- tauWindow)*lb, (1+ tauWindow)*ub];
pararange = [rangeMu; rangeTau];

%=== PRIORS ===%
priors.p1 = makedist('Uniform', 'lower', pararange(1,1), 'upper', pararange(1,2)); % Prior for first parameter
priors.p2 = makedist('Uniform', 'lower', pararange(2,1), 'upper', pararange(2,2)); % Prior for second parameter
%=== PRIORS ===%

MU=zeros(parchains, adjnloops);
TAU0=zeros(parchains, adjnloops);

% Arrays of parameters
parfor i=1:parchains
  cMU = [];
  cTAU0 = [];
  while length(cMU)<adjnloops % Random walk on parameters
    thismu = random(priors.p1);
    thistau0 = random(priors.p2);
    % Generation of generalised beta distributed data
    X=mlrnd(thismu, thistau0, 1, nrand);
    % Compute ecdf
    [FF,XX] = ecdf(X);
    try
      YQ = interp1(XX(2:end), 1-FF(2:end), DX);
      % Distance between quantiles
      dis = abs(DY-YQ);
      dismax = max(dis);
      if dismax < eps
        cMU = [cMU, thismu];
      end
    catch
      disp('Error in generating data or quantiles.');
    end
    cMU = [cMU, thismu];
  end
  cTAU0 = [cTAU0, thistau0];
end

% Process cMU and cTAU0 to finalize parameters
MU = mean(cMU); % Mean of scale parameters
TAU0 = mean(cTAU0); % Mean of shape parameters
cTAU0 = [cTAU0, thistau0];
end

catch
    % Do nothing - distance is too great!
end
MU(i,:) = cMU;
TAU0(i,:) = cTAU0;
end
MU = reshape(MU,1,[]);
TAU0 = reshape(TAU0,1,[]);

% Estimate of parameters
MMU = mean(MU);
MTAU0 = mean(TAU0);
SMU = std(MU);
STAU0 = std(TAU0);

ABC1 = ml(-(DX/(MTAU0+(2*STAU0))).^((MMU-(2*SMU)),MMU-(2*SMU)));
ABC2 = ml(-(DX/(MTAU0+(1*STAU0))).^((MMU-(1*SMU)),MMU-(1*SMU)));
ABC3 = ml(-(DX/MTAU0).^MMU,MMU);
ABC4 = ml(-(DX/(MTAU0-(1*STAU0))).^((MMU+(1*SMU)),MMU+(1*SMU)));
ABC5 = ml(-(DX/(MTAU0-(2*STAU0))).^((MMU+(2*SMU)),MMU+(2*SMU)));

fig = figure('DefaultAxesFontSize',18);
loglog(DX,DY,'Marker','x','MarkerSize',12,'LineStyle','none');
hold on
if nargin==paracount+3
    TRULINE = ml(-(DX/varargin{2}).^varargin{1},varargin{1});
    plot(DX,TRULINE,'Color',[0 0 0],'LineWidth',1.5);
end
plot(DX,ABC1,'.',Color','r','LineWidth',1.5);
plot(DX,ABC2,'--','Color','r','LineWidth',1.5);
plot(DX,ABC3,'-','Color','r','LineWidth',1.5);
plot(DX,ABC4,'--','Color','r','LineWidth',1.5);
plot(DX,ABC5,'.',Color','r','LineWidth',1.5);
xlabel('Time Between Events');
ylabel('CCDF');
title('Waiting Times');
fig.WindowState = 'maximized';
saveas(fig,[oF,'/','dataname','_ccdf.fig']);
saveas(fig,[oF_eps,'\'/',dataname,'_ccdf'],'epsc')

close(fig);

soln_ABC.mean = [MMU MTAU0];
soln_ABC.std = [SMU STAU0];
save([oF,'\'/',dataname,'_results.mat'],'soln_ABC');

B.2 acceptances.m

function [a1,a5,a10] = acceptances(data1,data2)
% ACCEPTANCES takes two vectors of data and returns whether the
% null-hypothesis for the Kolmogorov-Smirnov test is accepted at the 1, 5
% and 10 percent levels.
% 
% DATA1 is the first data vector
% DATA2 is the second data vector
% A1 is a binary value that takes the value 0 if H0 is rejected at the 1
% percent level and 1 if it is accepted
% A5 is a binary value that takes the value 0 if H0 is rejected at the 5
% percent level and 1 if it is accepted
% A10 is a binary value that takes the value 0 if H0 is rejected at the 10
% percent level and 1 if it is accepted

[thisH1,~,~] = kstest2(data1,data2,'Alpha',0.01);
[thisH5,~,~] = kstest2(data1,data2,'Alpha',0.05);
[thisH10,~,~] = kstest2(data1,data2,'Alpha',0.1);
a1 = 1-thisH1;
a5 = 1-thisH5;
a10 = 1-thisH10;
end

B.3 aicbic_ML.m

function [aic,bic,L2] = aicbic_ML(data,m1,m2)
%AICBIC_ML returns AIC, BIC and L2 values for given
% Mittag-Leffler data and
% estimated parameters
%
%DATA is the given sample vector
% M1 is the estimated first parameter(s)
% M2 is the estimated second parameter(s)
%
%AIC is the AIC value for these parameter(s)
%BIC is the BIC value for these parameter(s)
% L2 is the L2 distance for these parameter(s)

ll1 = zeros(1,length(m1));
L2 = zeros(1,length(m1));
[F,X] = ecdf(data);
G = 1-F;
for j=1:length(m1)
    indivterms = vpa(zeros(1,length(data)));
    dist2 = vpa(zeros(1,length(data)));
    m1j = m1(j);
    m2j = m2(j);
    parfor i=1:length(data)
        indivterms(i) = (1/data(i))*(data(i)/m2j)ˆm1j *ml
        (-((data(i)/m2j)^m1j,m1j,m1j));
    end
    parfor i=1:length(X)
        dist2(i) = ((ml(-(X(i)/m2j)^m1j,m1j,0))-G(i))ˆ2;
    end
    L2(j) = double(sqrt(sum(dist2)));
    prob = prod(indivterms);
    ll1(j) = double(log(prob));
end
[aic,bic] = aicbic(ll1,2,length(data));

**B.4 analyse.m**

```matlab
function [data2global,mins,maxs] = analyse(input_folder, input_filename, structure, timestamp)
% ANALYSE analyses a CSV file calculating best fit distributions, producing
% graphical representations, calculating statistical distances and
% estimating p-values
%
% INPUT_FOLDER is the folder location given as a string
% INPUT_FILENAME is the file name given as a string
% STRUCTURE is the format of the CSV file given as a string
% TIMESTAMP is the current time used to name the output folder given as a
```
102

% string
% DATA2GLOBAL is a structure containing the extracted data
% MINS is a structure containing minimum values for fitted distributions
% MAXS is a structure containing maximum values for fitted distributions

iF = ['input/','input_folder'];
oF = ['output_','timestamp'];
clean_input = strrep(input_filename,'.','
');
dir_ref = [oF,'\',clean_input];
mkdir(dir_ref);
input = [iF,'/input_filename];

fid = fopen(input);
rawdata = textscan(fid,structure,'Delimiter','
','
');fclose(fid);

%== Extract and Clean Data==%
data = cell2mat(rawdata);
data(:,1) = data(:,1)-data(1,1);
lowestID = min(min(data(:,2)),min(data(:,3)));
data(:,2) = data(:,2)-lowestID+1;
data(:,3) = data(:,3)-lowestID+1;
number_rows = size(data,1);
parfor i=1:number_rows
    thisrow = data(i,:);
    col2 = thisrow(1,2);
    col3 = thisrow(1,3);
    if col2 > col3
        thisrow(1,2) = col3;
        thisrow(1,3) = col2;
        data(i,:) = thisrow;
    end
end
all_IDs = [data(:,2); data(:,3)];
all_active = unique(all_IDs);
num_people = size(all_active,1);
data2 = data(:,2);
data3 = data(:,3);
for i=1:num_people
    oldID = all_active(i);
data2(data2==oldID) = -i;
data3(data3==oldID) = -i;
end
%== Perform Analysis ==%

[ActiveLinks_data, ActiveLinks_FitTool, ActiveLinks_MLE, 
ActiveLinks_Moments] = analyse_ActiveEdges(data, dir_ref);

[NodesActive_data, NodesActive_FitTool, NodesActive_MLE, 
NodesActive_Moments] = analyse_ActiveNodes(data, dir_ref);

[ActivityPotential_data, ActivityPotential_FitTool, 
ActivityPotential_MLE, ActivityPotential_Moments] = 
analyse_ActivityPotential(data, dir_ref);

[Clustering_data, Clustering_FitTool, Clustering_MLE, 
Clustering_Moments] = analyse_GlobalClusteringCoeff(data, dir_ref);

[InteractionTimes_data, InteractionTimes_FitTool, 
InteractionTimes_MLE, InteractionTimes_Moments] = 
analyse_InteractionTimes(data, dir_ref);

[Components_data, Components_FitTool, Components_MLE, 
Components_Moments] = analyse_NumberComponents(data, dir_ref);

[ComponentNodes_data, ComponentNodes_FitTool, 
ComponentNodes_MLE, ComponentNodes_Moments] = 
analyse_ComponentNodes(data, dir_ref);

[ComponentEdges_data, ComponentEdges_FitTool, 
ComponentEdges_MLE, ComponentEdges_Moments] = 
analyse_ComponentEdges(data, dir_ref);

%== Post-Processing & Export ==%

datafilename = [dir_ref, '/Distributions.mat'];
save(datafilename,...
    'ActiveLinks_FitTool',...
    'InteractionTimes_FitTool',...
    'ActivityPotential_FitTool',...
    'NoContactTimes_FitTool',...
    'NodesActive_FitTool',...
    'Components_FitTool',...
    'Clustering_FitTool',...
    'ComponentNodes_FitTool',...
    'ComponentEdges_FitTool',...
    'ActiveLinks_Moments',...
    'InteractionTimes_Moments',...
    'ActivityPotential_Moments',...
    'NoContactTimes_Moments'],
Analysis = struct(
    'ActiveLinks_FitTool',
    ActiveLinks_FitTool,
    'ActiveLinks_Moments',
    ActiveLinks_Moments,
    'ActiveLinks_MLE',
    ActiveLinks_MLE,
    'InteractionTimes_FitTool',
    InteractionTimes_FitTool,
    'InteractionTimes_Moments',
    InteractionTimes_Moments,
    'InteractionTimes_MLE',
    InteractionTimes_MLE,
    'ActivityPotential_FitTool',
    ActivityPotential_FitTool,
    'ActivityPotential_Moments',
    ActivityPotential_Moments,
    'ActivityPotential_MLE',
    ActivityPotential_MLE,
    'NoContactTimes_FitTool',
    NoContactTimes_FitTool,
    'NoContactTimes_Moments',
    NoContactTimes_Moments,
    'NoContactTimes_MLE',
    NoContactTimes_MLE,
    'NodesActive_FitTool',
    NodesActive_FitTool,
    'NodesActive_Moments',
    NodesActive_Moments,
    'NodesActive_MLE',
    NodesActive_MLE,
    'Components_FitTool',
    Components_FitTool,
    'Components_Moments',
    Components_Moments,
    'Components_MLE',
    Components_MLE,
    'Clustering_Moments',
    Clustering_Moments,
    'Clustering_MLE',
    Clustering_MLE,
    'ComponentNodes_Moments',
    ComponentNodes_Moments,
    'ComponentNodes_MLE',
    ComponentNodes_MLE,
    'ComponentEdges_Moments',
    ComponentEdges_Moments,
    'ComponentEdges_MLE',
    ComponentEdges_MLE,
);
'Components_MLE', Components_MLE, ...
'Clustering_FitTool',
    Clustering_FitTool, ...
'Clustering_Moments',
    Clustering_Moments, ...
'Clustering_MLE', Clustering_MLE, ...
'ComponentNodes_FitTool',
    ComponentNodes_FitTool, ...
'ComponentNodes_Moments',
    ComponentNodes_Moments, ...
'ComponentNodes_MLE',
    ComponentNodes_MLE, ...
'ComponentEdges_FitTool',
    ComponentEdges_FitTool, ...
'ComponentEdges_Moments',
    ComponentEdges_Moments, ...
'ComponentEdges_MLE',
    ComponentEdges_MLE, ...
'NumberPeople', num_people);

data2global = struct('ActiveLinks_data', ActiveLinks_data,
    'InteractionTimes_data', InteractionTimes_data, ...
    'ActivityPotential_data', ActivityPotential_data, ...
    'NoContactTimes_data', NoContactTimes_data, ...
    'NodesActive_data', NodesActive_data, ...
    'Components_data', Components_data, ...
    'Clustering_data', Clustering_data, ...
    'ComponentNodes_data', ComponentNodes_data, ...
    'ComponentEdges_data', ComponentEdges_data, ...
    'NumberPeople', num_people);

[mins, maxs] = dataMinMax(Analysis);

la2latex(ActiveLinks_FitTool, ActiveLinks_MLE,
    ActiveLinks_Moments, num_people, dir_ref, 'Active Links',
    1);
la2latex(InteractionTimes_FitTool, InteractionTimes_MLE,
    InteractionTimes_Moments, num_people, dir_ref, 'Int. Times',
    2);
la2latex(ActivityPotential_FitTool, ActivityPotential_MLE,
    ActivityPotential_Moments, num_people, dir_ref, 'Activity
B.5 analyse_ActiveEdges.m

```matlab
function [data2global, FitTool, MLE, Moments] = analyse_ActiveEdges(data, dir_ref)

% ANALYSE_ACTIVEEDGES analyses the active edges
% DATA is the data matrix produced during ANALYSE
% DIR_REF is the save directory given as a text string
% DATA2GLOBAL is a vector of individual measurements of this metric
% FITTOOL is a structure containing parameters, statistics and pvalues for
% the fit tool
% MLE is a structure containing parameters, statistics and pvalues for
% the maximum likelihood estimators
% MOMENTS is a structure containing parameters, statistics and pvalues for
% the method of moments

Ymin = 1E-3;
cutExtreme = 3;
num_times = size(unique(data(:,1)),1);
data_length = size(data(:,1),1);
num_people = max([data(:,2); data(:,3)]);
```
contact_time = 20;

links = zeros(1,num_times);
maxlinks = num_people*(num_people-1)/2;

parfor m=1:num_times
    thisadj = zeros(num_people);
    current_time = (m-1)*contact_time;
    for i=1:data_length
        test_time = data(i,1);
        if test_time==current_time
            person1 = data(i,2);
            person2 = data(i,3);
            thisadj(person1,person2) = 1;
            thisadj(person2,person1) = 1;
        end
    end
    adjsum = sum(sum(thisadj));
    numlinks = adjsum/2;
    links(m) = numlinks/maxlinks;
end

FitTool = buildStruc_ExpGamRayLN_FitTool(links,dir_ref,'ActiveEdges','Fraction of Edges Active',cutExtreme,Ymin);
MLE = buildStruc_ExpGamRayLN_MLE(links,dir_ref,'ActiveEdges','Fraction of Edges Active',cutExtreme,Ymin);
Moments = buildStruc_ExpGamRayLN_Moments(links,dir_ref,'ActiveEdges','Fraction of Edges Active',cutExtreme,Ymin);
data2global = links;
end

B.6 analyse_ActiveNodes.m

function [data2global,FitTool,MLE,Moments] = analyse_ActiveNodes(data,dir_ref)
% ANALYSE_ACTIVENODES analyses the active nodes
% DATA is the data matrix produced during ANALYSE
% DIR_REF is the save directory given as a text string
% DATA2GLOBAL is a vector of individual measurements of this metric
% FITTOOL is a structure containing parameters, statistics and pvalues for
% the fit tool
% MLE is a structure containing parameters, statistics
  and pvalues for
% the maximum likelihood estimators
% MOMENTS is a structure containing parameters,
  statistics and pvalues for
% the method of moments

Ymin = 1E-4;
cutExtreme = 1;

num_times = size(unique(data(:,1)),1);
data_length = size(data(:,1),1);
num_people = max([data(:,2); data(:,3)]);
contact_time = 20;

nodes = zeros(1,num_times);

parfor m=1:num_times
    thisactive = zeros(1,num_people);
    current_time = (m-1)*contact_time;
    for i=1:data_length
        test_time = data(i,1);
        if test_time==current_time
            person1 = data(i,2);
            person2 = data(i,3);
            thisactive(person1) = 1;
            thisactive(person2) = 1;
        end
    end
    nodes(m) = sum(thisactive)/num_people;
end

FitTool = buildStruc_ExpGamRayLN_FitTool(nnodes, dir_ref,
    'ActiveNodes', 'Fraction of Nodes Active', cutExtreme,
    Ymin);
MLE = buildStruc_ExpGamRayLN_MLE(nnodes, dir_ref,
    'ActiveNodes', 'Fraction of Nodes Active', cutExtreme,
    Ymin);
Moments = buildStruc_ExpGamRayLN_Moments(nnodes, dir_ref,
    'ActiveNodes', 'Fraction of Nodes Active', cutExtreme,
    Ymin);
data2global = nodes;
% ANALYSE_ACTIVITYPOTENTIAL analyses the node activity potentials

% DATA is the data matrix produced during ANALYSE
% DIR_REF is the save directory given as a text string
% DATA2GLOBAL is a vector of individual measurements of this metric
% FITTOOL is a structure containing parameters, statistics and pvalues for
% the fit tool
% MLE is a structure containing parameters, statistics and pvalues for
% the maximum likelihood estimators
% MOMENTS is a structure containing parameters, statistics and pvalues for
% the method of moments

Ymin = 1E-1;
cutExtreme = 3;

number_rows = size(data,1);
number_people = max([data(:,2); data(:,3)]);
contact_time = 20;

%== Sort Data by ID ==%
[~, order] = sort(data(:,3));
partsorteddata = data(order,:);
[~, order] = sort(partsorteddata(:,2));
sorteddata = partsorteddata(order,:);

%== Find Interaction Times ==%
j = 1;
interactions = zeros(1,number_people);
step_vector = [contact_time 0 0];
while j<number_rows+1
   ID1 = sorteddata(j,2);
   ID2 = sorteddata(j,3);
   interactions(ID1) = interactions(ID1)+1;
   interactions(ID2) = interactions(ID2)+1;
   current_row = sorteddata(j,:);
   if j == number_rows
      next_row = [0 0 0];
   else
      %... continue the code...
next_row = sorteddata(j+1,:);
end
while isequal(next_row,current_row+step_vector)
    j = j+1;
current_row = sorteddata(j,:);
    if j == number_rows
        next_row = [0 0 0];
    else
        next_row = sorteddata(j+1,:);
    end
end
j = j+1;
end
activityPot = 2*interactions/sum(interactions);
FitTool = buildStruc_ExpGamRayLN_FitTool(activityPot,
dir_ref,'ActivityPotential','Activity Potential',
cutExtreme,Ymin);
MLE = buildStruc_ExpGamRayLN_MLE(activityPot,dir_ref,'ActivityPotential','Activity Potential',
cutExtreme,Ymin);
Moments = buildStruc_ExpGamRayLN_Moments(activityPot,
dir_ref,'ActivityPotential','Activity Potential',
cutExtreme,Ymin);
data2global = activityPot;
end

B.8 analyse_ComponentEdges.m

function [data2global,FitTool,MLE,Moments] =
analyse_ComponentEdges(data,dir_ref)

% ANALYSE_COMPONENTEDGES analyses the edges per component
% DATA is the data matrix produced during ANALYSE
% DIR_REF is the save directory given as a text string
% DATA2GLOBAL is a vector of individual measurements of
% this metric
% FITTOOL is a structure containing parameters, statistics and pvalues for
% the fit tool
% MLE is a structure containing parameters, statistics and pvalues for
% the maximum likelihood estimators
% MOMENTS is a structure containing parameters, statistics and pvalues for
% the method of moments
Ymin = 1E-1;

cutExtreme = 3;

num_times = size(unique(data(:,1)),1);
data_length = size(data(:,1),1);
num_people = max([data(:,2); data(:,3)]);
contact_time = 20;

rawFracEdges = zeros(num_times,num_people);

parfor m=1:num_times
    thisadj = zeros(num_people);
current_time = (m-1)*contact_time;
for i=1:data_length
    test_time = data(i,1);
    if test_time==current_time
        person1 = data(i,2);
person2 = data(i,3);
        thisadj(person1,person2) = 1;
        thisadj(person2,person1) = 1;
    end
end
edgesActive = sum(sum(thisadj));
 [~,~ ,thisCompGroups] = networkComponents(thisadj);
thisNumComps = length(thisCompGroups);
thisEdges = zeros(1,thisNumComps);
for j=1:thisNumComps
    thisNodes = cell2mat(thisCompGroups(j));
    thisNodesSize = length(thisNodes);
    if thisNodesSize == 1
        thisEdges(j)=0;
    else
        thisSubMat = thisadj(thisNodes,thisNodes);
        thisAdjSum = sum(sum(thisSubMat));
        thisNumEdges = thisAdjSum/2;
        thisEdges(j) = thisNumEdges;
    end
end
if edgesActive > 0
    thisEdges = thisEdges/edgesActive;
end
thisPadding = num_people - length(thisEdges);
thisEdges = [thisEdges zeros(1,thisPadding)];
rawFracEdges(m,:) = thisEdges;
end
compEdgeFracs = rawFracEdges(:);
compEdgeFracs(compEdgeFracs==0) = [];

FitTool = buildStruc_ExpGamRayLN_FitTool(compEdgeFracs,
dir_ref,'ComponentEdges','Fraction of Edges Active per Component',cutExtreme,Ymin);
MLE = buildStruc_ExpGamRayLN_MLE(compEdgeFracs,dir_ref,'ComponentEdges','Fraction of Edges Active per Component',cutExtreme,Ymin);
Moments = buildStruc_ExpGamRayLN_Moments(compEdgeFracs,
dir_ref,'ComponentEdges','Fraction of Edges Active per Component',cutExtreme,Ymin);
data2global = compEdgeFracs;

B.9 analyse_ComponentNodes.m

function [data2global,FitTool,MLE,Moments] = analyse_ComponentNodes(data,dir_ref)

% ANALYSE_COMPONENTNODES analyses the nodes per component
% DATA is the data matrix produced during ANALYSE
% DIR_REF is the save directory given as a text string
% DATA2GLOBAL is a vector of individual measurements of this metric
% FITTOOL is a structure containing parameters, statistics and pvalues for
% the fit tool
% MLE is a structure containing parameters, statistics and pvalues for
% the maximum likelihood estimators
% MOMENTS is a structure containing parameters, statistics and pvalues for
% the method of moments

Ymin = 1E-1;
cutExtreme = 1;

num_times = size(unique(data(:,1)),1);
data_length = size(data(:,1),1);
num_people = max([data(:,2); data(:,3)]);
contact_time = 20;
rawCompSizes = zeros(num_times,num_people);
parfor m=1:num_times
    thisadj = zeros(num_people);
    current_time = (m-1)*contact_time;
    for i=1:data_length
        test_time = data(i,1);
        if test_time==current_time
            person1 = data(i,2);
            person2 = data(i,3);
            thisadj(person1,person2) = 1;
            thisadj(person2,person1) = 1;
        end
    end
    [~,thisCompSizes,~] = networkComponents(thisadj);
    thisCompSizes(thisCompSizes==1)=[];
    nodesActive = sum(thisCompSizes);
    thisPadding = num_people - length(thisCompSizes);
    thisCompSizes = [thisCompSizes zeros(1,thisPadding)];
    if nodesActive == 0
        thisCompFracs = thisCompSizes;
    else
        thisCompFracs = thisCompSizes/nodesActive;
    end
    rawCompSizes(m,:)=thisCompFracs;
end
compSizes = rawCompSizes(:,');
compSizes(compSizes==0) = [];
FitTool = buildStruc_ExpGamRayLN_FitTool(compSizes,
    dir_ref,'ComponentNodes','Fraction of Nodes per Component',cutExtreme,Ymin);
MLE = buildStruc_ExpGamRayLN_MLE(compSizes,dir_ref,'ComponentNodes','Fraction of Nodes per Component',
    cutExtreme,Ymin);
Moments = buildStruc_ExpGamRayLN_Moments(compSizes,
    dir_ref,'ComponentNodes','Fraction of Nodes per Component',
    cutExtreme,Ymin);
data2global = compSizes;
end

B.10 analyse_GlobalClusteringCoeff.m

function [data2global,FitTool,MLE,Moments] = analyse_GlobalClusteringCoeff(data,dir_ref)
%ANALYSE_GLOBALCLUSTERINGCOEFF analyses the global clustering coefficient
Ymin = 1E-2;

cutExtreme = 0;

num_times = size(unique(data(:,1)),1);
data_length = size(data(:,1),1);
num_people = max([data(:,2); data(:,3)]);
contact_time = 20;

clustering = zeros(1,num_times);

parfor m=1:num_times
    thisadj = zeros(num_people);
    current_time = (m-1)*contact_time;
    for i=1:data_length
        test_time = data(i,1);
        if test_time==current_time
            person1 = data(i,2);
            person2 = data(i,3);
            thisadj(person1,person2) = 1;
            thisadj(person2,person1) = 1;
        end
    end
    adj2 = thisadj^2;
    adj3 = thisadj^3;
    adj2sum = sum(sum(adj2));
    contrip = adj2sum - trace(adj2);
    if contrip==0
        clustering(m) = 0;
    else
        clustering(m) = trace(adj3)/contrip;
    end
end
clustering_autocorr = clustering;

clustering(clustering==0) = [];

FitTool = buildStruc_ExpGamRayLN_FitTool(clustering, dir_ref,'GlobalClusteringCoeff','Global Clustering Coefficient',cutExtreme,Ymin);
MLE = buildStruc_ExpGamRayLN_MLE(clustering,dir_ref,'GlobalClusteringCoeff','Global Clustering Coefficient',cutExtreme,Ymin);
Moments = buildStruc_ExpGamRayLN_Moments(clustering, dir_ref,'GlobalClusteringCoeff','Global Clustering Coefficient',cutExtreme,Ymin);

data2global = clustering;

maxTime = (num_times-1)*contact_time;
T = linspace(0,maxTime,num_times);

hwin = 50;
Tmod = T;
Tmod((num_times+1-hwin):num_times) = [];
Tmod(1:hwin) = [];

MA = zeros(1,num_times-2*hwin);
parfor i=1:(num_times-2*hwin)
    upper = i+2*hwin;
    MA(i) = sum(clustering_autocorr(i:upper))/(2*hwin+1);
end

clusteringfig = figure();
hold on
plot(T,clustering_autocorr)
plot(Tmod,MA,'LineWidth',4)
xlabel('Time (s)');
ylabel('Global Clustering Coefficient');
hold off
imagefilename = [dir_ref,'/GlobalClusteringCoeff_Additional_MovingAverage.png'];
print(imagefilename,'-dpng')
close(clusteringfig);

autocorrfig = figure();
subplot(3,1,1);
autocorr(clustering_autocorr,length(clustering_autocorr)-1);
subplot(3,1,2);
smallerlag = min(round(length(clustering_autocorr)-1,-2) /4,length(clustering_autocorr)-1);
autocorr(clustering_autocorr,smallerlag);
subplot(3,1,3);
evensmallerlag = min(round(length(clustering_autocorr) -1,-2)/16,length(clustering_autocorr)-1);
autocorr(clustering_autocorr,evensmallerlag);
imagefilename = [dir_ref,'/GlobalClusteringCoeff_Additional_AutoCorrelation.png' ];
print(imagefilename,'-dpng');
close(autocorrfig);
end

B.11 analyse_InteractionTimes.m

function [data2global,FitTool,MLE,Moments] = analyse_InteractionTimes(data,dir_ref)
% ANALYSE_INTERACTIONTIMES analyses the interaction times
% %
% DATA is the data matrix produced during ANALYSE
% % DIR_REF is the save directory given as a text string
% % DATA2GLOBAL is a vector of individual measurements of this metric
% % FITTOOL is a structure containing parameters, statistics and pvalues for
% % the fit tool
% % MLE is a structure containing parameters, statistics and pvalues for
% % the maximum likelihood estimators
% % MOMENTS is a structure containing parameters, statistics and pvalues for
% % the method of moments
% %
Ymin = 1E-3;
cutExtreme = 3;
number_rows = size(data,1);
contact_time = 20;
%== Sort Data by ID ==%
 [~, order] = sort(data(:,3));
partsorteddata = data(order,:);
 [~, order] = sort(partsorteddata(:,2));
sorteddata = partsorteddata(order,:);
%== Find Interaction Times ==%
times = zeros(1,number_rows);
j = 1;
times_k = 1;
step_vector = [contact_time 0 0];
while j<number_rows+1
    contact = contact_time;
current_row = sorteddata(j,:);
    if j == number_rows
        next_row = [0 0 0];
    else
        next_row = sorteddata(j+1,:);
    end
    while isequal(next_row,current_row+step_vector)
        contact = contact+contact_time;
j = j+1;
current_row = sorteddata(j,:);
        if j == number_rows
            next_row = [0 0 0];
        else
            next_row = sorteddata(j+1,:);
        end
    end
    times(times_k) = contact;
j = j+1;
times_k = times_k+1;
end
times(times_k:end) = [];

FitTool = buildStruc_ExpMLGPWei_FitTool(times,dir_ref,'
    InteractionTimes','Length of Interaction',cutExtreme,Ymin);
MLE = buildStruc_ExpMLGPWei_MLE(times,dir_ref,'
    InteractionTimes','Length of Interaction',cutExtreme,Ymin);
Moments = buildStruc_ExpMLGPWei_Moments(times,dir_ref,'
    InteractionTimes','Length of Interaction',cutExtreme,Ymin);
data2global = times;
end

B.12 analyse_interevents.m

function [dataout] = analyse_interevents(input_folder,
    input_filename)
% ANALYSE_INTEREVENTS extracts interevent data from a
given file and
% attempts to fit distributions and give statistical
distances
%
% INPUT_FOLDER is the location of the file, given as a
string
% INPUT_FILENAME is the filename, given as a string
% DATAOUT is a structure containing best-fit parameters
and statistics for
% all tested distributions

timestamp = datestr(now,'yyyyymmddTHHMSS');
structure = '%f %f %f %s %s';
iF = ['input/','input_folder'];
oF = ['output_','timestamp'];
clean_input = strrep(input_filename, '.', 'ts1');
dir_ref = [oF, '\', clean_input];
mkdir(dir_ref);
input = [iF, '/', input_filename];
fid = fopen(input);
rawdata = textscan(fid, structure, 'Delimiter', '%', '%s');
close(fid);

%== Extract and Clean Data==
data = cell2mat(rawdata);
data(:,1) = data(:,1) - data(1,1);
lowestID = min(min(data(:,2)), min(data(:,3)));
data(:,2) = data(:,2) - lowestID + 1;
data(:,3) = data(:,3) - lowestID + 1;
number_rows = size(data,1);
parfor i=1:number_rows
    thisrow = data(i,:);
    col2 = thisrow(1,2);
    col3 = thisrow(1,3);
    if col2 > col3
        thisrow(1,2) = col3;
        thisrow(1,3) = col2;
        data(i,:) = thisrow;
    end
end
all_IDs = [data(:,2); data(:,3)];
all_active = unique(all_IDs);
num_people = size(all_active,1);
data2 = data(:,2);
data3 = data(:,3);
for i=1:num_people
    oldID = all_active(i);
data2(data2==oldID) = -i;
data3(data3==oldID) = -i;
end
data(:,2) = -data2;
data(:,3) = -data3;
maxN=max(max(data(:,2)),max(data(:,3))); activationtimes = [];
for i=1:maxN-1
    for j=i+1:maxN
        S1 = data(:,2)==i;
        S2 = data(:,3)==j;
        T1 = data(:,3)==i;
        T2 = data(:,2)==j;
        S12 = S1 & S2;
        T12 = T1 & T2;
        ST12 = S12 | T12;
        changes = diff(ST12);
        binary = [0; changes];
        thistimes = data(binary==1,1);
        activationtimes = [activationtimes; thistimes];
    end
end
sorted = sort(activationtimes);
timebetween = diff(sorted);
[F,X] = ecdf(timebetween);
ccdf = 1-F;
M1 = mean(timebetween);
M2 = mean(timebetween.^2);
M3 = mean(timebetween.^3);
ex_lambda_start = M1;
if ex_lambda_start<=0 || isnan(ex_lambda_start) || isinf(ex_lambda_start)
ex_lambda_start = 0.1;
end
gm_b_start = (M2/M1)-M1;
gm_a_start = M1/gm_b_start;
if gm_b_start<=0 || isnan(gm_b_start) || isinf(gm_b_start)
gm_b_start = 0.1;
end
if gm_a_start <=0 || isnan(gm_a_start) || isnan(gm_a_start)
gm_a_start = 0.1;
end
rl_sigma_start = M1*sqrt(2/pi);
if rl_sigma_start <=0 || isnan(rl_sigma_start) || isnan(rl_sigma_start)
rl_sigma_start = 0.1;
end
ln_sigma_start = sqrt(log(M2*(M1^-2)));
if ln_sigma_start <=0 || isnan(ln_sigma_start) || isnan(ln_sigma_start)
ln_sigma_start = 0.1;
end
ln_mu_start = log(M1)-(0.5*ln_sigma_start^2) ;
if isnan(ln_mu_start) || isnan(ln_mu_start)
ln_mu_start = 0;
end
ml_beta_start = 0.5;
ml_gamma_start = 0.5;
[gp_k_start,gp_sigma_start, gp_theta_start] = gpSolve(M1,M2,M3);
if isnan(gp_k_start) || isnan(gp_k_start)
gp_k_start = 0;
end
if gp_sigma_start <=0 || isnan(gp_sigma_start) || isnan(gp_sigma_start)
gp_sigma_start = 0.1;
end
if isnan(gp_theta_start) || isnan(gp_theta_start)
gp_theta_start = 0;
end
wb_a_start = 0.5;
wb_b_start = 0.5;

try
    fo_ex = fitoptions('Method', 'NonlinearLeastSquares',
    'Lower', [0], 'Upper', [inf], 'StartPoint', [ex_lambda_start]);
    ft_ex = fittype('expcdf(x, lambda, 'upper')', 'options', fo_ex);
    [cf_ex, ~] = fit(X, ccdf, ft_ex);
    cv_ex = coeffvalues(cf_ex);
catch
    cv_ex = [ex_lambda_start];
end
try
    fo_gm = fitoptions('Method', 'NonlinearLeastSquares',
                      'Lower', [0 0], 'Upper', [inf inf], 'StartPoint', [
                        gm_a_start gm_b_start]);
    ft_gm = fittype('gamcdf(x,a,b,''upper'')', 'options',
                    fo_gm);
    [cf_gm,~] = fit(X,ccdf,ft_gm);
    cv_gm = coeffvalues(cf_gm);
    catch
        cv_gm = [gm_a_start gm_b_start];
    end

try
    fo_rl = fitoptions('Method', 'NonlinearLeastSquares',
                        'Lower', [0], 'Upper', [inf], 'StartPoint', [
                        rl_sigma_start]);
    ft_rl = fittype('raylcdf(x,sigma,''upper'')', 'options',
                    fo_rl);
    [cf_rl,~] = fit(X,ccdf,ft_rl);
    cv_rl = coeffvalues(cf_rl);
    catch
        cv_rl = [rl_sigma_start];
    end

try
    fo_ln = fitoptions('Method', 'NonlinearLeastSquares',
                        'Lower', [-inf 0], 'Upper', [inf inf], 'StartPoint', [
                        ln_mu_start ln_sigma_start]);
    ft_ln = fittype('logncdf(x,mu,sigma,''upper'')',
                    options, fo_ln);
    [cf_ln,~] = fit(X,ccdf,ft_ln);
    cv_ln = coeffvalues(cf_ln);
    catch
        cv_ln = [0 1];
    end

try
    fo_ml = fitoptions('Method', 'NonlinearLeastSquares',
                        'Lower', [0 0], 'Upper', [1 1], 'StartPoint', [
                        ml_beta_start ml_gamma_start]);
    ft_ml = fittype('mlf(beta,1,-gamma*x.^beta,6)', 'options',
                    fo_ml);
    [cf_ml,~] = fit(X,ccdf,ft_ml);
    cv_ml = coeffvalues(cf_ml);
    catch
        cv_ml = [ml_beta_start ml_gamma_start];
    end
try
    fo_gp = fitoptions('Method', 'NonlinearLeastSquares', 'Lower', [-inf -inf 0], 'StartPoint', [gp_k_start gp_sigma_start gp_theta_start]);
    ft_gp = fittype('gpcdf(x,k,sigma,theta,' upper')', 'options', fo_gp);
    [cf_gp, ~] = fit(X, ccdf, ft_gp);
    cv_gp = coeffvalues(cf_gp);
catch
    cv_gp = [gp_k_start gp_sigma_start gp_theta_start];
end

try
    fo_wb = fitoptions('Method', 'NonlinearLeastSquares', 'Lower', [0 0], 'StartPoint', [wb_a_start wb_b_start]);
    ft_wb = fittype('wblcdf(x,a,b,' upper')', 'options', fo_wb);
    [cf_wb, ~] = fit(X, ccdf, ft_wb);
    cv_wb = coeffvalues(cf_wb);
catch
    cv_wb = [wb_a_start wb_b_start];
end

ex_lambda = cv_ex(1);
gm_a = cv_gm(1);
gm_b = cv_gm(2);
rl_sigma = cv_rl(1);
ln_mu = cv_ln(1);
ln_sigma = cv_ln(2);
ml_beta = cv_ml(1);
ml_gamma = cv_ml(2);
gp_k = cv_gp(1);
gp_sigma = cv_gp(2);
gp_theta = cv_gp(3);
wb_a = cv_wb(1);
wb_b = cv_wb(2);
sortedtimes = sort(timebetween);

z_ex = expcdf(sortedtimes, ex_lambda);
z_gm = gamcdf(sortedtimes, gm_a, gm_b);
z_rl = raylcdf(sortedtimes, rl_sigma);
z_ln = logncdf(sortedtimes, ln_mu, ln_sigma);
z_ml = ones(length(sortedtimes), 1) - mlf(ml_beta, 1, ml_gamma .* sortedtimes.^ ml_beta, 6);
z_gp = gpcdf(sortedtimes, gp_k, gp_sigma, gp_theta);
z_wb = wblcdf(sortedtimes,wb_a,wb_b);
zp_ex = exppdf(sortedtimes,ex_lambda);
zp_gm = gampdf(sortedtimes,gm_a,gm_b);
zp_r1 = rayldpdf(sortedtimes,rl_sigma);
zp_ln = lognpdf(sortedtimes,ln_mu,ln_sigma);
zp_ml = (-ml_beta./sortedtimes).*mlf(ml_beta,1,-ml_gamma.*sortedtimes.^ml_beta,6);
zp_gp = gppdf(sortedtimes,gp_k,gp_sigma,gp_theta);

stats_ex = testStatistics(sortedtimes,zp_ex,zp_ex,20);
stats_gm = testStatistics(sortedtimes,zp_gm,zp_gm,20);
stats_r1 = testStatistics(sortedtimes,zp_r1,zp_r1,20);
stats_ln = testStatistics(sortedtimes,zp_ln,zp_ln,20);
stats_ml = struct(); %Issues with input here
stats_gp = testStatistics(sortedtimes,zp_gp,zp_gp,20);
stats_wb = testStatistics(sortedtimes,zp_wb,zp_wb,20);

struc_ex = struct('Scale',ex_lambda);
struc_gm = struct('Shape',gm_a,'Scale',gm_b);
struc_r1 = struct('Scale',rl_sigma);
struc_ln = struct('Location',ln_mu,'Scale',ln_sigma);
struc_ml = struct('Stability',ml_beta,'Scale',ml_gamma);
struc_gp = struct('Shape',gp_k,'Scale',gp_sigma,'Location',gp_theta);
struc_wb = struct('Scale',wb_a,'Shape',wb_b);

EX = struct('Parameters',struc_ex,'Statistics',stats_ex);
GM = struct('Parameters',struc_gm,'Statistics',stats_gm);
RL = struct('Parameters',struc_r1,'Statistics',stats_r1);
LN = struct('Parameters',struc_ln,'Statistics',stats_ln);
ML = struct('Parameters',struc_ml,'Statistics',stats_ml);
GP = struct('Parameters',struc_gp,'Statistics',stats_gp);
WB = struct('Parameters',struc_wb,'Statistics',stats_wb);

dataout = struct('Exponential',EX,'Gamma',GM,'Rayleigh',RL,'LogNormal',LN,'MittagLeffler',ML,'GenPareto',GP,'Weibull',WB);

%== Plotting==
ccdf_ex = expcdf(X,ex_lambda,'upper');
ccdf_gm = gamcdf(X,gm_a,gm_b,'upper');
ccdf_r1 = raylcdf(X,rl_sigma,'upper');
ccdf_ln = logncdf(X,ln_mu,ln_sigma,'upper');
ccdf_ml = mlf(ml_beta,1,-ml_gamma*X.^ml_beta,6);
ccdf_gp = gpcdf(X,gp_k,gp_sigma,gp_theta,'upper');
ccdf_wb = wblcdf(X,wb_a,wb_b,'upper');
fig = figure();
hold on
plot(X,ccdf,'o');
plot(X,ccdf_ex);
plot(X,ccdf_gm);
plot(X,ccdf_rl);
plot(X,ccdf_ln);
plot(X,ccdf_ml);
plot(X,ccdf_gp);
plot(X,ccdf_wb);
set(gca,'XScale','log');
set(gca,'YScale','log');
xlabel('Time Between Activations');
ylabel('CCDF');
axis([-inf,inf,1E-4,1E0]);
legend('Data','Exponential','Gamma','Rayleigh','Log-Normal','Mittag Leffler','Gen. Pareto','Weibull','Location','northeast');
imagefilename = [dir_ref,'/between_activations.png'];
figfilename = [dir_ref,'/between_activations'];
print(imagefilename,'-dpng')
savefig(figfilename);
close(fig);
end

B.13 analyse_NumberComponents.m

function [data2global,FitTool,MLE,Moments] = analyse_NumberComponents(data,dir_ref)

function [data2global,FitTool,MLE,Moments] = analyse_NumberComponents(data,dir_ref)

% ANALYSE_NUMBERCOMPONENTS analyses the component count
% DATA is the data matrix produced during ANALYSE
% DIR_REF is the save directory given as a text string
% DATA2GLOBAL is a vector of individual measurements of
% this metric
% FITTOOL is a structure containing parameters,
% statistics and pvalues for
% the fit tool
% MLE is a structure containing parameters, statistics
% and pvalues for
% the maximum likelihood estimators
% MOMENTS is a structure containing parameters,
% statistics and pvalues for
% the method of moments
Ymin = 1E-4;
cutExtreme = 0;
num_times = size(unique(data(:,1)),1);
data_length = size(data(:,1),1);
num_people = max([data(:,2); data(:,3)]);
contact_time = 20;
components = zeros(1,num_times);

parfor m=1:num_times
    thisadj = zeros(num_people);
    current_time = (m-1)*contact_time;
    for i=1:data_length
        test_time = data(i,1);
        if test_time==current_time
            person1 = data(i,2);
            person2 = data(i,3);
            thisadj(person1,person2) = 1;
            thisadj(person2,person1) = 1;
        end
    end
    [~,thisComp,~] = networkComponents(thisadj);
    thisComp(thisComp==1) = [];% Fixpoint
    thisCompCount = length(thisComp);
    components(m) = thisCompCount;
end

FitTool = buildStruc_ExpGamRayLN_FitTool(components, dir_ref, 'NumberComponents', 'Number of Components', cutExtreme, Ymin);
MLE = buildStruc_ExpGamRayLN_MLE(components, dir_ref, 'NumberComponents', 'Number of Components', cutExtreme, Ymin);
Moments = buildStruc_ExpGamRayLN_Moments(components, dir_ref, 'NumberComponents', 'Number of Components', cutExtreme, Ymin);
data2global = components;

B.14 analyse_TimeBetweenContacts.m

function [data2global,FitTool,MLE,Moments] = 
analyse_TimeBetweenContacts(data,dir_ref)
% ANALYSE_TIMEBETWEENCONTACTS analyses the time between contacts
% DATA is the data matrix produced during ANALYSE
% DIR_REF is the save directory given as a text string
% DATA2GLOBAL is a vector of individual measurements of this metric
% FITTOOL is a structure containing parameters, statistics and pvalues for
% the fit tool
% MLE is a structure containing parameters, statistics and pvalues for
% the maximum likelihood estimators
% MOMENTS is a structure containing parameters, statistics and pvalues for
% the method of moments

Ymin = 1E-4;
cutExtreme = 3;
step = 20;
min_time = min(data(:,1));
max_time = max(data(:,1));
times = ((max_time-min_time)/step)+1;
data_length = size(data(:,1),1);
num_people = max([data(:,2); data(:,3)]);
rawactivity = zeros(data_length,num_people+1);

parfor i=1:data_length
    thisrawactivity = zeros(1,num_people+1);
    thisrawactivity(1) = data(i,1);
    person1 = data(i,2);
    person2 = data(i,3);
    thisrawactivity(person1+1) = 1;
    thisrawactivity(person2+1) = 1;
    rawactivity(i,:) = thisrawactivity;
end

activity = zeros(times,num_people);

parfor i=1:times
    currenttime = ((i-1)*step)+min_time;
    activerows = rawactivity(rawactivity(:,1)==
        currenttime,:));
    activerows = activerows(:,2:end);
    thisactivity = sum(activerows,1);
    thisactivity = (thisactivity>0);
activity(i,:) = thisactivity;
end

activity = [activity; ones(1,num_people)];

long = activity(:);
long = long';
dlong = diff([1 long 1]);
startIndex = find(dlong < 0);
endIndex = find(dlong > 0)-1;
nocontact = endIndex-startIndex+1;
nocontact = nocontact*20;

FitTool = buildStruc_ExpGamRayLN_FitTool(nocontact, dir_ref,'TimeBetweenContacts','Length of Time Between Contacts',cutExtreme,Ymin);
MLE = buildStruc_ExpGamRayLN_MLE(nocontact,dir_ref,'TimeBetweenContacts','Length of Time Between Contacts',cutExtreme,Ymin);
Moments = buildStruc_ExpGamRayLN_Moments(nocontact, dir_ref,'TimeBetweenContacts','Length of Time Between Contacts',cutExtreme,Ymin);
data2global = nocontact;
end

B.15  buildStruc_ExpGamRayLN_FitTool.m

function [Structure] = buildStruc_ExpGamRayLN_FitTool(
data,dir_ref,property_title,graph_title,cutExtreme,Ymin)

%BUILDSTRUC_EXPGAMRAYLN_FITTOOL finds the best fit
% Exponential,
% Gamma, Rayleigh and Log-Normal distributions for the
given data using the
% fit tool saving graphs, statistics and p-values
%
% DATA is the data matrix
% DIR_REF is the save directory
% PROPERTY_TITLE is the name of the property being
% analysed
% GRAPH_TITLE is the desired name for the graph
% CUTEXTREME is the number of extreme points to be
% removed
% YMIN is the lower limit on the Y-axis on the graph
% STRUCTURE is a structure containing parameters,
% statistics and p-values
MC_Power = 6;

%== Prepare data ==%
[F,X] = ecdf(data);
ccdf = 1-F;
if cutExtreme >0
    Xrem = [X(end+1-cutExtreme:end)];
else
    Xrem = [];
end
X = X(2:end-cutExtreme);
ccdf = ccdf(2:end-cutExtreme);
dataMod = data(˜ismember(data,Xrem));
test_data = sort(dataMod);
difference = diff(test_data);
difference = difference(difference>0);
res = min(difference);

%== Choose initial values (using MoM) ==%
M1 = mean(dataMod);
M2 = mean(dataMod.^2);

ex_lambda_start = M1;
gm_b_start = (M2/M1)-M1;
gm_a_start = M1/gm_b_start;
rl_sigma_start = M1*sqrt(2/pi);
ln_sigma_start = sqrt(log(M2*(M1^-2)));
ln_mu_start = log(M1)-(0.5*ln_sigma_start^2);

%== Fit distributions ==%
fo_ex = fitoptions('Method', 'NonlinearLeastSquares',
    'Lower', [0], 'Upper', [inf], 'StartPoint', [ex_lambda_start ]);
ft_ex = fittype('expcdf(x,lambda,''upper'')','options',
    fo_ex);
[cf_ex,~] = fit(X,ccdf,ft_ex);
cv_ex = coeffvalues(cf_ex);

fo_gm = fitoptions('Method', 'NonlinearLeastSquares',
    'Lower', [0 0], 'Upper', [inf inf], 'StartPoint', [gm_a_start gm_b_start]);
ft_gm = fittype('gamcdf(x,a,b,''upper'')','options',fo_gm);
[cf_gm,~] = fit(X,ccdf,ft_gm);
cv_gm = coeffvalues(cf_gm);
fo_rl = fitoptions('Method', 'NonlinearLeastSquares', 'Lower', [0], 'Upper', [inf], 'StartPoint', [rl_sigma_start]);
ft_rl = fittype('raylcdf(x,sigma,''upper''),'options', fo_rl);
[cf_rl,~] = fit(X,ccdf,ft_rl);
cv_rl = coeffvalues(cf_rl);
fo_ln = fitoptions('Method', 'NonlinearLeastSquares', 'Lower', [-inf 0], 'Upper', [inf inf], 'StartPoint', [ln_mu_start ln_sigma_start]);
ft_ln = fittype('logncdf(x,mu,sigma,''upper''),'options', fo_ln);
[cf_ln,~] = fit(X,ccdf,ft_ln);
cv_ln = coeffvalues(cf_ln);

%== Extract parameters ==%
ex_lambda = cv_ex(1);
ccdf_ex = expcdf(X, ex_lambda, 'upper');
gm_a = cv_gm(1);
gm_b = cv_gm(2);
ccdf_gm = gamcdf(X, gm_a, gm_b, 'upper');
rl_sigma = cv_rl(1);
ccdf_rl = raylcdf(X, rl_sigma, 'upper');
ln_mu = cv_ln(1);
ln_sigma = cv_ln(2);
ccdf_ln = logncdf(X, ln_mu, ln_sigma, 'upper');

%== Extract GoF data ==%
z_ex = expcdf(test_data, ex_lambda);
z_gm = gamcdf(test_data, gm_a, gm_b);
z_rl = raylcdf(test_data, rl_sigma);
z_ln = logncdf(test_data, ln_mu, ln_sigma);
zp_ex = exppdf(test_data, ex_lambda);
zp_gm = gampdf(test_data, gm_a, gm_b);
zp_rl = raylpdf(test_data, rl_sigma);
zp_ln = lognpdf(test_data, ln_mu, ln_sigma);

stats_ex = testStatistics(test_data, z_ex, zp_ex, 0);
stats_gm = testStatistics(test_data, z_gm, zp_gm, 0);
stats_rl = testStatistics(test_data, z_rl, zp_rl, 0);
stats_ln = testStatistics(test_data, z_ln, zp_ln, 0);

%== Plotting ==%
fig = figure();
hold on
plot(X, ccdf, 'o');
plot(X, ccdf_ex);
plot(X, ccdf_gm);
plot(X, ccdf_rl);
plot(X, ccdf_ln);
set(gca, 'XScale', 'log');
set(gca, 'YScale', 'log');
xlabel(graph_title);
ylabel('CCDF');
axis([-inf, inf, Ymin, 1E0]);
legend('Data', 'Exponential', 'Gamma', 'Rayleigh', 'Log-Normal', 'Location', 'southwest');
imagefilename = [dir_ref, '/', property_title, '_FitTool.png '];
print(imagefilename, '-dpng')
figfilename = [dir_ref, '/', property_title, '_FitTool'];
savefig(figfilename);
close(fig);

%== Build data structure ==%
samplesize = max(size(data));

struc_ex = struct('Scale', ex_lambda);
struc_gm = struct('Shape', gm_a, 'Scale', gm_b);
struc_rl = struct('Scale', rl_sigma);
struc_ln = struct('Location', ln_mu, 'Scale', ln_sigma);

p_ex = pvals_ex(samplesize, ex_lambda, stats_ex, cutExtreme, MC_Power, res);
p_gm = pvals_gm(samplesize, gm_a, gm_b, stats_gm, cutExtreme, MC_Power, res);
p_rl = pvals_rl(samplesize, rl_sigma, stats_rl, cutExtreme, MC_Power, res);
p_ln = pvals_ln(samplesize, ln_mu, ln_sigma, stats_ln, cutExtreme, MC_Power, res);

EX = struct('Parameters', struc_ex, 'Statistics', stats_ex, 'pValues', p_ex);
GM = struct('Parameters', struc_gm, 'Statistics', stats_gm, 'pValues', p_gm);
RL = struct('Parameters', struc_rl, 'Statistics', stats_rl, 'pValues', p_rl);
LN = struct('Parameters', struc_ln, 'Statistics', stats_ln, 'pValues', p_ln);
Structure = struct('Exponential',EX,'Gamma',GM,'Rayleigh',RL,'LogNormal',LN);
end

B.16 buildStruc_ExpGamRayLN_MLE.m

function [Structure] = buildStruc_ExpGamRayLN_MLE(data, dir_ref, property_title, graph_title, cutExtreme, Ymin)
% BUILDSTRUC_EXPGRAMRAYLN_MLE finds the best fit
% Exponential, Gamma, Rayleigh and Log-Normal distributions for the
given data using the
% most likelihood estimators saving graphs, statistics
% and p-values
%
% DATA is the data matrix
% DIR_REF is the save directory
% PROPERTY_TITLE is the name of the property being
% analysed
% GRAPH_TITLE is the desired name for the graph
% CUTEXTREME is the number of extreme points to be
% removed
% YMIN is the lower limit on the Y-axis on the graph
% STRUCTURE is a structure containing parameters,
% statistics and p-values
% for this method

MC_Power = 6;

%%%Prepare data%%
[F,X] = ecdf(data);
ccdf = 1-F;
if cutExtreme >0
    Xrem = [X(end+1-cutExtreme:end)];
else
    Xrem = [];
end
X = X(2:end-cutExtreme);
ccdf = ccdf(2:end-cutExtreme);
dataMod = data(~ismember(data,Xrem));
test_data = sort(dataMod);
difference = diff(test_data);
difference = difference(difference>0);
res = min(difference);
%%%Perform MLEs%%
mod_test_data = test_data;
mod_test_data(mod_test_data==0)=[ ];

phat_ex = mle(mod_test_data,'distribution','Exponential');
phat_gm = mle(mod_test_data,'distribution','Gamma');
phat_rl = mle(mod_test_data,'distribution','Rayleigh');
phat_ln = mle(mod_test_data,'distribution','Lognormal');

%== Extract parameters ==%
ex_lambda = phat_ex(1);
ccdf_ex = expcdf(X,ex_lambda,'upper');
gm_a = phat_gm(1);
gm_b = phat_gm(2);
ccdf_gm = gamcdf(X,gm_a,gm_b,'upper');
rl_sigma = phat_rl(1);
ccdf_rl = raylcdf(X,rl_sigma,'upper');
ln_mu = phat_ln(1);
ln_sigma = phat_ln(2);
ccdf_ln = logncdf(X,ln_mu,ln_sigma,'upper');

%== Extract GoF data ==%
z_ex = expcdf(test_data,ex_lambda);
z_gm = gamcdf(test_data,gm_a,gm_b);
z_rl = raylcdf(test_data,rl_sigma);
z_ln = logncdf(test_data,ln_mu,ln_sigma);

zp_ex = exppdf(test_data,ex_lambda);
zp_gm = gampdf(test_data,gm_a,gm_b);
zp_rl = raylpdf(test_data,rl_sigma);
zp_ln = lognpdf(test_data,ln_mu,ln_sigma);

stats_ex = testStatistics(test_data,z_ex,zp_ex,0);
stats_gm = testStatistics(test_data,z_gm,zp_gm,0);
stats_rl = testStatistics(test_data,z_rl,zp_rl,0);
stats_ln = testStatistics(test_data,z_ln,zp_ln,0);

%== Plotting ==%
fig = figure();
hold on
plot(X,ccdf,'o');
plot(X,ccdf_ex);
plot(X,ccdf_gm);
plot(X,ccdf_rl);
plot(X,ccdf_ln);
set(gca,'XScale','log');
```matlab
set(gca,'YScale','log');
xlabel(graph_title);
ylabel('CCDF');
axis([-inf,inf,Ymin,1E0]);
legend('Data','Exponential','Gamma','Rayleigh','Log-
Normal','Location','southwest');
imagefilename = [dir_ref,'/property_title','_MLE.png'];
print(imagefilename,'-dpng')
figfilename = [dir_ref,'/property_title','_MLE'];
savefig(figfilename);
close(fig);

%% Build data structure%%
samplesize = max(size(data));

struc_ex = struct('Scale',ex_lambda);
struc_gm = struct('Shape',gm_a,'Scale',gm_b);
struc_rl = struct('Scale',rl_sigma);
struc_ln = struct('Location',ln_mu,'Scale',ln_sigma);

p_ex = pvals_ex(samplesize,ex_lambda,stats_ex,cutExtreme,
MC_Power,res);
p_gm = pvals_gm(samplesize,gm_a,gm_b,stats_gm,cutExtreme,
MC_Power,res);
p_rl = pvals_rl(samplesize,rl_sigma,stats_rl,cutExtreme,
MC_Power,res);
p_ln = pvals_ln(samplesize,ln_mu,ln_sigma,stats_ln,
cutExtreme,MC_Power,res);

EX = struct('Parameters',struc_ex,'Statistics',stats_ex,'pValues',p_ex);
GM = struct('Parameters',struc_gm,'Statistics',stats_gm,'pValues',p_gm);
RL = struct('Parameters',struc_rl,'Statistics',stats_rl,'pValues',p_rl);
LN = struct('Parameters',struc_ln,'Statistics',stats_ln,'pValues',p_ln);

Structure = struct('Exponential',EX,'Gamma',GM,'Rayleigh'
,RL,'LogNormal',LN);
end
```

B.17  buildStruc_ExpGamRayLN_Moments.m

```matlab
function [Structure] = buildStruc_ExpGamRayLN_Moments(
data,dir_ref,property_title,graph_title,cutExtreme,
Ymin)
```
% BUILDSTRUC_EXPGAMRAYLN_MOMENTS finds the best fit
% Exponential, Gamma, Rayleigh and Log-Normal distributions for the
% given data using the
% method of moments saving graphs, statistics and p-values
%
% DATA is the data matrix
% DIR_REF is the save directory
% PROPERTY_TITLE is the name of the property being
% analysed
% GRAPH_TITLE is the desired name for the graph
% CUTEXTREME is the number of extreme points to be
% removed
% YMIN is the lower limit on the Y-axis on the graph
% STRUCTURE is a structure containing parameters,
% statistics and pvalues
% for this method
%
MC_Power = 6;

%== Prepare data ==%
[F,X] = ecdf(data);
ccdf = 1-F;
if cutExtreme >0
    Xrem = [X(end+1-cutExtreme:end)];
else
    Xrem = [];
end
X = X(2:end-cutExtreme);
ccdf = ccdf(2:end-cutExtreme);
dataMod = data(~ismember(data,Xrem));
test_data = sort(dataMod);
difference = diff(test_data);
difference = difference(difference>0);
res = min(difference);

%== Prepare MoM ==%
M1 = mean(dataMod);
M2 = mean(dataMod.^2);

%== Extract parameters ==%
ex_lambda = M1;
ccdf_ex = expcdf(X,ex_lambda,'upper');
gm_b = (M2/M1)-M1;
gm_a = M1/gm_b;
ccdf_gm = gamcdf(X,gm_a,gm_b,'upper');
\[
\begin{align*}
rl_{\sigma} &= M_1 \cdot \sqrt{\frac{2}{\pi}}; \\
ccdf_{rl} &= \text{raylcdf}(X, rl_{\sigma}, \text{'upper'}); \\
ln_{\sigma} &= \sqrt{\log(M_2 \cdot (M_1^{-2}))}; \\
ln_{\mu} &= \log(M_1) - (0.5 \cdot \ln_{\sigma}^{-2}); \\
ccdf_{ln} &= \text{logncdf}(X, ln_{\mu}, ln_{\sigma}, \text{'upper'});
\end{align*}
\]

\%== Extract GoF data ==%  
\[ 
\begin{align*}
z_{ex} &= \text{expcdf}(\text{test\_data}, ex_{\lambda}); \\
z_{gm} &= \text{gamcdf}(\text{test\_data}, gm_a, gm_b); \\
z_{rl} &= \text{raylcdf}(\text{test\_data}, rl_{\sigma}); \\
z_{ln} &= \text{logncdf}(\text{test\_data}, ln_{\mu}, ln_{\sigma}); \\
zp_{ex} &= \text{exppdf}(\text{test\_data}, ex_{\lambda}); \\
zp_{gm} &= \text{gampdf}(\text{test\_data}, gm_a, gm_b); \\
zp_{rl} &= \text{raylpdf}(\text{test\_data}, rl_{\sigma}); \\
zp_{ln} &= \text{lognpdf}(\text{test\_data}, ln_{\mu}, ln_{\sigma}); \\
\end{align*}
\]

\%== Plotting ==%  
\[ 
\begin{align*}
\text{fig} &= \text{figure}(); \\
\text{hold on} &; \\
\text{plot}(X, ccdf, 'o'); \\
\text{plot}(X, ccdf_{\text{ex}}); \\
\text{plot}(X, ccdf_{\text{gm}}); \\
\text{plot}(X, ccdf_{\text{rl}}); \\
\text{plot}(X, ccdf_{\text{ln}}); \\
\text{set}(\text{gca}, 'XScale', 'log'); \\
\text{set}(\text{gca}, 'YScale', 'log'); \\
\text{xlabel}(\text{graph\_title}); \\
\text{ylabel}('CCDF'); \\
\text{axis}([-\infty, \infty, Ymin, 1E0]); \\
\text{legend}('Data', 'Exponential', 'Gamma', 'Rayleigh', 'Log\text{-}Normal', 'Location', 'southwest'); \\
\text{imagefilename} &= [\text{dir\_ref}, '/', \text{property\_title}, '_Moments.png']; \\
\text{print}(\text{imagefilename}, '-dpng') \\
\text{figfilename} &= [\text{dir\_ref}, '/', \text{property\_title}, '_Moments']; \\
\text{savefig}(\text{figfilename}); \\
\text{close}(\text{fig}); \\
\end{align*}
\]

\%== Build data structure ==%  
\[ 
\begin{align*}
samplesize &= \text{max}(\text{size}(\text{data}));
\end{align*}
\]
struc_ex = struct('Scale',ex_lambda);
struc_gm = struct('Shape',gm_a,'Scale',gm_b);
struc_rl = struct('Scale',rl_sigma);
struc_ln = struct('Location',ln_mu,'Scale',ln_sigma);
p_ex = pvals_ex(samplesize,ex_lambda,stats_ex,cutExtreme,
                  MC_Power,res);
p_gm = pvals_gm(samplesize,gm_a,gm_b,stats_gm,cutExtreme,
                  MC_Power,res);
p_rl = pvals_rl(samplesize,rl_sigma,stats_rl,cutExtreme,
                  MC_Power,res);
p_ln = pvals_ln(samplesize,ln_mu,ln_sigma,stats_ln,
                  cutExtreme,MC_Power,res);
EX = struct('Parameters',struc_ex,'Statistics',stats_ex,'
             pValues',p_ex);
GM = struct('Parameters',struc_gm,'Statistics',stats_gm,'
             pValues',p_gm);
RL = struct('Parameters',struc_rl,'Statistics',stats_rl,'
             pValues',p_rl);
LN = struct('Parameters',struc_ln,'Statistics',stats_ln,'
             pValues',p_ln);
Structure = struct('Exponential',EX,'Gamma',GM,'Rayleigh'
                   ,RL,'LogNormal',LN);
end

function [Structure] = buildStruc_ExpMLGPWei_FitTool(data
          ,dir_ref,property_title,graph_title,cutExtreme,Ymin)

%BUILDSTRUC_EXPMLGPWEI_FITTOOL finds the best fit
% Exponential,
% Mittag-Leffler, Generalised Pareto and Weibull
% distributions for the given
% data using the fit tool saving graphs, statistics and p
% -values
%
% DATA is the data matrix
% DIR_REF is the save directory
% PROPERTY_TITLE is the name of the property being
% analysed
% GRAPH_TITLE is the desired name for the graph
% CUTFEXTREME is the number of extreme points to be
% removed
% YMIN is the lower limit on the Y-axis on the graph
% STRUCTURE is a structure containing parameters, 
  statistics and pvalues 
% for this method
MC_Power = 6;
MC_Power_ML = 3;

%== Prepare data==%
[F,X] = ecdf(data);
ccdf = 1-F;
if cutExtreme > 0
  Xrem = [X(end+1-cutExtreme:end)];
else
  Xrem = [];
end
X = X(2:end-cutExtreme);
ccdf = ccdf(2:end-cutExtreme);
dataMod = data(˜ismember(data,Xrem));
test_data = sort(dataMod);
difference = diff(test_data);
difference = difference(difference>0);
res = min(difference);

%== Choose initial values (using MoM)==%
M1 = mean(dataMod);
M2 = mean(dataMod.^2);
M3 = mean(dataMod.^3);

ex_lambda_start = M1;
ml_beta_start = 0.5;
ml_gamma_start = 0.5;
[gp_k_start, gp_sigma_start, gp_theta_start] = gpSolve(M1, M2, M3);
wb_a_start = 0.5;
wb_b_start = 0.5;

%== Fit distributions==%
fo_ex = fitoptions('Method', 'NonlinearLeastSquares', 'Lower', [0], 'Upper', [inf], 'StartPoint', [ex_lambda_start]);
ft_ex = fittype('expcdf(x,lambda,''upper''),'options', fo_ex);
[cf_ex,~] = fit(X,ccdf,ft_ex);
cv_ex = coeffvalues(cf_ex);

fo_ml = fitoptions('Method', 'NonlinearLeastSquares', 'Lower', [0 0], 'Upper', [1 1], 'StartPoint', [ml_beta_start ml_gamma_start]);
```
53  ft_ml = fittype('mlf(beta,1,-gamma*x.^beta,6)','options',
54       fo_ml);
55  [cf_ml,""] = fit(X,ccdf,ft_ml);
56  cv_ml = coeffvalues(cf_ml);
57  fo_gp = fitoptions('Method', 'NonlinearLeastSquares','
58       Lower',[-inf -inf 0], 'StartPoint',[gp_k_start
59       gp_sigma_start gp_theta_start]);
60  ft_gp = fittype('gpcdf(x,k,sigma,theta,''upper''),'options',fo_gp);
61  [cf_gp,""] = fit(X,ccdf,ft_gp);
62  cv_gp = coeffvalues(cf_gp);
63  fo_wb = fitoptions('Method', 'NonlinearLeastSquares','
64       Lower',[0 0], 'StartPoint',[wb_a_start wb_b_start]);
65  ft_wb = fittype('wblcdf(x,a,b,''upper''),'options',fo_wb);
66  [cf_wb,""] = fit(X,ccdf,ft_wb);
67  cv_wb = coeffvalues(cf_wb);
68  
69  %==Extract parameters==%
70  ex_lambda = cv_ex(1);
71  ccdf_ex = expcdf(X, ex_lambda ,''upper'');
72  ml_beta = cv_ml(1);
73  ml_gamma = cv_ml(2);
74  ccdf_ml = mlf(ml_beta,1,-ml_gamma*X.^ml_beta,6);
75  gp_k = cv_gp(1);
76  gp_sigma = cv_gp(2);
77  gp_theta = cv_gp(3);
78  ccdf_gp = gpcdf(X,gp_k , gp_sigma , gp_theta ,''upper'');
79  wb_a = cv_wb(1);
80  wb_b = cv_wb(2);
81  ccdf_wb = wblcdf(X,wb_a,wb_b ,''upper'');
82  
83  %==Extract GoF data==%
84  zp_ex = exppdf(test_data ,ex_lambda);
85  z_m = ones(length(test_data),1) -mlf(ml_beta,1,-ml_gamma*'
86       test_data.^ml_beta,6);
87  z_gp = gppdf(test_data,gp_k , gp_sigma , gp_theta);
88  z_wb = wblcdf(test_data,wb_a,wb_b); 
89  zp_ex = exppdf(test_data ,ex_lambda);
90  zp_m = (-ml_beta./test_data).*mlf(ml_beta,1,-ml_gamma*'
91       test_data.^ml_beta,6);
92  zp_gp = gppdf(test_data,gp_k , gp_sigma , gp_theta);
```
zp_wb = wblpdf(test_data,wb_a,wb_b);
stats_ex = testStatistics(test_data,z_ex,zp_ex,0);
stats_ml = testStatistics(test_data,z_ml,zp_ml,0);
stats_gp = testStatistics(test_data,z_gp,zp_gp,0);
stats_wb = testStatistics(test_data,z_wb,zp_wb,0);

%== Plotting ==%
fig = figure();
hold on
plot(X,ccdf,'o');
plot(X,ccdf_ex);
plot(X,ccdf_ml);
plot(X,ccdf_gp);
plot(X,ccdf_wb);
set(gca,'XScale','log');
set(gca,'YScale','log');
xlabel(graph_title);
ylabel('CCDF');
axis([-inf,inf,Ymin,1E0]);
legend('Data','Exponential','Mittag Leffler','Gen. Pareto','Weibull','Location','southwest');
imagefilename = [dir_ref,'/',property_title,'_FitTool.png '];
print(imagefilename,'-dpng')
figfilename = [dir_ref,'/',property_title,'_FitTool'];
savefig(figfilename);
close(fig);

%== Build data structure ==%
samplesize = max(size(data));
struc_ex = struct('Scale',ex_lambda);
struc_ml = struct('Stability',ml_beta,'Scale',ml_gamma);
struc_gp = struct('Shape',gp_k,'Scale',gp_sigma,'Location',gp_theta);
struc_wb = struct('Scale',wb_a,'Shape',wb_b);
p_ex = pvals_ex(samplesize,ex_lambda,stats_ex,cutExtreme,MC_Power,res);
p_ml = pvals_ml(samplesize,ml_beta,ml_gamma,stats_ml,cutExtreme,MC_Power_ML,res);
p_gp = pvals_gp(samplesize,gp_k,gp_sigma,gp_theta,stats_gp,cutExtreme,MC_Power,res);
p_wb = pvals_wb(samplesize,wb_a,wb_b,stats_wb,cutExtreme,MC_Power,res);
EX = struct('Parameters',struc_ex,'Statistics',stats_ex,'pValues',p_ex);
ML = struct('Parameters',struc_ml,'Statistics',stats_ml,'pValues',p_ml);
GP = struct('Parameters',struc_gp,'Statistics',stats_gp,'pValues',p_gp);
WB = struct('Parameters',struc_wb,'Statistics',stats_wb,'pValues',p_wb);
Structure = struct('Exponential',EX,'MittagLeffler',ML,'GenPareto',GP,'Weibull',WB);
end

function [Structure] = buildStruc_ExpMLGPWei_MLE(data,
dir_ref,property_title,graph_title,cutExtreme,Ymin)
% BUILDSTRUC_EXPMLGPWEI_MLE finds the best fit Exponential ,
% Mittag-Leffler, Generalised Pareto and Weibull distributions for the given
% data using the most likelihood estimators saving graphs , statistics and
% p-values
% DATA is the data matrix
% DIR_REF is the save directory
% PROPERTY_TITLE is the name of the property being analysed
% GRAPH_TITLE is the desired name for the graph
% CUTEXTREME is the number of extreme points to be removed
% YMIN is the lower limit on the Y-axis on the graph
% STRUCTURE is a structure containing parameters, statistics and pvalues
% for this method
MC_Power = 6;

%== Prepare data==%
[F,X] = ecdf(data);
ccdf = 1-F;
if cutExtreme>0
    Xrem = [X(end+1-cutExtreme:end)];
else
    Xrem = [];
end
X = X(2:end-cutExtreme);
ccdf = ccdf(2:end-cutExtreme);
dataMod = data(ismember(data,Xrem));
test_data = sort(dataMod);
difference = diff(test_data);
difference = difference(difference>0);
res = min(difference);

%== Perform MLEs ==%
mod_test_data = test_data;
mod_test_data(mod_test_data==0) = [];
phat_ex = mle(mod_test_data,'distribution','Exponential');
phat_wb = mle(mod_test_data,'distribution','Weibull');

%== Extract parameters ==%
ex_lambda = phat_ex(1);
ccdf_ex = expcdf(X,ex_lambda,'upper');
wb_a = phat_wb(1);
wb_b = phat_wb(2);
ccdf_wb = wblcdf(X,wb_a,wb_b,'upper');

%== Extract GoF data ==%
z_ex = expcdf(test_data,ex_lambda);
z_wb = wblcdf(test_data,wb_a,wb_b);
zp_ex = exppdf(test_data,ex_lambda);
zp_wb = wblpdf(test_data,wb_a,wb_b);

stats_ex = testStatistics(test_data,z_ex,zp_ex,0);
stats_wb = testStatistics(test_data,z_wb,zp_wb,0);

%== Plotting ==%
fig = figure();
hold on
plot(X,ccdf,'o');
plot(X,ccdf_ex);
plot(X,ccdf_wb);
set(gca,'XScale','log');
set(gca,'YScale','log');
xlabel(graph_title);
ylabel('CCDF');
axis([-inf,inf,Ymin,1E0]);
legend(['Data','Exponential','Weibull','Location','southwest']);
imagefilename = [dir_ref,'/property_title'_MLE.png];
print(imagefilename, '-dpng')
figfilename = [dir_ref,'/quotesingle.ts1',property_title,'_MLE'];
savefig(figfilename);
close(fig);

%== Build data structure ==%
samplesize = max(size(data));
struc_ex = struct('Scale',ex_lambda);
struc_wb = struct('Scale',wb_a,'Shape',wb_b);
p_ex = pvals_ex(samplesize,ex_lambda,stats_ex,cutExtreme,
    MC_Power,res);
p_wb = pvals_wb(samplesize,wb_a,wb_b,stats_wb,cutExtreme,
    MC_Power,res);
EX = struct('Parameters',struc_ex,'Statistics',stats_ex,'pValues',p_ex);
WB = struct('Parameters',struc_wb,'Statistics',stats_wb,'pValues',p_wb);
Structure = struct('Exponential',EX,'Weibull',WB);

function [Structure] = buildStruc_ExpMLGPWei_Moments(data,dir_ref,property_title,graph_title,cutExtreme,Ymin)

% BUILDSTRUC_EXPMLGPWEI_MOMENTS finds the best fit Exponential,
% Mittag-Leffler, Generalised Pareto and Weibull distributions for the given
% data using the method of moments saving graphs,
% statistics and p-values
%
% DATA is the data matrix
% DIR_REF is the save directory
% PROPERTY_TITLE is the name of the property being analysed
% GRAPH_TITLE is the desired name for the graph
% CUTEXTREME is the number of extreme points to be removed
% YMIN is the lower limit on the Y-axis on the graph
% STRUCTURE is a structure containing parameters,
% statistics and p-values
% for this method
MC_Power = 6;

%== Prepare data ==%
[F,X] = ecdf(data);
ccdf = 1-F;
if cutExtreme>0
    Xrem = [X(end+1-cutExtreme:end)];
else
    Xrem = [];
end
X = X(2:end-cutExtreme);
ccdf = ccdf(2:end-cutExtreme);
dataMod = data(~ismember(data,Xrem));
test_data = sort(dataMod);
difference = diff(test_data);
difference = difference(difference>0);
res = min(difference);

%== Prepare MoM ==%
M1 = mean(dataMod);
M2 = mean(dataMod.^2);
M3 = mean(dataMod.^3);

%== Extract parameters ==%
ex_lambda = M1;
ccdf_ex = expcdf(X,ex_lambda,'upper');

[gp_k, gp_sigma, gp_theta] = gpSolve(M1, M2, M3);
ccdf_gp = gpcdf(X,gp_k, gp_sigma, gp_theta,'upper');

%== Extract GoF data ==%
z_ex = expcdf(test_data, ex_lambda);
z_gp = gpcdf(test_data, gp_k, gp_sigma, gp_theta);
zp_ex = exppdf(test_data, ex_lambda);
zp_gp = gppdf(test_data, gp_k, gp_sigma, gp_theta);
stats_ex = testStatistics(test_data, z_ex,zp_ex,0);
stats_gp = testStatistics(test_data, z_gp, zp_gp,0);

%== Plotting ==%
fig = figure();
hold on
plot(X,ccdf,'o');
plot(X,ccdf_ex);
plot(X,ccdf_gp);
set(gca,'XScale','log');
set(gca,'YScale','log');
xlabel(graph_title);
ylabel('CCDF');
axis([-inf,inf,Ymin,1E0]);
legend('Data','Exponential','Gen. Pareto','Location','southwest');
imagefilename = [dir_ref,'/','property_title','_Moments.png'];
print(imagefilename,'-dpng')
figfilename = [dir_ref,'/','property_title','_Moments'];
savefig(figfilename);
close(fig);

%==Build data structure==%
samplesize = max(size(data));
struc_ex = struct('Scale',ex_lambda);
struc_gp = struct('Shape',gp_k,'Scale',gp_sigma,'Location',gp_theta);
p_ex = pvals_ex(samplesize,ex_lambda,stats_ex,cutExtreme,MC_Power,res);
p_gp = pvals_gp(samplesize,gp_k,gp_sigma,gp_theta,stats_gp,cutExtreme,MC_Power,res);
EX = struct('Parameters',struc_ex,'Statistics',stats_ex,'pValues',p_ex);
GP = struct('Parameters',struc_gp,'Statistics',stats_gp,'pValues',p_gp);
Structure = struct('Exponential',EX,'GenPareto',GP);
end

B.21 calculateDistance.m

function dist = calculateDistance(genFolder,realFolder)
%CALCULATEDISTANCE takes two folders of data and compares them to each
% other by calculating all the 2 sample KS-test distances between them
%
% GENFOLDER is the folder path of the first data collection, given as a
% string
% REALFOLDER is the folder path of the second data collection, given as a
% string
DIST is a structure containing all distance and probability information, the mean, mode, min and max for each, and the number of acceptances of the null hypothesis of the 2-sample Kolomogorov-Smirnov test at the 5 percent level.

GiF = ['input/','genFolder'];
GtoExtract = [GiF,'/*.csv'];

RiF = ['input/','realFolder'];
RtoExtract = [RiF,'/*.csv'];

GfileData = dir(GtoExtract);
GfileList = {GfileData.name};
GfileList = GfileList(˜ contains(GfileList,'_.'));

RfileData = dir(RtoExtract);
RfileList = {RfileData.name};
RfileList = RfileList(˜ contains(RfileList,'_.'));

for i=1:length(GfileList)
    currentFile = GfileList{i};
    currentClean = strrep(currentFile, '.', '');
    currentClean = strrep(currentClean, '-', '');
    currentData = pullData(GiF,currentFile,'%f %f %f %s
%s');
    GActiveLinks.(currentClean) = currentData.ActiveLinks_data;
    GInteractionTimes.(currentClean) = currentData.InteractionTimes_data;
    GActivityPotential.(currentClean) = currentData.ActivityPotential_data;
    GNoContactTimes.(currentClean) = currentData.NoContactTimes_data;
    GNodesActive.(currentClean) = currentData.NodesActive_data;
    GComponents.(currentClean) = currentData.Components_data;
    GClustering.(currentClean) = currentData.Clustering_data;
    GComponentNodes.(currentClean) = currentData.ComponentNodes_data;
    GComponentEdges.(currentClean) = currentData.ComponentEdges_data;
GTriangles.(currentClean) = currentData.
  Triangles_data;
end

for i=1:length(RfileList)
  currentFile = RfileList{i};
  currentClean = strrep(currentFile, ' ', '');
  currentClean = strrep(currentClean, '-', '');
  currentData = pullData(RiF, currentFile, '%f %f %f %s %s');
  % Store Data
  RActiveLinks.(currentClean) = currentData.
    ActiveLinks_data;
  RInteractionTimes.(currentClean) = currentData.
    InteractionTimes_data;
  RActivityPotential.(currentClean) = currentData.
    ActivityPotential_data;
  RNoContactTimes.(currentClean) = currentData.
    NoContactTimes_data;
  RNodesActive.(currentClean) = currentData.
    NodesActive_data;
  RComponents.(currentClean) = currentData.
    Components_data;
  RClustering.(currentClean) = currentData.
    Clustering_data;
  RComponentNodes.(currentClean) = currentData.
    ComponentNodes_data;
  RComponentEdges.(currentClean) = currentData.
    ComponentEdges_data;
  RTriangles.(currentClean) = currentData.
    Triangles_data;
end

ActiveLinks_struc = distanceStruc_c(GActiveLinks, RActiveLinks);
OnTimes_struc = distanceStruc_c(GInteractionTimes, RInteractionTimes);
ActivityPot_struc = distanceStruc_c(GActivityPotential, RActivityPotential);
OffTimes_struc = distanceStruc_c(GNoContactTimes, RNoContactTimes);
ActiveNodes_struc = distanceStruc_c(GNodesActive, RNodesActive);
CompCount_struc = distanceStruc_c(GComponents, RComponents);
GCC_struc = distanceStruc_c(GClustering, RClustering);
CompNodes_struc = distanceStruc_c(GComponentNodes, RComponentNodes);
CompEdges_{struc} = distanceStruc_c(GComponentEdges, RComponentEdges);
TriangleCount_{struc} = distanceStruc_c(GTriangles, RTriangles);

dist.ActiveLinks = ActiveLinks_{struc};
dist.OnTimes = OnTimes_{struc};
dist.ActivityPot = ActivityPot_{struc};
dist.OffTimes = OffTimes_{struc};
dist.ActiveNodes = ActiveNodes_{struc};
dist.CompCount = CompCount_{struc};
gcc = GCC_{struc};
dist.CompNodes = CompNodes_{struc};
dist.CompEdges = CompEdges_{struc};
dist.TriangleCount = TriangleCount_{struc};

B.22 chooselink.m

function [startnode,endnode] = chooselink(M)
% CHOOSELINK selects link at random using weighting matrix
% M
% M specifies the edge preference matrix. The can be
% extracted from the
% data using the function EAPmat.m or generated using MC
% simulation using
% the function rndLPM.m
% STARTNODE is the first endpoint of the chosen link
% ENDNODE is the second endpoint of the chosen link

nodes = size(M,1); % Extract number of nodes
M = M/sum(sum(M)); % Normalise M so that total sum is 1 (if not already)
vecM = reshape(M',1,numel(M)); % Reshape into a row vector
cumM = cumsum(vecM);

rn = rand(1); % Choose a random number from UNIF(0,1)
idx = find(cumM>=rn,1); % Find point where cumsum of row vector first exceeds rn
modrem = mod(idx,nodes);
% Extracts nodes from this chosen entry
if modrem == 0
    startnode = idx/nodes;
    endnode = nodes;
else
    startnode = ((idx-modrem)/nodes)+1;
    endnode = modrem;
end

% Rearrange to ensure output is in correct half of matrix
if startnode>endnode
    s = startnode;
    e = endnode;
    startnode = e;
    endnode = s;
end
end

B.23  compareData.m

function [] = compareData(genFolder,realFolder)
% COMPAREDATA takes two folders of data and compares them
to each other,
% calculating statistical distances as well as plotting
the data sets
% invidually and combined (with error bars)
% GENFOLDER is the folder path of the first data collection
% REALFOLDER is the folder path of the second data collection

timestamp = datestr(now,'yyyyMMddTHHMMSS');

GiF = ['input/','genFolder];
GtoExtract = [GiF,'/*.csv'];

RiF = ['input/','realFolder];
RtoExtract = [RiF,'/*.csv'];

dir_ref = ['output_',timestamp];
mkdir(dir_ref);

GfileData = dir(GtoExtract);
GfileList = {GfileData.name};
GfileList = GfileList(contains(GfileList,'._'));

RfileData = dir(RtoExtract);
RfileList = {RfileData.name};
RfileList = RfileList(contains(RfileList,'._'));
for i = 1:length(GfileList)
    currentFile = GfileList{i};
    currentClean = strrep(currentFile, '\', '');
    currentClean = strrep(currentClean, '-', '');
    currentData = pullData(GiF, currentFile, '%f %f %f %*s %*s');
    % Store Data
    GActiveLinks.(currentClean) = currentData.ActiveLinks_data;
    GInteractionTimes.(currentClean) = currentData.InteractionTimes_data;
    GActivityPotential.(currentClean) = currentData.ActivityPotential_data;
    GNoContactTimes.(currentClean) = currentData.NoContactTimes_data;
    GNodesActive.(currentClean) = currentData.NodesActive_data;
    GComponents.(currentClean) = currentData.Components_data;
    GClustering.(currentClean) = currentData.Clustering_data;
    GComponentNodes.(currentClean) = currentData.ComponentNodes_data;
    GComponentEdges.(currentClean) = currentData.ComponentEdges_data;
    GTriangles.(currentClean) = currentData.Triangles_data;
end

for i = 1:length(RfileList)
    currentFile = RfileList{i};
    currentClean = strrep(currentFile, '\', '');
    currentClean = strrep(currentClean, '-', '');
    currentData = pullData(RiF, currentFile, '%f %f %f %*s %*s');
    % Store Data
    RActiveLinks.(currentClean) = currentData.ActiveLinks_data;
    RInteractionTimes.(currentClean) = currentData.InteractionTimes_data;
    RActivityPotential.(currentClean) = currentData.ActivityPotential_data;
    RNoContactTimes.(currentClean) = currentData.NoContactTimes_data;
    RNodesActive.(currentClean) = currentData.NodesActive_data;
    RComponents.(currentClean) = currentData.Components_data;
function [] = compareOriginal(folder)

%%COMPAREORIGINAL produces a latex file containing
Kolmogorov-Smirnov
%% acceptances for the hypothesis that pairs of data
samples in the given
%% folder come from the same underlying distribution
across chosen metrics
%%
%% FOLDER is the folder containing the files to compare,
given as a string

timestamp = datestr(now,'yyyyymmddTHHMMSS');
dir_ref = ['output_',timestamp];
mkdir(dir_ref);
filename = 'comparesamples.txt';
filepath = [dir_ref,'/','filename];
iF = ['input/','folder'];
toExtract = [iF,'/*.csv'];

fileData = dir(toExtract);
fileList = {fileData.name};
fileList = fileList('contains(fileList,'._'));

for i=1:length(fileList)
    currentFile = fileList{i};
    currentClean = strrep(currentFile, '.', ' ');
    currentClean = strrep(currentClean, '-', ' ');
    currentData = pullData(iF,currentFile,'%f %f %f %*s
    %*s');
    fActiveLinks.(currentClean) = currentData. ActiveLinks_data;
    fInteractionTimes.(currentClean) = currentData. InteractionTimes_data;
    fActivityPotential.(currentClean) = currentData. ActivityPotential_data;
    fNoContactTimes.(currentClean) = currentData. NoContactTimes_data;
    fNodesActive.(currentClean) = currentData. NodesActive_data;
    fComponents.(currentClean) = currentData. Components_data;
    fClustering.(currentClean) = currentData. Clustering_data;
    fComponentNodes.(currentClean) = currentData. ComponentNodes_data;
    fComponentEdges.(currentClean) = currentData. ComponentEdges_data;
    fTriangles.(currentClean) = currentData. Triangles_data;
end

fields = fieldnames(fActiveLinks);
numberComparisons = nchoosek(length(fields),2);
acceptmatrix1 = zeros(numberComparisons,10);
acceptmatrix5 = zeros(numberComparisons,10);
acceptmatrix10 = zeros(numberComparisons,10);
n = 0;
for i = 1:length(fields) - 1
    for j = i + 1:length(fields)
        n = n + 1;
        f1 = fields{i};
        f2 = fields{j};
        [acceptmatrix1(n,1), acceptmatrix5(n,1),
         acceptmatrix10(n,1)] = acceptances(fActiveLinks.(f1), fActiveLinks.(f2));
        [acceptmatrix1(n,2), acceptmatrix5(n,2),
         acceptmatrix10(n,2)] = acceptances(fInteractionTimes.(f1), fInteractionTimes.(f2));
        [acceptmatrix1(n,3), acceptmatrix5(n,3),
         acceptmatrix10(n,3)] = acceptances(fActivityPotential.(f1), fActivityPotential.(f2));
        [acceptmatrix1(n,4), acceptmatrix5(n,4),
         acceptmatrix10(n,4)] = acceptances(fNoContactTimes.(f1), fNoContactTimes.(f2));
        [acceptmatrix1(n,5), acceptmatrix5(n,5),
         acceptmatrix10(n,5)] = acceptances(fNodesActive.(f1), fNodesActive.(f2));
        [acceptmatrix1(n,6), acceptmatrix5(n,6),
         acceptmatrix10(n,6)] = acceptances(fComponents.(f1), fComponents.(f2));
        [acceptmatrix1(n,7), acceptmatrix5(n,7),
         acceptmatrix10(n,7)] = acceptances(fClustering.(f1), fClustering.(f2));
        [acceptmatrix1(n,8), acceptmatrix5(n,8),
         acceptmatrix10(n,8)] = acceptances(fComponentNodes.(f1), fComponentNodes.(f2));
        [acceptmatrix1(n,9), acceptmatrix5(n,9),
         acceptmatrix10(n,9)] = acceptances(fComponentEdges.(f1), fComponentEdges.(f2));
        [acceptmatrix1(n,10), acceptmatrix5(n,10),
         acceptmatrix10(n,10)] = acceptances(fTriangles.(f1), fTriangles.(f2));
    end
end

acceptsum1 = sum(acceptmatrix1);
acceptsum5 = sum(acceptmatrix5);
acceptsum10 = sum(acceptmatrix10);
ActiveLinks.accept1 = acceptsum1(1);
ActiveNodes.accept1 = acceptsum1(2);
ActivityPot.accept1 = acceptsum1(3);
GCC.accept1 = acceptsum1(4);
OnTimes.accept1 = acceptsum1(5);
OffTimes.accept1 = acceptsum1(6);
CompCount.accept1 = acceptsum1(7);
CompEdges.accept1 = acceptsum1(8);
CompNodes.accept1 = acceptsum1(9);
TriangleCount.accept1 = acceptsum1(10);

ActiveLinks.accept5 = acceptsum5(1);
ActiveNodes.accept5 = acceptsum5(2);
ActivityPot.accept5 = acceptsum5(3);
GCC.accept5 = acceptsum5(4);
OnTimes.accept5 = acceptsum5(5);
OffTimes.accept5 = acceptsum5(6);
CompCount.accept5 = acceptsum5(7);
CompEdges.accept5 = acceptsum5(8);
CompNodes.accept5 = acceptsum5(9);
TriangleCount.accept5 = acceptsum5(10);

ActiveLinks.accept10 = acceptsum10(1);
ActiveNodes.accept10 = acceptsum10(2);
ActivityPot.accept10 = acceptsum10(3);
GCC.accept10 = acceptsum10(4);
OnTimes.accept10 = acceptsum10(5);
OffTimes.accept10 = acceptsum10(6);
CompCount.accept10 = acceptsum10(7);
CompEdges.accept10 = acceptsum10(8);
CompNodes.accept10 = acceptsum10(9);
TriangleCount.accept10 = acceptsum10(10);

lines = 13;
tobuild = cell(lines,1);
tobuild{01} = '\begin{tabular}{l|c|c|c|} \cline{2-4}'
tobuild{02} = '& \textbf{1\%} & \textbf{5\%} & \textbf{10\%}\\ \hline';
tobuild{03} = ['\multicolumn{1}{|l|}{Active Links} & $',num2str(ActiveLinks.accept1),'$ & $',num2str(ActiveLinks.accept5),'$ & $',num2str(ActiveLinks.accept10),'$\\ \hline'];
tobuild{04} = ['\multicolumn{1}{|l|}{Active Nodes} & $',num2str(ActiveNodes.accept1),'$ & $',num2str(ActiveNodes.accept5),'$ & $',num2str(ActiveNodes.accept10),'$\\ \hline'];
tobuild{05} = ['\multicolumn{1}{|l|}{Node Activity Potential} & $',num2str(ActivityPot.accept1),'$ & $',num2str(ActivityPot.accept5),'$ & $',num2str(ActivityPot.accept10),'$\\ \hline'];
tobuild{06} = ['\mc{1}{|l|}{\bf{Global Clustering Coefficient}} & $',num2str(GCC.accept1),'$ & $',num2str(GCC.accept5),'$ & $',num2str(GCC.accept10),'$\\
\hline'];

```matlab
tobuild{07} = ['\mc{1}{|l|}{\bf{Interaction Time}} & $',num2str(OnTimes.accept1),'$ & $',num2str(OnTimes.accept5),'$ & $',num2str(OnTimes.accept10),'$\\
\hline'];
```

```matlab
tobuild{08} = ['\mc{1}{|l|}{\bf{Time Between Contacts}} & $',num2str(OffTimes.accept1),'$ & $',num2str(OffTimes.accept5),'$ & $',num2str(OffTimes.accept10),'$\\
\hline'];
```

```matlab
tobuild{09} = ['\mc{1}{|l|}{\bf{Component Count}} & $',num2str(CompCount.accept1),'$ & $',num2str(CompCount.accept5),'$ & $',num2str(CompCount.accept10),'$\\
\hline'];
```

```matlab
tobuild{10} = ['\mc{1}{|l|}{\bf{Links per Component}} & $',num2str(CompEdges.accept1),'$ & $',num2str(CompEdges.accept5),'$ & $',num2str(CompEdges.accept10),'$\\
\hline'];
```

```matlab
tobuild{11} = ['\mc{1}{|l|}{\bf{Nodes per Component}} & $',num2str(CompNodes.accept1),'$ & $',num2str(CompNodes.accept5),'$ & $',num2str(CompNodes.accept10),'$\\
\hline'];
```

```matlab
tobuild{12} = ['\mc{1}{|l|}{\bf{Triangle Count}} & $',num2str(TriangleCount.accept1),'$ & $',num2str(TriangleCount.accept5),'$ & $',num2str(TriangleCount.accept10),'$\\
\hline'];
```

```matlab
tobuild{13} = '\end{tabular}'
```

```matlab
fileID = fopen(filepath,'w');
fprintf(fileID,'%s\r\n',tobuild{:});
fclose(fileID);
```

end

**B.25 comparison.m**

```matlab
function [] = comparison(folder,count,timelength)

%COMPARISON produces simulated data for each model and then produces a
% latex text file containing the number of acceptances across a variety of
% metrics for the null hypothesis to the 2-sample Kolmogorov-Smirnov test
% when compared simulated and observed data samples
```
% FOLDER is the filepath of the parent folder containing the observed data
% COUNT is the number of samples to generate
% TIMELENGTH is the desired simulation time

timestamp = datestr(now,'yyyyymmdTHHMMSS');
dir_ref = ['./input/output_',timestamp];
mkdir(dir_ref);
iF = ['./input/folder'];
toExtract = [iF,'/*.csv'];
fileData = dir(toExtract);
fileList = {fileData.name};
fileList = fileList(~contains(fileList,'._'));
accept5matrix = zeros(10,4);

for i=1:length(fileList)
    currentFile = fileList{i};
    currentFilepath = [iF,'/currentFile];
    currentClean = strrep(currentFile, '.', '');
    currentClean = strrep(currentClean, '-', '');
    currentParent = [dir_ref,'/comparison'];
    currentFolder = [currentParent,'/',currentClean];
    currentFolderR = [currentFolder,'/original'];
    currentFolderM1 = [currentFolder,'/model1'];
    currentFolderM2a = [currentFolder,'/model2a'];
    currentFolderM2b = [currentFolder,'/model2b'];
    currentFolderM2c = [currentFolder,'/model2c'];
    mkdir(currentFolderR);
    mkdir(currentFolderM1);
    mkdir(currentFolderM2a);
    mkdir(currentFolderM2b);
    mkdir(currentFolderM2c);
    copyfile(currentFilepath,currentFolderR);
    currentData = pullData(iF,currentFile,'%f %f %f %s
%s');
    currentStudents = currentData.NumberStudents_data;
    currentMatrix = EAP_matrix(iF(7:end),currentFile);
    for j=1:count
        currentRandom = rndLPM(currentStudents);
        outputfile = ['./run_'.num2str(j),'.csv'];
        fp1 = [currentFolderM1,'/outputfile];
        fp2a = [currentFolderM2a,'/outputfile];
        fp2b = [currentFolderM2b,'/outputfile];
        fp2c = [currentFolderM2c,'/outputfile];
        model(currentStudents,timelength,fp1);
modelv2a_1(timelength,currentMatrix,fp2a);
modelv2_1(timelength,currentMatrix,fp2b);
modelv2_1(timelength,currentRandom,fp2c);
end

dist1 = calculateDistance(currentFolderM1(7:end),
                        currentFolderR(7:end));
dist2a = calculateDistance(currentFolderM2a(7:end),
                        currentFolderR(7:end));
dist2b = calculateDistance(currentFolderM2b(7:end),
                        currentFolderR(7:end));
dist2c = calculateDistance(currentFolderM2c(7:end),
                        currentFolderR(7:end));
currentAccept5 = zeros(10,4);
currentAccept5(1,1) = dist1.ActiveLinks.accept5;
currentAccept5(2,1) = dist1.OnTimes.accept5;
currentAccept5(3,1) = dist1.ActivityPot.accept5;
currentAccept5(4,1) = dist1.OffTimes.accept5;
currentAccept5(5,1) = dist1.ActiveNodes.accept5;
currentAccept5(6,1) = dist1.CompCount.accept5;
currentAccept5(7,1) = dist1.GCC.accept5;
currentAccept5(8,1) = dist1.CompNodes.accept5;
currentAccept5(9,1) = dist1.CompEdges.accept5;
currentAccept5(10,1) = dist1.TriangleCount.accept5;
currentAccept5(1,2) = dist2a.ActiveLinks.accept5;
currentAccept5(2,2) = dist2a.OnTimes.accept5;
currentAccept5(3,2) = dist2a.ActivityPot.accept5;
currentAccept5(4,2) = dist2a.OffTimes.accept5;
currentAccept5(5,2) = dist2a.ActiveNodes.accept5;
currentAccept5(6,2) = dist2a.CompCount.accept5;
currentAccept5(7,2) = dist2a.GCC.accept5;
currentAccept5(8,2) = dist2a.CompNodes.accept5;
currentAccept5(9,2) = dist2a.CompEdges.accept5;
currentAccept5(10,2) = dist2a.TriangleCount.accept5;
currentAccept5(1,3) = dist2b.ActiveLinks.accept5;
currentAccept5(2,3) = dist2b.OnTimes.accept5;
currentAccept5(3,3) = dist2b.ActivityPot.accept5;
currentAccept5(4,3) = dist2b.OffTimes.accept5;
currentAccept5(5,3) = dist2b.ActiveNodes.accept5;
currentAccept5(6,3) = dist2b.CompCount.accept5;
currentAccept5(7,3) = dist2b.GCC.accept5;
currentAccept5(8,3) = dist2b.CompNodes.accept5;
currentAccept5(9,3) = dist2b.CompEdges.accept5;
currentAccept5(10,3) = dist2b.TriangleCount.accept5;
currentAccept5(1,4) = dist2c.ActiveLinks.accept5;
currentAccept5(2,4) = dist2c.OnTimes.accept5;
currentAccept5(3,4) = dist2c.ActivityPot.accept5;
currentAccept5(4,4) = dist2c.OffTimes.accept5;
currentAccept5(5,4) = dist2c.ActiveNodes.accept5;
currentAccept5(6,4) = dist2c.CompCount.accept5;
currentAccept5(7,4) = dist2c.GCC.accept5;
currentAccept5(8,4) = dist2c.CompNodes.accept5;
currentAccept5(9,4) = dist2c.CompEdges.accept5;
currentAccept5(10,4) = dist2c.TriangleCount.accept5;
accept5matrix = accept5matrix + currentAccept5;
end

lines = 13;
tobuild = cell(lines,1);

tobuild{01} = '\begin{tabular}{|c|c|c|c|c|} \cline{2-5}';
tobuild{02} = '& $\textbf{Model 1}$ & $\textbf{Model 2a}$ & $\textbf{Model 2b}$ & $\textbf{Model 2c}$ \\
\hline';
tobuild{03} = ['\multicolumn{1}{|c|}{$\textbf{Active Links}$} & $\textbf{Active Links}$ & num2str(accept5matrix(1,1)),'$ & $',num2str(accept5matrix(1,2)),'$ & $',num2str(accept5matrix(1,3)),'$ & $',num2str(accept5matrix(1,4)),'$\ \hline'];
tobuild{04} = ['\multicolumn{1}{|c|}{$\textbf{Active Nodes}$} & $\textbf{Active Nodes}$ & num2str(accept5matrix(2,1)),'$ & $',num2str(accept5matrix(2,2)),'$ & $',num2str(accept5matrix(2,3)),'$ & $',num2str(accept5matrix(2,4)),'$\ \hline'];
tobuild{05} = ['\multicolumn{1}{|c|}{$\textbf{Node Activity Potential}$} & $\textbf{Node Activity Potential}$ & num2str(accept5matrix(3,1)),'$ & $',num2str(accept5matrix(3,2)),'$ & $',num2str(accept5matrix(3,3)),'$ & $',num2str(accept5matrix(3,4)),'$\ \hline'];
tobuild{06} = ['\multicolumn{1}{|c|}{$\textbf{Global Clustering Coefficient}$} & $\textbf{Global Clustering Coefficient}$ & num2str(accept5matrix(4,1)),'$ & $',num2str(accept5matrix(4,2)),'$ & $',num2str(accept5matrix(4,3)),'$ & $',num2str(accept5matrix(4,4)),'$\ \hline'];
tobuild{07} = ['\multicolumn{1}{|c|}{$\textbf{Interaction Time}$} & $\textbf{Interaction Time}$ & num2str(accept5matrix(5,1)),'$ & $',num2str(accept5matrix(5,2)),'$ & $',num2str(accept5matrix(5,3)),'$ & $',num2str(accept5matrix(5,4)),'$\ \hline'];
tobuild{08} = ['\multicolumn{1}{|c|}{$\textbf{Time Between Contacts}$} & $\textbf{Time Between Contacts}$ & num2str(accept5matrix(6,1)),'$ & $',num2str(accept5matrix(6,2)),'$ & $',num2str(accept5matrix(6,3)),'$ & $',num2str(accept5matrix(6,4)),'$\ \hline'];
tobuild{09} = ['\multicolumn{1}{|c|}{$\textbf{Component Count}$} & $\textbf{Component Count}$ & num2str(accept5matrix(7,1)),'$ & $',num2str(accept5matrix(7,2)),'$ & $',num2str(accept5matrix(7,3)),'$ & $',num2str(accept5matrix(7,4)),'$\ \hline'];
tobuild{10} = ['multicolumn{1}{|l|}{\textbf{Links per Component}} & $',num2str(accept5matrix(8,1)),'$ & $',num2str(accept5matrix(8,2)),'$ & $',num2str(accept5matrix(8,3)),'$ & $',num2str(accept5matrix(8,4)),$\\ \hline'];
tobuild{11} = ['multicolumn{1}{|l|}{\textbf{Nodes per Component}} & $',num2str(accept5matrix(9,1)),'$ & $',num2str(accept5matrix(9,2)),'$ & $',num2str(accept5matrix(9,3)),'$ & $',num2str(accept5matrix(9,4)),$\\ \hline'];
tobuild{12} = ['multicolumn{1}{|l|}{\textbf{Triangle Count}} & $',num2str(accept5matrix(10,1)),'$ & $',num2str(accept5matrix(10,2)),'$ & $',num2str(accept5matrix(10,3)),'$ & $',num2str(accept5matrix(10,3)),$\\ \hline'];
tobuild{13} = '\end{tabular}';

filepath = [dir_ref,'/comparisons.txt'];

fileID = fopen(filepath,'w');
fprintf(fileID,'%s\r\n',tobuild{:});
fclose(fileID);

end

B.26 create.avi.m

function [] = create.avi(realdata,gendata,dir_ref)
% CREATE_AVI creates two comparative avi animation files showing the network
% progress over time of two data sets
% %
% % REALDATA is the first set of data
% % GENDATA is the second set of data
% % DIR_REF is the save directory

mapfilename = [dir_ref,'/create.avi-map-vid.avi'];
linksfilename = [dir_ref,'/create.avi-links-vid.avi'];

contact_time = 20;
close all
scale = 20;

all_people1 = [realdata(:,2) realdata(:,3)];
num_times1 = size(unique(realdata(:,1)),1);
data_length1 = size(realdata(:,1),1);
num_people1 = max(all_people1);
coords1 = zeros(num_people1,2);

\[ \theta_1 = \frac{2 \pi}{\text{num\_people1}}; \]
\[ \text{all\_people2} = [\text{gendata}(:,2)'; \text{gendata}(:,3)']; \]
\[ \text{num\_times2} = \text{size(\text{unique(gendata(:,1))},1)}; \]
\[ \text{data\_length2} = \text{size(gendata(:,1),1)}; \]
\[ \text{num\_people2} = \text{max(all\_people2)}; \]
\[ \text{coords2} = \text{zeros(num\_people2,2)}; \]
\[ \theta_2 = \frac{2 \pi}{\text{num\_people2}}; \]
\[ \text{num\_times} = \text{min(num\_times1, num\_times2)}; \]
\[
\begin{align*}
\text{parfor } n &= 1: \text{num\_people1} \\
& \quad \text{coords1}(n,:) = \text{scale}*[\sin(n*\theta_1) \ \cos(n*\theta_1)];
\end{align*}
\]
\[
\begin{align*}
\text{parfor } n &= 1: \text{num\_people2} \\
& \quad \text{coords2}(n,:) = \text{scale}*[\sin(n*\theta_2) \ \cos(n*\theta_2)];
\end{align*}
\]
\[ \text{step\_nf1} = \text{zeros(num\_people1,1)}; \]
\[ \text{line\_freq1} = \text{zeros(num\_people1,num\_people1)}; \]
\[ \text{step\_nf2} = \text{zeros(num\_people2,1)}; \]
\[ \text{line\_freq2} = \text{zeros(num\_people2,num\_people2)}; \]
\[ \text{links}(\text{num\_times}) = \text{struct}('\text{cdata}',[],'\text{colormap}',[]); \]
\[ \text{map}(\text{num\_times}) = \text{struct}('\text{cdata}',[],'\text{colormap}',[]); \]
\[
\begin{align*}
\text{for } m &= 1: \text{num\_times} \\
& \quad \text{thisadj1} = \text{zeros(num\_people1)}; \\
& \quad \text{thisadj2} = \text{zeros(num\_people2)}; \\
& \quad \text{current\_time} = (m-1)*\text{contact\_time}; \\
& \quad \text{for } i &= 1: \text{data\_length1} \\
& \quad & \quad \text{test\_time} = \text{realdata}(i,1); \\
& \quad & \quad \text{if } \text{test\_time}==\text{current\_time} \\
& \quad & \quad & \quad \text{person1} = \text{realdata}(i,2); \\
& \quad & \quad & \quad \text{person2} = \text{realdata}(i,3); \\
& \quad & \quad & \quad \text{thisadj1(person1,person2)} = 1; \\
& \quad & \quad & \quad \text{thisadj1(person2,person1)} = 1;
\end{align*}
\]
\[
\begin{align*}
\text{for } i &= 1: \text{data\_length2} \\
& \quad \text{test\_time} = \text{gendata}(i,1); \\
& \quad \text{if } \text{test\_time}==\text{current\_time} \\
& \quad & \quad \text{person1} = \text{gendata}(i,2); \\
& \quad & \quad \text{person2} = \text{gendata}(i,3); \\
& \quad & \quad \text{thisadj2(person1,person2)} = 1; \\
& \quad & \quad \text{thisadj2(person2,person1)} = 1;
\end{align*}
\]
\begin{verbatim}
end
step_nf1 = step_nf1+sum(thisadj1,2);
step_nf2 = step_nf2+sum(thisadj2,2);

%==Create Adj Frame==%
map_fig = figure();
subplot(1,2,1)
gplot(thisadj1,coords1,'-*');
str = sprintf('Time: %d', current_time);
text(0,-1.2*scale,str);
axis([-1.5 1.5 -1.5 1.5]*scale);
axis off;
set(gcf,'color','w');
subplot(1,2,2)
gplot(thisadj2,coords2,'-*');
str = sprintf('Time: %d', current_time);
text(0,-1.2*scale,str);
axis([-1.5 1.5 -1.5 1.5]*scale);
axis off;
set(gcf,'color','w');
map(m) = getframe(map_fig);
close(map_fig);

%==End Frame Creation==%
this_rel_node_freq1 = step_nf1/max(step_nf1);
this_rel_node_freq2 = step_nf2/max(step_nf2);
line_freq1 = line_freq1+thisadj1;
line_freq2 = line_freq2+thisadj2;
thisRLF1 = line_freq1/(max(max(line_freq1)));
thisRLF2 = line_freq2/(max(max(line_freq2)));

%==Create Activity Frame==%
links_fig = figure();
subplot(1,2,1)
link_size1 = this_rel_node_freq1*50;
tempadj1 = logical(thisRLF1);
[row1,col1] = find(tempadj1);
tempcoords1 = zeros(num_people1,2);
hold on
for i=1:length(row1)
    thisrow = row1(i);
    thiscol = col1(i);
    if thisrow >= thiscol
        line_col = (1-thisRLF1(thisrow,thiscol))*[1 1 1];
        x1 = coords1(thisrow,1);
        y1 = coords1(thisrow,2);
        x2 = coords1(thiscol,1);
\end{verbatim}
y2 = coords1(thiscol,2);
    line([x1 x2],[y1 y2],'Color',line_col)
end
    tempcoords1(i,:) = coords1(thisrow,:);
end
tempcoords1 = tempcoords1(any(tempcoords1,2),:);
tempcoords1 = unique(tempcoords1,'rows','stable');
link_size1(link_size1==0) = [];
scatter(tempcoords1(:,1),tempcoords1(:,2),link_size1,
    'filled')
hold off
str = sprintf('Time: %d', current_time);
    text(0,-1.2*scale,str);
    axis([-1.5 1.5 -1.5 1.5]*scale);
    axis off;
    set(gcf,'color','w');
    subplot(1,2,2)
link_size2 = this_rel_node_freq2*50;
tempadj2 = logical(thisRLF2);
    [row2,col2] = find(tempadj2);
tempcoords2 = zeros(num_people2,2);
    hold on
for i=1:length(row2)
    thisrow = row2(i);
    thiscol = col2(i);
    if thisrow >= thiscol
        line_col = (1-thisRLF2(thisrow,thiscol))*[1 1 1];
        x1 = coords2(thisrow,1);
        y1 = coords2(thisrow,2);
        x2 = coords2(thiscol,1);
        y2 = coords2(thiscol,2);
        line([x1 x2],[y1 y2],'Color',line_col)
    end
    tempcoords2(i,:) = coords2(thisrow,:);
end
tempcoords2 = tempcoords2(any(tempcoords2,2),:);
tempcoords2 = unique(tempcoords2,'rows','stable');
    link_size2(link_size2==0) = [];
    scatter(tempcoords2(:,1),tempcoords2(:,2),link_size2,
        'filled')
hold off
    str = sprintf('Time: %d', current_time);
    text(0,-1.2*scale,str);
    axis([-1.5 1.5 -1.5 1.5]*scale);
    axis off;
    set(gcf,'color','w');
    links(m) = getframe(links_fig);
close ( links_fig );
end

v = VideoWriter ( mapfilename );
open ( v )
writeVideo ( v, map )
close ( v )

v = VideoWriter ( linksfilename );
open ( v )
writeVideo ( v, links )
close ( v )
end

B.27  cumx2xxyy.m

This function was written by Whiten [211].

function yy=cumx2xxyy ( x, xx )
% cumx2xxyy Convert cumulative plot of x, to (xx,yy) points for given xx
% 2014-09-22 Matlab2014 W. Whiten
% %
% % yy=cumx2xxyy (x,xx)
% x Sorted values of cumulative distribution
% xx X values to read yy values from cumulative
distribution
% %
% % yy Y values corresponding to the xx values
% %
% % Given xx(i) values of (xx(i),yy(i)) are read from cumulative graph
% % of (x(ix),(ix-1)/(nx-1)) using linear interpolation
% %
% % Copyright (C) 2014, W. Whiten (personal W.Whiten@uq.edu.au) BSD license
% (http://opensource.org/licenses/BSD-3-Clause)

nx=length ( x );
nxx=length ( xx );
yy=zeros ( size ( xx ));

ix=1;
for i=1:nxx
    while ( ix<= nx & xx ( i ) > x ( ix ) )
        ix=ix+1;
    end
if (ix == 1)
    yy(i) = 1;
elseif (ix > nx)
    yy(i) = nx;
else
    yy(i) = ix - (x(ix) - xx(i)) / (x(ix) - x(ix - 1));
end
end

yy = (yy - 1) / (nx - 1);
return
end

B.28 dataMinMax.m

function [mins, maxs] = dataMinMax(Analysis)

% MM_EXPMLGPWEI returns the minimum and maximum values of each parameter in
% ANALYSIS is the analytic data structure
% MINS is a structure containing all minimum parameters
% MAXS is a structure containing all maximum parameters

[ActiveLinks_mins, ActiveLinks_maxs] = mm_ExpGamRayLN(Analysis.ActiveLinks_FitTool, Analysis.ActiveLinks_MLE, Analysis.ActiveLinks_Moments);
[InteractionTimes_mins, InteractionTimes_maxs] = mm_ExpMLGPWei(Analysis.InteractionTimes_FitTool, Analysis.InteractionTimes_MLE, Analysis.InteractionTimes_Moments);
[NoContactTimes_mins, NoContactTimes_maxs] = mm_ExpGamRayLN(Analysis.NoContactTimes_FitTool, Analysis.NoContactTimes_MLE, Analysis.NoContactTimes_Moments);
[NodesActive_mins, NodesActive_maxs] = mm_ExpGamRayLN(Analysis.NodesActive_FitTool, Analysis.NodesActive_MLE, Analysis.NodesActive_Moments);
[Components_mins, Components_maxs] = mm_ExpGamRayLN(Analysis.Components_FitTool, Analysis.Components_MLE, Analysis.Components_Moments);
[Clustering_mins, Clustering_maxs] = mm_ExpGamRayLN(Analysis.Clustering_FitTool, Analysis.Clustering_MLE, Analysis.Clustering_Moments);

[ComponentNodes_mins, ComponentNodes_maxs] = mm_ExpGamRayLN(Analysis.ComponentNodes_FitTool, Analysis.ComponentNodes_MLE, Analysis.ComponentNodes_Moments);

[ComponentEdges_mins, ComponentEdges_maxs] = mm_ExpGamRayLN(Analysis.ComponentEdges_FitTool, Analysis.ComponentEdges_MLE, Analysis.ComponentEdges_Moments);

mins = struct('ActiveLinks', ActiveLinks_mins, ...
               'InteractionTimes', InteractionTimes_mins, ...
               'ActivityPotential', ActivityPotential_mins, ...
               'NoContactTimes', NoContactTimes_mins,
               ...'
               'NodesActive', NodesActive_mins, ...
               'Components', Components_mins, ...
               'Clustering', Clustering_mins, ...
               'ComponentNodes', ComponentNodes_mins,
               ...
               'ComponentEdges', ComponentEdges_mins ...
               );

maxs = struct('ActiveLinks', ActiveLinks_maxs, ...
               'InteractionTimes', InteractionTimes_maxs, ...
               'ActivityPotential', ActivityPotential_maxs, ...
               'NoContactTimes', NoContactTimes_maxs,
               ...'
               'NodesActive', NodesActive_maxs, ...
               'Components', Components_maxs, ...
               'Clustering', Clustering_maxs, ...
               'ComponentNodes', ComponentNodes_maxs,
               ...
               'ComponentEdges', ComponentEdges_maxs ...
               );

B.29  deg_in_time.m
function DinT = deg_in_time(data)
% DINT creates a matrix with each row showing the count of
% nodes of each
% degree. Each row represents a different time in the
% simulation.
%
% DATA is the extracted data matrix
% DINT is a matrix showing our count of nodes of each
degree over time

number_times = (max(data(:,1))/20)+1;
number_people = max([data(:,2); data(:,3)]);
contact_time = 20;

DinT = zeros(number_people,number_times);

parfor i=1:number_times
    this_DinT = zeros(number_people,1);
    this_time = (i-1)*contact_time;
    this_idx = (data(:,1) == this_time);
    this_data = data(this_idx,:);
    this_active = [this_data(:,2); this_data(:,3)];
    if isempty(this_active)==0
        for j=1:length(this_active)
            ID = this_active(j);
            this_DinT(ID,1) = this_DinT(ID,1)+1;
        end
    end
    DinT(:,i) = this_DinT;
end

B.30 distanceStruc_c.m

function distanceStruc = distanceStruc_c(gen_data, real_data)
% DISTANCESTRUC_C returns the minimum, maximum, mean and
% mode distances and
% probabilities between two sets of data, as well as a
% count of the number
% of acceptances of the null hypothesis of the two-sample
% Kolmogorov-Smirnov test at the 5 percent level for
% continuous data
%
% GEN_DATA is a structure containing generated data
% REAL_DATA is a structure containing observed data
% DISTANCESTRUC is a structure containing the distances, probabilities and
% acceptances

gen_fields = fieldnames (gen_data);
real_fields = fieldnames (real_data);

genCount = numel (gen_fields);
realCount = numel (real_fields);

distMat = zeros (genCount, realCount);
probMat = zeros (genCount, realCount);
acceptMat = zeros (genCount, realCount);

for i = 1: genCount
    thisGen = gen_data.(gen_fields{i});
    thisGen (isnan (thisGen)) = -1;
    parfor j = 1: realCount
        thisReal = real_data.(real_fields{j});
        thisReal (isnan (thisReal)) = -1;
        [thisH, thisP, thisKS] = kstest2 (thisGen, thisReal);
        distMat (i,j) = thisKS;
        probMat (i,j) = thisP;
        acceptMat (i,j) = 1 - thisH;
    end
end

distanceStruc.minDist = min (min (distMat));
distanceStruc.maxDist = max (max (distMat));
distanceStruc.meanDist = mean (mean (distMat));
distanceStruc.modeDist = mode (distMat (:));
distanceStruc.minProb = min (min (probMat));
distanceStruc.maxProb = max (max (probMat));
distanceStruc.meanProb = mean (mean (probMat));
distanceStruc.modeProb = mean (mean (probMat (:)));
distanceStruc.accept5 = sum (sum (acceptMat));

B.31 distanceStruc_d.m

function distanceStruc = distanceStruc_d (gen_data, real_data)
% DISTANCESTRUC_C returns the minimum, maximum, mean and
% mode distances and
% probabilities between two sets of data, as well as a
% count of the number
% of acceptances of the null hypothesis of the two-sample
% Kolmogorov-Smirnov test at the 5 percent level for discrete data
% GEN_DATA is a structure containing generated data
% REAL_DATA is a structure containing observed data
% DISTANCESTRUC is a structure containing the distances, probabilities and % acceptances

alpha = 0.05;

gen_fields = fieldnames(gen_data);
real_fields = fieldnames(real_data);
genCount = numel(gen_fields);
realCount = numel(real_fields);
distMat = zeros(genCount,realCount);
probMat = zeros(genCount,realCount);
acceptMat = zeros(genCount,realCount);

for i=1:genCount
    thisGen = gen_data.(gen_fields{i});
    thisGen(isnan(thisGen)) = -1;
    parfor j=1:realCount
        thisReal = real_data.(real_fields{j});
        thisReal(isnan(thisReal)) = -1;
        thisBins = unique([thisGen,thisReal]);
        topBin = max(thisBins)+1;
        thisBins = [thisBins,topBin];
        thisGenHist = histogram(thisGen,thisBins);
        thisGenFreq = thisGenHist.Values;
        thisRealHist = histogram(thisReal,thisBins);
        thisRealFreq = thisRealHist.Values;
        thisK = sqrt(sum(thisGenFreq)/sum(thisRealFreq));
        thisSum = ((thisRealFreq*thisK-thisGenFreq/thisK).^2)./(thisRealFreq+thisGenFreq);
        thisDF = (length(thisBins)-1)-logical(length(thisReal)==length(thisGen));
        thisP = 1-chi2cdf(thisChi,thisDF);
        thisH = logical(thisP<1-alpha);
        distMat(i,j) = thisChi;
        probMat(i,j) = thisP;
        acceptMat(i,j) = 1-thisH;
    end
end
distanceStruc.minDist = min(min(distMat));
distanceStruc.maxDist = max(max(distMat));
distanceStruc.meanDist = mean(mean(distMat));
distanceStruc.modeDist = mode(distMat(:));
distanceStruc.minProb = min(min(probMat));
distanceStruc.maxProb = max(max(probMat));
distanceStruc.meanProb = mean(mean(distMat));
distanceStruc.modeProb = mode(distMat(:));
distanceStruc.accept5 = sum(sum(acceptMat));

B.32 dualDDVid.m

function [] = dualDDVid(DinT_real,DinT_gen,dir_ref)
% DUALDDVID creates comparative degree distributions for
% two given data sets

% DINT_REAL is the file for the first data set
% DINT_GEN is the file for the second data set
% DIR_REF is the save folder location

filename = [dir_ref,'/DegreeDistribution.avi'];
contact_time = 20;
um_times1 = size(DinT_real,2);
num_times2 = size(DinT_gen,2);
num_times = min(num_times1,num_times2);
frame(num_times) = struct('cdata',[],'colormap',[]);

for m=1:num_times
    current_time = (m-1)*contact_time;
    thisframe = figure();
    subplot(1,2,1)
    ecdf(DinT_real(:,m));
    str = sprintf('Time: %d', current_time);
    text(4,0.1,str);
    axis([0 5 0 1]);
    subplot(1,2,2)
    ecdf(DinT_gen(:,m));
    str = sprintf('Time: %d', current_time);
    text(4,0.1,str);
    axis([0 5 0 1]);
    frame(m) = getframe(thisframe);
end

v = VideoWriter(filename);
open(v)
writeVideo(v,frame)
close(v)
end

B.33 dualvideo.m

```matlab
function [] = dualvideo(real_data,gen_data)
% DUALVIDEO Creates comparative videos showing current
and 'historic'
% progressions of the networks in the two data samples
and the distribution
% of node degree over time
%
% REAL_DATA is the file containing the first data sample
as a CSV file
% GEN_DATA is the file containing the second data sample
as a CSV file
structure = '%f %f %f %s %s';
timestamp = datestr(now,'yyyymmddTHHmmss');
dir_ref = ['output_',timestamp];
mkdir(dir_ref);

fid = fopen(real_data);
rawdata_real = textscan(fid,structure,'Delimiter',',');
fclose(fid);

fid = fopen(gen_data);
rawdata_gen = textscan(fid,structure,'Delimiter',',');
fclose(fid);

%== Extract and Clean Data==
	data_real = cell2mat(rawdata_real);
	data_real(:,1) = data_real(:,1)-data_real(1,1);
	targetID = min(min(data_real(:,2)),min(data_real(:,3)));
	data_real(:,2) = data_real(:,2)-targetID+1;
	data_real(:,3) = data_real(:,3)-targetID+1;

parfor i=1:number_rows
    thisrow = data_real(i,:);
    col2 = thisrow(1,2);
    col3 = thisrow(1,3);
    if col2 > col3
        thisrow(1,2) = col3;
        thisrow(1,3) = col2;
        data_real(i,:) = thisrow;
end
```
end
end
all_IDs = [data_real(:,2); data_real(:,3)];
all_active = unique(all_IDs);
num_people = size(all_active,1);
data2 = data_real(:,2);
data3 = data_real(:,3);
for i=1:num_people
    oldID = all_active(i);
data2(data2==oldID) = -i;
data3(data3==oldID) = -i;
end
data_real(:,2) = -data2;
data_real(:,3) = -data3;
data_gen = cell2mat(rawdata_gen);
data_gen(:,1) = data_gen(:,1)-data_gen(1,1);
lowestID = min(min(data_gen(:,2)),min(data_gen(:,3)));
data_gen(:,2) = data_gen(:,2)-lowestID+1;
data_gen(:,3) = data_gen(:,3)-lowestID+1;
number_rows = size(data_gen,1);
parfor i=1:number_rows
    thisrow = data_gen(i,:);
    col2 = thisrow(1,2);
    col3 = thisrow(1,3);
    if col2 > col3
        thisrow(1,2) = col3;
        thisrow(1,3) = col2;
data_gen(i,:) = thisrow;
end
end
all_IDs = [data_gen(:,2); data_gen(:,3)];
all_active = unique(all_IDs);
num_people = size(all_active,1);
data2 = data_gen(:,2);
data3 = data_gen(:,3);
for i=1:num_people
    oldID = all_active(i);
data2(data2==oldID) = -i;
data3(data3==oldID) = -i;
end
data_gen(:,2) = -data2;
data_gen(:,3) = -data3;
create_avi(data_real,data_gen,dir_ref);
DinT_real = deg_in_time(data_real);
DinT_gen = deg_in_time(data_gen);
B.34 EAP_matrix.m

```matlab
function [EAPmat] = EAP_matrix(input_folder, input_filename)
% EAP_MATRIX Creates weighted selection matrix EAPmat based on a given file
% INPUT_FOLDER points to the folder that the file is stored in
% INPUT_FILENAME points to the file within that folder for data extraction
% EAPMAT is the extracted weighted selection matrix

structure = ['\%f \%f \%f \%s \%s'];
if = ['input/','input_folder'];
input = [iF,'/','input_filename'];

fid = fopen(input);
rawdata = textscan(fid,structure,'Delimiter','
fclose(fid);

%== Extract and Clean Data==
data = cell2mat(rawdata);
data(:,1) = data(:,1) - data(1,1);
lowestID = min(min(data(:,2)),min(data(:,3)));
data(:,2) = data(:,2) - lowestID + 1;
data(:,3) = data(:,3) - lowestID + 1;
number_rows = size(data,1);
parfor i=1:number_rows
    thisrow = data(i,:);
    col2 = thisrow(1,2);
    col3 = thisrow(1,3);
    if col2 > col3
        thisrow(1,2) = col3;
        thisrow(1,3) = col2;
        data(i,:) = thisrow;
    end
end
all_IDs = [data(:,2); data(:,3)];
all_active = unique(all_IDs);
um_people = size(all_active,1);
data2 = data(:,2);
data3 = data(:,3);
for i=1:num_people
```
oldID = all_active(i);
data2(data2==oldID) = -i;
data3(data3==oldID) = -i;
end
data(:,2) = -data2;
data(:,3) = -data3;
N=max(max(data(:,2)),max(data(:,3)));
EAPmat = zeros(N);
for i=1:N-1
    parfor j=i+1:N
        S1 = data(:,2)==i;
        S2 = data(:,3)==j;
        T1 = data(:,3)==i;
        T2 = data(:,2)==j;
        S12 = S1 & S2;
        T12 = T1 & T2;
        ST12 = S12 | T12;
        changes = diff(ST12);
        count = sum(changes==1);
        EAPmat(i,j) = count;
    end
end
EAPmat = EAPmat+EAPmat'; %Symmetrises matrix
EAPmat = EAPmat/(sum(sum(EAPmat))); %Normalises matrix

B.35 echoes.m

function [] = echoes(datapath,sampletime)
%ECHOES shows an avi animation file showing the network progress over time
%DATAPATH is the path for the data
%SAMPLETIME is the length of time to animate for
structure = '%f %f %f %s %s';
fid = fopen(datapath);
rawdata = textscan(fid,structure,'Delimiter','','');
fclose(fid);
%== Extract and Clean Data==
data = cell2mat(rawdata);
data(:,1) = data(:,1)-data(1,1);
lowestID = min(min(data(:,2)),min(data(:,3)));  
data(:,2) = data(:,2) - lowestID + 1;  
data(:,3) = data(:,3) - lowestID + 1;  
number_rows = size(data,1);  
parfor i=1:number_rows  
    thisrow = data(i,:);  
    col2 = thisrow(1,2);  
    col3 = thisrow(1,3);  
    if col2 > col3  
        thisrow(1,2) = col3;  
        thisrow(1,3) = col2;  
        data(i,:) = thisrow;  
    end  
end  
all_IDs = [data(:,2); data(:,3)];  
all_active = unique(all_IDs);  
num_people = size(all_active,1);  
data2 = data(:,2);  
data3 = data(:,3);  
for i=1:num_people  
    oldID = all_active(i);  
    data2(data2==oldID) = -i;  
    data3(data3==oldID) = -i;  
end  
data(:,2) = -data2;  
data(:,3) = -data3;  
contact_time = 20;  
scale = 20;  
all_people = [data(:,2)' data(:,3)'];  
dat_times = size(unique(data(:,1)),1);  
data_length = size(data(:,1),1);  
num_people = max(all_people);  
coords = zeros(num_people,2);  
theta = 2*pi/num_people;  
sam_times = ceil(sampletime/contact_time);  
num_times = min(dat_times,sam_times);  
parfor n=1:num_people  
    coords(n,:) = scale*[sin(n*theta) cos(n*theta)];  
end  
step_nf = zeros(num_people,1);  
line_freq = zeros(num_people,num_people);  
for m=1:num_times
thisadj = zeros(num_people);
current_time = (m-1)*contact_time;
for i=1:data_length
    test_time = data(i,1);
    if test_time==current_time
        person1 = data(i,2);
        person2 = data(i,3);
        thisadj(person1,person2) = 1;
        thisadj(person2,person1) = 1;
    end
end
step_nf = step_nf+sum(thisadj,2);
line_freq = line_freq+thisadj;
end

rel_node_freq = step_nf/max(step_nf);
rel_link_freq = line_freq/(max(max(line_freq)));
figure('rend','painters','pos',[250 250 500 500]);
link_size = rel_node_freq*50;
tempadj = logical(rel_link_freq);
[row,col] = find(tempadj);
tempcoords = zeros(num_people,2);
hold on
for i=1:length(row)
    thisrow = row(i);
    thiscol = col(i);
    if thisrow >= thiscol
        line_col = (1-rel_link_freq(thisrow,thiscol))*[1 1 1];
        x1 = coords(thisrow,1);
        y1 = coords(thisrow,2);
        x2 = coords(thiscol,1);
        y2 = coords(thiscol,2);
        line([x1 x2],[y1 y2],Color,line_col)
    end
    tempcoords(i,:) = coords(thisrow,:);
end
tempcoords = tempcoords(any(tempcoords,2),:);
tempcoords = unique(tempcoords,'rows','stable');
link_size(link_size==0) = [];
scatter(tempcoords(:,1),tempcoords(:,2),link_size,'filled ')
hold off
axis([-1.0 1.0 -1.0 1.0]*scale);
axis off;
set(gcf,'color','w');
function xr = emprand(dist, varargin)
% EMPRAND Generates random numbers from empirical
distribution of data.
% This is useful when you do not know the distribution
type (i.e. normal or
% uniform), but you have the data and you want to
% generate random
% numbers form the data. The idea is to first construct
cumulative distribution
% function (cdf) from the given data. Then generate
uniform random number and
% interpolate from cdf.
%
% USAGE:
% xr = EMPRAND(dist) - one random number
% xr = EMPRAND(dist, m) - m-by-m random
% numbers
% xr = EMPRAND(dist, m, n) - m-by-n random
% numbers
%
% INPUT:
% dist - vector of distribution i.e. data values
% m - generates m-by-m matrix of random numbers
% n - generates m-by-n matrix of random numbers
%
% OUTPUT:
% xr - generated random numbers
%
% EXAMPLES:
% % Generate 1000 normal random numbers
% mu = 0; sigma = 1; nr = 1000;
% givenDist = mu + sigma * randn(nr, 1);
% generatedDist = emprand(givenDist, nr, 1);
% %
% % % Plot histogram to check given and generated
distribution
% [n, xout] = hist(givenDist);
% hist(givenDist);
% hold on
% hist(generatedDist, xout)
% %
% % Plot cdf to check given and generated distribution
% figure
x = sort(givenDist(:));  % Given distribution
p = 1:length(x);
plot(x,p,'color','r');
hold on

xr = sort(generatedDist(:));  % Generated distribution
pr = 1:length(xr);
plot(xr,pr,'color','b');

xlabel('x')
ylabel('cdf')
legend('Given Dist.', 'Generated Dist.')
title('1000 random numbers generated from given normal distribution of data');

error(nargchk(1,3,nargin));
if ~isvector(dist)
    error('Invalid data size: input data must be vector')
end
if nargin == 2
    m = varargin{1};
    n = m;
elseif nargin == 3
m = varargin{1};
n = varargin{2};

else
    m = 1;
n = 1;
end

%% COMPUTATION
x = dist(:);
% Remove missing observations indicated by NaN's.
t = ~isnan(x);
x = x(t);

% Compute empirical cumulative distribution function (cdf )
xlen = length(x);
x = sort(x);
p = 1:xlen;
p = p./xlen;

% Generate uniform random number between 0 and 1
ur = rand(m,n);

% Interpolate ur from empirical cdf and extrapolate for out of range % values.
xr = interp1(p,x,ur,[],'extrap');

B.37 explognconvhist.m

function [pdf] = explognconvhist(t)

% Set parameters for exponential and log-normal distributions
mu = 6.3512;
sigma = 1.3688;
lambda = lognrnd(3.5348,0.2807);

Xstep = 1E0; %PDF slice width
maxn = 1+ceil(t/20); %Maximum values of n
Xmax = 1E4; %Last slice for PDFs
N = 1E4; %Number of MC simulations
M = 1E6; %Number of each variable in each simulation

X = 0:Xstep:Xmax;
tidx = find(X==t);
pdf = zeros(1,maxn+1);

lnpdf = lognpdf(X,mu,sigma);
lnpdf = lnpdf/sum(lnpdf); % Normalise PDF
expdf = exppdf(X,lambda);
expdf = expdf/sum(expdf); % Normalise PDF
copdf = conv(lnpdf,expdf); % Create PDF for EX+LN

l = length(copdf);
CCDF = zeros(1,l);
parfor i =1:l
    CCDF(i) = 1- sum(copdf(1: i)); % Create CCDF needed
end

thispdf = copdf;
for i =2: maxn+1
    pdf(i) = sum(thispdf(1: tidx).*fliplr(CCDF(1: tidx)));
    % Convolved * CCDF
    thispdf = conv(thispdf,copdf);
    % Next convolve
end
pdf(1)=1-sum(pdf); % Calculate probability n=0, by inverse

num=zeros(1,N);
% Begin MC simulation
parfor k =1: N
    dt= exprnd(lambda,1,M)+ lognrnd(mu,sigma,1,M);
time = cumsum(dt);
    index=find(time > t,1);
    num(k)=min(index)-1;
end

freq = zeros(1,maxn+1);
parfor j=1:max(num)
    freq(j)=length(find(num==j-1))/N;
end

% Plot on graph
mini = 1E-4;
n=0:maxn;
histimage = figure();
hold on
bar(n,pdf,'hist');
plot(n,freq,'x-');
xlabel('n');
text = ['Prob[N(',num2str(t),',')=n']];
ylabel('text');
histmax = find(pdf >mini,1,'last');
freqmax = find(freq >mini,1,'last');
lastidx = max(histmax,freqmax)+5;
xlim([-0.5 lastidx+0.5])
titleText = ['Parameters: \lambda=',num2str(lambda),', \mu=',num2str(mu),', \sigma=',num2str(sigma)];
title(titleText);
legend('Discrete PDF using Convolve','Monte-Carlo Simulation');
set(gca,'FontSize',18);
end

B.38 extractTriangles.m

function [triangles,mlpm] = extractTriangles(times,nodes,
currenttime,lpm)

%EXTRACT TRIANGLES Calculates current triangle count as well as creating a
% modified weighting matrix that will complete any current triangles
%
% TRIANGLES is the current triangle count
% MLPM is the modified weighting matrix completing any current triangles
%
% TIMES is the (current) output of one of the MODEL functions
% NODES is the node count of the simulation
% CURRENTTIME is the current time in the simulation
% LPM is the weighting matrix used in the MODEL function
% TRIANGLES is the current triangle count at the given time
% MLPM is the reweighted link selection matrix
%
% Calculates current adjacency matrix
adjMat = zeros(nodes);
for i=1:nodes-1
    for j=i+1:nodes
        ID_ref = sprintf('n%d_n%d',i,j);
currentswitch = times(ID_ref);
        if ~isempty(currentswitch)
            if (currenttime<currentswitch(end))&&(currenttime>currentswitch(end-1))
                adjMat(i,j) = 1;
                adjMat(j,i) = 1;
            end
        end
    end
end
triangles = trace(adjMat^3)/6;
%Creates logical matrix showing all edges that complete triangles
trianglesMat = zeros(nodes);
for i=1:nodes
    thisrow = adjMat(i,:);
    nonzeros = find(thisrow);
    nonzeros = [i,nonzeros];
    nonzeros = sort(nonzeros);
    if length(nonzeros)>1
        combos = nchoosek(nonzeros,2);
        cs = size(combos,1);
        for j=1:cs
            n = combos(j,1);
            m = combos(j,2);
            trianglesMat(n,m) = 1;
            trianglesMat(m,n) = 1;
        end
    end
end
mlpm_unw = (trianglesMat-adjMat).*lpm; %Term-by-term multiplication
if sum(sum(mlpm_unw)) == 0 %If new matrix is 0, return old matrix
    mlpm = zeros(nodes);
else
    mlpm = mlpm_unw/(sum(sum(mlpm_unw))); %Otherwise normalise
end

function [shape,scale,location] = gpSolve(M1,M2,M3)
%GPSOLVE takes the first three moments of data and
%attempts to calculate
%the shape, scale and location parameters of a
generalised pareto
%distribution that matches this data


% M1 is the first moment
% M2 is the second moment
% M3 is the third moment
% SHAPE is the calculated shape parameter using the given moments
% SCALE is the calculated scale parameter using the given moments
% LOCATION is the calculated location parameter using the given moments

syms xi sigma mu

eqn1 = mu + sigma/(1-xi) == M1;
eqn2 = mu^2 + (2*mu*sigma)/(1-xi) + (2*sigma^2)/((1-xi)*(1-2*xi)) == M2;
eqn3 = mu^3 + (3*mu^2*sigma)/(1-xi) + (6*mu*sigma^2)/(1-xi)*(1-2*xi) + ... 
(6*sigma^3)/((1-xi)*(1-2*xi)*(1-3*xi)) == M3;
eqn = [eqn1, eqn2, eqn3];
para = [xi, sigma, mu];
[xiSol,sigmaSol,muSol] = solve(eqn,para);

xiSolNum = vpa(xiSol);
sigmaSolNum = vpa(sigmaSol);
muSolNum = vpa(muSol);

xiSolNumR = real(xiSolNum);
sigmaSolNumR = real(sigmaSolNum);
muSolNumR = real(muSolNum);
correctSol = find(sigmaSolNumR >0,1);

xiTrue = xiSolNumR(correctSol);
sigmaTrue = sigmaSolNumR(correctSol);
muTrue = muSolNumR(correctSol);

shape = double(xiTrue);
scale = double(sigmaTrue);
location = double(muTrue);

B.40 individualgraphs.m

function [] = individualgraphs(folder,model,dir_ref)
% INDIVIDUALGRAPHS produces figures comparing all
% simulated samples to
% observed data by plotting eCCDFs in individual pairs
\[ \text{GiF} = [\text{input/folder/model}, \text{model}]; \]
\[ \text{GtoExtract} = [\text{GiF}, \text{/*.csv}]; \]
\[ \text{RiF} = [\text{input/folder/original}]; \]
\[ \text{RtoExtract} = [\text{RiF}, \text{/*.csv}]; \]
\[ \text{mkdir(dir_ref);} \]
\[ \text{GfileData} = \text{dir(GtoExtract);} \]
\[ \text{GfileList} = \{\text{GfileData.name}\}; \]
\[ \text{GfileList} = \text{GfileList(contains(GfileList,'_'))}; \]
\[ \text{RfileData} = \text{dir(RtoExtract);} \]
\[ \text{RfileList} = \{\text{RfileData.name}\}; \]
\[ \text{RfileList} = \text{RfileList(contains(RfileList,'_'))}; \]
\[ \text{for} i=1: \text{length(GfileList)} \]
\[ \text{currentFile} = \text{GfileList}{i}; \]
\[ \text{currentClean} = \text{strrep(currentFile, ' ', '')}; \]
\[ \text{currentClean} = \text{strrep(currentClean, '- ', '')}; \]
\[ \text{currentData} = \text{pullData(GiF, currentFile, '%f %f %f %s %*s'} \]
\[ \text{GActivityPotential.(currentClean) = currentData. ActivityPotential_data; } \]
\[ \text{GInteractionTimes.(currentClean) = currentData. InteractionTimes_data; } \]
\[ \text{GActivityPotential.(currentClean) = currentData. ActivityPotential_data; } \]
\[ \text{GNoContactTimes.(currentClean) = currentData. NoContactTimes_data; } \]
\[ \text{GNodesActive.(currentClean) = currentData. NodesActive_data; } \]
\[ \text{GComponents.(currentClean) = currentData. Components_data; } \]
\[ \text{GClustering.(currentClean) = currentData. Clustering_data; } \]
\[ \text{GComponentNodes.(currentClean) = currentData. ComponentNodes_data; } \]
\[ \text{GComponentEdges.(currentClean) = currentData. ComponentEdges_data; } \]
\[ \text{GTriangles.(currentClean) = currentData. Triangles_data; } \]
for i = 1:length(RfileList)
    currentFile = RfileList{i};
    currentClean = strrep(currentFile, '.', '');
    currentClean = strrep(currentClean, '-', '');
    currentData = pullData(RiF, currentFile, '%f %f %f %*s %*s');
    % Store Data
    RActiveLinks.(currentClean) = currentData.
        ActiveLinks_data;
    RInteractionTimes.(currentClean) = currentData.
        InteractionTimes_data;
    RActivityPotential.(currentClean) = currentData.
        ActivityPotential_data;
    RNoContactTimes.(currentClean) = currentData.
        NoContactTimes_data;
    RNodesActive.(currentClean) = currentData.
        NodesActive_data;
    RComponents.(currentClean) = currentData.
        Components_data;
    RClustering.(currentClean) = currentData.
        Clustering_data;
    RComponentNodes.(currentClean) = currentData.
        ComponentNodes_data;
    RComponentEdges.(currentClean) = currentData.
        ComponentEdges_data;
    RTriangles.(currentClean) = currentData.
        Triangles_data;
end

modelID = strrep(model, '.', '');
modelID = strrep(modelID, '-', '');

for i = 1:length(GfileList)
    for j = 1:length(RfileList)
        currentGfile = GfileList{i};
        currentG = strrep(currentGfile, '.', '');
        currentG = strrep(currentG, '-', '');
        currentRfile = RfileList{j};
        currentR = strrep(currentRfile, '.', '');
        currentR = strrep(currentR, '-', '');
        current_dir = [currentR, '_against_model', modelID, '_', currentG];
        save_dir = [dir_ref, '/', current_dir];
        mkdir(save_dir);
        currentG_AL = GActiveLinks.(currentG);
        currentG_IT = GInteractionTimes.(currentG);
currentG_AP = GActivityPotential.(currentG);
currentG_NC = GNoContactTimes.(currentG);
currentG_AN = GNodesActive.(currentG);
currentG_CC = GComponents.(currentG);
currentG_GC = GClustering.(currentG);
currentG_CN = GComponentNodes.(currentG);
currentG_CL = GComponentEdges.(currentG);
currentG_TR = GTriangles.(currentG);
currentR_AL = RActiveLinks.(currentR);
currentR_IT = RInteractionTimes.(currentR);
currentR_AP = RActivityPotential.(currentR);
currentR_NC = RNoContactTimes.(currentR);
currentR_AN = RNodesActive.(currentR);
currentR_CC = RComponents.(currentR);
currentR_GC = RClustering.(currentR);
currentR_CN = RComponentNodes.(currentR);
currentR_CL = RComponentEdges.(currentR);
currentR_TR = RTriangles.(currentR);
pairgraphs(currentG_AL, currentR_AL, 'Active Links', save_dir);
pairgraphs(currentG_IT, currentR_IT, 'Interaction Time', save_dir);
pairgraphs(currentG_AP, currentR_AP, 'Node Activity Potential', save_dir);
pairgraphs(currentG_NC, currentR_NC, 'Time Between Contacts', save_dir);
pairgraphs(currentG_AN, currentR_AN, 'Active Nodes', save_dir);
pairgraphs(currentG_CC, currentR_CC, 'Component Count', save_dir);
pairgraphs(currentG_GC, currentR_GC, 'Global Clustering Coefficient', save_dir);
pairgraphs(currentG_CN, currentR_CN, 'Nodes per Component', save_dir);
pairgraphs(currentG_CL, currentR_CL, 'Links per Component', save_dir);
pairgraphs(currentG_TR, currentR_TR, 'Triangle Count', save_dir);
end

B.41 JSDiv.m

function dist=JSDiv(P,Q)

% Jensen-Shannon divergence of two probability distributions
% dist = JSD(P,Q) Kullback-Leibler divergence of two
discrete probability
% distributions
% P and Q are automatically normalised to have the sum
% of one on rows
% have the length of one at each
% P = n x nbins
% Q = 1 x nbins
% dist = n x 1

if size(P,2)~=size(Q,2)
    error('the number of columns in P and Q should be the
    same');
end

% normalizing the P and Q
Q = Q ./ sum(Q);
Q = repmat(Q,[size(P,1) 1]);
P = P ./ repmat(sum(P,2),[1 size(P,2)]);
M = 0.5.*(P + Q);

dist = 0.5.*KLDiv(P,M) + 0.5*KLDiv(Q,M);

B.42 KLDiv.m

function dist=KLDiv(P,Q)
% dist = KLDiv(P,Q) Kullback-Leibler divergence of two
discrete probability
% distributions
% P and Q are automatically normalised to have the sum
% of one on rows
% have the length of one at each
% P = n x nbins
% Q = 1 x nbins or n x nbins(one to one)
% dist = n x 1

if size(P,2)~=size(Q,2)
    error('the number of columns in P and Q should be the
    same');
end

if sum(~isfinite(P(:))) + sum(~isfinite(Q(:)))
    error('the inputs contain non-finite values!')
end
end

% normalizing the P and Q
if size(Q,1)==1
    Q = Q ./ sum(Q);
    P = P ./ repmat(sum(P,2),[1 size(P,2)]);
    dist = sum(P.*log(P./ repmat(Q,[size(P,1) 1])),2);
elseif size(Q,1)==size(P,1)
    Q = Q ./ repmat(sum(Q,2),[1 size(Q,2)]);
    P = P ./ repmat(sum(P,2),[1 size(P,2)]);
    dist = sum(P.*log(P./Q),2);
end

% resolving the case when P(i)==0
dist(isnan(dist))=0;

B.43  ksampleKS.m

function [] = ksampleKS(folder)
  % KSAMPLEKS produces a latex txt file containing
  % statistics and p-values for
  % the multisample Kolmogorov-Smirnov test
  %
  % FOLDER is the folder path of the directory containing
  % all samples to be
  % tested
  timestamp = datestr(now,'yyyyMMddTHHMMSs');
  dir_ref = ['output_',timestamp];
  mkdir(dir_ref);
  filename = 'ksampleKS.txt';
  filepath = [dir_ref,'/ksampleKS.txt'];
  iF = ['input/',folder];
  toExtract = [iF,'/*.csv'];
  fileData = dir(toExtract);
  fileList = {fileData.name};
  fileList = fileList(~contains(fileList,'._'));
  fActiveLinks = cell(length(fileList),1);
  fInteractionTimes = cell(length(fileList),1);
  fActivityPotential = cell(length(fileList),1);
  fNoContactTimes = cell(length(fileList),1);
for i = 1:length(fileList)
    currentFile = fileList{i};
    currentData = pullData(iF, currentFile, '%f %f %f %*s %*s');
    fActiveLinks{i,1} = currentData.ActiveLinks_data;
    fInteractionTimes{i,1} = currentData.InteractionTimes_data;
    fActivityPotential{i,1} = currentData.ActivityPotential_data;
    fNoContactTimes{i,1} = currentData.NoContactTimes_data;
    fNodesActive{i,1} = currentData.NodesActive_data;
    fComponents{i,1} = currentData.Components_data;
    fClustering{i,1} = currentData.Clustering_data;
    fComponentNodes{i,1} = currentData.ComponentNodes_data;
    fComponentEdges{i,1} = currentData.ComponentEdges_data;
    fTriangles{i,1} = currentData.Triangles_data;
end

AL = multiKS(fActiveLinks);
IT = multiKS(fInteractionTimes);
AP = multiKS(fActivityPotential);
NC = multiKS(fNoContactTimes);
NA = multiKS(fNodesActive);
CO = multiKS(fComponents);
CC = multiKS(fClustering);
CN = multiKS(fComponentNodes);
CE = multiKS(fComponentEdges);
TC = multiKS(fTriangles);

[pAL,˜] = probKS(fActiveLinks);
[pIT,˜] = probKS(fInteractionTimes);
[pAP,˜] = probKS(fActivityPotential);
[pNC,˜] = probKS(fNoContactTimes);
[pNA,˜] = probKS(fNodesActive);
[pCO,˜] = probKS(fComponents);
[pCC,˜] = probKS(fClustering);
[pCN,˜] = probKS(fComponentNodes);
\[ \frac{1}{\text{Component Count}} = \text{probKS( Component Edges )}; \]
\[ \frac{1}{\text{Component Count}} = \text{probKS( Component Edges )}; \]
\[ \text{lines} = 13; \]
\[ \text{tobuild} = \text{cell}(\text{lines},1); \]
\[ \text{tobuild\{01\}} = '\begin{tabular}{l|c|c|} \hline 2 & 3 \\ \hline \end{tabular}'. \]
\[ \text{tobuild\{02\}} = ' & \text{Statistic} & \$p$-Value \hline'; \]
\[ \text{tobuild\{03\}} = ['\text{multicolumn}{1}{|l}{Active Links}\rangle & \$\frac{1}{2}\,$, \text{num2str(AL,4)},'\$ & ',$\frac{1}{2}\,$, \text{num2str(1-pAL,4)},'\$\\ \hline' ]; \]
\[ \text{tobuild\{04\}} = ['\text{multicolumn}{1}{|l}{Active Nodes}\rangle & \$\frac{1}{2}\,$, \text{num2str(IT,4)},'\$ & ',$\frac{1}{2}\,$, \text{num2str(1-pIT,4)},'\$\\ \hline' ]; \]
\[ \text{tobuild\{05\}} = ['\text{multicolumn}{1}{|l}{Node Activity Potential}\rangle & \$\frac{1}{2}\,$, \text{num2str(AP,4)},'\$ & ',$\frac{1}{2}\,$, \text{num2str(1-pAP,4)},'\$\\ \hline' ]; \]
\[ \text{tobuild\{06\}} = ['\text{multicolumn}{1}{|l}{Global Clustering Coefficient}\rangle & \$\frac{1}{2}\,$, \text{num2str(NC,4)},'\$ & ',$\frac{1}{2}\,$, \text{num2str(1-pNC,4)},'\$\\ \hline' ]; \]
\[ \text{tobuild\{07\}} = ['\text{multicolumn}{1}{|l}{Interaction Time}\rangle & \$\frac{1}{2}\,$, \text{num2str(NA,4)},'\$ & ',$\frac{1}{2}\,$, \text{num2str(1-pNA,4)},'\$\\ \hline' ]; \]
\[ \text{tobuild\{08\}} = ['\text{multicolumn}{1}{|l}{Time Between Contacts}\rangle & \$\frac{1}{2}\,$, \text{num2str(CO,4)},'\$ & ',$\frac{1}{2}\,$, \text{num2str(1-pCO,4)},'\$\\ \hline' ]; \]
\[ \text{tobuild\{09\}} = ['\text{multicolumn}{1}{|l}{Component Count}\rangle & \$\frac{1}{2}\,$, \text{num2str(CC,4)},'\$ & ',$\frac{1}{2}\,$, \text{num2str(1-pCC,4)},'\$\\ \hline' ]; \]
\[ \text{tobuild\{10\}} = ['\text{multicolumn}{1}{|l}{Links per Component}\rangle & \$\frac{1}{2}\,$, \text{num2str(CN,4)},'\$ & ',$\frac{1}{2}\,$, \text{num2str(1-pCN,4)},'\$\\ \hline' ]; \]
\[ \text{tobuild\{11\}} = ['\text{multicolumn}{1}{|l}{Nodes per Component}\rangle & \$\frac{1}{2}\,$, \text{num2str(CE,4)},'\$ & ',$\frac{1}{2}\,$, \text{num2str(1-pCE,4)},'\$\\ \hline' ]; \]
\[ \text{tobuild\{12\}} = ['\text{multicolumn}{1}{|l}{Triangle Count}\rangle & \$\frac{1}{2}\,$, \text{num2str(TC,4)},'\$ & ',$\frac{1}{2}\,$, \text{num2str(1-pTC,4)},'\$\\ \hline' ]; \]
\[ \text{tobuild\{13\}} = '\end{tabular}'. \]
\[ \text{fileID} = \text{fopen}(\text{filepath},'w'); \]
\[ \text{fprintf(fileID,}'\%s\n\text{\textbackslash
',\text{tobuild{}\{}); \]
\[ \text{fclose(fileID);} \]

B.44 \texttt{la2latex.m}
function [] = la2latex(FitTool, MLE, Moments, num_people, dir_ref, name, opstat)
% LA2LATEX outputs data as a latex matrix
% FITTOOL is the fit tool data structure
% MLE is the most likelihood estimators data structure
% MOMENTS is the method of moments data structure
% NUM_PEOPLE is the number of people in the network
% DIR_REF is the save directory
% NAME is the property name
% OPSTAT is a flag that determines the format of the given data structure

cleanName = strrep(name, '.', '');
cleanName = strrep(cleanName, '', '_');
filename = ['LaTexTables_Local_', cleanName, '.txt'];
filepath = [dir_ref, '/', filename];
datasource = dir_ref(24:end-3);

if opstat == 1 % ExpGamRayLN
    lines = 53;
tobuild = cell(lines,1);

    tobuild{001} = ['Data From: ', datasource];
    tobuild{002} = ['subsection{Results: ', datasource, ' ', num2str(num_people), ' Students}'];
    tobuild{003} = ['begin{tabular}{{|c|c|c||c|c||c|c||c|c}}

    tobuild{004} = ['parbox[t]{2mm}{\multirow{16}{*}{'
    rotatebox[origin=c]{90}{Method of Moments}} & \multicolumn{2}{|c||}{\textbf{Parameters}} & \multirow{2}{*}{\textbf{Rate}} & \multirow{2}{*}{$\textbf{Scale}$} & \multirow{2}{*}{$\textbf{Loc.}$} & \multirow{2}{*}{$\textbf{Location}$} \cline{6-7}\cline{10-11}'];

    tobuild{005} = ['parbox[t]{2mm}{\multirow{16}{*}{\rotatebox[origin=c]{90}{\textbf{Parameters}}}} & \multirow{2}{*}{\textbf{Rate}} & \multirow{2}{*}{$\textbf{Scale}$} & \multirow{2}{*}{$\textbf{Location}$} \cline{6-7}\cline{10-11}'];

    tobuild{006} = ['& \multicolumn{2}{|c||}{} & & & \textbf{Scale} & \textbf{Location} & \textbf{Scale} & \textbf{Location} & \textbf{Scale} & \textbf{Location} \cline{6-7}\cline{10-11}'];
| 30 | \begin{verbatim}
\end{verbatim} |
| 31 | \begin{verbatim}
\end{verbatim} |
| 32 | \begin{verbatim}
tobuild{009} = ['\& Kuiper & \multicolumn{2}{c||}{num2matlabstr(Moments.Exponential.Statistics.Kuiper)},'\$ & \multicolumn{2}{c||}{num2matlabstr(Moments.Gamma.Statistics.Kuiper)},'\$ & \multicolumn{2}{c||}{num2matlabstr(Moments.Rayleigh.Statistics.Kuiper)},'\$ & \multicolumn{2}{c||}{num2matlabstr(Moments.LogNormal.Statistics.Kuiper)},'\$ \ \ \ \ \cline{2-11}\$
\end{verbatim} |
| 33 | \begin{verbatim}
tobuild{010} = ['\& Watson & \multicolumn{2}{c||}{num2matlabstr(Moments.Exponential.Statistics.Watson)},'\$ & \multicolumn{2}{c||}{num2matlabstr(Moments.Gamma.Statistics.Watson)},'\$ & \multicolumn{2}{c||}{num2matlabstr(Moments.Rayleigh.Statistics.Watson)},'\$ & \multicolumn{2}{c||}{num2matlabstr(Moments.LogNormal.Statistics.Watson)},'\$ \ \ \ \ \cline{2-11}\$
\end{verbatim} |
| 34 | \begin{verbatim}
tobuild{011} = ['\& And-Dar & \multicolumn{2}{c||}{num2matlabstr(Moments.Exponential.Statistics.Anderson Darling)},'\$ & \multicolumn{2}{c||}{num2matlabstr(Moments.Gamma.Statistics.Anderson Darling)},'\$ & \multicolumn{2}{c||}{num2matlabstr(Moments.Rayleigh.Statistics.Anderson Darling)},'\$ & \multicolumn{2}{c||}{num2matlabstr(Moments.LogNormal.Statistics.Anderson Darling)},'\$ \ \ \ \ \cline{2-11}\$
\end{verbatim} |
| 35 | \begin{verbatim}
tobuild{012} = ['\& Kul-Lei & \multicolumn{2}{c||}{num2matlabstr(Moments.Exponential.Statistics.} |

The text above contains LaTeX code for building various statistical tests and their associated MATLAB commands for handling moments of different distributions.
<table>
<thead>
<tr>
<th>Tobuild</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
<th>Column 5</th>
<th>Column 6</th>
<th>Column 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>{014}</td>
<td>&amp; multirow{7}{*}{\rotatebox{90}{p-Values}} &amp; Kol D &amp; \multicolumn{2}{c</td>
<td></td>
<td>}{{}'}, num2matlabstr(Moments.Exponential.pValues.Kolmogorov_D), '${}' &amp; \multicolumn{2}{c</td>
<td></td>
<td>}{{}'}, num2matlabstr(Moments.Gamma.pValues.Kolmogorov_D), '${}' &amp; \multicolumn{2}{c</td>
<td></td>
<td>}{{}'}, num2matlabstr(Moments.Rayleigh.pValues.Kolmogorov_D), '${}' &amp; \multicolumn{2}{c</td>
</tr>
<tr>
<td>{017}</td>
<td>&amp; Watson &amp; \multicolumn{2}{c</td>
<td></td>
<td>}{{}'}, num2matlabstr(Moments.Exponential.pValues.Watson) , '${}' &amp; \multicolumn{2}{c</td>
<td></td>
<td>}{{}'}, num2matlabstr(Moments.Gamma.pValues.Watson) , '${}' &amp; \multicolumn{2}{c</td>
<td></td>
<td>}{{}'}, num2matlabstr(Moments.Rayleigh.pValues.Watson) , '${}' \ cline{3-11}</td>
</tr>
</tbody>
</table>
tobuild{018} = ['& & And-Dar & \multicolumn{2}{c||}{{$'}},
num2matlabstr(Moments.Exponential.pValues.Anderson_Darling),'$'} \ \
\cline{3-11}'];

tobuild{019} = ['& & Kul-Lei & \multicolumn{2}{c||}{{$'}},
num2matlabstr(Moments.Exponential.pValues.
Kullback_Leibler),'$'} \ \
\cline{3-11}'];

tobuild{020} = ['& & Jen-Sha & \multicolumn{2}{c||}{{$'}},
num2matlabstr(Moments.Exponential.pValues.
Jensen_Shannon),'$'} \ \
\cline{3-11}'];

tobuild{021} = ['\parbox[t]{2mm}{\multirow{16}{*}{\rotatebox[origin=c]{90}{Curve Fitting Tool}}}&
\multicolumn{2}{c||}{\multirow{2}{*}{Parameters}}& Rate &
\multirow{2}{*}{$'}\textbf{\num2matlabstr(FitTool.Exponential.}
Parameters.Scale),'$'} & \num2matlabstr(FitTool.Gamma.}
Parameters.Shape),'$'}\ & \multirow{2}{*}{$'}\textbf{\num2matlabstr(FitTool.Rayleigh.}
Parameters.Scale),'$'} & \num2matlabstr(FitTool.LogNormal.}
Parameters.Location),'$'} \ \
\cline{6-7}\cline{10-11}'];

tobuild{022} = ['& & Scale & $'}\num2matlabstr(FitTool.Gamma.}
Parameters.Scale),'$'}& \& & Scale & $'}\num2matlabstr(FitTool.
LogNormal.Parameters.Scale),'$'} \ \
\cline{2-11}'];

tobuild{023} = ['\parbox[t]{2mm}{\multirow{7}{*}{\rotatebox[origin=c]{90}{Statistics}}}&
\multicolumn{2}{c||}{\multirow{2}{*}{Scale}} &\textbf{\num2matlabstr(FitTool.Rayleigh.}
Parameters.Scale),'$'} & \& & \textbf{\num2matlabstr(FitTool.LogNormal.}
Parameters.Scale),'$'}\ & \textbf{\num2matlabstr(FitTool.}
LogNormal.Parameters.Scale),'$'} \ \
\cline{2-11}']:
| 47 | tobuild{024} = ['& & CvM & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Exponential.Simulation.Kolmogorov_D), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Gamma.Simulation.Kolmogorov_D), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Rayleigh.Simulation.Kolmogorov_D), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.LogNormal.Simulation.Kolmogorov_D), '$} \ \ \ \ \ \ cline{3-11}'; |
| 48 | tobuild{025} = ['& & Kuiper & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Exponential.Simulation.Kramer_von_Mises), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Gamma.Simulation.Kramer_von_Mises), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Rayleigh.Simulation.Kramer_von_Mises), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.LogNormal.Simulation.Kramer_von_Mises), '$} \ \ \ \ \ \ cline{3-11}'; |
| 49 | tobuild{026} = ['& & Watson & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Exponential.Simulation.Kullback_Leibler), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Gamma.Simulation.Kullback_Leibler), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Rayleigh.Simulation.Kullback_Leibler), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.LogNormal.Simulation.Kullback_Leibler), '$} \ \ \ \ \ \ cline{3-11}'; |
| 50 | tobuild{027} = ['& & And-Dar & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Exponential.Simulation.Anderson_Darling), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Gamma.Simulation.Anderson_Darling), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Rayleigh.Simulation.Anderson_Darling), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.LogNormal.Simulation.Anderson_Darling), '$} \ \ \ \ \ \ cline{3-11}'; |
| 51 | tobuild{028} = ['& & Kul-Lei & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Exponential.Simulation.Kullback_Leibler), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Gamma.Simulation.Kullback_Leibler), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.Rayleigh.Simulation.Kullback_Leibler), '$} & \multicolumn{2}{c||}{$', num2matlabstr(FitTool.LogNormal.Simulation.Kullback_Leibler), '$} \ \ \ \ \ \ cline{3-11}'; |
\texttt{num2matlabstr(FitTool.LogNormal.Statistics.Kullback-Leibler)},'\$' \ \ \texttt{\cline\{3-11\}}';
\texttt{tobuild\{029\} = ['& & Jen-Sha & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Exponential.Statistics.Jensen_Shannon),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Gamma.Statistics.Jensen_Shannon),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Rayleigh.Statistics.Jensen_Shannon),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.LogNormal.Statistics.Jensen_Shannon),'\$' \ \ \texttt{\cline\{2-11\}}'];
\texttt{tobuild\{030\} = ['& \ \parbox[t]\{2mm\}\{\multicolumn\{7\}\{*\}\{\rotatebox[origin=c]{90}{p-Values}}\} & Kol D & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Exponential.pValues.Kolmogorov_D),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Gamma.pValues.Kolmogorov_D),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Rayleigh.pValues.Kolmogorov_D),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.LogNormal.pValues.Kolmogorov_D),'\$' \ \ \texttt{\cline\{3-11\}}'];
\texttt{tobuild\{031\} = ['& & CvM & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Exponential.pValues.Cramer_von_Mises),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Gamma.pValues.Cramer_von_Mises),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Rayleigh.pValues.Cramer_von_Mises),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.LogNormal.pValues.Cramer_von_Mises),'\$' \ \ \texttt{\cline\{3-11\}}'];
\texttt{tobuild\{032\} = ['& & Kuiper & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Exponential.pValues.Kuiper),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Gamma.pValues.Kuiper),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Rayleigh.pValues.Kuiper),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.LogNormal.pValues.Kuiper),'\$' \ \ \texttt{\cline\{3-11\}}'];
\texttt{tobuild\{033\} = ['& & Watson & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Exponential.pValues.Watson),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Gamma.pValues.Watson),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Rayleigh.pValues.Watson),'\$' & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.LogNormal.pValues.Watson),'\$' \ \ \texttt{\cline\{3-11\}}'];
\texttt{tobuild\{034\} = ['& & And-Dar & \ multicolumn\{2\}\{c\|c\\}\{\$'
\',num2matlabstr(FitTool.Exponential.pValues.
<table>
<thead>
<tr>
<th>Tobuild</th>
<th></th>
<th>Statistics</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>040</td>
<td>num2matlabstr(MLE.LogNormal.Statistics.Kolmogorov_D),</td>
<td>&amp; &amp; Cvm &amp;</td>
<td>multicol {2}{c</td>
<td></td>
</tr>
<tr>
<td>041</td>
<td>num2matlabstr(MLE.Exponential.Statistics.Kuiper),</td>
<td>&amp; &amp; Kuiper &amp;</td>
<td>multicol {2}{c</td>
<td></td>
</tr>
<tr>
<td>042</td>
<td>num2matlabstr(MLE.Exponential.Statistics.Watson),</td>
<td>&amp; &amp; Watson &amp;</td>
<td>multicol {2}{c</td>
<td></td>
</tr>
<tr>
<td>043</td>
<td>num2matlabstr(MLE.Exponential.Statistics.Anderson_Darling),</td>
<td>&amp; &amp; And-Dar &amp;</td>
<td>multicol {2}{c</td>
<td></td>
</tr>
<tr>
<td>044</td>
<td>num2matlabstr(MLE.Exponential.Statistics.Kullback_Leibler),</td>
<td>&amp; &amp; Kul-Lei &amp;</td>
<td>multicol {2}{c</td>
<td></td>
</tr>
<tr>
<td>045</td>
<td>num2matlabstr(MLE.Exponential.Statistics.Jensen_Shannon),</td>
<td>&amp; &amp; Jen-Sha &amp;</td>
<td>multicol {2}{c</td>
<td></td>
</tr>
<tr>
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
<td>Column 4</td>
<td>Column 5</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>Kolmogorov-D</td>
<td>Cramer-von-Mises</td>
<td>Kuiper</td>
<td>Watson</td>
<td>Anderson-Darling</td>
</tr>
<tr>
<td>num2matlabstr(MLE.Exponential.pValues.Kuiper), '}' &amp; \ multicolumn{2}{</td>
<td>c</td>
<td>}{}, num2matlabstr(MLE.Gamma.pValues.Kuiper), '}' &amp; \ multicolumn{2}{</td>
<td>c</td>
<td>}{}, num2matlabstr(MLE.Rayleigh.pValues.Kuiper), '}' &amp; \ multicolumn{2}{</td>
</tr>
<tr>
<td>num2matlabstr(MLE.Exponential.pValues.Watson), '}' &amp; \ multicolumn{2}{</td>
<td>c</td>
<td>}{}, num2matlabstr(MLE.Gamma.pValues.Watson), '}' &amp; \ multicolumn{2}{</td>
<td>c</td>
<td>}{}, num2matlabstr(MLE.Rayleigh.pValues.Watson), '}' &amp; \ multicolumn{2}{</td>
</tr>
<tr>
<td>num2matlabstr(MLE.Exponential.pValues.Anderson_Darling), '}' &amp; \ multicolumn{2}{</td>
<td>c</td>
<td>}{}, num2matlabstr(MLE.Gamma.pValues.Anderson_Darling), '}' &amp; \ multicolumn{2}{</td>
<td>c</td>
<td>}{}, num2matlabstr(MLE.Rayleigh.pValues.Anderson_Darling), '}' &amp; \ multicolumn{2}{</td>
</tr>
<tr>
<td>num2matlabstr(MLE.Exponential.pValues.Kullback-Leibler), '}' &amp; \ multicolumn{2}{</td>
<td>c</td>
<td>}{}, num2matlabstr(MLE.Gamma.pValues.Kullback-Leibler), '}' &amp; \ multicolumn{2}{</td>
<td>c</td>
<td>}{}, num2matlabstr(MLE.Rayleigh.pValues.Kullback-Leibler), '}' &amp; \ multicolumn{2}{</td>
</tr>
<tr>
<td>Method</td>
<td>Exponential</td>
<td>Mittag-Leffler</td>
<td>Gen. Pareto</td>
<td>Weibull</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
<td>----------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>Parameters</td>
<td>$\mu$</td>
<td>$\alpha$</td>
<td>$\beta$</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>Rate</td>
<td>$\lambda$</td>
<td>$\lambda$</td>
<td>$\lambda$</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>Stability</td>
<td>$\kappa$</td>
<td>$\kappa$</td>
<td>$\kappa$</td>
<td>$\kappa$</td>
</tr>
<tr>
<td>Shape</td>
<td>$\kappa$</td>
<td>$\kappa$</td>
<td>$\kappa$</td>
<td>$\kappa$</td>
</tr>
<tr>
<td>Scale</td>
<td>$\sigma$</td>
<td>$\sigma$</td>
<td>$\sigma$</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>Location</td>
<td>$\mu$</td>
<td>$\mu$</td>
<td>$\mu$</td>
<td>$\mu$</td>
</tr>
<tr>
<td>Kol D</td>
<td>$\delta$</td>
<td>$\delta$</td>
<td>$\delta$</td>
<td>$\delta$</td>
</tr>
</tbody>
</table>

elseif opstat == 2 %ExpMLGPWei

lines = 56;
tobuild = cell(lines,1);

tobuild{001} = [%Data From: ,datasource];
tobuild{002} = ['\subsection{Results: ,datasource, 
[',num2str(num_people), ' Students]}'];
tobuild{003} = ['\begin{tabular}{| c|c|c||c|c||c|c||c|c|} \\ hline 
\textbf{name} ,
\textbf{Parameters} \multicolumn{2}{| c ||}{\textbf{Rate}} \multicolumn{2}{| c ||}{\textbf{Shape}} \multicolumn{2}{| c |}{\textbf{Shape}} \multicolumn{2}{| c |}{\textbf{Scale}} \multicolumn{2}{| c ||}{\textbf{Shape}} \multicolumn{2}{| c ||}{\textbf{Shape}} \multicolumn{2}{| c |}{\textbf{Scale}} \multicolumn{2}{| c |}{\textbf{Scale}} \multicolumn{2}{| c ||}{\textbf{Location}} \multicolumn{2}{| c |}{\textbf{Scale}} \multicolumn{2}{| c ||}{\textbf{Scale}} \multicolumn{2}{| c |}{\textbf{Kol D}} \multicolumn{2}{| c ||}{\textbf{Kol D}} \multicolumn{2}{| c |}{\textbf{Kol D}} \multicolumn{2}{| c ||}{\textbf{Kol D}} \\ hline 
\textbf{Method of Moments}} & \textbf{Parameters}} & \textbf{Rate}} & \textbf{Shape}} & \textbf{Shape}} & \textbf{Scale}} & \textbf{Shape}} & \textbf{Scale}} & \textbf{Location}} & \textbf{Kol D}} & \textbf{Kol D}} & \textbf{Kol D}} \\ \cline{6-11} 
\end{tabular}'];
tobuild{004} = [']
\parbox[t]{2 mm}{\multirow{19}{*}{
rotatebox[origin=c]{90}{Method of Moments}}}
\parbox[t]{2 mm}{\multirow{3}{*}{
\textbf{Parameters}}} & \multirow{3}{*}{\textbf{Rate}} & \multirow{3}{*}{\textbf{Shape}} & \multirow{2}{*}{\textbf{Scale}} & \multirow{2}{*}{\textbf{Shape}} & \multirow{2}{*}{\textbf{Scale}} & \multirow{2}{*}{\textbf{Location}} & \multirow{2}{*}{\textbf{Kol D}} \\ \cline{6-11} 
\end{tabular}'];
tobuild{005} = [']
\parbox[t]{2 mm}{\multirow{7}{*}{
rotatebox[origin=c]{90}{Statistics}}}
\parbox[t]{2 mm}{\multirow{2}{*}{Kol D}} & \multirow{2}{*}{\textbf{Shape}} & \multirow{2}{*}{\textbf{Scale}} & \multirow{2}{*}{\textbf{Location}} & \multirow{2}{*}{\textbf{Kol D}} \\ \cline{6-11} 
\end{tabular}'];
tobuild{006} = [']
\parbox[t]{2 mm}{\multirow{7}{*}{
rotatebox[origin=c]{90}{Statistics}}}
\parbox[t]{2 mm}{\multirow{2}{*}{Kol D}} & \multirow{2}{*}{\textbf{Shape}} & \multirow{2}{*}{\textbf{Scale}} & \multirow{2}{*}{\textbf{Location}} & \multirow{2}{*}{\textbf{Kol D}} \\ \cline{6-11} 
\end{tabular}'];
tobuild{007} = [']
\parbox[t]{2 mm}{\multirow{7}{*}{
rotatebox[origin=c]{90}{Statistics}}}
\parbox[t]{2 mm}{\multirow{2}{*}{Kol D}} & \multirow{2}{*}{\textbf{Shape}} & \multirow{2}{*}{\textbf{Scale}} & \multirow{2}{*}{\textbf{Location}} & \multirow{2}{*}{\textbf{Kol D}} \\ \cline{6-11} 
\end{tabular}'];
tobuild{008} = [']
\parbox[t]{2 mm}{\multirow{7}{*}{
rotatebox[origin=c]{90}{Statistics}}}
\parbox[t]{2 mm}{\multirow{2}{*}{Kol D}} & \multirow{2}{*}{\textbf{Shape}} & \multirow{2}{*}{\textbf{Scale}} & \multirow{2}{*}{\textbf{Location}} & \multirow{2}{*}{\textbf{Kol D}} \\ \cline{6-11} 
\end{tabular}'];
| & & Watson \ & \multicolumn{2}{c||}{ stap } \par, num2matlabstr ( Moments . Exponential . Statistics . Watson ) ; |
| & & And\_Dar & \multicolumn{2}{c||}{ stap } \par, num2matlabstr ( Moments . Exponential . Statistics . Anderson\_Darling ) ; |
| & & Kul\_Leib & \multicolumn{2}{c||}{ stap } \par, num2matlabstr ( Moments . Exponential . Statistics . Kullback\_Leibler ) ; |
| & & Jen\_Sha & \multicolumn{2}{c||}{ stap } \par, num2matlabstr ( Moments . Exponential . Statistics . Jensen\_Shannon ) ; |
| & parbox[t]{2 mm }{ multirow{7}{*}{\rotatebox{90}{ p\_Values } \par, num2matlabstr ( Moments . Exponential . pValues . Kolmogorov\_D ) ; |

```
cline{3-11}';
tobuild{016} = ['& & CvM & \multicolumn{2}{c||}{$',
num2matlabstr(Moments.Exponential.pValues.
Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$X$}
& \multicolumn{2}{c||}{$',num2matlabstr(Moments.
GenPareto.pValues.Cramer_von_Mises),'$} & \multicolumn{2}{c||}{
\cline{3-11}'};
tobuild{017} = ['& & Kuiper & \multicolumn{2}{c||}{$'
,num2matlabstr(Moments.Exponential.pValues.Kuiper)
,'$} & \multicolumn{2}{c||}{$X$}
& \multicolumn{2}{c||}{$',num2matlabstr(Moments.GenPareto.
pValues.Kuiper),'$} & \multicolumn{2}{c||}{
\cline{3-11}'};
tobuild{018} = ['& & Watson & \multicolumn{2}{c||}{$'
,num2matlabstr(Moments.Exponential.pValues.Watson)
,'$} & \multicolumn{2}{c||}{$X$}
& \multicolumn{2}{c||}{$',num2matlabstr(Moments.GenPareto.
pValues.Watson),'$} & \multicolumn{2}{c||}{
\cline{3-11}'};
tobuild{019} = ['& & And-Dar & \multicolumn{2}{c||}{$'
,num2matlabstr(Moments.Exponential.pValues.
Anderson_Darling),'$} & \multicolumn{2}{c||}{$X$}
& \multicolumn{2}{c||}{$',num2matlabstr(Moments.
GenPareto.pValues.Anderson_Darling),'$} & \multicolumn{2}{c||}{
\cline{3-11}'};
tobuild{020} = ['& & Kul-Lei & \multicolumn{2}{c||}{$'
,num2matlabstr(Moments.Exponential.pValues.
Kullback_Leibler),'$} & \multicolumn{2}{c||}{$X$}
& \multicolumn{2}{c||}{$',num2matlabstr(Moments.
GenPareto.pValues.Kullback_Leibler),'$} & \multicolumn{2}{c||}{
\cline{3-11}'};
tobuild{021} = ['& & Jen-Sha & \multicolumn{2}{c||}{$'
,num2matlabstr(Moments.Exponential.pValues.
Jensen_Shannon),'$} & \multicolumn{2}{c||}{$X$}
& \multicolumn{2}{c||}{$',num2matlabstr(Moments.
GenPareto.pValues.Jensen_Shannon),'$} & \multicolumn{2}{c||}{
\cline{3-11}'};
tobuild{022} = ['\parbox[t]{2mm}{\multirow{19}{*}{\rotatebox[origin=c]{90}{Curve Fitting Tool}}}} & \
\multicolumn{2}{|c||}{\multirow{3}{*}{Parameters}} & \multirow{3}{*}{Rate} & \multirow{3}{*}{$',num2matlabstr(FitTool.Exponential.
Parameters.Scale),'$} & Stability & $',
num2matlabstr(FitTool.MittagLeffler.Parameters.
Stability),'$ & Shape & $',num2matlabstr(FitTool.
GenPareto.Parameters.Shape),'$ & Scale & $',
num2matlabstr(FitTool.Weibull.Parameters.Scale),'$ 
\\ & \cline{6-11}']";
<table>
<thead>
<tr>
<th>Build</th>
<th>Multicolumn</th>
<th>Multicolumn</th>
<th>Multicolumn</th>
<th>Multicolumn</th>
<th>Multicolumn</th>
<th>Multicolumn</th>
<th>Multicolumn</th>
<th>Multicolumn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale</td>
<td>Scale</td>
<td>$\text{Scale}$</td>
<td>$\text{Shape}$</td>
<td>$\text{Scale}$</td>
<td>$\text{Shape}$</td>
<td>$\text{Location}$</td>
<td>$\text{Statistics}$</td>
<td>$\text{Statistics}$</td>
</tr>
<tr>
<td>$\text{Shape}$</td>
<td>$\text{Shape}$</td>
<td>$\text{Shape}$</td>
<td>$\text{Shape}$</td>
<td>$\text{Shape}$</td>
<td>$\text{Shape}$</td>
<td>$\text{Shape}$</td>
<td>$\text{Shape}$</td>
<td>$\text{Shape}$</td>
</tr>
</tbody>
</table>
tobuild{029} = ['& & And-Dar & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.Exponential.Statistics.
Anderson_Darling),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.MittagLeffler.Statistics.
Anderson_Darling),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.GenPareto.Statistics.
Anderson_Darling),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.Weibull.Statistics.
Anderson_Darling),'$} \ \ \cline{3-11}
];
tobuild{030} = ['& & Kul-Lei & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.Exponential.Statistics.
Kullback_Leibler),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.MittagLeffler.Statistics.
Kullback_Leibler),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.GenPareto.Statistics.
Kullback_Leibler),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.Weibull.Statistics.
Kullback_Leibler),'$} \ \ \cline{3-11}
];
tobuild{031} = ['& & Jen-Sha & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.Exponential.Statistics.
Jensen_Shannon),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.MittagLeffler.Statistics.
Jensen_Shannon),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.GenPareto.Statistics.
Jensen_Shannon),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.Weibull.Statistics.
Jensen_Shannon),'$} \ \ \cline{3-11}
];
tobuild{032} = ['& & CvM & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.Exponential.pValues.
Kolmogorov_D),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.MittagLeffler.pValues.
Kolmogorov_D),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.GenPareto.pValues.
Kolmogorov_D),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.Weibull.pValues.
Kolmogorov_D),'$} \ \ \cline{3-11}
];
tobuild{033} = ['& & Kuiper & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.Exponential.pValues.
Cramer_von_Mises),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.MittagLeffler.pValues.
Cramer_von_Mises),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.GenPareto.pValues.
Cramer_von_Mises),'$} & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.Weibull.pValues.
Cramer_von_Mises),'$} \ \ \cline{3-11}
];
tobuild{034} = ['& & Kuiper & multicolumn{2}{c||}{$',
num2matlabstr(FitTool.Exponential.pValues.Kuiper)
\ & num2matlabstr(FitTool.Exponential.pValues.Kuiper)
\ & num2matlabstr(FitTool.Exponential.pValues.Kuiper)
\ & num2matlabstr(FitTool.Exponential.pValues.Kuiper)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>MLE</th>
<th>Stability</th>
<th>Shape</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponential</td>
<td>MLE.Exponential.Parameters.Scale</td>
<td>MLE.Exponential.Parameters.Rate</td>
<td>MLE.Exponential.Parameters.X</td>
<td>MLE.Exponential.Parameters.X</td>
</tr>
</tbody>
</table>
Weibull.Parameters.Scale),'\$
\cline{6-11}'
};
tobuild{040} = ['& \multicolumn{2}{|c||}{\multirow{2}{*}{Scale}} & \multicolumn{2}{|c||}{\multirow{2}{*}{\$X\$}} & Scale & \$X\$ & \multicolumn{2}{|c||}{\multirow{2}{*}{Shape}} & \multicolumn{2}{|c||}{\multirow{2}{*}{\$\mathcal{X}\$}},num2matlabstr(MLE.Weibull.Parameters.
Shape),'\$
\cline{8-9}'
];
tobuild{041} = ['& \multicolumn{2}{|c||}{\multirow{2}{*}{Loc.}} & \multicolumn{2}{|c||}{\multirow{2}{*}{\$\mathcal{X}\$}} & \multicolumn{2}{|c||}{\multirow{2}{*}{Scale}} & \multicolumn{2}{|c||}{\multirow{2}{*}{\$\mathcal{X}\$}} & \multicolumn{2}{|c||}{\multirow{2}{*}{Shape}} & \multicolumn{2}{|c||}{\multirow{2}{*}{\$\mathcal{X}\$}} \n\cline{10-21}'
];
tobuild{042} = ['& \parbox[t]{2 mm}{\rotatebox{90}{Statistics}} & kol D & \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}} & \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}} & \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}} & \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}} \n\cline{13-24}'
];
tobuild{043} = ['& \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}},num2matlabstr(MLE.Exponential.Statistics.
Cramer_von_Mises),'\$'} & \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}},num2matlabstr(MLE.
Weibull.Statistics.Kolmogorov_D),'\$'} & \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}},num2matlabstr(MLE.
Weibull.Statistics.Kolmogorov_D),'\$'} \n\cline{16-27}'
];
tobuild{044} = ['& \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}},num2matlabstr(MLE.Exponential.Statistics.
Kullback_Leibler),'\$'} & \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}},num2matlabstr(MLE.
Weibull.Statistics.Kullback_Leibler),'\$'} \n\cline{19-30}'
];
tobuild{045} = ['& \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}},num2matlabstr(MLE.Exponential.Statistics.
Jenkinson & Shaw),'\$'} & \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}},num2matlabstr(MLE.
Weibull.Statistics.Jenkinson & Shaw),'\$'} \n\cline{22-33}'
];
tobuild{046} = ['& \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}},num2matlabstr(MLE.Exponential.Statistics.
Anderson-Darling),'\$'} & \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}},num2matlabstr(MLE.
Weibull.Statistics.Anderson-Darling),'\$'} \n\cline{25-36}'
];
tobuild{047} = ['& \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}},num2matlabstr(MLE.Exponential.Statistics.
Kullback-Leibler),'\$'} & \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}},num2matlabstr(MLE.
Weibull.Statistics.Kullback-Leibler),'\$'} \n\cline{28-39}'
];
tobuild{048} = ['& \multicolumn{2}{|c||}{\multicolumn{2}{c||}{\$\mathcal{X}\$}},num2matlabstr(MLE.Exponential.Statistics.
Kolmogorov-D})', '}'
\linebreak
<table>
<thead>
<tr>
<th>p- Values</th>
<th>Kol D</th>
<th>Cramer_von_Mises</th>
<th>Kuiper</th>
<th>Watson</th>
<th>And - Dar</th>
<th>Kul - Lei</th>
<th>Jen - Sha</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
</tr>
</tbody>
</table>

num2matlabstr(MLE. Exponential.pValues.Kolmogorov_D)

num2matlabstr(MLE. Weibull.pValues.Kolmogorov_D)

num2matlabstr(MLE. Weibull.pValues.Kolmogorov_D)

num2matlabstr(MLE. Weibull.pValues.Kolmogorov_D)
```matlab
end

fileID = fopen(filepath,'w');
fprintf(fileID,'%s\r\n',tobuild{:});
fclose(fileID);
end

B.45 makeGraphs.m

function [stats] = makeGraphs(gen_data,real_data,variable,dir_ref,slice)

% MAKEGRAPHS takes two data sets and plots individual and combined data sets
% and returns statistical distances between these
%
% GEN_DATA is the first data set
% REAL_DATA is the second data set
% VARIABLE is the name of the variable being compared
% DIR_REF is the save directory
% SLICE is the bin width when the data is grouped
% STATS is a structure containing the Kolmogorov-Smirnov and Jensen-Shannon
% distances

modelName = strrep(variable,'.','');
modelName = strrep(modelName,'-','');
gen_fields = fieldnames(gen_data);
real_fields = fieldnames(real_data);

plotallthethings = figure();
hold on
for i = 1:numel(gen_fields)
    thisdata = gen_data.(gen_fields{i});
    [F,X] = ecdf(thisdata);
    ccdf = 1-F;
    plot(X,ccdf,'x');
end
for i = 1:numel(real_fields)
    thisdata = real_data.(real_fields{i});
    [F,X] = ecdf(thisdata);
    ccdf = 1-F;
    plot(X,ccdf,':');
end
set(gca,'XScale','log');
set(gca,'YScale','log');
xlabel(variable);```
ylabel('CCDF');
imagefilename1 = [dir_ref,'/','cleanName','_indiv.png'];
figurefilename1 = [dir_ref,'/','cleanName','_indiv'];
print(imagefilename1,'-dpng')
savefig(figurefilename1);
hold off
close(plotallthethings);
gen_all = [];
real_all = [];
for i = 1:numel(gen_fields)
    thisdata = gen_data.(gen_fields{i});
    gen_all = [gen_all,thisdata];
end
for i = 1:numel(real_fields)
    thisdata = real_data.(real_fields{i});
    real_all = [real_all,thisdata];
end
plotsomeofthethings = figure();
hold on
[Fg,Xg] = ecdf(gen_all);
ccdfg = 1-Fg;
errnegg = zeros(1,length(Xg));
errposg = zeros(1,length(Xg));
for i = 1:numel(gen_fields)
    thisdata = gen_data.(gen_fields{i});
    [F,X] = ecdf(thisdata);
    ccdf = 1-F;
    diffvec = zeros(1,length(Xg));
    parfor j=1:length(Xg)
        idx = find(X==Xg(j),1);
        if ~isempty(idx)
            diffvec(j) = ccdf(idx)-ccdfg(j);
        end
    end
    errnegg = min(errnegg,diffvec);
    errposg = min(errposg,diffvec);
end
zerog = zeros(1,length(Xg));
errorbar(Xg,ccdfg,errnegg,errposg,zerog,zerog,'o');
[Fr,Xr] = ecdf(real_all);
ccdfr = 1-Fr;
errnegr = zeros(1,length(Xr));
errposr = zeros(1,length(Xr));
for i = 1:numel(gen_fields)
    thisdata = real_data.(real_fields{i});
    [F,X] = ecdf(thisdata);
ccdf = 1-F;
diffvec = zeros(1,length(Xr));
parfor j=1:length(Xr)
    idx = find(X==Xr(j),1);
    if ~isempty(idx)
        diffvec(j) = ccdf(idx)-ccdfr(j);
    end
end
errnegr = min(errnegr,diffvec);
errposr = min(errposr,diffvec);
zeror = zeros(1,length(Xr));
errorbar(Xr,ccdfr,errnegr,errposr,zeror,zeror,'x');
set(gca,'XScale','log');
set(gca,'YScale','log');
xlabel(variable);
ylabel('CCDF');
imagefilename2 = [dir_ref,'/','cleanName','_combine.png'];
figurefilename2 = [dir_ref,'/','cleanName','_combine'];
print(imagefilename2,'-dpng')
savefig(figurefilename2);
hold off
close(plotsomeofthethings);
stats = stats_KLJS(gen_all,real_all,slice);
end

function [datadump] = manymodels(runs)
% MANYMODELS generates many batches of simulated data and gives average
% acceptance rates of the null hypothesis of the two-sample
% Kolmogorov-Smirnov test at the 5 percent level
% RUNS is the number of runs to be conducted
% DATADUMP is a structure containing all metric data extracted from the
% generated samples
runtime = 20000;

nodesvec = [22,23,25,25,22,23,25,26,23,
            23,21,21,21,21,22,22,22,21,23,23];
EAPfixed.run1 = EAP_matrix('Primary','s12879-014-0695-9-s1-1A-T1.csv');
EAPfixed.run2 = EAP_matrix('Primary','s12879-014-0695-9-s1-1A-T2.csv');
EAPfixed.run3 = EAP_matrix('Primary','s12879-014-0695-9-s1-1B-T1.csv');
EAPfixed.run4 = EAP_matrix('Primary','s12879-014-0695-9-s1-1B-T2.csv');
EAPfixed.run5 = EAP_matrix('Primary','s12879-014-0695-9-s1-2A-T1.csv');
EAPfixed.run6 = EAP_matrix('Primary','s12879-014-0695-9-s1-2A-T2.csv');
EAPfixed.run7 = EAP_matrix('Primary','s12879-014-0695-9-s1-2B-T1.csv');
EAPfixed.run8 = EAP_matrix('Primary','s12879-014-0695-9-s1-2B-T2.csv');
EAPfixed.run9 = EAP_matrix('Primary','s12879-014-0695-9-s1-3A-T1.csv');
EAPfixed.run10 = EAP_matrix('Primary','s12879-014-0695-9-s1-3A-T2.csv');
EAPfixed.run11 = EAP_matrix('Primary','s12879-014-0695-9-s1-3B-T1.csv');
EAPfixed.run12 = EAP_matrix('Primary','s12879-014-0695-9-s1-3B-T2.csv');
EAPfixed.run13 = EAP_matrix('Primary','s12879-014-0695-9-s1-4A-T1.csv');
EAPfixed.run14 = EAP_matrix('Primary','s12879-014-0695-9-s1-4A-T2.csv');
EAPfixed.run15 = EAP_matrix('Primary','s12879-014-0695-9-s1-4B-T1.csv');
EAPfixed.run16 = EAP_matrix('Primary','s12879-014-0695-9-s1-4B-T2.csv');
EAPfixed.run17 = EAP_matrix('Primary','s12879-014-0695-9-s1-5A-T1.csv');
EAPfixed.run18 = EAP_matrix('Primary','s12879-014-0695-9-s1-5A-T2.csv');
EAPfixed.run19 = EAP_matrix('Primary','s12879-014-0695-9-s1-5B-T1.csv');
EAPfixed.run20 = EAP_matrix('Primary','s12879-014-0695-9-s1-5B-T2.csv');

ActiveLinks = zeros(4,runs);
OnTimes = zeros(4,runs);
ActivityPot = zeros(4,runs);
OffTimes = zeros(4,runs);
ActiveNodes = zeros(4,runs);
CompCount = zeros(4,runs);
GCC = zeros(4,runs);
CompNodes = zeros(4,runs);
CompEdges = zeros(4,runs);
TriangleCount = zeros(4,runs);

for i=1:runs
    EAPrandom.run1 = rndLPM(nodesvec(1));
    EAPrandom.run2 = rndLPM(nodesvec(2));
    EAPrandom.run3 = rndLPM(nodesvec(3));
    EAPrandom.run4 = rndLPM(nodesvec(4));
    EAPrandom.run5 = rndLPM(nodesvec(5));
    EAPrandom.run6 = rndLPM(nodesvec(6));
    EAPrandom.run7 = rndLPM(nodesvec(7));
    EAPrandom.run8 = rndLPM(nodesvec(8));
    EAPrandom.run9 = rndLPM(nodesvec(9));
    EAPrandom.run10 = rndLPM(nodesvec(10));
    EAPrandom.run11 = rndLPM(nodesvec(11));
    EAPrandom.run12 = rndLPM(nodesvec(12));
    EAPrandom.run13 = rndLPM(nodesvec(13));
    EAPrandom.run14 = rndLPM(nodesvec(14));
    EAPrandom.run15 = rndLPM(nodesvec(15));
    EAPrandom.run16 = rndLPM(nodesvec(16));
    EAPrandom.run17 = rndLPM(nodesvec(17));
    EAPrandom.run18 = rndLPM(nodesvec(18));
    EAPrandom.run19 = rndLPM(nodesvec(19));
    EAPrandom.run20 = rndLPM(nodesvec(20));

m1_ref = modelv1_multiple(nodesvec, runtime);
m2a_ref = modelv2a_1_multiple(runtime, EAPfixed);
m2b_ref = modelv2_1_multiple(runtime, EAPfixed);
m2c_ref = modelv2_1_multiple(runtime, EAPRandom);

dist1 = calculateDistance(m1_ref, 'Primary');
dist2a = calculateDistance(m2a_ref, 'Primary');
dist2b = calculateDistance(m2b_ref, 'Primary');
dist2c = calculateDistance(m2c_ref, 'Primary');

ActiveLinks(1,i) = dist1.ActiveLinks.accept5;
ActiveLinks(2,i) = dist2a.ActiveLinks.accept5;
ActiveLinks(3,i) = dist2b.ActiveLinks.accept5;
ActiveLinks(4,i) = dist2c.ActiveLinks.accept5;

OnTimes(1,i) = dist1.OnTimes.accept5;
OnTimes(2,i) = dist2a.OnTimes.accept5;
OnTimes(3,i) = dist2b.OnTimes.accept5;
OnTimes(4,i) = dist2c.OnTimes.accept5;

ActivityPot(1,i) = dist1.ActivityPot.accept5;
ActivityPot(2,i) = dist2a.ActivityPot.accept5;
ActivityPot(3,i) = dist2b.ActivityPot.accept5;
ActivityPot(4,i) = dist2c.ActivityPot.accept5;

OffTimes(1,i) = dist1.OffTimes.accept5;
OffTimes(2,i) = dist2a.OffTimes.accept5;
OffTimes(3,i) = dist2b.OffTimes.accept5;
OffTimes(4,i) = dist2c.OffTimes.accept5;

ActiveNodes(1,i) = dist1.ActiveNodes.accept5;
ActiveNodes(2,i) = dist2a.ActiveNodes.accept5;
ActiveNodes(3,i) = dist2b.ActiveNodes.accept5;
ActiveNodes(4,i) = dist2c.ActiveNodes.accept5;

CompCount(1,i) = dist1.CompCount.accept5;
CompCount(2,i) = dist2a.CompCount.accept5;
CompCount(3,i) = dist2b.CompCount.accept5;
CompCount(4,i) = dist2c.CompCount.accept5;

GCC(1,i) = dist1.GCC.accept5;
GCC(2,i) = dist2a.GCC.accept5;
GCC(3,i) = dist2b.GCC.accept5;
GCC(4,i) = dist2c.GCC.accept5;

CompNodes(1,i) = dist1.CompNodes.accept5;
CompNodes(2,i) = dist2a.CompNodes.accept5;
CompNodes(3,i) = dist2b.CompNodes.accept5;
CompNodes(4,i) = dist2c.CompNodes.accept5;

CompEdges(1,i) = dist1.CompEdges.accept5;
CompEdges(2,i) = dist2a.CompEdges.accept5;
CompEdges(3,i) = dist2b.CompEdges.accept5;
CompEdges(4,i) = dist2c.CompEdges.accept5;

TriangleCount(1,i) = dist1.TriangleCount.accept5;
TriangleCount(2,i) = dist2a.TriangleCount.accept5;
TriangleCount(3,i) = dist2b.TriangleCount.accept5;
TriangleCount(4,i) = dist2c.TriangleCount.accept5;

end

datadump.ActiveLinks = ActiveLinks;
datadump.OnTimes = OnTimes;
datadump.ActivityPot = ActivityPot;
datadump.OffTimes = OffTimes;
datadump.ActiveNodes = ActiveNodes;
datadump.CompCount = CompCount;
datadump.GCC = GCC;
datadump.CompNodes = CompNodes;
datadump.CompEdges = CompEdges;
datadump.TriangleCount = TriangleCount;
averages.ActiveLinks = mean(ActiveLinks,2);
averages.OnTimes = mean(OnTimes,2);
averages.ActivityPot = mean(ActivityPot,2);
averages.OffTimes = mean(OffTimes,2);
averages.ActiveNodes = mean(ActiveNodes,2);
averages.CompCount = mean(CompCount,2);
averages.CompNodes = mean(CompNodes,2);
averages.CompEdges = mean(CompEdges,2);
averages.TriangleCount = mean(TriangleCount,2);

lines = 13;
tobuild = cell(lines,1);
tobuild{01} = '\begin{tabular}{|l|c|c|c|c|} \hline 2-5 \hline';
tobuild{02} = '& \textbf{ Model 1} & \textbf{ Model 2a} & \textbf{ Model 2b} & \textbf{ Model 2c} \\
\hline';
tobuild{03} = ['\textbf{ Active Links} & $',num2str(averages.ActiveLinks(1)),'$ & $',num2str(averages.ActiveLinks(2)),'$ & $',num2str(averages.ActiveLinks(3)),'$ & $',num2str(averages.ActiveLinks(4)),'$
\hline';
tobuild{04} = ['\textbf{ Active Nodes} & $',num2str(averages.ActiveNodes(1)),'$ & $',num2str(averages.ActiveNodes(2)),'$ & $',num2str(averages.ActiveNodes(3)),'$ & $',num2str(averages.ActiveNodes(4)),'$
\hline';
tobuild{05} = ['\textbf{ Node Activity Potential} & $',num2str(averages.ActivityPot(1)),'$ & $',num2str(averages.ActivityPot(2)),'$ & $',num2str(averages.ActivityPot(3)),'$ & $',num2str(averages.ActivityPot(4)),'$
\hline';
tobuild{06} = ['\textbf{ Global Clustering Coefficient} & $',num2str(averages.GCC(1)),'$ & $',num2str(averages.GCC(2)),'$ & $',num2str(averages.GCC(3)),'$ & $',num2str(averages.GCC(4)),'$
\hline';
tobuild{07} = ['\textbf{ Interaction Time} & $',num2str(averages.OnTimes(1)),'$ & $',num2str(averages.OnTimes(2)),'$ & $',num2str(averages.OnTimes(3)),'$ & $',num2str(averages.OnTimes(4)),'$
\hline';
tobuild{08} = ['\textbf{ Time Between Contacts} & $',num2str(averages.OffTimes(1)),'$ & $',num2str(averages.OffTimes(2)),'$ & $',num2str(averages.OffTimes(3)),'$ & $',num2str(averages.OffTimes(4)),'$ \hline']
OffTimes(4),'
\ hline'];
tobuild{09} = ['\multicolumn{1}{l}{\textbf{Component Count}} & $',num2str(averages.CompCount(1)),'$ & $',
num2str(averages.CompCount(2)),'$ & $',num2str(
averages.CompCount(3)),'$ & $',num2str(averages.
CompCount(4)),'$
\ hline'];
tobuild{10} = ['\multicolumn{1}{l}{\textbf{Links per Component}} & $',num2str(averages.CompEdges(1)),'$ & $'
,num2str(averages.CompEdges(2)),'$ & $',num2str(
averages.CompEdges(3)),'$ & $',num2str(averages.
CompEdges(4)),'$
\ hline'];
tobuild{11} = ['\multicolumn{1}{l}{\textbf{Nodes per Component}} & $',num2str(averages.CompNodes(1)),'$ & $'
,num2str(averages.CompNodes(2)),'$ & $',num2str(
averages.CompNodes(3)),'$ & $',num2str(averages.
CompNodes(4)),'$
\ hline'];
tobuild{12} = ['\multicolumn{1}{l}{\textbf{Triangle Count}} & $',num2str(averages.TriangleCount(1)),'$ & $'
,num2str(averages.TriangleCount(2)),'$ & $',num2str(
averages.TriangleCount(3)),'$ & $',num2str(averages.
TriangleCount(4)),'$
\ hline'];
tobuild{13} = '\end{tabular}';

168 timestamp = datestr(now,'yyyymmddTHHMMSS');
dir_ref = ['output_','timestamp'];
mkdir(dir_ref);

172 filepath = [dir_ref,'/averageaccept.txt'];

174 fileID = fopen(filepath,'w');
fprintf(fileID,'%s\r\n',tobuild{:});
fclose(fileID);

177 end

B.47 MCMCestimatesig_mlf2.m

function sig = MCMCestimatesig_mlf2(pararange, data, 
MCMClength, parchains, priors, tarAR)

% MCMCestimatesig_mlf2 estimates the best MCMC step variation
% PARARANGE is a 2x2 matrix containing the search range for mu and tau0
% DATA is the given data sample vector
% MCMCLENGTH is the number of MCMC steps to use in the estimation
% PARCHAINS is the number of parallel chains to run simultaneously
% PRIORS is a 1x2 structure containing distribution objects for the priors
% TARAR is a 1x2 vector containing the minimum and maximum acceptance rates
%
% SIG is the returned estimation

paracount = 2;
bsig = [0,range(pararange(1,:));0,range(pararange(2,:))];
sig = zeros(1,paracount);

for i = 1:paracount
  lowsig = bsig(i,1);
  higsig = bsig(i,2);
  AR = 0;
  while AR < tarAR(1) || AR > tarAR(2)
    testsig = zeros(1,paracount);
    testsig(i) = lowsig + abs(higsig - lowsig)/2;
    chains = MCMCmultiCW_mlf2(pararange, data, MCMClength, testsig, parchains, priors);
    pararefs = fieldnames(chains);
    AR = sum(reshape(logical(diff(chains.(pararefs{i}))),1,[]))/numel(logical(diff(chains.(pararefs{i}'))));
    if AR < tarAR(1)
      higsig = testsig(i);
    elseif AR > tarAR(2)
      lowsig = testsig(i);
    end
  end
  sig(i) = testsig(i);
end

B.48 MCMCmultiCW_mlf2.m

function chains = MCMCmultiCW_mlf2(pararange, data, MCMClength, sig, parchains, priors)
%MCMC_MULTI_CW_MLF2 creates the multidimensional distribution posterior
%using the MCMC method
% % PARARANGE is a 2x2 matrix containing the search range for mu and tau0
% DATA is the given data sample vector
% MCMCLENGTH is the number of MCMC steps
% SIG is a 1x2 vector containing the variation in the MCMC steps
% PARCHAINS is the number of parallel chains to run simultaneously
% PRIORS is a 1x2 structure containing distribution objects for the priors
% CHAINS is a 1x2 structure containing MCMCLENGTHxPARCHAINS vectors
% containing the MCMC value for each parameter at each step

paracount = 2;

priordist_p1 = priors.p1;
priordist_p2 = priors.p2;

pr1 = pararange(:,1);
pr2 = pararange(:,2);

fullchains_p1 = zeros(parchains,MCMClength);
fullchains_p2 = zeros(parchains,MCMClength);

parfor i = 1:parchains
    currparas = unifrnd(pr1,pr2);
    for j = 1:MCMClength
        fullchains_p1(i,j) = currparas(1);
        fullchains_p2(i,j) = currparas(2);
        scp = currparas;
        for k = 1:paracount
            if sig(k)>0
                spp = scp;
                jd = makedist('Normal','mu',scp(k),'sigma',sig(k));
                tj = truncate(jd,pr1(k),pr2(k))
                spp(k) = random(tj);
            end
            indivterms = vpa(zeros(2,length(data)));
            for m = 1:length(data)
                indivterms(:,m) = [-(1/data(m))*ml(-(data(m)/spp(2))ˆspp(1),spp(1),0);
                                  -(1/data(m))*ml(-(data(m)/scp(2))ˆscp(1),scp(1),0)];
            end
            prob = prod(indivterms,2);
            scaling = vpa(zeros(paracount,1));
            for m = 1:paracount
                if sig(m)>0
cp = makedist('Normal','mu',scp(m), 'sigma', sig(m));
tcp = truncate(cp,pr1(m),pr2(m));
pc = makedist('Normal','mu',spp(m), 'sigma', sig(m));
tpc = truncate(pc,pr1(m),pr2(m));
probcp = pdf(tcp,spp(m));
probpc = pdf(tpc,scp(m));
scaling(m) = probpc/probcp;
else
  scaling(m) = 1;
end
P_prop = prob(1)*pdf(priordist_p1,spp(1)) *pdf(priordist_p2,spp(2));
P_curr = prob(2)*pdf(priordist_p1,scp(1)) *pdf(priordist_p2,scp(2));
P_move = (P_prop/P_curr)*prod(scaling);
p_move = min([1 double(P_move)]);
if cv<p_move
  scp = spp;
end
currparas = scp;
end
chains.p1 = fullchains_p1;
chains.p2 = fullchains_p2;

B.49 MCMCSP_mlf2.m

function [soln_MCMC,soln_SP] = MCMCSP_mlf2(data,dataname, oF,varargin)
%MCMCSP_MLFS produces several graphical visualisations and Bayesian
%estimates using MCMC estimation and exact methods.
%Specifics of methods
% and priors can be set in SETTINGS and PRIORS in code below.
%DATA is the given data sample vector
%DATANAME is the name for the data sample
%OF is the save folder for outputs
%VARGIN is an optional input, if from generated data this
  is true values of
%each parameter

%SOLN_MCMC is a structure containing the MCMC estimate
  and standard
%deviation of this for each parameter
%SOLN_SP is a structure containing the exact estimate and
  standard
%deviation of this for each parameter

%=== SETTINGS ===%
rangeMu = [0.01, 0.99]; %Search range for mu
tauWindow = 0.1; %Extent tau0 from by
  this proportion
eMCMC_length = 100; %MCMC chain length
  for sig estimation
tarAR = [0.39, 0.49]; %Target acceptance
  rate
MCMC_length = 3000; %MCMC chain length
  for Bayesian estimate
MCMCburn = 1500; %MCMC initialisation
  amount
SPpoints = [300, 300]; %Points in each
  dimension for exact method
paracount = 2; %Parameter count
varstr = {'\mu', '\tau_0'}; %Parameter names
%=== SETTINGS ===%
oF_eps = [oF, '/eps'];
mkdir(oF);
mkdir(oF_eps);
save([oF, '/', dataname, '_data.mat'], 'data');
parchains = max(2, maxNumCompThreads);

[F,X] = ecdf(data);
G = 1-F;
idx = find(exp(-1)<G & G<0.5);
lb = X(idx(1)-1);
ub = X(idx(end)+1);
rangeTau = [(1-tauWindow)*lb,(1+tauWindow)*ub];
pararange = [rangeMu; rangeTau];

%===PRIORS===%
priors.p1 = makedist('Uniform','lower',pararange(1,1),'
  upper',pararange(1,2)); %Prior for first parameter
priors.p2 = makedist('Uniform','lower',pararange(2,1),'upper',pararange(2,2));  % Prior for second parameter

sig = MCMCestimatesig_mlf2(pararange,data,eMCMClength,parchains,priors,tarAR);

chains = MCMCmultiCW_mlf2(pararange,data,MCMClength,sig,parchains,priors);
[X,P] = scaledposterior2_mlf2(pararange,data,SPpoints,priors);
[marginals,SPmean,SPstd] = scaledmarginals(P,X);

bchain1 = chains.p1(:,MCMCburn+1:end);
bchain2 = chains.p2(:,MCMCburn+1:end);
bmchain.p1 = reshape(bchain1,1,[]);
bmchain.p2 = reshape(bchain2,1,[]);
MCMCmean = [mean(bchain1(:)),mean(bchain2(:))];
MCMCstd = [std(bchain1(:)),std(bchain2(:))];

g1 = ceil(sqrt(paracount));
g2 = ceil(paracount/g1);

pararefs = fieldnames(chains);

nf1 = figure('DefaultAxesFontSize',18);
for i=1:paracount
    hax = subplot(g1,g2,i);
    thischains = chains.(pararefs{i});
    hold on
    for j=1:parchains
        plot(thischains(j,:));
    end
    axis manual
    axis([0 MCMClength pararange(i,1) pararange(i,2)]);
    line(get(hax,'XLim'),[MCMCmean(i) MCMCmean(i)],'Color',[1 0 0],'LineWidth',1.5);
    line(get(hax,'XLim'),[MCMCmean(i)+MCMCstd(i) MCMCmean(i)+MCMCstd(i)],'Color',[1 0 0],'LineStyle','-','LineWidth',1.5);
    line(get(hax,'XLim'),[MCMCmean(i)-MCMCstd(i) MCMCmean(i)-MCMCstd(i)],'Color',[1 0 0],'LineStyle','-','LineWidth',1.5);
    h = fill([0 0 MCMCBurn MCMCBurn],[get(hax,'YLim') fliplr(get(hax,'YLim'))],'r','EdgeColor','none');
    set(h,'FaceAlpha',0.25);
end
if nargin==paracount+3
line(get(hax,'XLim'),[varargin{i} varargin{i}],'Color',[0 0 0],'LineWidth',1.5);
end

title(['Markov Chains for ',varstr{i}]);
hold off
end

nf1.WindowState = 'maximized';
saveas(nf1,[oF,'/_quotesingle.ts1',dataname,'_chains.fig']);
saveas(nf1,[oF_eps,'/_quotesingle.ts1',dataname,'_chains'],'epsc');
close(nf1);

for i=1:paracount
    hax = subplot(g1,g2,i);
    hold on
    if paracount==2
        h = histogram(bmchain.(pararefs{i}),'Normalization','pdf');
        histdata(1,:) = h.BinEdges(2:end)-h.BinWidth;
        histdata(2,:) = h.Values;
        hdata.(pararefs{i}) = histdata;
        clear histdata
    else
        histogram(bmchain.(pararefs{i}),'Normalization','pdf');
    end
    plot(X.(pararefs{i}), marginals.(pararefs{i}),'LineWidth',1.5);
    plot(X.(pararefs{i}), pdf(priors.(pararefs{i}),X.(pararefs{i})),X.(pararefs{i}));
    axis manual
    xlim([pararange(i,1) pararange(i,2)]);
    h1 = fill([MCMCmean(i)-MCMCstd(i) MCMCmean(i)+MCMCStd(i) MCMCmean(i)+MCMCstd(i) MCMCmean(i)-MCMCstd(i)],[get(hax,'YLim') fliplr(get(hax,'YLim'))],'r','EdgeColor','none');
    set(h1,'FaceAlpha',0.25);
    line([MCMCmean(i) MCMCmean(i)],get(hax,'YLim'),'Color','r','LineWidth',1.5);
    h2 = fill([SPmean(i)-SPstd(i) SPmean(i)+SPstd(i) SPmean(i)+SPstd(i) SPmean(i)-SPstd(i)],[get(hax,'YLim') fliplr(get(hax,'YLim'))],'b','EdgeColor','none');
    set(h2,'FaceAlpha',0.25);
    line([SPmean(i) SPmean(i)],get(hax,'YLim'),'Color','b','LineWidth',1.5);
end
if nargin==paracount+3
```
line([varargin{i} varargin{i}]){get(hax,'YLim'),'
  Color',[0 0 0],'LineWidth',1.5);
end
title(['Estimated Densities for ',varstr{i}]);
hold off
end

nf2.WindowState = 'maximized';
saveas(nf2,[oF,'/',dataname,'_densities.fig']);
saveas(nf2,[oF_eps,'/',dataname,'_densities'],'epsc');
close(nf2);

if paracount==2
  h1 = hdata.(pararefs{1});
  h2 = hdata.(pararefs{2});
  x1 = h1(1,:);
  y1 = h1(2,:);
  x2 = h2(1,:);
  y2 = h2(2,:);
  y = y2.*y1;
  bottom = min(min(min(double(P))),min(min(y)));
  top = max(max(max(double(P))),max(max(y)));
  nf3 = figure('DefaultAxesFontSize',18);
  subplot(1,2,1);
  h = surf(X.(pararefs{2}),X.(pararefs{1}),double(P));
  set(h,'edgecolor','none');
  hold on
  line([SPmean(2) SPmean(2)],pararange(1,1) pararange(1,2),[top top],'color','b','linewidth',1.5);
  line([pararange(2,1) pararange(2,2)],[SPmean(1) SPmean(1)],[top top],'color','b','linewidth',1.5);
  if nargin==paracount+3
    line([varargin{2} varargin{2}],[pararange(1,1) pararange(1,2)],pararange(1,2),[top top],'color',[0 0 0],'linewidth',1.5);
    line([pararange(2,1) pararange(2,2)],[varargin{1} varargin{1}],[varargin{1}],[top top],'color',[0 0 0],'linewidth',1.5);
  end
caxis manual
caxis([bottom top]);
title('Multidimensional Density Function (Exact Method)')
xlim([pararange(2,1) pararange(2,2)]);
ylim([pararange(1,1) pararange(1,2)]);
view(0,90);
```
xlabel(varstr{2});
ylabel(varstr{1});

subplot(1,2,2);
h = surf(x2,x1,y);
set(h,'edgecolor','none');
hold on
h = surf([pararange(2,1)pararange(2,2)],[pararange(1,1)pararange(1,2)],[[0,0];[0,0]]);
set(h,'edgecolor','none');
hold on
line([MCMCmean(2)MCMCmean(2)],[pararange(1,1)
pararange(1,2)],[top top],'color','r','linewidth',1.5);
line([pararange(2,1)pararange(2,2)],[MCMCmean(1)
MCMCmean(1)],[top top],'color','r','linewidth',1.5);
if nargin==paracount+3
    line([varargin{2}varargin{2}],[pararange(1,1)
pararange(1,2)],[top top],'color',[0 0 0],'linewidth',1.5);
    line([pararange(2,1)pararange(2,2)],[varargin{1}
varargin{1}],[top top],'color',[0 0 0],'linewidth',1.5);
end
caxis manual
caxis([bottom top]);
title('Multidimensional Density Function (Monte Carlo Method)');
xlim([pararange(2,1)pararange(2,2)]);
ylim([pararange(1,1)pararange(1,2)]);
view(0,90);
xlabel(varstr{2});
ylabel(varstr{1});
nf3.WindowState = 'maximized';
saveas(nf3,[oF,'/dataname,_mdf.fig']);
saveas(nf3,[oF_eps,'/dataname,_mdf.epsc']);
close(nf3);
end

[F,X] = ecdf(data);

MCMC1 = ml(-(X/(MCMCmean(2)+(2*MCMCstd(2)))) .ˆ( MCMCmean(1)-(2*MCMCstd(1)) ), MCMCmean(1)-(2*MCMCstd(1)));
MCMC2 = ml(-(X/(MCMCmean(2)+(1*MCMCstd(2)))) .ˆ( MCMCmean(1)-(1*MCMCstd(1)) ), MCMCmean(1)-(1*MCMCstd(1)));
MCMC3 = ml(-(X/MCMCmean(2)) .ˆ MCMCmean(1), MCMCmean(1));
MCMC4 = ml(-(X/(MCMCmean(2)-(1*MCMCstd(2))))).*(MCMCmean(1)+(1*MCMCstd(1))),MCMCmean(1)+(1*MCMCstd(1)));
MCMC5 = ml(-(X/(MCMCmean(2)-(2*MCMCstd(2))))).*(MCMCmean(1)+(2*MCMCstd(1))),MCMCmean(1)+(2*MCMCstd(1)));

SP1 = ml(-(X/(SPmean(2)+(2*SPstd(2))))).*(SPmean(1)-(2*SPstd(1))),SPmean(1)-(2*SPstd(1)));
SP2 = ml(-(X/(SPmean(2)+(1*SPstd(2))))).*(SPmean(1)-(1*SPstd(1))),SPmean(1)-(1*SPstd(1)));
SP3 = ml(-(X/(SPmean(2))).*SPmean(1),SPmean(1));
SP4 = ml(-(X/(SPmean(2)-(1*SPstd(2))))).*(SPmean(1)+(1*SPstd(1))),SPmean(1)+(1*SPstd(1)));
SP5 = ml(-(X/(SPmean(2)-(2*SPstd(2))))).*(SPmean(1)+(2*SPstd(1))),SPmean(1)+(2*SPstd(1)));

nf4 = figure('DefaultAxesFontSize',18);
loglog(X,1-F,'Marker','x','MarkerSize',12,'LineStyle','none');
hold on
plot(X,MCMC1,'-','Color','r','LineWidth',1.5);
plot(X,MCMC2,'--','Color','r','LineWidth',1.5);
plot(X,MCMC3,'-','Color','r','LineWidth',1.5);
plot(X,MCMC4,'-','Color','r','LineWidth',1.5);
plot(X,MCMC5,'-','Color','r','LineWidth',1.5);
plot(X,SP1,'-','Color','b','LineWidth',1.5);
plot(X,SP2,'-','Color','b','LineWidth',1.5);
plot(X,SP3,'-','Color','b','LineWidth',1.5);
plot(X,SP4,'-','Color','b','LineWidth',1.5);
plot(X,SP5,'-','Color','b','LineWidth',1.5);
if nargin==paracount+3
    TRULINE = ml(-(X/varargin{2})).*varargin{1},varargin{1});
    plot(X,TRULINE,'-','Color',[0 0 0],'LineWidth',1.5);
end
xlabel('Time Between Events');
ylabel('CCDF');
title('Waiting Times')

nf4.WindowState = 'maximized';
saveas(nf4,[oF,'/','dataname,'_ccdf.fig']);
saveas(nf4,[oF_eps,'/','dataname,'_ccdf'],'epsc');
close(nf4);
soln_MCMC.mean = MCMCmean;
soln_MCMC.std = MCMCstd;
soln_SP.mean = SPmean;
soln_SP.std = SPstd;
B.50  ml.m

This function was written by Garrappa [73].

function E = ml(z, alpha, beta, gamma)

% Evaluation of the Mittag-Leffler (ML) function with 1, 2 or 3 parameters
% by means of the OPC algorithm [1]. The routine evaluates an approximation
% Et of the ML function E such that |E-Et|/(1+|E|) approx 1.0e-15
%

% E = ML(z, alpha) evaluates the ML function with one parameter alpha for
% the corresponding elements of z; alpha must be a real and positive
% scalar. The one parameter ML function is defined as

% E = sum_{k=0}^{infty} z^k / Gamma(alpha*k+1)
% with Gamma the Euler's gamma function.

% E = ML(z, alpha, beta) evaluates the ML function with two parameters alpha
% and beta for the corresponding elements of z; alpha must be a real and
% positive scalar and beta a real scalar. The two parameters ML function is
% defined as

% E = sum_{k=0}^{infty} z^k / Gamma(alpha*k+beta)
%
% E = ML(z, alpha, beta, gamma) evaluates the ML function with three parameters
% alpha, beta and gamma for the corresponding elements of z; alpha must be a
% real scalar such that 0<alpha<1, beta any real scalar and gamma a real and
% positive scalar; the arguments z must satisfy |Arg(z)| > alpha*pi. The
% three parameters ML function is defined as
% E = \sum_{k=0}^{\infty} \frac{\Gamma(gama+k) \cdot z^k}{\Gamma(gama) / k!} \cdot \frac{\Gamma(\alpha \cdot k + \beta)}{\Gamma(\alpha \cdot k + \beta)}

% NOTE:
% This routine implements the optimal parabolic contour (OPC) algorithm
% described in [1] and based on the inversion of the Laplace transform on a
% parabolic contour suitably chosen in one of the regions of analyticity
% of the Laplace transform.
%
% REFERENCES
% [1] R. Garrappa, Numerical evaluation of two and three parameter
% Mittag-Leffler functions, SIAM Journal of Numerical Analysis, 2015,
% 53(3), 1350-1369
%
% Please, report any problem or comment to:
% roberto dot garrappa at uniba dot it
%
% Copyright (c) 2015, Roberto Garrappa, University of Bari, Italy
% roberto dot garrappa at uniba dot it
% Homepage: http://www.dm.uniba.it/Members/garrappa
% Revision: 1.4 - Date: October 8 2015

% Check inputs
if nargin < 4
    gama = 1 ;
    if nargin < 3
        beta = 1 ;
        if nargin < 2
            error('MATLAB:ml:NumberParameters', ...
                  'The parameter ALPHA must be specified.');
        end
    end
end

% Check whether the parameters ALPHA, BETA and GAMMA are
scalars

if length(alpha) > 1 || length(beta) > 1 || length(gama) > 1
    alpha = alpha(1); beta = beta(1); gama = gama(1);
    warning('MATLAB:ml:ScalarParameters', ...
        ['ALPHA, BETA and GAMA must be scalar parameters. ', ...
        'Only the first values ALPHA=%f BETA=%f and GAMA =%f will be used. '], ...
        alpha, beta, gama);
end

% Check whether the parameters meet the contraints
if real(alpha) <= 0 || real(gama) <= 0 || ~isreal(alpha) || ...
    ~isreal(beta) || ~isreal(gama)
    error('MATLAB:ml:ParametersOutOfRange', ...
        ['Error in the parameters of the Mittag-Leffler function. ', ...
        'Parameters ALPHA and GAMA must be real and positive. ', ...
        'The parameter BETA must be real.']);
end

% Check parameters and arguments for the three parameter case
if abs(gama-1) > eps
    if alpha > 1
        error('MATLAB:ml:ALPHAOutOfRange',...
            ['With the three parameters Mittag-Leffler function ', ...
            'the parameter ALPHA must satisfy 0 < ALPHA < 1']);
    end
    if min(abs(angle(z(abs(z)>eps)))) <= alpha*pi
        error('MATLAB:ml:ThreeParametersArgument',...
            ['With the three parameters Mittag-Leffler function ', ...
            'this code works only when |Arg(z)|>alpha*pi.']);
    end
end

% Target precision
log_epsilon = log(10^(-15)) ;

% Inversion of the LT for each element of z
E = zeros(size(z)) ;
for k = 1 : length(z)
    if abs(z(k)) < 1.0e-15
        E(k) = 1/gamma(beta);
    else
        E(k) = LTInversion(1,z(k),alpha,beta,gama,
                          log_epsilon);
    end
end

% ==============================================================
% Evaluation of the ML function by Laplace transform
% inversion
% ==============================================================
function E = LTInversion(t,lambda,alpha,beta,gama,
                          log_epsilon)

% Evaluation of the relevant poles
theta = angle(lambda);

kmin = ceil(-alpha/2 - theta/2/pi);
kmax = floor(alpha/2 - theta/2/pi);
k_vett = kmin : kmax;
s_star = abs(lambda)^(1/alpha) * exp(1i*(theta+2*k_vett*pi)/alpha);

% Evaluation of phi(s_star) for each pole
phi_s_star = (real(s_star)+abs(s_star))/2;

% Sorting of the poles according to the value of phi(s_star)
[phi_s_star, index_s_star] = sort(phi_s_star);
s_star = s_star(index_s_star);

% Deleting possible poles with phi_s_star=0
index_save = phi_s_star > 1.0e-15;
s_star = s_star(index_save);
phi_s_star = phi_s_star(index_save);

% Inserting the origin in the set of the singularities
s_star = [0, s_star];
phi_s_star = [0, phi_s_star];
J1 = length(s_star); J = J1 - 1;

% Strength of the singularities
p = [ max(0,-2*(alpha*gama-beta+1)) , ones(1,J)*gama ];
\texttt{q} = \texttt{[ ones(1,\texttt{J})*gama, +Inf]} ;
\texttt{phi_s_star} = \texttt{[phi_s_star, +Inf]} ;

\texttt{\% Looking for the admissible regions with respect to round-off errors}
\texttt{admissible_regions} = \texttt{find( ...}
\texttt{    (phi_s_star(1:end-1) < (log(epsilon) - log(eps))/t) &}
\texttt{    ...}
\texttt{    (phi_s_star(1:end-1) < phi_s_star(2:end))) ;}

\texttt{\% Initializing vectors for optimal parameters}
\texttt{JJ1 = admissible_regions(end) ;}
\texttt{mu_vett = ones(1,\texttt{JJ1})*Inf ;}
\texttt{N_vett = ones(1,\texttt{JJ1})*Inf ;}
\texttt{h_vett = ones(1,\texttt{JJ1})*Inf ;}

\texttt{\% Evaluation of parameters for inversion of LT in each admissible region}
\texttt{find_region} = 0 ;
\texttt{while \sim find_region}
\texttt{    for \texttt{j1} = admissible_regions}
\texttt{        if \texttt{j1} < \texttt{J1}}
\texttt{            [muj, hj, Nj] = OptimalParam_RB ...
\texttt{                (t, phi_s_star(j1), phi_s_star(j1+1), p(j1), q(j1), log(epsilon)) ;}
\texttt{        }\texttt{else}
\texttt{            [muj, hj, Nj] = OptimalParam_RU(t, phi_s_star(j1), p(j1), log(epsilon)) ;}
\texttt{        end}
\texttt{        mu_vett(j1) = muj ; h_vett(j1) = hj ; N_vett(j1) = Nj ;}
\texttt{    end}
\texttt{    if min(N_vett) > 200}
\texttt{        log(epsilon) = log(epsilon) + log(10) ;}
\texttt{    }\texttt{else}
\texttt{        find_region = 1 ;}
\texttt{    end}
\texttt{end}

\texttt{\% Selection of the admissible region for integration which involves the}
\texttt{\% minimum number of nodes}
\texttt{[N, iN] = min(N_vett) ; mu = mu_vett(iN) ; h = h_vett(iN) ;}

\texttt{\% Evaluation of the inverse Laplace transform}
\texttt{k = -N : N ;}
\[ u = h \ast k; \]
\[ z = \mu \ast (1i \ast u + 1) \ast 2; \]
\[ zd = -2 \ast \mu \ast u + 2 \ast \mu \ast 1i; \]
\[ zexp = \exp(z \ast t); \]
\[ F = z \ast (\alpha \ast gama - \beta) \ast (z \ast \alpha - \lambda) \ast gama \ast zd; \]
\[ S = zexp \ast F; \]
\[ Integral = h \ast \text{sum}(S) / 2 / \pi / 1i; \]

\% Evaluation of residues
\[ ss\_star = s\_star(iN+1:end); \]
\[ Residues = \text{sum}(1/\alpha \ast (ss\_star) \ast (1 - \beta) \ast \exp(t \ast ss\_star)); \]

\% Evaluation of the ML function
\[ E = Integral + Residues; \]
\[ \text{if } \text{isreal}(\lambda) \]
\[ \quad E = \text{real}(E); \]
\[ \text{end} \]
\[ \text{end} \]

\% ===================================================================
\% Finding optimal parameters in a right-bounded region
\% ===================================================================

\% Definition of some constants
\[ \log\_eps = -36.043653389117154; \% \log(\varepsilon) \]
\[ \text{fac} = 1.01; \]
\[ \text{conservative\_error\_analysis} = 0; \]

\% Maximum value of fbar as the ratio between tolerance and round-off unit
\[ f\_max = \exp(\text{log\_epsilon} - \log\_eps); \]

\% Evaluation of the starting values for sq\_phi\_star\_j and sq\_phi\_star\_j1
\[ sq\_phi\_star\_j = \text{sqrt}(\phi\_s\_star\_j); \]
\[ \text{threshold} = 2 \ast \text{sqrt}((\log\_epsilon - \log\_eps) / t); \]
\[ sq\_phi\_star\_j1 = \text{min}(\text{sqrt}(\phi\_s\_star\_j1), \text{threshold} - \text{sq\_phi\_star\_j}); \]

\% Zero or negative values of pj and qj
\[ \text{if } pj < 1.0\text{e}-14 \& \& qj < 1.0\text{e}-14 \]
\[ \quad sq\_phibar\_star\_j = sq\_phi\_star\_j; \]
\[ \quad sq\_phibar\_star\_j1 = sq\_phi\_star\_j1; \]
adm_region = 1;
end

% Zero or negative values of just pj
if pj < 1.0e-14 && qj >= 1.0e-14
    sq_phibar_star_j = sq_phi_star_j;
    if sq_phi_star_j > 0
        f_min = fac*(sq_phi_star_j/(sq_phi_star_j1-sq_phi_star_j))ˆqj;
    else
        f_min = fac;
    end
    f_min = fac;
end
if f_min < f_max
    f_bar = f_min + f_min/f_max*(f_max-f_min);
    fp = f_bar^(-1/pj);
    sq_phibar_star_j1 = (2*sq_phi_star_j1-fp*sq_phi_star_j)/(2-fp);
    adm_region = 1;
else
    adm_region = 0;
end

% Zero or negative values of just qj
if pj >= 1.0e-14 && qj < 1.0e-14
    sq_phibar_star_j1 = sq_phi_star_j1;
    f_min = fac*(sq_phi_star_j1/(sq_phi_star_j1-sq_phi_star_j))ˆpj;
    if f_min < f_max
        f_bar = f_min + f_min/f_max*(f_max-f_min);
        fp = f_bar^(-1/pj);
        sq_phibar_star_j = (2*sq_phi_star_j1+fp*sq_phi_star_j1)/(2-fp);
        adm_region = 1;
    else
        adm_region = 0;
    end

% Positive values of both pj and qj
if pj >= 1.0e-14 && qj >= 1.0e-14
    f_min = fac*(sq_phi_star_j+sq_phi_star_j1)/...
             (sq_phi_star_j1-sq_phi_star_j)^max(pj,qj);
    if f_min < f_max
        f_min = max(f_min,1.5);
        f_bar = f_min + f_min/f_max*(f_max-f_min);
        fp = f_bar^(-1/pj);
        fq = f_bar^(-1/qj);
if ~conservative_error_analysis
  w = -phi_s_star_j1*t/log_epsilon ;
else
  w = -2*phi_s_star_j1*t/(log_epsilon-phi_s_star_j1*t) ;
end

den = 2+w - (1+w)*fp + fq ;
sq_phibar_star_j = ((2+w+fq)* sq_phi_star_j + fp* sq_phi_star_j1 )/ den ;
sq_phibar_star_j1 = ( -(1+w)*fq* sq_phi_star_j ... + (2+w-(1+w)*fp)*sq_phi_star_j1)/ den ;
adm_region = 1 ;
else
  adm_region = 0 ;
end

if adm_region
  log_epsilon = log_epsilon - log ( f_bar ) ;
  if ~conservative_error_analysis
    w = -sq_phibar_star_j1^2*t/log_epsilon ;
  else
    w = -2*sq_phibar_star_j1^2*t/(log_epsilon- ... sq_phibar_star_j1^2*t) ;
  end
  muj = (((1+w)* sq_phibar_star_j + sq_phibar_star_j1 )/(2+w))^2 ;
  hj = -2*pi/log_epsilon*(sq_phibar_star_j1- ... sq_phibar_star_j)... /
    (((1+w)*sq_phibar_star_j + sq_phibar_star_j1) ;
  if
    muj = 0 ; hj = 0 ; Nj = +Inf ;
else
  end
end

% ==================================================
% Finding optimal parameters in a right-unbounded region
% ==================================================
function [muj,hj,Nj] = OptimalParam_RU (t, phi_s_star_j , pj, log_epsilon )

% Evaluation of the starting values for sq_phi_star_j
sq_phi_s_star_j = sqrt( phi_s_star_j ) ;
if phi_s_star_j > 0
  phibar_star_j = phi_s_star_j*1.01 ;
else
phibar_star_j = 0.01 ;
end
sq_phibar_star_j = sqrt(phibar_star_j) ;

% Definition of some constants
f_min = 1 ; f_max = 10 ; f_tar = 5 ;

% Iterative process to look for fbar in [f_min,f_max]
stop = 0 ;
while ~stop
  phi_t = phibar_star_j * t ; log_eps_phi_t = log_epsilon / phi_t ;
  Nj = ceil(phi_t / pi * (1 - 3 * log_eps_phi_t / 2 + sqrt(1 - 2 * log_eps_phi_t))) ;
  A = pi * Nj / phi_t ;
  sq_muj = sq_phibar_star_j * abs(4 - A) / abs(7 - sqrt(1 + 12 * A)) ;
  fbar = ((sq_phibar_star_j - sq_phi_s_star_j) / sq_muj) ^ (- pj) ;
  stop = (pj < 1.0e-14) || (f_min < fbar && fbar < f_max) ;
  if ~stop
    sq_phibar_star_j = f_tar ^ (-1/pj) * sq_muj +
    sq_phi_s_star_j ;
    phibar_star_j = sq_phibar_star_j ^ 2 ;
  end
end
muj = sq_muj ^ 2 ;
hj = (-3*A - 2 + 2*sqrt(1+12*A))/(4-A)/Nj ;

% Adjusting integration parameters to keep round-off errors under control
log_eps = log(eps) ; threshold = (log_eepsilon - log_eps) / t ;
if muj > threshold
  if abs(pj) < 1.0e-14 , Q = 0 ; else Q = f_tar ^ (-1/pj) * sqrt(muj) ; end
  phibar_star_j = (Q + sqrt(phi_s_star_j)) ^ 2 ;
  if phibar_star_j < threshold
    w = sqrt(log_eepsilon/(log_eepsilon-log_eepsilon)) ;
    u = sqrt(-phibar_star_j * t / log_eepsilon) ;
    muj = threshold ;
    Nj = ceil(w*log_eepsilon/2/pi/(u*w-1)) ;
    hj = sqrt(log_eepsilon/(log_eepsilon-log_eepsilon))/Nj ;
  else
    Nj = + Inf ; hj = 0 ;
  end
end
This function was written by Podlubny and Kacenak [151].

```matlab
function [e]=mlf(alf,bet,c,fi)
%
% MLF -- Mittag-Leffler function.
% MLF (alpha,beta,Z,P) is the Mittag-Leffler function E_{alpha,beta}(Z)
% evaluated with accuracy 10^{-P} for each element of Z.
% (C) 2001-2005 Igor Podlubny, Martin Kacenak
%
if nargin<4 , fi=6; end
if nargin<3 || alf<=0 || fi<=0
else
[r,s]=size(c); [r1,s1]=size(alf); [r2,s2]=size(bet);
mx=max([r,s]); mx1=max([r1,s1]); mx2=max([r2,s2]);
if (r>1 && s>1) || (r1>1 && s1>1) || (r2>1 && s2>1) ||
  (mx1>1 && mx2>1)
  sprintf('wrong number of input parameters')
else
if mx1>mx2 , mxx=mx1; e=zeros(mx,mx1);
else mxx=mx2; e=zeros(mx,mx2);end;
for i1= 1: mx
  for i2 =1: mxx
    if r>s , z=c(i1,1); else z=c(1,i1); end
    if mx1>mx2 , if r1>s1 , alfa=alf(i2,1); else
      alfa=alf(1,i2);end , beta=bet;
    else if r2>s2 ,beta=bet(i2,1); else beta=bet
      (1,i2); end, alfa=alf; end
    if beta<0 , rc=(-2*log(10^{-fi})*pi/(6*(abs( beta)+2)*(2*abs(bet)^2*(abs(bet)))))^alfa
      ;
    else rc=(-2*log(10^{-fi})*pi/6)^alfa; end
    r0=max([1,2*abs(z),rc]);
    if (alfa==1 && beta==1)
      e(i1,i2)=exp(z);
    else
      if (alfa<1 && abs(z)<=1) || ((1<=alfa &&
      alfa<2) && abs(z)<=floor(20/(2.1-alfa) ^5.5-2*alfa))) || (alfa>=2 && abs(z)
      <=50)
        end
    end
  end
end
```
oldsum=0;
k=0;
while (alfa*k+beta)<=0
    k=k+1;
end
newsum=z^k/gamma(alfa*k+beta);
while newsum~=oldsum
    oldsum=newsum;
k=k+1;
term=z^k/gamma(alfa*k+beta);
newsum=newsum+term;
k=k+1;
term=z^k/gamma(alfa*k+beta);
newsum=newsum+term;
end
e(i1,i2)=newsum;
else
    if (alfa<=1 && abs(z)<=fix(5*alfa+10))
        if ((abs(angle(z))>pi*alfa) && (abs(abs(angle(z))-(pi*alfa))>10^(-fi)))
            if beta<=1
                e(i1,i2)=rombint('K',0,r0,fi,alfa,beta,z);
            else
                eps=1;
                e(i1,i2)=rombint('K',eps,r0,fi,alfa,beta,z)+ ... 
                rombint('P',-pi*alfa,pi*alfa,fi,alfa,beta,z,eps);
            end
        elseif (abs(angle(z))<pi*alfa && abs(abs(angle(z))-(pi*alfa))>10^(-fi))
            if beta<=1
                e(i1,i2)=rombint('K',0,r0,fi,alfa,beta,z)+ ... 
                (z^-((1-beta)/alfa))*(exp(z^-((1/alfa))/alfa);
            else
                eps=abs(z)/2;
                e(i1,i2)=rombint('K',eps,r0,fi,alfa,beta,z)+ ... 
                rombint('P',-pi*alfa,pi*alfa,fi,alfa,beta,z,eps) + ... 
                (z^-((1-beta)/alfa))*(exp(z^-((1/alfa))/alfa);
            end
else
    eps = abs(z) + 0.5;
e(i1,i2) = rombint('K',eps,r0,fi,
    alfa,beta,z) + ... 
    rombint('P',-pi*alfa,pi*alfa,
     fi,alfa,beta,z,eps);
end
else
    if alfa <= 1
        if (abs(angle(z)) < (pi*alfa/2 + min(pi,pi*alfa))/2)
            newsum = (z^((1-beta)/alfa)) * exp
                      (z^(1/alfa))/alfa;
            for k = 1:floor(fi/log10(abs(z))
                newsum = newsum - ((z^(-k))/
                    gamma(beta - alfa*k));
            end
        end
        e(i1,i2) = newsum;
    else
        newsum = 0;
        for k = 1:floor(fi/log10(abs(z))
            newsum = newsum - ((z^(-k))/gamma(beta - alfa*k));
        end
        e(i1,i2) = newsum;
    end
else
    if alfa >= 2
        m = floor(alfa/2);
sum = 0;
        for h = 0:m
            zn = (z^((1/(m+1)))) * exp((2*pi*
i*h)/(m+1));
            sum = sum + mlf(alfa/(m+1),beta,
                            zn,fi);
        end
        e(i1,i2) = (1/(m+1)) * sum;
    else
        e(i1,i2) = (mlf(alfa/2,beta,z
                       ^((1/2),fi)+mlf(alfa/2,beta,
                       -z^((1/2),fi))/2;
    end
end
function [res] = rombint(funfcn,a,b,order,varargin)
if nargin <4,order=6; end
if nargin<3
    Warning ('Error in input format')
else
    rom=zeros(2,order);
    h=b-a;
    rom(1,1)=h*(feval(funfcn,a,varargin{:})+feval(funfcn,b,varargin{:}))/2;
    ipower=1;
    for i=2:order
        sum=0;
        for j=1:ipower
            sum=sum+feval(funfcn,(a+h*(j-0.5)),varargin{:});
        end
        rom(2,1)=(rom(1,1)+h*sum)/2;
        for k=1:i-1
            rom(2,k+1)=((4ˆk)*rom(2,k)-rom(1,k))/((4ˆk)-1);
        end
        for j=0:i-1
            rom(1,j+1)=rom(2,j+1);
        end
        ipower=ipower*2;
        h=h/2;
    end
    res=rom(1,order);
end

function [res] = K(r,alfa,beta,z)
res=r.ˆ((1-beta)/alfa).*exp(-r.ˆ(1/alfa)).*(r*sin(pi*(1-beta))-z*sin(pi*(1-beta+alfa)))/(pi*alfa*(r.ˆ2-2*r*z*cos(pi*alfa)+z.ˆ2));

function [res] = P(r,alfa,beta,z,eps)
w=(epsˆ(1/alfa))*sin(r/alfa)+r*(1+(1-beta)/alfa);
res = ((eps^(1+(1-beta)/alfa))/(2*pi*alfa)) * ((exp((eps^(1/alfa))*cos(r/alfa)).*... 
(cos(w)+1i*sin(w)))/(eps*exp(1i*r)-z));

B.52  mlrnd.m

This function was written by Germano, Fulger and Scalas [76].

function t = mlrnd(beta, gamma_t, m, n)
%
% mlrnd.m: Mittag-Leffler pseudo-random number generator
% Authors: Guido Germano, Daniel Fulger, Enrico Scalas (2006-2008)
% Description: Returns a matrix of iid random numbers distributed according to
% the one-parameter Mittag-Leffler distribution with index (or exponent) beta
% and scale parameter gamma_t. The size of the returned matrix is the same as
% that of the input matrices beta and gamma_t, that must match. Alternatively,
% if beta and gamma_t are scalars, mlrnd(beta, gamma_t, m) returns an m by m
% matrix, and mlrnd(beta, gamma_t, m, n) returns an m by n matrix.
%
% References:
% continuous-time random walks yielding a stochastic solution of the space-
%
% Check input
if nargin < 2
    error('mlrnd requires at least two arguments.')
elseif nargin == 2
    sb = size(beta);
    sg = size(gamma_t);
    if ~isscalar(beta) && ~isscalar(gamma_t) && any(sb ~= sg)
error('mlrnd: size mismatch between beta and gamma_t.')
end
m = max(sb(1),sg(1));
n = max(sb(2),sg(2));
else
    if ~isscalar(beta) && ~isscalar(gamma_t)
        error('mlrnd: beta and gamma_t must be scalars when m is given.')
    end
    if ~(isscalar(m) && m > 0 && m == round(m))
        error('mlrnd: m must be a positive integer.')
    end
    if nargin == 3
        n = m;
    elseif ~(isscalar(n) && n > 0 && n == round(n))
        error('mlrnd: n must be a positive integer.')
    end
end
if any(any(beta <= 0)) || any(any(beta > 1))
    error('mlrnd: the elements of beta must belong to the interval (0,1].')
end
if any(any(gamma_t <= 0))
    error('mlrnd: the elements of gamma_t must be positive.')
end
% Generate m x n Mittag-Leffler pseudo-random numbers
\( t = -\log(\text{rand}(m,n)) \); \% Standard exponential deviate
if (beta < 1)
    u = rand(m,n); \% Uniform deviate on the unit interval, independent of \( t \)
    w = sin(beta*pi)./tan(beta*pi.*u) - cos(beta*pi); \% Auxiliary variable
    t = t.*w.^(1./beta); \% Standard one-parameter Mittag-Leffler deviate
end
\( t = \gamma_t^*t; \)
return

B.53  \texttt{mm\_ExpGamRayLN.m}

\texttt{function \[\text{mins, maxs}\] = mm\_ExpGamRayLN(FT, MLE, Moments)}
%MM\_EXPMLGPEWI returns the minimum and maximum values of each parameter
% FT is the fit tool data structure
% MLE is the most likelihood estimators data structure
% MOMENTS is the method of moments data structure
% MINS is a vector containing the minimum values of each parameter
% MACS is a vector containing the maximum values of each parameter

brokenMins = Inf(1,6);
brokenMaxs = -Inf(1,6);

FT_Vec = zeros(1,6);
MLE_Vec = zeros(1,6);
Moments_Vec = zeros(1,6);

FT_Vec(1) = FT.Exponential.Parameters.Scale;
FT_Vec(2) = FT.Gamma.Parameters.Shape;
FT_Vec(3) = FT.Gamma.Parameters.Scale;
FT_Vec(4) = FT.Rayleigh.Parameters.Scale;
FT_Vec(5) = FT.LogNormal.Parameters.Location;
FT_Vec(6) = FT.LogNormal.Parameters.Scale;

MLE_Vec(1) = MLE.Exponential.Parameters.Scale;
MLE_Vec(2) = MLE.Gamma.Parameters.Shape;
MLE_Vec(3) = MLE.Gamma.Parameters.Scale;
MLE_Vec(4) = MLE.Rayleigh.Parameters.Scale;
MLE_Vec(5) = MLE.LogNormal.Parameters.Location;
MLE_Vec(6) = MLE.LogNormal.Parameters.Scale;

Moments_Vec(1) = Moments.Exponential.Parameters.Scale;
Moments_Vec(2) = Moments.Gamma.Parameters.Shape;
Moments_Vec(3) = Moments.Gamma.Parameters.Scale;
Moments_Vec(4) = Moments.Rayleigh.Parameters.Scale;
Moments_Vec(5) = Moments.LogNormal.Parameters.Location;
Moments_Vec(6) = Moments.LogNormal.Parameters.Scale;

mins = min([brokenMins; FT_Vec; MLE_Vec; Moments_Vec]);
maxs = max([brokenMaxs; FT_Vec; MLE_Vec; Moments_Vec]);

B.54 mm_ExpMLGPWei.m

function [mins,maxs] = mm_ExpMLGPWei(FT,MLE,Moments)
%MM_EXPMLGPWEI returns the minimum and maximum values of each parameter


% FT is the fit tool data structure
% MLE is the most likelihood estimators data structure
% MOMENTS is the method of moments data structure
% MINS is a vector containing the minimum values of each parameter
% MACS is a vector containing the maximum values of each parameter

brokenMins = Inf(1,8);
brokenMaxs = -Inf(1,8);

FT_Vec = zeros(1,8);
MLE_Vec = zeros(1,8);
Moments_Vec = zeros(1,8);

FT_Vec(1) = FT.Exponential.Parameters.Scale;
FT_Vec(2) = FT.MittagLeffler.Parameters.Stability;
FT_Vec(3) = FT.MittagLeffler.Parameters.Scale;
FT_Vec(4) = FT.GenPareto.Parameters.Shape;
FT_Vec(5) = FT.GenPareto.Parameters.Scale;
FT_Vec(6) = FT.GenPareto.Parameters.Location;
FT_Vec(7) = FT.Weibull.Parameters.Scale;
FT_Vec(8) = FT.Weibull.Parameters.Shape;

MLE_Vec(1) = MLE.Exponential.Parameters.Scale;
MLE_Vec(2) = NaN;
MLE_Vec(3) = NaN;
MLE_Vec(4) = NaN;
MLE_Vec(5) = NaN;
MLE_Vec(6) = NaN;
MLE_Vec(7) = MLE.Weibull.Parameters.Scale;
MLE_Vec(8) = MLE.Weibull.Parameters.Shape;

Moments_Vec(1) = Moments.Exponential.Parameters.Scale;
Moments_Vec(2) = NaN;
Moments_Vec(3) = NaN;
Moments_Vec(4) = Moments.GenPareto.Parameters.Shape;
Moments_Vec(5) = Moments.GenPareto.Parameters.Scale;
Moments_Vec(6) = Moments.GenPareto.Parameters.Location;
Moments_Vec(7) = NaN;
Moments_Vec(8) = NaN;

mins = min([brokenMins;FT_Vec;MLE_Vec;Moments_Vec]);
maxs = max([brokenMaxs;FT_Vec;MLE_Vec;Moments_Vec]);
function [] = model(nodes, runtime, filepath)
%MODEL Takes a runtime and a weighted selection matrix
% and generates a
% CSV file sampled every (default 20) seconds using
% method 1.
%
% This is needed to compare simulation with empirical results from these two
% papers: V. Gemmetto et al. Mitigation of infectious diseases at school:
% targeted class closure vs school closure, BMC Infectious Diseases
% 2014:695 https://doi.org/10.1186/s12879-014-0695-9; R.
% Mastrandrea
% et al., Contact Patterns in a High School: A Comparison between Data
% Collected Using Wearable Sensors, Contact Diaries and Friendship Surveys,
% PLoS ONE 10(9): e0136497. https://doi.org/10.1371/
% journal.pone.0136497
%
% NODES specifies the number of nodes present in the network
% RUNTIME specifies the total sampling period of the simulation

cut = 20;
preruntime = zeros(nodes);
switchon = exprnd(7384.5, nodes);
startthings = switchon - preruntime;
initial = zeros(nodes);
ex1_mu = 3.2434;

EXpara1 = lognrnd(ex1_mu, sigma_for_mu_and_mean(30.552, ex1_mu), nodes);
LNpara1 = 6.3512*ones(nodes);
LNpara2 = 1.3688*ones(nodes);

ontimes = struct();
offtimes = struct();

for i=1:nodes-1
for j = i+1: nodes
    init = initial(i,j);
    currenttime = startthings(i,j);
    if init == 0
        if startthings(i,j)< runtime
            thisoff = [startthings(i,j)];
        else
            thisoff = [];
        end
        thison = [];
    end
    while currenttime< runtime
        thisoffduration = lognrnd(LNpara1(i,j), LNpara2(i,j));
        switch_on = currenttime+thisoffduration;
        thisonduration = exprnd(EXpara1(i,j));
        switch_off = switch_on+thisonduration;
        if switch_on< runtime
            thison = [thison, switch_on];
            if switch_off< runtime
                thisoff = [thisoff, switch_off];
            else
                thisoff = [thisoff, runtime];
            end
        else
            thison = [thison, runtime];
        end
        currenttime = switch_off;
    end
    elseif init == 1
        thisoff = [];
        if startthings(i,j)< runtime
            thison = [startthings(i,j)];
        else
            thison = [];
        end
    end
    while currenttime< runtime
        thisonduration = exprnd(EXpara1(i,j));
        switch_off = currenttime+thisonduration;
        thisoffduration = lognrnd(LNpara1(i,j), LNpara2(i,j));
        switch_on = switch_off+thisoffduration;
        if switch_off< runtime
            thisoff = [thisoff, switch_off];
            if switch_on< runtime
                thison = [thison, switch_on];
            else
                thison = [thison, runtime];
            end
        end
else
    thisoff = [thisoff, runtime];
end
currentTime = switch_on;
end
end
firstonIDX = find(thison > 0, 1);
firstoffIDX = find(thisoff > 0, 1);
firston = thison(firstonIDX);
firstoff = thisoff(firstoffIDX);
thison(thison < 0) = [];
thisoff(thisoff < 0) = [];
thison(thison == 0) = [];
thisoff(thisoff == runtime) = [];
thisoff(thisoff > runtime) = [];
thison(thison > runtime) = [];
if firston > firstoff
    thison = [switchon(i,j), thison];
end
ID_ref = sprintf('/quotesingle.ts1n%d_n%d/quotesingle.ts1', i, j);
ontimes.(ID_ref) = thison;
offtimes.(ID_ref) = thisoff;
end
end
sampleCSV(ontimes, offtimes, nodes, runtime, cut, filepath);
end

B.56 modelv1_multiple.m

function [dir_ref] = modelv1_multiple(nodes, runtime)
% MODELV1_MULTIPLE creates as many simulations using
% method 1 as there are
% entries in the vector NODES
% % NODES is a vector containing the nodes for each of the
% desired
% % RUNTIME is the desired simulation time
% % DIR_REF is the output save directory as a string

timestamp = datestr(now,'yyyyymmdhhmmss');
dir_ref = ['lots/output_',timestamp];
dir_ref_full = ['input/','dir_ref'];
mkdir(dir_ref_full);

cycles = length(nodes);
for i=1:cycles
    filename = ['data',num2str(i),'.csv']; %Save file name
    filepath = [dir_ref_full,'/',filename];
    thisnodes = nodes(i);
    model(thisnodes,runtime,filepath);
end

B.57 modelv2_1.m

function [] = modelv2_1(runtime,probM,filepath)
    %MODELV2_1 Takes a runtime and a weighted selection matrix and generates a
    % CSV file sampled every (default 20) seconds using either method 2b or 2c.
    %
    % This is needed to compare simulation with empirical results from these two
    % papers: V. Gemmetto et al. Mitigation of infectious diseases at school:
    % targeted class closure vs school closure, BMC Infectious Diseases
    % 201414:695 https://doi.org/10.1186/s12879-014-0695-9; R
    % . Mastrandrea % et al,. Contact Patterns in a High School: A Comparison
    % between Data
    % Collected Using Wearable Sensors, Contact Diaries and Friendship Surveys,
    % PLoS ONE 10(9): e0136497. https://doi.org/10.1371/
    % journal.pone.0136497
    %
    % RUNTIME specifies the total sampling period of the simulation
    %
    % PROBM specifies the edge preference matrix. The can be
    % extracted from the
    % data using the function EAPmat.m or generated using MC
    % simulation using
    % the function rndLPM.m
    cut = 20;
    %CUT is the sampling interval in seconds (DEFAULT: 20)
    nodes = size(probM,1);
    linkprobmatrix = probM;
    initialoff = zeros(nodes);
%INITIALOFF is the initial state for the network given as an adjacency matrix

ondurationpara1 = lognrnd(3.2434,sigma_for_mu_and_mean(30.552,3.2434),nodes);
times = struct();
%TIMES will end up having entries labeled nX_nY with entries showing the on
% and off times for the link X-Y as a single vector.
for i=1: nodes-1
    for j=i +1: nodes
        ID_ref = sprintf('n%d_n%d',i,j);
        if initialoff(i,j)>0
            times.(ID_ref) = [0,initialoff(i,j)];
        else
            times.(ID_ref) = [];
        end
    end
end
currenttime = lognrnd(5.6901e-04,1.7957);  %CURRENTTIME should be set to the desired distribution for on interevent

triangleActivations = 0.0556;
estFailure = 0.1316;
adjustedThreshold = triangleActivations/(1-estFailure);

while currenttime<runtime
    flip = unifrnd(0 ,1);
    if flip<adjustedThreshold
        [~,m_lpm] = extractTriangles(times,nodes,
currenttime,linkprobmatrix);
        if sum(sum(m_lpm))~=0
            thislinkmatrix = m_lpm;
        else
            thislinkmatrix = linkprobmatrix;
        end
    else
        thislinkmatrix = linkprobmatrix;
    end
    accept = 0;
cycles = 0;
    while accept == 0
        [ri,rj] = chooselink(thislinkmatrix);
ID_ref = sprintf('%d\n%d', ri, rj);
vec = times.(ID_ref);
if isempty(vec) || currenttime > vec(end)
    % Ensure link is not already active
    accept = 1;
    ontime = currenttime;
    offtime = currenttime + exprnd(ondurationpara1(ri, rj));
    vec = [vec, ontime, offtime];
    % Append to entry in TIMES
    times.(ID_ref) = vec;
end
if cycles > 100
    thislinkmatrix = linkprobmatrix;
else
    cycles = cycles + 1;
end
IET = lognrnd(5.6901e-04, 1.7957);
% IET should be set to the desired distribution for on times
currenttime = currenttime + IET;
end
sampleCSV2(times, nodes, runtime, cut, filepath);
% Passes data to sampleCSV.m for sampling and conversion to CSV file
end

B.58 modelv2_1_multiple.m

function [dir_ref] = modelv2_1_multiple(runtime, probMstruc)
% MODELV1_MULTIPLE creates as many simulations using method 2b as there are
% entries in the structure PROBMSTRUC
% PROBMSTRUC is a structure containing the link selection matrices for each
% of the desired simulations
% RUNTIME is the desired simulation time
% DIR_REF is the output save directory as a string

timestamp = datestr(now, 'yyyyymmmddTHHMMSS');
dir_ref = ['lots/output_', timestamp];
dir_ref_full = ['input/', dir_ref];
mkdir(dir_ref_full);
fields = fieldnames(probMstruc);
cycles = numel(fields);

for i = 1:cycles
    filename = ['data',num2str(i),'.csv']; % Save file name
    filepath = [dir_ref_full,'/',filename];
    probM = probMstruc.{fields{i}};
    modelv2_1(runtime,probM,filepath);
end
end

B.59 modelv2a_1.m

function [] = modelv2a_1(runtime,probM,filepath)
    %MODELV2A_1 Takes a runtime and a weighted selection matrix and generates a
    % CSV file sampled every (default 20) seconds using method 2a.
    %
    % This is needed to compare simulation with empirical results from these two
    % papers: V. Gemmetto et al. Mitigation of infectious diseases at school:
    % targeted class closure vs school closure, BMC Infectious Diseases 2014:695 https://doi.org/10.1186/s12879-014-0695-9; R.
    % Mastrandrea et al., Contact Patterns in a High School: A Comparison between Data
    % Collected Using Wearable Sensors, Contact Diaries and Friendship Surveys,
    % PLoS ONE 10(9): e0136497. https://doi.org/10.1371/journal.pone.0136497
    %
    % RUNTIME specifies the total sampling period of the simulation
    %
    % PROBM specifies the edge preference matrix. The can be extracted from the
    % data using the function EAPmat.m or generated using MC simulation using
    % the function rndLPM.m
    cut = 20;
    %CUT is the sampling interval in seconds (DEFAULT: 20)
nodes = size(probM,1);
linkprobmatrix = probM;
initialoff = zeros(nodes);

%INITIALOFF is the initial state for the network given as an adjacency matrix

ondurationpara1 = lognrnd(3.2434, sigma_for_mu_and_mean(30.552,3.2434),nodes);

times = struct();

%TIMES will end up having entries labeled nX_nY with entries showing the on and off times for the link X-Y as a single vector.
for i=1: nodes-1
    for j=i+1: nodes
        ID_ref = sprintf('/%d_n%d',i,j);
        if initialoff(i,j)>0
            times.(ID_ref) = [0, initialoff(i,j)];
        else
            times.(ID_ref) = [];
        end
    end
end

currenttime = lognrnd(5.6901e-04,1.7957);

%CURRENTTIME should be set to the desired distribution for on interevent timings

while currenttime<runtime
    thislinkmatrix = linkprobmatrix;
    accept = 0;
    while accept == 0
        [ri,rj] = chooselink(thislinkmatrix);
        ID_ref = sprintf('/%d_n%d',ri,rj);
        vec = times.(ID_ref);
        if isempty(vec)||currenttime>vec(end)
            %Ensure link is not already active
            accept = 1;
            ontime = currenttime;
            offtime = currenttime+exprnd(ondurationpara1(ri,rj));
            vec = [vec, ontime, offtime];
            %Append to entry in TIMES
            times.(ID_ref) = vec;
        end
    end
end
IET = lognrnd(5.6901e-04,1.7957);
% IET should be set to the desired distribution for on
% interevent
% timings
currenttime = currenttime+IET;
end
sampleCSV2(times,nodes,runtime,cut,filepath);
%Passes data to sampleCSV.m for sampling and conversion
to CSV file
end

B.60  modelv2a_1_multiple.m

function [dir_ref] = modelv2a_1_multiple(runtime, probMstruc)
% MODELV1_MULTIPLE creates as many simulations using
% method 2a as there are
% entries in the structure PROBMSTRUC
% PROBMSTRUC is a structure containing the link selection
% matrices for each
% of the desired simulations
% RUNTIME is the desired simulation time
% DIR_REF is the output save directory as a string

timestamp = datestr(now,'yyyyymmddTHHMMSS');
dir_ref = ['lots/output_',timestamp];
dir_ref_full = ['input/',dir_ref];
mkdir(dir_ref_full);
fields = fieldnames(probMstruc);
cycles = numel(fields);
for i=1:cycles
    filename = ['data',num2str(i),'.csv']; %Save file
    name
    filepath = [dir_ref_full,'/','',filename];
    probM = probMstruc(fields{i});
    modelv2a_1(runtime,probM,filepath);
end
end

B.61  multiKS.m

This function was written by Whiten [211].
function s=multiKS(a,nx)
% multiKS Multiple distribution Kolomogorov Smirnov test statistic
% 2014-09-22 Matlab2014 W. Whiten
% s=multiKS(a) or s=multiKS(a,nx)
% a Matrix of columns of distribution samples or cells each
% containing a vector or columns of distribution samples
% nx Number of divisions in x direction for a (optional: default 100)
%
% s Maximum probability distance between samples
%
% Examples
% a=rand(20,4); % Put your data here, 4 distributions of 20 samples
% s1=multiKS(a) % Probability of random case being greater
% b=rand(30,2); % Extra data of different size 2 of 30 samples
% s2=multiKS({a,b}) % Probability for combined data
%
% Distribution of statistic can be calculated by probKS e.g.
% [~,d]=probKS(a);
% pr1=sum(multiKS(a)>d)/length(d) % same as pr=multiKS(a)
% pr2=sum(multiKS(rand(size(a))>d)/length(d) % for a second data set
%
% Statistic used is an extension of Kolomogorov Smirnov test:
% maximum difference in probability of the cumulative distributions.
% multiKS is used by probKS to calculate distribution of statistic.
%
% Finds boundary for cumulative probabilities for matrices first at
% constant probabilities, then boundaries are converted to distributions
% with given x values so that probability differences can be calculated
% and if cell data used these are combined for overall upper and lower
% boundaries. Then maximum difference between boundaries
% is found.

% Copyright (C) 2014, W. Whiten (personal W. Whiten@uq.edu.au) BSD license
% (http://opensource.org/licenses/BSD-3-Clause)
%
% See also
% probKS

% divide range of a values into nx intervals for calc of
% prob difference
if(nargin<2)
    nx=100;
end

% case for cell array of vectors
if(iscell(a))
    n=length(a);

    % find xx values to get probability differences
    amin1=realmax;
    amax1=-realmax;
    for j=1:n
        t=a{j};
        amin1=min(min(t(:)),amin1);
        amax1=max(max(t(:)),amax1);
    end
    xx=amin1+((amax1-amin1)/nx)*(0:nx)';

    % find min and max probability values at xx values
    yymin=realmax*ones(nx+1,1);
    yymax=-yymin;
    for j=1:n
        t=sort(a{j}); % t may have more than one column
        if(size(t,1)==1)
            t=t';
        end
        if(size(t,2)==1)
            yy=cumx2xxyy(t,xx);
            yymin=min(yymin,yy);
            yymax=max(yymax,yy);
        else
            yymin=min(yymin,cumx2xxyy(max(t,[],2),xx));
            yymax=max(yymax,cumx2xxyy(min(t,[],2),xx));
            % for k=1:size(t,2)
            %     tt=cumx2xxyy(t(:,k),xx);
            %     yymin=min(yymin,tt);
        end
    end
end
% yymax=max(yymax,tt);
% end
end

else

% a simple array
% find xx values to get probability differences
a1=min(a(:));
a2=max(a(:));
xx=a1+(a2-a1)/nx*(0:nx)';

a=sort(a); % sort individual columns of a

% convert to probabilities at same xx
yymin=cumx2xxyy(max(a,[],2),xx); % lower limit in y direction
yymax=cumx2xxyy(min(a,[],2),xx); % upper limit in y direction

end

s=max(yymax-yymin); % maximum difference of extreme values
return
end

B.62 networkComponents.m

This function was written by Larremore [113].

% [nComponents,sizes,members] = networkComponents(A)
%
% Daniel Larremore
% April 24, 2014
% larremor@hsph.harvard.edu
% http://danlarremore.com
% Comments and suggestions always welcome.
%
% INPUTS:
% A       Matrix. This function takes as an input a
% network adjacency matrix A, for a network that is undirected. If you
% provide a network that is directed, this code is going to make it
% undirected before continuing. Since link weights will not affect
% component sizes, weighted and unweighted networks work equally well. You
% may provide a "full" or a "sparse" matrix.
%
% OUTPUTS:
% nComponents     INT - The number of components
%   in the network.
% sizes            vector<INT> - a vector of
%   component sizes, sorted,
% members          cell<vector<INT>> a cell array of
%   vectors, each
%   entry of which is a membership list for that
%   component, sorted,
% descending by component size.
%
% Example: (uncomment and copy and paste into MATLAB
% command window)
% % Generate a 1000 node network adjacency matrix, A
% A = floor(1.0015*rand(1000,1000)); A=A+A'; A(A==2)=1; A
% (1:1001:end) = 0;
% % Call networkComponents function
% [nComponents,sizes,members] = networkComponents(A);
% % get the size of the largest component
% sizeLC = sizes(1);
% % get a network adjacency matrix for ONLY the largest
% % component
% LC = A(members{1},members{1});

function [nComponents,sizes,members] = networkComponents(~
    A)
% Number of nodes
    N = size(A,1);
% Remove diagonals
    A(1:N+1:end) = 0;
% make symmetric, just in case it isn't
    A=A+A';
% Have we visited a particular node yet?
    isDiscovered = zeros(N,1);
% Empty members cell
    members = {};
% check every node
    for n=1:N
        if ~isDiscovered(n)
            % started a new group so add it to members
            members{end+1} = n;
        end
    end

    % % Generate a 1000 node network adjacency matrix, A
    % A = floor(1.0015*rand(1000,1000)); A=A+A'; A(A==2)=1; A
    % (1:1001:end) = 0;
    % % Call networkComponents function
    % [nComponents,sizes,members] = networkComponents(A);
    % % get the size of the largest component
    % sizeLC = sizes(1);
    % % get a network adjacency matrix for ONLY the largest
    % % component
    % LC = A(members{1},members{1});
% account for discovering n
isDiscovered(n) = 1;
% set the ptr to 1
ptr = 1;
while (ptr <= length(members{end}))
    % find neighbors
    nbrs = find(A(:,members{end}(ptr)));
    % here are the neighbors that are undiscovered
    newNbrs = nbrs(isDiscovered(nbrs)==0);
    % we can now mark them as discovered
    isDiscovered(newNbrs) = 1;
    % add them to member list
    members{end}(end+1:end+length(newNbrs)) = newNbrs;
    % increment ptr so we check the next member of this component
    ptr = ptr+1;
end
% number of components
nComponents = length(members);
for n=1:nComponents
    % compute sizes of components
    sizes(n) = length(members{n});
end
[sizes,idx] = sort(sizes,'descend');
members = members(idx);

B.63  num2matlabstr.m

function [str] = num2matlabstr(r)
%NUM2MATLABSTR converts a number to a latex-prepared string expressed in
% scientific form
% R is the number to convert
% STR is the string prepared for latex
if r==Inf
    str = '\infty';
elseif r==-Inf
    str = '-\infty';
```matlab
elseif isnan(r)
    str = 'X';
elseif r==0
    str = '0';
else
    expon = floor(log10(abs(r)));
    coeff = r/(10^expon);
    exonstr = num2str(expon);
    coeffstr = num2str(coeff,'%.4f');
    str = strcat(coeffstr,'\times10\{}{\exonstr}\}');
end
end

B.64 pairgraphs.m

function [] = pairgraphs(gendata,realdata,property_name,save_dir)
% PAIRGRAPHS produces a figure containing the eCCDFs for the two given sets
% of data, and prints the Kolmogorov-Smirnov distance between them
%
% GENDATA is the generated data as a vector
% REALDATA is the observed data as a vector
% PROPERTY_NAME is the property contained in the vectors as a string
% SAVE_DIR is the location to save the figure, given as a string

cleanName = strrep(property_name,' ','-');
figurefilename = [save_dir,'/',cleanName];
[~,~,KSdist] = kstest2(gendata,realdata);
[F_gen,X_gen] = ecdf(gendata);
ccdf_gen = 1-F_gen;
[F_real,X_real] = ecdf(realdata);
ccdf_real = 1-F_real;
plotthings = figure();
hold on
plot(X_gen,cdf_gen,'x');
plot(X_real,cdf_real,'o');
set(gca,'XScale','log');
set(gca,'YScale','log');
xlabel(property_name);
ylabel('CCDF');
```
B.65 probKS.m

This function was written by Whiten [211].

```matlab
function [pr,d]=probKS(a,rpts,nx)
% probKS  Probability for multi Kolmogorov Smirnov statistic
% 2014-09-22 Matlab2014 W. Whiten
%
% pr=probKS(a) or [pr,d]=probKS(a,rpts,nx)
% a  Matrix of columns of distribution samples or cells each containing
%    a vector or a matrix of columns of distribution samples
% rpts Number of repeats (optional: default 10000)
% nx Number of divisions in x direction for a (optional: default 100)
%
% pr  Probability of statistic for random case being greater than
% statistic for given data (a)
% d   Sorted array of rpts samples of statistic
%
% Examples
% a=rand(20,4); % Put your data here, 4 distributions of 20 samples
% pr1=probKS(a) % Probability of random case being greater
% b=rand(30,2); % Extra data of different size 2 of 30 samples
% pr2=probKS({a,b}) % Probability for combined data
%
% Use multiKS for multiple cases of data with same dimensions e.g.
% [pr1,d]=probKS(a); % get probability and distribution
% s1=multiKS(rand(size(a))); % statistic for second data set (same size)
% pr2=sum(s1>d)/length(d) % much faster for a second data set
```
% Statistic used is an extension of Kolomogorov Smirnov test:
% maximum difference in probability of the cumulative distributions.
% Distribution of statistic is calculated by simulation.
%
% Finds boundary for cumulative probabilities for matrices first at
% constant probabilities, then boundaries are converted to distributions
% with given x values so that probability differences can be calculated
% and if cell data used these are combined for overall upper and lower
% boundaries. Then maximum difference between boundaries is found.
%
% Copyright (C) 2014, W. Whiten (personal W.Whiten@uq.edu.au) BSD license
% (http://opensource.org/licenses/BSD-3-Clause)
%
% See also
% multiKS

if( nargin<3)
    nx=100; % number of divisions for cumx2xxyy
end

if( nargin<2 || isempty ( rpts ))
    rpts=10000; % number of repeats for distribution calculation
end

s=multiKS(a,nx);

d=zeros(rpts,1);
xx=(1:nx)/(nx+1);
for i=1:rpts
    if(iscell(a))
        for j=1:length(a);
            a{j}=rand(size(a{j}));
        end
        d(i)=multiKS(a,nx);
    else
        a=sort(rand(size(a))); % sort individual columns of a
        amax=max(a,[],2); % upper limit of values
        amin=min(a,[],2); % lower limit of values
    end
end
% convert to probabilities at same xx
yymax=cumx2xxyy(amax,xx);
yymin=cumx2xxyy(amin,xx);
d(i)=max(yymin-yymax);  % maximum difference of extreme values

pr=sum(s<d)/rpts;
if(nargout>1)
d=sort(d);
end
return
end

B.66 pullData.m

function dataStructure = pullData(folder,file,format)

% PULLDATA takes a CSV file and extracts Active Edges, Active Node, Node Activity Potential, Component Edges, Component Nodes, Global Clustering % Coefficient, Interaction Times, Component Counts and Time Between % Contacts. It outputs these into a structure.

% FOLDER is the path of the folder containing the data % FILE is the file within this folder % FORMAT is the formatting of this CSV file % DATASTRUCTURE is a structure containing vectors with individual % measurements of all of our chosen metrics

input = [folder,'/',$file$];

fid = fopen(input);
rawdata = textscan(fid,format,'Delimiter','
fclose(fid);

%== Extract and Clean Data==%
data = cell2mat(rawdata);
data(:,1) = data(:,1)-data(1,1);
lowestID = min(min(data(:,2)),min(data(:,3)));
data(:,2) = data(:,2)-lowestID+1;
data(:,3) = data(:,3)-lowestID+1;
number_rows = size(data,1);
parfor i=1:number_rows
    thisrow = data(i,:);
    col2 = thisrow(1,2);
    col3 = thisrow(1,3);
    if col2 > col3
        thisrow(1,2) = col3;
        thisrow(1,3) = col2;
        data(i,:) = thisrow;
    end
end
all_IDs = [data(:,2); data(:,3)];
all_active = unique(all_IDs);
um_people = size(all_active,1);
data2 = data(:,2);
data3 = data(:,3);
for i=1:num_people
    oldID = all_active(i);
    data2(data2==oldID) = -i;
    data3(data3==oldID) = -i;
end
data(:,2) = -data2;
data(:,3) = -data3;

% Global Variables
num_times = size(unique(data(:,1)),1);
data_length = size(data(:,1),1);
um_people = max([data(:,2); data(:,3)]);
number_rows = size(data,1);
contact_time = 20;
dataStructure.NumberStudents_data = num_people;

% Sorted Stuff
[~, order] = sort(data(:,3));
partsorteddata = data(order,:);
[~, order] = sort(partsorteddata(:,2));
sorteddata = partsorteddata(order,:);

% Active Edges
links = zeros(1,num_times);
maxlinks = num_people*(num_people-1)/2;
parfor m=1:num_times
    thisadj = zeros(num_people);
    current_time = (m-1)*contact_time;
    for i=1:data_length
        test_time = data(i,1);
if test_time==current_time
    person1 = data(i,2);
    person2 = data(i,3);
    thisadj(person1,person2) = 1;
    thisadj(person2,person1) = 1;
end
end
adjsum = sum(sum(thisadj));
numlinks = adjsum/2;
links(m) = numlinks/maxlinks;
end
dataStructure.ActiveLinks_data = links;

% Active Nodes
nodes = zeros(1,num_times);

parfor m=1:num_times
    thisactive = zeros(1,num_people);
    current_time = (m-1)*contact_time;
    for i=1:data_length
        test_time = data(i,1);
        if test_time==current_time
            person1 = data(i,2);
            person2 = data(i,3);
            thisactive(person1) = 1;
            thisactive(person2) = 1;
        end
    end
    nodes(m) = sum(thisactive)/num_people;
end
dataStructure.NodesActive_data = nodes;

% Activity Potential
j = 1;
interactions = zeros(1,num_people);
step_vector = [contact_time 0 0];
while j<number_rows+1
    ID1 = sorteddata(j,2);
    ID2 = sorteddata(j,3);
    interactions(ID1) = interactions(ID1)+1;
    interactions(ID2) = interactions(ID2)+1;
    current_row = sorteddata(j,:);
    if j == number_rows
        next_row = [0 0 0];
    else
        next_row = sorteddata(j+1,:);
    end
    while isequal(next_row,current_row+step_vector)
j = j+1;
current_row = sorteddata(j,:);
if j == number_rows
    next_row = [0 0 0];
else
    next_row = sorteddata(j+1,:);
end
end
j = j+1;
end
activityPot = interactions/sum(interactions);
dataStructure.ActivityPotential_data = activityPot;

\% Component Edges
rawFracEdges = zeros(num_times,num_people);
parfor m=1:num_times
    thisadj = zeros(num_people);
current_time = (m-1)*contact_time;
    for i=1:data_length
        test_time = data(i,1);
            if test_time==current_time
                person1 = data(i,2);
                person2 = data(i,3);
                thisadj(person1,person2) = 1;
                thisadj(person2,person1) = 1;
            end
        end
    end
edgesActive = sum(sum(thisadj));
[~,~,thisCompGroups] = networkComponents(thisadj);
thisNumComps = length(thisCompGroups);
thisEdges = zeros(1,thisNumComps);
for j=1:thisNumComps
    thisNodes = cell2mat(thisCompGroups(j));
    thisNodesSize = length(thisNodes);
        if thisNodesSize == 1
            thisEdges(j)=0;
        else
            thisSubMat = thisadj(thisNodes,thisNodes);
            thisAdjSum = sum(sum(thisSubMat));
            thisNumEdges = thisAdjSum/2;
            thisEdges(j) = thisNumEdges;
        end
    end
if edgesActive > 0
    thisEdges = thisEdges/edgesActive;
end
thisPadding = num_people - length(thisEdges);
thisEdges = [thisEdges zeros(1,thisPadding)];
rawFracEdges(m,:) = thisEdges;

compEdgeFracs = rawFracEdges(:,');
dataStructure.ComponentEdges_data = compEdgeFracs;

%Component Nodes
rawCompSizes = zeros(num_times,num_people);

parfor m=1:num_times
    thisadj = zeros(num_people);
    current_time = (m-1)*contact_time;
    for i=1:data_length
        test_time = data(i,1);
        if test_time==current_time
            person1 = data(i,2);
            person2 = data(i,3);
            thisadj(person1,person2) = 1;
            thisadj(person2,person1) = 1;
        end
    end

    [~,thisCompSizes,:) = networkComponents(thisadj);
    thisCompSizes(thisCompSizes==1) = [];
    nodesActive = sum(thisCompSizes);
    thisPadding = num_people - length(thisCompSizes);
    thisCompSizes = [thisCompSizes zeros(1,thisPadding)];
    if nodesActive == 0
        thisCompFracs = thisCompSizes;
    else
        thisCompFracs = thisCompSizes/nodesActive;
    end
    rawCompSizes(m,:) = thisCompFracs;
end

compSizes = rawCompSizes(:,');
dataStructure.ComponentNodes_data = compSizes;

%Global Clustering Coefficient
clustering = zeros(1,num_times);
triangles = zeros(1,num_times);

parfor m=1:num_times
    thisadj = zeros(num_people);
    current_time = (m-1)*contact_time;
    for i=1:data_length
        test_time = data(i,1);
        if test_time==current_time
            person1 = data(i,2);
            person2 = data(i,3);
            thisadj(person1,person2) = 1;
            thisadj(person2,person1) = 1;
        end
    end
person1 = data(i,2);
person2 = data(i,3);
thisadj(person1,person2) = 1;
thisadj(person2,person1) = 1;

end

end
adj2 = thisadj^2;
adj3 = thisadj^3;
adj2sum = sum(sum(adj2));
contrip = adj2sum - trace(adj2);
if contrip==0
  clustering(m) = 0;
else
  clustering(m) = trace(adj3)/contrip;
end
triangles(m) = trace(adj3)/6;
end
dataStructure.Clustering_data = clustering;
dataStructure.Triangles_data = triangles;

%Interaction Times
times = zeros(1,number_rows);
j = 1;
times_k = 1;
step_vector = [contact_time 0 0];
while j<number_rows+1
  contact = contact_time;
current_row = sorteddata(j,:);
  if j == number_rows
    next_row = [0 0 0];
  else
    next_row = sorteddata(j+1,:);
  end
  while isequal(next_row,current_row+step_vector)
    contact = contact+contact_time;
j = j+1;
current_row = sorteddata(j,:);
    if j == number_rows
      next_row = [0 0 0];
    else
      next_row = sorteddata(j+1,:);
    end
end
  times(times_k) = contact;
j = j+1;
times_k = times_k+1;
end
times(times_k:end) = []; 
dataStructure.InteractionTimes_data = times;

%Number of Components
components = zeros(1,num_times);

parfor m=1:num_times
    thisadj = zeros(num_people);
    current_time = (m-1)*contact_time;
    for i=1:data_length
        test_time = data(i,1);
        if test_time==current_time
            person1 = data(i,2);
            person2 = data(i,3);
            thisadj(person1,person2) = 1;
            thisadj(person2,person1) = 1;
        end
    end
    [~,thisComp,~] = networkComponents(thisadj);
    thisComp(thisComp==1)=[1];
    thisCompCount = length(thisComp);
    components(m) = thisCompCount;
end
dataStructure.Components_data = components;

%Time Between Contacts
step = 20;
min_time = min(data(:,1));
max_time = max(data(:,1));
times = ((max_time-min_time)/step)+1;
data_length = size(data(:,1),1);
num_people = max([data(:,2); data(:,3)]); rawactivity = zeros(data_length,num_people+1);

parfor i=1:data_length
    thisrawactivity = zeros(1,num_people+1);
    thisrawactivity(1) = data(i,1);
    person1 = data(i,2);
    person2 = data(i,3);
    thisrawactivity(person1+1) = 1;
    thisrawactivity(person2+1) = 1;
    rawactivity(i,:) = thisrawactivity;
end

activity = zeros(times,num_people);

parfor i=1:times
    currenttime = ((i-1)*step)+min_time;
activerows = rawactivity(:,1)==
currenttime,:);
activerows = activerows(:,2:end);
thisactivity = sum(activerows,1);
thisactivity = (thisactivity>0);
activity(i,:) = thisactivity;
end

activity = [activity; ones(1,num_people)];

long = activity(:,);
long = long';
dlong = diff([1 long 1]);
startIndex = find(dlong < 0);
endIndex = find(dlong > 0)-1;
nocontact = endIndex-startIndex+1;
nocontact = nocontact*20;
dataStructure.NoContactTimes_data = nocontact;

maxN=max(max(data(:,2)),max(data(:,3)));
activationtimes = [];

for i=1:maxN-1
    for j=i+1:maxN
        S1 = data(:,2)==i;
        S2 = data(:,3)==j;
        T1 = data(:,3)==i;
        T2 = data(:,2)==j;
        S12 = S1 & S2;
        T12 = T1 & T2;
        ST12 = S12 | T12;
        changes = diff(ST12);
        binary = [0;changes];
        thistimes = data(binary==1,1);
        activationtimes = [activationtimes;thistimes];
    end
end

sorted = sort(activationtimes);
timebetween = diff(sorted);
dataStructure.IntereventTimes_data = timebetween;

firstactivations = [];
for i=1:maxN-1
    for j=i+1:maxN
        S1 = data(:,2)==i;
        S2 = data(:,3)==j;
T1 = data(:,3)==i;
T2 = data(:,2)==j;
S12 = S1 & S2;
T12 = T1 & T2;
ST12 = S12|T12;
currentTimes = data(ST12,1);
if isempty(currentTimes)
    thisactivation = Inf;
else
    thisactivation = currentTimes(1);
end
firstactivations = [firstactivations;
                    thisactivation];
end
dataStructure.FirstActivationTimes_data =
    firstactivations;
end

B.67 pvals_ex.m

function [p_vals] = pvals_ex(dataLength,lambda,Statistics,
cut,n,gap)
% PVALS_EX estimates the p-values for the Exponential distributions
% DATALENGTH is the number of entries in the data
% LAMBDA are the parameters in the Exponential distribution
% STATISTICS is the statistics structure
% CUT is the number of extreme data points removed
% N is the precision
% GAP is sampling interval
% P_VALS is a structure containing our estimated p-values
KolD = Statistics.Kolmogorov_D;
CvM = Statistics.Cramer_von_Mises;
Kuiper = Statistics.Kuiper;
Watson = Statistics.Watson;
AD = min(Statistics.Anderson_Darling,1E99);
KL = Statistics.Kullback_Leibler;
JS = Statistics.Jensen_Shannon;
num_MC = 10^n;
KolD_stat = zeros(1,num_MC);
CvM_stat = zeros(1,num_MC);
Kuiper_stat = zeros(1,num_MC);
Watson_stat = zeros(1,num_MC);
AD_stat = zeros(1,num_MC);
KL_stat = zeros(1,num_MC);
JS_stat = zeros(1,num_MC);

for i=1:num_MC
    data = exprnd(lambda,dataLength,1);
data = sort(data);
if cut>0
    data(end-cut+1:end) = [];
end
    PDF = exppdf(data,lambda);
    CDF = expcdf(data,lambda);
    thisfit = testStatistics(data,CDF,PDF,gap);
    KolD_stat(i) = thisfit.Kolmogorov_D;
    CvM_stat(i) = thisfit.Cramer_von_Mises;
    Kuiper_stat(i) = thisfit.Kuiper;
    Watson_stat(i) = thisfit.Watson;
    AD_stat(i) = thisfit.Anderson_Darling;
    KL_stat(i) = thisfit.Kullback_Leibler;
    JS_stat(i) = thisfit.Jensen_Shannon;
end

[F_KolD,X_KolD] = ecdf(KolD_stat);
[F_CvM,X_CvM] = ecdf(CvM_stat);
[F_Kuiper,X_Kuiper] = ecdf(Kuiper_stat);
[F_Watson,X_Watson] = ecdf(Watson_stat);
[F_AD,X_AD] = ecdf(AD_stat);
[F_KL,X_KL] = ecdf(KL_stat);
[F_JS,X_JS] = ecdf(JS_stat);
clearvars KolD_stat CvM_stat Kuiper_stat Watson_stat AD_stat KL_stat JS_stat

X_KolD = [-1E99;X_KolD(2:end);1E99];
X_CvM = [-1E99;X_CvM(2:end);1E99];
X_Kuiper = [-1E99;X_Kuiper(2:end);1E99];
X_Watson = [-1E99;X_Watson(2:end);1E99];
X_AD = [-1E99;X_AD(2:end);1E99];
X_KL = [-1E99;X_KL(2:end);1E99];
X_JS = [-1E99;X_JS(2:end);1E99];

F_KolD = [0;F_KolD(2:end);1];
F_CvM = [0;F_CvM(2:end);1];
F_Kuiper = [0;F_Kuiper(2:end);1];
F_Watson = [0;F_Watson(2:end);1];
F_AD = [0; F_AD(2:end); 1];
F_KL = [0; F_KL(2:end); 1];
F_JS = [0; F_JS(2:end); 1];
p_KolD = 1 - interp1(X_KolD, F_KolD, KolD, 'next');
p_CvM = 1 - interp1(X_CvM, F_CvM, CvM, 'next');
p_Kuiper = 1 - interp1(X_Kuiper, F_Kuiper, Kuiper, 'next');
p_Watson = 1 - interp1(X_Watson, F_Watson, Watson, 'next');
p_AD = 1 - interp1(X_AD, F_AD, AD, 'next');
p_KL = 1 - interp1(X_KL, F_KL, KL, 'next');
p_JS = 1 - interp1(X_JS, F_JS, JS, 'next');

p_vals = struct( 'Kolmogorov_D', p_KolD, ...
    'Cramer_von_Mises', p_CvM, ...
    'Kuiper', p_Kuiper, ...
    'Watson', p_Watson, ...
    'Anderson_Darling', p_AD, ...
    'Kullback_Leibler', p_KL, ...
    'Jensen_Shannon', p_JS);
end

B.68 pvals_gm.m

function [p_vals] = pvals_gm(dataLength, a, b, Statistics, cut, n, gap)
% PVALS_WB estimates the p-values for the Gamma distributions
% DATALENGTH is the number of entries in the data
% A,B are the parameters in the Gamma distribution
% STATISTICS is the statistics structure
% CUT is the number of extreme data points removed
% N is the precision
% GAP is sampling interval
% P_VALS is a structure containing our estimated p-values
KolD = Statistics.Kolmogorov_D;
CvM = Statistics.Cramer_von_Mises;
Kuiper = Statistics.Kuiper;
Watson = Statistics.Watson;
AD = min(Statistics.Anderson_Darling, 1E99);
KL = Statistics.Kullback_Leibler;
JS = Statistics.Jensen_Shannon;
num_MC = 10^n;
KolD_stat = zeros(1,num_MC);
CvM_stat = zeros(1,num_MC);
Kuiper_stat = zeros(1,num_MC);
Watson_stat = zeros(1,num_MC);
AD_stat = zeros(1,num_MC);
KL_stat = zeros(1,num_MC);
JS_stat = zeros(1,num_MC);

parfor i=1:num_MC
    data = gamrnd(a,b,dataLength,1);
    data = sort(data);
    if cut>0
        data(end-cut+1:end) = [];
    end
    PDF = gampdf(data,a,b);
    CDF = gamcdf(data,a,b);
    thisfit = testStatistics(data,CDF,PDF,gap);
    KolD_stat(i) = thisfit.Kolmogorov_D;
    CvM_stat(i) = thisfit.Cramer_von_Mises;
    Kuiper_stat(i) = thisfit.Kuiper;
    Watson_stat(i) = thisfit.Watson;
    AD_stat(i) = thisfit.Anderson_Darling;
    KL_stat(i) = thisfit.Kullback_Leibler;
    JS_stat(i) = thisfit.Jensen_Shannon;
end

[F_KolD,X_KolD] = ecdf(KolD_stat);
[F_CvM,X_CvM] = ecdf(CvM_stat);
[F_Kuiper,X_Kuiper] = ecdf(Kuiper_stat);
[F_Watson,X_Watson] = ecdf(Watson_stat);
[F_AD,X_AD] = ecdf(AD_stat);
[F_KL,X_KL] = ecdf(KL_stat);
[F_JS,X_JS] = ecdf(JS_stat);

clearvars KolD_stat CvM_stat Kuiper_stat Watson_stat
          AD_stat KL_stat JS_stat

X_KolD = [-1E99;X_KolD(2:end);1E99];
X_CvM = [-1E99;X_CvM(2:end);1E99];
X_Kuiper = [-1E99;X_Kuiper(2:end);1E99];
X_Watson = [-1E99;X_Watson(2:end);1E99];
X_AD = [-1E99;X_AD(2:end);1E99];
X_KL = [-1E99;X_KL(2:end);1E99];
X_JS = [-1E99;X_JS(2:end);1E99];
F_KolD = [0;F_KolD(2:end);1];
F_CvM = [0;F_CvM(2:end);1];
F_Kuiper = [0; F_Kuiper(2:end);1];
F_Watson = [0; F_Watson(2:end);1];
F_AD = [0; F_AD(2:end);1];
F_KL = [0; F_KL(2:end);1];
F_JS = [0; F_JS(2:end);1];
p_KolD = 1-interp1(X_KolD,F_KolD,KolD,'next');
p_CvM = 1-interp1(X_CvM,F_CvM,CvM,'next');
p_Kuiper = 1-interp1(X_Kuiper,F_Kuiper,Kuiper,'next');
p_Watson = 1-interp1(X_Watson,F_Watson,Watson,'next');
p_AD = 1-interp1(X_AD,F_AD,AD,'next');
p_KL = 1-interp1(X_KL,F_KL,KL,'next');
p_JS = 1-interp1(X_JS,F_JS,JS,'next');

p_vals = struct('Kolmogorov_D',p_KolD,...
    'Cramer_von_Mises',p_CvM,...
    'Kuiper',p_Kuiper,...
    'Watson',p_Watson,...
    'Anderson_Darling',p_AD,...
    'Kullback_Leibler',p_KL,...
    'Jensen_Shannon',p_JS);

function [p_vals] = pvals_gp(dataLength,k,sigma,theta,Statistics,cut,n,gap)
    % PVALS_WB estimates the p-values for the Generalised Pareto distributions
    % DATALENGTH is the number of entries in the data
    % K,SIGMA,THETA are the parameters in the Generalised Pareto distribution
    % STATISTICS is the statistics structure
    % CUT is the number of extreme data points removed
    % N is the precision
    % GAP is sampling interval
    % P_VALS is a structure containing our estimated p-values
    KolD = Statistics.Kolmogorov_D;
    CvM = Statistics.Cramer_von_Mises;
    Kuiper = Statistics.Kuiper;
    Watson = Statistics.Watson;
    AD = min(Statistics.Anderson_Darling,1E99);
    KL = Statistics.Kullback_Leibler;
    JS = Statistics.Jensen_Shannon;
num_MC = 10^n;
KolD_stat = zeros(1,num_MC);
CvM_stat = zeros(1,num_MC);
Kuiper_stat = zeros(1,num_MC);
Watson_stat = zeros(1,num_MC);
AD_stat = zeros(1,num_MC);
KL_stat = zeros(1,num_MC);
JS_stat = zeros(1,num_MC);
parfor i = 1 : num_MC
    data = gprnd(k,sigma,theta,dataLength,1);
data = sort(data);
    if cut > 0
        data(end-cut+1:end) = [];
    end
    CDF = gpcdf(data,k,sigma,theta);
    PDF = gppdf(data,k,sigma,theta);
    thisfit = testStatistics(data,CDF,PDF,gap);
    KolD_stat(i) = thisfit.Kolmogorov_D;
    CvM_stat(i) = thisfit.Cramer_von_Mises;
    Kuiper_stat(i) = thisfit.Kuiper;
    Watson_stat(i) = thisfit.Watson;
    AD_stat(i) = thisfit.Anderson_Darling;
    KL_stat(i) = thisfit.Kullback_Leibler;
    JS_stat(i) = thisfit.Jensen_Shannon;
end
[F_KolD,X_KolD] = ecdf(KolD_stat);
[F_CvM,X_CvM] = ecdf(CvM_stat);
[F_Kuiper,X_Kuiper] = ecdf(Kuiper_stat);
[F_Watson,X_Watson] = ecdf(Watson_stat);
[F_AD,X_AD] = ecdf(AD_stat);
[F_KL,X_KL] = ecdf(KL_stat);
[F_JS,X_JS] = ecdf(JS_stat);
clearvars KolD_stat CvM_stat Kuiper_stat Watson_stat
    AD_stat KL_stat JS_stat
X_KolD = [-1E99;X_KolD(2:end);1E99];
X_CvM = [-1E99;X_CvM(2:end);1E99];
X_Kuiper = [-1E99;X_Kuiper(2:end);1E99];
X_Watson = [-1E99;X_Watson(2:end);1E99];
X_AD = [-1E99;X_AD(2:end);1E99];
X_KL = [-1E99;X_KL(2:end);1E99];
X_JS = [-1E99;X_JS(2:end);1E99];
F_KolD = [0; F_KolD(2:end);1];
F_CvM = [0; F_CvM(2:end);1];
F_Kuiper = [0; F_Kuiper(2:end);1];
F_Watson = [0; F_Watson(2:end);1];
F_AD = [0; F_AD(2:end);1];
F_KL = [0; F_KL(2:end);1];
F_JS = [0; F_JS(2:end);1];
p_KolD = 1- interp1 ( X_KolD , F_KolD ,KolD ,'
ext');
p_CvM = 1- interp1(X_CvM,F_CvM,CvM,'
ext');
p_Kuiper = 1- interp1(X_Kuiper,F_Kuiper,Kuiper,'
ext');
p_Watson = 1- interp1(X_Watson,F_Watson,Watson,'
ext');
p_AD = 1- interp1(X_AD,F_AD,AD,'
ext');
p_KL = 1- interp1(X_KL,F_KL,KL,'
ext');
p_JS = 1- interp1(X_JS,F_JS,JS,'
ext');
p_vals = struct ('Kolmogorov_D',p_KolD,...
              'Cramer_von_Mises',p_CvM,...
              'Kuiper',p_Kuiper,...
              'Watson',p_Watson,...
              'Anderson_Darling',p_AD,...
              'Kullback_Leibler',p_KL,...
              'Jensen_Shannon',p_JS);
end

B.70  pvals_ln.m

function [p_vals] = pvals_ln(dataLength,mu,sigma,
    Statistics,cut,n,gap)
  %PVALS_WB estimates the p-values for the Log-Normal
distributions
  % DATALENGTH is the number of entries in the data
  % MU,SIGMA are the parameter in the Log-Normal
  % distribution
  % STATISTICS is the statistics structure
  % CUT is the number of extreme data points removed
  % N is the precision
  % GAP is sampling interval
  % P_VALS is a structure containing our estimated p-values
  KolD = Statistics.Kolmogorov_D;
  CvM = Statistics.Cramer_von_Mises;
  Kuiper = Statistics.Kuiper;
  Watson = Statistics.Watson;
AD = min(Statistics.Anderson_Darling,1E99);
KL = Statistics.Kullback_Leibler;
JS = Statistics.Jensen_Shannon;

num_MC = 10^n;

KolD_stat = zeros(1,num_MC);
CvM_stat = zeros(1,num_MC);
Kuiper_stat = zeros(1,num_MC);
Watson_stat = zeros(1,num_MC);
AD_stat = zeros(1,num_MC);
KL_stat = zeros(1,num_MC);
JS_stat = zeros(1,num_MC);

parfor i=1:num_MC
    data = lognrnd(mu,sigma,dataLength,1);
    data = sort(data);
    if cut>0
        data(end-cut+1:end) = [];
    end
    PDF = lognpdf(data,mu,sigma);
    CDF = logncdf(data,mu,sigma);
    thisfit = testStatistics(data,CDF,PDF,gap);
    KolD_stat(i) = thisfit.Kolmogorov_D;
    CvM_stat(i) = thisfit.Cramer_von_Mises;
    Kuiper_stat(i) = thisfit.Kuiper;
    Watson_stat(i) = thisfit.Watson;
    AD_stat(i) = thisfit.Anderson_Darling;
    KL_stat(i) = thisfit.Kullback_Leibler;
    JS_stat(i) = thisfit.Jensen_Shannon;
end

[F_KolD,X_KolD] = ecdf(KolD_stat);
[F_CvM,X_CvM] = ecdf(CvM_stat);
[F_Kuiper,X_Kuiper] = ecdf(Kuiper_stat);
[F_Watson,X_Watson] = ecdf(Watson_stat);
[F_AD,X_AD] = ecdf(AD_stat);
[F_KL,X_KL] = ecdf(KL_stat);
[F_JS,X_JS] = ecdf(JS_stat);

clearvars KolD_stat CvM_stat Kuiper_stat Watson_stat
    AD_stat KL_stat JS_stat

X_KolD = [-1E99;X_KolD(2:end);1E99];
X_CvM = [-1E99;X_CvM(2:end);1E99];
X_Kuiper = [-1E99;X_Kuiper(2:end);1E99];
X_Watson = [-1E99;X_Watson(2:end);1E99];
p_vals = struct('Kolmogorov_D',p_KolD,...
    'Cramer_von_Mises',p_CvM,...
    'Kuiper',p_Kuiper,...
    'Watson',p_Watson,...
    'Anderson_Darling',p_AD,...
    'Kullback_Leibler',p_KL,...
    'Jensen_Shannon',p_JS);

end

B.71 pvals_ml.m

function [p_vals] = pvals_ml(dataLength,beta,gamma,
    Statistics,cut,n,gap)

%PVALS_ML estimates the p-values for the Mittag-Leffler distributions
% DATALENGTH is the number of entries in the data
% BETA,GAMMA are the parameters in the Mittag-Leffler distribution
% STATISTICS is the statistics structure
% CUT is the number of extreme data points removed
% N is the precision
% GAP is sampling interval
% P_VALS is a structure containing our estimated p-values
KolD = Statistics.Kolmogorov_D;
CvM = Statistics.Cramer_von_Mises;
Kuiper = Statistics.Kuiper;
Watson = Statistics.Watson;
AD = min(Statistics.Anderson_Darling,1E99);
KL = Statistics.Kullback_Leibler;
JS = Statistics.Jensen_Shannon;

num_MC = 10^n;

KolD_stat = zeros(1,num_MC);
CvM_stat = zeros(1,num_MC);
Kuiper_stat = zeros(1,num_MC);
Watson_stat = zeros(1,num_MC);
AD_stat = zeros(1,num_MC);
KL_stat = zeros(1,num_MC);
JS_stat = zeros(1,num_MC);

for i=1:num_MC
    data = mlrnd(beta,gamma,dataLength,1);
    data = sort(data);
    if cut>0
        data(end-cut+1:end) = [];
    end
    PDF = (-beta ./ data).*mlf(beta,1,-gamma*data.^beta,6);
    CDF = ones(length(data),1)-mlf(beta,1,-gamma*data.^beta,6);
    thisfit = testStatistics(data,CDF,PDF,gap);
    KolD_stat(i) = thisfit.Kolmogorov_D;
    CvM_stat(i) = thisfit.Cramer_von_Mises;
    Kuiper_stat(i) = thisfit.Kuiper;
    Watson_stat(i) = thisfit.Watson;
    AD_stat(i) = thisfit.Anderson_Darling;
    KL_stat(i) = thisfit.Kullback_Leibler;
    JS_stat(i) = thisfit.Jensen_Shannon;
end

[F_KolD,X_KolD] = ecdf(KolD_stat);
[F_CvM,X_CvM] = ecdf(CvM_stat);
[F_Kuiper,X_Kuiper] = ecdf(Kuiper_stat);
[F_Watson,X_Watson] = ecdf(Watson_stat);
[F_AD,X_AD] = ecdf(AD_stat);
[F_KL,X_KL] = ecdf(KL_stat);
[F_JS,X_JS] = ecdf(JS_stat);

clearvars KolD_stat CvM_stat Kuiper_stat Watson_stat
    AD_stat KL_stat JS_stat
X_KolD = [-1E99; X_KolD(2:end);1E99];
X_CvM = [-1E99; X_CvM(2:end);1E99];
X_Kuiper = [-1E99; X_Kuiper(2:end);1E99];
X_Watson = [-1E99; X_Watson(2:end);1E99];
X_AD = [-1E99; X_AD(2:end);1E99];
X_KL = [-1E99; X_KL(2:end);1E99];
X_JS = [-1E99; X_JS(2:end);1E99];

F_KolD = [0; F_KolD(2:end);1];
F_CvM = [0; F_CvM(2:end);1];
F_Kuiper = [0; F_Kuiper(2:end);1];
F_Watson = [0; F_Watson(2:end);1];
F_AD = [0; F_AD(2:end);1];
F_KL = [0; F_KL(2:end);1];
F_JS = [0; F_JS(2:end);1];

p_KolD = 1- interp1 ( X_KolD , F_KolD ,KolD ,'/quotesingle.ts1
next'/quotesingle.ts1);
p_CvM = 1- interp1 ( X_CvM , F_CvM ,CvM ,'/quotesingle.ts1
next'/quotesingle.ts1);
p_Kuiper = 1- interp1 ( X_Kuiper , F_Kuiper , Kuiper ,'/quotesingle.ts1
next'/quotesingle.ts1);
p_Watson = 1- interp1 ( X_Watson , F_Watson , Watson ,'/quotesingle.ts1
next'/quotesingle.ts1);
p_AD = 1- interp1 ( X_AD , F_AD , AD ,'/quotesingle.ts1
next'/quotesingle.ts1);
p_KL = 1- interp1 ( X_KL , F_KL , KL ,'/quotesingle.ts1
next'/quotesingle.ts1);
p_JS = 1- interp1 ( X_JS , F_JS , JS ,'/quotesingle.ts1
next'/quotesingle.ts1);

p_vals = struct ('Kolmogorov_D',p_KolD,...
  'Cramer_von_Mises',p_CvM,...
  'Kuiper',p_Kuiper,...
  'Watson',p_Watson,...
  'Anderson_Darling',p_AD,...
  'Kullback_Leibler',p_KL,...
  'Jensen_Shannon',p_JS);

end

B.72  pvals_rl.m

function [p_vals] = pvals_rl(dataLength,sigma,Statistics,
cut,n,gap)
  %PVALS_RL estimates the p-values for the Rayleigh
  % distributions
  %
  % DATALENGTH is the number of entries in the data
  % SIGMA is the parameter in the Rayleigh distribution
  % STATISTICS is the statistics structure
  % CUT is the number of extreme data points removed
  % N is the precision
  % GAP is sampling interval
% P_VALS is a structure containing our estimated p-values

KolD = Statistics.Kolmogorov_D;
CvM = Statistics.Cramer_von_Mises;
Kuiper = Statistics.Kuiper;
Watson = Statistics.Watson;
AD = min(Statistics.Anderson_Darling,1E99);
KL = Statistics.Kullback_Leibler;
JS = Statistics.Jensen_Shannon;

num_MC = 10^n;

KolD_stat = zeros(1,num_MC);
CvM_stat = zeros(1,num_MC);
Kuiper_stat = zeros(1,num_MC);
Watson_stat = zeros(1,num_MC);
AD_stat = zeros(1,num_MC);
KL_stat = zeros(1,num_MC);
JS_stat = zeros(1,num_MC);

parfor i = 1:num_MC
    data = raylrnd(sigma,dataLength,1);
    data = sort(data);
    if cut > 0
        data(end-cut+1:end) = [];
    end
    PDF = raylpdf(data,sigma);
    CDF = raylcdf(data,sigma);
    thisfit = testStatistics(data,CDF,PDF,gap);
    KolD_stat(i) = thisfit.Kolmogorov_D;
    CvM_stat(i) = thisfit.Cramer_von_Mises;
    Kuiper_stat(i) = thisfit.Kuiper;
    Watson_stat(i) = thisfit.Watson;
    AD_stat(i) = thisfit.Anderson_Darling;
    KL_stat(i) = thisfit.Kullback_Leibler;
    JS_stat(i) = thisfit.Jensen_Shannon;
end

[F_KolD,X_KolD] = ecdf(KolD_stat);
[F_CvM,X_CvM] = ecdf(CvM_stat);
[F_Kuiper,X_Kuiper] = ecdf(Kuiper_stat);
[F_Watson,X_Watson] = ecdf(Watson_stat);
[F_AD,X_AD] = ecdf(AD_stat);
[F_KL,X_KL] = ecdf(KL_stat);
[F_JS,X_JS] = ecdf(JS_stat);
clearvars KolD_stat CvM_stat Kuiper_stat Watson_stat AD_stat KL_stat JS_stat

X_KolD = [-1E99; X_KolD(2:end);1E99];
X_CvM = [-1E99; X_CvM(2:end);1E99];
X_Kuiper = [-1E99; X_Kuiper(2:end);1E99];
X_Watson = [-1E99; X_Watson(2:end);1E99];
X_AD = [-1E99; X_AD(2:end);1E99];
X_KL = [-1E99; X_KL(2:end);1E99];
X_JS = [-1E99; X_JS(2:end);1E99];

F_KolD = [0; F_KolD(2:end);1];
F_CvM = [0; F_CvM(2:end);1];
F_Kuiper = [0; F_Kuiper(2:end);1];
F_Watson = [0; F_Watson(2:end);1];
F_AD = [0; F_AD(2:end);1];
F_KL = [0; F_KL(2:end);1];
F_JS = [0; F_JS(2:end);1];

p_KolD = 1- interp1 ( X_KolD , F_KolD ,KolD ,'/quotesingle.ts1
next'/quotesingle.ts1);

end

B.73  pvals_wb.m

function [ p_vals ] = pvals_wb(dataLength,a,b,Statistics,cut,n,gap)

%PVALS_WB estimates the p-values for the Weibull distributions
% DATALENGTH is the number of entries in the data
% A,B are the parameters in the Weibull distribution
% STATISTICS is the statistics structure
% CUT is the number of extreme data points removed
% N is the precision
% GAP is sampling interval
% P_VALS is a structure containing our estimated p-values
KolD = Statistics.Kolmogorov_D;
CvM = Statistics.Cramer_von_Mises;
Kuiper = Statistics.Kuiper;
Watson = Statistics.Watson;
AD = min(Statistics.Anderson_Darling,1E99);
KL = Statistics.Kullback_Leibler;
JS = Statistics.Jensen_Shannon;
num_MC = 10^n;
KolD_stat = zeros(1,num_MC);
CvM_stat = zeros(1,num_MC);
Kuiper_stat = zeros(1,num_MC);
Watson_stat = zeros(1,num_MC);
AD_stat = zeros(1,num_MC);
KL_stat = zeros(1,num_MC);
JS_stat = zeros(1,num_MC);
parfor i = 1:num_MC
    data = wblrnd(a,b,dataLength,1);
data = sort(data);
    if cut > 0
        data(end-cut+1:end) = [];
    end
    PDF = wblpdf(data,a,b);
    CDF = wblcdf(data,a,b);
    thisfit = testStatistics(data,CDF,PDF,gap);
    KolD_stat(i) = thisfit.Kolmogorov_D;
    CvM_stat(i) = thisfit.Cramer_von_Mises;
    Kuiper_stat(i) = thisfit.Kuiper;
    Watson_stat(i) = thisfit.Watson;
    AD_stat(i) = thisfit.Anderson_Darling;
    KL_stat(i) = thisfit.Kullback_Leibler;
    JS_stat(i) = thisfit.Jensen_Shannon;
end

[F_KolD,X_KolD] = ecdf(KolD_stat);
[F_CvM,X_CvM] = ecdf(CvM_stat);
[F_Kuiper,X_Kuiper] = ecdf(Kuiper_stat);
[F_Watson,X_Watson] = ecdf(Watson_stat);
[F_AD,X_AD] = ecdf(AD_stat);
[F_KL,X_KL] = ecdf(KL_stat);
```matlab
[F_JS, X_JS] = ecdf(JS_stat);
clearvars KolD_stat CvM_stat Kuiper_stat Watson_stat AD_stat KL_stat JS_stat
X_KolD = [-1E99; X_KolD(2:end);1E99];
X_CvM = [-1E99; X_CvM(2:end);1E99];
X_Kuiper = [-1E99; X_Kuiper(2:end);1E99];
X_Watson = [-1E99; X_Watson(2:end);1E99];
X_AD = [-1E99; X_AD(2:end);1E99];
X_KL = [-1E99; X_KL(2:end);1E99];
X_JS = [-1E99; X_JS(2:end);1E99];
F_KolD = [0; F_KolD(2:end);1];
F_CvM = [0; F_CvM(2:end);1];
F_Kuiper = [0; F_Kuiper(2:end);1];
F_Watson = [0; F_Watson(2:end);1];
F_AD = [0; F_AD(2:end);1];
F_KL = [0; F_KL(2:end);1];
F_JS = [0; F_JS(2:end);1];
p_KolD = 1- interp1(X_KolD,F_KolD,KolD,'next');
p_CvM = 1- interp1(X_CvM,F_CvM,CvM,'next');
p_Kuiper = 1- interp1(X_Kuiper,F_Kuiper,Kuiper,'next');
p_Watson = 1- interp1(X_Watson,F_Watson,Watson,'next');
p_AD = 1- interp1(X_AD,F_AD,AD,'next');
p_KL = 1- interp1(X_KL,F_KL,KL,'next');
p_JS = 1- interp1(X_JS,F_JS,JS,'next');
p_vals = struct('Kolmogorov_D',p_KolD,...
    'Cramer_von_Mises',p_CvM,...
    'Kuiper',p_Kuiper,...
    'Watson',p_Watson,...
    'Anderson_Darling',p_AD,...
    'Kullback_Leibler',p_KL,...
    'Jensen_Shannon',p_JS);
end
```

### B.74 rndLPM.m

```matlab
function lpm = rndLPM(nodes)
% RNDLPM returns a symmetric square randomly weighted
% selection matrix for
% the given number of nodes
%
% NODES is the number of nodes in the network
```
% LPM is the generated random matrix

sum_gm_a = 12.3109; %Parameter for row/column sum distribution
sum_gm_b = 0.0037; %Parameter for row/column sum distribution

indiv_gm_a = sum_gm_a /(2*(nodes-1)); %Term-by-term parameter
indiv_gm_b = sum_gm_b; %Term-by-term parameter

fullmat = gamrnd(indiv_gm_a+indiv_gm_a,indiv_gm_b,nodes,nodes); %Creates matrix
for i=1: nodes
    fullmat(i,i) = 0; %Sets diagonal to be 0
end

uppmat = triu(fullmat); %Eliminates low triangular half of matrix
unnorm_lpm = uppmat+uppmat'; %Creates symmetric matrix
lpm = unnorm_lpm/(sum(sum(unnorm_lpm))); %Normalises matrix
end

B.75 sampleCSV.m

function [] = sampleCSV(ontimes,offtimes,nodes/runtime,sampletime,filepath)
%SAMPLECSV Runs through the given structure, sampling every given interval
% and creating a CSV file containing this data
%
% ONTIMES is the on-time data created by a given MODEL function
% OFFTIMES is the off-time data created by a given MODEL function
% NODES is the node count of the network
% RUNTIME is the duration of the model
% SAMPLETIME is how often this data should be sampled for the CSV file

numint = floor(runtime/sampletime)+1;
timesteps = 0:sampletime:runtime;
massivematrix = [];
for i=1:nodes-1
    for j=i+1:nodes
ID_ref = sprintf('n%d_n%d', i,j);
thison = ontimes.(ID_ref);
thisoff = offtimes.(ID_ref);

thisindicator = zeros(1,numint);
parfor k=1:numint
    currenttime = (k-1)*sampletime;
    if sum(thison<=currenttime & thisoff>=
currenttime)
        thisindicator(k) = 1;
    end
end

thistimes = thisindicator.*timesteps;
(thistimes==0) = [];

thistimes = thistimes';
n = length(thistimes);
thisc2 = i*ones(n,1);
thisc3 = j*ones(n,1);
thisc4 = ones(n,1);
thisc5 = ones(n,1);
thisblock = [thistimes,thisc2,thisc3,thisc4,
            thisc5];
massivematrix = [massivematrix;thisblock];
end
end

[^,idx] = sort(massivematrix(:,1));
sortedbytime = massivematrix(idx,:);
csvwrite(filepath,sortedbytime)

B.76 sampleCSV2.m

function [] = sampleCSV2(times,nodes,runtime,sampletime,filepath)
%SAMPLECSV2 Runs through the given structure, sampling
% every given interval
% and creating a CSV file containing this data
%
% TIMES is the temporal data created by a given MODEL
% function
% NODES is the node count of the network
% RUNTIME is the duration of the model
% SAMPLETIME is how often this data should be sampled for
% the CSV file

numint = floor(runtime/sampletime)+1;
timesteps = 0:sampletime:runtime;
massivematrix = [];

% Runs through each node pair and samples every 20 seconds, appending this
% to MASSIVEMATRIX
for i = 1: nodes - 1
    for j = i + 1: nodes
        ID_ref = sprintf('n%d_n%d', i, j);
        vec = times.(ID_ref);

        if ~isempty(vec)
            ontimes = vec(1:2:end);
            offtimes = vec(2:2:end);

            thisindicator = zeros(1, numint);
            parfor k = 1: numint
                currenttime = (k-1)*sampletime;
                if sum(ontimes <= currenttime & offtimes >= currenttime)
                    thisindicator(k) = 1;
                end
            end
            thistimes = thisindicator.*timesteps;
            thistimes(thistimes == 0) = [];
            thistimes = thistimes';
            n = length(thistimes);
            thisc2 = i*ones(n, 1);
            thisc3 = j*ones(n, 1);
            thisc4 = ones(n, 1);
            thisc5 = ones(n, 1);
            thisblock = [thistimes, thisc2, thisc3, thisc4, thisc5];
            massivematrix = [massivematrix; thisblock];
        end
    end
end
[~, idx] = sort(massivematrix(:, 1)); % Extract idx of entries in time order
sortedbytime = massivematrix(idx, :); % Sort into time order
csvwrite(filepath, sortedbytime)
end

B.77 scaledmarginals.m

function [marginals, mean1, std1] = scaledmarginals(P, X)
%SCALDMARGINALS Computes scaled marginal distribution functions given
% n-dimensional array and vectors of x_i values
% P is an n-dimensional double array containing the values of the
% multidimensional distribution function
% X is a 1xn structure with the i-th entry containing the x values in the
% i-th dimension in a vector
% MARGINALS is a 1xn structure with the i-th entry containing the y values of
% the i-th marginal in a vector
% MEAN1 is a 1xn vector of the means in each variable
% STD1 is a 1xn vector of the standard deviations in each variable

fields = fieldnames(X);
varcount = length(fields);
marginals = struct();
mean1 = zeros(1,varcount);
std1 = zeros(1,varcount);
msize = size(P);
res = zeros(1,varcount);
parfor i=1:varcount
    res(i) = abs(X.(fields{i})(2)-X.(fields{i})(1));
end
resprod = prod(res);
inds = cell(1,varcount);
parfor i=1:varcount
    inds{i} = 1:(msize(i)-1);
end
for i=1:varcount
    fn = fields{i};
    mg = zeros(1,msize(i));
    parfor j=1:msize(i)
        thisinds = inds;
        thisinds{i} = j;
        thissheet = P(thisinds{:});
        thisentries = reshape(thissheet,1,numel(thissheet));
        thissum = sum(thisentries);
        mg(j) = thissum*resprod/res(i);
    end
    marginals.(fn) = mg;
E1 = sum(mg.*X.(fn))*res(i);
E2 = sum(mg.*X.(fn).*X.(fn))*res(i);
mean1(i) = E1;
std1(i) = sqrt(E2-(E1^2));

B.78 scaledposterior2_mlf2.m

function [X,P] = scaledposterior2_mlf2(pararange,data,SPpoints,priors)
%SCALEDPOSTERIOR2_MLF2 creates the multidimensional distribution posterior
%using the exact method
%
% PARARANGE is a 2x2 matrix containing the search range for mu and tau0
% DATA is the given data sample vector
% SPPOINTS is a 1x2 vector containing the number of grid steps in each parameter
% PRIORS is a 1x2 structure containing distribution objects for the priors
%
% P is an n-dimensional double array containing the values of the
%multidimensional distribution function
%X is a 1xn structure with the i-th entry containing the x values in the
%i-th dimension in a vector

X1 = linspace(pararange(1,1),pararange(1,2),SPpoints(1));
X2 = linspace(pararange(2,1),pararange(2,2),SPpoints(2));
unscaledvec = vpa(zeros(length(X1),length(X2)));
priordist_p1 = priors.p1;
priordist_p2 = priors.p2;
for i=1:length(X1)
    for j=1:length(X2)
        thispp = [X1(i),X2(j)];
p paras = pdf(priordist_p1,thispp(1))*pdf(priordist_p2,thispp(2));
        indivterms = vpa(zeros(1,length(data)));
        parfor k=1:length(data)
            indivterms(k) = -(1/data(k))*ml(-(data(k)/thispp(2))^thispp(1),thispp(1),0);
        end
        unscaledvec(i,j) = prod(indivterms)*pparas;
    end
end
```matlab
msize = size(unscaledvec);
inds = cell(1,size(pararange,1));
parfor i=1:size(pararange,1)
    inds{i} = 1:(msize(i)-1);
end
truc_usv = unscaledvec(inds{:});
usv_rs = reshape(truc_usv,1,[]);
vecsum = sum(usv_rs)*abs(X1(2)-X1(1))*abs(X2(2)-X2(1));
P = double(unscaledvec/vecsum);
X.p1 = X1;
X.p2 = X2;
```

B.79  sigma_for_mu_and_mean.m

```matlab
function [sigma] = sigma_for_mu_and_mean(m,mu)
%SIGMA_FOR_MU_AND_MEAN Calculates the variance of the
% with mean MU of the associated log-normal distribution
% with mean M
%
% M is the mean of the log-normal distribution
% MU is the mean of the associated normal distribution
% SIGMA is the calculated variance

if log(m)>mu
    sigma = sqrt(2*(log(m)-mu));
else
    sigma = 0;
end
end
```

B.80  stats_KLJS.m

```matlab
function fit = stats_KLJS(data1,data2,slice)
%STATS_KLJS takes two data sets and computes the Kullbeck
%Leibler and
% Jensen Shannon distances between them
%
% DATA1 is the first data set
% DATA2 is the second data set
% SLICE is the width of the intervals into which the data
% is sorted
```
FIT is a structure containing the Kullback-Leibler and Jensen-Shannon distances

```
mindata = min([data1, data2]);
maxdata = max([data1, data2]);
sliced = mindata:slice:maxdata;
Phist = histcounts(data1, sliced);
Qhist = histcounts(data2, sliced);

P = Phist/sum(Phist);
Q = Qhist/sum(Qhist);
P(P==0) = 1^-50;
Q(Q==0) = 1^-50;
KL = KLDiv(P, Q);
JS = JSDiv(P, Q);
fit = struct('Kullback_Leibler', KL, ...
             'Jensen_Shannon', JS);
```

B.81 testStatistics.m

```
function fit = testStatistics(X, Z, Zprime, gap)
    % TESTSTATISTICS returns a list of statistical measures comparing two
    % distributions
    % X is the list of values in the real world data
    % Z is the comparative CDF at these values
    % ZPRIME is the comparative PDF at these values
    % GAP is the sampling interval
    % FIT is a structure containing all statistical measurements chosen
    n = length(X);
    uni = unique(X);
    m = length(uni);
    if length(uni)==n
        fullecdf = 0:1/n:1;
    else
        h = hist(X, X);
        cumh = cumsum(h);
```
fullecdf = [0 cumh/n];
end
fullecdf = fullecdf';
lowerecdf = fullecdf(1:n);
uppercdf = fullecdf(2:n+1);
middlecdf = (lowerecdf+uppercdf)/2;

if gap>0
    Xr = floor(X/gap)*gap;
else
    Xr = X;
end
 [~,ia,~] = unique(Xr);
[hp,xp] = hist(Xr,Xr);
P = (hp/trapz(xp,hp));
P = P(ia);
Q = Zprime(ia');

Dplu = max(lowerecdf-Z);
Dmin = max(Z-uppercdf);
D = max(Dplu,Dmin);

CvM_vec = (Z-middlecdf).^2;
Wsq = sum(CvM_vec)+(1/(12*n));
V = Dplu+Dmin;
WatMod = n*(mean(Z) - 0.5)^2;
Usq = abs(Wsq-WatMod);

Zswitch = flipud(Z);
AD_vec = (2*middlecdf).*(log(Z)+log(1-Zswitch));
Asq = -sum(AD_vec) - n;
if Asq == inf
    Asq = 10^50;
elseif Asq == -inf
    Asq = -10^50;
end
P(P==inf) = 10^50;
P(P==-inf) = -10^50;
Q(Q==inf) = 10^50;
Q(Q==-inf) = -10^50;

KL = KLDiv(P,Q);
JS = JSDiv(P,Q);
fit = struct('Kolmogorov_D',D,...
B.82  triangleClosed.m

```matlab
function [frac,FiT,NTP] = triangleClosed(input_folder, input_filename)

%TRIANGLECLOSED calculates the fraction of triangle closures and times
% at which one is not possible, given a data file
%
% INPUT_FOLDER is the input folder, given as a string
% INPUT_FILENAME is the filename, given as a string
% FRAC is a vector containing the fraction of triangle closures up to
% that point
% FIT is the fraction of triangle closures over the entire data set
% NTP is the fraction of times at which no triangle activation is possible

iF = ['input/','input_folder'];
input = [iF,'/','input_filename'];
structure = '%f %f %f %s %s %s';

fid = fopen(input);
rawdata = textscan(fid,structure,'Delimiter','
fclose(fid);

%== Extract and Clean Data==%
data = cell2mat(rawdata);
data(:,1) = data(:,1)-data(1,1);
lowestID = min(min(data(:,2)),min(data(:,3)));
data(:,2) = data(:,2)-lowestID+1;
data(:,3) = data(:,3)-lowestID+1;
number_rows = size(data,1);
parfor i=1:number_rows
    thisrow = data(i,:);
    col2 = thisrow(1,2);
    col3 = thisrow(1,3);
    if col2 > col3
        thisrow(1,2) = col3;
    end
```

'Cramer_von_Mises',Wsq,...
'Kuiper',V,...
'Watson',Usq,...
'Anderson_Darling',Asq,...
'Kullback_Leibler',KL,...
'Jensen_Shannon',JS);
thisrow(1,3) = col2;
data(i,:) = thisrow;

end

all_IDs = [data(:,2); data(:,3)];
all_active = unique(all_IDs);
num_people = size(all_active,1);
data2 = data(:,2);
data3 = data(:,3);
for i=1:num_people
    oldID = all_active(i);
data2(data2==oldID) = -i;
data3(data3==oldID) = -i;
end
data(:,2) = -data2;
data(:,3) = -data3;
times = unique(data(:,1));
numbertimes = length(times);
triangleactivations = zeros(1,numbertimes);
totalactivations = zeros(1,numbertimes);
notrianglespossible = zeros(1,numbertimes);
for i=1:numbertimes
    thistime = times(i);
    lasttime = thistime-20;
    thisadj = zeros(num_people);
    lastadj = zeros(num_people);
    thisdata = data(data(:,1)==thistime,2:3);
    lastdata = data(data(:,1)==lasttime,2:3);
    thisamount = size(thisdata,1);
    lastamount = size(lastdata,1);
    for j=1:thisamount
        person1 = thisdata(j,1);
        person2 = thisdata(j,2);
        thisadj(person1,person2) = 1;
        thisadj(person2,person1) = 1;
    end
    for j=1:lastamount
        person1 = lastdata(j,1);
        person2 = lastdata(j,2);
        lastadj(person1,person2) = 1;
        lastadj(person2,person1) = 1;
    end
    changes = thisadj-lastadj;
    newadj = changes==1;
    paths2 = lastadj^2;
    tricomp = paths2==1;
    activationbinary = newadj & tricomp;
triangleactivations(i) = sum(sum(activationbinary))/2;
totalactivations(i) = sum(sum(newadj))/2;
m1 = (thisadj^2)==1;
m2 = m1-thisadj;
m3 = m2==1;
for j=1:size(num_people)
    m3(j,j)=0;
end
possibletriangles = sum(sum(m3))/2;
if possibletriangles==0
    notrianglespossible(i)=1;
end
triangleactivationscount = cumsum(triangleactivations);
totalactivationscount = cumsum(totalactivations);
FiT = triangleactivationscount./totalactivationscount;
frac = FiT(end);
NTP = sum(notrianglespossible)/numbertimes;
end

B.83 validate1.m

function [] = validate1(nodes,runtime,filepath,
    initialdata,ontimes,oftimes)
    %VALIDATE1 produces a simulation validation run using
    % method 1
    %
    % NODES is the nodes count from the underlying data as a
    vector
    % RUNTIME is the simulation length in seconds
    % FILEPATH is the save directory given as a string
    % INITIALDATA is the initialisation times from the
    % underlying data as a
    % vector
    % ONTIMES is the on-times from the underlying data as a
    % vector
    % OFFTIMES is the on-times from the underlying data as a
    vector
    cut = 20;
    preruntime = zeros(nodes);
    switchon = emprand(initialdata,nodes);
    startthings = switchon-preruntime;
    initial = zeros(nodes);

    cut = 20;
    preruntime = zeros(nodes);
    switchon = emprand(initialdata,nodes);
    startthings = switchon-preruntime;
    initial = zeros(nodes);

ontimes = struct();
oftimes = struct();

for i=1:nodes-1
    for j=i+1:nodes
        init = initial(i,j);
        currenttime = startthings(i,j);
        if init == 0
            if startthings(i,j)<runtime
                thisoff = [startthings(i,j)];
            else
                thisoff = [];  
            end
            thison = [];
            while currenttime<runtime
                thisoffduration = emprand(offdata);
                switch_on = currenttime+thisoffduration;
                thisonduration = emprand(ondata);
                switch_off = switch_on+thisonduration;
                if switch_on<runtime
                    thison = [thison,switch_on];
                    if switch_off<runtime
                        thisoff = [thisoff,switch_off];
                    else
                        thisoff = [thisoff,runtime];
                    end
                else
                    thison = [thison,runtime];
                end
                currenttime = switch_off;
            end
        elseif init == 1
            thisoff = [];
            if startthings(i,j)<runtime
                thison = [startthings(i,j)];
            else
                thison = [];
            end
            while currenttime<runtime
                thisonduration = emprand(ondata);
                switch_off = currenttime+thisonduration;
                thisoffduration = emprand(offdata);
                switch_on = switch_off+thisoffduration;
                if switch_off<runtime
                    thisoff = [thisoff,switch_off];
                    if switch_on<runtime
                        thison = [thison,switch_on];
                else
                    thison = [thison,runtime];
                end
            currenttime = switch_off;
        end
```matlab
else
    thison = [thison, runtime];
end
else
    thisoff = [thisoff, runtime];
end
currenttime = switch_on;
end
firstonIDX = find(thison > 0, 1);
firstoffIDX = find(thisoff > 0, 1);
firston = thison(firstonIDX);
firstoff = thisoff(firstoffIDX);
thison(thison < 0) = [];
thisoff(thisoff < 0) = [];
thisoff(thisoff == 0) = [];
thison(thison == runtime) = [];
thisoff(thisoff > runtime) = [];
if firston > firstoff
    thison = [switchon(i, j), thison];
end
ID_ref = sprintf('/quotesingle.ts1\n%d\n%d/quotesingle', i, j);
ontimes.(ID_ref) = thison;
offtimes.(ID_ref) = thisoff;
end
sampleCSV(ontimes, offtimes, nodes, runtime, cut, filepath);
end

B.84 validate2a.m

function [] = validate2a(runtime, probM, filepath, IETdata, onData)

%VALIDATE2A produces a simulation validation run using
% RUNTIME is the simulation length in seconds
% PROBM is the extracted link preference selection matrix
% FILEPATH is the save directory given as a string
% IETTIMES is the interevent times from the underlying
% ONTIMES is the on-times from the underlying data as a vector
% CUT is the sampling interval in seconds (DEFAULT: 20)
cut = 20;
```
nodes = size(probM,1);
linkprobmatrix = probM;

initialoff = zeros(nodes);
%INITIALOFF is the initial state for the network given as an adjacency matrix

times = struct();
%TIMES will end up having entries labeled nX_nY with entries showing the on and off times for the link X-Y as a single vector.
for i = 1:nodes - 1
    for j = i + 1:nodes
        ID_ref = sprintf('%d_%d',i,j);
        if initialoff(i,j) > 0
            times.(ID_ref) = [0, initialoff(i,j)];
        else
            times.(ID_ref) = [];
        end
    end
end

currenttime = emprand(IETdata);
%CURRENTTIME should be set to the desired distribution for on interevent
%timings

while currenttime < runtime
    thislinkmatrix = linkprobmatrix;
    accept = 0;
    while accept == 0
        [ri,rj] = chooselink(thislinkmatrix);
        ID_ref = sprintf('%d_%d',ri,rj);
        vec = times.(ID_ref);
        if isempty(vec) || currenttime > vec(end)
            %Ensure link is not already active
            accept = 1;
            ontime = currenttime;
            offtime = currenttime + emprand(onData);
            vec = [vec, ontime, offtime];
            %Append to entry in TIMES
            times.(ID_ref) = vec;
        end
    end
    IET = emprand(IETdata);
    %IET should be set to the desired distribution for on interevent
% timings
    currenttime = currenttime + IET;
end
sampleCSV2(times, nodes, runtime, cut, filepath);
% Passes data to sampleCSV.m for sampling and conversion
to CSV file
end

B.85 validate2b.m

function [] = validate2b(runtime, probM, filepath, IETdata, onData)
    % VALIDATE2A produces a simulation validation run using
    % method 2b
    %
    % RUNTIME is the simulation length in seconds
    % PROBM is the extracted link preference selection matrix
    % FILEPATH is the save directory given as a string
    % IETTIMES is the interevent times from the underlying
    % data as a vector
    % ONTIMES is the on-times from the underlying data as a
    % vector
    cut = 20;
    % CUT is the sampling interval in seconds (DEFAULT: 20)
    nodes = size(probM,1);
    linkprobmatrix = probM;
    initialoff = zeros(nodes);
    % INITIALOFF is the initial state for the network given as an
    % matrix
    times = struct();
    % TIMES will end up having entries labeled nX_nY with
    % entries showing the on
    % and off times for the link X-Y as a single vector.
    for i = 1:nodes-1
        for j = i+1:nodes
            ID_ref = sprintf('%d_%d', i, j);
            if initialoff(i, j) > 0
                times.(ID_ref) = [0, initialoff(i, j)];
            else
                times.(ID_ref) = [];
            end
        end
    end
currenttime = emprand(IETdata);
% CURRENTTIME should be set to the desired distribution for on interevent
% timings

triangleActivations = 0.0556;
estFailure = 0.1316;
adjustedThreshold = triangleActivations/(1-estFailure);

while currenttime<runtime
    flip = unifrnd(0,1);
    if flip<adjustedThreshold
        [~,m_lpm] = extractTriangles(times,nodes,currenttime,linkprobmatrix);
        if sum(sum(m_lpm))~=0
            thislinkmatrix = m_lpm;
        else
            thislinkmatrix = linkprobmatrix;
        end
    else
        thislinkmatrix = linkprobmatrix;
    end
    accept = 0;
cycles = 0;
    while accept == 0
        [ri,rj] = chooselink(thislinkmatrix);
        ID_ref = sprintf('%d_%d',ri,rj);
        vec = times.(ID_ref);
        if isempty(vec)||(currenttime>vec(end)
            % Ensure link is not already active
            accept = 1;
            ontime = currenttime;
            offtime = currenttime+emprand(onData);
            vec = [vec,ontime,offtime];
            % Append to entry in TIMES
            times.(ID_ref) = vec;
        end
        if cycles>100
            thislinkmatrix = linkprobmatrix;
        else
            cycles = cycles+1;
        end
    end
    IET = emprand(IETdata);
% IET should be set to the desired distribution for on interevent
% timings
currenttime = currenttime + IET;
end

call sampleCSV2(times, nodes, runtime, cut, filepath);

% Passes data to sampleCSV.m for sampling and conversion to CSV file
end

### B.86 validation.m

```matlab
function [] = validation(folder, count, timelength)

% VALIDATION creates and saves a latex ready file containing validation
% counts for each model and metric
%
% FOLDER is the location of data, given as a string
% COUNT is the number of validation samples to generate
% TIMELENGTH is the simulation time for the validation

timestamp = datestr(now, 'yyyyymmdTHHMMSs');
dir_ref = ['input/output_', timestamp];
mkdir(dir_ref);

iF = ['input/', folder];
toExtract = [iF, '*.csv'];

fileData = dir(toExtract);
fileName = {fileData.name};
fileName = fileName(contains(fileName, '_'));

accept5matrix = zeros(10, 3);

for i = 1:length(fileName)
    currentFile = fileName{i};
    currentFilepath = [iF, '/', currentFile];
    currentClean = strrep(currentFile, '.', '');
    currentClean = strrep(currentClean, '-', '');
    currentParent = [dir_ref, '/validation'];
    currentFolder = [currentParent, '/', currentClean];
    currentFolderR = [currentFolder, '/original'];
    currentFolderM1 = [currentFolder, '/model1'];
    currentFolderM2a = [currentFolder, '/model2a'];
    currentFolderM2b = [currentFolder, '/model2b'];
    mkdir(currentFolderR);
    mkdir(currentFolderM1);
    mkdir(currentFolderM2a);
    mkdir(currentFolderM2b);
    copyfile(currentFilepath, currentFolderR);
```
currentData = pullData(iF, currentFile, '%f %f %f %*s %*s');
currentInitial = currentData.
  FirstActivationTimes_data;
currentInitial(currentInitial==Inf) = timelength;
currentOnTimes = currentData.InteractionTimes_data;
currentOffTimes = currentData.NoContactTimes_data;
currentIETimes = currentData.IntereventTimes_data;
currentStudents = currentData.NumberStudents_data;
currentMatrix = EAP_matrix(folder, currentFile);
for j=1:count
  outputfile = ['run_', num2str(j), '.csv'];
  fp1 = [currentFolderM1, '/', outputfile];
  fp2a = [currentFolderM2a, '/', outputfile];
  fp2b = [currentFolderM2b, '/', outputfile];
  validate1(currentStudents, timelength, fp1, currentInitial, currentOnTimes, currentOffTimes);
  validate2a(timelength, currentMatrix, fp2a, currentIETimes, currentOnTimes);
  validate2b(timelength, currentMatrix, fp2b, currentIETimes, currentOnTimes);
end
dist1 = calculateDistance(currentFolderM1(7:end), currentFolderR(7:end));
dist2a = calculateDistance(currentFolderM2a(7:end), currentFolderR(7:end));
dist2b = calculateDistance(currentFolderM2b(7:end), currentFolderR(7:end));
currentAccept5 = zeros(10,3);
currentAccept5(1,1) = dist1.ActiveLinks.accept5;
currentAccept5(2,1) = dist1.OnTimes.accept5;
currentAccept5(3,1) = dist1.ActivityPot.accept5;
currentAccept5(4,1) = dist1.OffTimes.accept5;
currentAccept5(5,1) = dist1.ActiveNodes.accept5;
currentAccept5(6,1) = dist1.CompCount.accept5;
currentAccept5(7,1) = dist1.GCC.accept5;
currentAccept5(8,1) = dist1.CompNodes.accept5;
currentAccept5(9,1) = dist1.CompEdges.accept5;
currentAccept5(10,1) = dist1.TriangleCount.accept5;
currentAccept5(1,2) = dist2a.ActiveLinks.accept5;
currentAccept5(2,2) = dist2a.OnTimes.accept5;
currentAccept5(3,2) = dist2a.ActivityPot.accept5;
currentAccept5(4,2) = dist2a.OffTimes.accept5;
currentAccept5(5,2) = dist2a.ActiveNodes.accept5;
currentAccept5(6,2) = dist2a.CompCount.accept5;
currentAccept5(7,2) = dist2a.GCC.accept5;
currentAccept5(8,2) = dist2a.CompNodes.accept5;
currentAccept5(9,2) = dist2a.CompEdges.accept5;
currentAccept5(10,2) = dist2a.TriangleCount.accept5;
currentAccept5(1,3) = dist2b.ActiveLinks.accept5;
currentAccept5(2,3) = dist2b.OnTimes.accept5;
currentAccept5(3,3) = dist2b.ActivityPot.accept5;
currentAccept5(4,3) = dist2b.OffTimes.accept5;
currentAccept5(5,3) = dist2b.ActiveNodes.accept5;
currentAccept5(6,3) = dist2b.CompCount.accept5;
currentAccept5(7,3) = dist2b.GCC.accept5;
currentAccept5(8,3) = dist2b.CompNodes.accept5;
currentAccept5(9,3) = dist2b.CompEdges.accept5;
currentAccept5(10,3) = dist2b.TriangleCount.accept5;
accept5matrix = accept5matrix + currentAccept5;
end

lines = 13;
tobuild = cell(lines,1);
tobuild{01} = '\begin{tabular}{|c|c|c|c|} \cline{2-4}';
tobuild{02} = '& \textbf{Model 1} & \textbf{Model 2a} & \textbf{Model 2b}\\\n\hline';
tobuild{03} = ['\begin{multicolumn}{c}{|l|} \begin{tabular}{|c|}
\hline
\end{tabular} \end{multicolumn}' & & num2str(accept5matrix(1,1)),' & & num2str(accept5matrix(1,2)),' & & num2str(accept5matrix(1,3))&
\\\n\hline';
tobuild{04} = ['\begin{multicolumn}{c}{|l|} \begin{tabular}{|c|}
\hline
\end{tabular} \end{multicolumn}' & & num2str(accept5matrix(2,1)),' & & num2str(accept5matrix(2,2)),' & & num2str(accept5matrix(2,3))&
\\\n\hline';
tobuild{05} = ['\begin{multicolumn}{c}{|l|} \begin{tabular}{|c|}
\hline
\end{tabular} \end{multicolumn}' & & num2str(accept5matrix(3,1)),' & & num2str(accept5matrix(3,2)),' & & num2str(accept5matrix(3,3))&
\\\n\hline';
tobuild{06} = ['\begin{multicolumn}{c}{|l|} \begin{tabular}{|c|}
\hline
\end{tabular} \end{multicolumn}' & & num2str(accept5matrix(4,1)),' & & num2str(accept5matrix(4,2)),' & & num2str(accept5matrix(4,3))&
\\\n\hline';
tobuild{07} = ['\begin{multicolumn}{c}{|l|} \begin{tabular}{|c|}
\hline
\end{tabular} \end{multicolumn}' & & num2str(accept5matrix(5,1)),' & & num2str(accept5matrix(5,2)),' & & num2str(accept5matrix(5,3))&
\\\n\hline';
tobuild{08} = ['\begin{multicolumn}{c}{|l|} \begin{tabular}{|c|}
\hline
\end{tabular} \end{multicolumn}' & & num2str(accept5matrix(6,1)),' & & num2str(accept5matrix(6,2)),' & & num2str(accept5matrix(6,3))&
\\\n\hline';
tobuild{09} = ['\begin{multicolumn}{c}{|l|} \begin{tabular}{|c|}
\hline
\end{tabular} \end{multicolumn}' & & num2str(accept5matrix(7,1)),' & & num2str(accept5matrix(7,2)),' & & num2str(accept5matrix(7,3))&
\\\n\hline';
accept5matrix(7,3),'\$
\hline
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
\$
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Example of a categorical SIR model with a fixed population such as in [101] - here, $\beta$ is the infection rate and $\gamma$ is the recovery rate.</td>
</tr>
<tr>
<td>1.2</td>
<td>Example of a categorical SIR model with a dynamic population and vaccination such as in [42] - here, $\beta$ is the infection rate, $\gamma$ is the recovery rate, $\rho$ is the vaccination rate, $\lambda$ is the birth rate and $\nu$ is the death rate.</td>
</tr>
<tr>
<td>1.3</td>
<td>Example contact network generated by a random walk study in Canberra, Australia, with nodes placed in relation to the locations within the city where participants were resident (adapted from Figure 2 of [109]).</td>
</tr>
<tr>
<td>1.4</td>
<td>Example of a square grid lattice network with 36 nodes.</td>
</tr>
<tr>
<td>1.5</td>
<td>Example of a small-world network.</td>
</tr>
<tr>
<td>1.6</td>
<td>Example of small-world network construction. In (a), a random network is created. In (b), nodes $a$ and $d$ are replaced by densely connected sets $A$ and $D$. In (c), an edge between $A$ and $b$ is introduced to replace the original edge $ab$. In (d), the multiple edges between $A$ and $D$ are introduced to replace the original edge $ad$. (From [220]).</td>
</tr>
<tr>
<td>1.7</td>
<td>Examples of a scale-free network, showing the preservation of the underlying structure between various sizes. (From [197]).</td>
</tr>
<tr>
<td>1.8</td>
<td>A variety of common directed and undirected network motifs.</td>
</tr>
<tr>
<td>1.9</td>
<td>Example of a HRG production rule (adapted from [56]).</td>
</tr>
<tr>
<td>1.10</td>
<td>A clique tree of a simple network (adapted from [31]).</td>
</tr>
<tr>
<td>1.11</td>
<td>Example CCDFs for a Mittag-Leffler distribution.</td>
</tr>
<tr>
<td>3.1</td>
<td>PDF produced using Convolve (bar) vs. PDF from Monte-Carlo Simulation (line) for $N_{2000}$.</td>
</tr>
<tr>
<td>3.2</td>
<td>Long term behaviours for original data and models - Size of nodes &amp; transparency of links represent relative activities (see last paragraph of the opening of section 3.3 for full explanation and the relevant subsections of section 3.3 and section 3.4 for an analysis of these results). One immediate observation is that Model 1 homogenises much faster - note the limited number of darker links.</td>
</tr>
<tr>
<td>3.3</td>
<td>Selected results for Model 1. Simulated data is represented by crosses whereas observed data is represented by dotted lines. Each colour represents a different simulation or data set. See subsection 3.3.1 for full explanation. Additional figures are available in the supplemental materials.</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>3.4</td>
<td>Selected results for Model 2a. Simulated data is represented by crosses whereas observed data is represented by dotted lines. Each colour represents a different simulation or data set. See subsection 3.3.2 for full explanation. Additional figures are available in the supplemental materials.</td>
</tr>
<tr>
<td>3.5</td>
<td>Selected results for Model 2b. Simulated data is represented by crosses whereas observed data is represented by dotted lines. Each colour represents a different simulation or data set. See subsection 3.3.3 for full explanation. Additional figures are available in the supplemental materials.</td>
</tr>
<tr>
<td>3.6</td>
<td>Selected results for Model 2c. Simulated data is represented by crosses whereas observed data is represented by dotted lines. Each colour represents a different simulation or data set. See subsection 3.3.4 for full explanation. Additional figures are available in the supplemental materials.</td>
</tr>
<tr>
<td>4.1</td>
<td>Example MCMC chains for varying choices of $\sigma$ with same distribution of given data, initial point and scales showing parameter values against MCMC step.</td>
</tr>
<tr>
<td>4.2</td>
<td>Sample MCMC chains for $\mu = 0.5$ and $\tau_0 = 1$. Varying colours in each image represent each of the parallel chains. Highlighted region shows equilibration period. Solid red lines represent estimated parameters with red dashed lines showing one standard deviation either side of this. Solid black lines represent true parameters.</td>
</tr>
<tr>
<td>4.3</td>
<td>Sample multidimensional posterior PDF plots for $\mu = 0.5$ and $\tau_0 = 1$ using exact and MCMC methods. Red and blue lines represent estimated parameters. Black lines represent true parameters.</td>
</tr>
<tr>
<td>4.4</td>
<td>Sample marginal posterior PDF plots and histograms of MCMC chains for $\mu = 0.5$ and $\tau_0 = 1$. Prior densities are shown as yellow horizontal lines. Red vertical line shows estimated mean using MCMC method (with red shaded region being one standard deviation either side of this). Blue vertical line shows estimated mean using exact method (with blue shaded region being one standard deviation either side of this). Black vertical line shows true parameters.</td>
</tr>
<tr>
<td>4.5</td>
<td>Sample CCDF plots showing generated data and distributions using true and estimated parameters for $\mu = 0.5$ and $\tau_0 = 1$. Black line shows true parameters, red lines show parameters estimated using MCMC method, blue lines show parameters estimated using exact method. Solid lines represent estimates, dashed lines represent one standard deviation from estimates, dotted lines represent two standard deviations from estimates.</td>
</tr>
<tr>
<td>4.6</td>
<td>$E_\mu(z)$ plotted again $\mu$ showing relationship. Different curves show this relationship for various values between $z = -1.1$ (lowest curve, blue) and $z = -0.9$ (highest curve, red).</td>
</tr>
<tr>
<td>4.7</td>
<td>Method for approximating $\tau_0$, showing ECCDF (blue), $\exp(-1)$ and 0.5 (orange), first values outside of this region (yellow) and the resulting search range (purple) - here $\tau_0 = 1$ which is captured within the search range.</td>
</tr>
</tbody>
</table>
4.8 Earthquake location data

4.9 MCMC chains for earthquake data. Varying colours in each image represent each of the parallel chains. Highlighted region shows equi-libration period. Solid red lines represent estimated parameters with red dashed lines showing one standard deviation either side of this.

4.10 Multidimensional posterior PDF plots for earthquake data using exact and MCMC methods. Red and blue lines represent estimated parameters.

4.11 Marginal posterior PDF plots and histograms of MCMC chains for earthquake data. Prior densities are shown as yellow horizontal lines. Red vertical line shows estimated mean using MCMC method (with region being one standard deviation either side of this). Blue vertical line shows estimated mean using exact method (with region being one standard deviation either side of this).

4.12 CCDF plot showing generated data and distributions using estimated parameters for earthquake data. Red lines show parameters estimated using MCMC method, blue lines show parameters estimated using exact method. Solid lines represent estimates, dashed lines represent one standard deviation from estimates, dotted lines represent two standard deviations from estimates.

4.13 Sample CCDF plots showing generated data and distributions using true and estimated parameters for $\mu = 0.5$ and $\tau_0 = 1$. Black line shows true parameters, red lines show parameters estimated using ABC method. Solid lines represent estimate, dashed lines represent one standard deviation from estimate, dotted lines represent two standard deviations from estimate.

4.14 CCDFs comparing fits generated by Stage and Fedotov (black) to those generated by the ABC (red) and the original summary data (blue).
List of Tables

2.1 Mean and variance of observed parameters .......................... 23
2.2 Chosen parameters .................................................... 23
3.1 Acceptances of $H_0$ (see equation 3.1) at 1% and 5% Levels (max: 190)
    (see subsection 3.1.1 for full explanation) .......................... 24
3.2 Summary of model dependencies (see subsection 3.2.5 for full expla-
    nation, subsection 3.2.2 for the definitions of $M$ and subsection 3.2.3
    for the definitions of $u_f$) .......................................... 33
3.3 Selected two-sample Kolmogorov-Smirnov distances (see section 3.3
    for full explanation) .................................................. 34
3.4 Acceptances of $H_0$ (see equation 3.8) at 5% Level (max: 1000) (see
    section 3.3 for full explanation) ...................................... 35
3.5 Validation Acceptances of $H_0$ (see equation 3.9) at 5% Level (max:
    1000) (see subsection 3.5 for full explanation) ....................... 44
4.1 Mean and standard deviation estimates for the MCMC method ........ 51
4.2 Mean and standard deviation estimates for the exact method .......... 56
4.3 Mean and standard deviation estimates for the MCMC method for
    various data sample sizes .............................................. 56
4.4 Mean and standard deviation estimates for the exact method for var-
    ious data sample sizes ................................................. 57
4.5 Mean and standard deviation estimates for earthquake data .......... 59
4.6 Mean and standard deviation estimates for the ABC method .......... 62
4.7 Mean and standard deviation estimates for the ABC method for var-
    ious data sample sizes ............................................... 64
4.8 Mean and standard deviation estimates for recidivism of drug ex-
    prisoners (RDEP) and duration of human residence (DHR) .......... 64
5.1 AIC, BIC and $L^2$ analysis for Mittag-Leffler data sets. For simplicity,
    the set of generated data consisting of $n$ points with true parameters
    $\mu^*$ and $\tau_0^*$ is represented as $G_n(\mu^*, \tau_0^*)$ .................. 69
Index of Names

Akaike, Hirotugu, 66, 68, 69, 88
Albert, Réka, 9
Anderson, Theodore Wilbur, 19, 21
Barabási, László, 9
Barrat, Alain, iv
Bayes, Thomas, iv, 21, 46, 49, 59
Beaumont, Mark A., 49
Bishop, Christopher Michael, 46
Cattuto, Ciro, iv
Cramér, Harald, 19, 21
Darling, Donald Allan, 19, 21
Eames, Ken T.D., 3
Erdős, Paul, 4, 6, 8
Euclid, 66
Fedotov, Sergei, iv, 64, 65
Fisher, Ronald Aylmer, 68
Fournet, Julie, iv
Frost, Wade Hampton, 2
Fulger, Daniel, 236
Garrappa, Roberto, 223
Gauss, Carl Friedrich, 14, 48
Gemmeto, Valerio, iv
Georgiou, Nicos, iv, 1, 13, 15
Germano, Guido, 236
Gorenflo, Rudolf, 59
Granovetter, Mark, 3
Hastings, Wilfred Keith, 47
Hawkes, Alan Geoffrey, 59
Jensen, Johan, 19, 22, 91
Kacenak, Martin, 232
Keeling, Matt J., 3
Kertész, János, iv
Kiss, István Zoltán, iv, 1, 13, 15
Kolmogorov, Andrey, 19, 21, 25, 34, 35, 37, 39, 40, 42, 91, 92, 96
Kuiper, Nicolaas, 19, 21
Kullback, Solomon, 19, 21, 22, 91
Lambert, Ben, 18
Larremore, Daniel, 251
Leibler, Richard, 19, 21, 22, 91
Lord Rayleigh, see Strutt, John William
Mainardi, Francesco, 59
Markov, Andrey, 1, 2, 15, 47, 49, 52
Mastrandrea, Rossana, iv
Metropolis, Nicholas, 47, 48
Mittag-Leffler, Gösta, 1, 13, 15, 19, 20, 46, 50, 56, 68, 91, 92, 98, 69, 87, 88, 94
Pareto, Vilfredo, 14, 15, 19
Podlubny, Igor, 232
Poisson, Siméon Denis, 4, 8, 13
Reed, Lowell, 2
Rényi, Alfréd, 4, 6, 8
Rice, Stephen Oswald, 14
Roychowdhury, Sugata, 9
Scalas, Enrico, iv, 1, 13, 15, 59, 236
Schelling, Thomas C., 3
Shannon, Claude, 19, 22, 91
Simkin, Mikhail, 9
Simon, Herbert Alexander, 9
Smirnov, Nikolai, 20, 25, 34, 35, 37, 39, 40, 42, 91, 92, 96
Stage, Helena, iv, 64, 65
Strogatz, Steven Henry, 7
<table>
<thead>
<tr>
<th>Name</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strutt, John William</td>
<td>14, 15, 19</td>
</tr>
<tr>
<td>von Mises, Richard</td>
<td>19, 21</td>
</tr>
<tr>
<td>Watson, Geoffrey Stuart</td>
<td>19, 21</td>
</tr>
<tr>
<td>Watts, Duncan James</td>
<td>7</td>
</tr>
<tr>
<td>Weibull, Waloddi</td>
<td>15, 19</td>
</tr>
<tr>
<td>Whiten, William</td>
<td>162, 248, 255</td>
</tr>
<tr>
<td>Yule, George Udny</td>
<td>9</td>
</tr>
</tbody>
</table>
### Index of Subjects

<table>
<thead>
<tr>
<th>Subject</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>activation time, see initialisation time</td>
<td></td>
</tr>
<tr>
<td>active links</td>
<td>18, 37, 40, 45</td>
</tr>
<tr>
<td>active nodes</td>
<td>18, 33, 37, 40, 45</td>
</tr>
<tr>
<td>activity potential</td>
<td></td>
</tr>
<tr>
<td>link</td>
<td>18, 28, 37</td>
</tr>
<tr>
<td>node</td>
<td>18, 30, 33, 37, 39, 40, 43, 70</td>
</tr>
<tr>
<td>approximate Bayesian computation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>49, 50, 51, 61, 65</td>
</tr>
<tr>
<td>distance</td>
<td></td>
</tr>
<tr>
<td>$L^2$</td>
<td>66, 68, 69, 88</td>
</tr>
<tr>
<td>statistical, see EDF statistic</td>
<td></td>
</tr>
<tr>
<td>divergence</td>
<td></td>
</tr>
<tr>
<td>Jensen-Shannon</td>
<td>19, 22, 91</td>
</tr>
<tr>
<td>Kullback-Leibler</td>
<td>19, 21, 22, 91</td>
</tr>
<tr>
<td>EDF statistic</td>
<td>19, 39, 40, 43, 44, 66</td>
</tr>
<tr>
<td>Anderson-Darling</td>
<td>19, 21</td>
</tr>
<tr>
<td>Cramér-von-Mises</td>
<td>19, 21</td>
</tr>
<tr>
<td>Kolmogorov-D</td>
<td>19, 21, 25, 35, 37, 39, 40</td>
</tr>
<tr>
<td>Kolmogorov-Smirnov, see Kolmogorov-D</td>
<td></td>
</tr>
<tr>
<td>Kuiper</td>
<td>19, 21</td>
</tr>
<tr>
<td>quadratic</td>
<td>20, 21</td>
</tr>
<tr>
<td>two-sample</td>
<td>20, 25, 35, 37, 39, 40</td>
</tr>
<tr>
<td>Watson</td>
<td>19, 21</td>
</tr>
<tr>
<td>exact Bayesian estimate</td>
<td>40, 47</td>
</tr>
<tr>
<td></td>
<td>50, 56, 58, 61, 94</td>
</tr>
<tr>
<td>global clustering coefficient</td>
<td>18, 33</td>
</tr>
<tr>
<td></td>
<td>37, 40, 13, 45</td>
</tr>
<tr>
<td>information criteria</td>
<td></td>
</tr>
<tr>
<td>Akaike</td>
<td>66, 69, 88</td>
</tr>
<tr>
<td>Bayesian</td>
<td>66, 69, 88</td>
</tr>
<tr>
<td>initialisation time, see on-duration</td>
<td></td>
</tr>
<tr>
<td>interaction time, see on-duration</td>
<td></td>
</tr>
<tr>
<td>interevent time, see on-duration</td>
<td></td>
</tr>
<tr>
<td>link spread</td>
<td>37, 39, 40, 43, 93</td>
</tr>
<tr>
<td>Markov-chain Monte Carlo method,</td>
<td></td>
</tr>
<tr>
<td>model</td>
<td></td>
</tr>
<tr>
<td>version 1</td>
<td>25, 26, 33, 35, 40, 42, 45</td>
</tr>
<tr>
<td>version 2a</td>
<td>28, 33, 35, 36, 39, 40, 42, 45</td>
</tr>
<tr>
<td>version 2b</td>
<td>28, 30, 33, 35, 36, 40, 42, 45</td>
</tr>
<tr>
<td>version 2c</td>
<td>30, 33, 36, 37, 42, 45</td>
</tr>
<tr>
<td>node degree</td>
<td>8, 12, 18</td>
</tr>
<tr>
<td>non-Markovianity</td>
<td>1, 15</td>
</tr>
<tr>
<td>off-duration</td>
<td>19, 22, 23, 25, 27, 37, 40, 44</td>
</tr>
<tr>
<td>on-duration</td>
<td>19, 22, 23, 25, 29, 37, 40, 44</td>
</tr>
<tr>
<td>probability distribution</td>
<td>13, 19, 25</td>
</tr>
<tr>
<td>exponential</td>
<td>13, 15</td>
</tr>
<tr>
<td>gamma</td>
<td>13, 14, 30</td>
</tr>
<tr>
<td>generalised Pareto</td>
<td>14, 15, 19</td>
</tr>
<tr>
<td>heavy-tailed</td>
<td>1, 13, 15, 20, 21, 46</td>
</tr>
<tr>
<td>log-normal</td>
<td>1, 14, 19, 22, 26, 27, 93</td>
</tr>
<tr>
<td>Mittag-Leffler</td>
<td>15, 19, 20, 46, 50</td>
</tr>
<tr>
<td></td>
<td>56, 58, 61, 68, 69, 87, 88</td>
</tr>
<tr>
<td>Poisson</td>
<td>1, 8</td>
</tr>
<tr>
<td>power-law</td>
<td>8, 13, 15, 46, 61</td>
</tr>
<tr>
<td>Rayleigh</td>
<td>14, 15, 19</td>
</tr>
<tr>
<td>truncated</td>
<td>61</td>
</tr>
<tr>
<td>uniform</td>
<td>4, 29, 48, 50, 58, 59, 62, 64</td>
</tr>
<tr>
<td>Weibull</td>
<td>15, 19</td>
</tr>
<tr>
<td>triangle count</td>
<td>37, 40, 43</td>
</tr>
<tr>
<td>triangle count, see on-duration</td>
<td></td>
</tr>
<tr>
<td>interevent time, see on-duration</td>
<td></td>
</tr>
<tr>
<td>initialisation time, see on-duration</td>
<td></td>
</tr>
<tr>
<td>interaction time, see on-duration</td>
<td></td>
</tr>
<tr>
<td>link spread</td>
<td></td>
</tr>
<tr>
<td>Markov-chain Monte Carlo method,</td>
<td></td>
</tr>
<tr>
<td>model</td>
<td></td>
</tr>
<tr>
<td>version 1</td>
<td></td>
</tr>
<tr>
<td>version 2a</td>
<td></td>
</tr>
<tr>
<td>version 2b</td>
<td></td>
</tr>
<tr>
<td>version 2c</td>
<td></td>
</tr>
<tr>
<td>node degree</td>
<td></td>
</tr>
<tr>
<td>non-Markovianity</td>
<td></td>
</tr>
<tr>
<td>off-duration</td>
<td></td>
</tr>
<tr>
<td>on-duration</td>
<td></td>
</tr>
<tr>
<td>probability distribution</td>
<td></td>
</tr>
<tr>
<td>exponential</td>
<td></td>
</tr>
<tr>
<td>gamma</td>
<td></td>
</tr>
<tr>
<td>generalised Pareto</td>
<td></td>
</tr>
<tr>
<td>heavy-tailed</td>
<td></td>
</tr>
<tr>
<td>log-normal</td>
<td></td>
</tr>
<tr>
<td>Mittag-Leffler</td>
<td></td>
</tr>
<tr>
<td>Poisson</td>
<td></td>
</tr>
<tr>
<td>power-law</td>
<td></td>
</tr>
<tr>
<td>Rayleigh</td>
<td></td>
</tr>
<tr>
<td>truncated</td>
<td></td>
</tr>
<tr>
<td>uniform</td>
<td></td>
</tr>
<tr>
<td>Weibull</td>
<td></td>
</tr>
<tr>
<td>triangle count</td>
<td></td>
</tr>
</tbody>
</table>