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;

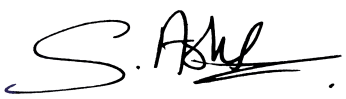
And to you, the reader.

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].

I acknowledge useful discussion with János Kertész at the 13th Econophysics Colloquium & 9th Polish Symposium on Physics in Economy and Social Sciences, Warsaw, July 2017.

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

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

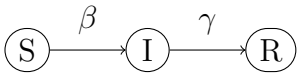


Figure 1.1: Example of a categorical SIR model with a fixed population such as in [101] - here, β is the infection rate and γ is the recovery rate

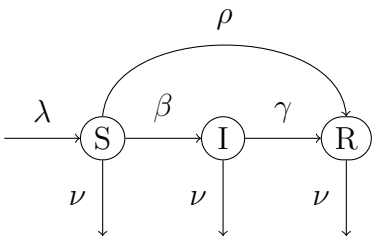


Figure 1.2: Example of a categorical SIR model with a dynamic population and vaccination such as in [42] - here, β is the infection rate, γ is the recovery rate, ρ is the vaccination rate, λ is the birth rate and ν is the death rate

form of the transition rates [35, 90].

1.2.1.2 Stochastic Categorical Models

Simple stochastic forms of these models exist, which can introduce a noise component [128, 224, 225] 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 [36]. The most common stochastic variations are the general stochastic epidemic model [19, 55] and the Reed-Frost model [1]. In the general stochastic epidemic model, a fixed population SIR model may be used with the transition probabilities in (1.1) [83], where S_t and I_t and the numbers of susceptible and infected individuals at time t, N is the total population size, and β and γ are the infection and recovery rates:

$$P\left((S_{t+\Delta t}, I_{t+\Delta t}) - (S_t, I_t) = (-1, 1)\right) = \beta \frac{S_t I_t}{N} \Delta t + o\left(\Delta t\right), \tag{1.1a}$$

$$P\left((S_{t+\Delta t}, I_{t+\Delta t}) - (S_t, I_t) = (0, -1)\right) = \gamma I_t \Delta t + o\left(\Delta t\right), \tag{1.1b}$$

$$P\left((S_{t+\Delta t}, I_{t+\Delta t}) - (S_t, I_t) = (0, 0)\right) = 1 - \left(\beta \frac{S_t}{N} + \gamma\right) I_t \Delta t + o\left(\Delta t\right). \tag{1.1c}$$

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:

$$C_{t+1} = S_t \left(1 - (1-p)^{I_t}\right), \tag{1.2}$$

where C_{t+1} is the number of new infections in the interval $(t, t+1]$, S_t and I_t and 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 [72, 152]. Other models may use Markovian

switching, where the infection and recovery rates have multiple states with transitional probabilities [82], or jump perturbations [222], where some movement between states occurs stochastically [96], in order to introduce an element of stochastic behaviour 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 θ_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 θ_v represent the different tendencies of nodes to copy their neighbours - 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 θ_v are distributed 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 models 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

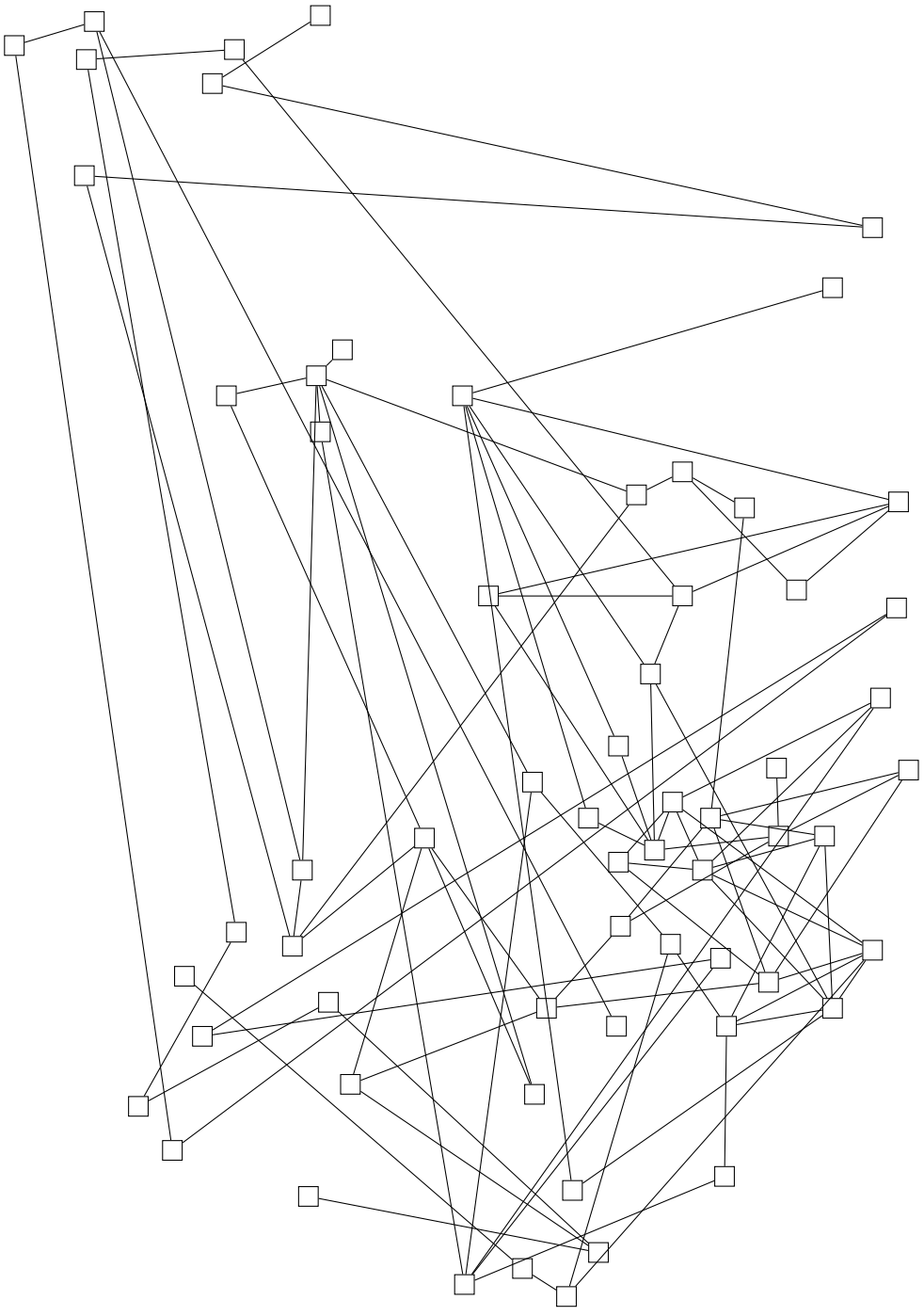


Figure 1.3: Example contact network generated by a random walk study in Canberra, Australia, with nodes placed in relation to the locations within the city where participants were resident (adapted from Figure 2 of [109])

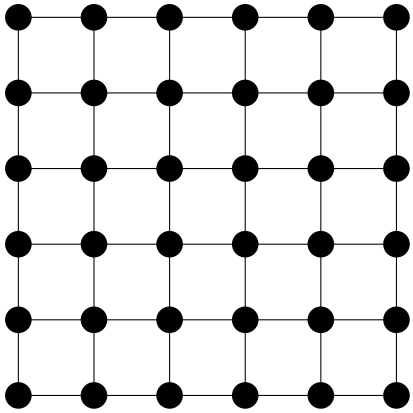


Figure 1.4: Example of a square grid lattice network with 36 nodes

node becomes healthy and susceptible again at rate 1 independent of other nodes, and a susceptible node becomes infected at a rate of λ 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

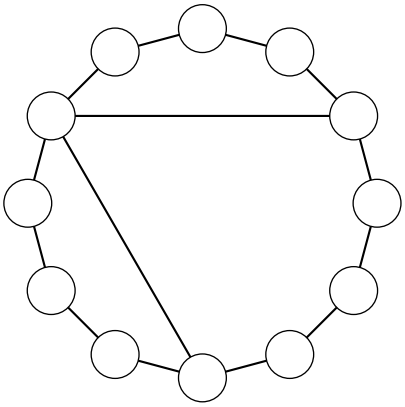


Figure 1.5: Example of a small-world network

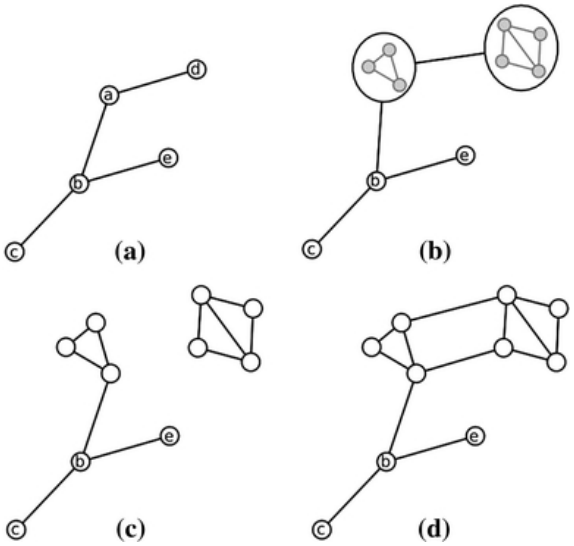


Figure 1.6: Example of small-world network construction. In (a), a random network is created. In (b), nodes a and d are replaced by densely connected sets A and D. In (c), an edge between A and b is introduced to replace the original edge ab. In (d), the multiple edges between A and D are introduced to replace the original edge ad. (From [220])

random and multiple links can be placed between sets for each link in the original network. An example of this construction can be seen in Figure 1.6 [220].

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

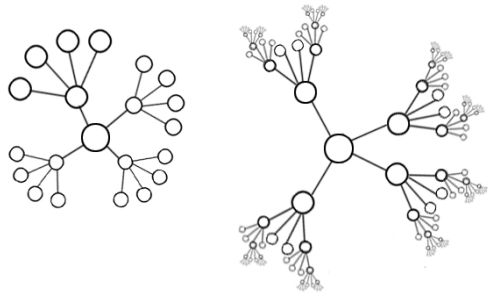


Figure 1.7: Examples of a scale-free network, showing the preservation of the underlying structure between various sizes. (From [197])

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:

$$p_i = \frac{k_i}{\sum_j k_j}. \tag{1.5}$$

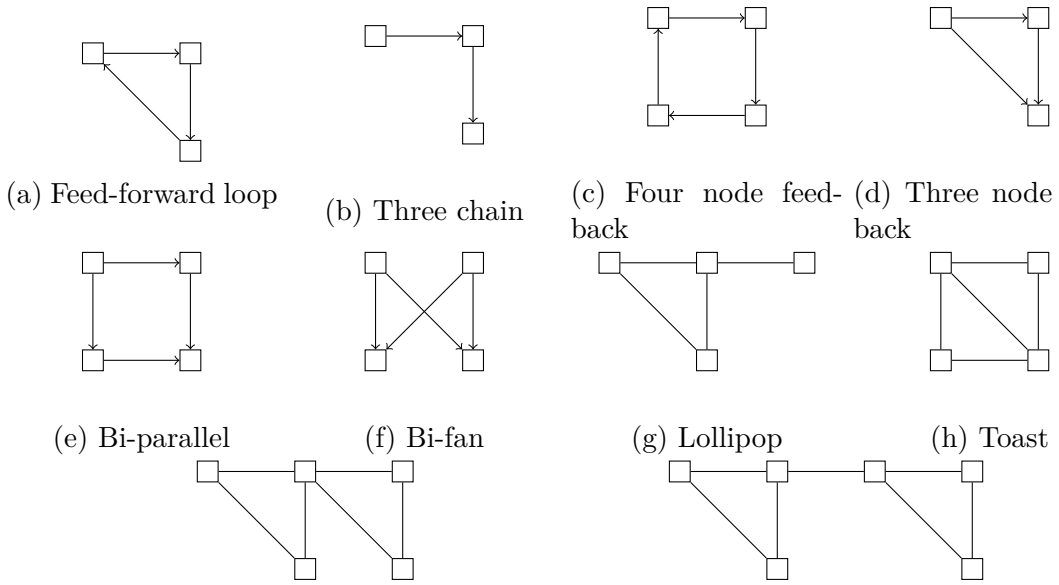
After t time steps, this results in a network with $t + m_0$ nodes and mt edges [8]. This model is related to Yule's process and Simon's process as discussed by Simkin and Roychowdhury[176]. Simulations and studies of scale-free network, reveal that the efficiency of random vaccination strategies is 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 $\mathcal{G}_n$, the set of all networks on n nodes, is given by (1.6):

$$P(G = g) = \exp\left(\sum_i \theta_i T_i(g) - \psi\left(\boldsymbol{\theta}\right)\right). \tag{1.6}$$

Here $\boldsymbol{\theta} = \{\theta_i\}_{i=1}^m$ are m real valued parameters, T_i are real valued statistics defined on $\mathcal{G}_n$, and ψ is a normalising function. Put simply, a researcher specifies a ERGM



(a) Feed-forward loop (b) Three chain (c) Four node feedback (d) Three node feedback

(e) Bi-parallel (f) Bi-fan (g) Lollipop (h) Toast

(i) Bow tie (j) Bow tie with bridge

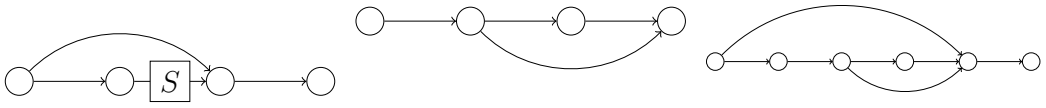
Figure 1.8: A variety of common directed and undirected network motifs [106, 145]

by choosing a set of network configurations of theoretical interest. Then, by applying this particular model to an observed network, parameters $\boldsymbol{\theta}$ 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 α, while existing links are independently deactivated at some other rate ω [104, 193]. Additionally constraints can be placed on this globally such as reducing α 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



(a) Original network $\mathcal{G}$ (b) Replacement G' (c) New network

Figure 1.9: Example of a HRG production rule (adapted from [56])

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 $\mathcal{G}$ (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 $\mathcal{G} = (V, E)$ is a collection of nodes V and links E defined between pairs of nodes - alternative language refers to $\mathcal{G}$ as a graph, with vertices $V = V(\mathcal{G})$ and edges $E = E(\mathcal{G}) \subseteq V \times V$ [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 $\mathcal{G}$, $A = A(\mathcal{G})$. This matrix (for undirected and unweighted networks) is a logical symmetric matrix defined by the rules in (1.7):

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

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, 58]. 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 $\frac{1}{2}(A_3)_{i,i}$, with the total number of triangles in the network equal to $\frac{1}{6}\operatorname{Tr}(A^3)$ [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):

$$f : V(\mathcal{G}) \to V(\mathcal{H}) \tag{1.8}$$

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 [54], 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):

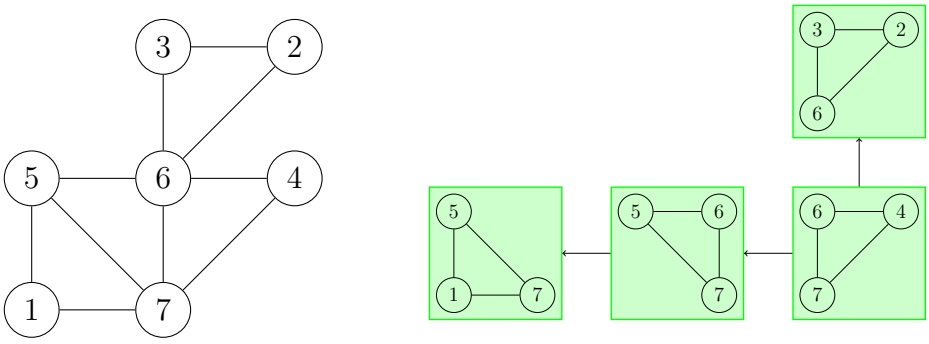
$$p_k = \text{probability that a node has degree } k. \tag{1.9}$$

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 1.10.



(a) A simple network, $\mathcal{G}$

(b) A clique tree of $\mathcal{G}$, $K(\mathcal{G})$

Figure 1.10: A clique tree of a simple network (adapted from [31])

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 λ^{-1} :

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

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 $k\theta$:

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

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

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 $\theta = 2$) [184]. This distribution is one of the most common, seeing uses such as describing inter-event intervals in neuroscience[161, 215] and cancer incidence in oncology[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:

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

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

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

Here the parameter $\sigma \in (0, \infty)$ represents scale, and the distribution has support $x \in [0, \infty)$ [69]. It can be generalised into the Rice distribution, being equal to a Rice distribution with distance parameter, ν, equal to 0 [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 [7, 51, 175] and for wind energy analysis [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 $\exp\left(\mu + \frac{\sigma^2}{2}\right)$:

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

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 $x \in (0, \infty)$ [69]. It is a heavy-tailed distribution for all parameters [135]. It appears in many natural phenomena and growth processes, such as forum post lengths [183] and dwell times [217]. It also appears in such things as citation counts [194] and disease incubation rates [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 $k < 1$:

$$f(x;k,\sigma,\theta) = \begin{cases} \frac{1}{\sigma}\left(1 + k\frac{x-\theta}{\sigma}\right)^{-\frac{k+1}{k}}, & k > 0, x > \theta, \\ \frac{1}{\sigma} e^{-\frac{x-\theta}{\sigma}}, & k = 0, x > \theta, \\ \frac{1}{\sigma}\left(1 + k\frac{x-\theta}{\sigma}\right)^{-\frac{k+1}{k}}, & k < 0, \theta - \frac{\sigma}{k} > x > \theta, \\ 0, & \text{otherwise.} \end{cases} \tag{1.16}$$

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 $x \in \left[\theta, \theta - \frac{\sigma}{k}\right]$ for $k < 0$ [46, 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 $\alpha = \frac{1}{k}$ [93, 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 $\lambda\Gamma\left(1 + \frac{1}{k}\right)$:

$$f(x; \lambda, k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-\left(\frac{x}{\lambda}\right)^k}, & x \geq 0, \\ 0, & x < 0. \end{cases} \tag{1.17}$$

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 ∞, 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:

$$\Psi(t) = P(T > t) = E_\mu\left(-\left(\frac{t}{\tau_0}\right)^\mu\right), t \geq 0, \tau_0 > 0, 0 < \mu \leq 1, \tag{1.18}$$

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

$$E_\mu(z) = \sum_{n=0}^{\infty} \frac{z^n}{\Gamma(\mu n + 1)}, \mu > 0, z \in \mathbb{C}, \tag{1.19}$$

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 μ 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 μ are presented in Figure 1.11.

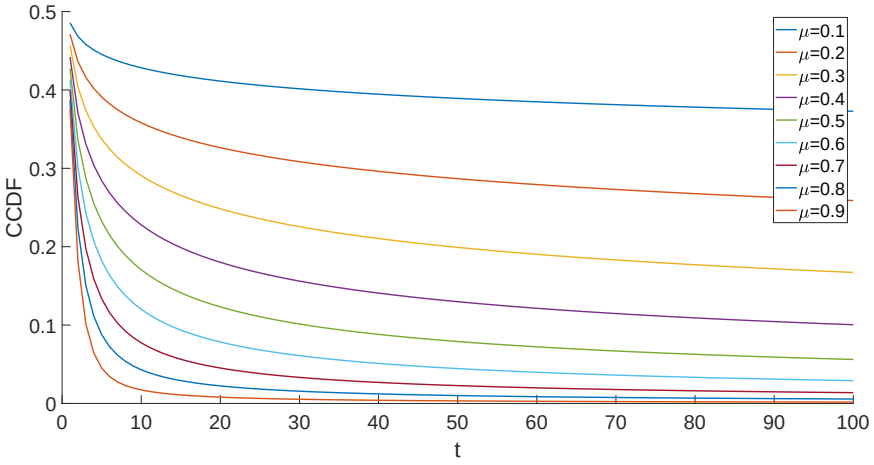


Figure 1.11: Example CCDFs for a Mittag-Leffler distribution

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 $\ell_{\max} = N(N-1)/2$, 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 [149]. 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 [138]. 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 [54].

[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 $z_i = F(x_i)$:

$$x_1 \leq x_2 \leq \ldots \leq x_n. \tag{2.1}$$

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:

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

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

$$D = \max \left[D^+, D^- \right]. \tag{2.2c}$$

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]:

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

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

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

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]:

$$n \int_{-\infty}^{\infty} \left(F_n(x) - F(x) \right)^2 \Psi(x) dF(x). \tag{2.5}$$

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].

Kuiper 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:

$$V = D^+ + D^-. \tag{2.6}$$

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.

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

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

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

Anderson-Darling 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):

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

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

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.

Kullback-Leibler 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:

$$D_{KL}\left(P||Q\right) = \sum_{x} P(x) \log\left(\frac{P(x)}{Q(x)}\right). \tag{2.10}$$

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]:

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

where

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

2.2 Results of Analysis

Due to their expansive nature, tables of results for this section are presented in supplemental materials[3]. 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[4], 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[5] 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: $X_{ij,n}^{On} \sim \text{Exp}\left(Y_{ij}\right)$
- On-Duration Parameter: $Y_{ij} \sim \text{LogNormal}\left(\mu_1, \sigma_1\right)$

[3] Available from `https://drive.google.com/open?id=13eUXp3xGph1B5T7dlIPvEVPZcHyACmom`.
[4] See appendix A.3 for details.
[5] See appendix A.4 for details.

Parameter	Mean	Variance
X_{ij}^{Init}	6.2780×10^3	4.4029×10^7
$X_{ij,n}^{On}$	33.2949	1.2254×10^3
Y_{ij}	35.6676	104.3023
$X_{ij,n}^{Off}$	1.4626×10^3	1.1791×10^7
t_i	2.4557	30.2956

Table 2.1: Mean and variance of observed parameters

Parameter	Value
λ	6.2780×10^3
μ_1	3.5348
σ_1	0.2807
μ_2	6.3512
σ_2	1.3688
μ_3	5.6901×10^{-4}
σ_3	1.7957

Table 2.2: Chosen parameters

- 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: $t_i \sim \text{LogNormal}(5.6901 \times 10^{-04}, 1.7957)$

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

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

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 $\mathcal{H}_0$ is:

$$\mathcal{H}_0 : \text{The two observed samples come from a common distribution.} \tag{3.1}$$

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[1]. Examining these results, I conclude that while there is not a unanimous degree of acceptances for $\mathcal{H}_0$, 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

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

[1] See appendix A.6 for details.

2. **ARP On-Phase:** Assign the link the on-duration

$$X_{ij,n}^{\text{On}} \sim \text{Exp}\left(Y_{ij}\right)$$

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].

$$\begin{aligned}
\mathbb{P}\left(N_t = n\right) &= \mathbb{P}\left(\sum_{m=1}^{\infty} \mathbb{I}_{\sum_{i=1}^{m} Z_i \leq t} = n\right) \\
&= \mathbb{P}\left(\left\{\sum_{i=1}^{n} Z_i \leq t\right\} \cap \left\{\sum_{i=1}^{n+1} Z_i > t\right\}\right) \\
&= \mathbb{P}\left(\left\{\sum_{i=1}^{n} Z_i \leq t\right\} \cap \left\{Z_{n+1} > t - \sum_{i=1}^{n} Z_i\right\}\right) \\
&= \int_0^t f_{\sum_{i=1}^{n} Z_i}(w) \left(\int_{t-w}^{\infty} f_{Z_{n+1}}(u) du\right) dw
\end{aligned}$$

[2]See appendices B.55 and B.22 for details.

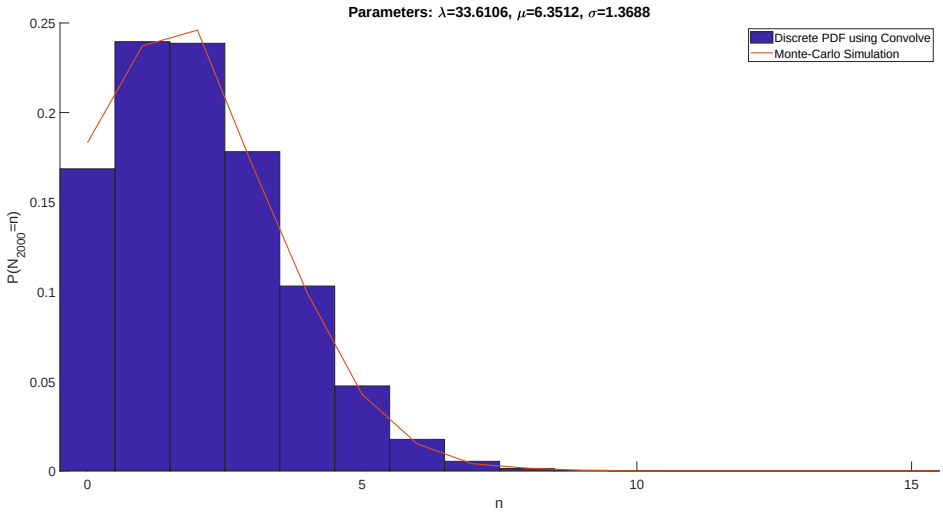


Figure 3.1: PDF produced using Convolve (bar) vs. PDF from Monte-Carlo Simulation (line) for N_{2000}

$$= \int_0^t f_{\sum_{i=1}^n Z_i}(w)\left(1 - F_{Z_{n+1}}(t-w)\right) dw$$
$$= \int_0^t (\underbrace{h \star h \star \ldots \star h}_{n-times})(w)\,(1 - H(t-w))\, dw$$

Here, $h(x) = (f \star g)(x)$ is the PDF of $X + Y$ and $H(x)$ is its associated CDF. For $\mathbb{P}(N'_t = n)$, it should be clear that this only needs to be slightly modified to 'add in' the extra distribution, giving (3.2):

$$\mathbb{P}(N'_t = n) = \int_0^t (g \star \underbrace{h \star h \star \ldots \star h}_{n-times})(w)\,(1 - H(t-w))\, dw. \tag{3.2}$$

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):

$$h(z) = (f \star g)(z) = \int_{-\infty}^{\infty} f(z-t)g(t)dt = \int_{-\infty}^{\infty} f(t)g(z-t)dt/ \tag{3.3}$$
$$\widetilde{h}(s) = \widetilde{(f \star g)}(s) = \widetilde{f}(s)\widetilde{g}(s). \tag{3.4}$$

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[3].

[3]See 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

$$t_i \sim \text{LogNormal}(5.6901 \times 10^{-04}, 1.7957).$$

[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[5]), 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 $C = \mathbb{I}$. 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

$$X^k_{n_1 n_2} \sim \text{Exp}(Y_{n_1 n_2})$$

as usual 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 and the methods for extracting the matrix M and selecting a link can be found in the appendices[6].

[5] See appendix A.11 for details.

[6] See appendices B.57, 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[7] available in the supplemental materials[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

$$M_{ij} \sim \Gamma(k_i, 0.0037) + \Gamma(k_j, 0.0037)$$

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

$$M_{ij} \sim \Gamma(k_i + k_j, 0.0037)$$

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

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

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:

$$\begin{aligned} M_{i\Sigma} = \sum_{j=1}^{n} M_{ij} \sim \sum_{\substack{j=1 \\ j\neq i}}^{n} \Gamma\left(k_{ij}, 0.0037\right) &= \Gamma\left(\sum_{\substack{j=1 \\ j\neq i}}^{n} k_{ij}, 0.0037\right) \\ &= \Gamma\left(k_{i\Sigma}, 0.0037\right) = \Gamma(12.3109, 0.0037). \end{aligned} \tag{3.5}$$

[7] See appendix A.3 for details.

[8] Available from `https://drive.google.com/open?id=13eUXp3xGph1B5T7dlIPvEVPZcHyACmom`.

Thus to identify the individual parameters, I wish to solve the system (3.6):

$$\left(k_{i\Sigma} = \sum_{\substack{j=1 \\ j\neq i}}^{n} k_{ij} = 12.3109 \right)_{i=1,2,\cdots,n} . \tag{3.6}$$

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:

$$\begin{aligned}
(n-2)k_1 + \sum_j k_j & \\
&= (n-2)k_2 + \sum_j k_j \\
&= (n-2)k_3 + \sum_j k_j \\
&= \dots \\
&= (n-2)k_{n-1} + \sum_j k_j \\
&= (n-2)k_n + \sum_j k_j = 12.3109
\end{aligned}$$

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

$$\begin{aligned}
\sum_i k_{i\Sigma} &= \sum_i \left((n-2)k_i + \sum_j k_j \right) \\
&= \sum_i \left((n-2)k_i \right) + n \sum_j k_j \\
&= (n-2) \sum k_i + n \sum k_i \\
&= 2(n-1) \sum k_i \\
&= 2(n-1) \sum k^\star = 2(n-1)nk^\star = n(12.3109).
\end{aligned}$$

Thus $k^\star = 12.3109/2(n-1)$, 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):

$$\left(k_{i\Sigma} = (n-2)k_i - k_i^2 + \sum_j (1 + k_i)\, k_j = 12.3109\right)_{i=1,2,\cdots,n} . \tag{3.7}$$

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].

[9]See appendices B.57, B.74 and B.22 for details.

Model	Parameters	Para.Values	Count
1	$X_{ij}^{\text{Init}} \sim \text{Exp}(\lambda)$	6278.0	5
	$Y_{ij} \sim \text{LogNormal}(\mu_1, \sigma_1^2)$	$(3.5348, 0.2807)$	
	$X_{ij,n}^{\text{Off}} \sim \text{LogNormal}(\mu_2, \sigma_2^2)$	$(6.3512, 1.3688)$	
2a	$t_i \sim \text{LogNormal}(\mu_1, \sigma_1^2)$	$(5.6901 \times 10^{-4}, 1.7957)$	$4 + \frac{n(n-1)}{2}$
	$Y_{n_1 n_2} \sim \text{LogNormal}(\mu_2, \sigma_2^2)$	$(3.5348, 0.2807)$	
	M	$n \times n$ symmetric matrix	
2b	$t_i \sim \text{LogNormal}(\mu_1, \sigma_1^2)$	$(5.6901 \times 10^{-4}, 1.7957)$	$5 + \frac{n(n-1)}{2}$
	$Y_{n_1 n_2} \sim \text{LogNormal}(\mu_2, \sigma_2^2)$	$(3.5348, 0.2807)$	
	$u \geq u_f$ (the 'forcing' rate)	0.0640	
	M	$n \times n$ symmetric matrix	
2c	$t_i \sim \text{LogNormal}(\mu_1, \sigma_1^2)$	$(5.6901 \times 10^{-4}, 1.7957)$	7
	$Y_{n_1 n_2} \sim \text{LogNormal}(\mu_2, \sigma_2^2)$	$(3.5348, 0.2807)$	
	$u \geq u_f$ (the 'forcing' rate)	0.0640	
	$M_{ij} \sim \Gamma(k, \theta)$	$\left(\frac{12.3109}{2(n-1)}, 0.0037\right)$	

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)

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 ECCDFs 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,

		Node Activity Potential	Global Clustering Coefficient	Component Count	Links per Component	Nodes per Component	Triangle Count
1	min	0.1304	0.02517	0.01813	0.009406	0.006316	0.02394
	max	0.5769	0.2876	0.6083	0.07935	0.07935	0.2876
	mean	0.3365	0.1191	0.2895	0.03934	0.03856	0.1189
	mode	0.3043	0.03524	0.01813	0.009406	0.006316	0.03524
2a	min	0.08696	0.01835	0.03448	0.004412	0.00467	0.002869
	max	0.4249	0.2677	0.6067	0.09015	0.09015	0.2641
	mean	0.2212	0.09876	0.308	0.04169	0.04195	0.06908
	mode	0.1739	0.01835	0.03448	0.004412	0.00467	0.002869
2b	min	0.08	0.02513	0.03112	0.004466	0.003064	0.005029
	max	0.4377	0.2065	0.5274	0.08203	0.08203	0.2011
	mean	0.2241	0.08137	0.2838	0.0373	0.03754	0.05605
	mode	0.2273	0.06793	0.03112	0.004466	0.003064	0.005029
2c	min	0.08696	0.03049	0.03836	0.005304	0.006754	0.003863
	max	0.4945	0.2036	0.552	0.08004	0.08004	0.1842
	mean	0.2485	0.07893	0.3087	0.04149	0.04166	0.04898
	mode	0.2273	0.07942	0.03836	0.005304	0.006754	0.006151

Table 3.3: Selected two-sample Kolmogorov-Smirnov distances (see section 3.3 for full explanation)

	Model 1	Model 2a	Model 2b	Model 2c
Active Links	1	0	1	0
Active Nodes	0	0	0	0
Node Activity Potential	954	1000	1000	999
GCC	0	0	0	0
Interaction Time	6	0	0	0
Time Between Contacts	24	2	2	2
Component Count	151	264	240	306
Links per Component	62	35	56	42
Nodes per Component	33	68	79	63
Triangle Count	152	503	637	637

Table 3.4: Acceptances of $\mathcal{H}_0$ (see equation 3.8) at 5% Level (max: 1000) (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[10].

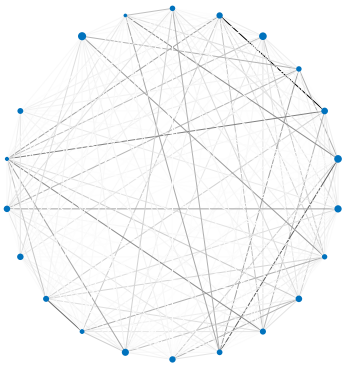
I also present comparative data in two tables. In Table 3.3, I show a summary of the two-sample Kolmogorov-Smirnov distances - see (2.3) in subsection 2.1.2.1 - between the collection of 20 empirical samples and 20 simulated data samples from each of the 4 models presented, showing the minimum, maximum, 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[11]. I test the hypothesis $\mathcal{H}_0$, in this case, this is given in (3.8) below:

$$\mathcal{H}_0 : \text{The chosen empirical and generated data samples come from a common distribution.} \tag{3.8}$$

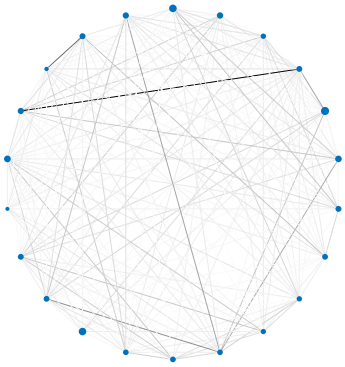
In Table 3.4, I present the total number of acceptances (out of a possible 1000) at the 5%-level of this hypothesis when tested on a particular metric.

Additionally, I present Figure 3.2 to highlight long-term behaviours in the models[12]. 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

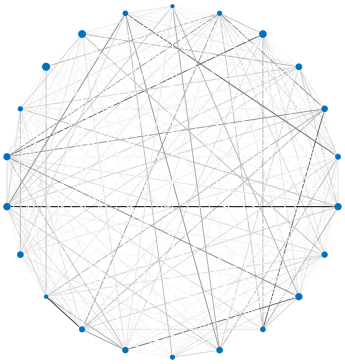
[10] Available from `https://drive.google.com/open?id=13eUXp3xGph1B5T7dlIPvEVPZcHyACmom`.
[11] See appendix A.5 for details.
[12] See appendix A.8 for details.



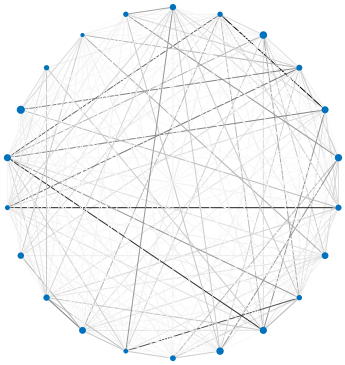
(a) Original Data



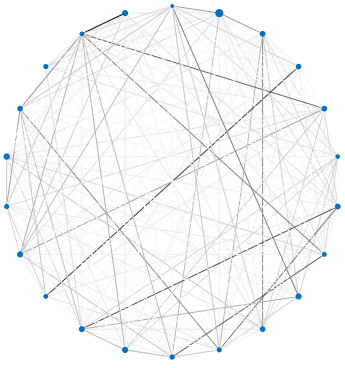
(b) Model 1



(c) Model 2a



(d) Model 2b



(e) Model 2c

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[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=13eUXp3xGph1B5T7dlIPvEVPZcHyACmom`.

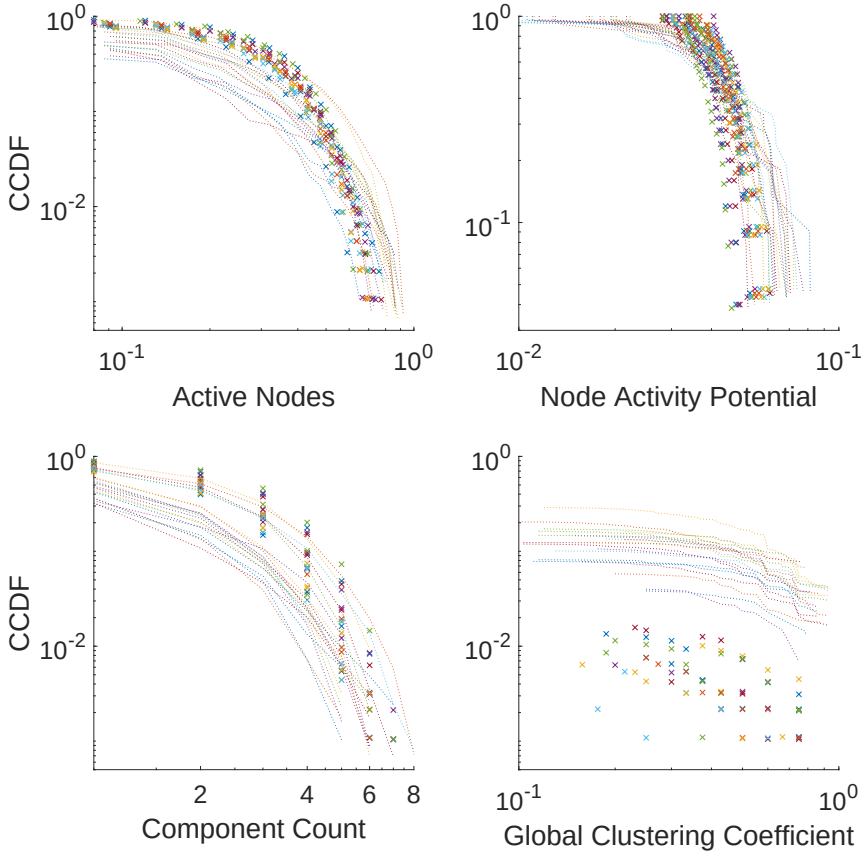


Figure 3.3: Selected results for Model 1. Simulated data is represented by crosses whereas observed data is represented by dotted lines. Each colour represents a different simulation or data set. See subsection 3.3.1 for full explanation. Additional figures are available in the supplemental materials.

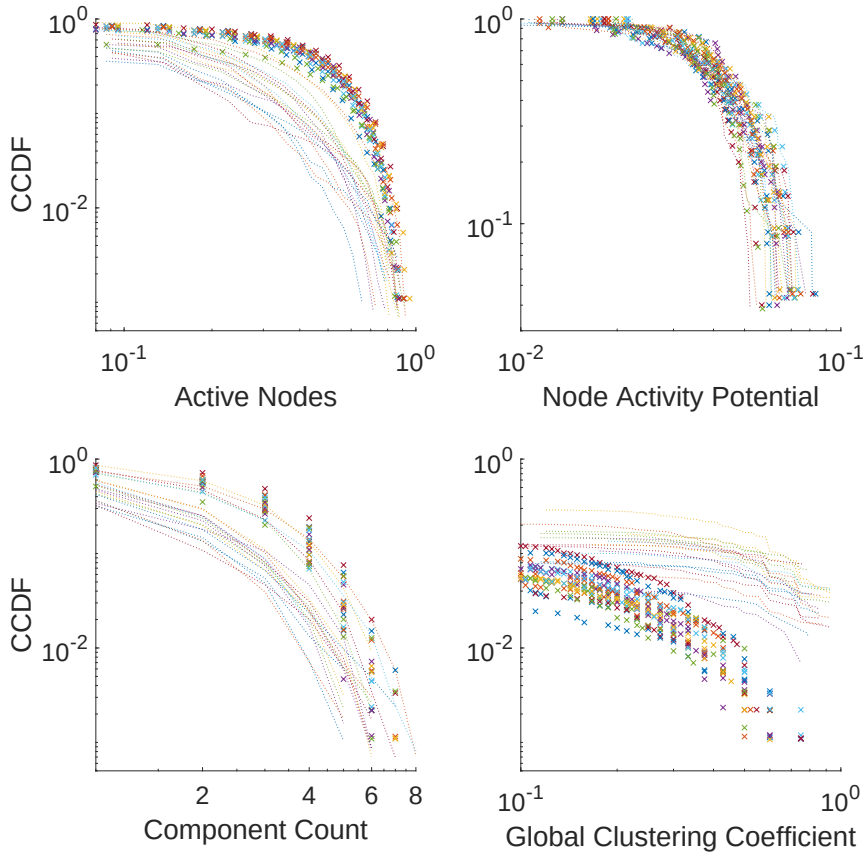


Figure 3.4: Selected results for Model 2a. Simulated data is represented by crosses whereas observed data is represented by dotted lines. Each colour represents a different simulation or data set. See subsection 3.3.2 for full explanation. Additional figures are available in the supplemental materials.

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.

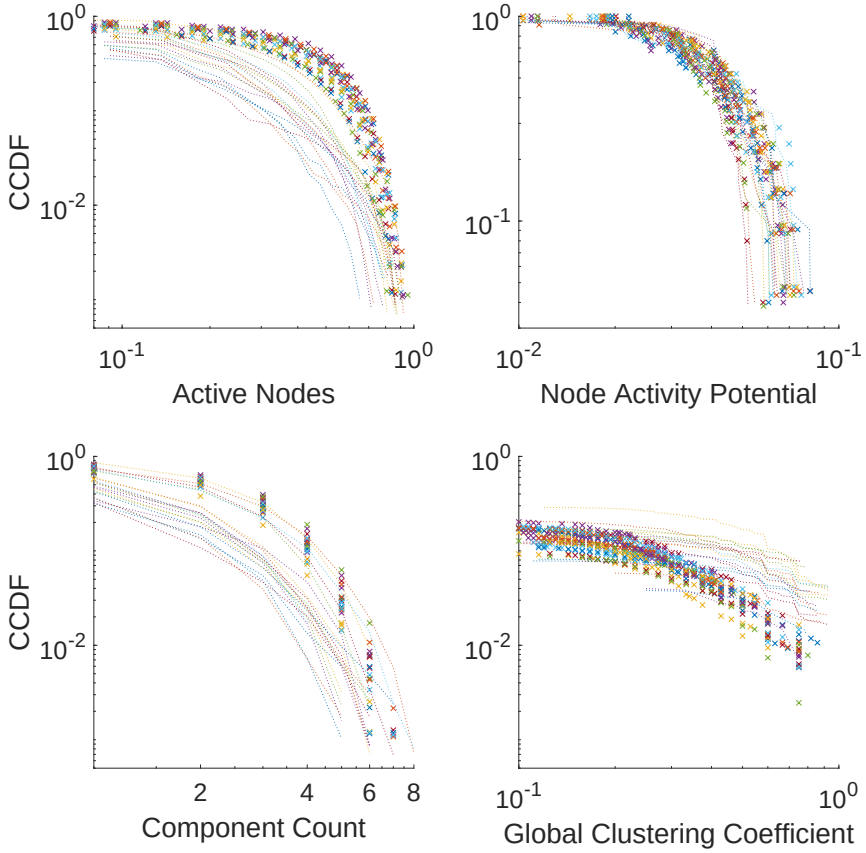


Figure 3.5: Selected results for Model 2b. Simulated data is represented by crosses whereas observed data is represented by dotted lines. Each colour represents a different simulation or data set. See subsection 3.3.3 for full explanation. Additional figures are available in the supplemental materials.

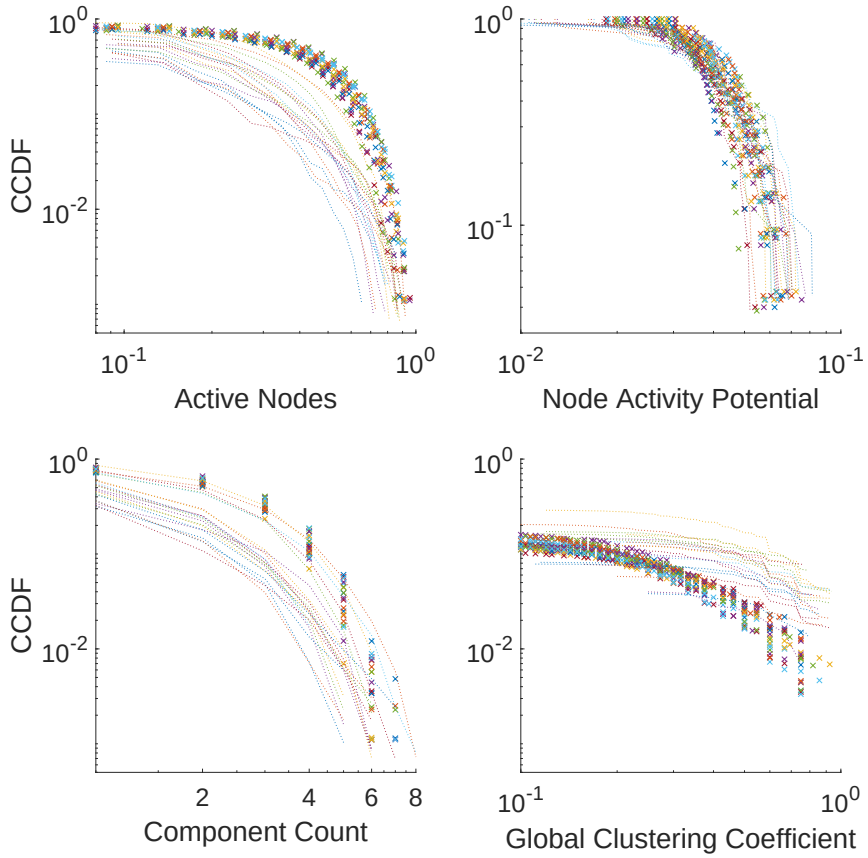


Figure 3.6: Selected results for Model 2c. Simulated data is represented by crosses whereas observed data is represented by dotted lines. Each colour represents a different simulation or data set. See subsection 3.3.4 for full explanation. Additional figures are available in the supplemental materials.

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 3.2e.

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[14]). 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[15]. 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=13eUXp3xGph1B5T7dlIPvEVPZcHyACmom`.

	Model 1	Model 2a	Model 2b
Active Links	0	2	0
Active Nodes	56	389	362
Node Activity Potential	881	1000	1000
GCC	0	0	0
Time Between Contacts	–	2	2
Component Count	0	211	364
Links per Component	0	43	45
Nodes per Component	0	42	48
Triangle Count	0	367	771

Table 3.5: Validation Acceptances of $\mathcal{H}_0$ (see equation 3.9) at 5% Level (max: 1000) (see subsection 3.5 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[16], 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, $\mathcal{H}_0$ is given as

$$\mathcal{H}_0 : \text{The chosen empirical and generated data samples come from a common distribution.} \tag{3.9}$$

[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:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}d\theta. \tag{4.1}$$

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 μ and semi-infinite nature of τ_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 $\boldsymbol{x}$ given the parameters $\boldsymbol{\theta}$) and a *prior* probability $p(\boldsymbol{\theta})$:

$$p(\boldsymbol{\theta}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{x})}, \tag{4.2}$$

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

$$p(\boldsymbol{x}) = \int_{\boldsymbol{\theta}} p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}). \tag{4.3}$$

In this chapter, I will always assume compact support for the prior $p(\boldsymbol{\theta})$ 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[1].

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

Given $X^{(t)} = x^{(t)}$,

1. Generate $Y_t \sim J\left(y|\, x^{(t)}\right)$.
2. Take

$$X^{(t+1)} = \begin{cases} Y_t, & \text{with probability } A\left(x^{(t)}, Y_t\right), \\ x^{(t)}, & \text{with probability } 1 - A\left(x^{(t)}, Y_t\right), \end{cases} \tag{4.4}$$

where

$$A(x,y) = \min\left\{\frac{P(y)}{P(x)}\frac{J(x|y)}{J(y|x)}, 1\right\}. \tag{4.5}$$

For long chains, the distribution of the Markov chain $\left(X^{(t)}\right)$ produced by this algorithm will resemble that of the underlying distribution $P(x)$ and can be generated without complete knowledge of $P(x)$, only the ratio between two terms $\frac{P(y)}{P(x)}$. With the correct construction, this ratio can be formed of known terms, allowing for the estimation of $P(x)$.

In subsubsection 4.2.2.2 below, we wish to estimate an unknown posterior distribution $P\left(\theta|\boldsymbol{x}\right)$. Through Bayes' theorem, we know that $P\left(\theta|\boldsymbol{x}\right) = \frac{P(\boldsymbol{x}|\theta)P(\theta)}{P(\boldsymbol{x})}$ - of which we have information on the prior, $P(\theta)$, and the likelihood, $P\left(\boldsymbol{x}|\theta\right)$. It is our lack of knowledge on the evidence, $P(\boldsymbol{x})$, 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):

[1] See appendix A.10 for details.

$$\frac{P\left(\theta_1|\boldsymbol{x}\right)}{P\left(\theta_2|\boldsymbol{x}\right)} = \frac{\frac{P(\boldsymbol{x}|\theta_1)P(\theta_1)}{P(\boldsymbol{x})}}{\frac{P(\boldsymbol{x}|\theta_2)P(\theta_2)}{P(\boldsymbol{x})}} = \frac{P\left(\boldsymbol{x}|\theta_1\right)P\left(\theta_1\right)}{P\left(\boldsymbol{x}|\theta_2\right)P\left(\theta_2\right)}. \tag{4.6}$$

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 $P\left(\theta|\boldsymbol{x}\right)$.

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

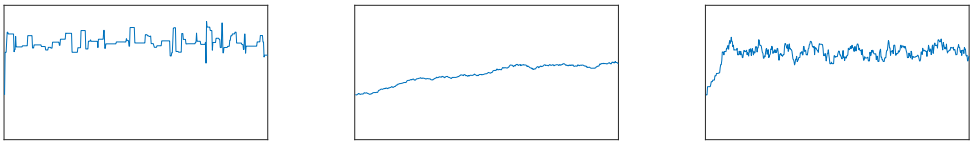
If this were a single parameter distribution, I would start by first generating an initial set of parameters θ_0 at $t = 0$. I then iterate, first generating a candidate state $\hat{\theta}$ equal to a random perturbation from the current state, with this random perturbation distributed according to some chosen distribution. I shall be using a truncated multivariate Gaussian $\mathcal{N}_{a'}^{b'}(0, \sigma)$ with appropriate choices for σ - 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 $A\left(\hat{\theta}, \theta_t\right)$ as given in (4.7) below. As I wish to estimate the posterior probability, $P\left(\theta|\boldsymbol{x}\right)$, 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, $P\left(\boldsymbol{x}|\theta\right)$ is the likelihood of observing the data $\boldsymbol{x}$ given the parameters θ, and $P\left(\theta\right)$ is the chosen prior. This acceptance probability is then compared to a uniform random number $u \in [0, 1]$:

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

If $u \leq A\left(\hat{\theta}, \theta_t\right)$, 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 $\boldsymbol{\theta}_t = (\mu_t, \tau_t)$ to (μ', τ_t), setting the result of the acceptance or rejection as an intermediary state $\boldsymbol{\theta}_t^+ = (\mu_{t+1}, \tau_t)$. I then propose a move from this state $\boldsymbol{\theta}_t^+ = (\mu_{t+1}, \tau_t)$ to (μ_{t+1}, τ'), setting the result of the acceptance or rejection as $\boldsymbol{\theta}_{t+1} = (\mu_{t+1}, \tau_{t+1})$.

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 $s_i \sim \mathcal{U}(a', b')$, 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



(a) σ value too large (b) σ value too small (c) ideal σ value

Figure 4.1: Example MCMC chains for varying choices of σ 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 [112][2].

4.2.2.3 Choice of σ

The choice of σ 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 σ 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 σ is too large. Examples of chains with too large, small and ideal values of σ 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 σ 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 [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 σ 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 [158].

The MATLAB code for selecting σ can be found in the appendices[3].

4.2.3 Approximate Bayesian Computation

In this method, as explored by Beaumont [24], I use summary statistics rather than the full dataset. The aim is, as always, to create an estimate for the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{x})$. 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

[2]See appendix A.10 for details.
[3]See appendix B.47 for details.

computation (ABC) analysis specifies a vector of summary statistics $\boldsymbol{s} = S(\boldsymbol{x})$, where $\dim(\boldsymbol{s}) \ll \dim(\boldsymbol{x})$, and then outputs a posterior distribution for $p(\boldsymbol{\theta}|\boldsymbol{s})$ [179]. I sample parameter points $\hat{\boldsymbol{\theta}}$ from chosen priors, then for each set of points, I generate a summary data set $\hat{\boldsymbol{x}}$ 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 ρ, accepting this set of parameter points if this value is below a defined ε, and rejecting this set of parameter points if it is above (see Equation (4.8) below)[4]:

$$\rho\left(D, \hat{D}\right) \leq \varepsilon. \tag{4.8}$$

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 $\boldsymbol{s}$ 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 ρ.

If this vector $\boldsymbol{s} = S(\boldsymbol{x})$ is sufficient for the model parameters, then $p(\boldsymbol{\theta}|\boldsymbol{s}) \equiv p(\boldsymbol{\theta}|\boldsymbol{x})$, and thus the posterior produced is equivalent to the true posterior. If this vector $\boldsymbol{s}$ is not sufficient, then ABC analysis shall produce the posterior $p(\boldsymbol{\theta}|\boldsymbol{s})$, 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 μ and τ_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 μ, 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 τ_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 μ and τ_0 used were $\mathcal{U}(0.01, 0.99)$ and $\mathcal{U}(a, b)$ - 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

[4] See appendix A.1 for details.

[5] Available from `https://drive.google.com/open?id=13eUXp3xGph1B5T7dlIPvEVPZcHyACmom`.

Actual Parameters		**Estimated Parameters**			
μ	τ_0	μ		τ_0	
		Mean	Std.	Mean	Std.
0.1	0.01	0.0975	0.0039	0.0089	0.0100
0.1	1.00	0.1000	0.0037	0.9968	0.8791
0.1	100	0.0999	0.0038	168.4740	154.2302
0.5	0.01	0.5392	0.0171	0.0131	0.0017
0.5	1.00	0.5018	0.0161	0.8425	0.1217
0.5	100	0.5306	0.0175	92.4037	12.2062
0.9	0.01	0.8954	0.0106	0.0184	6.4292×10^{-4}
0.9	1.00	0.8905	0.0184	0.9644	0.0594
0.9	100	0.8844	0.0183	108.9281	6.6858

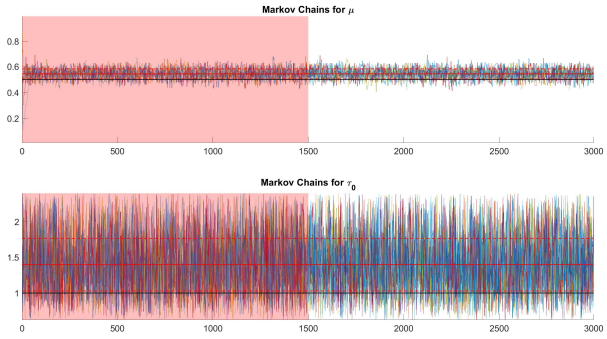
Table 4.1: Mean and standard deviation estimates for the MCMC method

the first 1500 data points, I obtain the chains such as those given in Figure 4.2 and the estimated parameter values given in Table 4.1. Sample histograms of the chain values will be given (and compared to the density generated by the exact method) later in this chapter in Figure 4.4.

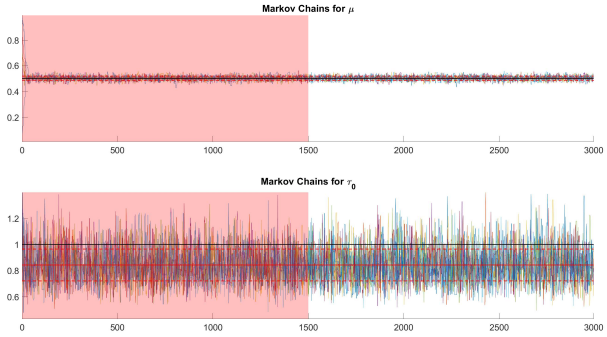
My approximation to the exact posterior with the same priors as those used in the MCMC method returns the 2-dimensional posterior PDFs such as those displayed in Figure 4.3 and estimated parameters given in Table 4.1. Sample plots of the marginal density functions with histograms of the chain values are given in Figure 4.4. I also present sample CCDFs showing the generated data, and the distributions based on true parameters and those estimated by the MCMC and exact methods in Figure 4.5.

In both methods, I observe that the estimated parameters do differ somewhat from the actual parameters, especially in regards to τ_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.

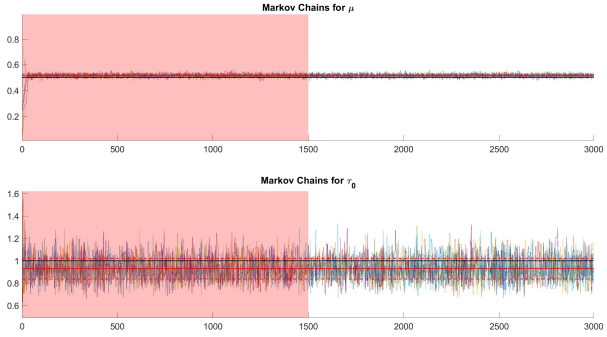
Furthermore, it should be noted that despite any inconsistencies between the estimated and true parameters, the MCMC method that I have presented returns results very close to those presented by the exact method, showing that despite a significant reduction in calculation requirements, it is as equally effective a method and any discrepancy is likely due to a bias induced by the data.



(a) 100 data points

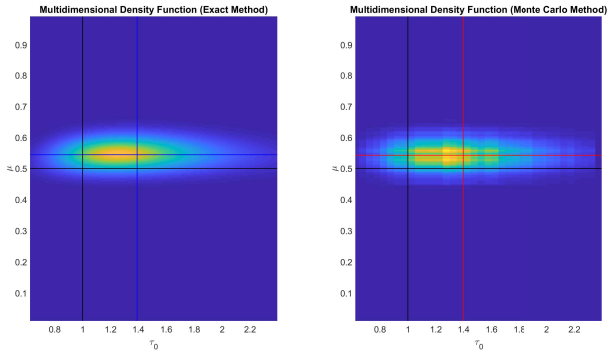


(b) 500 data points

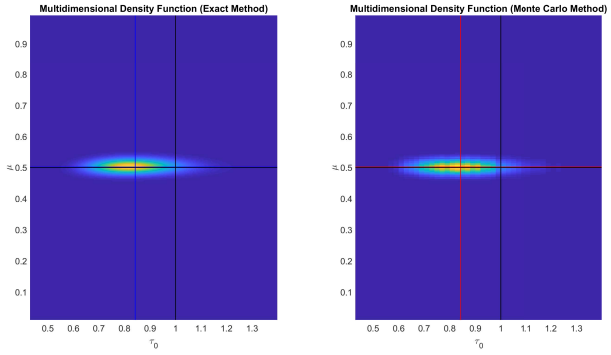


(c) 1000 data points

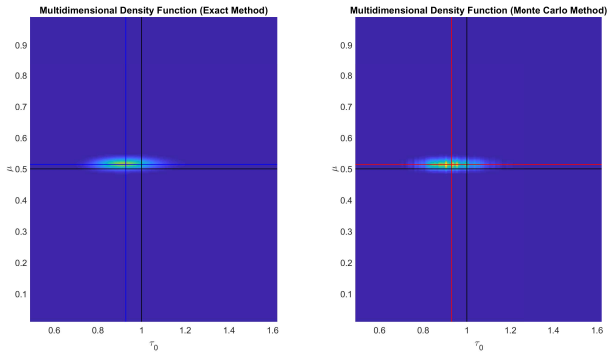
Figure 4.2: Sample MCMC chains for $\mu = 0.5$ and $\tau_0 = 1$. Varying colours in each image represent each of the parallel chains. Highlighted region shows equilibration period. Solid red lines represent estimated parameters with red dashed lines showing one standard deviation either side of this. Solid black lines represent true parameters.



(a) 100 data points

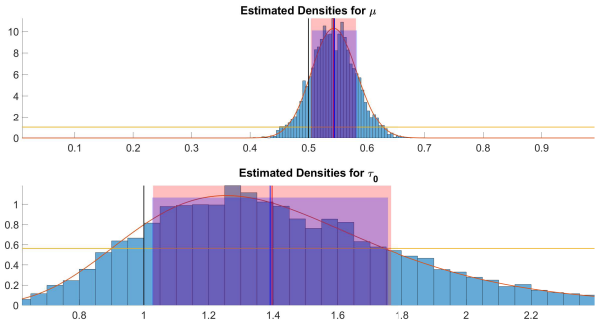


(b) 500 data points

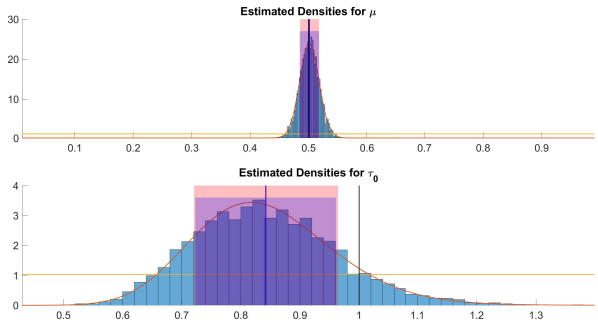


(c) 1000 data points

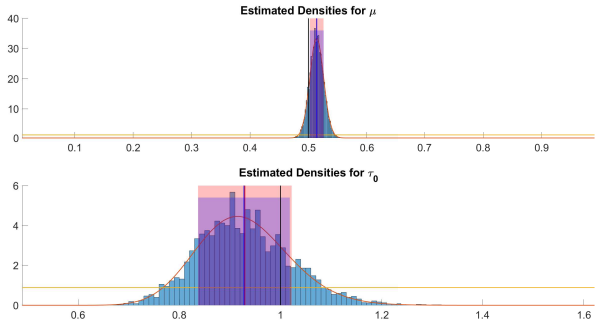
Figure 4.3: Sample multidimensional posterior PDF plots for $\mu = 0.5$ and $\tau_0 = 1$ using exact and MCMC methods. Red and blue lines represent estimated parameters. Black lines represent true parameters.



(a) 100 data points

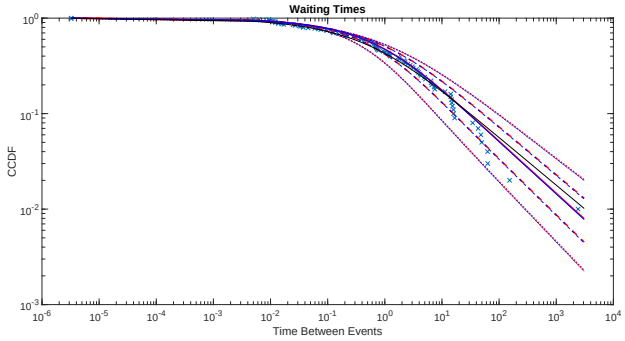


(b) 500 data points

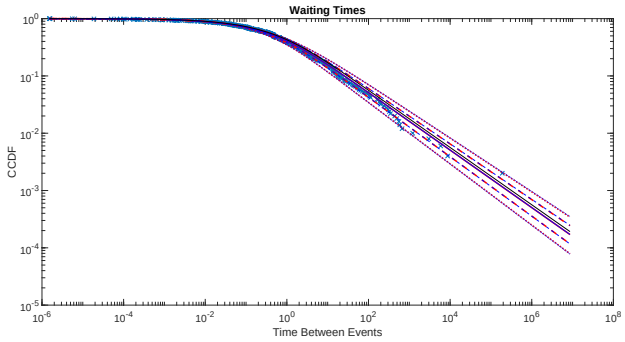


(c) 1000 data points

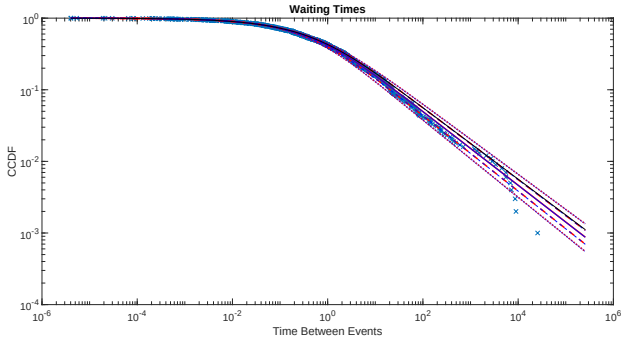
Figure 4.4: Sample marginal posterior PDF plots and histograms of MCMC chains for $\mu = 0.5$ and $\tau_0 = 1$. Prior densities are shown as yellow horizontal lines. Red vertical line shows estimated mean using MCMC method (with red shaded region being one standard deviation either side of this). Blue 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.



(a) 100 data points



(b) 500 data points



(c) 1000 data points

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**			
μ	τ_0	μ		τ_0	
		Mean	Std.	Mean	Std.
0.1	0.01	0.0976	0.0036	0.0088	0.0082
0.1	1.00	0.0998	0.0037	0.9581	0.8757
0.1	100	0.0999	0.0037	168.0859	155.1107
0.5	0.01	0.5389	0.0173	0.0132	0.0017
0.5	1.00	0.5016	0.0167	0.8420	0.1188
0.5	100	0.5307	0.0173	92.3081	12.2334
0.9	0.01	0.8950	0.0181	0.0106	6.0751×10^{-4}
0.9	1.00	0.8905	0.0183	0.9643	0.0575
0.9	100	0.8840	0.0183	108.9947	6.2481

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

Sample Size	**Estimated Parameters**			
	μ		τ_0	
	Mean	Std.	Mean	Std.
100	0.5427	0.0386	1.3968	0.3684
500	0.5018	0.0161	0.8425	0.1217
1000	0.5140	0.0117	0.9296	0.0925

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 μ and τ_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 σ of the true values.

4.3.1.2 Prior Information on the Range of τ_0

For τ_0, I studied the survival function for the Mittag-Leffler distribution given in (4.9):

$$\Psi(t) = E_\mu \left(- (t/\tau_0)^\mu \right). \tag{4.9}$$

Sample Size	Estimated Parameters			
	μ		τ_0	
	Mean	Std.	Mean	Std.
100	0.5444	0.0387	1.3907	0.3648
500	0.5016	0.0167	0.8420	0.1188
1000	0.5142	0.0119	0.9278	0.0906

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

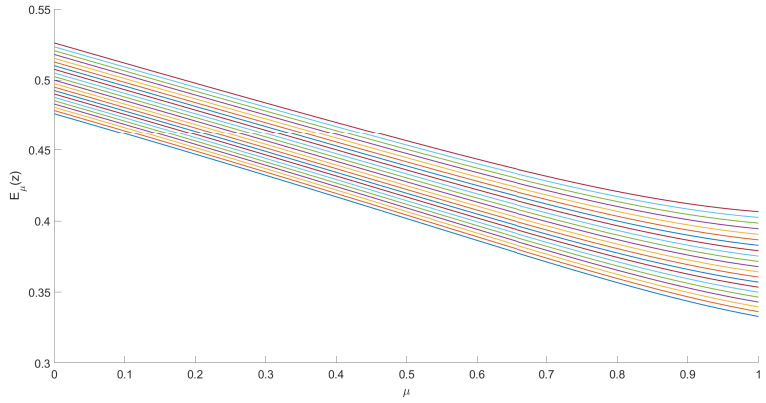


Figure 4.6: $E_\mu(z)$ plotted again μ showing relationship. Different curves show this relationship for various values between $z = -1.1$ (lowest curve, blue) and $z = -0.9$ (highest curve, red).

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 μ increases and thus in this neighbourhood, Equation (4.10) below must hold:

$$\sum_{k=0}^{\infty} z^k = E_0(z) > E_\mu(z) > E_1(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(k+1)} = \sum_{k=0}^{\infty} \frac{z^k}{k!} = \exp(z). \tag{4.10}$$

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 $|z_*| < 1$:

$$\frac{1}{2} > \frac{1}{1 - z_*} = \sum_{k=0}^{\infty} z_*^k > E_\mu(z_*) = \Psi(t_*) > \exp(z_*) > \exp(-1). \tag{4.11}$$

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 τ_0 belongs to $\mathcal{T}_\varepsilon$ given in Equation (4.13):

$$\mathcal{T} = \{t \in \mathbb{R} : \Psi(t) \in [\exp(-1), 0.5]\}. \tag{4.12}$$

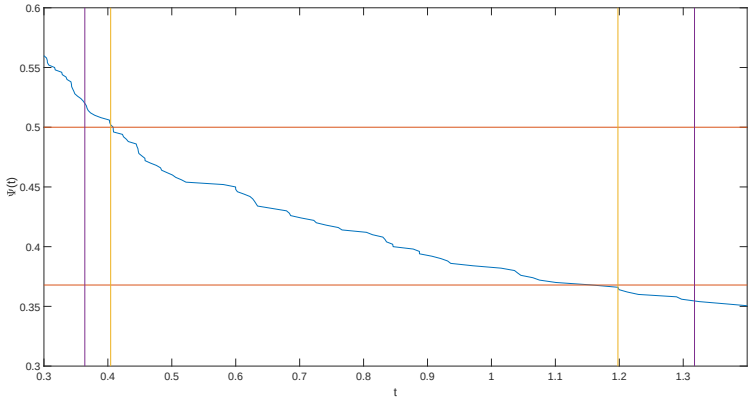


Figure 4.7: Method for approximating τ_0, showing ECCDF (blue), $\exp(-1)$ and 0.5 (orange), first values outside of this region (yellow) and the resulting search range (purple) - here $\tau_0 = 1$ which is captured within the search range

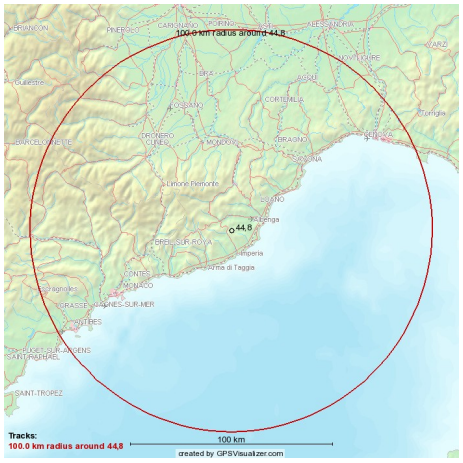
$$\mathcal{T}_\varepsilon = [\inf(\mathcal{T}) - \varepsilon, \sup(\mathcal{T}) + \varepsilon]. \tag{4.13}$$

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 ε-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 τ_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 4.1×10^3, 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.



(a) GPS region selected for study

(b) Earthquake activity showing fault lines - dots represent individual earthquakes with relative size indicating relative magnitudes (courtesy of the United States Geological Survey)

Figure 4.8: Earthquake location data

Method	**Estimated Parameters**			
	μ		τ_0	
	Mean	Std.	Mean	Std.
Exact Method	0.8490	0.0217	4.0903×10^5	2.6573×10^4
Monte Carlo Markov Chain	0.8490	0.0219	4.0865×10^5	2.7019×10^4

Table 4.5: Mean and standard deviation estimates for earthquake data

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 $p(\boldsymbol{x}|\boldsymbol{\theta})$:

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \prod_i p(x_i|\boldsymbol{\theta}), \tag{4.14}$$

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.

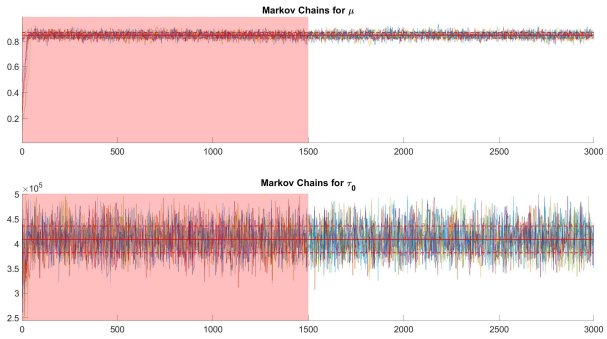


Figure 4.9: MCMC chains for earthquake data. Varying colours in each image represent each of the parallel chains. Highlighted region shows equilibration period. Solid red lines represent estimated parameters with red dashed lines showing one standard deviation either side of this.

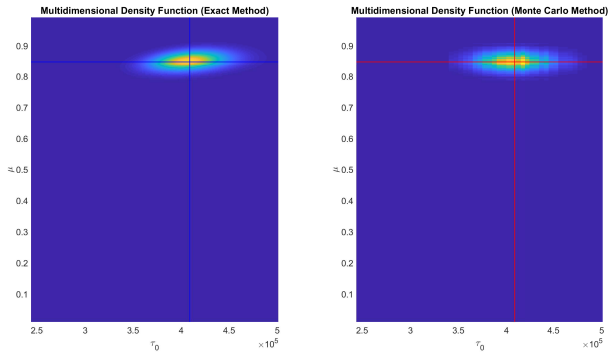


Figure 4.10: Multidimensional posterior PDF plots for earthquake data using exact and MCMC methods. Red and blue lines represent estimated parameters.

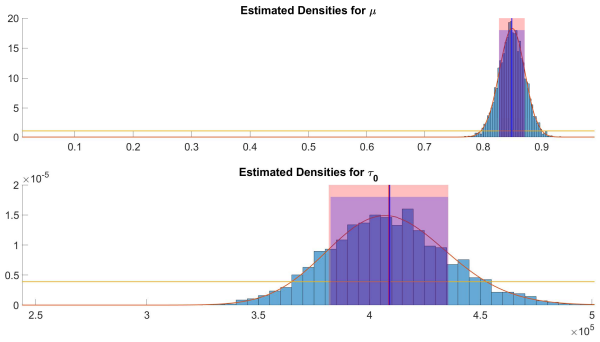


Figure 4.11: Marginal posterior PDF plots and histograms of MCMC chains for earthquake data. Prior densities are shown as yellow horizontal lines. Red vertical line shows estimated mean using MCMC method (with red shaded region being one standard deviation either side of this). Blue vertical line shows estimated mean using exact method (with blue shaded region being one standard deviation either side of this).

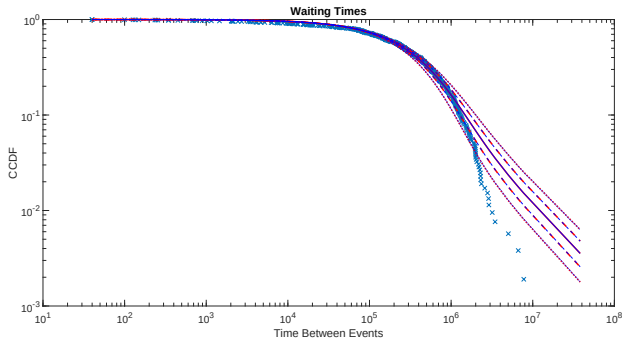


Figure 4.12: CCDF plot showing generated data and distributions using estimated parameters for earthquake data. Red lines show parameters estimated using MCMC method, blue lines show parameters estimated using exact method. Solid lines represent estimates, dashed lines represent one standard deviation 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 2×10^6s ($\sim$ 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):

$$P(x) \propto x^{\alpha-1} e^{\frac{x}{x_c}} \tag{4.15a}$$

$$P(x) \propto x^{-\gamma} \tag{4.15b}$$

where α is a constant value, x_c is the truncation or cut-off value and γ 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

Actual Parameters		Estimated Parameters			
μ	τ_0	μ		τ_0	
		Mean	Std.	Mean	Std.
0.1	0.01	0.0970	0.0072	0.0065	0.0036
0.1	1.00	0.0974	0.0070	1.0563	0.5550
0.1	100	0.0981	0.0080	123.9806	66.7454
0.5	0.01	0.5251	0.0356	0.0133	0.0018
0.5	1.00	0.5190	0.0416	0.8390	0.1234
0.5	100	0.5241	0.0328	89.6304	13.0851
0.9	0.01	0.8870	0.0489	0.0105	6.0704×10^{-4}
0.9	1.00	0.8829	0.0426	0.9722	0.0717
0.9	100	0.8843	0.0486	109.1119	6.8380

Table 4.6: Mean and standard deviation estimates for the ABC method

in the supplemental materials[6]. The ABC algorithm uses a uniform prior for μ and τ_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.

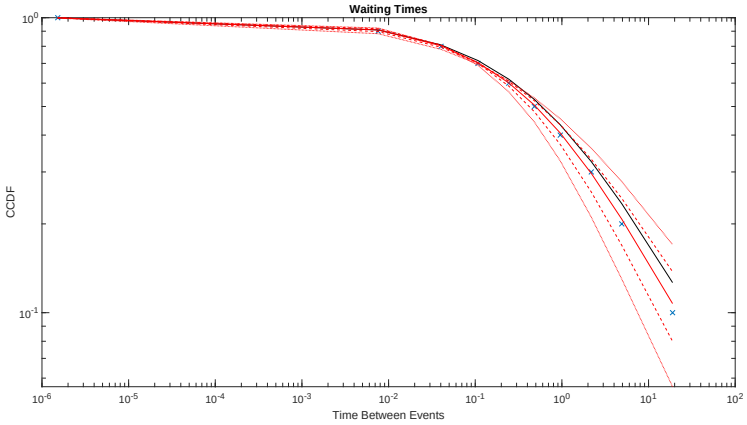
Similar to the results on generated data using the exact and MCMC methods, the ABC method returns good approximations to the true values, with most estimates being within 1-2 standard deviations of these. As is expected, the ABC method does not return estimates as accurately as these other methods (as 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.

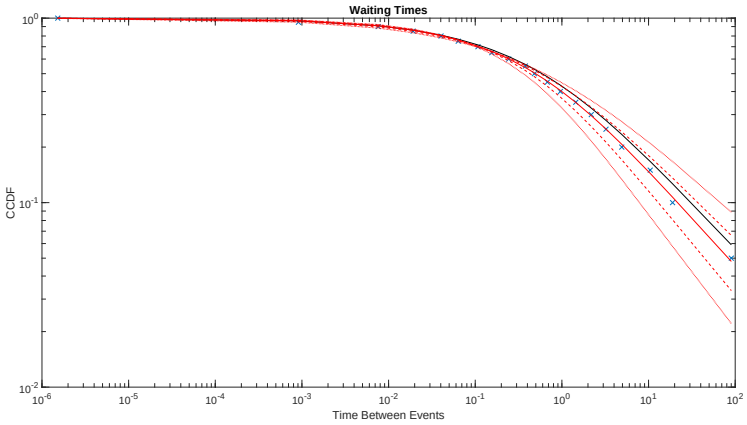
4.3.3.1 Comments on Data Size

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

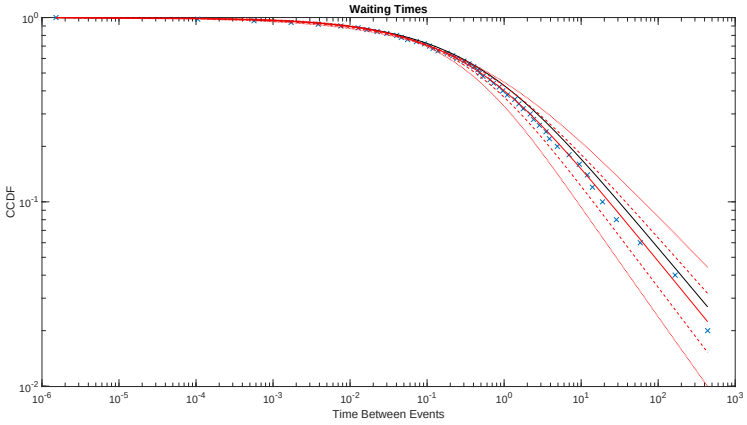
[6] Available from `https://drive.google.com/open?id=13eUXp3xGph1B5T7dlIPvEVPZcHyACmom`.



(a) 10 summary data points



(b) 20 summary data points



(c) 100 summary data points

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.

Sample Size	Estimated Parameters			
	μ		τ_0	
	Mean	Std.	Mean	Std.
10	0.5186	0.0428	0.8415	0.1342
20	0.5190	0.0416	0.8390	0.1234
50	0.5118	0.0358	0.8353	0.1247

Table 4.7: Mean and standard deviation estimates for the ABC method for various data sample sizes

Data	Method	Estimated Parameters			
		μ		τ_0	
		Mean	Std.	Mean	Std.
RDEP	Stage-Fedotov	0.8400	–	23.2558	–
	ABC	0.7687	0.0579	24.0621	2.3633
DHR	Stage-Fedotov	0.8900	–	9.4340	–
	ABC	0.7954	0.0462	9.3578	0.6022

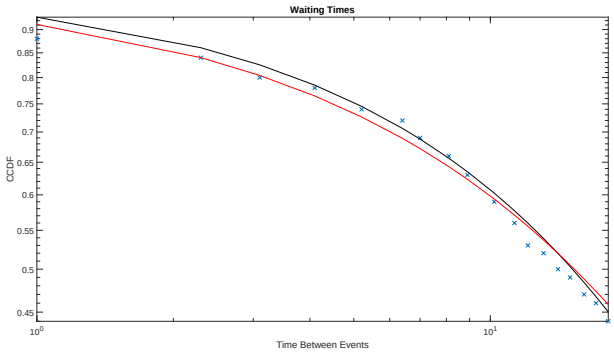
Table 4.8: Mean and standard deviation estimates for recidivism of drug ex-prisoners (RDEP) and duration of human residence (DHR)

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.

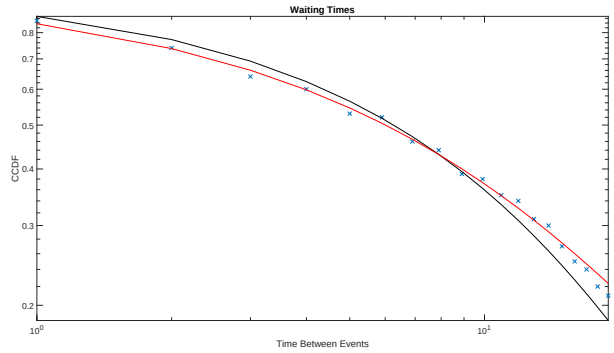
Similarly to the results for the exact and MCMC methods, the ABC method does show improvement as it is given more data, although the degree of this improvement is fairly small. However, based on these results, I believe that using 20 pieces of summary data is justified as the algorithm produces fairly 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-prisioners 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 $\Psi(\tau) = E_\mu\left(-(\tau/\tau_0)^\mu\right)$ (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 μ are taken from [186], the Stage values for τ_0 have had to be approximated using the graphical results presented in the paper. The ABC algorithm uses a uniform prior for μ and τ_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.



(a) Recidivism of drug ex-prisoners



(b) Duration of human residence

Figure 4.14: CCDFs comparing fits generated by Stage and Fedotov (black) to those generated by the ABC (red) and the original summary data (blue)

From this table, it is noticeable that the estimates for τ_0 generated from the Stage graphical results are very close to those estimated by the ABC, although the estimates for μ 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]:

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

$$AIC(\boldsymbol{x}|\boldsymbol{\theta}) = 2k - 2\log(P(\boldsymbol{x}|\boldsymbol{\theta})). \tag{5.2}$$

$$BIC(\boldsymbol{x}|\boldsymbol{\theta}) = \log(n)k - 2\log(P(\boldsymbol{x}|\boldsymbol{\theta})). \tag{5.3}$$

In the definition of the L^2 distance, I assume a ECCDF for the data evaluated at discrete points, which has values $\mathbf{p}$, and a CCDF of the chosen distribution and parameters evaluated at the points, which has values $\mathbf{q}$. In the definition of AIC and BIC, θ refers to the chosen parameters to be examined, k refers to the number of parameters being estimated and n to the number of observations. $P(\boldsymbol{x}|\boldsymbol{\theta})$ 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 [107]. 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 $\log(P(\boldsymbol{x}|\boldsymbol{\theta}))$, else AIC will indicate a poorer model [111]. The combination of these terms results in a trade-off between under-fitting and over-fitting that is core to developing a good model [40].

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 [40].

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

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 [5, 40]. 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 [203]. 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 [40].

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[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.

[1] See appendix A.2 for details.

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

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 μ^* and τ_0^* is represented as $G_n(\mu^*, \tau_0^*)$.

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 τ_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

[1] H. Abbey. An examination of the Reed-Frost theory of epidemics. *Human Biology*, 24(3):201–233, Sept. 1952.

[2] K. T. Abou-Moustafa and F. P. Ferrie. A note on metric properties for some divergence measures: The Gaussian case. In S. C. H. Hoi and W. Buntine, editors, *Proceedings of the Asian Conference on Machine Learning*, volume 25 of *Proceedings of Machine Learning Research*, pages 1–15, Singapore Management University, Singapore, Nov. 2012. PMLR.

[3] S. Achard. A resilient, low-frequency, small-world human brain functional network with highly connected association cortical hubs. *Journal of Neuroscience*, 26(1):63–72, Jan. 2006.

[4] S. Aguiñaga, R. Palacios, D. Chiang, and T. Weninger. Growing graphs from hyperedge replacement graph grammars. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management - CIKM '16*, pages 469–478. ACM Press, 2016.

[5] K. Aho, D. Derryberry, and T. Peterson. Model selection for ecologists: the worldviews of AIC and BIC. *Ecology*, 95(3):631–636, Mar. 2014.

[6] S. Aja-Fernández and G. Vegas-Sánchez-Ferrero. *Statistical Analysis of Noise in MRI*. Springer International Publishing, 2016.

[7] A. S. Akhter and A. S. Hirai. Estimation of the scale parameter from the Rayleigh distribution from type II singly and doubly censored data. *Pakistan Journal of Statistics and Operation Research*, 5(1):31, Jan. 2009.

[8] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, Jan. 2002.

[9] R. Albert, H. Jeong, and A.-L. Barabási. Diameter of the world-wide web. *Nature*, 401(6749):130–131, Sept. 1999.

[10] R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, July 2000.

[11] G. M. Ames, D. B. George, C. P. Hampson, A. R. Kanarek, C. D. McBee, D. R. Lockwood, J. D. Achter, and C. T. Webb. Using network properties to predict disease dynamics on human contact networks. *Proceedings of the Royal Society B: Biological Sciences*, 278(1724):3544–3550, Apr. 2011.

[12] R. M. Anderson. *Infectious Diseases of Humans*. OUP Oxford, 1992.

[13] P. Andersson. Formal languages: Hyperedge replacement grammars. `https://www8.cs.umu.se/kurser/TDBC92/VT06/final/4.pdf`, 2006.

[14] S. Anily, J. Hornik, and M. Israeli. Inferring the distribution of households' duration of residence from data on current residence time. *Journal of Business & Economic Statistics*, 17(3):373–381, July 1999.

[15] T. B. Arnold and J. W. Emerson. Nonparametric goodness-of-fit tests for discrete null distributions. *The R Journal*, 3(2):34, Dec. 2011.

[16] S. Ashton, E. Scalas, N. Georgiou, and I. Z. Kiss. The mathematics of human contact: Developing a model for social interaction in school children. *Acta Physica Polonica A*, 133(6):1421–1432, June 2018.

[17] S. Asmussen, J. L. Jensen, and L. Rojas-Nandayapa. On the Laplace transform of the lognormal distribution. *Methodology and Computing in Applied Probability*, 18(2):441–458, Dec. 2014.

[18] G. J. Babu and A. Toreti. A goodness-of-fit test for heavy tailed distributions with unknown parameters and its application to simulated precipitation extremes in the Euro-Mediterranean region. *Journal of Statistical Planning and Inference*, 174:11–19, July 2016.

[19] N. T. J. Bailey. *The Mathematical Theory of Infectious Diseases*. Hafner Press, 1975.

[20] P. Bak, K. Chen, and C. Tang. A forest-fire model and some thoughts on turbulence. *Physics Letters A*, 147(5-6):297–300, July 1990.

[21] S. Bansal, B. T. Grenfell, and L. A. Meyers. When individual behaviour matters: homogeneous and network models in epidemiology. *Journal of The Royal Society Interface*, 4(16):879–891, Oct. 2007.

[22] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, Oct. 1999.

[23] M. Barthélemy. Spatial networks. *Physics Reports*, 499(1–3):1–101, Feb. 2011.

[24] M. A. Beaumont. Approximate bayesian computation in evolution and ecology. *Annual Review of Ecology, Evolution, and Systematics*, 41(1):379–406, Dec. 2010.

[25] F. Beichelt and L. P. Fatti. *Stochastic Processes and Their Applications*. CRC Press, 2001.

[26] M. T. Belachew and N. Gillis. Solving the maximum clique problem with symmetric rank-one nonnegative matrix approximation. *arXiv preprints*, page 1505.07077, 2015.

[27] A. V. Belikov. The number of key carcinogenic events can be predicted from cancer incidence. *Scientific Reports*, 7(1):12170, Sept. 2017.

[28] E. A. Bender and S. G. Williamson. *Lists, Decisions and Graphs with an Introduction to Probability*. University of California, San Diego, 2010.

[29] E. Berger. Dynamic monopolies of constant size. *Journal of Combinatorial Theory, Series B*, 83(2):191–200, Nov. 2001.

[30] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York Inc., 2006.

[31] J. R. S. Blair and B. Peyton. An introduction to chordal graphs and clique trees. In *Graph Theory and Sparse Matrix Computation*, pages 1–29. Springer New York, 1993.

[32] B. Bollobas. *Random Graphs*. Cambridge University Press, 2001.

[33] K. O. Bowman and L. R. Shenton. Estimation: Method of moments. In S. Kotz, C. B. Read, N. Balakrishnan, and B. Vidakovic, editors, *Encyclopedia of Statistical Sciences*, volume 3, pages 2092–2098. American Cancer Society, 2006.

[34] B. B. Brabson and J. P. Palutikof. Tests of the generalized Pareto distribution for predicting extreme wind speeds. *Journal of Applied Meteorology*, 39(9):1627–1640, Sept. 2000.

[35] F. Brauer. Compartmental models in epidemiology. In F. Brauer, P. van den Driessche, and J. Wu, editors, *Mathematical Epidemiology*, pages 19–79. Springer Berlin Heidelberg, 2008.

[36] T. Britton. Stochastic epidemic models: A survey. *Mathematical Biosciences*, 225(1):24–35, May 2010.

[37] D. Brockmann. Introduction to complex systems: Introduction to complex networks. `http://rocs.hu-berlin.de/complex_sys_2015/resources/Introduction-to-Networks.pdf`, June 2015.

[38] M. C. Bryson. Heavy-tailed distributions: Properties and tests. *Technometrics*, 16(1):61–68, Feb. 1974.

[39] E. Bullmore and O. Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature Reviews Neuroscience*, 10(3):186–198, Feb. 2009.

[40] K. P. Burnham and D. R. Anderson. *Model Selection and Multimodel Inference*. Springer New York, 2004.

[41] D. J. T. Carter and P. G. Challenor. Application of extreme value analysis to Weibull data. *Quarterly Journal of the Royal Meteorological Society*, 109(460):429–433, Apr. 1983.

[42] S. Chauhan, O. P. Misra, and J. Dhar. Stability analysis of SIR model with vaccination. *American Journal of Computational and Applied Mathematics*, 4(1):17–23, 2014.

[43] D. Chiang, K. M. Hermann, J. Andreas, B. Jones, D. Bauer, and K. Knight. Parsing graphs with hyperedge replacement grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 924–932, 2013.

[44] A. Clauset, M. Young, and K. S. Gleditsch. On the frequency of severe terrorist events. *Journal of Conflict Resolution*, 51(1):58–87, Feb. 2007.

[45] R. Cohen, S. Havlin, and D. ben Avraham. Efficient immunization strategies for computer networks and populations. *Physical Review Letters*, 91(24):247901, Dec. 2003.

[46] S. Coles. *An Introduction to Statistical Modeling of Extreme Values*. Springer-Verlag GmbH, 2001.

[47] N. S. Contractor, S. Wasserman, and K. Faust. Testing multitheoretical, multilevel hypotheses about organizational networks: An analytic framework and empirical example. *Academy of Management Review*, 31(3):681–703, July 2006.

[48] L. Danon, A. P. Ford, T. House, C. P. Jewell, M. J. Keeling, G. O. Roberts, J. V. Ross, and M. C. Vernon. Networks and the epidemiology of infectious disease. *Interdisciplinary Perspectives on Infectious Diseases*, 2011:1–28, 2011.

[49] G. R. Dargahi-Noubary. On tail estimation: An improved method. *Mathematical Geology*, 21(8):829–842, Oct. 1989.

[50] S. DeDeo, R. Hawkins, S. Klingenstein, and T. Hitchcock. Bootstrap methods for the empirical study of decision-making and information flows in social systems. *Entropy*, 15(12):2246–2276, June 2013.

[51] A. den Dekker and J. Sijbers. Data distributions in magnetic resonance images: A review. *Physica Medica*, 30(7):725–741, Nov. 2014.

[52] B. S. Dhillion. Topics in Reliability. In *Engineering Maintainability*, pages 224–249. Elsevier, 1999.

[53] J. Diebolt, M. Garrid, and S. Girard. A goodness-of-fit test for the distribution tail. In M. Ahsanullah and S. Kirmani, editors, *Topics in Extreme Values*, chapter 5, pages 95–110. Nova Science, 2008.

[54] R. Diestel. *Graph Theory*. Springer-Verlag GmbH, 2017.

[55] K. Dietz. Epidemics and rumours: A survey. *Journal of the Royal Statistical Society. Series A (General)*, 130(4):505, Jan. 1967.

[56] F. Drewes, H.-J. Kreowski, and A. Habel. Hyperedge replacement graph grammars. In *Handbook of Graph Grammars and Computing by Graph Transformation*, pages 95–162. World Scientific, Feb. 1997.

[57] B. Drossel and F. Schwabl. Self-organized critical forest-fire model. *Physical Review Letters*, 69(11):1629–1632, Sept. 1992.

[58] A. Duncan. Powers of the adjacency matrix and the walk matrix. *The Collection*, 9:4–11, 2004.

[59] R. Durrett. Some features of the spread of epidemics and information on a random graph. *Proceedings of the National Academy of Sciences*, 107(10):4491–4498, Feb. 2010.

[60] K. T. D. Eames and M. J. Keeling. Modeling dynamic and network heterogeneities in the spread of sexually transmitted diseases. *Proceedings of the National Academy of Sciences*, 99(20):13330–13335, Sept. 2002.

[61] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets*. Cambridge University Press, 2010.

[62] E. A. Enns and M. L. Brandeau. Inferring model parameters in network-based disease simulation. *Health Care Management Science*, 14(2):174–188, Mar. 2011.

[63] P. Erdős and A. Rényi. On random graphs, I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[64] S. Eubank. Network based models of infectious disease spread. *Japanese Journal of Infectious Diseases*, 58:S9–13, Dec. 2005.

[65] S. Eubank, H. Guclu, V. S. A. Kumar, M. V. Marathe, A. Srinivasan, Z. Toroczkai, and N. Wang. Modelling disease outbreaks in realistic urban social networks. *Nature*, 429(6988):180–184, May 2004.

[66] T. S. Evans. Clique graphs and overlapping communities. *arXiv preprint*, page 1009.0638, 2010.

[67] F. J. Fabozzi, S. M. Focardi, S. T. Rachev, and B. G. Arshanapalli. *The Basics of Financial Econometrics*. John Wiley & Sons, Inc., Mar. 2014.

[68] S. Ferraro, M. Manzini, A. Masoero, and E. Scalas. A random telegraph signal of Mittag-Leffler type. *Physica A: Statistical Mechanics and its Applications*, 388(19):3991–3999, Oct. 2009.

[69] C. Forbes, M. Evans, N. Hastings, and B. Peacock. *Statistical Distributions*. John Wiley & Sons, 2010.

[70] O. Frank and D. Strauss. Markov graphs. *Journal of the American Statistical Association*, 81(395):832–842, Sept. 1986.

[71] B. Fuglede and F. Topsøe. Jensen-Shannon divergence and Hilbert space embedding. In *International Symposium on Information Theory, 2004. ISIT 2004. Proceedings.*, page 31. IEEE, 2004.

[72] K. Fukuda, K. Sugawa, and K. Ishii. Simulation of infectious disease by reed-frost model with proportion of immune and inapparent infection. *Computers in Biology and Medicine*, 14(2):209–215, Jan. 1984.

[73] R. Garrappa. The Mittag-Leffler function. `https://uk.mathworks.com/matlabcentral/fileexchange/48154-the-mittag-leffler-function`, Dec. 2015.

[74] V. Gemmetto, A. Barrat, and C. Cattuto. Mitigation of infectious disease at school: targeted class closure vs school closure. *BMC Infectious Diseases*, 14(1):695, Dec. 2014.

[75] N. Georgiou, I. Z. Kiss, and E. Scalas. Solvable non-Markovian dynamic network. *Physical Review E*, 92(4):042801, Oct. 2015.

[76] G. Germano, D. Fulger, and E. Scalas. Mittag-Leffler random number generator. `https://uk.mathworks.com/matlabcentral/fileexchange/19392-mittag-leffler-random-number-generator`, Apr. 2016.

[77] E. N. Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, Dec. 1959.

[78] J. Goldenberg, B. Libai, and E. Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, 12(3):211–223, Aug. 2001.

[79] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *arXiv preprint*, page 1406.2661, 2014.

[80] M. Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83(6):1420–1443, May 1978.

[81] T. L. Graves. Automatic step size selection in random walk Metropolis algorithms. *arXiv preprint*, page 1103.5986, Mar. 2011.

[82] A. Gray, D. Greenhalgh, X. Mao, and J. Pan. The SIS epidemic model with markovian switching. *Journal of Mathematical Analysis and Applications*, 394(2):496–516, Oct. 2012.

[83] P. E. Greenwood and L. F. Gordillo. Stochastic epidemic modeling. In *Mathematical and Statistical Estimation Approaches in Epidemiology*, pages 31–52. Springer Netherlands, 2009.

[84] B. T. Grenfell, O. N. Bjørnstad, and J. Kappey. Travelling waves and spatial hierarchies in measles epidemics. *Nature*, 414(6865):716–723, Dec. 2001.

[85] M. Griesser, Q. Ma, S. Webber, K. Bowgen, and D. J. T. Sumpter. Understanding animal group-size distributions. *PLoS ONE*, 6(8):e23438, Aug. 2011.

[86] P. Haggett and R. J. Chorley. *Network Analysis in Geography*. Hodder & Stoughton Educational, 1969.

[87] M. E. Halloran. Containing bioterrorist smallpox. *Science*, 298(5597):1428–1432, Nov. 2002.

[88] T. E. Harris. Contact interactions on a lattice. *The Annals of Probability*, 2(6):969–988, Dec. 1974.

[89] A. G. Hawkes. Point spectra of some mutually exciting point processes. *Journal of the Royal Statistical Society: Series B (Methodological)*, 33(3):438–443, Oct. 1971.

[90] H. W. Hethcote. The mathematics of infectious diseases. *SIAM Review*, 42(4):599–653, Jan. 2000.

[91] H. W. Hethcote and J. A. Yorke. *Gonorrhea Transmission Dynamics and Control*. Springer Berlin Heidelberg, 1984.

[92] J. Holmes and W. Moriarty. Application of the generalized Pareto distribution to extreme value analysis in wind engineering. *Journal of Wind Engineering and Industrial Aerodynamics*, 83(1-3):1–10, Nov. 1999.

[93] J. R. M. Hosking and J. R. Wallis. Parameter and quantile estimation for the generalized Pareto distribution. *Technometrics*, 29(3):339–349, Aug. 1987.

[94] T. E. Huillet. On Mittag-Leffler distributions and related stochastic processes. *Journal of Computational and Applied Mathematics*, 296:181–211, Apr. 2016.

[95] L. Huyse, R. Chen, and J. A. Stamatakos. Application of generalized pareto distribution to constrain uncertainty in peak ground accelerations. *Bulletin of the Seismological Society of America*, 100(1):87–101, Jan. 2010.

[96] O. C. Ibe. Introduction to markov processes. In *Markov Processes for Stochastic Modeling*, pages 49–57. Elsevier, 2013.

[97] H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A.-L. Barabási. The large-scale organization of metabolic networks. *Nature*, 407(6804):651–654, Oct. 2000.

[98] C. Jiao, T. Wang, J. Liu, H. Wu, F. Cui, and X. Peng. Using exponential random graph models to analyze the character of peer relationship networks and their effects on the subjective well-being of adolescents. *Frontiers in Psychology*, 8:583, Apr. 2017.

[99] C. G. Justus, W. R. Hargraves, A. Mikhail, and D. Graber. Methods for estimating wind speed frequency distributions. *Journal of Applied Meteorology*, 17(3):350–353, Mar. 1978.

[100] M. Keeling. The implications of network structure for epidemic dynamics. *Theoretical Population Biology*, 67(1):1–8, Feb. 2005.

[101] M. J. Keeling and L. Danon. Mathematical modelling of infectious diseases. *British Medical Bulletin*, 92(1):33–42, Oct. 2009.

[102] M. J. Keeling and K. T. Eames. Networks and epidemic models. *Journal of The Royal Society Interface*, 2(4):295–307, Sept. 2005.

[103] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '03*, pages 137–146. ACM Press, 2003.

[104] I. Z. Kiss, L. Berthouze, T. J. Taylor, and P. L. Simon. Modelling approaches for simple dynamic networks and applications to disease transmission models. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 468(2141):1332–1355, Jan. 2012.

[105] I. Z. Kiss, J. C. Miller, and P. L. Simon. *Mathematics of Epidemics on Networks*. Springer International Publishing, 2017.

[106] I. Z. Kiss, C. G. Morris, F. Sélley, P. L. Simon, and R. R. Wilkinson. Exact deterministic representation of Markovian SIR epidemics on networks with and without loops. *Journal of Mathematical Biology*, 70(3):437–464, Mar. 2014.

[107] C. Kisslinger. Aftershocks and fault-zone properties. In R. Dmowska and B. Saltzman, editors, *Advances in Geophysics*, volume 38, pages 1–36. Elsevier, 1996.

[108] A. Klaus, S. Yu, and D. Plenz. Statistical analyses support power law distributions found in neuronal avalanches. *PLoS ONE*, 6(5):e19779, May 2011.

[109] A. S. Klovdahl, Z. Dhofier, G. Oddy, J. O'Hara, S. Stoutjesdijk, and A. Whish. Social networks in an urban area: First Canberra study. *The Australian and New Zealand Journal of Sociology*, 13(2):169–172, Aug. 1977.

[110] S. Klus and T. Sahai. A spectral assignment approach for the graph isomorphism problem. *arXiv preprint*, page 1411.0969, 2014.

[111] F. Korner-Nievergelt, T. Roth, S. von Felten, J. Guélat, B. Almasi, and P. Korner-Nievergelt. Model selection and multimodel inference. In *Bayesian Data Analysis in Ecology Using Linear Models with R, BUGS, and STAN*, chapter 11, pages 175–196. Elsevier, 2015.

[112] B. Lambert. *A Student's Guide to Bayesian Statistics*. SAGE Publications Ltd, 2018.

[113] D. Larremore. Find network components. `https://uk.mathworks.com/matlabcentral/fileexchange/42040-find-network-components`, Apr. 2014.

[114] V. Latora, V. Nicosia, and G. Russo. *Complex Networks*. Cambridge University Press, Sept. 2017.

[115] E. H. Lau, C. A. Hsiung, B. J. Cowling, C.-H. Chen, L.-M. Ho, T. Tsang, C.-W. Chang, C. A. Donnelly, and G. M. Leung. A comparative epidemiologic analysis of SARS in Hong Kong, Beijing and Taiwan. *BMC Infectious Diseases*, 10(1):50, Mar. 2010.

[116] T. M. Liggett. *Interacting Particle Systems*. Springer Berlin Heidelberg, 2005.

[117] F. Liljeros, C. R. Edling, L. A. N. Amaral, H. E. Stanley, and Y. Åberg. The web of human sexual contacts. *Nature*, 411(6840):907–908, June 2001.

[118] G. D. Lin. On the Mittag-Leffler distributions. *Journal of Statistical Planning and Inference*, 74(1):1–9, Oct. 1998.

[119] A. L. Lloyd. How viruses spread among computers and people. *Science*, 292(5520):1316–1317, May 2001.

[120] D. J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Pr., 2003.

[121] M. W. Macy. Chains of cooperation: Threshold effects in collective action. *American Sociological Review*, 56(6):730, Dec. 1991.

[122] M. W. Macy and R. Willer. From factors to actors: Computational sociology and agent-based modeling. *Annual Review of Sociology*, 28(1):143–166, Aug. 2002.

[123] M. Marozzi. Some notes on the location–scale cucconi test. *Journal of Nonparametric Statistics*, 21(5):629–647, July 2009.

[124] D. M. Mason and J. H. Schuenemeyer. A modified kolmogorov-smirnov test sensitive to tail alternatives. *The Annals of Statistics*, 11(3):933–946, Sept. 1983.

[125] R. Mastrandrea, J. Fournet, and A. Barrat. Contact patterns in a high school: A comparison between data collected using wearable sensors, contact diaries and friendship surveys. *PLOS ONE*, 10(9):e0136497, Sept. 2015.

[126] L. A. Meyers, M. Newman, M. Martin, and S. Schrag. Applying network theory to epidemics: Control measures for Mycoplasma pneumoniae outbreaks. *Emerging Infectious Diseases*, 9(2):204–210, Feb. 2003.

[127] L. A. Meyers, B. Pourbohloul, M. Newman, D. M. Skowronski, and R. C. Brunham. Network theory and SARS: predicting outbreak diversity. *Journal of Theoretical Biology*, 232(1):71–81, Jan. 2005.

[128] A. Miao, J. Zhang, T. Zhang, and B. G. S. A. Pradeep. Threshold dynamics of a stochastic SIR model with vertical transmission and vaccination. *Computational and Mathematical Methods in Medicine*, 2017:1–10, July 2017.

[129] J. Miles. The Laplace transform of the lognormal distribution. *arXiv preprint*, page 1803.05878, 2018.

[130] P. Minasandra and K. Isvaran. Truncated power-law distribution of group sizes in antelope. *bioRxiv*, page 411199, Sept. 2018.

[131] E. A. Mohammed, C. Naugler, and B. H. Far. Emerging business intelligence framework for a clinical laboratory through big data analytics. In Q. N. Tran and H. Arabnia, editors, *Emerging Trends in Computational Biology, Bioinformatics, and Systems Biology*, chapter 32, pages 577–602. Elsevier, 2015.

[132] D. Mollison. Spatial contact models for ecological and epidemic spread. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(3):283–313, July 1977.

[133] S. Morris. Contagion. *Review of Economic Studies*, 67(1):57–78, Jan. 2000.

[134] J. Mun, editor. *Advanced Analytical Models*. John Wiley & Sons, Inc., Jan. 2012.

[135] J. Nair, A. Wierman, and B. Zwart. The fundamentals of heavy-tails. In *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems - SIGMETRICS '13*, pages 387–388. ACM Press, 2013.

[136] M. E. J. Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences*, 98(2):404–409, Jan. 2001.

[137] M. E. J. Newman. Spread of epidemic disease on networks. *Physical Review E*, 66(1):016128, July 2002.

[138] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, Jan. 2003.

[139] F. Nielsen and V. Garcia. Statistical exponential families: A digest with flash cards. *arXiv preprint*, page 0911.4863, 2009.

[140] O. Ozel, B. Sinopoli, and O. Yagan. Robustness of flow networks against cascading failures under partial load redistribution. *arXiv preprint*, page 1802.07664, Oct. 2018.

[141] R. Pastor-Satorras and A. Vespignani. Epidemic spreading in scale-free networks. *Physical Review Letters*, 86(14):3200–3203, Apr. 2001.

[142] R. Pastor-Satorras and A. Vespignani. Immunization of complex networks. *Physical Review E*, 65(3):036104, Feb. 2002.

[143] R. Pastor-Satorras and A. Vespignani. Epidemics and immunization in scale-free networks. In S. Bornholdt and H. G. Schuster, editors, *Handbook of Graphs and Networks*, chapter 5, pages 111–130. Wiley-VCH Verlag GmbH & Co. KGaA, Dec. 2004.

[144] P. Pattison and S. Wasserman. Logit models and logistic regressions for social networks: II. multivariate relations. *British Journal of Mathematical and Statistical Psychology*, 52(2):169–193, Nov. 1999.

[145] G. A. Pavlopoulos, M. Secrier, C. N. Moschopoulos, T. G. Soldatos, S. Kossida, J. Aerts, R. Schneider, and P. G. Bagos. Using graph theory to analyze biological networks. *BioData Mining*, 4(1):10, Apr. 2011.

[146] R. J. Pavur, R. L. Edgeman, and R. C. Scott. Quadratic statistics for the goodness-of-fit test of the inverse Gaussian distribution. *IEEE Transactions on Reliability*, 41(1):118–123, Mar. 1992.

[147] D. Peleg. Local majorities, coalitions and monopolies in graphs: a review. *Theoretical Computer Science*, 282(2):231–257, June 2002.

[148] C. Pennycuff, S. Aguinaga, and T. Weninger. A temporal tree decomposition for generating temporal graphs. *arXiv preprint*, page 1706.09480, June 2017.

[149] N. Perra, B. Gonçalves, R. Pastor-Satorras, and A. Vespignani. Activity driven modeling of time varying networks. *Scientific Reports*, 2(1):469, June 2012.

[150] R. N. Pillai. On Mittag-Leffler functions and related distributions. *Annals of the Institute of Statistical Mathematics*, 42(1):157–161, 1990.

[151] I. Podlubny and M. Kacenak. Mittag-Leffler function. `https://uk.mathworks.com/matlabcentral/fileexchange/8738-mittag-leffler-function`, Mar. 2009.

[152] M. Porta, editor. *A Dictionary of Epidemiology*. Oxford University Press, Jan. 2014.

[153] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in Fortran 77: The Art of Scientific Computing*. Cambridge University Press, 1992.

[154] S. Purohit, L. B. Holder, and G. Chin. Temporal graph generation based on a distribution of temporal motifs. In *Proceedings of the 14th International Workshop on Mining and Learning with Graphs*, Aug. 2018.

[155] J. M. Read and M. J. Keeling. Disease evolution on networks: the role of contact structure. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 270(1516):699–708, Apr. 2003.

[156] S. Riley. Transmission dynamics of the etiological agent of SARS in Hong Kong: Impact of public health interventions. *Science*, 300(5627):1961–1966, June 2003.

[157] C. P. Robert. The Metropolis-Hastings algorithm. *arXiv preprint*, page 1504.01896, 2015.

[158] G. O. Roberts and J. S. Rosenthal. Optimal scaling for various Metropolis-Hastings algorithms. *Statistical Science*, 16(4):351–367, Nov. 2001.

[159] G. Robins and D. Lusher. What are exponential random graph models? In D. Lusher, J. Koskinen, and G. Robins, editors, *Exponential Random Graph Models for Social Networks*, pages 9–15. Cambridge University Press, 2012.

[160] G. Robins, P. Pattison, and S. Wasserman. Logit models and logistic regressions for social networks: III. valued relations. *Psychometrika*, 64(3):371–394, Sept. 1999.

[161] J. G. Robson and J. B. Troy. Nature of the maintained discharge of Q, X, and Y retinal ganglion cells of the cat. *Journal of the Optical Society of America. A, Optics and image science*, 4:2301–2307, Dec. 1987.

[162] L. E. C. Rocha and N. Masuda. Individual-based approach to epidemic processes on arbitrary dynamic contact networks. *Scientific Reports*, 6(1):31456, Aug. 2016.

[163] T. Rolski, H. Schmidli, V. Schmidt, and J. Teugels, editors. *Stochastic Processes for Insurance & Finance*. John Wiley & Sons, Inc., Feb. 1999.

[164] S. Ross. *Introduction to Probability Models*. Elsevier LTD, Oxford, 2019.

[165] A. F. Rozenfeld, R. Cohen, D. ben Avraham, and S. Havlin. Scale-free networks on lattices. *Physical Review Letters*, 89(21):218701, Nov. 2002.

[166] P. Sah, M. Otterstatter, S. T. Leu, S. Leviyang, and S. Bansal. Revealing mechanisms of infectious disease transmission through empirical contact networks. *bioRxiv*, page 169573, July 2017.

[167] N. F. Samatova, W. Hendrix, J. Jenkins, K. Padmanabhan, and A. Chakraborty, editors. *Practical Graph Mining with R*, chapter An introduction to graph theory, pages 9–26. CRC Press, 2014.

[168] M. Sánchez-Silva and J. Riascos-Ochoa. Seismic risk models for aging and deteriorating buildings and civil infrastructure. In S. Tesfamariam and K. Goda, editors, *Handbook of Seismic Risk Analysis and Management of Civil Infrastructure Systems*, pages 387–409. Elsevier, 2013.

[169] E. Scalas, R. Gorenflo, and F. Mainardi. Fractional calculus and continuous-time finance. *Physica A: Statistical Mechanics and its Applications*, 284(1-4):376–384, Sept. 2000.

[170] T. C. Schelling. *Micromotives and Macrobehavior*. WW Norton & Co, 2006.

[171] F. W. Scholz. Maximum likelihood estimation. In S. Kotz, C. B. Read, N. Balakrishnan, and B. Vidakovic, editors, *Encyclopedia of Statistical Sciences*, volume 7, pages 4629–4639. American Cancer Society, 2006.

[172] J. Seguro and T. Lambert. Modern estimation of the parameters of the weibull wind speed distribution for wind energy analysis. *Journal of Wind Engineering and Industrial Aerodynamics*, 85(1):75–84, Mar. 2000.

[173] P. Shakarian, A. Bhatnagar, A. Aleali, E. Shaabani, and R. Guo. The independent cascade and linear threshold models. In *SpringerBriefs in Computer Science*, pages 35–48. Springer International Publishing, 2015.

[174] P. Shakarian, A. Bhatnagar, A. Aleali, E. Shaabani, and R. Guo. The SIR model and identification of spreaders. In *SpringerBriefs in Computer Science*, pages 3–18. Springer International Publishing, 2015.

[175] J. Sijbers, A. J. den Dekker, E. Raman, and D. V. Dyck. Parameter estimation from magnitude MR images. *International Journal of Imaging Systems and Technology*, 10(2):109–114, Mar. 1999.

[176] M. Simkin and V. Roychowdhury. Re-inventing Willis. *Physics Reports*, Dec. 2010.

[177] G. E. Sims, S.-R. Jun, G. A. Wu, and S.-H. Kim. Alignment-free genome comparison with feature frequency profiles (FFP) and optimal resolutions. *Proceedings of the National Academy of Sciences*, 106(8):2677–2682, Feb. 2009.

[178] V. P. Singh and H. Guo. Parameter estimation for 3-parameter generalized Pareto distribution by the principle of maximum entropy (POME). *Hydrological Sciences Journal*, 40(2):165–181, Apr. 1995.

[179] S. A. Sisson, Y. Fan, and M. A. Beaumont. Overview of approximate bayesian computation. *arXiv preprint*, page 1802.09720, Feb. 2018.

[180] B. Skyrms and R. Pemantle. A dynamic model of social network formation. *Proceedings of the National Academy of Sciences*, 97(16):9340–9346, Aug. 2000.

[181] N. V. Smirnov. On the estimate of the discrepancy between empirical curves of distributions for two independent samples. *Mathematical Bulletin of Moscow University*, 2(2):3–16, 1939.

[182] T. A. B. Snijders, P. E. Pattison, G. L. Robins, and M. S. Handcock. New specifications for exponential random graph models. *Sociological Methodology*, 36(1):99–153, Aug. 2006.

[183] P. Sobkowicz, M. Thelwall, K. Buckley, G. Paltoglou, and A. Sobkowicz. Lognormal distributions of user post lengths in internet discussions - a consequence of the Weber-Fechner law? *EPJ Data Science*, 2(1):2, Feb. 2013.

[184] W. T. Song. Relationships among some univariate distributions. *IIE Transactions*, 37(7):651–656, July 2005.

[185] C. Spohn and F. Holleran. The efect of imprisonment on recidivism rates of felony offenders: A focus on drug offenders. *Criminology*, 40(2):329–358, May 2002.

[186] H. Stage and S. Fedotov. Anomalous cumulative inertia in human behaviour. *arXiv preprint*, page 1806.00613, June 2018.

[187] J. Stehlé, N. Voirin, A. Barrat, C. Cattuto, V. Colizza, L. Isella, C. Régis, J.-F. Pinton, N. Khanafer, W. V. den Broeck, and P. Vanhems. Simulation of an SEIR infectious disease model on the dynamic contact network of conference attendees. *BMC Medicine*, 9(1):87, July 2011.

[188] J. Stehlé, N. Voirin, A. Barrat, C. Cattuto, L. Isella, J.-F. Pinton, M. Quaggiotto, W. V. den Broeck, C. Régis, B. Lina, and P. Vanhems. High-resolution measurements of face-to-face contact patterns in a primary school. *PLoS ONE*, 6(8):e23176, Aug. 2011.

[189] K. Steinhaeuser and N. V. Chawla. A network-based approach to understanding and predicting diseases. In M. J. Young, J. Salerno, and H. Liu, editors, *Social Computing and Behavioral Modeling*, pages 209–216. Springer US, 2009.

[190] M. A. Stephens. EDF statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737, Sept. 1974.

[191] J. L. Szwarcfiter and C. F. Bornstein. Clique graphs of chordal and path graphs. *SIAM Journal on Discrete Mathematics*, 7(2):331–336, May 1994.

[192] K. Takagi. A distribution model of functional connectome based on criticality and energy constraints. *PLOS ONE*, 12(5):e0177446, May 2017.

[193] M. Taylor, T. J. Taylor, and I. Z. Kiss. Epidemic threshold and control in a dynamic network. *Physical Review E*, 85(1):016103, Jan. 2012.

[194] M. Thelwall and P. Wilson. Regression for citation data: An evaluation of different methods. *Journal of Informetrics*, 8(4):963–971, Oct. 2014.

[195] J. Travers and S. Milgram. An experimental study of the small world problem. *Sociometry*, 32(4):425, Dec. 1969.

[196] United States Geological Survey. USGS earthquake catalog. https://earthquake.usgs.gov/earthquakes/search/.

[197] University of Groningen. Decision making in a complex and uncertain world: Scale-free networks. https://www.futurelearn.com/courses/complexity-and-uncertainty/0/steps/1855.

[198] S. Vajna, B. Tóth, and J. Kertész. Modelling bursty time series. *New Journal of Physics*, 15(10):103023, Oct. 2013.

[199] T. W. Valente. *Network Models of the Diffusion of Innovations (Quantitative Methods in Communication Subseries)*. Hampton Press (NJ), 1995.

[200] T. W. Valente. Social network thresholds in the diffusion of innovations. *Social Networks*, 18(1):69–89, Jan. 1996.

[201] M. A. J. van Montfort and J. V. Witter. The generalized Pareto distribution applied to rainfall depths. *Hydrological Sciences Journal*, 31(2):151–162, June 1986.

[202] P. Vermeesch. Dissimilarity measures in detrital geochronology. *Earth-Science Reviews*, 178:310–321, Mar. 2018.

[203] S. I. Vrieze. Model selection and psychological theory: A discussion of the differences between the akaike information criterion (AIC) and the bayesian information criterion (BIC). *Psychological Methods*, 17(2):228–243, 2012.

[204] J. Wallinga, W. J. Edmunds, and M. Kretzschmar. Perspective: human contact patterns and the spread of airborne infectious diseases. *Trends in Microbiology*, 7:372–377, Sept. 1999.

[205] W. Wang, Q.-H. Liu, S.-M. Cai, M. Tang, L. A. Braunstein, and H. E. Stanley. Suppressing disease spreading by using information diffusion on multiplex networks. *Scientific Reports*, 6(1):29259, July 2016.

[206] C. P. Warren, L. M. Sander, and I. M. Sokolov. Geography in a scale-free network model. *Physical Review E*, 66(5):056105, Nov. 2002.

[207] S. Wasserman and P. Pattison. Logit models and logistic regressions for social networks: I. an introduction to Markov graphs and p. *Psychometrika*, 61(3):401–425, Sept. 1996.

[208] D. J. Watts. A simple model of global cascades on random networks. *Proceedings of the National Academy of Sciences*, 99(9):5766–5771, Apr. 2002.

[209] D. J. Watts. *Small Worlds*. Princeton University Press, 2018.

[210] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

[211] W. Whiten. KS statistic. https://uk.mathworks.com/matlabcentral/fileexchange/47900-ks-statistic-zip, Sept. 2014.

[212] S. Widgren, S. Engblom, P. Bauer, J. Frössling, U. Emanuelson, and A. Lindberg. Data-driven network modelling of disease transmission using complete population movement data: spread of VTEC O157 in Swedish cattle. *Veterinary Research*, 47(1):81, Aug. 2016.

[213] T. Wiecki. MCMC sampling for dummies. `https://twiecki.io/blog/2015/11/10/mcmc-sampling/`, Nov. 2015.

[214] E. Wong, B. Baur, S. Quader, and C.-H. Huang. Biological network motif detection: principles and practice. *Briefings in Bioinformatics*, 13(2):202–215, June 2011.

[215] M. Wright, I. Winter, J. Forster, and S. Bleeck. Response to best-frequency tone bursts in the ventral cochlear nucleus is governed by ordered inter-spike interval statistics. *Hearing Research*, 317:23–32, Nov. 2014.

[216] S. Wuchty. Scale-free behavior in protein domain networks. *Molecular Biology and Evolution*, 18(9):1694–1702, Sept. 2001.

[217] P. Yin, P. Luo, W.-C. Lee, and M. Wang. Silence is also evidence: Interpreting dwell time for recommendation from psychological perspective. In R. Ghani, T. E. Senator, P. Bradley, and R. Parekh, editors, *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*, pages 989–997. ACM Press, 2013.

[218] H. P. Young. *Individual Strategy and Social Structure*. Princeton University Press, 2001.

[219] H. P. Young. The diffusion of innovations in social networks. In L. E. Blume and S. N. Durlauf, editors, *The Economy as an Evolving Complex System, III*, pages 267–282. Oxford University Press, Oct. 2005.

[220] F. Zaidi. Small world networks and clustered small world networks with random connectivity. *Social Network Analysis and Mining*, 3(1):51–63, Feb. 2012.

[221] J. Zhang and M. A. Stephens. A new and efficient estimation method for the generalized pareto distribution. *Technometrics*, 51(3):316–325, Aug. 2009.

[222] X. Zhang and K. Wang. Stochastic SIR model with jumps. *Applied Mathematics Letters*, 26(8):867–874, Aug. 2013.

[223] Z. Zhang. Parametric regression model for survival data: Weibull regression model as an example. *Annals of Translational Medicine*, 4(24):484–484, Dec. 2016.

[224] Y. Zhao and D. Jiang. The threshold of a stochastic SIRS epidemic model with saturated incidence. *Applied Mathematics Letters*, 34:90–93, Aug. 2014.

[225] Y. Zhao, D. Jiang, and D. O'Regan. The extinction and persistence of the stochastic SIS epidemic model with vaccination. *Physica A: Statistical Mechanics and its Applications*, 392(20):4916–4927, Oct. 2013.

[226] Y. D. Zhong, V. Srivastava, and N. E. Leonard. On the linear threshold model for diffusion of innovations in multiplex social networks. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2593–2598. IEEE, Dec. 2017.

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`

Summary ABC method for the Mittag-Leffler distribution. Code is given in appendix B.1.

Key Uses ABC calculations in subsections 4.3.3 and 4.3.4.

Dependencies `ml.m`, `mlrnd.m`[1]

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

Inputs

- `data` - $2 \times n$ 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 τ_0 can be given here as 2 doubles.

Outputs

- `soln_ABC` - structure containing the means and standard deviations of the estimates for μ and τ_0

[1] See 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`[2]

MathWorks Products Required MATLAB, Symbolic Math Toolbox, Statistics and Machine Learning Toolbox, Econometrics Toolbox, Parallel Computing Toolbox, MATLAB Distributed Computing Server

Inputs

- `data` - $1 \times m$ vector of data
- `m1` - $1 \times n$ vector of β parameters for Mittag-Leffler fits
- `m2` - $1 \times n$ vector of τ_0 parameters for Mittag-Leffler fits

Outputs

- `AIC` - $1 \times n$ vector of AIC values of the given Mittag-Leffler fits
- `BIC` - $1 \times n$ vector of BIC values of the given Mittag-Leffler fits
- `L2` - $1 \times n$ 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 $1 \times m$ vector of data.

Working Example `[aic,bic,L2] = aibbic_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.

[2]See 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_TimeBetweenContacts.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`[3]

MathWorks Products Required MATLAB, Symbolic Math Toolbox, Statistics and Machine Learning Toolbox, Curve Fitting Toolbox, Econometrics Toolbox, 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
- `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.

[3]See appendices B.5, B.6, B.7, B.8, B.9, B.10, B.11, B.13, B.14, B.15, B.16, B.17, B.18, B.19, B.20, B.28, B.39, B.41, B.42, B.44, B.51, B.52, B.53, B.54, B.62, B.63, B.67, B.68, B.69, B.70, B.71, B.72, B.73 and B.81 for code.

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`[4]

MathWorks Products Required MATLAB, Symbolic Math Toolbox, Statistics and Machine Learning Toolbox, Curve Fitting Toolbox, 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

- `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`[5]

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

[5] See appendices B.41, B.42, B.45, B.62, B.66 and B.80 for code.

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

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`[6]

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

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')`

[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`, `distanceStruc_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`[7]

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

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 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 `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[8] and used for Figure 3.2.

Dependencies None

[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 and B.79 for code.

[8] Available from `https://drive.google.com/open?id=13eUXp3xGph1B5T7dlIPvEVPZcHyACmom`.

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

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

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`[9]

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

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, μ and τ_0 can be given here as 2 doubles.

Outputs

- `soln_MCMC` - structure containing the means and standard deviations of the estimates for μ and τ_0 using MCMC method
- `soln_SP` - structure containing the means and standard deviations of the estimates for μ and τ_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
```
[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.

[9] 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`[10]

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

[10] See 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[1].

B.1 ABC_ml2.m

```
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,:);
```

[1] Available from https://drive.google.com/open?id=13eUXp3xGph1B5T7dlIPvEVPZcHyACmom.

```
23 DY = data(2,:);
24
25 idx = find(exp(-1)<DY & DY<0.5);
26 if idx(1) == 1
27     lb = 0;
28 else
29     lb = DX(idx(1)-1);
30 end
31 if idx(end) == length(DX)
32     f = fit(DY',DX','exp1');
33     est = (f.a)*exp((f.b)*exp(-1));
34     ub = 10^(ceil(log10(est)));
35 else
36     ub = DX(idx(end)+1);
37 end
38 rangeTau = [(1-tauWindow)*lb,(1+tauWindow)*ub];
39
40 pararange = [rangeMu;rangeTau];
41
42 %===PRIORS===%
43 priors.p1 = makedist('Uniform','lower',pararange(1,1),'
      upper',pararange(1,2));  %Prior for first parameter
44 priors.p2 = makedist('Uniform','lower',pararange(2,1),'
      upper',pararange(2,2));  %Prior for second parameter
45 %===PRIORS===%
46
47 MU=zeros(parchains,adjnloops);
48 TAU0=zeros(parchains,adjnloops);
49
50 % Arrays of parameters
51 parfor i=1:parchains
52     cMU = []; %scale parameter
53     cTAU0 = []; %shape parameter
54     while length(cMU)<adjnloops   % Random walk on
          parameters
55         thismu = random(priors.p1);
56         thistau0 = random(priors.p2);
57         % Generation of generalised beta distributed data
58         X=mlrnd(thismu,thistau0,1,nrand);
59         % Compute ecdf
60         [FF,XX] = ecdf(X);
61         try
62             YQ = interp1(XX(2:end),1-FF(2:end),DX);
63             % Distance between quantiles
64             dis = abs(DY-YQ);
65             dismax = max(dis);
66             if dismax < eps
67                 cMU = [cMU,thismu];
```

```
                cTAU0 = [cTAU0,thistau0];
            end
        catch
            %Do nothing - distance is too great!
        end
    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)^m1j*ml
           (-(data(i)/m2j)^m1j,m1j,m1j);
    end
    parfor i=1:length(X)
        dist2(i) = ((ml(-(X(i)/m2j)^m1j,m1j,0))-G(i))^2;
    end
    L2(j) = double(sqrt(sum(dist2)));
    prob = prod(indivterms);
    ll1(j) = double(log(prob));
end
[aic,bic] = aicbic(ll1,2,length(data));
end
```

B.4 analyse.m

```
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
```

```
% 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
```

```
55 data(:,2) = -data2;
56 data(:,3) = -data3;
57
58 %==Perform Analysis==%
59 [ActiveLinks_data,ActiveLinks_FitTool,ActiveLinks_MLE,
      ActiveLinks_Moments] = analyse_ActiveEdges(data,
      dir_ref);
60 [NodesActive_data,NodesActive_FitTool,NodesActive_MLE,
      NodesActive_Moments] = analyse_ActiveNodes(data,
      dir_ref);
61 [ActivityPotential_data,ActivityPotential_FitTool,
      ActivityPotential_MLE,ActivityPotential_Moments] =
      analyse_ActivityPotential(data,dir_ref);
62 [Clustering_data,Clustering_FitTool,Clustering_MLE,
      Clustering_Moments] = analyse_GlobalClusteringCoeff(
      data,dir_ref);
63 [InteractionTimes_data,InteractionTimes_FitTool,
      InteractionTimes_MLE,InteractionTimes_Moments] =
      analyse_InteractionTimes(data,dir_ref);
64 [Components_data,Components_FitTool,Components_MLE,
      Components_Moments] = analyse_NumberComponents(data,
      dir_ref);
65 [NoContactTimes_data,NoContactTimes_FitTool,
      NoContactTimes_MLE,NoContactTimes_Moments] =
      analyse_TimeBetweenContacts(data,dir_ref);
66 [ComponentNodes_data,ComponentNodes_FitTool,
      ComponentNodes_MLE,ComponentNodes_Moments] =
      analyse_ComponentNodes(data,dir_ref);
67 [ComponentEdges_data,ComponentEdges_FitTool,
      ComponentEdges_MLE,ComponentEdges_Moments] =
      analyse_ComponentEdges(data,dir_ref);
68
69 %==Post-Processing & Export==%
70 datafilename = [dir_ref,'/Distributions.mat'];
71 save(datafilename,...
72     'ActiveLinks_FitTool',...
73     'InteractionTimes_FitTool',...
74     'ActivityPotential_FitTool',...
75     'NoContactTimes_FitTool',...
76     'NodesActive_FitTool',...
77     'Components_FitTool',...
78     'Clustering_FitTool',...
79     'ComponentNodes_FitTool',...
80     'ComponentEdges_FitTool',...
81     'ActiveLinks_Moments',...
82     'InteractionTimes_Moments',...
83     'ActivityPotential_Moments',...
84     'NoContactTimes_Moments',...
```

```
85      'NodesActive_Moments',...
86      'Components_Moments',...
87      'Clustering_Moments',...
88      'ComponentNodes_Moments',...
89      'ComponentEdges_Moments',...
90      'ActiveLinks_MLE',...
91      'InteractionTimes_MLE',...
92      'ActivityPotential_MLE',...
93      'NoContactTimes_MLE',...
94      'NodesActive_MLE',...
95      'Components_MLE',...
96      'Clustering_MLE',...
97      'ComponentNodes_MLE',...
98      'ComponentEdges_MLE'...
99  )
100
101 Analysis = struct(  'ActiveLinks_FitTool',
        ActiveLinks_FitTool,...
102                     'ActiveLinks_Moments',
                          ActiveLinks_Moments,...
103                     'ActiveLinks_MLE',ActiveLinks_MLE,...
104                     'InteractionTimes_FitTool',
                          InteractionTimes_FitTool,...
105                     'InteractionTimes_Moments',
                          InteractionTimes_Moments,...
106                     'InteractionTimes_MLE',
                          InteractionTimes_MLE,...
107                     'ActivityPotential_FitTool',
                          ActivityPotential_FitTool,...
108                     'ActivityPotential_Moments',
                          ActivityPotential_Moments,...
109                     'ActivityPotential_MLE',
                          ActivityPotential_MLE,...
110                     'NoContactTimes_FitTool',
                          NoContactTimes_FitTool,...
111                     'NoContactTimes_Moments',
                          NoContactTimes_Moments,...
112                     'NoContactTimes_MLE',
                          NoContactTimes_MLE,...
113                     'NodesActive_FitTool',
                          NodesActive_FitTool,...
114                     'NodesActive_Moments',
                          NodesActive_Moments,...
115                     'NodesActive_MLE',NodesActive_MLE,...
116                     'Components_FitTool',
                          Components_FitTool,...
117                     'Components_Moments',
                          Components_Moments,...
```

```
118                         'Components_MLE',Components_MLE,...
119                         'Clustering_FitTool',
                              Clustering_FitTool,...
120                         'Clustering_Moments',
                              Clustering_Moments,...
121                         'Clustering_MLE',Clustering_MLE,...
122                         'ComponentNodes_FitTool',
                              ComponentNodes_FitTool,...
123                         'ComponentNodes_Moments',
                              ComponentNodes_Moments,...
124                         'ComponentNodes_MLE',
                              ComponentNodes_MLE,...
125                         'ComponentEdges_FitTool',
                              ComponentEdges_FitTool,...
126                         'ComponentEdges_Moments',
                              ComponentEdges_Moments,...
127                         'ComponentEdges_MLE',
                              ComponentEdges_MLE,...
128                         'NumberPeople',num_people);
129
130 data2global = struct('ActiveLinks_data',ActiveLinks_data,
      ...
131                         'InteractionTimes_data',
                              InteractionTimes_data,...
132                         'ActivityPotential_data',
                              ActivityPotential_data,...
133                         'NoContactTimes_data',
                              NoContactTimes_data,...
134                         'NodesActive_data',NodesActive_data,
                              ...
135                         'Components_data',Components_data,...
136                         'Clustering_data',Clustering_data,...
137                         'ComponentNodes_data',
                              ComponentNodes_data,...
138                         'ComponentEdges_data',
                              ComponentEdges_data,...
139                         'NumberPeople',num_people);
140
141 [mins,maxs] = dataMinMax(Analysis);
142
143 la2latex(ActiveLinks_FitTool,ActiveLinks_MLE,
      ActiveLinks_Moments,num_people,dir_ref,'Active Links'
      ,1);
144 la2latex(InteractionTimes_FitTool,InteractionTimes_MLE,
      InteractionTimes_Moments,num_people,dir_ref,'Int.
      Times',2);
145 la2latex(ActivityPotential_FitTool,ActivityPotential_MLE,
      ActivityPotential_Moments,num_people,dir_ref,'Activity
```

```
    Pot.',1);
146 la2latex(NoContactTimes_FitTool,NoContactTimes_MLE,
    NoContactTimes_Moments,num_people,dir_ref,'Time
    Between Contacts',1);
147 la2latex(NodesActive_FitTool,NodesActive_MLE,
    NodesActive_Moments,num_people,dir_ref,'Active Nodes'
    ,1);
148 la2latex(Components_FitTool,Components_MLE,
    Components_Moments,num_people,dir_ref,'Number of Comp.
    ',1);
149 la2latex(Clustering_FitTool,Clustering_MLE,
    Clustering_Moments,num_people,dir_ref,'GCC',1);
150 la2latex(ComponentNodes_FitTool,ComponentNodes_MLE,
    ComponentNodes_Moments,num_people,dir_ref,'Active
    Nodes per Comp.',1);
151 la2latex(ComponentEdges_FitTool,ComponentEdges_MLE,
    ComponentEdges_Moments,num_people,dir_ref,'Active
    Edges per Comp.',1);
152 end
```

B.5 analyse_ActiveEdges.m

```
1  function [data2global,FitTool,MLE,Moments] =
     analyse_ActiveEdges(data,dir_ref)
2  %ANALYSE_ACTIVEEDGES analyses the active edges
3  %
4  % DATA is the data matrix produced during ANALYSE
5  % DIR_REF is the save directory given as a text string
6  % DATA2GLOBAL is a vector of individual measurements of
     this metric
7  % FITTOOL is a structure containing parameters,
     statistics and pvalues for
8  % the fit tool
9  % MLE is a structure containing parameters, statistics
     and pvalues for
10 % the maximum likelihood estimators
11 % MOMENTS is a structure containing parameters,
     statistics and pvalues for
12 % the method of moments
13
14 Ymin = 1E-3;
15
16 cutExtreme = 3;
17
18 num_times = size(unique(data(:,1)),1);
19 data_length = size(data(:,1),1);
20 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(nodes,dir_ref,'ActiveNodes','Fraction of Nodes Active',cutExtreme,Ymin);
MLE = buildStruc_ExpGamRayLN_MLE(nodes,dir_ref,'ActiveNodes','Fraction of Nodes Active',cutExtreme,Ymin);
Moments = buildStruc_ExpGamRayLN_Moments(nodes,dir_ref,'ActiveNodes','Fraction of Nodes Active',cutExtreme,Ymin);

data2global = nodes;
end
```

B.7 analyse_ActivityPotential.m

```
function [data2global,FitTool,MLE,Moments] =
    analyse_ActivityPotential(data,dir_ref)
%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
```

```
        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
```

```
13
14 Ymin = 1E-1;
15
16 cutExtreme = 3;
17
18 num_times = size(unique(data(:,1)),1);
19 data_length = size(data(:,1),1);
20 num_people = max([data(:,2); data(:,3)]);
21 contact_time = 20;
22
23 rawFracEdges = zeros(num_times,num_people);
24
25 parfor m=1:num_times
26     thisadj = zeros(num_people);
27     current_time = (m-1)*contact_time;
28     for i=1:data_length
29         test_time = data(i,1);
30         if test_time==current_time
31             person1 = data(i,2);
32             person2 = data(i,3);
33             thisadj(person1,person2) = 1;
34             thisadj(person2,person1) = 1;
35         end
36     end
37     edgesActive = sum(sum(thisadj));
38     [~,~,thisCompGroups] = networkComponents(thisadj);
39     thisNumComps = length(thisCompGroups);
40     thisEdges = zeros(1,thisNumComps);
41     for j=1:thisNumComps
42         thisNodes = cell2mat(thisCompGroups(j));
43         thisNodesSize = length(thisNodes);
44         if thisNodesSize == 1
45             thisEdges(j)=0;
46         else
47             thisSubMat = thisadj(thisNodes,thisNodes);
48             thisAdjSum = sum(sum(thisSubMat));
49             thisNumEdges = thisAdjSum/2;
50             thisEdges(j) = thisNumEdges;
51         end
52     end
53     if edgesActive > 0
54         thisEdges = thisEdges/edgesActive;
55     end
56     thisPadding = num_people - length(thisEdges);
57     thisEdges = [thisEdges zeros(1,thisPadding)];
58     rawFracEdges(m,:) = thisEdges;
59 end
60
```

```
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;
end
```

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
```

```
3  %
4  % DATA is the data matrix produced during ANALYSE
5  % DIR_REF is the save directory given as a text string
6  % DATA2GLOBAL is a vector of individual measurements of
       this metric
7  % FITTOOL is a structure containing parameters,
       statistics and pvalues for
8  % the fit tool
9  % MLE is a structure containing parameters, statistics
       and pvalues for
10 % the maximum likelihood estimators
11 % MOMENTS is a structure containing parameters,
       statistics and pvalues for
12 % the method of moments
13
14 Ymin = 1E-2;
15
16 cutExtreme = 0;
17
18 num_times = size(unique(data(:,1)),1);
19 data_length = size(data(:,1),1);
20 num_people = max([data(:,2); data(:,3)]);
21 contact_time = 20;
22
23 clustering = zeros(1,num_times);
24
25 parfor m=1:num_times
26     thisadj = zeros(num_people);
27     current_time = (m-1)*contact_time;
28     for i=1:data_length
29         test_time = data(i,1);
30         if test_time==current_time
31             person1 = data(i,2);
32             person2 = data(i,3);
33             thisadj(person1,person2) = 1;
34             thisadj(person2,person1) = 1;
35         end
36     end
37     adj2 = thisadj^2;
38     adj3 = thisadj^3;
39     adj2sum = sum(sum(adj2));
40     contrip = adj2sum - trace(adj2);
41     if contrip==0
42         clustering(m) = 0;
43     else
44         clustering(m) = trace(adj3)/contrip;
45     end
46 end
```

```
47
48 clustering_autocorr = clustering;
49
50 clustering(clustering==0) = [];
51
52 FitTool = buildStruc_ExpGamRayLN_FitTool(clustering,
      dir_ref,'GlobalClusteringCoeff','Global Clustering
      Coefficient',cutExtreme,Ymin);
53 MLE = buildStruc_ExpGamRayLN_MLE(clustering,dir_ref,'
      GlobalClusteringCoeff','Global Clustering Coefficient'
      ,cutExtreme,Ymin);
54 Moments = buildStruc_ExpGamRayLN_Moments(clustering,
      dir_ref,'GlobalClusteringCoeff','Global Clustering
      Coefficient',cutExtreme,Ymin);
55
56 data2global = clustering;
57
58 maxTime = (num_times-1)*contact_time;
59 T = linspace(0,maxTime,num_times);
60
61 hwin = 50;
62 Tmod = T;
63 Tmod((num_times+1-hwin):num_times) = [];
64 Tmod(1:hwin) = [];
65
66 MA = zeros(1,num_times-2*hwin);
67 parfor i=1:(num_times-2*hwin)
68     upper = i+2*hwin;
69     MA(i) = sum(clustering_autocorr(i:upper))/(2*hwin+1);
70 end
71
72 clusteringfig = figure();
73 hold on
74 plot(T,clustering_autocorr)
75 plot(Tmod,MA,'LineWidth',4)
76 xlabel('Time (s)');
77 ylabel('Global Clustering Coefficient');
78 hold off
79 imagefilename = [dir_ref,'/
      GlobalClusteringCoeff_Additional_MovingAverage.png'];
80 print(imagefilename,'-dpng')
81 close(clusteringfig);
82
83 autocorrfig = figure();
84 subplot(3,1,1);
85 autocorr(clustering_autocorr,length(clustering_autocorr)
      -1);
86 subplot(3,1,2);
```

```
87 smallerlag = min(round(length(clustering_autocorr)-1,-2)
      /4,length(clustering_autocorr)-1);
88 autocorr(clustering_autocorr,smallerlag);
89 subplot(3,1,3);
90 evensmallerlag = min(round(length(clustering_autocorr)
      -1,-2)/16,length(clustering_autocorr)-1);
91 autocorr(clustering_autocorr,evensmallerlag);
92 imagefilename = [dir_ref,'/
      GlobalClusteringCoeff_Additional_AutoCorrelation.png'
      ];
93 print(imagefilename,'-dpng');
94 close(autocorrfig);
95
96 end
```

B.11 analyse_InteractionTimes.m

```
1  function [data2global,FitTool,MLE,Moments] =
      analyse_InteractionTimes(data,dir_ref)
2  %ANALYSE_INTERACTIONTIMES analyses the interaction times
3  %
4  % DATA is the data matrix produced during ANALYSE
5  % DIR_REF is the save directory given as a text string
6  % DATA2GLOBAL is a vector of individual measurements of
      this metric
7  % FITTOOL is a structure containing parameters,
      statistics and pvalues for
8  % the fit tool
9  % MLE is a structure containing parameters, statistics
      and pvalues for
10 % the maximum likelihood estimators
11 % MOMENTS is a structure containing parameters,
      statistics and pvalues for
12 % the method of moments
13
14 Ymin = 1E-3;
15
16 cutExtreme = 3;
17
18 number_rows = size(data,1);
19 contact_time = 20;
20
21 %==Sort Data by ID==%
22 [~, order] = sort(data(:,3));
23 partsorteddata = data(order,:);
24 [~, order] = sort(partsorteddata(:,2));
25 sorteddata = partsorteddata(order,:);
```

```
26
27 %==Find Interaction Times==%
28 times = zeros(1,number_rows);
29 j = 1;
30 times_k = 1;
31 step_vector = [contact_time 0 0];
32 while j<number_rows+1
33     contact = contact_time;
34     current_row = sorteddata(j,:);
35     if j == number_rows
36         next_row = [0 0 0];
37     else
38         next_row = sorteddata(j+1,:);
39     end
40     while isequal(next_row,current_row+step_vector)
41         contact = contact+contact_time;
42         j = j+1;
43         current_row = sorteddata(j,:);
44         if j == number_rows
45             next_row = [0 0 0];
46         else
47             next_row = sorteddata(j+1,:);
48         end
49     end
50     times(times_k) = contact;
51     j = j+1;
52     times_k = times_k+1;
53 end
54 times(times_k:end) = [];
55
56 FitTool = buildStruc_ExpMLGPWei_FitTool(times,dir_ref,'
       InteractionTimes','Length of Interaction',cutExtreme,
       Ymin);
57 MLE = buildStruc_ExpMLGPWei_MLE(times,dir_ref,'
       InteractionTimes','Length of Interaction',cutExtreme,
       Ymin);
58 Moments = buildStruc_ExpMLGPWei_Moments(times,dir_ref,'
       InteractionTimes','Length of Interaction',cutExtreme,
       Ymin);
59
60 data2global = times;
61 end
```

B.12 analyse_interevents.m

```
1 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,'yyyymmddTHHMMSS');
structure = '%f %f %f %*s %*s';


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
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
   )
```

```
92      gm_b_start = 0.1;
93  end
94  if gm_a_start<=0 || isnan(gm_a_start) || isinf(gm_a_start
        )
95      gm_a_start = 0.1;
96  end
97  rl_sigma_start = M1*sqrt(2/pi);
98  if rl_sigma_start<=0 || isnan(rl_sigma_start) || isinf(
        rl_sigma_start)
99      rl_sigma_start = 0.1;
100 end
101 ln_sigma_start = sqrt(log(M2*(M1^-2)));
102 if ln_sigma_start<=0 || isnan(ln_sigma_start) || isinf(
        ln_sigma_start)
103     ln_sigma_start = 0.1;
104 end
105 ln_mu_start = log(M1)-(0.5*ln_sigma_start^2);
106 if isnan(ln_mu_start) || isinf(ln_mu_start)
107     ln_mu_start = 0;
108 end
109 ml_beta_start = 0.5;
110 ml_gamma_start = 0.5;
111 [gp_k_start,gp_sigma_start,gp_theta_start] = gpSolve(M1,
        M2,M3);
112 if isnan(gp_k_start) || isinf(gp_k_start)
113     gp_k_start = 0;
114 end
115 if gp_sigma_start<=0 || isnan(gp_sigma_start) || isinf(
        gp_sigma_start)
116     gp_sigma_start = 0.1;
117 end
118 if isnan(gp_theta_start) || isinf(gp_theta_start)
119     gp_theta_start = 0;
120 end
121 wb_a_start = 0.5;
122 wb_b_start = 0.5;
123
124 try
125     fo_ex = fitoptions('Method', 'NonlinearLeastSquares',
            'Lower',[0],'Upper',[inf],'StartPoint',[
            ex_lambda_start]);
126     ft_ex = fittype('expcdf(x,lambda,''upper'')','options
            ',fo_ex);
127     [cf_ex,~] = fit(X,ccdf,ft_ex);
128     cv_ex = coeffvalues(cf_ex);
129 catch
130     cv_ex = [ex_lambda_start];
131 end
```

```
132
133 try
134     fo_gm = fitoptions('Method', 'NonlinearLeastSquares',
          'Lower',[0 0],'Upper',[inf inf],'StartPoint',[
          gm_a_start gm_b_start]);
135     ft_gm = fittype('gamcdf(x,a,b,''upper'')','options',
          fo_gm);
136     [cf_gm,~] = fit(X,ccdf,ft_gm);
137     cv_gm = coeffvalues(cf_gm);
138 catch
139     cv_gm = [gm_a_start gm_b_start];
140 end
141
142 try
143     fo_rl = fitoptions('Method', 'NonlinearLeastSquares',
          'Lower',[0],'Upper',[inf],'StartPoint',[
          rl_sigma_start]);
144     ft_rl = fittype('raylcdf(x,sigma,''upper'')','options
          ',fo_rl);
145     [cf_rl,~] = fit(X,ccdf,ft_rl);
146     cv_rl = coeffvalues(cf_rl);
147 catch
148     cv_rl = [rl_sigma_start];
149 end
150
151 try
152     fo_ln = fitoptions('Method', 'NonlinearLeastSquares',
          'Lower',[-inf 0],'Upper',[inf inf],'StartPoint',[
          ln_mu_start ln_sigma_start]);
153     ft_ln = fittype('logncdf(x,mu,sigma,''upper'')','
          options',fo_ln);
154     [cf_ln,~] = fit(X,ccdf,ft_ln);
155     cv_ln = coeffvalues(cf_ln);
156 catch
157     cv_ln = [0 1];
158 end
159
160 try
161     fo_ml = fitoptions('Method', 'NonlinearLeastSquares',
          'Lower',[0 0],'Upper',[1 1],'StartPoint',[
          ml_beta_start ml_gamma_start]);
162     ft_ml = fittype('mlf(beta,1,-gamma*x.^beta,6)','
          options',fo_ml);
163     [cf_ml,~] = fit(X,ccdf,ft_ml);
164     cv_ml = coeffvalues(cf_ml);
165 catch
166     cv_ml = [ml_beta_start ml_gamma_start];
167 end
```

```
168
169 try
170     fo_gp = fitoptions('Method', 'NonlinearLeastSquares',
           'Lower',[-inf -inf 0],'StartPoint',[gp_k_start
           gp_sigma_start gp_theta_start]);
171     ft_gp = fittype('gpcdf(x,k,sigma,theta,''upper'')','
           options',fo_gp);
172     [cf_gp,~] = fit(X,ccdf,ft_gp);
173     cv_gp = coeffvalues(cf_gp);
174 catch
175     cv_gp = [gp_k_start gp_sigma_start gp_theta_start];
176 end
177
178 try
179     fo_wb = fitoptions('Method', 'NonlinearLeastSquares',
           'Lower',[0 0],'StartPoint',[wb_a_start wb_b_start
           ]);
180     ft_wb = fittype('wblcdf(x,a,b,''upper'')','options',
           fo_wb);
181     [cf_wb,~] = fit(X,ccdf,ft_wb);
182     cv_wb = coeffvalues(cf_wb);
183 catch
184     cv_wb = [wb_a_start wb_b_start];
185 end
186
187 ex_lambda = cv_ex(1);
188 gm_a = cv_gm(1);
189 gm_b = cv_gm(2);
190 rl_sigma = cv_rl(1);
191 ln_mu = cv_ln(1);
192 ln_sigma = cv_ln(2);
193 ml_beta = cv_ml(1);
194 ml_gamma = cv_ml(2);
195 gp_k = cv_gp(1);
196 gp_sigma = cv_gp(2);
197 gp_theta = cv_gp(3);
198 wb_a = cv_wb(1);
199 wb_b = cv_wb(2);
200
201 sortedtimes = sort(timebetween);
202
203 z_ex = expcdf(sortedtimes,ex_lambda);
204 z_gm = gamcdf(sortedtimes,gm_a,gm_b);
205 z_rl = raylcdf(sortedtimes,rl_sigma);
206 z_ln = logncdf(sortedtimes,ln_mu,ln_sigma);
207 z_ml = ones(length(sortedtimes),1)-mlf(ml_beta,1,-
       ml_gamma*sortedtimes.^ml_beta,6);
208 z_gp = gpcdf(sortedtimes,gp_k,gp_sigma,gp_theta);
```

```
209 z_wb = wblcdf(sortedtimes,wb_a,wb_b);
210
211 zp_ex = exppdf(sortedtimes,ex_lambda);
212 zp_gm = gampdf(sortedtimes,gm_a,gm_b);
213 zp_rl = raylpdf(sortedtimes,rl_sigma);
214 zp_ln = lognpdf(sortedtimes,ln_mu,ln_sigma);
215 zp_ml = (-ml_beta./sortedtimes).*mlf(ml_beta,1,-ml_gamma*
    sortedtimes.^ml_beta,6);
216 zp_gp = gppdf(sortedtimes,gp_k,gp_sigma,gp_theta);
217 zp_wb = wblpdf(sortedtimes,wb_a,wb_b);
218
219 stats_ex = testStatistics(sortedtimes,z_ex,zp_ex,20);
220 stats_gm = testStatistics(sortedtimes,z_gm,zp_gm,20);
221 stats_rl = testStatistics(sortedtimes,z_rl,zp_rl,20);
222 stats_ln = testStatistics(sortedtimes,z_ln,zp_ln,20);
223 stats_ml = struct(); %Issues with input here
224 stats_gp = testStatistics(sortedtimes,z_gp,zp_gp,20);
225 stats_wb = testStatistics(sortedtimes,z_wb,zp_wb,20);
226
227 struc_ex = struct('Scale',ex_lambda);
228 struc_gm = struct('Shape',gm_a,'Scale',gm_b);
229 struc_rl = struct('Scale',rl_sigma);
230 struc_ln = struct('Location',ln_mu,'Scale',ln_sigma);
231 struc_ml = struct('Stability',ml_beta,'Scale',ml_gamma);
232 struc_gp = struct('Shape',gp_k,'Scale',gp_sigma,'Location
    ',gp_theta);
233 struc_wb = struct('Scale',wb_a,'Shape',wb_b);
234
235 EX = struct('Parameters',struc_ex,'Statistics',stats_ex);
236 GM = struct('Parameters',struc_gm,'Statistics',stats_gm);
237 RL = struct('Parameters',struc_rl,'Statistics',stats_rl);
238 LN = struct('Parameters',struc_ln,'Statistics',stats_ln);
239 ML = struct('Parameters',struc_ml,'Statistics',stats_ml);
240 GP = struct('Parameters',struc_gp,'Statistics',stats_gp);
241 WB = struct('Parameters',struc_wb,'Statistics',stats_wb);
242
243 dataout = struct('Exponential',EX,'Gamma',GM,'Rayleigh',
    RL,'LogNormal',LN,'MittagLeffler',ML,'GenPareto',GP,'
    Weibull',WB);
244
245
246 %==Plotting==%
247 ccdf_ex = expcdf(X,ex_lambda,'upper');
248 ccdf_gm = gamcdf(X,gm_a,gm_b,'upper');
249 ccdf_rl = raylcdf(X,rl_sigma,'upper');
250 ccdf_ln = logncdf(X,ln_mu,ln_sigma,'upper');
251 ccdf_ml = mlf(ml_beta,1,-ml_gamma*X.^ml_beta,6);
252 ccdf_gp = gpcdf(X,gp_k,gp_sigma,gp_theta,'upper');
```

```
253 ccdf_wb = wblcdf(X,wb_a,wb_b,'upper');
254
255 fig = figure();
256 hold on
257 plot(X,ccdf,'o');
258 plot(X,ccdf_ex);
259 plot(X,ccdf_gm);
260 plot(X,ccdf_rl);
261 plot(X,ccdf_ln);
262 plot(X,ccdf_ml);
263 plot(X,ccdf_gp);
264 plot(X,ccdf_wb);
265 set(gca,'XScale','log');
266 set(gca,'YScale','log');
267 xlabel('Time Between Activations');
268 ylabel('CCDF');
269 axis([-inf,inf,1E-4,1E0]);
270 legend('Data','Exponential','Gamma','Rayleigh','Log-
      Normal','Mittag Leffler','Gen. Pareto','Weibull','
      Location','northeast');
271 imagefilename = [dir_ref,'/between_activations.png'];
272 figfilename = [dir_ref,'/between_activations'];
273 print(imagefilename,'-dpng')
274 savefig(figfilename);
275 close(fig);
276
277 end
```

B.13 analyse_NumberComponents.m

```
1  function [data2global,FitTool,MLE,Moments] =
       analyse_NumberComponents(data,dir_ref)
2  %ANALYSE_NUMBERCOMPONENTS analyses the component count
3  %
4  % DATA is the data matrix produced during ANALYSE
5  % DIR_REF is the save directory given as a text string
6  % DATA2GLOBAL is a vector of individual measurements of
       this metric
7  % FITTOOL is a structure containing parameters,
       statistics and pvalues for
8  % the fit tool
9  % MLE is a structure containing parameters, statistics
       and pvalues for
10 % the maximum likelihood estimators
11 % MOMENTS is a structure containing parameters,
       statistics and pvalues for
12 % the method of moments
```

```
13
14 Ymin = 1E-4;
15
16 cutExtreme = 0;
17
18 num_times = size(unique(data(:,1)),1);
19 data_length = size(data(:,1),1);
20 num_people = max([data(:,2); data(:,3)]);
21 contact_time = 20;
22
23 components = zeros(1,num_times);
24
25 parfor m=1:num_times
26     thisadj = zeros(num_people);
27     current_time = (m-1)*contact_time;
28     for i=1:data_length
29         test_time = data(i,1);
30         if test_time==current_time
31             person1 = data(i,2);
32             person2 = data(i,3);
33             thisadj(person1,person2) = 1;
34             thisadj(person2,person1) = 1;
35         end
36     end
37 [~,thisComp,~] = networkComponents(thisadj);
38 thisComp(thisComp==1)=[];
39 thisCompCount = length(thisComp);
40 components(m) = thisCompCount;
41 end
42
43 FitTool = buildStruc_ExpGamRayLN_FitTool(components,
      dir_ref,'NumberComponents','Number of Components',
      cutExtreme,Ymin);
44 MLE = buildStruc_ExpGamRayLN_MLE(components,dir_ref,'
      NumberComponents','Number of Components',cutExtreme,
      Ymin);
45 Moments = buildStruc_ExpGamRayLN_Moments(components,
      dir_ref,'NumberComponents','Number of Components',
      cutExtreme,Ymin);
46
47 data2global = components;
48 end
```

B.14 analyse_TimeBetweenContacts.m

```
1 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);
```

```
44      activity(i,:) = thisactivity;
45  end
46
47  activity = [activity; ones(1,num_people)];
48
49  long = activity(:);
50  long = long';
51  dlong = diff([1 long 1]);
52  startIndex = find(dlong < 0);
53  endIndex = find(dlong > 0)-1;
54  nocontact = endIndex-startIndex+1;
55  nocontact = nocontact*20;
56
57  FitTool = buildStruc_ExpGamRayLN_FitTool(nocontact,
        dir_ref,'TimeBetweenContacts','Length of Time Between
        Contacts',cutExtreme,Ymin);
58  MLE = buildStruc_ExpGamRayLN_MLE(nocontact,dir_ref,'
        TimeBetweenContacts','Length of Time Between Contacts'
        ,cutExtreme,Ymin);
59  Moments = buildStruc_ExpGamRayLN_Moments(nocontact,
        dir_ref,'TimeBetweenContacts','Length of Time Between
        Contacts',cutExtreme,Ymin);
60
61  data2global = nocontact;
62  end
```

B.15 buildStruc_ExpGamRayLN_FitTool.m

```
1   function [Structure] = buildStruc_ExpGamRayLN_FitTool(
        data,dir_ref,property_title,graph_title,cutExtreme,
        Ymin)
2   %BUILDSTRUC_EXPGAMRAYLN_FITTOOL finds the best fit
        Exponential,
3   % Gamma, Rayleigh and Log-Normal distributions for the
        given data using the
4   % fit tool saving graphs, statistics and p-values
5   %
6   % DATA is the data matrix
7   % DIR_REF is the save directory
8   % PROPERTY_TITLE is the name of the property being
        analysed
9   % GRAPH_TITLE is the desired name for the graph
10  % CUTEXTREME is the number of extreme points to be
        removed
11  % YMIN is the lower limit on the Y-axis on the graph
12  % 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);

%==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);

```

```
55 fo_rl = fitoptions('Method', 'NonlinearLeastSquares','
       Lower',[0],'Upper',[inf],'StartPoint',[rl_sigma_start
       ]);
56 ft_rl = fittype('raylcdf(x,sigma,''upper'')','options',
       fo_rl);
57 [cf_rl,~] = fit(X,ccdf,ft_rl);
58 cv_rl = coeffvalues(cf_rl);
59
60 fo_ln = fitoptions('Method', 'NonlinearLeastSquares','
       Lower',[-inf 0],'Upper',[inf inf],'StartPoint',[
       ln_mu_start ln_sigma_start]);
61 ft_ln = fittype('logncdf(x,mu,sigma,''upper'')','options'
       ,fo_ln);
62 [cf_ln,~] = fit(X,ccdf,ft_ln);
63 cv_ln = coeffvalues(cf_ln);
64
65 %==Extract parameters==%
66 ex_lambda = cv_ex(1);
67 ccdf_ex = expcdf(X,ex_lambda,'upper');
68
69 gm_a = cv_gm(1);
70 gm_b = cv_gm(2);
71 ccdf_gm = gamcdf(X,gm_a,gm_b,'upper');
72
73 rl_sigma = cv_rl(1);
74 ccdf_rl = raylcdf(X,rl_sigma,'upper');
75
76 ln_mu = cv_ln(1);
77 ln_sigma = cv_ln(2);
78 ccdf_ln = logncdf(X,ln_mu,ln_sigma,'upper');
79
80 %==Extract GoF data==%
81 z_ex = expcdf(test_data,ex_lambda);
82 z_gm = gamcdf(test_data,gm_a,gm_b);
83 z_rl = raylcdf(test_data,rl_sigma);
84 z_ln = logncdf(test_data,ln_mu,ln_sigma);
85
86 zp_ex = exppdf(test_data,ex_lambda);
87 zp_gm = gampdf(test_data,gm_a,gm_b);
88 zp_rl = raylpdf(test_data,rl_sigma);
89 zp_ln = lognpdf(test_data,ln_mu,ln_sigma);
90
91 stats_ex = testStatistics(test_data,z_ex,zp_ex,0);
92 stats_gm = testStatistics(test_data,z_gm,zp_gm,0);
93 stats_rl = testStatistics(test_data,z_rl,zp_rl,0);
94 stats_ln = testStatistics(test_data,z_ln,zp_ln,0);
95
96 %==Plotting==%
```

```
97  fig = figure();
98  hold on
99  plot(X,ccdf,'o');
100 plot(X,ccdf_ex);
101 plot(X,ccdf_gm);
102 plot(X,ccdf_rl);
103 plot(X,ccdf_ln);
104 set(gca,'XScale','log');
105 set(gca,'YScale','log');
106 xlabel(graph_title);
107 ylabel('CCDF');
108 axis([-inf,inf,Ymin,1E0]);
109 legend('Data','Exponential','Gamma','Rayleigh','Log-
      Normal','Location','southwest');
110 imagefilename = [dir_ref,'/',property_title,'_FitTool.png
      '];
111 print(imagefilename,'-dpng')
112 figfilename = [dir_ref,'/',property_title,'_FitTool'];
113 savefig(figfilename);
114 close(fig);
115
116 %==Build data structure==%
117 samplesize = max(size(data));
118
119 struc_ex = struct('Scale',ex_lambda);
120 struc_gm = struct('Shape',gm_a,'Scale',gm_b);
121 struc_rl = struct('Scale',rl_sigma);
122 struc_ln = struct('Location',ln_mu,'Scale',ln_sigma);
123
124 p_ex = pvals_ex(samplesize,ex_lambda,stats_ex,cutExtreme,
      MC_Power,res);
125 p_gm = pvals_gm(samplesize,gm_a,gm_b,stats_gm,cutExtreme,
      MC_Power,res);
126 p_rl = pvals_rl(samplesize,rl_sigma,stats_rl,cutExtreme,
      MC_Power,res);
127 p_ln = pvals_ln(samplesize,ln_mu,ln_sigma,stats_ln,
      cutExtreme,MC_Power,res);
128
129 EX = struct('Parameters',struc_ex,'Statistics',stats_ex,'
      pValues',p_ex);
130 GM = struct('Parameters',struc_gm,'Statistics',stats_gm,'
      pValues',p_gm);
131 RL = struct('Parameters',struc_rl,'Statistics',stats_rl,'
      pValues',p_rl);
132 LN = struct('Parameters',struc_ln,'Statistics',stats_ln,'
      pValues',p_ln);
133
```

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

B.16 buildStruc_ExpGamRayLN_MLE.m

```
1  function [Structure] = buildStruc_ExpGamRayLN_MLE(data,
       dir_ref,property_title,graph_title,cutExtreme,Ymin)
2  %BUILDSTRUC_EXPGAMRAYLN_MLE finds the best fit
       Exponential,
3  % Gamma, Rayleigh and Log-Normal distributions for the
       given data using the
4  % most likelihood estimators saving graphs, statistics
       and p-values
5  %
6  % DATA is the data matrix
7  % DIR_REF is the save directory
8  % PROPERTY_TITLE is the name of the property being
       analysed
9  % GRAPH_TITLE is the desired name for the graph
10 % CUTEXTREME is the number of extreme points to be
       removed
11 % YMIN is the lower limit on the Y-axis on the graph
12 % STRUCTURE is a structure containing parameters,
       statistics and pvalues
13 % for this method
14
15 MC_Power = 6;
16
17 %==Prepare data==%
18 [F,X] = ecdf(data);
19 ccdf = 1-F;
20 if cutExtreme>0
21     Xrem = [X(end+1-cutExtreme:end)];
22 else
23     Xrem = [];
24 end
25 X = X(2:end-cutExtreme);
26 ccdf = ccdf(2:end-cutExtreme);
27 dataMod = data(~ismember(data,Xrem));
28 test_data = sort(dataMod)';
29 difference = diff(test_data);
30 difference = difference(difference>0);
31 res = min(difference);
32
33 %==Perform MLEs==%
34 mod_test_data = test_data;
```

```
35 mod_test_data(mod_test_data==0)=[];
36
37 phat_ex = mle(mod_test_data,'distribution','Exponential')
      ;
38 phat_gm = mle(mod_test_data,'distribution','Gamma');
39 phat_rl = mle(mod_test_data,'distribution','Rayleigh');
40 phat_ln = mle(mod_test_data,'distribution','Lognormal');
41
42 %==Extract parameters==%
43 ex_lambda = phat_ex(1);
44 ccdf_ex = expcdf(X,ex_lambda,'upper');
45
46 gm_a = phat_gm(1);
47 gm_b = phat_gm(2);
48 ccdf_gm = gamcdf(X,gm_a,gm_b,'upper');
49
50 rl_sigma = phat_rl(1);
51 ccdf_rl = raylcdf(X,rl_sigma,'upper');
52
53 ln_mu = phat_ln(1);
54 ln_sigma = phat_ln(2);
55 ccdf_ln = logncdf(X,ln_mu,ln_sigma,'upper');
56
57 %==Extract GoF data==%
58 z_ex = expcdf(test_data,ex_lambda);
59 z_gm = gamcdf(test_data,gm_a,gm_b);
60 z_rl = raylcdf(test_data,rl_sigma);
61 z_ln = logncdf(test_data,ln_mu,ln_sigma);
62
63 zp_ex = exppdf(test_data,ex_lambda);
64 zp_gm = gampdf(test_data,gm_a,gm_b);
65 zp_rl = raylpdf(test_data,rl_sigma);
66 zp_ln = lognpdf(test_data,ln_mu,ln_sigma);
67
68 stats_ex = testStatistics(test_data,z_ex,zp_ex,0);
69 stats_gm = testStatistics(test_data,z_gm,zp_gm,0);
70 stats_rl = testStatistics(test_data,z_rl,zp_rl,0);
71 stats_ln = testStatistics(test_data,z_ln,zp_ln,0);
72
73 %==Plotting==%
74 fig = figure();
75 hold on
76 plot(X,ccdf,'o');
77 plot(X,ccdf_ex);
78 plot(X,ccdf_gm);
79 plot(X,ccdf_rl);
80 plot(X,ccdf_ln);
81 set(gca,'XScale','log');
```

```
82  set(gca,'YScale','log');
83  xlabel(graph_title);
84  ylabel('CCDF');
85  axis([-inf,inf,Ymin,1E0]);
86  legend('Data','Exponential','Gamma','Rayleigh','Log-
        Normal','Location','southwest');
87  imagefilename = [dir_ref,'/',property_title,'_MLE.png'];
88  print(imagefilename,'-dpng')
89  figfilename = [dir_ref,'/',property_title,'_MLE'];
90  savefig(figfilename);
91  close(fig);
92
93  %==Build data structure==%
94  samplesize = max(size(data));
95
96  struc_ex = struct('Scale',ex_lambda);
97  struc_gm = struct('Shape',gm_a,'Scale',gm_b);
98  struc_rl = struct('Scale',rl_sigma);
99  struc_ln = struct('Location',ln_mu,'Scale',ln_sigma);
100
101 p_ex = pvals_ex(samplesize,ex_lambda,stats_ex,cutExtreme,
        MC_Power,res);
102 p_gm = pvals_gm(samplesize,gm_a,gm_b,stats_gm,cutExtreme,
        MC_Power,res);
103 p_rl = pvals_rl(samplesize,rl_sigma,stats_rl,cutExtreme,
        MC_Power,res);
104 p_ln = pvals_ln(samplesize,ln_mu,ln_sigma,stats_ln,
        cutExtreme,MC_Power,res);
105
106 EX = struct('Parameters',struc_ex,'Statistics',stats_ex,'
        pValues',p_ex);
107 GM = struct('Parameters',struc_gm,'Statistics',stats_gm,'
        pValues',p_gm);
108 RL = struct('Parameters',struc_rl,'Statistics',stats_rl,'
        pValues',p_rl);
109 LN = struct('Parameters',struc_ln,'Statistics',stats_ln,'
        pValues',p_ln);
110
111 Structure = struct('Exponential',EX,'Gamma',GM,'Rayleigh'
        ,RL,'LogNormal',LN);
112 end
```

B.17 buildStruc_ExpGamRayLN_Moments.m

```
1   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');
```

```
44
45 rl_sigma = M1*sqrt(2/pi);
46 ccdf_rl = raylcdf(X,rl_sigma,'upper');
47
48 ln_sigma = sqrt(log(M2*(M1^-2)));
49 ln_mu = log(M1)-(0.5*ln_sigma^2);
50 ccdf_ln = logncdf(X,ln_mu,ln_sigma,'upper');
51
52 %==Extract GoF data==%
53 z_ex = expcdf(test_data,ex_lambda);
54 z_gm = gamcdf(test_data,gm_a,gm_b);
55 z_rl = raylcdf(test_data,rl_sigma);
56 z_ln = logncdf(test_data,ln_mu,ln_sigma);
57
58 zp_ex = exppdf(test_data,ex_lambda);
59 zp_gm = gampdf(test_data,gm_a,gm_b);
60 zp_rl = raylpdf(test_data,rl_sigma);
61 zp_ln = lognpdf(test_data,ln_mu,ln_sigma);
62
63 stats_ex = testStatistics(test_data,z_ex,zp_ex,0);
64 stats_gm = testStatistics(test_data,z_gm,zp_gm,0);
65 stats_rl = testStatistics(test_data,z_rl,zp_rl,0);
66 stats_ln = testStatistics(test_data,z_ln,zp_ln,0);
67
68 %==Plotting==%
69 fig = figure();
70 hold on
71 plot(X,ccdf,'o');
72 plot(X,ccdf_ex);
73 plot(X,ccdf_gm);
74 plot(X,ccdf_rl);
75 plot(X,ccdf_ln);
76 set(gca,'XScale','log');
77 set(gca,'YScale','log');
78 xlabel(graph_title);
79 ylabel('CCDF');
80 axis([-inf,inf,Ymin,1E0]);
81 legend('Data','Exponential','Gamma','Rayleigh','Log-
     Normal','Location','southwest');
82 imagefilename = [dir_ref,'/',property_title,'_Moments.png
     '];
83 print(imagefilename,'-dpng')
84 figfilename = [dir_ref,'/',property_title,'_Moments'];
85 savefig(figfilename);
86 close(fig);
87
88 %==Build data structure==%
89 samplesize = max(size(data));
```

```
90
91  struc_ex = struct('Scale',ex_lambda);
92  struc_gm = struct('Shape',gm_a,'Scale',gm_b);
93  struc_rl = struct('Scale',rl_sigma);
94  struc_ln = struct('Location',ln_mu,'Scale',ln_sigma);
95
96  p_ex = pvals_ex(samplesize,ex_lambda,stats_ex,cutExtreme,
        MC_Power,res);
97  p_gm = pvals_gm(samplesize,gm_a,gm_b,stats_gm,cutExtreme,
        MC_Power,res);
98  p_rl = pvals_rl(samplesize,rl_sigma,stats_rl,cutExtreme,
        MC_Power,res);
99  p_ln = pvals_ln(samplesize,ln_mu,ln_sigma,stats_ln,
        cutExtreme,MC_Power,res);
100
101 EX = struct('Parameters',struc_ex,'Statistics',stats_ex,'
        pValues',p_ex);
102 GM = struct('Parameters',struc_gm,'Statistics',stats_gm,'
        pValues',p_gm);
103 RL = struct('Parameters',struc_rl,'Statistics',stats_rl,'
        pValues',p_rl);
104 LN = struct('Parameters',struc_ln,'Statistics',stats_ln,'
        pValues',p_ln);
105
106 Structure = struct('Exponential',EX,'Gamma',GM,'Rayleigh'
        ,RL,'LogNormal',LN);
107 end
```

B.18 buildStruc_ExpMLGPWei_FitTool.m

```
1  function [Structure] = buildStruc_ExpMLGPWei_FitTool(data
       ,dir_ref,property_title,graph_title,cutExtreme,Ymin)
2  %BUILDSTRUC_EXPMLGPWEI_FITTOOL finds the best fit
       Exponential,
3  % Mittag-Leffler, Generalised Pareto and Weibull
       distributions for the given
4  % data using the fit tool saving graphs, statistics and p
       -values
5  %
6  % DATA is the data matrix
7  % DIR_REF is the save directory
8  % PROPERTY_TITLE is the name of the property being
       analysed
9  % GRAPH_TITLE is the desired name for the graph
10 % CUTEXTREME is the number of extreme points to be
       removed
11 % 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',
       fo_ml);
54 [cf_ml,~] = fit(X,ccdf,ft_ml);
55 cv_ml = coeffvalues(cf_ml);
56
57 fo_gp = fitoptions('Method', 'NonlinearLeastSquares','
       Lower',[-inf -inf 0],'StartPoint',[gp_k_start
       gp_sigma_start gp_theta_start]);
58 ft_gp = fittype('gpcdf(x,k,sigma,theta,''upper'')','
       options',fo_gp);
59 [cf_gp,~] = fit(X,ccdf,ft_gp);
60 cv_gp = coeffvalues(cf_gp);
61
62 fo_wb = fitoptions('Method', 'NonlinearLeastSquares','
       Lower',[0 0],'StartPoint',[wb_a_start wb_b_start]);
63 ft_wb = fittype('wblcdf(x,a,b,''upper'')','options',fo_wb
       );
64 [cf_wb,~] = fit(X,ccdf,ft_wb);
65 cv_wb = coeffvalues(cf_wb);
66
67 %==Extract parameters==%
68 ex_lambda = cv_ex(1);
69 ccdf_ex = expcdf(X,ex_lambda,'upper');
70
71 ml_beta = cv_ml(1);
72 ml_gamma = cv_ml(2);
73 ccdf_ml = mlf(ml_beta,1,-ml_gamma*X.^ml_beta,6);
74
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
80 wb_a = cv_wb(1);
81 wb_b = cv_wb(2);
82 ccdf_wb = wblcdf(X,wb_a,wb_b,'upper');
83
84 %==Extract GoF data==%
85 z_ex = expcdf(test_data,ex_lambda);
86 z_ml = ones(length(test_data),1)-mlf(ml_beta,1,-ml_gamma*
       test_data.^ml_beta,6);
87 z_gp = gpcdf(test_data,gp_k,gp_sigma,gp_theta);
88 z_wb = wblcdf(test_data,wb_a,wb_b);
89
90 zp_ex = exppdf(test_data,ex_lambda);
91 zp_ml = (-ml_beta./test_data).*mlf(ml_beta,1,-ml_gamma*
       test_data.^ml_beta,6);
92 zp_gp = gppdf(test_data,gp_k,gp_sigma,gp_theta);
```

```
93  zp_wb = wblpdf(test_data,wb_a,wb_b);
94
95  stats_ex = testStatistics(test_data,z_ex,zp_ex,0);
96  stats_ml = testStatistics(test_data,z_ml,zp_ml,0);
97  stats_gp = testStatistics(test_data,z_gp,zp_gp,0);
98  stats_wb = testStatistics(test_data,z_wb,zp_wb,0);
99
100 %==Plotting==%
101 fig = figure();
102 hold on
103 plot(X,ccdf,'o');
104 plot(X,ccdf_ex);
105 plot(X,ccdf_ml);
106 plot(X,ccdf_gp);
107 plot(X,ccdf_wb);
108 set(gca,'XScale','log');
109 set(gca,'YScale','log');
110 xlabel(graph_title);
111 ylabel('CCDF');
112 axis([-inf,inf,Ymin,1E0]);
113 legend('Data','Exponential','Mittag Leffler','Gen. Pareto
    ','Weibull','Location','southwest');
114 imagefilename = [dir_ref,'/',property_title,'_FitTool.png
    '];
115 print(imagefilename,'-dpng')
116 figfilename = [dir_ref,'/',property_title,'_FitTool'];
117 savefig(figfilename);
118 close(fig);
119
120 %==Build data structure==%
121 samplesize = max(size(data));
122
123 struc_ex = struct('Scale',ex_lambda);
124 struc_ml = struct('Stability',ml_beta,'Scale',ml_gamma);
125 struc_gp = struct('Shape',gp_k,'Scale',gp_sigma,'Location
    ',gp_theta);
126 struc_wb = struct('Scale',wb_a,'Shape',wb_b);
127
128 p_ex = pvals_ex(samplesize,ex_lambda,stats_ex,cutExtreme,
    MC_Power,res);
129 p_ml = pvals_ml(samplesize,ml_beta,ml_gamma,stats_ml,
    cutExtreme,MC_Power_ML,res);
130 p_gp = pvals_gp(samplesize,gp_k,gp_sigma,gp_theta,
    stats_gp,cutExtreme,MC_Power,res);
131 p_wb = pvals_wb(samplesize,wb_a,wb_b,stats_wb,cutExtreme,
    MC_Power,res);
132
```

```
133 EX = struct('Parameters',struc_ex,'Statistics',stats_ex,'
    pValues',p_ex);
134 ML = struct('Parameters',struc_ml,'Statistics',stats_ml,'
    pValues',p_ml);
135 GP = struct('Parameters',struc_gp,'Statistics',stats_gp,'
    pValues',p_gp);
136 WB = struct('Parameters',struc_wb,'Statistics',stats_wb,'
    pValues',p_wb);
137
138 Structure = struct('Exponential',EX,'MittagLeffler',ML,'
    GenPareto',GP,'Weibull',WB);
139 end
```

B.19 buildStruc_ExpMLGPWei_MLE.m

```
1  function [Structure] = buildStruc_ExpMLGPWei_MLE(data,
       dir_ref,property_title,graph_title,cutExtreme,Ymin)
2  %BUILDSTRUC_EXPMLGPWEI_MLE finds the best fit Exponential
       ,
3  % Mittag-Leffler, Generalised Pareto and Weibull
       distributions for the given
4  % data using the most likelihoid estimators saving graphs
       , statistics and
5  % p-values
6  %
7  % DATA is the data matrix
8  % DIR_REF is the save directory
9  % PROPERTY_TITLE is the name of the property being
       analysed
10 % GRAPH_TITLE is the desired name for the graph
11 % CUTEXTREME is the number of extreme points to be
       removed
12 % YMIN is the lower limit on the Y-axis on the graph
13 % STRUCTURE is a structure containing parameters,
       statistics and pvalues
14 % for this method
15
16 MC_Power = 6;
17
18 %==Prepare data==%
19 [F,X] = ecdf(data);
20 ccdf = 1-F;
21 if cutExtreme>0
22     Xrem = [X(end+1-cutExtreme:end)];
23 else
24     Xrem = [];
25 end
```

```
26 X = X(2:end-cutExtreme);
27 ccdf = ccdf(2:end-cutExtreme);
28 dataMod = data(~ismember(data,Xrem));
29 test_data = sort(dataMod)';
30 difference = diff(test_data);
31 difference = difference(difference>0);
32 res = min(difference);
33
34 %==Perform MLEs==%
35 mod_test_data = test_data;
36 mod_test_data(mod_test_data==0)=[];
37
38 phat_ex = mle(mod_test_data,'distribution','Exponential')
      ;
39 phat_wb = mle(mod_test_data,'distribution','Weibull');
40
41 %==Extract parameters==%
42 ex_lambda = phat_ex(1);
43 ccdf_ex = expcdf(X,ex_lambda,'upper');
44
45 wb_a = phat_wb(1);
46 wb_b = phat_wb(2);
47 ccdf_wb = wblcdf(X,wb_a,wb_b,'upper');
48
49 %==Extract GoF data==%
50 z_ex = expcdf(test_data,ex_lambda);
51 z_wb = wblcdf(test_data,wb_a,wb_b);
52
53 zp_ex = exppdf(test_data,ex_lambda);
54 zp_wb = wblpdf(test_data,wb_a,wb_b);
55
56 stats_ex = testStatistics(test_data,z_ex,zp_ex,0);
57 stats_wb = testStatistics(test_data,z_wb,zp_wb,0);
58
59 %==Plotting==%
60 fig = figure();
61 hold on
62 plot(X,ccdf,'o');
63 plot(X,ccdf_ex);
64 plot(X,ccdf_wb);
65 set(gca,'XScale','log');
66 set(gca,'YScale','log');
67 xlabel(graph_title);
68 ylabel('CCDF');
69 axis([-inf,inf,Ymin,1E0]);
70 legend('Data','Exponential','Weibull','Location','
      southwest');
71 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_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);
end
```

B.20 buildStruc_ExpMLGPWei_Moments.m

```
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 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);
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');
```

```
63 xlabel(graph_title);
64 ylabel('CCDF');
65 axis([-inf,inf,Ymin,1E0]);
66 legend('Data','Exponential','Gen. Pareto','Location','
   southwest');
67 imagefilename = [dir_ref,'/',property_title,'_Moments.png
   '];
68 print(imagefilename,'-dpng')
69 figfilename = [dir_ref,'/',property_title,'_Moments'];
70 savefig(figfilename);
71 close(fig);
72
73 %==Build data structure==%
74 samplesize = max(size(data));
75
76 struc_ex = struct('Scale',ex_lambda);
77 struc_gp = struct('Shape',gp_k,'Scale',gp_sigma,'Location
   ',gp_theta);
78
79 p_ex = pvals_ex(samplesize,ex_lambda,stats_ex,cutExtreme,
   MC_Power,res);
80 p_gp = pvals_gp(samplesize,gp_k,gp_sigma,gp_theta,
   stats_gp,cutExtreme,MC_Power,res);
81
82 EX = struct('Parameters',struc_ex,'Statistics',stats_ex,'
   pValues',p_ex);
83 GP = struct('Parameters',struc_gp,'Statistics',stats_gp,'
   pValues',p_gp);
84
85 Structure = struct('Exponential',EX,'GenPareto',GP);
86 end
```

B.21 calculateDistance.m

```
1 function dist = calculateDistance(genFolder,realFolder)
2 %CALCULATEDISTANCE takes two folders of data and compares
     them to each
3 % other by calculating all the 2 sample KS-test distances
     between them
4 %
5 % GENFOLDER is the folder path of the first data
   collection, given as a
6 % string
7 % REALFOLDER is the folder path of the second data
   collection, given as a
8 % 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');
    %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;
```

```
43      GTriangles.(currentClean) = currentData.
            Triangles_data;
44  end
45
46  for i=1:length(RfileList)
47      currentFile = RfileList{i};
48      currentClean = strrep(currentFile, '.', '');
49      currentClean = strrep(currentClean, '-', '');
50      currentData = pullData(RiF,currentFile,'%f %f %f %*s
            %*s');
51      %Store Data
52      RActiveLinks.(currentClean) = currentData.
            ActiveLinks_data;
53      RInteractionTimes.(currentClean) = currentData.
            InteractionTimes_data;
54      RActivityPotential.(currentClean) = currentData.
            ActivityPotential_data;
55      RNoContactTimes.(currentClean) = currentData.
            NoContactTimes_data;
56      RNodesActive.(currentClean) = currentData.
            NodesActive_data;
57      RComponents.(currentClean) = currentData.
            Components_data;
58      RClustering.(currentClean) = currentData.
            Clustering_data;
59      RComponentNodes.(currentClean) = currentData.
            ComponentNodes_data;
60      RComponentEdges.(currentClean) = currentData.
            ComponentEdges_data;
61      RTriangles.(currentClean) = currentData.
            Triangles_data;
62  end
63
64  ActiveLinks_struc = distanceStruc_c(GActiveLinks,
        RActiveLinks);
65  OnTimes_struc = distanceStruc_c(GInteractionTimes,
        RInteractionTimes);
66  ActivityPot_struc = distanceStruc_c(GActivityPotential,
        RActivityPotential);
67  OffTimes_struc = distanceStruc_c(GNoContactTimes,
        RNoContactTimes);
68  ActiveNodes_struc = distanceStruc_c(GNodesActive,
        RNodesActive);
69  CompCount_struc = distanceStruc_c(GComponents,RComponents
        );
70  GCC_struc = distanceStruc_c(GClustering,RClustering);
71  CompNodes_struc = distanceStruc_c(GComponentNodes,
        RComponentNodes);
```

```
72 CompEdges_struc = distanceStruc_c(GComponentEdges,
      RComponentEdges);
73 TriangleCount_struc = distanceStruc_c(GTriangles,
      RTriangles);
74
75 dist.ActiveLinks = ActiveLinks_struc;
76 dist.OnTimes = OnTimes_struc;
77 dist.ActivityPot = ActivityPot_struc;
78 dist.OffTimes = OffTimes_struc;
79 dist.ActiveNodes = ActiveNodes_struc;
80 dist.CompCount = CompCount_struc;
81 dist.GCC = GCC_struc;
82 dist.CompNodes = CompNodes_struc;
83 dist.CompEdges = CompEdges_struc;
84 dist.TriangleCount = TriangleCount_struc;
```

B.22 chooselink.m

```
1  function [startnode,endnode] = chooselink(M)
2  %CHOOSELINK selects link at random using weighting matrix
       M
3  %
4  % M specifies the edge preference matrix. The can be
      extracted from the
5  % data using the function EAPmat.m or generated using MC
      simulation using
6  % the function rndLPM.m
7  % STARTNODE is the first endpoint of the chosen link
8  % ENDNODE is the second endpoint of the chosen link
9
10 nodes = size(M,1); %Extract number of nodes
11
12 M = M/sum(sum(M)); %Normalise M so that total sum is 1 (
      if not already)
13 vecM = reshape(M',1,numel(M)); %Reshape into a row vector
14 cumM = cumsum(vecM);
15
16 rn = rand(1); %Choose a random number from UNIF(0,1)
17
18 idx = find(cumM>=rn,1);
19 %Find point where cumsum of row vector first exceeds rn
20
21 modrem = mod(idx,nodes);
22 %Extracts nodes from this chosen entry
23 if modrem == 0
24     startnode = idx/nodes;
25     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;
```

```
58      RClustering.(currentClean) = currentData.
           Clustering_data;
59      RComponentNodes.(currentClean) = currentData.
           ComponentNodes_data;
60      RComponentEdges.(currentClean) = currentData.
           ComponentEdges_data;
61      RTriangles.(currentClean) = currentData.
           Triangles_data;
62  end
63
64  ActiveLinkStats = makeGraphs(GActiveLinks,RActiveLinks,'
       Active Links',dir_ref,0.0001);
65  OnTimesStats = makeGraphs(GInteractionTimes,
       RInteractionTimes,'Interaction Times',dir_ref,20);
66  ActivityPotStats = makeGraphs(GActivityPotential,
       RActivityPotential,'Activity Potential',dir_ref
       ,0.0001);
67  OffTimesStats = makeGraphs(GNoContactTimes,
       RNoContactTimes,'Time between Contacts',dir_ref,20);
68  ActiveNodesStats = makeGraphs(GNodesActive,RNodesActive,'
       Active Nodes',dir_ref,0.0001);
69  CompCountStats = makeGraphs(GComponents,RComponents,'
       Number of Components',dir_ref,1);
70  GCCStats = makeGraphs(GClustering,RClustering,'Clustering
        Coefficient',dir_ref,0.0001);
71  CompNodesStats = makeGraphs(GComponentNodes,
       RComponentNodes,'Nodes per Component',dir_ref,0.0001);
72  CompEdgesStats = makeGraphs(GComponentEdges,
       RComponentEdges,'Links per Component',dir_ref,0.0001);
73  TriangleCountStats = makeGraphs(GTriangles,RTriangles,'
       Number of Triangles',dir_ref,1);
```

B.24 compareOriginal.m

```
1  function [] = compareOriginal(folder)
2  %COMPAREORIGINAL produces a latex file containing
       Kolmogorov-Smirnov
3  % acceptances for the hypothesis that pairs of data
       samples in the given
4  % folder come from the same underlying distribution
       across chosen metrics
5  %
6  % FOLDER is the folder containing the files to compare,
       given as a string
7
8  timestamp = datestr(now,'yyyymmddTHHMMSS');
9  dir_ref = ['output_',timestamp];
```

```
10 mkdir(dir_ref);
11
12 filename = 'comparesamples.txt';
13 filepath = [dir_ref,'/',filename];
14
15 iF = ['input/',folder];
16 toExtract = [iF,'/*.csv'];
17
18 fileData = dir(toExtract);
19 fileList = {fileData.name};
20 fileList = fileList(~contains(fileList,'._'));
21
22 for i=1:length(fileList)
23     currentFile = fileList{i};
24     currentClean = strrep(currentFile, '.', '');
25     currentClean = strrep(currentClean, '-', '');
26     currentData = pullData(iF,currentFile,'%f %f %f %*s
          %*s');
27     %Store Data
28     fActiveLinks.(currentClean) = currentData.
          ActiveLinks_data;
29     fInteractionTimes.(currentClean) = currentData.
          InteractionTimes_data;
30     fActivityPotential.(currentClean) = currentData.
          ActivityPotential_data;
31     fNoContactTimes.(currentClean) = currentData.
          NoContactTimes_data;
32     fNodesActive.(currentClean) = currentData.
          NodesActive_data;
33     fComponents.(currentClean) = currentData.
          Components_data;
34     fClustering.(currentClean) = currentData.
          Clustering_data;
35     fComponentNodes.(currentClean) = currentData.
          ComponentNodes_data;
36     fComponentEdges.(currentClean) = currentData.
          ComponentEdges_data;
37     fTriangles.(currentClean) = currentData.
          Triangles_data;
38 end
39
40 fields = fieldnames(fActiveLinks);
41 numberComparisons = nchoosek(length(fields),2);
42 acceptmatrix1 = zeros(numberComparisons,10);
43 acceptmatrix5 = zeros(numberComparisons,10);
44 acceptmatrix10 = zeros(numberComparisons,10);
45
46 n = 0;
```

```
47 for i=1:length(fields)-1
48     for j=i+1:length(fields)
49         n = n+1;
50         f1 = fields{i};
51         f2 = fields{j};
52         [acceptmatrix1(n,1),acceptmatrix5(n,1),
              acceptmatrix10(n,1)] = acceptances(
              fActiveLinks.(f1),fActiveLinks.(f2));
53         [acceptmatrix1(n,2),acceptmatrix5(n,2),
              acceptmatrix10(n,2)] = acceptances(
              fInteractionTimes.(f1),fInteractionTimes.(f2))
              ;
54         [acceptmatrix1(n,3),acceptmatrix5(n,3),
              acceptmatrix10(n,3)] = acceptances(
              fActivityPotential.(f1),fActivityPotential.(f2
              ));
55         [acceptmatrix1(n,4),acceptmatrix5(n,4),
              acceptmatrix10(n,4)] = acceptances(
              fNoContactTimes.(f1),fNoContactTimes.(f2));
56         [acceptmatrix1(n,5),acceptmatrix5(n,5),
              acceptmatrix10(n,5)] = acceptances(
              fNodesActive.(f1),fNodesActive.(f2));
57         [acceptmatrix1(n,6),acceptmatrix5(n,6),
              acceptmatrix10(n,6)] = acceptances(fComponents
              .(f1),fComponents.(f2));
58         [acceptmatrix1(n,7),acceptmatrix5(n,7),
              acceptmatrix10(n,7)] = acceptances(fClustering
              .(f1),fClustering.(f2));
59         [acceptmatrix1(n,8),acceptmatrix5(n,8),
              acceptmatrix10(n,8)] = acceptances(
              fComponentNodes.(f1),fComponentNodes.(f2));
60         [acceptmatrix1(n,9),acceptmatrix5(n,9),
              acceptmatrix10(n,9)] = acceptances(
              fComponentEdges.(f1),fComponentEdges.(f2));
61         [acceptmatrix1(n,10),acceptmatrix5(n,10),
              acceptmatrix10(n,10)] = acceptances(fTriangles
              .(f1),fTriangles.(f2));
62     end
63 end
64
65 acceptsum1 = sum(acceptmatrix1);
66 acceptsum5 = sum(acceptmatrix5);
67 acceptsum10 = sum(acceptmatrix10);
68
69 ActiveLinks.accept1 = acceptsum1(1);
70 ActiveNodes.accept1 = acceptsum1(2);
71 ActivityPot.accept1 = acceptsum1(3);
72 GCC.accept1 = acceptsum1(4);
```

```
73  OnTimes.accept1 = acceptsum1(5);
74  OffTimes.accept1 = acceptsum1(6);
75  CompCount.accept1 = acceptsum1(7);
76  CompEdges.accept1 = acceptsum1(8);
77  CompNodes.accept1 = acceptsum1(9);
78  TriangleCount.accept1 = acceptsum1(10);
79
80  ActiveLinks.accept5 = acceptsum5(1);
81  ActiveNodes.accept5 = acceptsum5(2);
82  ActivityPot.accept5 = acceptsum5(3);
83  GCC.accept5 = acceptsum5(4);
84  OnTimes.accept5 = acceptsum5(5);
85  OffTimes.accept5 = acceptsum5(6);
86  CompCount.accept5 = acceptsum5(7);
87  CompEdges.accept5 = acceptsum5(8);
88  CompNodes.accept5 = acceptsum5(9);
89  TriangleCount.accept5 = acceptsum5(10);
90
91  ActiveLinks.accept10 = acceptsum10(1);
92  ActiveNodes.accept10 = acceptsum10(2);
93  ActivityPot.accept10 = acceptsum10(3);
94  GCC.accept10 = acceptsum10(4);
95  OnTimes.accept10 = acceptsum10(5);
96  OffTimes.accept10 = acceptsum10(6);
97  CompCount.accept10 = acceptsum10(7);
98  CompEdges.accept10 = acceptsum10(8);
99  CompNodes.accept10 = acceptsum10(9);
100 TriangleCount.accept10 = acceptsum10(10);
101
102 lines = 13;
103 tobuild = cell(lines,1);
104
105 tobuild{01} = '\begin{tabular}{l|c|c|c|} \cline{2-4}';
106 tobuild{02} = '& \textbf{1\%} & \textbf{5\%} & \textbf
       {10\%}\\ \hline';
107 tobuild{03} = ['\multicolumn{1}{|l|}{\textbf{Active Links
       }} & $',num2str(ActiveLinks.accept1),'$ & $',num2str(
       ActiveLinks.accept5),'$ & $',num2str(ActiveLinks.
       accept10),'$\\ \hline'];
108 tobuild{04} = ['\multicolumn{1}{|l|}{\textbf{Active Nodes
       }} & $',num2str(ActiveNodes.accept1),'$ & $',num2str(
       ActiveNodes.accept5),'$ & $',num2str(ActiveNodes.
       accept10),'$\\ \hline'];
109 tobuild{05} = ['\multicolumn{1}{|l|}{\textbf{Node
       Activity Potential}} & $',num2str(ActivityPot.accept1)
       ,'$ & $',num2str(ActivityPot.accept5),'$ & $',num2str(
       ActivityPot.accept10),'$\\ \hline'];
```

```
tobuild{06} = ['\multicolumn{1}{|l|}{\textbf{Global Clustering Coefficient}} & $',num2str(GCC.accept1),'$ & $',num2str(GCC.accept5),'$ & $',num2str(GCC.accept10),'$\\ \hline'];
tobuild{07} = ['\multicolumn{1}{|l|}{\textbf{Interaction Time}} & $',num2str(OnTimes.accept1),'$ & $',num2str(OnTimes.accept5),'$ & $',num2str(OnTimes.accept10),'$\\ \hline'];
tobuild{08} = ['\multicolumn{1}{|l|}{\textbf{Time Between Contacts}} & $',num2str(OffTimes.accept1),'$ & $',num2str(OffTimes.accept5),'$ & $',num2str(OffTimes.accept10),'$\\ \hline'];
tobuild{09} = ['\multicolumn{1}{|l|}{\textbf{Component Count}} & $',num2str(CompCount.accept1),'$ & $',num2str(CompCount.accept5),'$ & $',num2str(CompCount.accept10),'$\\ \hline'];
tobuild{10} = ['\multicolumn{1}{|l|}{\textbf{Links per Component}} & $',num2str(CompEdges.accept1),'$ & $',num2str(CompEdges.accept5),'$ & $',num2str(CompEdges.accept10),'$\\ \hline'];
tobuild{11} = ['\multicolumn{1}{|l|}{\textbf{Nodes per Component}} & $',num2str(CompNodes.accept1),'$ & $',num2str(CompNodes.accept5),'$ & $',num2str(CompNodes.accept10),'$\\ \hline'];
tobuild{12} = ['\multicolumn{1}{|l|}{\textbf{Triangle Count}} & $',num2str(TriangleCount.accept1),'$ & $',num2str(TriangleCount.accept5),'$ & $',num2str(TriangleCount.accept10),'$\\ \hline'];
tobuild{13} = '\end{tabular}';

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

B.25 comparison.m

```
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,'yyyymmddTHHMMSS');
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);
```

```
53          modelv2a_1(timelength,currentMatrix,fp2a);
54          modelv2_1(timelength,currentMatrix,fp2b);
55          modelv2_1(timelength,currentRandom,fp2c);
56      end
57      dist1 = calculateDistance(currentFolderM1(7:end),
           currentFolderR(7:end));
58      dist2a = calculateDistance(currentFolderM2a(7:end),
           currentFolderR(7:end));
59      dist2b = calculateDistance(currentFolderM2b(7:end),
           currentFolderR(7:end));
60      dist2c = calculateDistance(currentFolderM2c(7:end),
           currentFolderR(7:end));
61      currentAccept5 = zeros(10,4);
62      currentAccept5(1,1) = dist1.ActiveLinks.accept5;
63      currentAccept5(2,1) = dist1.OnTimes.accept5;
64      currentAccept5(3,1) = dist1.ActivityPot.accept5;
65      currentAccept5(4,1) = dist1.OffTimes.accept5;
66      currentAccept5(5,1) = dist1.ActiveNodes.accept5;
67      currentAccept5(6,1) = dist1.CompCount.accept5;
68      currentAccept5(7,1) = dist1.GCC.accept5;
69      currentAccept5(8,1) = dist1.CompNodes.accept5;
70      currentAccept5(9,1) = dist1.CompEdges.accept5;
71      currentAccept5(10,1) = dist1.TriangleCount.accept5;
72      currentAccept5(1,2) = dist2a.ActiveLinks.accept5;
73      currentAccept5(2,2) = dist2a.OnTimes.accept5;
74      currentAccept5(3,2) = dist2a.ActivityPot.accept5;
75      currentAccept5(4,2) = dist2a.OffTimes.accept5;
76      currentAccept5(5,2) = dist2a.ActiveNodes.accept5;
77      currentAccept5(6,2) = dist2a.CompCount.accept5;
78      currentAccept5(7,2) = dist2a.GCC.accept5;
79      currentAccept5(8,2) = dist2a.CompNodes.accept5;
80      currentAccept5(9,2) = dist2a.CompEdges.accept5;
81      currentAccept5(10,2) = dist2a.TriangleCount.accept5;
82      currentAccept5(1,3) = dist2b.ActiveLinks.accept5;
83      currentAccept5(2,3) = dist2b.OnTimes.accept5;
84      currentAccept5(3,3) = dist2b.ActivityPot.accept5;
85      currentAccept5(4,3) = dist2b.OffTimes.accept5;
86      currentAccept5(5,3) = dist2b.ActiveNodes.accept5;
87      currentAccept5(6,3) = dist2b.CompCount.accept5;
88      currentAccept5(7,3) = dist2b.GCC.accept5;
89      currentAccept5(8,3) = dist2b.CompNodes.accept5;
90      currentAccept5(9,3) = dist2b.CompEdges.accept5;
91      currentAccept5(10,3) = dist2b.TriangleCount.accept5;
92      currentAccept5(1,4) = dist2c.ActiveLinks.accept5;
93      currentAccept5(2,4) = dist2c.OnTimes.accept5;
94      currentAccept5(3,4) = dist2c.ActivityPot.accept5;
95      currentAccept5(4,4) = dist2c.OffTimes.accept5;
96      currentAccept5(5,4) = dist2c.ActiveNodes.accept5;
```

```
97      currentAccept5(6,4) = dist2c.CompCount.accept5;
98      currentAccept5(7,4) = dist2c.GCC.accept5;
99      currentAccept5(8,4) = dist2c.CompNodes.accept5;
100     currentAccept5(9,4) = dist2c.CompEdges.accept5;
101     currentAccept5(10,4) = dist2c.TriangleCount.accept5;
102     accept5matrix = accept5matrix+currentAccept5;
103 end
104
105 lines = 13;
106 tobuild = cell(lines,1);
107
108 tobuild{01} = '\begin{tabular}{l|c|c|c|c|} \cline{2-5}';
109 tobuild{02} = '& \textbf{Model 1} & \textbf{Model 2a} & \
       textbf{Model 2b} & \textbf{Model 2c}\\ \hline';
110 tobuild{03} = ['\multicolumn{1}{|l|}{\textbf{Active Links
       }} & $',num2str(accept5matrix(1,1)),'$ & $',num2str(
       accept5matrix(1,2)),'$ & $',num2str(accept5matrix(1,3)
       ),'$ & $',num2str(accept5matrix(1,4)),'$\\ \hline'];
111 tobuild{04} = ['\multicolumn{1}{|l|}{\textbf{Active Nodes
       }} & $',num2str(accept5matrix(2,1)),'$ & $',num2str(
       accept5matrix(2,2)),'$ & $',num2str(accept5matrix(2,3)
       ),'$ & $',num2str(accept5matrix(2,4)),'$\\ \hline'];
112 tobuild{05} = ['\multicolumn{1}{|l|}{\textbf{Node
       Activity Potential}} & $',num2str(accept5matrix(3,1)),
       '$ & $',num2str(accept5matrix(3,2)),'$ & $',num2str(
       accept5matrix(3,3)),'$ & $',num2str(accept5matrix(3,4)
       ),'$\\ \hline'];
113 tobuild{06} = ['\multicolumn{1}{|l|}{\textbf{Global
       Clustering Coefficient}} & $',num2str(accept5matrix
       (4,1)),'$ & $',num2str(accept5matrix(4,2)),'$ & $',
       num2str(accept5matrix(4,3)),'$ & $',num2str(
       accept5matrix(4,4)),'$\\ \hline'];
114 tobuild{07} = ['\multicolumn{1}{|l|}{\textbf{Interaction
       Time}} & $',num2str(accept5matrix(5,1)),'$ & $',
       num2str(accept5matrix(5,2)),'$ & $',num2str(
       accept5matrix(5,3)),'$ & $',num2str(accept5matrix(5,4)
       ),'$\\ \hline'];
115 tobuild{08} = ['\multicolumn{1}{|l|}{\textbf{Time Between
        Contacts}} & $',num2str(accept5matrix(6,1)),'$ & $',
       num2str(accept5matrix(6,2)),'$ & $',num2str(
       accept5matrix(6,3)),'$ & $',num2str(accept5matrix(6,4)
       ),'$\\ \hline'];
116 tobuild{09} = ['\multicolumn{1}{|l|}{\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);
```

```
theta1 = 2*pi/num_people1;

all_people2 = [gendata(:,2)' gendata(:,3)'];
num_times2 = size(unique(gendata(:,1)),1);
data_length2 = size(gendata(:,1),1);
num_people2 = max(all_people2);
coords2 = zeros(num_people2,2);
theta2 = 2*pi/num_people2;

num_times = min(num_times1,num_times2);

parfor n=1:num_people1
    coords1(n,:) = scale*[sin(n*theta1) cos(n*theta1)];
end

parfor n=1:num_people2
    coords2(n,:) = scale*[sin(n*theta2) cos(n*theta2)];
end

step_nf1 = zeros(num_people1,1);
line_freq1 = zeros(num_people1,num_people1);
step_nf2 = zeros(num_people2,1);
line_freq2 = zeros(num_people2,num_people2);

links(num_times) = struct('cdata',[],'colormap',[]);
map(num_times) = struct('cdata',[],'colormap',[]);

for m=1:num_times
    thisadj1 = zeros(num_people1);
    thisadj2 = zeros(num_people2);
    current_time = (m-1)*contact_time;
    for i=1:data_length1
        test_time = realdata(i,1);
        if test_time==current_time
            person1 = realdata(i,2);
            person2 = realdata(i,3);
            thisadj1(person1,person2) = 1;
            thisadj1(person2,person1) = 1;
        end
    end
    for i=1:data_length2
        test_time = gendata(i,1);
        if test_time==current_time
            person1 = gendata(i,2);
            person2 = gendata(i,3);
            thisadj2(person1,person2) = 1;
            thisadj2(person2,person1) = 1;
        end
```

```
    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);
```

```
            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
% all measured metrics
%
% 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);
[ActivityPotential_mins,ActivityPotential_maxs] = mm_ExpGamRayLN(Analysis.ActivityPotential_FitTool,Analysis.ActivityPotential_MLE,Analysis.ActivityPotential_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);
```

```
15 [Clustering_mins,Clustering_maxs] = mm_ExpGamRayLN(
      Analysis.Clustering_FitTool,Analysis.Clustering_MLE,
      Analysis.Clustering_Moments);
16 [ComponentNodes_mins,ComponentNodes_maxs] =
      mm_ExpGamRayLN(Analysis.ComponentNodes_FitTool,
      Analysis.ComponentNodes_MLE,Analysis.
      ComponentNodes_Moments);
17 [ComponentEdges_mins,ComponentEdges_maxs] =
      mm_ExpGamRayLN(Analysis.ComponentEdges_FitTool,
      Analysis.ComponentEdges_MLE,Analysis.
      ComponentEdges_Moments);
18
19
20 mins =          struct('ActiveLinks',ActiveLinks_mins,...
21                        'InteractionTimes',
                             InteractionTimes_mins,...
22                        'ActivityPotential',
                             ActivityPotential_mins,...
23                        'NoContactTimes',NoContactTimes_mins,
                             ...
24                        'NodesActive',NodesActive_mins,...
25                        'Components',Components_mins,...
26                        'Clustering',Clustering_mins,...
27                        'ComponentNodes',ComponentNodes_mins,
                             ...
28                        'ComponentEdges',ComponentEdges_mins
                             ...
29                       );
30
31 maxs =          struct('ActiveLinks',ActiveLinks_maxs,...
32                        'InteractionTimes',
                             InteractionTimes_maxs,...
33                        'ActivityPotential',
                             ActivityPotential_maxs,...
34                        'NoContactTimes',NoContactTimes_maxs,
                             ...
35                        'NodesActive',NodesActive_maxs,...
36                        'Components',Components_maxs,...
37                        'Clustering',Clustering_maxs,...
38                        'ComponentNodes',ComponentNodes_maxs,
                             ...
39                        'ComponentEdges',ComponentEdges_maxs
                             ...
40                       );
```

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
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(distMat));
distanceStruc.modeProb = mode(distMat(:));
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);
        thisChi = sum(thisSum);
        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;
num_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);
    close(thisframe);
end

v = VideoWriter(filename);
open(v)
writeVideo(v,frame)
```

```
close(v)

end
```

B.33 dualvideo.m

```
function [] = dualvideo(real_data,gen_data)
%DUALVIDEO Creates compariative 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);
lowestID = min(min(data_real(:,2)),min(data_real(:,3)));
data_real(:,2) = data_real(:,2)-lowestID+1;
data_real(:,3) = data_real(:,3)-lowestID+1;
number_rows = size(data_real,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
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);
```

```
dualDDVid(DinT_real,DinT_gen,dir_ref);
```

B.34 EAP_matrix.m

```
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);
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;

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
end
```

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');
```

```
111 end
```

B.36 emprand.m

```
1  function xr = emprand(dist,varargin)
2  %EMPRAND Generates random numbers from empirical
       distribution of data.
3  % This is useful when you do not know the distribution
       type (i.e. normal or
4  % uniform), but you have the data and you want to
       generate random
5  % numbers form the data. The idea is to first construct
       cumulative distribution
6  % function (cdf) from the given data. Then generate
       uniform random number and
7  % interpolate from cdf.
8  %
9  % USAGE:
10 %          xr = EMPRAND(dist)          - one random number
11 %          xr = EMPRAND(dist,m)        - m-by-m random
       numbers
12 %          xr = EMPRAND(dist,m,n)      - m-by-n random
       numbers
13 %
14 % INPUT:
15 %    dist - vector of distribution i.e. data values
16 %       m - generates m-by-m matrix of random numbers
17 %       n - generates m-by-n matrix of random numbers
18 %
19 % OUTPUT:
20 %    xr - generated random numbers
21 %
22 % EXAMPLES:
23 % % Generate 1000 normal random numbers
24 % mu = 0; sigma = 1; nr = 1000;
25 % givenDist = mu + sigma * randn(nr,1);
26 % generatedDist = emprand(givenDist,nr,1);
27 % %
28 % % % Plot histogram to check given and generated
       distribution
29 % [n,xout] = hist(givenDist);
30 % hist(givenDist);
31 % hold on
32 % hist(generatedDist,xout)
33 % %
34 % Plot cdf to check given and generated distribution
35 % figure
```

```
% x = sort(givenDist(:));      % Given distribution
% p = 1:length(x);
% p = p./length(x);
% plot(x,p,'color','r');
% hold on
%
% xr = sort(generatedDist(:)); % Generated distribution
% pr = 1:length(xr);
% pr = pr./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');
%
% HISTORY:
% version 1.0.0, Release 05-Jul-2005: Initial release
% version 1.1.0, Release 16-Oct-2007: Some bug fixes and
   improvement of help text
%    1. Can handle NaN values in dist
%    2. Extraplolate for out of range
%    3. Calling function EMPCDF is included within this
   function
%
% See also:

% Author: Durga Lal Shrestha
% UNESCO-IHE Institute for Water Education, Delft, The
   Netherlands
% eMail: durgals@hotmail.com
% Website: http://www.hi.ihe.nl/durgalal/index.htm
% Copyright 2004-2007 Durga Lal Shrestha.
% $First created: 05-Jul-2005
% $Revision: 1.1.0 $ $Date: 16-Oct-2007 21:47:47 $

% **********************
%% INPUT ARGUMENTS CHECK

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 extraplolate 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);
```

```
16
17 pdf = zeros(1,maxn+1);
18
19 lnpdf = lognpdf(X,mu,sigma);
20 lnpdf = lnpdf/sum(lnpdf);        %Normalise PDF
21 expdf = exppdf(X,lambda);
22 expdf = expdf/sum(expdf);        %Normalise PDF
23 copdf = conv(lnpdf,expdf);       %Create PDF for EX+LN
24
25 l = length(copdf);
26 CCDF = zeros(1,l);
27 parfor i=1:l
28     CCDF(i) = 1-sum(copdf(1:i));     %Create CCDF needed
29 end
30
31 thispdf = copdf;
32 for i=2:maxn+1
33     pdf(i) = sum(thispdf(1:tidx).*fliplr(CCDF(1:tidx)));
          %Convolved*CCDF
34     thispdf = conv(thispdf,copdf);
          %Next convolve
35 end
36 pdf(1)=1-sum(pdf);       %Calculate probability n=0, by
      inverse
37
38 num=zeros(1,N);
39 %Begin MC simulation
40 parfor k=1:N
41 dt=exprnd(lambda,1,M)+lognrnd(mu,sigma,1,M);
42 time=cumsum(dt);
43 index=find(time>t,1);
44 num(k)=min(index)-1;
45 end
46
47 freq = zeros(1,maxn+1);
48 parfor j=1:max(num)
49     freq(j)=length(find(num==j-1))/N;
50 end
51
52 %Plot on graph
53 mini = 1E-4;
54 n=0:maxn;
55 histimage = figure();
56 hold on
57 bar(n,pdf,'hist');
58 plot(n,freq,'x-');
59 xlabel('n');
60 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

end
```

B.39 gpSolve.m

```
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
```

```
5  %
6  % M1 is the first moment
7  % M2 is the second moment
8  % M3 is the third moment
9  % SHAPE is the calculated shape parameter using the given
       moments
10 % SCALE is the calculated scale parameter using the given
       moments
11 % LOCATION is the calculated location parameter using the
       given moments
12
13 syms xi sigma mu
14 eqn1 = mu + sigma/(1-xi) == M1;
15 eqn2 = mu^2 + (2*mu*sigma)/(1-xi) + (2*sigma^2)/((1-xi)
       *(1-2*xi)) == M2;
16 eqn3 = mu^3 + (3*mu^2*sigma)/(1-xi) + (6*mu*sigma^2)/((1-
       xi)*(1-2*xi)) + ...
17     (6*sigma^3)/((1-xi)*(1-2*xi)*(1-3*xi)) == M3;
18 eqn = [eqn1, eqn2, eqn3];
19 para = [xi, sigma, mu];
20 [xiSol,sigmaSol,muSol] = solve(eqn,para);
21
22 xiSolNum = vpa(xiSol);
23 sigmaSolNum = vpa(sigmaSol);
24 muSolNum = vpa(muSol);
25
26 xiSolNumR = real(xiSolNum);
27 sigmaSolNumR = real(sigmaSolNum);
28 muSolNumR = real(muSolNum);
29
30 correctSol = find(sigmaSolNumR>0,1);
31
32 xiTrue = xiSolNumR(correctSol);
33 sigmaTrue = sigmaSolNumR(correctSol);
34 muTrue = muSolNumR(correctSol);
35
36 shape = double(xiTrue);
37 scale = double(sigmaTrue);
38 location = double(muTrue);
39 end
```

B.40 individualgraphs.m

```
1  function [] = individualgraphs(folder,model,dir_ref)
2  %INDIVIDUALGRAPHS produces figures comparing all
       simulated samples to
3  % observed data by plotting eCCDFs in individual pairs
```

```
4  %
5  % FOLDER is the file path as a string
6  % MODEL is the model number to be compared
7  % DIR_REF is the save directory as a string
8
9  GiF = ['input/',folder,'/model',model];
10 GtoExtract = [GiF,'/*.csv'];
11
12 RiF = ['input/',folder,'/original'];
13 RtoExtract = [RiF,'/*.csv'];
14
15 mkdir(dir_ref);
16
17 GfileData = dir(GtoExtract);
18 GfileList = {GfileData.name};
19 GfileList = GfileList(~contains(GfileList,'._'));
20
21 RfileData = dir(RtoExtract);
22 RfileList = {RfileData.name};
23 RfileList = RfileList(~contains(RfileList,'._'));
24
25 for i=1:length(GfileList)
26     currentFile = GfileList{i};
27     currentClean = strrep(currentFile, '.', '');
28     currentClean = strrep(currentClean, '-', '');
29     currentData = pullData(GiF,currentFile,'%f %f %f %*s
          %*s');
30     %Store Data
31     GActiveLinks.(currentClean) = currentData.
          ActiveLinks_data;
32     GInteractionTimes.(currentClean) = currentData.
          InteractionTimes_data;
33     GActivityPotential.(currentClean) = currentData.
          ActivityPotential_data;
34     GNoContactTimes.(currentClean) = currentData.
          NoContactTimes_data;
35     GNodesActive.(currentClean) = currentData.
          NodesActive_data;
36     GComponents.(currentClean) = currentData.
          Components_data;
37     GClustering.(currentClean) = currentData.
          Clustering_data;
38     GComponentNodes.(currentClean) = currentData.
          ComponentNodes_data;
39     GComponentEdges.(currentClean) = currentData.
          ComponentEdges_data;
40     GTriangles.(currentClean) = currentData.
          Triangles_data;
```

```
41 end
42
43 for i=1:length(RfileList)
44     currentFile = RfileList{i};
45     currentClean = strrep(currentFile, '.', '');
46     currentClean = strrep(currentClean, '-', '');
47     currentData = pullData(RiF,currentFile,'%f %f %f %*s
          %*s');
48     %Store Data
49     RActiveLinks.(currentClean) = currentData.
          ActiveLinks_data;
50     RInteractionTimes.(currentClean) = currentData.
          InteractionTimes_data;
51     RActivityPotential.(currentClean) = currentData.
          ActivityPotential_data;
52     RNoContactTimes.(currentClean) = currentData.
          NoContactTimes_data;
53     RNodesActive.(currentClean) = currentData.
          NodesActive_data;
54     RComponents.(currentClean) = currentData.
          Components_data;
55     RClustering.(currentClean) = currentData.
          Clustering_data;
56     RComponentNodes.(currentClean) = currentData.
          ComponentNodes_data;
57     RComponentEdges.(currentClean) = currentData.
          ComponentEdges_data;
58     RTriangles.(currentClean) = currentData.
          Triangles_data;
59 end
60
61 modelID = strrep(model, '.', '');
62 modelID = strrep(modelID, '-', '');
63
64 for i=1:length(GfileList)
65     for j=1:length(RfileList)
66         currentGfile = GfileList{i};
67         currentG = strrep(currentGfile, '.', '');
68         currentG = strrep(currentG, '-', '');
69         currentRfile = RfileList{j};
70         currentR = strrep(currentRfile, '.', '');
71         currentR = strrep(currentR, '-', '');
72         current_dir = [currentR,'_against_model',modelID,
              '__',currentG];
73         save_dir = [dir_ref,'/',current_dir];
74         mkdir(save_dir);
75         currentG_AL = GActiveLinks.(currentG);
76         currentG_IT = GInteractionTimes.(currentG);
```

```
77          currentG_AP = GActivityPotential.(currentG);
78          currentG_NC = GNoContactTimes.(currentG);
79          currentG_AN = GNodesActive.(currentG);
80          currentG_CC = GComponents.(currentG);
81          currentG_GC = GClustering.(currentG);
82          currentG_CN = GComponentNodes.(currentG);
83          currentG_CL = GComponentEdges.(currentG);
84          currentG_TR = GTriangles.(currentG);
85          currentR_AL = RActiveLinks.(currentR);
86          currentR_IT = RInteractionTimes.(currentR);
87          currentR_AP = RActivityPotential.(currentR);
88          currentR_NC = RNoContactTimes.(currentR);
89          currentR_AN = RNodesActive.(currentR);
90          currentR_CC = RComponents.(currentR);
91          currentR_GC = RClustering.(currentR);
92          currentR_CN = RComponentNodes.(currentR);
93          currentR_CL = RComponentEdges.(currentR);
94          currentR_TR = RTriangles.(currentR);
95          pairgraphs(currentG_AL,currentR_AL,'Active Links'
               ,save_dir);
96          pairgraphs(currentG_IT,currentR_IT,'Interaction
               Time',save_dir);
97          pairgraphs(currentG_AP,currentR_AP,'Node Activity
                Potential',save_dir);
98          pairgraphs(currentG_NC,currentR_NC,'Time Between
               Contacts',save_dir);
99          pairgraphs(currentG_AN,currentR_AN,'Active Nodes'
               ,save_dir);
100         pairgraphs(currentG_CC,currentR_CC,'Component
               Count',save_dir);
101         pairgraphs(currentG_GC,currentR_GC,'Global
               Clustering Coefficient',save_dir);
102         pairgraphs(currentG_CN,currentR_CN,'Nodes per
               Component',save_dir);
103         pairgraphs(currentG_CL,currentR_CL,'Links per
               Component',save_dir);
104         pairgraphs(currentG_TR,currentR_TR,'Triangle
               Count',save_dir);
105     end
106 end
107 end
```

B.41 JSDiv.m

```
1 function dist=JSDiv(P,Q)
2 % 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

% 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,'/',filename];

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);
```

```
fNodesActive = cell(length(fileList),1);
fComponents = cell(length(fileList),1);
fClustering = cell(length(fileList),1);
fComponentNodes = cell(length(fileList),1);
fComponentEdges = cell(length(fileList),1);
fTriangles = cell(length(fileList),1);

for i=1:length(fileList)
    currentFile = fileList{i};
    currentData = pullData(iF,currentFile,'%f %f %f %*s %*s');
    %Store Data
    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);
```

```
[pCE,~] = probKS(fComponentEdges);
[pTC,~] = probKS(fTriangles);

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

tobuild{01} = '\begin{tabular}{l|c|c|} \cline{2-3}';
tobuild{02} = '& Statistic & $p$-Value \hline';
tobuild{03} = ['\multicolumn{1}{|l}{\textbf{Active Links}} & $',num2str(AL,4),'$ & $',num2str(1-pAL,4),'$\\ \hline'];
tobuild{04} = ['\multicolumn{1}{|l}{\textbf{Active Nodes}} & $',num2str(IT,4),'$ & $',num2str(1-pIT,4),'$\\ \hline'];
tobuild{05} = ['\multicolumn{1}{|l}{\textbf{Node Activity Potential}} & $',num2str(AP,4),'$ & $',num2str(1-pAP,4),'$\\ \hline'];
tobuild{06} = ['\multicolumn{1}{|l}{\textbf{Global Clustering Coefficient}} & $',num2str(NC,4),'$ & $',num2str(1-pNC,4),'$\\ \hline'];
tobuild{07} = ['\multicolumn{1}{|l}{\textbf{Interaction Time}} & $',num2str(NA,4),'$ & $',num2str(1-pNA,4),'$\\ \hline'];
tobuild{08} = ['\multicolumn{1}{|l}{\textbf{Time Between Contacts}} & $',num2str(CO,4),'$ & $',num2str(1-pCO,4),'$\\ \hline'];
tobuild{09} = ['\multicolumn{1}{|l}{\textbf{Component Count}} & $',num2str(CC,4),'$ & $',num2str(1-pCC,4),'$\\ \hline'];
tobuild{10} = ['\multicolumn{1}{|l}{\textbf{Links per Component}} & $',num2str(CN,4),'$ & $',num2str(1-pCN,4),'$\\ \hline'];
tobuild{11} = ['\multicolumn{1}{|l}{\textbf{Nodes per Component}} & $',num2str(CE,4),'$ & $',num2str(1-pCE,4),'$\\ \hline'];
tobuild{12} = ['\multicolumn{1}{|l}{\textbf{Triangle Count}} & $',num2str(TC,4),'$ & $',num2str(1-pTC,4),'$\\ \hline'];
tobuild{13} = '\end{tabular}';

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

B.44 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
       ||c|c|} \hline';
    tobuild{004} = ['\multicolumn{3}{|c||}{',name,'} & \
       multicolumn{2}{c||}{\textbf{Exponential}} & \
       multicolumn{2}{c||}{\textbf{Gamma}} & \multicolumn
       {2}{c||}{\textbf{Rayleigh}} & \multicolumn{2}{c
       |}{\textbf{Log-Normal}} \\ \hline'];
    tobuild{005} = ['\parbox[t]{2mm}{\multirow{16}{*}{\
       rotatebox[origin=c]{90}{Method of Moments}}} & \
       multicolumn{2}{|c||}{\multirow{2}{*}{\textbf{
       Parameters}}} & \multirow{2}{*}{Rate} & \multirow
       {2}{*}{$',num2matlabstr(Moments.Exponential.
       Parameters.Scale),'$} & Shape & $',num2matlabstr(
       Moments.Gamma.Parameters.Shape),'$ & \multirow
       {2}{*}{Scale} & \multirow{2}{*}{$',num2matlabstr(
       Moments.Rayleigh.Parameters.Scale),'$} & Loc. & $'
       ,num2matlabstr(Moments.LogNormal.Parameters.
       Location),'$ \\ \cline{6-7}\cline{10-11}'];
    tobuild{006} = ['& \multicolumn{2}{|c||}{} & & &
       Scale & $',num2matlabstr(Moments.Gamma.Parameters.
```

```
    Scale),'$ & & & Scale & $',num2matlabstr(Moments.
    LogNormal.Parameters.Scale),'$ \\ \cline{2-11}'];
30  tobuild{007} = ['& \parbox[t]{2mm}{\multirow{7}{*}{\
    rotatebox[origin=c]{90}{Statistics}}} & Kol D & \
    multicolumn{2}{c||}{$',num2matlabstr(Moments.
    Exponential.Statistics.Kolmogorov_D),'$} & \
    multicolumn{2}{c||}{$',num2matlabstr(Moments.Gamma
    .Statistics.Kolmogorov_D),'$} & \multicolumn{2}{c
    ||}{$',num2matlabstr(Moments.Rayleigh.Statistics.
    Kolmogorov_D),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(Moments.LogNormal.Statistics.
    Kolmogorov_D),'$} \\ \cline{3-11}'];
31  tobuild{008} = ['& & CvM & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Exponential.Statistics.
    Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Gamma.Statistics.
    Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Rayleigh.Statistics.
    Cramer_von_Mises),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(Moments.LogNormal.Statistics.
    Cramer_von_Mises),'$} \\ \cline{3-11}'];
32  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{3-11}'];
33  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{3-11}'];
34  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),'$} \\ \cline{3-11}'];
35  tobuild{012} = ['& & Kul-Lei & \multicolumn{2}{c||}{$
    ',num2matlabstr(Moments.Exponential.Statistics.
```

```
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Gamma.Statistics.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Rayleigh.Statistics.
    Kullback_Leibler),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(Moments.LogNormal.Statistics.
    Kullback_Leibler),'$} \\ \cline{3-11}'];
tobuild{013} = ['& & Jen-Sha & \multicolumn{2}{c||}{$
    ',num2matlabstr(Moments.Exponential.Statistics.
    Jensen_Shannon),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Gamma.Statistics.
    Jensen_Shannon),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Rayleigh.Statistics.
    Jensen_Shannon),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(Moments.LogNormal.Statistics.
    Jensen_Shannon),'$} \\ \cline{2-11}'];
tobuild{014} = ['& \parbox[t]{2mm}{\multirow{7}{*}{\
    rotatebox[origin=c]{90}{p-Values}}} & Kol D & \
    multicolumn{2}{c||}{$',num2matlabstr(Moments.
    Exponential.pValues.Kolmogorov_D),'$} & \
    multicolumn{2}{c||}{$',num2matlabstr(Moments.Gamma
    .pValues.Kolmogorov_D),'$} & \multicolumn{2}{c
    ||}{$',num2matlabstr(Moments.Rayleigh.pValues.
    Kolmogorov_D),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(Moments.LogNormal.pValues.
    Kolmogorov_D),'$} \\ \cline{3-11}'];
tobuild{015} = ['& & CvM & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Exponential.pValues.
    Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Gamma.pValues.
    Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Rayleigh.pValues.
    Cramer_von_Mises),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(Moments.LogNormal.pValues.
    Cramer_von_Mises),'$} \\ \cline{3-11}'];
tobuild{016} = ['& & Kuiper & \multicolumn{2}{c||}{$'
    ,num2matlabstr(Moments.Exponential.pValues.Kuiper)
    ,'$} & \multicolumn{2}{c||}{$',num2matlabstr(
    Moments.Gamma.pValues.Kuiper),'$} & \multicolumn
    {2}{c||}{$',num2matlabstr(Moments.Rayleigh.pValues
    .Kuiper),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(Moments.LogNormal.pValues.Kuiper),'
    $} \\ \cline{3-11}'];
tobuild{017} = ['& & Watson & \multicolumn{2}{c||}{$'
    ,num2matlabstr(Moments.Exponential.pValues.Watson)
    ,'$} & \multicolumn{2}{c||}{$',num2matlabstr(
    Moments.Gamma.pValues.Watson),'$} & \multicolumn
    {2}{c||}{$',num2matlabstr(Moments.Rayleigh.pValues
```

```
    .Watson),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(Moments.LogNormal.pValues.Watson),'
    $} \\ \cline{3-11}'];
41  tobuild{018} = ['& & And-Dar & \multicolumn{2}{c||}{$
    ',num2matlabstr(Moments.Exponential.pValues.
    Anderson_Darling),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Gamma.pValues.
    Anderson_Darling),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Rayleigh.pValues.
    Anderson_Darling),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(Moments.LogNormal.pValues.
    Anderson_Darling),'$} \\ \cline{3-11}'];
42  tobuild{019} = ['& & Kul-Lei & \multicolumn{2}{c||}{$
    ',num2matlabstr(Moments.Exponential.pValues.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Gamma.pValues.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Rayleigh.pValues.
    Kullback_Leibler),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(Moments.LogNormal.pValues.
    Kullback_Leibler),'$} \\ \cline{3-11}'];
43  tobuild{020} = ['& & Jen-Sha & \multicolumn{2}{c||}{$
    ',num2matlabstr(Moments.Exponential.pValues.
    Jensen_Shannon),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Gamma.pValues.Jensen_Shannon
    ),'$} & \multicolumn{2}{c||}{$',num2matlabstr(
    Moments.Rayleigh.pValues.Jensen_Shannon),'$} & \
    multicolumn{2}{c|}{$',num2matlabstr(Moments.
    LogNormal.pValues.Jensen_Shannon),'$} \\ \hline \
    hline'];
44  tobuild{021} = ['\parbox[t]{2mm}{\multirow{16}{*}{\
    rotatebox[origin=c]{90}{Curve Fitting Tool}}} & \
    multicolumn{2}{|c||}{\multirow{2}{*}{\textbf{
    Parameters}}} & \multirow{2}{*}{Rate} & \multirow
    {2}{*}{$',num2matlabstr(FitTool.Exponential.
    Parameters.Scale),'$} & Shape & $',num2matlabstr(
    FitTool.Gamma.Parameters.Shape),'$ & \multirow
    {2}{*}{Scale} & \multirow{2}{*}{$',num2matlabstr(
    FitTool.Rayleigh.Parameters.Scale),'$} & Loc. & $'
    ,num2matlabstr(FitTool.LogNormal.Parameters.
    Location),'$ \\ \cline{6-7}\cline{10-11}'];
45  tobuild{022} = ['& \multicolumn{2}{|c||}{} & & &
    Scale & $',num2matlabstr(FitTool.Gamma.Parameters.
    Scale),'$ & & & Scale & $',num2matlabstr(FitTool.
    LogNormal.Parameters.Scale),'$ \\ \cline{2-11}'];
46  tobuild{023} = ['& \parbox[t]{2mm}{\multirow{7}{*}{\
    rotatebox[origin=c]{90}{Statistics}}} & Kol D & \
    multicolumn{2}{c||}{$',num2matlabstr(FitTool.
```

```
    Exponential.Statistics.Kolmogorov_D),'$} & \
    multicolumn{2}{c||}{$',num2matlabstr(FitTool.Gamma
    .Statistics.Kolmogorov_D),'$} & \multicolumn{2}{c
    ||}{$',num2matlabstr(FitTool.Rayleigh.Statistics.
    Kolmogorov_D),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(FitTool.LogNormal.Statistics.
    Kolmogorov_D),'$} \\ \cline{3-11}'];
47  tobuild{024} = ['& & CvM & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Exponential.Statistics.
    Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Gamma.Statistics.
    Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Rayleigh.Statistics.
    Cramer_von_Mises),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(FitTool.LogNormal.Statistics.
    Cramer_von_Mises),'$} \\ \cline{3-11}'];
48  tobuild{025} = ['& & Kuiper & \multicolumn{2}{c||}{$'
    ,num2matlabstr(FitTool.Exponential.Statistics.
    Kuiper),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Gamma.Statistics.Kuiper),'$}
     & \multicolumn{2}{c||}{$',num2matlabstr(FitTool.
    Rayleigh.Statistics.Kuiper),'$} & \multicolumn{2}{
    c|}{$',num2matlabstr(FitTool.LogNormal.Statistics.
    Kuiper),'$} \\ \cline{3-11}'];
49  tobuild{026} = ['& & Watson & \multicolumn{2}{c||}{$'
    ,num2matlabstr(FitTool.Exponential.Statistics.
    Watson),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Gamma.Statistics.Watson),'$}
     & \multicolumn{2}{c||}{$',num2matlabstr(FitTool.
    Rayleigh.Statistics.Watson),'$} & \multicolumn{2}{
    c|}{$',num2matlabstr(FitTool.LogNormal.Statistics.
    Watson),'$} \\ \cline{3-11}'];
50  tobuild{027} = ['& & And-Dar & \multicolumn{2}{c||}{$
    ',num2matlabstr(FitTool.Exponential.Statistics.
    Anderson_Darling),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Gamma.Statistics.
    Anderson_Darling),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Rayleigh.Statistics.
    Anderson_Darling),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(FitTool.LogNormal.Statistics.
    Anderson_Darling),'$} \\ \cline{3-11}'];
51  tobuild{028} = ['& & Kul-Lei & \multicolumn{2}{c||}{$
    ',num2matlabstr(FitTool.Exponential.Statistics.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Gamma.Statistics.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Rayleigh.Statistics.
    Kullback_Leibler),'$} & \multicolumn{2}{c|}{$',
```

```
    num2matlabstr(FitTool.LogNormal.Statistics.
    Kullback_Leibler),'$} \\ \cline{3-11}'];
52  tobuild{029} = ['& & Jen-Sha & \multicolumn{2}{c||}{$
    ',num2matlabstr(FitTool.Exponential.Statistics.
    Jensen_Shannon),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Gamma.Statistics.
    Jensen_Shannon),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Rayleigh.Statistics.
    Jensen_Shannon),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(FitTool.LogNormal.Statistics.
    Jensen_Shannon),'$} \\ \cline{2-11}'];
53  tobuild{030} = ['& \parbox[t]{2mm}{\multirow{7}{*}{\
    rotatebox[origin=c]{90}{p-Values}}} & Kol D & \
    multicolumn{2}{c||}{$',num2matlabstr(FitTool.
    Exponential.pValues.Kolmogorov_D),'$} & \
    multicolumn{2}{c||}{$',num2matlabstr(FitTool.Gamma
    .pValues.Kolmogorov_D),'$} & \multicolumn{2}{c
    ||}{$',num2matlabstr(FitTool.Rayleigh.pValues.
    Kolmogorov_D),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(FitTool.LogNormal.pValues.
    Kolmogorov_D),'$} \\ \cline{3-11}'];
54  tobuild{031} = ['& & CvM & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Exponential.pValues.
    Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Gamma.pValues.
    Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Rayleigh.pValues.
    Cramer_von_Mises),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(FitTool.LogNormal.pValues.
    Cramer_von_Mises),'$} \\ \cline{3-11}'];
55  tobuild{032} = ['& & Kuiper & \multicolumn{2}{c||}{$'
    ,num2matlabstr(FitTool.Exponential.pValues.Kuiper)
    ,'$} & \multicolumn{2}{c||}{$',num2matlabstr(
    FitTool.Gamma.pValues.Kuiper),'$} & \multicolumn
    {2}{c||}{$',num2matlabstr(FitTool.Rayleigh.pValues
    .Kuiper),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(FitTool.LogNormal.pValues.Kuiper),'
    $} \\ \cline{3-11}'];
56  tobuild{033} = ['& & Watson & \multicolumn{2}{c||}{$'
    ,num2matlabstr(FitTool.Exponential.pValues.Watson)
    ,'$} & \multicolumn{2}{c||}{$',num2matlabstr(
    FitTool.Gamma.pValues.Watson),'$} & \multicolumn
    {2}{c||}{$',num2matlabstr(FitTool.Rayleigh.pValues
    .Watson),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(FitTool.LogNormal.pValues.Watson),'
    $} \\ \cline{3-11}'];
57  tobuild{034} = ['& & And-Dar & \multicolumn{2}{c||}{$
    ',num2matlabstr(FitTool.Exponential.pValues.
```

```
    Anderson_Darling),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Gamma.pValues.
    Anderson_Darling),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Rayleigh.pValues.
    Anderson_Darling),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(FitTool.LogNormal.pValues.
    Anderson_Darling),'$} \\ \cline{3-11}'];
tobuild{035} = ['& & Kul-Lei & \multicolumn{2}{c||}{$
    ',num2matlabstr(FitTool.Exponential.pValues.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Gamma.pValues.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Rayleigh.pValues.
    Kullback_Leibler),'$} & \multicolumn{2}{c|}{$',
    num2matlabstr(FitTool.LogNormal.pValues.
    Kullback_Leibler),'$} \\ \cline{3-11}'];
tobuild{036} = ['& & Jen-Sha & \multicolumn{2}{c||}{$
    ',num2matlabstr(FitTool.Exponential.pValues.
    Jensen_Shannon),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Gamma.pValues.Jensen_Shannon
    ),'$} & \multicolumn{2}{c||}{$',num2matlabstr(
    FitTool.Rayleigh.pValues.Jensen_Shannon),'$} & \
    multicolumn{2}{c|}{$',num2matlabstr(FitTool.
    LogNormal.pValues.Jensen_Shannon),'$} \\ \hline \
    hline'];
tobuild{037} = ['\parbox[t]{2mm}{\multirow{16}{*}{\
    rotatebox[origin=c]{90}{Maximum Likelihood
    Estimators}}} & \multicolumn{2}{|c||}{\multirow
    {2}{*}{\textbf{Parameters}}} & \multirow{2}{*}{
    Rate} & \multirow{2}{*}{$',num2matlabstr(MLE.
    Exponential.Parameters.Scale),'$} & Shape & $',
    num2matlabstr(MLE.Gamma.Parameters.Shape),'$ & \
    multirow{2}{*}{Scale} & \multirow{2}{*}{$',
    num2matlabstr(MLE.Rayleigh.Parameters.Scale),'$} &
     Loc. & $',num2matlabstr(MLE.LogNormal.Parameters.
    Location),'$ \\ \cline{6-7}\cline{10-11}'];
tobuild{038} = ['& \multicolumn{2}{|c||}{} & & &
    Scale & $',num2matlabstr(MLE.Gamma.Parameters.
    Scale),'$ & & & Scale & $',num2matlabstr(MLE.
    LogNormal.Parameters.Scale),'$ \\ \cline{2-11}'];
tobuild{039} = ['& \parbox[t]{2mm}{\multirow{7}{*}{\
    rotatebox[origin=c]{90}{Statistics}}} & Kol D & \
    multicolumn{2}{c||}{$',num2matlabstr(MLE.
    Exponential.Statistics.Kolmogorov_D),'$} & \
    multicolumn{2}{c||}{$',num2matlabstr(MLE.Gamma.
    Statistics.Kolmogorov_D),'$} & \multicolumn{2}{c
    ||}{$',num2matlabstr(MLE.Rayleigh.Statistics.
    Kolmogorov_D),'$} & \multicolumn{2}{c|}{$',
```

```
        num2matlabstr(MLE.LogNormal.Statistics.
        Kolmogorov_D),'$} \\ \cline{3-11}'];
63      tobuild{040} = ['& & CvM & \multicolumn{2}{c||}{$',
        num2matlabstr(MLE.Exponential.Statistics.
        Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$',
        num2matlabstr(MLE.Gamma.Statistics.
        Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$',
        num2matlabstr(MLE.Rayleigh.Statistics.
        Cramer_von_Mises),'$} & \multicolumn{2}{c|}{$',
        num2matlabstr(MLE.LogNormal.Statistics.
        Cramer_von_Mises),'$} \\ \cline{3-11}'];
64      tobuild{041} = ['& & Kuiper & \multicolumn{2}{c||}{$'
        ,num2matlabstr(MLE.Exponential.Statistics.Kuiper),
        '$} & \multicolumn{2}{c||}{$',num2matlabstr(MLE.
        Gamma.Statistics.Kuiper),'$} & \multicolumn{2}{c
        ||}{$',num2matlabstr(MLE.Rayleigh.Statistics.
        Kuiper),'$} & \multicolumn{2}{c|}{$',num2matlabstr
        (MLE.LogNormal.Statistics.Kuiper),'$} \\ \cline
        {3-11}'];
65      tobuild{042} = ['& & Watson & \multicolumn{2}{c||}{$'
        ,num2matlabstr(MLE.Exponential.Statistics.Watson),
        '$} & \multicolumn{2}{c||}{$',num2matlabstr(MLE.
        Gamma.Statistics.Watson),'$} & \multicolumn{2}{c
        ||}{$',num2matlabstr(MLE.Rayleigh.Statistics.
        Watson),'$} & \multicolumn{2}{c|}{$',num2matlabstr
        (MLE.LogNormal.Statistics.Watson),'$} \\ \cline
        {3-11}'];
66      tobuild{043} = ['& & And-Dar & \multicolumn{2}{c||}{$
        ',num2matlabstr(MLE.Exponential.Statistics.
        Anderson_Darling),'$} & \multicolumn{2}{c||}{$',
        num2matlabstr(MLE.Gamma.Statistics.
        Anderson_Darling),'$} & \multicolumn{2}{c||}{$',
        num2matlabstr(MLE.Rayleigh.Statistics.
        Anderson_Darling),'$} & \multicolumn{2}{c|}{$',
        num2matlabstr(MLE.LogNormal.Statistics.
        Anderson_Darling),'$} \\ \cline{3-11}'];
67      tobuild{044} = ['& & Kul-Lei & \multicolumn{2}{c||}{$
        ',num2matlabstr(MLE.Exponential.Statistics.
        Kullback_Leibler),'$} & \multicolumn{2}{c||}{$',
        num2matlabstr(MLE.Gamma.Statistics.
        Kullback_Leibler),'$} & \multicolumn{2}{c||}{$',
        num2matlabstr(MLE.Rayleigh.Statistics.
        Kullback_Leibler),'$} & \multicolumn{2}{c|}{$',
        num2matlabstr(MLE.LogNormal.Statistics.
        Kullback_Leibler),'$} \\ \cline{3-11}'];
68      tobuild{045} = ['& & Jen-Sha & \multicolumn{2}{c||}{$
        ',num2matlabstr(MLE.Exponential.Statistics.
        Jensen_Shannon),'$} & \multicolumn{2}{c||}{$',
```

```
    num2matlabstr(MLE.Gamma.Statistics.Jensen_Shannon)
    ,'$} & \multicolumn{2}{c||}{$',num2matlabstr(MLE.
    Rayleigh.Statistics.Jensen_Shannon),'$} & \
    multicolumn{2}{c|}{$',num2matlabstr(MLE.LogNormal.
    Statistics.Jensen_Shannon),'$} \\ \cline{2-11}'];
69  tobuild{046} = ['& \parbox[t]{2mm}{\multirow{7}{*}{\
    rotatebox[origin=c]{90}{p-Values}}} & Kol D & \
    multicolumn{2}{c||}{$',num2matlabstr(MLE.
    Exponential.pValues.Kolmogorov_D),'$} & \
    multicolumn{2}{c||}{$',num2matlabstr(MLE.Gamma.
    pValues.Kolmogorov_D),'$} & \multicolumn{2}{c||}{$
    ',num2matlabstr(MLE.Rayleigh.pValues.Kolmogorov_D)
    ,'$} & \multicolumn{2}{c|}{$',num2matlabstr(MLE.
    LogNormal.pValues.Kolmogorov_D),'$} \\ \cline
    {3-11}'];
70  tobuild{047} = ['& & CvM & \multicolumn{2}{c||}{$',
    num2matlabstr(MLE.Exponential.pValues.
    Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(MLE.Gamma.pValues.Cramer_von_Mises),
    '$} & \multicolumn{2}{c||}{$',num2matlabstr(MLE.
    Rayleigh.pValues.Cramer_von_Mises),'$} & \
    multicolumn{2}{c|}{$',num2matlabstr(MLE.LogNormal.
    pValues.Cramer_von_Mises),'$} \\ \cline{3-11}'];
71  tobuild{048} = ['& & Kuiper & \multicolumn{2}{c||}{$'
    ,num2matlabstr(MLE.Exponential.pValues.Kuiper),'$}
     & \multicolumn{2}{c||}{$',num2matlabstr(MLE.Gamma
    .pValues.Kuiper),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(MLE.Rayleigh.pValues.Kuiper),'$} & \
    multicolumn{2}{c|}{$',num2matlabstr(MLE.LogNormal.
    pValues.Kuiper),'$} \\ \cline{3-11}'];
72  tobuild{049} = ['& & Watson & \multicolumn{2}{c||}{$'
    ,num2matlabstr(MLE.Exponential.pValues.Watson),'$}
     & \multicolumn{2}{c||}{$',num2matlabstr(MLE.Gamma
    .pValues.Watson),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(MLE.Rayleigh.pValues.Watson),'$} & \
    multicolumn{2}{c|}{$',num2matlabstr(MLE.LogNormal.
    pValues.Watson),'$} \\ \cline{3-11}'];
73  tobuild{050} = ['& & And-Dar & \multicolumn{2}{c||}{$
    ',num2matlabstr(MLE.Exponential.pValues.
    Anderson_Darling),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(MLE.Gamma.pValues.Anderson_Darling),
    '$} & \multicolumn{2}{c||}{$',num2matlabstr(MLE.
    Rayleigh.pValues.Anderson_Darling),'$} & \
    multicolumn{2}{c|}{$',num2matlabstr(MLE.LogNormal.
    pValues.Anderson_Darling),'$} \\ \cline{3-11}'];
74  tobuild{051} = ['& & Kul-Lei & \multicolumn{2}{c||}{$
    ',num2matlabstr(MLE.Exponential.pValues.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$',
```

```
        num2matlabstr(MLE.Gamma.pValues.Kullback_Leibler),
        '$} & \multicolumn{2}{c||}{$',num2matlabstr(MLE.
        Rayleigh.pValues.Kullback_Leibler),'$} & \
        multicolumn{2}{c|}{$',num2matlabstr(MLE.LogNormal.
        pValues.Kullback_Leibler),'$} \\ \cline{3-11}'];
    tobuild{052} = ['& & Jen-Sha & \multicolumn{2}{c||}{$
        ',num2matlabstr(MLE.Exponential.pValues.
        Jensen_Shannon),'$} & \multicolumn{2}{c||}{$',
        num2matlabstr(MLE.Gamma.pValues.Jensen_Shannon),'
        $} & \multicolumn{2}{c||}{$',num2matlabstr(MLE.
        Rayleigh.pValues.Jensen_Shannon),'$} & \
        multicolumn{2}{c|}{$',num2matlabstr(MLE.LogNormal.
        pValues.Jensen_Shannon),'$} \\ \hline \hline'];
    tobuild{053} = '\end{tabular}';
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
        ||c|c|} \hline';
    tobuild{004} = ['\multicolumn{3}{|c||}{',name,'} & \
        multicolumn{2}{c||}{\textbf{Exponential}} & \
        multicolumn{2}{c||}{\textbf{Mittag-Leffler}} & \
        multicolumn{2}{c||}{\textbf{Gen. Pareto}} & \
        multicolumn{2}{c|}{\textbf{Weibull}} \\ \hline'];
    tobuild{005} = ['\parbox[t]{2mm}{\multirow{19}{*}{\
        rotatebox[origin=c]{90}{Method of Moments}}} & \
        multicolumn{2}{|c||}{\multirow{3}{*}{\textbf{
        Parameters}}} & \multirow{3}{*}{Rate} & \multirow
        {3}{*}{$',num2matlabstr(Moments.Exponential.
        Parameters.Scale),'$} & Stability & $X$ & Shape &
        $',num2matlabstr(Moments.GenPareto.Parameters.
        Shape),'$ & Scale & $X$ \\ \cline{6-11}'];
    tobuild{006} = ['& \multicolumn{2}{|c||}{} & & & \
        multirow{2}{*}{Scale} & \multirow{2}{*}{$X$} &
        Scale & $',num2matlabstr(Moments.GenPareto.
        Parameters.Scale),'$ & \multirow{2}{*}{Shape} & \
        multirow{2}{*}{$X$} \\ \cline{8-9}'];
    tobuild{007} = ['& \multicolumn{2}{|c||}{} & & & & &
        Loc. & $',num2matlabstr(Moments.GenPareto.
        Parameters.Location),'$ & & \\ \cline{2-11}'];
    tobuild{008} = ['& \parbox[t]{2mm}{\multirow{7}{*}{\
        rotatebox[origin=c]{90}{Statistics}}} & Kol D & \
        multicolumn{2}{c||}{$',num2matlabstr(Moments.
```

```
    Exponential.Statistics.Kolmogorov_D),'$} & \
    multicolumn{2}{c||}{$X$} & \multicolumn{2}{c||}{$'
    ,num2matlabstr(Moments.GenPareto.Statistics.
    Kolmogorov_D),'$} & \multicolumn{2}{c||}{$X$} \\ \
    cline{3-11}'];
tobuild{009} = ['& & CvM & \multicolumn{2}{c||}{$',
    num2matlabstr(Moments.Exponential.Statistics.
    Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$X$}
    & \multicolumn{2}{c||}{$',num2matlabstr(Moments.
    GenPareto.Statistics.Cramer_von_Mises),'$} & \
    multicolumn{2}{c||}{$X$} \\ \cline{3-11}'];
tobuild{010} = ['& & Kuiper & \multicolumn{2}{c||}{$'
    ,num2matlabstr(Moments.Exponential.Statistics.
    Kuiper),'$} & \multicolumn{2}{c||}{$X$} & \
    multicolumn{2}{c||}{$',num2matlabstr(Moments.
    GenPareto.Statistics.Kuiper),'$} & \multicolumn
    {2}{c||}{$X$} \\ \cline{3-11}'];
tobuild{011} = ['& & Watson & \multicolumn{2}{c||}{$'
    ,num2matlabstr(Moments.Exponential.Statistics.
    Watson),'$} & \multicolumn{2}{c||}{$X$} & \
    multicolumn{2}{c||}{$',num2matlabstr(Moments.
    GenPareto.Statistics.Watson),'$} & \multicolumn
    {2}{c||}{$X$} \\ \cline{3-11}'];
tobuild{012} = ['& & And-Dar & \multicolumn{2}{c||}{$
    ',num2matlabstr(Moments.Exponential.Statistics.
    Anderson_Darling),'$} & \multicolumn{2}{c||}{$X$}
    & \multicolumn{2}{c||}{$',num2matlabstr(Moments.
    GenPareto.Statistics.Anderson_Darling),'$} & \
    multicolumn{2}{c||}{$X$} \\ \cline{3-11}'];
tobuild{013} = ['& & Kul-Lei & \multicolumn{2}{c||}{$
    ',num2matlabstr(Moments.Exponential.Statistics.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$X$}
    & \multicolumn{2}{c||}{$',num2matlabstr(Moments.
    GenPareto.Statistics.Kullback_Leibler),'$} & \
    multicolumn{2}{c||}{$X$} \\ \cline{3-11}'];
tobuild{014} = ['& & Jen-Sha & \multicolumn{2}{c||}{$
    ',num2matlabstr(Moments.Exponential.Statistics.
    Jensen_Shannon),'$} & \multicolumn{2}{c||}{$X$} &
    \multicolumn{2}{c||}{$',num2matlabstr(Moments.
    GenPareto.Statistics.Jensen_Shannon),'$} & \
    multicolumn{2}{c||}{$X$} \\ \cline{2-11}'];
tobuild{015} = ['& \parbox[t]{2mm}{\multirow{7}{*}{\
    rotatebox[origin=c]{90}{p-Values}}} & Kol D & \
    multicolumn{2}{c||}{$',num2matlabstr(Moments.
    Exponential.pValues.Kolmogorov_D),'$} & \
    multicolumn{2}{c||}{$X$} & \multicolumn{2}{c||}{$'
    ,num2matlabstr(Moments.GenPareto.pValues.
    Kolmogorov_D),'$} & \multicolumn{2}{c||}{$X$} \\ \
```

```
    cline{3-11}'];
97  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||}{$X$} \\ \cline{3-11}'];
98  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||}{$X$} \\
     \cline{3-11}'];
99  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||}{$X$} \\
     \cline{3-11}'];
100 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),'$} & \
    multicolumn{2}{c||}{$X$} \\ \cline{3-11}'];
101 tobuild{020} = ['& & Kul-Lei & \multicolumn{2}{c||}{$
    ',num2matlabstr(Moments.Exponential.pValues.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$X$}
    & \multicolumn{2}{c||}{$',num2matlabstr(Moments.
    GenPareto.pValues.Kullback_Leibler),'$} & \
    multicolumn{2}{c||}{$X$} \\ \cline{3-11}'];
102 tobuild{021} = ['& & Jen-Sha & \multicolumn{2}{c||}{$
    ',num2matlabstr(Moments.Exponential.pValues.
    Jensen_Shannon),'$} & \multicolumn{2}{c||}{$X$} &
    \multicolumn{2}{c||}{$',num2matlabstr(Moments.
    GenPareto.pValues.Jensen_Shannon),'$} & \
    multicolumn{2}{c||}{$X$} \\ \hline \hline'];
103 tobuild{022} = ['\parbox[t]{2mm}{\multirow{19}{*}{\
    rotatebox[origin=c]{90}{Curve Fitting Tool}}} & \
    multicolumn{2}{|c||}{\multirow{3}{*}{\textbf{
    Parameters}}} & \multirow{3}{*}{Rate} & \multirow
    {3}{*}{$',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}'];
```

```
104     tobuild{023} = ['& \multicolumn{2}{|c||}{} & & & \
          multirow{2}{*}{Scale} & \multirow{2}{*}{$',
          num2matlabstr(FitTool.MittagLeffler.Parameters.
          Scale),'$} & Scale & $',num2matlabstr(FitTool.
          GenPareto.Parameters.Scale),'$ & \multirow{2}{*}{
          Shape} & \multirow{2}{*}{$',num2matlabstr(FitTool.
          Weibull.Parameters.Shape),'$} \\ \cline{8-9}'];
105     tobuild{024} = ['& \multicolumn{2}{|c||}{} & & & & &
          Loc. & $',num2matlabstr(FitTool.GenPareto.
          Parameters.Location),'$ & & \\ \cline{2-11}'];
106     tobuild{025} = ['& \parbox[t]{2mm}{\multirow{7}{*}{\
          rotatebox[origin=c]{90}{Statistics}}} & Kol D & \
          multicolumn{2}{c||}{$',num2matlabstr(FitTool.
          Exponential.Statistics.Kolmogorov_D),'$} & \
          multicolumn{2}{c||}{$',num2matlabstr(FitTool.
          MittagLeffler.Statistics.Kolmogorov_D),'$} & \
          multicolumn{2}{c||}{$',num2matlabstr(FitTool.
          GenPareto.Statistics.Kolmogorov_D),'$} & \
          multicolumn{2}{c||}{$',num2matlabstr(FitTool.
          Weibull.Statistics.Kolmogorov_D),'$} \\ \cline
          {3-11}'];
107     tobuild{026} = ['& & CvM & \multicolumn{2}{c||}{$',
          num2matlabstr(FitTool.Exponential.Statistics.
          Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$',
          num2matlabstr(FitTool.MittagLeffler.Statistics.
          Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$',
          num2matlabstr(FitTool.GenPareto.Statistics.
          Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$',
          num2matlabstr(FitTool.Weibull.Statistics.
          Cramer_von_Mises),'$} \\ \cline{3-11}'];
108     tobuild{027} = ['& & Kuiper & \multicolumn{2}{c||}{$'
          ,num2matlabstr(FitTool.Exponential.Statistics.
          Kuiper),'$} & \multicolumn{2}{c||}{$',
          num2matlabstr(FitTool.MittagLeffler.Statistics.
          Kuiper),'$} & \multicolumn{2}{c||}{$',
          num2matlabstr(FitTool.GenPareto.Statistics.Kuiper)
          ,'$} & \multicolumn{2}{c||}{$',num2matlabstr(
          FitTool.Weibull.Statistics.Kuiper),'$} \\ \cline
          {3-11}'];
109     tobuild{028} = ['& & Watson & \multicolumn{2}{c||}{$'
          ,num2matlabstr(FitTool.Exponential.Statistics.
          Watson),'$} & \multicolumn{2}{c||}{$',
          num2matlabstr(FitTool.MittagLeffler.Statistics.
          Watson),'$} & \multicolumn{2}{c||}{$',
          num2matlabstr(FitTool.GenPareto.Statistics.Watson)
          ,'$} & \multicolumn{2}{c||}{$',num2matlabstr(
          FitTool.Weibull.Statistics.Watson),'$} \\ \cline
          {3-11}'];
```

```
110     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),'$} \\ \cline{3-11}'];
111     tobuild{030} = ['& & Kul-Lei & \multicolumn{2}{c||}{$
           ',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),'$} \\ \cline{3-11}'];
112     tobuild{031} = ['& & Jen-Sha & \multicolumn{2}{c||}{$
           ',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),'$} \\ \cline{2-11}'];
113     tobuild{032} = ['& \parbox[t]{2mm}{\multirow{7}{*}{\
           rotatebox[origin=c]{90}{p-Values}}} & Kol D & \
           multicolumn{2}{c||}{$',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),'$} \\ \cline{3-11}'];
114     tobuild{033} = ['& & CvM & \multicolumn{2}{c||}{$',
           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),'$} \\ \cline{3-11}'];
115     tobuild{034} = ['& & Kuiper & \multicolumn{2}{c||}{$'
           ,num2matlabstr(FitTool.Exponential.pValues.Kuiper)
```

```
    ,'$} & \multicolumn{2}{c||}{$',num2matlabstr(
    FitTool.MittagLeffler.pValues.Kuiper),'$} & \
    multicolumn{2}{c||}{$',num2matlabstr(FitTool.
    GenPareto.pValues.Kuiper),'$} & \multicolumn{2}{c
    ||}{$',num2matlabstr(FitTool.Weibull.pValues.
    Kuiper),'$} \\ \cline{3-11}'];
116 tobuild{035} = ['& & Watson & \multicolumn{2}{c||}{$'
    ,num2matlabstr(FitTool.Exponential.pValues.Watson)
    ,'$} & \multicolumn{2}{c||}{$',num2matlabstr(
    FitTool.MittagLeffler.pValues.Watson),'$} & \
    multicolumn{2}{c||}{$',num2matlabstr(FitTool.
    GenPareto.pValues.Watson),'$} & \multicolumn{2}{c
    ||}{$',num2matlabstr(FitTool.Weibull.pValues.
    Watson),'$} \\ \cline{3-11}'];
117 tobuild{036} = ['& & And-Dar & \multicolumn{2}{c||}{$
    ',num2matlabstr(FitTool.Exponential.pValues.
    Anderson_Darling),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.MittagLeffler.pValues.
    Anderson_Darling),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.GenPareto.pValues.
    Anderson_Darling),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Weibull.pValues.
    Anderson_Darling),'$} \\ \cline{3-11}'];
118 tobuild{037} = ['& & Kul-Lei & \multicolumn{2}{c||}{$
    ',num2matlabstr(FitTool.Exponential.pValues.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.MittagLeffler.pValues.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.GenPareto.pValues.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Weibull.pValues.
    Kullback_Leibler),'$} \\ \cline{3-11}'];
119 tobuild{038} = ['& & Jen-Sha & \multicolumn{2}{c||}{$
    ',num2matlabstr(FitTool.Exponential.pValues.
    Jensen_Shannon),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.MittagLeffler.pValues.
    Jensen_Shannon),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.GenPareto.pValues.
    Jensen_Shannon),'$} & \multicolumn{2}{c||}{$',
    num2matlabstr(FitTool.Weibull.pValues.
    Jensen_Shannon),'$} \\ \hline \hline'];
120 tobuild{039} = ['\parbox[t]{2mm}{\multirow{19}{*}{\
    rotatebox[origin=c]{90}{Maximum Likelihood
    Estimators}}} & \multicolumn{2}{|c||}{\multirow
    {3}{*}{\textbf{Parameters}}} & \multirow{3}{*}{
    Rate} & \multirow{3}{*}{$',num2matlabstr(MLE.
    Exponential.Parameters.Scale),'$} & Stability & $X
    $ & Shape & $X$ & Scale & $',num2matlabstr(MLE.
```

```
    Weibull.Parameters.Scale),'$ \\ \cline{6-11}'];
tobuild{040} = ['& \multicolumn{2}{|c||}{} & & & \
    multirow{2}{*}{Scale} & \multirow{2}{*}{$X$} &
    Scale & $X$ & \multirow{2}{*}{Shape} & \multirow
    {2}{*}{$',num2matlabstr(MLE.Weibull.Parameters.
    Shape),'$} \\ \cline{8-9}'];
tobuild{041} = ['& \multicolumn{2}{|c||}{} & & & & &
    Loc. & $X$ & & \\ \cline{2-11}'];
tobuild{042} = ['& \parbox[t]{2mm}{\multirow{7}{*}{\
    rotatebox[origin=c]{90}{Statistics}}} & Kol D & \
    multicolumn{2}{c||}{$',num2matlabstr(MLE.
    Exponential.Statistics.Kolmogorov_D),'$} & \
    multicolumn{2}{c||}{$X$} & \multicolumn{2}{c||}{$X
    $} & \multicolumn{2}{c||}{$',num2matlabstr(MLE.
    Weibull.Statistics.Kolmogorov_D),'$} \\ \cline
    {3-11}'];
tobuild{043} = ['& & CvM & \multicolumn{2}{c||}{$',
    num2matlabstr(MLE.Exponential.Statistics.
    Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$X$}
    & \multicolumn{2}{c||}{$X$} & \multicolumn{2}{c
    ||}{$',num2matlabstr(MLE.Weibull.Statistics.
    Cramer_von_Mises),'$} \\ \cline{3-11}'];
tobuild{044} = ['& & Kuiper & \multicolumn{2}{c||}{$'
    ,num2matlabstr(MLE.Exponential.Statistics.Kuiper),
    '$} & \multicolumn{2}{c||}{$X$} & \multicolumn{2}{
    c||}{$X$} & \multicolumn{2}{c||}{$',num2matlabstr(
    MLE.Weibull.Statistics.Kuiper),'$} \\ \cline{3-11}
    '];
tobuild{045} = ['& & Watson & \multicolumn{2}{c||}{$'
    ,num2matlabstr(MLE.Exponential.Statistics.Watson),
    '$} & \multicolumn{2}{c||}{$X$} & \multicolumn{2}{
    c||}{$X$} & \multicolumn{2}{c||}{$',num2matlabstr(
    MLE.Weibull.Statistics.Watson),'$} \\ \cline{3-11}
    '];
tobuild{046} = ['& & And-Dar & \multicolumn{2}{c||}{$
    ',num2matlabstr(MLE.Exponential.Statistics.
    Anderson_Darling),'$} & \multicolumn{2}{c||}{$X$}
    & \multicolumn{2}{c||}{$X$} & \multicolumn{2}{c
    ||}{$',num2matlabstr(MLE.Weibull.Statistics.
    Anderson_Darling),'$} \\ \cline{3-11}'];
tobuild{047} = ['& & Kul-Lei & \multicolumn{2}{c||}{$
    ',num2matlabstr(MLE.Exponential.Statistics.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$X$}
    & \multicolumn{2}{c||}{$X$} & \multicolumn{2}{c
    ||}{$',num2matlabstr(MLE.Weibull.Statistics.
    Kullback_Leibler),'$} \\ \cline{3-11}'];
tobuild{048} = ['& & Jen-Sha & \multicolumn{2}{c||}{$
    ',num2matlabstr(MLE.Exponential.Statistics.
```

```
    Jensen_Shannon),'$} & \multicolumn{2}{c||}{$X$} &
    \multicolumn{2}{c||}{$X$} & \multicolumn{2}{c||}{$
    ',num2matlabstr(MLE.Weibull.Statistics.
    Jensen_Shannon),'$} \\ \cline{2-11}'];
130 tobuild{049} = ['& \parbox[t]{2mm}{\multirow{7}{*}{\
    rotatebox[origin=c]{90}{p-Values}}} & Kol D & \
    multicolumn{2}{c||}{$',num2matlabstr(MLE.
    Exponential.pValues.Kolmogorov_D),'$} & \
    multicolumn{2}{c||}{$X$} & \multicolumn{2}{c||}{$X
    $} & \multicolumn{2}{c||}{$',num2matlabstr(MLE.
    Weibull.pValues.Kolmogorov_D),'$} \\ \cline{3-11}'
    ];
131 tobuild{050} = ['& & CvM & \multicolumn{2}{c||}{$',
    num2matlabstr(MLE.Exponential.pValues.
    Cramer_von_Mises),'$} & \multicolumn{2}{c||}{$X$}
    & \multicolumn{2}{c||}{$X$} & \multicolumn{2}{c
    ||}{$',num2matlabstr(MLE.Weibull.pValues.
    Cramer_von_Mises),'$} \\ \cline{3-11}'];
132 tobuild{051} = ['& & Kuiper & \multicolumn{2}{c||}{$'
    ,num2matlabstr(MLE.Exponential.pValues.Kuiper),'$}
     & \multicolumn{2}{c||}{$X$} & \multicolumn{2}{c
    ||}{$X$} & \multicolumn{2}{c||}{$',num2matlabstr(
    MLE.Weibull.pValues.Kuiper),'$} \\ \cline{3-11}'];
133 tobuild{052} = ['& & Watson & \multicolumn{2}{c||}{$'
    ,num2matlabstr(MLE.Exponential.pValues.Watson),'$}
     & \multicolumn{2}{c||}{$X$} & \multicolumn{2}{c
    ||}{$X$} & \multicolumn{2}{c||}{$',num2matlabstr(
    MLE.Weibull.pValues.Watson),'$} \\ \cline{3-11}'];
134 tobuild{053} = ['& & And-Dar & \multicolumn{2}{c||}{$
    ',num2matlabstr(MLE.Exponential.pValues.
    Anderson_Darling),'$} & \multicolumn{2}{c||}{$X$}
    & \multicolumn{2}{c||}{$X$} & \multicolumn{2}{c
    ||}{$',num2matlabstr(MLE.Weibull.pValues.
    Anderson_Darling),'$} \\ \cline{3-11}'];
135 tobuild{054} = ['& & Kul-Lei & \multicolumn{2}{c||}{$
    ',num2matlabstr(MLE.Exponential.pValues.
    Kullback_Leibler),'$} & \multicolumn{2}{c||}{$X$}
    & \multicolumn{2}{c||}{$X$} & \multicolumn{2}{c
    ||}{$',num2matlabstr(MLE.Weibull.pValues.
    Kullback_Leibler),'$} \\ \cline{3-11}'];
136 tobuild{055} = ['& & Jen-Sha & \multicolumn{2}{c||}{$
    ',num2matlabstr(MLE.Exponential.pValues.
    Jensen_Shannon),'$} & \multicolumn{2}{c||}{$X$} &
    \multicolumn{2}{c||}{$X$} & \multicolumn{2}{c||}{$
    ',num2matlabstr(MLE.Weibull.pValues.Jensen_Shannon
    ),'$} \\ \hline'];
137 tobuild{056} = '\end{tabular}';
138
```

```
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

cleanName = strrep(variable, '.', '');
cleanName = strrep(cleanName, '-', '');

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);
end
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
```

B.46 manymodels.m

```
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;
```

```
138 datadump.TriangleCount = TriangleCount;
139
140 averages.ActiveLinks = mean(ActiveLinks,2);
141 averages.OnTimes = mean(OnTimes,2);
142 averages.ActivityPot = mean(ActivityPot,2);
143 averages.OffTimes = mean(OffTimes,2);
144 averages.ActiveNodes = mean(ActiveNodes,2);
145 averages.CompCount = mean(CompCount,2);
146 averages.GCC = mean(GCC,2);
147 averages.CompNodes = mean(CompNodes,2);
148 averages.CompEdges = mean(CompEdges,2);
149 averages.TriangleCount = mean(TriangleCount,2);
150
151 lines = 13;
152 tobuild = cell(lines,1);
153
154 tobuild{01} = '\begin{tabular}{l|c|c|c|c|} \cline{2-5}';
155 tobuild{02} = '& \textbf{Model 1} & \textbf{Model 2a} & \
    textbf{Model 2b} & \textbf{Model 2c}\\ \hline';
156 tobuild{03} = ['\multicolumn{1}{|l|}{\textbf{Active Links
    }} & $',num2str(averages.ActiveLinks(1)),'$ & $',
    num2str(averages.ActiveLinks(2)),'$ & $',num2str(
    averages.ActiveLinks(3)),'$ & $',num2str(averages.
    ActiveLinks(4)),'$\\ \hline'];
157 tobuild{04} = ['\multicolumn{1}{|l|}{\textbf{Active Nodes
    }} & $',num2str(averages.ActiveNodes(1)),'$ & $',
    num2str(averages.ActiveNodes(2)),'$ & $',num2str(
    averages.ActiveNodes(3)),'$ & $',num2str(averages.
    ActiveNodes(4)),'$\\ \hline'];
158 tobuild{05} = ['\multicolumn{1}{|l|}{\textbf{Node
    Activity Potential}} & $',num2str(averages.ActivityPot
    (1)),'$ & $',num2str(averages.ActivityPot(2)),'$ & $',
    num2str(averages.ActivityPot(3)),'$ & $',num2str(
    averages.ActivityPot(4)),'$\\ \hline'];
159 tobuild{06} = ['\multicolumn{1}{|l|}{\textbf{Global
    Clustering Coefficient}} & $',num2str(averages.GCC(1))
    ,'$ & $',num2str(averages.GCC(2)),'$ & $',num2str(
    averages.GCC(3)),'$ & $',num2str(averages.GCC(4)),'$\\
     \hline'];
160 tobuild{07} = ['\multicolumn{1}{|l|}{\textbf{Interaction
    Time}} & $',num2str(averages.OnTimes(1)),'$ & $',
    num2str(averages.OnTimes(2)),'$ & $',num2str(averages.
    OnTimes(3)),'$ & $',num2str(averages.OnTimes(4)),'$\\
    \hline'];
161 tobuild{08} = ['\multicolumn{1}{|l|}{\textbf{Time Between
     Contacts}} & $',num2str(averages.OffTimes(1)),'$ & $'
    ,num2str(averages.OffTimes(2)),'$ & $',num2str(
    averages.OffTimes(3)),'$ & $',num2str(averages.
```

```
   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}';

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

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

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

```
7  % MCMCLENGTH is the number of MCMC steps
8  % SIG is a 1x2 vector containing the variation in the
       MCMC steps
9  % PARCHAINS is the number of parallel chains to run
       simultaneously
10 % PRIORS is a 1x2 structure containing distribution
       objects for the priors
11 %
12 % CHAINS is a 1x2 structure containing
       MCMCLENGTHxPARCHAINS vectors
13 % containing the MCMC value for each parameter at each
       step
14
15 paracount = 2;
16
17 priordist_p1 = priors.p1;
18 priordist_p2 = priors.p2;
19
20 pr1 = pararange(:,1);
21 pr2 = pararange(:,2);
22
23 fullchains_p1 = zeros(parchains,MCMClength);
24 fullchains_p2 = zeros(parchains,MCMClength);
25
26 parfor i=1:parchains
27     currparas = unifrnd(pr1,pr2);
28     for j=1:MCMClength
29         fullchains_p1(i,j) = currparas(1);
30         fullchains_p2(i,j) = currparas(2);
31         scp = currparas;
32         for k=1:paracount
33             if sig(k)>0
34                 spp = scp;
35                 jd = makedist('Normal','mu',scp(k),'sigma
                       ',sig(k));
36                 tj = truncate(jd,pr1(k),pr2(k))
37                 spp(k) = random(tj);
38                 indivterms = vpa(zeros(2,length(data)));
39                 for m=1:length(data)
40                     indivterms(:,m) = [-(1/data(m))*ml(-(
                           data(m)/spp(2))^spp(1),spp(1),0)
                           ;-(1/data(m))*ml(-(data(m)/scp(2))
                           ^scp(1),scp(1),0)];
41                 end
42                 prob = prod(indivterms,2);
43                 scaling = vpa(zeros(paracount,1));
44                 for m=1:paracount
45                     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
                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)]);
                cv = unifrnd(0,1);
                if cv<p_move
                    scp = spp;
                end
            end
        end
        currparas = scp;
    end
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
eMCMClength = 100;                  %MCMC chain length for sig estimation
tarAR = [0.39,0.49];                %Target acceptance rate
MCMClength = 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
```

```
47 priors.p2 = makedist('Uniform','lower',pararange(2,1),'
      upper',pararange(2,2));  %Prior for second parameter
48 %===PRIORS===%
49
50 sig = MCMCestimatesig_mlf2(pararange,data,eMCMClength,
      parchains,priors,tarAR);
51
52 chains = MCMCmultiCW_mlf2(pararange,data,MCMClength,sig,
      parchains,priors);
53 [X,P] = scaledposterior2_mlf2(pararange,data,SPpoints,
      priors);
54 [marginals,SPmean,SPstd] = scaledmarginals(P,X);
55
56 bchain1 = chains.p1(:,MCMCburn+1:end);
57 bchain2 = chains.p2(:,MCMCburn+1:end);
58 bmchain.p1 = reshape(bchain1,1,[]);
59 bmchain.p2 = reshape(bchain2,1,[]);
60 MCMCmean = [mean(bchain1(:)),mean(bchain2(:))];
61 MCMCstd = [std(bchain1(:)),std(bchain2(:))];
62
63 g1 = ceil(sqrt(paracount));
64 g2 = ceil(paracount/g1);
65
66 pararefs = fieldnames(chains);
67
68 nf1 = figure('DefaultAxesFontSize',18);
69 for i=1:paracount
70     hax = subplot(g1,g2,i);
71     thischains = chains.(pararefs{i});
72     hold on
73     for j=1:parchains
74         plot(thischains(j,:));
75     end
76     axis manual
77     axis([0 MCMClength pararange(i,1) pararange(i,2)]);
78     line(get(hax,'XLim'),[MCMCmean(i) MCMCmean(i)],'Color
          ',[1 0 0],'LineWidth',1.5);
79     line(get(hax,'XLim'),[MCMCmean(i)+MCMCstd(i) MCMCmean
          (i)+MCMCstd(i)],'Color',[1 0 0],'LineStyle','--','
          LineWidth',1.5);
80     line(get(hax,'XLim'),[MCMCmean(i)-MCMCstd(i) MCMCmean
          (i)-MCMCstd(i)],'Color',[1 0 0],'LineStyle','--','
          LineWidth',1.5);
81     h = fill([0 0 MCMCburn MCMCburn],[get(hax,'YLim')
          fliplr(get(hax,'YLim'))],'r','EdgeColor','none');
82     set(h,'FaceAlpha',0.25);
83     if nargin==paracount+3
```

```
84          line(get(hax,'XLim'),[varargin{i} varargin{i}],'
               Color',[0 0 0],'LineWidth',1.5);
85      end
86      title(['Markov Chains for ',varstr{i}]);
87      hold off
88  end
89
90  nf1.WindowState = 'maximized';
91  saveas(nf1,[oF,'/',dataname,'_chains.fig']);
92  saveas(nf1,[oF_eps,'/',dataname,'_chains'],'epsc');
93  close(nf1);
94
95  nf2 = figure('DefaultAxesFontSize',18);
96  for i=1:paracount
97      hax = subplot(g1,g2,i);
98      hold on
99      if paracount==2
100         h = histogram(bmchain.(pararefs{i}),'
               Normalization','pdf');
101         histdata(1,:) = h.BinEdges(2:end)-h.BinWidth;
102         histdata(2,:) = h.Values;
103         hdata.(pararefs{i}) = histdata;
104         clear histdata
105     else
106         histogram(bmchain.(pararefs{i}),'Normalization','
               pdf');
107     end
108     plot(X.(pararefs{i}),marginals.(pararefs{i}),'
           LineWidth',1.5);
109     plot(X.(pararefs{i}),pdf(priors.(pararefs{i}),X.(
           pararefs{i})),'LineWidth',1.5);
110     axis manual
111     xlim([pararange(i,1) pararange(i,2)]);
112     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');
113     set(h1,'FaceAlpha',0.25);
114     line([MCMCmean(i) MCMCmean(i)],get(hax,'YLim'),'Color
           ','r','LineWidth',1.5);
115     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');
116     set(h2,'FaceAlpha',0.25);
117     line([SPmean(i) SPmean(i)],get(hax,'YLim'),'Color','b
           ','LineWidth',1.5);
118     if nargin==paracount+3
```

```
119         line([varargin{i} varargin{i}],get(hax,'YLim'),'
               Color',[0 0 0],'LineWidth',1.5);
120     end
121     title(['Estimated Densities for ',varstr{i}]);
122     hold off
123 end
124 
125 nf2.WindowState = 'maximized';
126 saveas(nf2,[oF,'/',dataname,'_densities.fig']);
127 saveas(nf2,[oF_eps,'/',dataname,'_densities'],'epsc');
128 close(nf2);
129 
130 if paracount==2
131     h1 = hdata.(pararefs{1});
132     h2 = hdata.(pararefs{2});
133     x1 = h1(1,:);
134     y1 = h1(2,:);
135     x2 = h2(1,:);
136     y2 = h2(2,:);
137     y = y2.*y1';
138 
139     bottom = min(min(min(double(P))),min(min(y)));
140     top  = max(max(max(double(P))),max(max(y)));
141 
142     nf3 = figure('DefaultAxesFontSize',18);
143     subplot(1,2,1);
144     h = surf(X.(pararefs{2}),X.(pararefs{1}),double(P));
145     set(h,'edgecolor','none');
146     hold on
147     line([SPmean(2) SPmean(2)],[pararange(1,1) pararange
           (1,2)],[top top],'color','b','linewidth',1.5);
148     line([pararange(2,1) pararange(2,2)],[SPmean(1)
           SPmean(1)],[top top],'color','b','linewidth',1.5);
149     if nargin==paracount+3
150         line([varargin{2} varargin{2}],[pararange(1,1)
               pararange(1,2)],[top top],'color',[0 0 0],'
               linewidth',1.5);
151         line([pararange(2,1) pararange(2,2)],[varargin{1}
                varargin{1}],[top top],'color',[0 0 0],'
               linewidth',1.5);
152     end
153     caxis manual
154     caxis([bottom top]);
155     title('Multidimensional Density Function (Exact
           Method)')
156     xlim([pararange(2,1) pararange(2,2)]);
157     ylim([pararange(1,1) pararange(1,2)]);
158     view(0,90);
```

```
159     xlabel(varstr{2});
160     ylabel(varstr{1});
161 
162     subplot(1,2,2);
163     h = surf(x2,x1,y);
164     set(h,'edgecolor','none');
165     hold on
166     h = surf([pararange(2,1) pararange(2,2)],[pararange
           (1,1) pararange(1,2)],[[0,0];[0,0]]);
167     set(h,'edgecolor','none');
168     hold on
169     line([MCMCmean(2) MCMCmean(2)],[pararange(1,1)
           pararange(1,2)],[top top],'color','r','linewidth'
           ,1.5);
170     line([pararange(2,1) pararange(2,2)],[MCMCmean(1)
           MCMCmean(1)],[top top],'color','r','linewidth'
           ,1.5);
171     if nargin==paracount+3
172         line([varargin{2} varargin{2}],[pararange(1,1)
               pararange(1,2)],[top top],'color',[0 0 0],'
               linewidth',1.5);
173         line([pararange(2,1) pararange(2,2)],[varargin{1}
                varargin{1}],[top top],'color',[0 0 0],'
               linewidth',1.5);
174     end
175     caxis manual
176     caxis([bottom top]);
177     title('Multidimensional Density Function (Monte Carlo
            Method)')
178     xlim([pararange(2,1) pararange(2,2)]);
179     ylim([pararange(1,1) pararange(1,2)]);
180     view(0,90);
181     xlabel(varstr{2});
182     ylabel(varstr{1});
183 
184     nf3.WindowState = 'maximized';
185     saveas(nf3,[oF,'/',dataname,'_mdf.fig']);
186     saveas(nf3,[oF_eps,'/',dataname,'_mdf'],'epsc');
187     close(nf3);
188 end
189 
190 [F,X] = ecdf(data);
191 
192 MCMC1 = ml(-(X/(MCMCmean(2)+(2*MCMCstd(2)))).^(MCMCmean
       (1)-(2*MCMCstd(1))),MCMCmean(1)-(2*MCMCstd(1)));
193 MCMC2 = ml(-(X/(MCMCmean(2)+(1*MCMCstd(2)))).^(MCMCmean
       (1)-(1*MCMCstd(1))),MCMCmean(1)-(1*MCMCstd(1)));
194 MCMC3 = ml(-(X/MCMCmean(2)).^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;

```

```
235 save([oF,'/',dataname,'_results.mat'],'soln_MCMC','
    soln_SP');
```

B.50 ml.m

This function was written by Garrappa [73].

```
1  function E = ml(z,alpha,beta,gama)
2  %
3  % Evaluation of the Mittag-Leffler (ML) function with 1,
     2 or 3 parameters
4  % by means of the OPC algorithm [1]. The routine
     evaluates an approximation
5  % Et of the ML function E such that |E-Et|/(1+|E|) approx
      1.0e-15
6  %
7  %
8  % E = ML(z,alpha) evaluates the ML function with one
     parameter alpha for
9  % the corresponding elements of z; alpha must be a real
     and positive
10 % scalar. The one parameter ML function is defined as
11 %
12 % E = sum_{k=0}^{infty} z^k/Gamma(alpha*k+1)
13 %
14 % with Gamma the Euler's gamma function.
15 %
16 %
17 % E = ML(z,alpha,beta) evaluates the ML function with two
      parameters alpha
18 % and beta for the corresponding elements of z; alpha
     must be a real and
19 % positive scalar and beta a real scalar. The two
     parameters ML function is
20 % defined as
21 %
22 % E = sum_{k=0}^{infty} z^k/Gamma(alpha*k+beta)
23 %
24 %
25 % E = ML(z,alpha,beta,gama) evaluates the ML function
     with three parameters
26 % alpha, beta and gama for the corresponding elements of
     z; alpha must be a
27 % real scalar such that 0<alpha<1, beta any real scalar
     and gama a real and
28 % positive scalar; the arguments z must satisfy |Arg(z)|
     > alpha*pi. The
29 % three parameters ML function is defined as
```

```
30 %
31 % E = sum_{k=0}^{infty} Gamma(gama+k)*z^k/Gamma(gama)/k!/
       Gamma(alpha*k+beta)
32 %
33 %
34 % NOTE:
35 % This routine implements the optimal parabolic contour (
       OPC) algorithm
36 % described in [1] and based on the inversion of the
       Laplace transform on a
37 % parabolic contour suitably choosen in one of the
       regions of analyticity
38 % of the Laplace transform.
39 %
40 %
41 % REFERENCES
42 %
43 %   [1] R. Garrappa, Numerical evaluation of two and
       three parameter
44 %   Mittag-Leffler functions, SIAM Journal of Numerical
       Analysis, 2015,
45 %   53(3), 1350-1369
46 %
47 %
48 %   Please, report any problem or comment to :
49 %        roberto dot garrappa at uniba dot it
50 %
51 %   Copyright (c) 2015, Roberto Garrappa, University of
       Bari, Italy
52 %   roberto dot garrappa at uniba dot it
53 %   Homepage: http://www.dm.uniba.it/Members/garrappa
54 %   Revision: 1.4 - Date: October 8 2015
55
56
57 % Check inputs
58 if nargin < 4
59     gama = 1 ;
60     if nargin < 3
61         beta = 1 ;
62         if nargin < 2
63             error('MATLAB:ml:NumberParameters', ...
64                 'The parameter ALPHA must be specified.')
                    ;
65         end
66     end
67 end
68
69 % 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)) ;
```

```
106 for k = 1 : length(z)
107     if abs(z(k)) < 1.0e-15
108         E(k) = 1/gamma(beta) ;
109     else
110         E(k) = LTInversion(1,z(k),alpha,beta,gama,
                log_epsilon) ;
111     end
112 end
113
114
115 end
116
117
118 % ================================================
119 % Evaluation of the ML function by Laplace transform
    inversion
120 % ================================================
121 function E = LTInversion(t,lambda,alpha,beta,gama,
    log_epsilon)
122
123 % Evaluation of the relevant poles
124 theta = angle(lambda) ;
125 kmin = ceil(-alpha/2 - theta/2/pi) ;
126 kmax = floor(alpha/2 - theta/2/pi) ;
127 k_vett = kmin : kmax ;
128 s_star = abs(lambda)^(1/alpha) * exp(1i*(theta+2*k_vett*
    pi)/alpha) ;
129
130 % Evaluation of phi(s_star) for each pole
131 phi_s_star = (real(s_star)+abs(s_star))/2 ;
132
133 % Sorting of the poles according to the value of phi(
    s_star)
134 [phi_s_star , index_s_star ] = sort(phi_s_star) ;
135 s_star = s_star(index_s_star) ;
136
137 % Deleting possible poles with phi_s_star=0
138 index_save = phi_s_star > 1.0e-15 ;
139 s_star = s_star(index_save) ;
140 phi_s_star = phi_s_star(index_save) ;
141
142 % Inserting the origin in the set of the singularities
143 s_star = [0, s_star] ;
144 phi_s_star = [0, phi_s_star] ;
145 J1 = length(s_star) ; J = J1 - 1 ;
146
147 % Strength of the singularities
148 p = [ max(0,-2*(alpha*gama-beta+1)) , ones(1,J)*gama ]  ;
```

```
149 q = [ ones(1,J)*gama , +Inf] ;
150 phi_s_star = [phi_s_star, +Inf] ;
151
152 % Looking for the admissible regions with respect to
      round-off errors
153 admissible_regions = find( ...
154     (phi_s_star(1:end-1) < (log_epsilon - log(eps))/t) &
          ...
155     (phi_s_star(1:end-1) < phi_s_star(2:end))) ;
156
157 % Initializing vectors for optimal parameters
158 JJ1 = admissible_regions(end) ;
159 mu_vett = ones(1,JJ1)*Inf ;
160 N_vett = ones(1,JJ1)*Inf ;
161 h_vett = ones(1,JJ1)*Inf ;
162
163 % Evaluation of parameters for inversion of LT in each
      admissible region
164 find_region = 0 ;
165 while ~find_region
166     for j1 = admissible_regions
167         if j1 < J1
168             [muj,hj,Nj] = OptimalParam_RB ...
169                 (t,phi_s_star(j1),phi_s_star(j1+1),p(j1),
                        q(j1),log_epsilon) ;
170         else
171             [muj,hj,Nj] = OptimalParam_RU(t,phi_s_star(j1
                    ),p(j1),log_epsilon) ;
172         end
173         mu_vett(j1) = muj ; h_vett(j1) = hj ; N_vett(j1)
                = Nj ;
174     end
175     if min(N_vett) > 200
176         log_epsilon = log_epsilon +log(10) ;
177     else
178         find_region = 1 ;
179     end
180 end
181
182
183 % Selection of the admissible region for integration
      which involves the
184 % minimum number of nodes
185 [N, iN] = min(N_vett) ; mu = mu_vett(iN) ; h = h_vett(iN)
      ;
186
187 % Evaluation of the inverse Laplace transform
188 k = -N : N ;
```

```
189 u = h*k ;
190 z = mu*(1i*u+1).^2 ;
191 zd = -2*mu*u + 2*mu*1i ;
192 zexp = exp(z*t) ;
193 F = z.^(alpha*gama-beta)./(z.^alpha - lambda).^gama.*zd ;
194 S = zexp.*F ;
195 Integral = h*sum(S)/2/pi/1i ;
196
197 % Evaluation of residues
198 ss_star = s_star(iN+1:end) ;
199 Residues = sum(1/alpha*(ss_star).^(1-beta).*exp(t*ss_star
       )) ;
200
201 % Evaluation of the ML function
202 E = Integral + Residues ;
203 if isreal(lambda)
204     E = real(E) ;
205 end
206
207 end
208
209
210 % ================================================
211 % Finding optimal parameters in a right-bounded region
212 % ================================================
213 function [muj,hj,Nj] = OptimalParam_RB ...
214     (t, phi_s_star_j, phi_s_star_j1, pj, qj, log_epsilon)
215
216 % Definition of some constants
217 log_eps = -36.043653389117154 ; % log(eps)
218 fac = 1.01 ;
219 conservative_error_analysis = 0 ;
220
221 % Maximum value of fbar as the ration between tolerance
       and round-off unit
222 f_max = exp(log_epsilon - log_eps) ;
223
224 % Evaluation of the starting values for sq_phi_star_j and
       sq_phi_star_j1
225 sq_phi_star_j = sqrt(phi_s_star_j) ;
226 threshold = 2*sqrt((log_epsilon - log_eps)/t) ;
227 sq_phi_star_j1 = min(sqrt(phi_s_star_j1), threshold -
       sq_phi_star_j) ;
228
229 % Zero or negative values of pj and qj
230 if pj < 1.0e-14 && qj < 1.0e-14
231     sq_phibar_star_j = sq_phi_star_j ;
232     sq_phibar_star_j1 = sq_phi_star_j1 ;
```

```
233     adm_region = 1 ;
234 end
235
236 % Zero or negative values of just pj
237 if pj < 1.0e-14 && qj >= 1.0e-14
238     sq_phibar_star_j = sq_phi_star_j ;
239     if sq_phi_star_j > 0
240         f_min = fac*(sq_phi_star_j/(sq_phi_star_j1-
              sq_phi_star_j))^qj ;
241     else
242         f_min = fac ;
243     end
244     if f_min < f_max
245         f_bar = f_min + f_min/f_max*(f_max-f_min) ;
246         fq = f_bar^(-1/qj) ;
247         sq_phibar_star_j1 = (2*sq_phi_star_j1-fq*
              sq_phi_star_j)/(2+fq) ;
248         adm_region = 1 ;
249     else
250         adm_region = 0 ;
251     end
252 end
253
254 % Zero or negative values of just qj
255 if pj >= 1.0e-14 && qj < 1.0e-14
256     sq_phibar_star_j1 = sq_phi_star_j1 ;
257     f_min = fac*(sq_phi_star_j1/(sq_phi_star_j1-
          sq_phi_star_j))^pj ;
258     if f_min < f_max
259         f_bar = f_min + f_min/f_max*(f_max-f_min) ;
260         fp = f_bar^(-1/pj) ;
261         sq_phibar_star_j = (2*sq_phi_star_j+fp*
              sq_phi_star_j1)/(2-fp) ;
262         adm_region = 1 ;
263     else
264         adm_region = 0 ;
265     end
266 end
267
268 % Positive values of both pj and qj
269 if pj >= 1.0e-14 && qj >= 1.0e-14
270     f_min = fac*(sq_phi_star_j+sq_phi_star_j1)/...
271         (sq_phi_star_j1-sq_phi_star_j)^max(pj,qj) ;
272     if f_min < f_max
273         f_min = max(f_min,1.5) ;
274         f_bar = f_min + f_min/f_max*(f_max-f_min) ;
275         fp = f_bar^(-1/pj) ;
276         fq = f_bar^(-1/qj) ;
```

```
277         if ~conservative_error_analysis
278             w = -phi_s_star_j1*t/log_epsilon ;
279         else
280             w = -2*phi_s_star_j1*t/(log_epsilon-
                   phi_s_star_j1*t) ;
281         end
282         den = 2+w - (1+w)*fp + fq ;
283         sq_phibar_star_j = ((2+w+fq)*sq_phi_star_j + fp*
               sq_phi_star_j1)/den ;
284         sq_phibar_star_j1 = (-(1+w)*fq*sq_phi_star_j ...
285             + (2+w-(1+w)*fp)*sq_phi_star_j1)/den ;
286         adm_region = 1 ;
287     else
288         adm_region = 0 ;
289     end
290 end
291
292 if adm_region
293     log_epsilon = log_epsilon  - log(f_bar) ;
294     if ~conservative_error_analysis
295         w = -sq_phibar_star_j1^2*t/log_epsilon ;
296     else
297         w = -2*sq_phibar_star_j1^2*t/(log_epsilon-
               sq_phibar_star_j1^2*t) ;
298     end
299     muj = (((1+w)*sq_phibar_star_j + sq_phibar_star_j1)
           /(2+w))^2 ;
300     hj = -2*pi/log_epsilon*(sq_phibar_star_j1-
           sq_phibar_star_j)...
301         /((1+w)*sq_phibar_star_j + sq_phibar_star_j1) ;
302     Nj = ceil(sqrt(1-log_epsilon/t/muj)/hj) ;
303 else
304     muj = 0 ; hj = 0 ; Nj = +Inf ;
305 end
306
307 end
308
309 % ====================================================
310 % Finding optimal parameters in a right-unbounded region
311 % ====================================================
312 function [muj,hj,Nj] = OptimalParam_RU (t, phi_s_star_j,
       pj, log_epsilon)
313
314 % Evaluation of the starting values for sq_phi_star_j
315 sq_phi_s_star_j = sqrt(phi_s_star_j) ;
316 if phi_s_star_j > 0
317     phibar_star_j = phi_s_star_j*1.01 ;
318 else
```

```
319     phibar_star_j = 0.01 ;
320 end
321 sq_phibar_star_j = sqrt(phibar_star_j) ;
322
323 % Definition of some constants
324 f_min = 1 ; f_max = 10 ; f_tar = 5 ;
325
326 % Iterative process to look for fbar in [f_min,f_max]
327 stop = 0 ;
328 while ~stop
329     phi_t = phibar_star_j*t ; log_eps_phi_t = log_epsilon
        /phi_t ;
330     Nj = ceil(phi_t/pi*(1 - 3*log_eps_phi_t/2 + sqrt(1-2*
        log_eps_phi_t))) ;
331     A = pi*Nj/phi_t ;
332     sq_muj = sq_phibar_star_j*abs(4-A)/abs(7-sqrt(1+12*A)
        ) ;
333     fbar = ((sq_phibar_star_j-sq_phi_s_star_j)/sq_muj)^(-
        pj) ;
334     stop = (pj < 1.0e-14) || (f_min < fbar && fbar <
        f_max) ;
335     if ~stop
336         sq_phibar_star_j = f_tar^(-1/pj)*sq_muj +
            sq_phi_s_star_j ;
337         phibar_star_j = sq_phibar_star_j^2 ;
338     end
339 end
340 muj = sq_muj^2 ;
341 hj = (-3*A - 2 + 2*sqrt(1+12*A))/(4-A)/Nj ;
342
343 % Adjusting integration parameters to keep round-off
    errors under control
344 log_eps = log(eps) ; threshold = (log_epsilon - log_eps)/
    t ;
345 if muj > threshold
346     if abs(pj) < 1.0e-14 , Q = 0 ; else Q = f_tar^(-1/pj)
        *sqrt(muj) ; end
347     phibar_star_j = (Q + sqrt(phi_s_star_j))^2 ;
348     if phibar_star_j < threshold
349         w = sqrt(log_eps/(log_eps-log_epsilon)) ;
350         u = sqrt(-phibar_star_j*t/log_eps) ;
351         muj = threshold ;
352         Nj = ceil(w*log_epsilon/2/pi/(u*w-1)) ;
353         hj = sqrt(log_eps/(log_eps - log_epsilon))/Nj ;
354     else
355         Nj = +Inf ; hj = 0 ;
356     end
357 end
```

```
358
359 end
```

B.51 mlf.m

This function was written by Podludny and Kacenak [151].

```
1  function [e]=mlf(alf,bet,c,fi)
2  %
3  % MLF -- Mittag-Leffler function.
4  %           MLF (alpha,beta,Z,P) is the Mittag-Leffler
       function E_{alpha,beta}(Z)
5  %           evaluated with accuracy 10^(-P) for each
       element of Z.
6  % (C) 2001-2005 Igor Podlubny, Martin Kacenak
7
8
9  if nargin<4 , fi=6;                end
10 if nargin<3 || alf<=0 || fi<=0
11 else
12    [r,s]=size(c); [r1,s1]=size(alf); [r2,s2]=size(bet);
13    mx=max([r,s]); mx1=max([r1,s1]); mx2=max([r2,s2]);
14    if (r>1 && s>1) || (r1>1 && s1>1) || (r2>1 && s2>1) ||
          (mx1>1 && mx2>1)
15       sprintf('wrong number of input parameters')
16    else
17       if mx1>mx2 , mxx=mx1; e=zeros(mx,mx1);
18       else mxx=mx2; e=zeros(mx,mx2);end;
19       for i1= 1:mx
20          for i2=1:mxx
21
22             if r>s , z=c(i1,1); else z=c(1,i1); end
23             if mx1>mx2 , if r1>s1 , alfa=alf(i2,1); else
                  alfa=alf(1,i2);end, beta=bet;
24             else if r2>s2 ,beta=bet(i2,1); else beta=bet
                  (1,i2); end, alfa=alf; end
25             if beta<0 , rc=(-2*log(10^(-fi)*pi/(6*(abs(
                  beta)+2)*(2*abs(beta))^(abs(beta)))))^alfa
                  ;
26                else  rc=(-2*log(10^(-fi)*pi/6))^alfa; end
27             r0=max([1,2*abs(z),rc]);
28             if (alfa==1 && beta==1)
29                e(i1,i2)=exp(z);
30             else
31                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)
```

```
32                  oldsum=0;
33                  k=0;
34                  while (alfa*k+beta)<=0
35                     k=k+1;
36                  end
37                  newsum=z^k/gamma(alfa*k+beta);
38                  while newsum~=oldsum
39                     oldsum=newsum;
40                     k=k+1;
41                     term=z^k/gamma(alfa*k+beta);
42                     newsum=newsum+term;
43                     k=k+1;
44                     term=z^k/gamma(alfa*k+beta);
45                     newsum=newsum+term;
46                  end
47                  e(i1,i2)=newsum;
48               else
49                  if (alfa<=1 && abs(z)<=fix(5*alfa+10))
50                     if ((abs(angle(z))>pi*alfa) && (abs(
                          abs(angle(z))-(pi*alfa))>10^(-fi)
                          ))
51                        if beta<=1
52                           e(i1,i2)=rombint('K',0,r0,fi,
                                alfa,beta,z);
53                        else
54                          eps=1;
55                          e(i1,i2)=rombint('K',eps,r0,fi,
                                alfa,beta,z)+ ...
56                             rombint('P',-pi*alfa,pi*alfa
                                ,fi,alfa,beta,z,eps);
57                        end
58                     elseif (abs(angle(z))<pi*alfa && abs(
                          abs(angle(z))-(pi*alfa))>10^(-fi))
59                        if beta<=1
60                           e(i1,i2)=rombint('K',0,r0,fi,
                                alfa,beta,z)+ ...
61                              (z^((1-beta)/alfa))*(exp(z
                                ^(1/alfa))/alfa);
62                        else
63                           eps=abs(z)/2;
64                           e(i1,i2)=rombint('K',eps,r0,fi
                                ,alfa,beta,z)+ ...
65                              rombint('P',-pi*alfa,pi*
                                alfa,fi,alfa,beta,z,eps)
                                + ...
66                              (z^((1-beta)/alfa))*(exp(z
                                ^(1/alfa))/alfa);
67                        end
```

```
68                      else
69                          eps=abs(z)+0.5;
70                          e(i1,i2)=rombint('K',eps,r0,fi,
                                alfa,beta,z)+ ...
71                              rombint('P',-pi*alfa,pi*alfa,
                                    fi,alfa,beta,z,eps);
72                      end
73                  else
74                      if alfa<=1
75                          if (abs(angle(z))<(pi*alfa/2+min(
                                pi,pi*alfa))/2)
76                              % alfa
77                              newsum=(z^((1-beta)/alfa))*exp
                                    (z^(1/alfa))/alfa;
78                              for k=1:floor(fi/log10(abs(z))
                                    )
79                                  newsum=newsum-((z^(-k))/
                                        gamma(beta-alfa*k));
80                                  % k
81                              end
82                              e(i1,i2)=newsum;
83                          else
84                              newsum=0;
85                              for k=1:floor(fi/log10(abs(z))
                                    )
86                                  newsum=newsum-((z^-k)/gamma
                                        (beta-alfa*k));
87                              end
88                              e(i1,i2)=newsum;
89                          end
90                      else
91                          if alfa>=2
92                              m=floor(alfa/2);
93                              sum=0;
94                              for h=0:m
95                                  zn=(z^(1/(m+1)))*exp((2*pi
                                        *1i*h)/(m+1));
96                                  sum=sum+mlf(alfa/(m+1),beta
                                        ,zn,fi);
97                              end
98                              e(i1,i2)=(1/(m+1))*sum;
99                          else
100                             e(i1,i2)=(mlf(alfa/2,beta,z
                                    ^(1/2),fi)+mlf(alfa/2,beta
                                    ,-z^(1/2),fi))/2;
101                         end
102                     end
103                 end
```

```
104                 end
105             end
106         end
107     end
108   end
109 end
110
111
112 function [res]=rombint(funfcn,a,b,order,varargin)
113 if nargin<4 ,order=6; end
114 if nargin<3
115    Warning ('Error in input format')
116 else
117    rom=zeros(2,order);
118    h=b-a;
119    rom(1,1)=h*(feval(funfcn,a,varargin{:})+feval(funfcn,b
          ,varargin{:}))/2;
120
121    ipower=1;
122    for i= 2:order
123       sum=0;
124       for j=1:ipower
125         sum=sum+feval(funfcn,(a+h*(j-0.5)),varargin{:});
126       end
127       rom(2,1)=(rom(1,1)+h*sum)/2;
128       for k=1:i-1
129          rom(2,k+1)=((4^k)*rom(2,k)-rom(1,k))/((4^k)-1);
130       end
131
132       for j=0:i-1
133          rom(1,j+1)=rom(2,j+1);
134       end
135       ipower=ipower*2;
136       h=h/2;
137    end
138    res=rom(1,order);
139 end
140
141
142 function [res]=K(r,alfa,beta,z)
143 res=r.^((1-beta)/alfa).*exp(-r.^(1/alfa)).*(r*sin(pi*(1-
       beta))-...
144 z*sin(pi*(1-beta+alfa)))/(pi*alfa*(r.^2-2*r*z*cos(pi*alfa
       )+z.^2));
145
146
147 function [res]=P(r,alfa,beta,z,eps)
148 w=(eps^(1/alfa))*sin(r/alfa)+r*(1+(1-beta)/alfa);
```

```
149 res=((eps^(1+(1-beta)/alfa))/(2*pi*alfa))*((exp((eps^(1/
    alfa))*cos(r/alfa)).*...
150     (cos(w)+1i*sin(w))))/(eps*exp(1i*r)-z);
```

B.52 mlrnd.m

This function was written by Germano, Fulger and Scalas [76].

```
1  function t = mlrnd(beta, gamma_t, m, n)
2
3  % mlrnd.m: Mittag-Leffler pseudo-random number generator
4  %
5  % Authors: Guido Germano, Daniel Fulger, Enrico Scalas
       (2006-2008)
6  %
7  % Description: Returns a matrix of iid random numbers
       distributed according to
8  % the one-parameter Mittag-Leffler distribution with
       index (or exponent) beta
9  % and scale parameter gamma_t. The size of the returned
       matrix is the same as
10 % that of the input matrices beta and gamma_t, that must
       match. Alternatively,
11 % if beta and gamma_t are scalars, mlrnd(beta, gamma_t, m
       ) returns an m by m
12 % matrix, and mlrnd(beta, gamma_t, m, n) returns an m by
       n matrix.
13 %
14 % References:
15 % [1] T. J. Kozubowski and S. T. Rachev, "Univariate
       Geometric Stable Laws"
16 %     Journal of Computational Analysis and Applications
       1, 177-219 (1999).
17 % [2] D. Fulger, E. Scalas, G. Germano, "Monte Carlo
       simulation of uncoupled
18 %     continuous-time random walks yielding a stochastic
       solution of the space-
19 %     time fractional diffusion equation", Physical
       Review E 77, 021122 (2008).
20
21 % Check input
22 if nargin < 2
23     error('mlrnd requires at least two arguments.')
24 elseif nargin == 2
25     sb = size(beta);
26     sg = size(gamma_t);
27     if ~isscalar(beta) && ~isscalar(gamma_t) && any(sb ~=
           sg)
```

```
28          error('mlrnd: size mismatch between beta and
               gamma_t.')
29      end
30      m = max(sb(1),sg(1));
31      n = max(sb(2),sg(2));
32  else
33      if ~isscalar(beta) && ~isscalar(gamma_t)
34          error('mlrnd: beta and gamma_t must be scalars
               when m is given.')
35      end
36      if ~(isscalar(m) && m > 0 && m == round(m))
37          error('mlrnd: m must be a positive integer.')
38      end
39      if nargin == 3
40          n = m;
41      elseif ~(isscalar(n) && n > 0 && n == round(n))
42          error('mlrnd: n must be a positive integer.')
43      end
44  end
45  if any(any(beta <= 0)) || any(any(beta > 1))
46      error('mlrnd: the elements of beta must belong to the
           interval (0,1].')
47  end
48  if any(any(gamma_t <= 0))
49      error('mlrnd: the elements of gamma_t must be
           positive.')
50  end
51
52  % Generate m x n Mittag-Leffler pseudo-random numbers
53  t = -log(rand(m,n)); % Standard exponential deviate
54  if (beta < 1)
55      u = rand(m,n); % Uniform deviate on the unit interval
           , independent of t
56      w = sin(beta*pi)./tan(beta*pi.*u) - cos(beta*pi); %
           Auxiliary variable
57      t = t.*w.^(1./beta); % Standard one-parameter Mittag-
           Leffler deviate
58  end
59  t = gamma_t*t;
60
61  return
```

B.53 mm_ExpGamRayLN.m

```
1  function [mins,maxs] = mm_ExpGamRayLN(FT,MLE,Moments)
2  %MM_EXPMLGPWEI returns the minimum and maximum values of
       each parameter
```

```
3  %
4  % FT is the fit tool data structure
5  % MLE is the most likelihood estimators data structure
6  % MOMENTS is the method of moments data structure
7  % MINS is a vector containing the minimum values of each
       parameter
8  % MACS is a vector containing the maximum values of each
       parameter
9
10 brokenMins = Inf(1,6);
11 brokenMaxs = -Inf(1,6);
12
13 FT_Vec = zeros(1,6);
14 MLE_Vec = zeros(1,6);
15 Moments_Vec = zeros(1,6);
16
17 FT_Vec(1) = FT.Exponential.Parameters.Scale;
18 FT_Vec(2) = FT.Gamma.Parameters.Shape;
19 FT_Vec(3) = FT.Gamma.Parameters.Scale;
20 FT_Vec(4) = FT.Rayleigh.Parameters.Scale;
21 FT_Vec(5) = FT.LogNormal.Parameters.Location;
22 FT_Vec(6) = FT.LogNormal.Parameters.Scale;
23
24 MLE_Vec(1) = MLE.Exponential.Parameters.Scale;
25 MLE_Vec(2) = MLE.Gamma.Parameters.Shape;
26 MLE_Vec(3) = MLE.Gamma.Parameters.Scale;
27 MLE_Vec(4) = MLE.Rayleigh.Parameters.Scale;
28 MLE_Vec(5) = MLE.LogNormal.Parameters.Location;
29 MLE_Vec(6) = MLE.LogNormal.Parameters.Scale;
30
31 Moments_Vec(1) = Moments.Exponential.Parameters.Scale;
32 Moments_Vec(2) = Moments.Gamma.Parameters.Shape;
33 Moments_Vec(3) = Moments.Gamma.Parameters.Scale;
34 Moments_Vec(4) = Moments.Rayleigh.Parameters.Scale;
35 Moments_Vec(5) = Moments.LogNormal.Parameters.Location;
36 Moments_Vec(6) = Moments.LogNormal.Parameters.Scale;
37
38
39 mins = min([brokenMins;FT_Vec;MLE_Vec;Moments_Vec]);
40 maxs = max([brokenMaxs;FT_Vec;MLE_Vec;Moments_Vec]);
```

B.54 mm_ExpMLGPWei.m

```
1  function [mins,maxs] = mm_ExpMLGPWei(FT,MLE,Moments)
2  %MM_EXPMLGPWEI returns the minimum and maximum values of
       each parameter
3  %
```

```
4  % FT is the fit tool data structure
5  % MLE is the most likelihood estimators data structure
6  % MOMENTS is the method of moments data structure
7  % MINS is a vector containing the minimum values of each
       parameter
8  % MACS is a vector containing the maximum values of each
       parameter
9
10 brokenMins = Inf(1,8);
11 brokenMaxs = -Inf(1,8);
12
13 FT_Vec = zeros(1,8);
14 MLE_Vec = zeros(1,8);
15 Moments_Vec = zeros(1,8);
16
17 FT_Vec(1) = FT.Exponential.Parameters.Scale;
18 FT_Vec(2) = FT.MittagLeffler.Parameters.Stability;
19 FT_Vec(3) = FT.MittagLeffler.Parameters.Scale;
20 FT_Vec(4) = FT.GenPareto.Parameters.Shape;
21 FT_Vec(5) = FT.GenPareto.Parameters.Scale;
22 FT_Vec(6) = FT.GenPareto.Parameters.Location;
23 FT_Vec(7) = FT.Weibull.Parameters.Scale;
24 FT_Vec(8) = FT.Weibull.Parameters.Shape;
25
26 MLE_Vec(1) = MLE.Exponential.Parameters.Scale;
27 MLE_Vec(2) = NaN;
28 MLE_Vec(3) = NaN;
29 MLE_Vec(4) = NaN;
30 MLE_Vec(5) = NaN;
31 MLE_Vec(6) = NaN;
32 MLE_Vec(7) = MLE.Weibull.Parameters.Scale;
33 MLE_Vec(8) = MLE.Weibull.Parameters.Shape;
34
35 Moments_Vec(1) = Moments.Exponential.Parameters.Scale;
36 Moments_Vec(2) = NaN;
37 Moments_Vec(3) = NaN;
38 Moments_Vec(4) = Moments.GenPareto.Parameters.Shape;
39 Moments_Vec(5) = Moments.GenPareto.Parameters.Scale;
40 Moments_Vec(6) = Moments.GenPareto.Parameters.Location;
41 Moments_Vec(7) = NaN;
42 Moments_Vec(8) = NaN;
43
44 mins = min([brokenMins;FT_Vec;MLE_Vec;Moments_Vec]);
45 maxs = max([brokenMaxs;FT_Vec;MLE_Vec;Moments_Vec]);
```

B.55 model.m

```
function [] = model(nodes,runtime,filepath)
%MODDEL 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 emprical 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
%
% 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
```

```
36      for j=i+1:nodes
37          init = initial(i,j);
38          currenttime = startthings(i,j);
39          if init == 0
40              if startthings(i,j)<runtime
41                  thisoff = [startthings(i,j)];
42              else
43                  thisoff = [];
44              end
45              thison = [];
46              while currenttime<runtime
47                  thisoffduration = lognrnd(LNpara1(i,j),
                        LNpara2(i,j));
48                  switch_on = currenttime+thisoffduration;
49                  thisonduration = exprnd(EXpara1(i,j));
50                  switch_off = switch_on+thisonduration;
51                  if switch_on<runtime
52                      thison = [thison,switch_on];
53                      if switch_off<runtime
54                          thisoff = [thisoff,switch_off];
55                      else
56                          thisoff = [thisoff,runtime];
57                      end
58                  else
59                      thison = [thison,runtime];
60                  end
61                  currenttime = switch_off;
62              end
63          elseif init == 1
64              thisoff = [];
65              if startthings(i,j)<runtime
66                  thison = [startthings(i,j)];
67              else
68                  thison = [];
69              end
70              while currenttime<runtime
71                  thisonduration = exprnd(EXpara1(i,j));
72                  switch_off = currenttime+thisonduration;
73                  thisoffduration = lognrnd(LNpara1(i,j),
                        LNpara2(i,j));
74                  switch_on = switch_off+thisoffduration;
75                  if switch_off<runtime
76                      thisoff = [thisoff,switch_off];
77                      if switch_on<runtime
78                          thison = [thison,switch_on];
79                      else
80                          thison = [thison,runtime];
81                      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) = [];
        thisoff(thisoff==0) = [];
        thison(thison==runtime) = [];
        thisoff(thisoff>runtime) = [];
        thison(thison>runtime) = [];
        if firston > firstoff
            thison = [switchon(i,j),thison];
        end
        ID_ref = sprintf('n%d_n%d', 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
% simulations
% RUNTIME is the desired simulation time
% DIR_REF is the output save directory as a string

timestamp = datestr(now,'yyyymmddTHHMMSS');
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
end
```

B.57 modelv2_1.m

```
function [] = modelv2_1(runtime,probM,filepath)
%MODDELV2_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 emprical
   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);
```

```
25 %INITIALOFF is the initial state for the network given as
       an adjacency
26 % matrix
27
28 ondurationpara1 = lognrnd(3.2434,sigma_for_mu_and_mean
       (30.552,3.2434),nodes);
29
30 times = struct();
31 %TIMES will end up having entries labeled nX_nY with
       entries showing the on
32 % and off times for the link X-Y as a single vector.
33 for i=1:nodes-1
34     for j=i+1:nodes
35         ID_ref = sprintf('n%d_n%d',i,j);
36         if initialoff(i,j)>0
37             times.(ID_ref) = [0,initialoff(i,j)];
38         else
39             times.(ID_ref) = [];
40         end
41     end
42 end
43
44 currenttime = lognrnd(5.6901e-04,1.7957);
45 %CURRENTTIME should be set to the desired distribution
       for on interevent
46 % timings
47
48 triangleActivations = 0.0556;
49 estFailure = 0.1316;
50 adjustedThreshold = triangleActivations/(1-estFailure);
51
52 while currenttime<runtime
53     flip = unifrnd(0,1);
54     if flip<adjustedThreshold
55         [~,m_lpm] = extractTriangles(times,nodes,
               currenttime,linkprobmatrix);
56         if sum(sum(m_lpm))~=0
57             thislinkmatrix = m_lpm;
58         else
59             thislinkmatrix = linkprobmatrix;
60         end
61     else
62         thislinkmatrix = linkprobmatrix;
63     end
64     accept = 0;
65     cycles = 0;
66     while accept == 0
67         [ri,rj] = chooselink(thislinkmatrix);
```

```
68          ID_ref = sprintf('n%d_n%d',ri,rj);
69          vec = times.(ID_ref);
70          if isempty(vec)||currenttime>vec(end)
71              %Ensure link is not already active
72              accept = 1;
73              ontime = currenttime;
74              offtime = currenttime+exprnd(ondurationpara1(
                    ri,rj));
75              vec = [vec,ontime,offtime];
76              %Append to entry in TIMES
77              times.(ID_ref) = vec;
78          end
79          if cycles>100
80              thislinkmatrix = linkprobmatrix;
81          else
82              cycles = cycles+1;
83          end
84      end
85      IET = lognrnd(5.6901e-04,1.7957);
86      %IET should be set to the desired distribution for on
            interevent
87      % timings
88      currenttime = currenttime+IET;
89  end
90  sampleCSV2(times,nodes,runtime,cut,filepath);
91  %Passes data to sampleCSV.m for sampling and conversion
        to CSV file
92  end
```

B.58 modelv2_1_multiple.m

```
1   function [dir_ref] = modelv2_1_multiple(runtime,
        probMstruc)
2   %MODELV1_MULTIPLE creates as many simulations using
        method 2b as there are
3   % entries in the structure PROBMSTRUC
4   %
5   % PROBMSTRUC is a structure containing the link selection
        matrices for each
6   % of the desired simulations
7   % RUNTIME is the desired simulation time
8   % DIR_REF is the output save directory as a string
9
10  timestamp = datestr(now,'yyyymmddTHHMMSS');
11  dir_ref = ['lots/output_',timestamp];
12  dir_ref_full = ['input/',dir_ref];
13  mkdir(dir_ref_full);
```

```
14
15 fields = fieldnames(probMstruc);
16 cycles = numel(fields);
17
18 for i=1:cycles
19     filename = ['data',num2str(i),'.csv']; %Save file
          name
20     filepath = [dir_ref_full,'/',filename];
21     probM = probMstruc.(fields{i});
22     modelv2_1(runtime,probM,filepath);
23 end
24 end
```

B.59 modelv2a_1.m

```
1  function [] = modelv2a_1(runtime,probM,filepath)
2  %MODDELV2A_1 Takes a runtime and a weighted selection
       matrix and generates a
3  % CSV file sampled every (default 20) seconds using
       method 2a.
4  %
5  % This is needed to compare simulation with emprical
       results from these two
6  % papers: V. Gemmetto et al. Mitigation of infectious
       diseases at school:
7  % targeted class closure vs school closure, BMC
       Infectious Diseases
8  % 201414:695 https://doi.org/10.1186/s12879-014-0695-9; R
       . Mastrandrea
9  % et al,. Contact Patterns in a High School: A Comparison
        between Data
10 % Collected Using Wearable Sensors, Contact Diaries and
       Friendship Surveys,
11 % PLoS ONE 10(9): e0136497. https://doi.org/10.1371/
       journal.pone.0136497
12 %
13 % RUNTIME specifies the total sampling period of the
       simulation
14 %
15 % PROBM specifies the edge preference matrix. The can be
       extracted from the
16 % data using the function EAPmat.m or generated using MC
       simulation using
17 % the function rndLPM.m
18
19 cut = 20;
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
% timings

while currenttime<runtime
    thislinkmatrix = linkprobmatrix;
    accept = 0;
    while accept == 0
        [ri,rj] = chooselink(thislinkmatrix);
        ID_ref = sprintf('n%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
```

```
64      end
65      IET = lognrnd(5.6901e-04,1.7957);
66      %IET should be set to the desired distribution for on
            interevent
67      % timings
68      currenttime = currenttime+IET;
69  end
70  sampleCSV2(times,nodes,runtime,cut,filepath);
71  %Passes data to sampleCSV.m for sampling and conversion
        to CSV file
72  end
```

B.60 modelv2a_1_multiple.m

```
1   function [dir_ref] = modelv2a_1_multiple(runtime,
        probMstruc)
2   %MODELV1_MULTIPLE creates as many simulations using
        method 2a as there are
3   % entries in the structure PROBMSTRUC
4   %
5   % PROBMSTRUC is a structure containing the link selection
        matrices for each
6   % of the desired simulations
7   % RUNTIME is the desired simulation time
8   % DIR_REF is the output save directory as a string
9
10  timestamp = datestr(now,'yyyymmddTHHMMSS');
11  dir_ref = ['lots/output_',timestamp];
12  dir_ref_full = ['input/',dir_ref];
13  mkdir(dir_ref_full);
14
15  fields = fieldnames(probMstruc);
16  cycles = numel(fields);
17
18  for i=1:cycles
19      filename = ['data',num2str(i),'.csv']; %Save file
            name
20      filepath = [dir_ref_full,'/',filename];
21      probM = probMstruc.(fields{i});
22      modelv2a_1(runtime,probM,filepath);
23  end
24  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
```

```
31 %  boundaries. Then maximum difference between boundaries
       is found.
32 %
33 % Copyright (C) 2014, W.Whiten (personal W.Whiten@uq.edu.
     au) BSD license
34 %  (http://opensource.org/licenses/BSD-3-Clause)
35 %
36 % See also
37 %  probKS
38
39 % divide range of a values into nx intervals for calc of
     prob difference
40 if(nargin<2)
41     nx=100;
42 end
43
44 % case for cell array of vectors
45 if(iscell(a))
46     n=length(a);
47
48     % find xx values to get probability differences
49     amin1=realmax;
50     amax1=-realmax;
51     for j=1:n
52         t=a{j};
53         amin1=min(min(t(:)),amin1);
54         amax1=max(max(t(:)),amax1);
55     end
56     xx=amin1+((amax1-amin1)/nx)*(0:nx)';
57
58     % find min and max probability values at xx values
59     yymin=realmax*ones(nx+1,1);
60     yymax=-yymin;
61     for j=1:n
62         t=sort(a{j});   % t may have more than one column
63         if(size(t,1)==1)
64             t=t';
65         end
66         if(size(t,2)==1)
67             yy=cumx2xxyy(t,xx);
68             yymin=min(yymin,yy);
69             yymax=max(yymax,yy);
70         else
71             yymin=min(yymin,cumx2xxyy(max(t,[],2),xx));
72             yymax=max(yymax,cumx2xxyy(min(t,[],2),xx));
73             % for k=1:size(t,2)
74             %     tt=cumx2xxyy(t(:,k),xx);
75             %     yymin=min(yymin,tt);
```

```
            %     yymax=max(yymax,tt);
            % end
        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=cumx2xyy(max(a,[],2),xx);  % lower limit in y
       direction
    yymax=cumx2xyy(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
```

```
13 % undirected before continuing. Since link weights will
       not affect
14 % component sizes, weighted and unweighted networks work
       equally well. You
15 % may provide a "full" or a "sparse" matrix.
16 %
17 % OUTPUTS:
18 % nComponents             INT - The number of components
       in the network.
19 % sizes                   vector<INT> - a vector of
       component sizes, sorted,
20 %   descending.
21 % members                 cell<vector<INT>> a cell array of
        vectors, each
22 %   entry of which is a membership list for that
       component, sorted,
23 %   descending by component size.
24 %
25 % Example: (uncomment and copy and paste into MATLAB
       command window)
26 % % Generate a 1000 node network adjacency matrix, A
27 % A = floor(1.0015*rand(1000,1000)); A=A+A'; A(A==2)=1; A
       (1:1001:end) = 0;
28 % % Call networkComponents function
29 % [nComponents,sizes,members] = networkComponents(A);
30 % % get the size of the largest component
31 % sizeLC = sizes(1);
32 % % get a network adjacency matrix for ONLY the largest
       component
33 % LC = A(members{1},members{1});
34
35 function [nComponents,sizes,members] = networkComponents(
       A)
36 % Number of nodes
37 N = size(A,1);
38 % Remove diagonals
39 A(1:N+1:end) = 0;
40 % make symmetric, just in case it isn't
41 A=A+A';
42 % Have we visited a particular node yet?
43 isDiscovered = zeros(N,1);
44 % Empty members cell
45 members = {};
46 % check every node
47 for n=1:N
48     if ~isDiscovered(n)
49         % started a new group so add it to members
50         members{end+1} = n;
```

```
        % 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
    end
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);

end
```

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';
```

```
elseif isnan(r)
    str = 'X';
elseif r==0
    str = '0';
else
    expon = floor(log10(abs(r)));
    coeff = r/(10^expon);
    exponstr = num2str(expon);
    coeffstr = num2str(coeff,'%.4f');
    str = strcat(coeffstr,'\times10^{',exponstr,'}');
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,ccdf_gen,'x');
plot(X_real,ccdf_real,'o');
set(gca,'XScale','log');
set(gca,'YScale','log');
xlabel(property_name);
ylabel('CCDF');
```

```
lgd = legend('Generated Data','Observed Data','Location',
    'southwest');
lgd.Title.String = ['KS Distance: ',num2str(KSdist)];
savefig(figurefilename);
hold off
close(plotthings);

end
```

B.65 probKS.m

This function was written by Whiten [211].

```
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 matirx 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
```

```
25 %
26 % Statistic used is an extension of Kolomogorov Smirnov test:
27 %  maximum difference in probability of the cumulative distributions.
28 %  Distribution of statistic is calculated by simulation.
29 %
30 % Finds boundary for cumulative probabilities for matrices first at
31 %  constant probabilities, then boundaries are converted to distributions
32 %  with given x values so that probability differences can be calculated
33 %  and if cell data used these are combined for overall upper and lower
34 %  boundaries. Then maximum difference between boundaries is found.
35 %
36 % Copyright (C) 2014, W.Whiten (personal W.Whiten@uq.edu.au) BSD license
37 %  (http://opensource.org/licenses/BSD-3-Clause)
38 %
39 % See also
40 %  multiKS
41
42 if(nargin<3)
43     nx=100;  % number of divisions for cumx2xxyy
44 end
45 if(nargin<2 || isempty(rpts))
46     rpts=10000;  % number of repeats for distribution calculation
47 end
48
49 s=multiKS(a,nx);
50
51 d=zeros(rpts,1);
52 xx=(1:nx)/(nx+1);
53 for i=1:rpts
54     if(iscell(a))
55         for j=1:length(a);
56             a{j}=rand(size(a{j}));
57         end
58         d(i)=multiKS(a,nx);
59     else
60         a=sort(rand(size(a)));   % sort individual columns of a
61         amax=max(a,[],2);     % upper limit of values
62         amin=min(a,[],2);     % lower limit of values
```

```
63
64          % convert to probabilities at same xx
65          yymax=cumx2xxyy(amax,xx);
66          yymin=cumx2xxyy(amin,xx);
67          d(i)=max(yymin-yymax);  % maximum difference of
               extreme values
68      end
69  end
70
71  pr=sum(s<d)/rpts;
72  if(nargout>1)
73      d=sort(d);
74  end
75
76  return
77  end
```

B.66 pullData.m

```
1   function dataStructure = pullData(folder,file,format)
2   %PULLDATA takes a CSV file and extracts Active Edges,
       Active Node, Node
3   % Activity Potential, Component Edges, Component Nodes,
       Global Clustering
4   % Coefficient, Interaction Times, Component Counts and
       Time Between
5   % Contacts. It outputs these into a structure.
6   %
7   % FOLDER is the path of the folder containing the data
8   % FILE is the file within this folder
9   % FORMAT is the formatting of this CSV file
10  % DATASTRUCTURE is a structure containing vectors with
       individual
11  % measurements of all of our chosen metrics
12
13  input = [folder,'/',file];
14
15  fid = fopen(input);
16  rawdata = textscan(fid,format,'Delimiter',',');
17  fclose(fid);
18
19  %==Extract and Clean Data==%
20  data = cell2mat(rawdata);
21  data(:,1) = data(:,1)-data(1,1);
22  lowestID = min(min(data(:,2)),min(data(:,3)));
23  data(:,2) = data(:,2)-lowestID+1;
24  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;

%Global Variables
num_times = size(unique(data(:,1)),1);
data_length = size(data(:,1),1);
num_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
    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);
```

```
169     thisEdges = [thisEdges zeros(1,thisPadding)];
170     rawFracEdges(m,:) = thisEdges;
171 end
172 
173 compEdgeFracs = rawFracEdges(:)';
174 dataStructure.ComponentEdges_data = compEdgeFracs;
175 
176 %Component Nodes
177 rawCompSizes = zeros(num_times,num_people);
178 
179 parfor m=1:num_times
180     thisadj = zeros(num_people);
181     current_time = (m-1)*contact_time;
182     for i=1:data_length
183         test_time = data(i,1);
184         if test_time==current_time
185             person1 = data(i,2);
186             person2 = data(i,3);
187             thisadj(person1,person2) = 1;
188             thisadj(person2,person1) = 1;
189         end
190     end
191     [~,thisCompSizes,~] = networkComponents(thisadj);
192     thisCompSizes(thisCompSizes==1)=[];
193     nodesActive = sum(thisCompSizes);
194     thisPadding = num_people - length(thisCompSizes);
195     thisCompSizes = [thisCompSizes zeros(1,thisPadding)];
196     if nodesActive == 0
197         thisCompFracs = thisCompSizes;
198     else
199         thisCompFracs = thisCompSizes/nodesActive;
200     end
201     rawCompSizes(m,:) = thisCompFracs;
202 end
203 
204 compSizes = rawCompSizes(:)';
205 dataStructure.ComponentNodes_data = compSizes;
206 
207 %Global Clustering Coefficient
208 clustering = zeros(1,num_times);
209 triangles = zeros(1,num_times);
210 
211 parfor m=1:num_times
212     thisadj = zeros(num_people);
213     current_time = (m-1)*contact_time;
214     for i=1:data_length
215         test_time = data(i,1);
216         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
    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
```

```
265 times(times_k:end) = [];
266 dataStructure.InteractionTimes_data = times;
267
268 %Number of Components
269 components = zeros(1,num_times);
270
271 parfor m=1:num_times
272     thisadj = zeros(num_people);
273     current_time = (m-1)*contact_time;
274     for i=1:data_length
275         test_time = data(i,1);
276         if test_time==current_time
277             person1 = data(i,2);
278             person2 = data(i,3);
279             thisadj(person1,person2) = 1;
280             thisadj(person2,person1) = 1;
281         end
282     end
283     [~,thisComp,~] = networkComponents(thisadj);
284     thisComp(thisComp==1)=[];
285     thisCompCount = length(thisComp);
286     components(m) = thisCompCount;
287 end
288 dataStructure.Components_data = components;
289
290 %Time Between Contacts
291 step = 20;
292 min_time = min(data(:,1));
293 max_time = max(data(:,1));
294 times = ((max_time-min_time)/step)+1;
295 data_length = size(data(:,1),1);
296 num_people = max([data(:,2); data(:,3)]);
297 rawactivity = zeros(data_length,num_people+1);
298
299 parfor i=1:data_length
300     thisrawactivity = zeros(1,num_people+1);
301     thisrawactivity(1) = data(i,1);
302     person1 = data(i,2);
303     person2 = data(i,3);
304     thisrawactivity(person1+1) = 1;
305     thisrawactivity(person2+1) = 1;
306     rawactivity(i,:) = thisrawactivity;
307 end
308
309 activity = zeros(times,num_people);
310
311 parfor i=1:times
312     currenttime = ((i-1)*step)+min_time;
```

```
313     activerows = rawactivity(rawactivity(:,1)==
            currenttime,:);
314     activerows = activerows(:,2:end);
315     thisactivity = sum(activerows,1);
316     thisactivity = (thisactivity>0);
317     activity(i,:) = thisactivity;
318 end
319
320 activity = [activity; ones(1,num_people)];
321
322 long = activity(:);
323 long = long';
324 dlong = diff([1 long 1]);
325 startIndex = find(dlong < 0);
326 endIndex = find(dlong > 0)-1;
327 nocontact = endIndex-startIndex+1;
328 nocontact = nocontact*20;
329 dataStructure.NoContactTimes_data = nocontact;
330
331 maxN=max(max(data(:,2)),max(data(:,3)));
332 activationtimes = [];
333
334 for i=1:maxN-1
335     for j=i+1:maxN
336         S1 = data(:,2)==i;
337         S2 = data(:,3)==j;
338         T1 = data(:,3)==i;
339         T2 = data(:,2)==j;
340         S12 = S1 & S2;
341         T12 = T1 & T2;
342         ST12 = S12|T12;
343         changes = diff(ST12);
344         binary = [0;changes];
345         thistimes = data(binary==1,1);
346         activationtimes = [activationtimes;thistimes];
347     end
348 end
349
350 sorted = sort(activationtimes);
351 timebetween = diff(sorted);
352
353 dataStructure.InterеventTimes_data = timebetween;
354
355 firstactivations = [];
356 for i=1:maxN-1
357     for j=i+1:maxN
358         S1 = data(:,2)==i;
359         S2 = data(:,3)==j;
```

```
360         T1 = data(:,3)==i;
361         T2 = data(:,2)==j;
362         S12 = S1 & S2;
363         T12 = T1 & T2;
364         ST12 = S12|T12;
365         currentTimes = data(ST12,1);
366         if isempty(currentTimes)
367             thisactivation = Inf;
368         else
369             thisactivation = currentTimes(1);
370         end
371         firstactivations = [firstactivations;
               thisactivation];
372     end
373 end
374 dataStructure.FirstActivationTimes_data =
       firstactivations;
375 end
```

B.67 pvals_ex.m

```
1  function [p_vals] = pvals_ex(dataLength,lambda,Statistics
       ,cut,n,gap)
2  %PVALS_WB estimates the p-values for the Exponential
       distributions
3  %
4  % DATALENGTH is the number of entries in the data
5  % LAMBDA are the parameters in the Exponential
       distribution
6  % STATISTICS is the statistics structure
7  % CUT is the number of extreme data points removed
8  % N is the precision
9  % GAP is sampling interval
10 % P_VALS is a structure containing our estimated p-values
11
12 KolD = Statistics.Kolmogorov_D;
13 CvM = Statistics.Cramer_von_Mises;
14 Kuiper = Statistics.Kuiper;
15 Watson = Statistics.Watson;
16 AD = min(Statistics.Anderson_Darling,1E99);
17 KL = Statistics.Kullback_Leibler;
18 JS = Statistics.Jensen_Shannon;
19
20 num_MC = 10^n;
21
22 KolD_stat = zeros(1,num_MC);
23 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);

end
```

B.69 pvals_gp.m

```
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;
```

```
19
20 num_MC = 10^n;
21
22 KolD_stat = zeros(1,num_MC);
23 CvM_stat = zeros(1,num_MC);
24 Kuiper_stat = zeros(1,num_MC);
25 Watson_stat = zeros(1,num_MC);
26 AD_stat = zeros(1,num_MC);
27 KL_stat = zeros(1,num_MC);
28 JS_stat = zeros(1,num_MC);
29
30 parfor i=1:num_MC
31     data = gprnd(k,sigma,theta,dataLength,1);
32     data = sort(data);
33     if cut>0
34         data(end-cut+1:end) = [];
35     end
36     CDF = gpcdf(data,k,sigma,theta);
37     PDF = gppdf(data,k,sigma,theta);
38     thisfit = testStatistics(data,CDF,PDF,gap);
39
40     KolD_stat(i) = thisfit.Kolmogorov_D;
41     CvM_stat(i) = thisfit.Cramer_von_Mises;
42     Kuiper_stat(i) = thisfit.Kuiper;
43     Watson_stat(i) = thisfit.Watson;
44     AD_stat(i) = thisfit.Anderson_Darling;
45     KL_stat(i) = thisfit.Kullback_Leibler;
46     JS_stat(i) = thisfit.Jensen_Shannon;
47 end
48
49 [F_KolD,X_KolD] = ecdf(KolD_stat);
50 [F_CvM,X_CvM] = ecdf(CvM_stat);
51 [F_Kuiper,X_Kuiper] = ecdf(Kuiper_stat);
52 [F_Watson,X_Watson] = ecdf(Watson_stat);
53 [F_AD,X_AD] = ecdf(AD_stat);
54 [F_KL,X_KL] = ecdf(KL_stat);
55 [F_JS,X_JS] = ecdf(JS_stat);
56
57 clearvars KolD_stat CvM_stat Kuiper_stat Watson_stat
     AD_stat KL_stat JS_stat
58
59 X_KolD = [-1E99;X_KolD(2:end);1E99];
60 X_CvM = [-1E99;X_CvM(2:end);1E99];
61 X_Kuiper = [-1E99;X_Kuiper(2:end);1E99];
62 X_Watson = [-1E99;X_Watson(2:end);1E99];
63 X_AD = [-1E99;X_AD(2:end);1E99];
64 X_KL = [-1E99;X_KL(2:end);1E99];
65 X_JS = [-1E99;X_JS(2:end);1E99];
```

```
66
67 F_KolD = [0;F_KolD(2:end);1];
68 F_CvM = [0;F_CvM(2:end);1];
69 F_Kuiper = [0;F_Kuiper(2:end);1];
70 F_Watson = [0;F_Watson(2:end);1];
71 F_AD = [0;F_AD(2:end);1];
72 F_KL = [0;F_KL(2:end);1];
73 F_JS = [0;F_JS(2:end);1];
74
75 p_KolD = 1-interp1(X_KolD,F_KolD,KolD,'next');
76 p_CvM = 1-interp1(X_CvM,F_CvM,CvM,'next');
77 p_Kuiper = 1-interp1(X_Kuiper,F_Kuiper,Kuiper,'next');
78 p_Watson = 1-interp1(X_Watson,F_Watson,Watson,'next');
79 p_AD = 1-interp1(X_AD,F_AD,AD,'next');
80 p_KL = 1-interp1(X_KL,F_KL,KL,'next');
81 p_JS = 1-interp1(X_JS,F_JS,JS,'next');
82
83 p_vals = struct('Kolmogorov_D',p_KolD,...
84                 'Cramer_von_Mises',p_CvM,...
85                 'Kuiper',p_Kuiper,...
86                 'Watson',p_Watson,...
87                 'Anderson_Darling',p_AD,...
88                 'Kullback_Leibler',p_KL,...
89                 'Jensen_Shannon',p_JS);
90
91 end
```

B.70 pvals_ln.m

```
1  function [p_vals] = pvals_ln(dataLength,mu,sigma,
       Statistics,cut,n,gap)
2  %PVALS_WB estimates the p-values for the Log-Normal
       distributions
3  %
4  % DATALENGTH is the number of entries in the data
5  % MU,SIGMA are the parameter in the Log-Normal
       distribution
6  % STATISTICS is the statistics structure
7  % CUT is the number of extreme data points removed
8  % N is the precision
9  % GAP is sampling interval
10 % P_VALS is a structure containing our estimated p-values
11
12 KolD = Statistics.Kolmogorov_D;
13 CvM = Statistics.Cramer_von_Mises;
14 Kuiper = Statistics.Kuiper;
15 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];
```

```
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.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,'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.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,'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.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);
```

```
55 [F_JS,X_JS] = ecdf(JS_stat);
56
57 clearvars KolD_stat CvM_stat Kuiper_stat Watson_stat
      AD_stat KL_stat JS_stat
58
59 X_KolD = [-1E99;X_KolD(2:end);1E99];
60 X_CvM = [-1E99;X_CvM(2:end);1E99];
61 X_Kuiper = [-1E99;X_Kuiper(2:end);1E99];
62 X_Watson = [-1E99;X_Watson(2:end);1E99];
63 X_AD = [-1E99;X_AD(2:end);1E99];
64 X_KL = [-1E99;X_KL(2:end);1E99];
65 X_JS = [-1E99;X_JS(2:end);1E99];
66
67 F_KolD = [0;F_KolD(2:end);1];
68 F_CvM = [0;F_CvM(2:end);1];
69 F_Kuiper = [0;F_Kuiper(2:end);1];
70 F_Watson = [0;F_Watson(2:end);1];
71 F_AD = [0;F_AD(2:end);1];
72 F_KL = [0;F_KL(2:end);1];
73 F_JS = [0;F_JS(2:end);1];
74
75 p_KolD = 1-interp1(X_KolD,F_KolD,KolD,'next');
76 p_CvM = 1-interp1(X_CvM,F_CvM,CvM,'next');
77 p_Kuiper = 1-interp1(X_Kuiper,F_Kuiper,Kuiper,'next');
78 p_Watson = 1-interp1(X_Watson,F_Watson,Watson,'next');
79 p_AD = 1-interp1(X_AD,F_AD,AD,'next');
80 p_KL = 1-interp1(X_KL,F_KL,KL,'next');
81 p_JS = 1-interp1(X_JS,F_JS,JS,'next');
82
83 p_vals = struct('Kolmogorov_D',p_KolD,...
84                 'Cramer_von_Mises',p_CvM,...
85                 'Kuiper',p_Kuiper,...
86                 'Watson',p_Watson,...
87                 'Anderson_Darling',p_AD,...
88                 'Kullback_Leibler',p_KL,...
89                 'Jensen_Shannon',p_JS);
90
91 end
```

B.74 rndLPM.m

```
1 function lpm = rndLPM(nodes)
2 %RNDLPM returns a symmetric square randomly weighted
     selection matrix for
3 % the given number of nodes
4 %
5 % 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 trianglular 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(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)
```

```
%SCALEDMARGINALS 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 containg 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));
end
```

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)];
        pparas = 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

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

```
function [sigma] = sigma_for_mu_and_mean(m,mu)
%SIGMA_FOR_MU_AND_MEAN Calculates the variance of the
   normal distribution
% with mean MU of the associated log-normal distribuition
    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

```
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 Kullbeck-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);
end
```

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 comparitive CDF at these values
% ZPRIME is the comparitive 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);
upperecdf = fullecdf(2:n+1);
middlecdf = (lowerecdf+upperecdf)/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-upperecdf);
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,...
```

```
                    'Cramer_von_Mises',Wsq,...
                    'Kuiper',V,...
                    'Watson',Usq,...
                    'Anderson_Darling',Asq,...
                    'Kullback_Leibler',KL,...
                    'Jensen_Shannon',JS);
end
```

B.82 triangleClosed.m

```
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';

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
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
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,ondata,offdata)
%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);
```

```
19
20 ontimes = struct();
21 offtimes = struct();
22
23 for i=1:nodes-1
24     for j=i+1:nodes
25         init = initial(i,j);
26         currenttime = startthings(i,j);
27         if init == 0
28             if startthings(i,j)<runtime
29                 thisoff = [startthings(i,j)];
30             else
31                 thisoff = [];
32             end
33             thison = [];
34             while currenttime<runtime
35                 thisoffduration = emprand(offdata);
36                 switch_on = currenttime+thisoffduration;
37                 thisonduration = emprand(ondata);
38                 switch_off = switch_on+thisonduration;
39                 if switch_on<runtime
40                     thison = [thison,switch_on];
41                     if switch_off<runtime
42                         thisoff = [thisoff,switch_off];
43                     else
44                         thisoff = [thisoff,runtime];
45                     end
46                 else
47                     thison = [thison,runtime];
48                 end
49                 currenttime = switch_off;
50             end
51         elseif init == 1
52             thisoff = [];
53             if startthings(i,j)<runtime
54                 thison = [startthings(i,j)];
55             else
56                 thison = [];
57             end
58             while currenttime<runtime
59                 thisonduration = emprand(ondata);
60                 switch_off = currenttime+thisonduration;
61                 thisoffduration = emprand(offdata);
62                 switch_on = switch_off+thisoffduration;
63                 if switch_off<runtime
64                     thisoff = [thisoff,switch_off];
65                     if switch_on<runtime
66                         thison = [thison,switch_on];
```

```
                    else
                        thison = [thison,runtime];
                    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) = [];
        thisoff(thisoff==0) = [];
        thison(thison==runtime) = [];
        thisoff(thisoff>runtime) = [];
        thison(thison>runtime) = [];
        if firston > firstoff
            thison = [switchon(i,j),thison];
        end
        ID_ref = sprintf('n%d_n%d', i,j);
        ontimes.(ID_ref) = thison;
        offtimes.(ID_ref) = thisoff;
    end
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
    method 2a
%
% 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 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('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 = 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('n%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+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
```

```
56      % timings
57      currenttime = currenttime+IET;
58  end
59  sampleCSV2(times,nodes,runtime,cut,filepath);
60  %Passes data to sampleCSV.m for sampling and conversion
       to CSV file
61  end
```

B.85 validate2b.m

```
1   function [] = validate2b(runtime,probM,filepath,IETdata,
       onData)
2   %VALIDATE2A produces a simulation validation run using
       method 2b
3   %
4   % RUNTIME is the simulation length in seconds
5   % PROBM is the extracted link preference selection matrix
6   % FILEPATH is the save directory given as a string
7   % IETTIMES is the interevent times from the underlying
       data as a vector
8   % ONTIMES is the on-times from the underlying data as a
       vector
9
10  cut = 20;
11  %CUT is the sampling interval in seconds (DEFAULT: 20)
12  nodes = size(probM,1);
13  linkprobmatrix = probM;
14
15  initialoff = zeros(nodes);
16  %INITIALOFF is the initial state for the network given as
       an adjacency
17  % matrix
18
19  times = struct();
20  %TIMES will end up having entries labeled nX_nY with
       entries showing the on
21  % and off times for the link X-Y as a single vector.
22  for i=1:nodes-1
23      for j=i+1:nodes
24          ID_ref = sprintf('n%d_n%d',i,j);
25          if initialoff(i,j)>0
26              times.(ID_ref) = [0,initialoff(i,j)];
27          else
28              times.(ID_ref) = [];
29          end
30      end
31  end
```

```
32
33 currenttime = emprand(IETdata);
34 %CURRENTTIME should be set to the desired distribution
      for on interevent
35 % timings
36
37 triangleActivations = 0.0556;
38 estFailure = 0.1316;
39 adjustedThreshold = triangleActivations/(1-estFailure);
40
41 while currenttime<runtime
42     flip = unifrnd(0,1);
43     if flip<adjustedThreshold
44         [~,m_lpm] = extractTriangles(times,nodes,
              currenttime,linkprobmatrix);
45         if sum(sum(m_lpm))~=0
46             thislinkmatrix = m_lpm;
47         else
48             thislinkmatrix = linkprobmatrix;
49         end
50     else
51         thislinkmatrix = linkprobmatrix;
52     end
53     accept = 0;
54     cycles = 0;
55     while accept == 0
56         [ri,rj] = chooselink(thislinkmatrix);
57         ID_ref = sprintf('n%d_n%d',ri,rj);
58         vec = times.(ID_ref);
59         if isempty(vec)||currenttime>vec(end)
60             %Ensure link is not already active
61             accept = 1;
62             ontime = currenttime;
63             offtime = currenttime+emprand(onData);
64             vec = [vec,ontime,offtime];
65             %Append to entry in TIMES
66             times.(ID_ref) = vec;
67         end
68         if cycles>100
69             thislinkmatrix = linkprobmatrix;
70         else
71             cycles = cycles+1;
72         end
73     end
74     IET = emprand(IETdata);
75     %IET should be set to the desired distribution for on
          interevent
76     % timings
```

```
77      currenttime = currenttime+IET;
78  end
79  sampleCSV2(times,nodes,runtime,cut,filepath);
80  %Passes data to sampleCSV.m for sampling and conversion
       to CSV file
81  end
```

B.86 validation.m

```
1   function [] = validation(folder,count,timelength)
2   %VALIDATION creates and saves a latex ready file
       containing validation
3   % counts for each model and metric
4   %
5   % FOLDER is the location of data, given as a string
6   % COUNT is the number of validation samples to generate
7   % TIMELENGTH is the simulation time for the validation
8
9   timestamp = datestr(now,'yyyymmddTHHMMSS');
10  dir_ref = ['input/output_',timestamp];
11  mkdir(dir_ref);
12
13  iF = ['input/',folder];
14  toExtract = [iF,'/*.csv'];
15
16  fileData = dir(toExtract);
17  fileList = {fileData.name};
18  fileList = fileList(~contains(fileList,'._'));
19
20  accept5matrix = zeros(10,3);
21
22  for i=1:length(fileList)
23      currentFile = fileList{i};
24      currentFilepath = [iF,'/',currentFile];
25      currentClean = strrep(currentFile, '.', '');
26      currentClean = strrep(currentClean, '-', '');
27      currentParent = [dir_ref,'/validation'];
28      currentFolder = [currentParent,'/',currentClean];
29      currentFolderR = [currentFolder,'/original'];
30      currentFolderM1 = [currentFolder,'/model1'];
31      currentFolderM2a = [currentFolder,'/model2a'];
32      currentFolderM2b = [currentFolder,'/model2b'];
33      mkdir(currentFolderR);
34      mkdir(currentFolderM1);
35      mkdir(currentFolderM2a);
36      mkdir(currentFolderM2b);
37      copyfile(currentFilepath,currentFolderR);
```

```
38      currentData = pullData(iF,currentFile,'%f %f %f %*s
           %*s');
39      currentInitial = currentData.
           FirstActivationTimes_data;
40      currentInitial(currentInitial==Inf) = timelength;
41      currentOnTimes = currentData.InteractionTimes_data;
42      currentOffTimes = currentData.NoContactTimes_data;
43      currentIETimes = currentData.IntereventTimes_data;
44      currentStudents = currentData.NumberStudents_data;
45      currentMatrix = EAP_matrix(folder,currentFile);
46      for j=1:count
47          outputfile = ['run_',num2str(j),'.csv'];
48          fp1 = [currentFolderM1,'/',outputfile];
49          fp2a = [currentFolderM2a,'/',outputfile];
50          fp2b = [currentFolderM2b,'/',outputfile];
51          validate1(currentStudents,timelength,fp1,
               currentInitial,currentOnTimes,currentOffTimes)
               ;
52          validate2a(timelength,currentMatrix,fp2a,
               currentIETimes,currentOnTimes);
53          validate2b(timelength,currentMatrix,fp2b,
               currentIETimes,currentOnTimes);
54      end
55      dist1 = calculateDistance(currentFolderM1(7:end),
           currentFolderR(7:end));
56      dist2a = calculateDistance(currentFolderM2a(7:end),
           currentFolderR(7:end));
57      dist2b = calculateDistance(currentFolderM2b(7:end),
           currentFolderR(7:end));
58      currentAccept5 = zeros(10,3);
59      currentAccept5(1,1) = dist1.ActiveLinks.accept5;
60      currentAccept5(2,1) = dist1.OnTimes.accept5;
61      currentAccept5(3,1) = dist1.ActivityPot.accept5;
62      currentAccept5(4,1) = dist1.OffTimes.accept5;
63      currentAccept5(5,1) = dist1.ActiveNodes.accept5;
64      currentAccept5(6,1) = dist1.CompCount.accept5;
65      currentAccept5(7,1) = dist1.GCC.accept5;
66      currentAccept5(8,1) = dist1.CompNodes.accept5;
67      currentAccept5(9,1) = dist1.CompEdges.accept5;
68      currentAccept5(10,1) = dist1.TriangleCount.accept5;
69      currentAccept5(1,2) = dist2a.ActiveLinks.accept5;
70      currentAccept5(2,2) = dist2a.OnTimes.accept5;
71      currentAccept5(3,2) = dist2a.ActivityPot.accept5;
72      currentAccept5(4,2) = dist2a.OffTimes.accept5;
73      currentAccept5(5,2) = dist2a.ActiveNodes.accept5;
74      currentAccept5(6,2) = dist2a.CompCount.accept5;
75      currentAccept5(7,2) = dist2a.GCC.accept5;
76      currentAccept5(8,2) = dist2a.CompNodes.accept5;
```

```
77      currentAccept5(9,2) = dist2a.CompEdges.accept5;
78      currentAccept5(10,2) = dist2a.TriangleCount.accept5;
79      currentAccept5(1,3) = dist2b.ActiveLinks.accept5;
80      currentAccept5(2,3) = dist2b.OnTimes.accept5;
81      currentAccept5(3,3) = dist2b.ActivityPot.accept5;
82      currentAccept5(4,3) = dist2b.OffTimes.accept5;
83      currentAccept5(5,3) = dist2b.ActiveNodes.accept5;
84      currentAccept5(6,3) = dist2b.CompCount.accept5;
85      currentAccept5(7,3) = dist2b.GCC.accept5;
86      currentAccept5(8,3) = dist2b.CompNodes.accept5;
87      currentAccept5(9,3) = dist2b.CompEdges.accept5;
88      currentAccept5(10,3) = dist2b.TriangleCount.accept5;
89      accept5matrix = accept5matrix+currentAccept5;
90  end
91
92  lines = 13;
93  tobuild = cell(lines,1);
94
95  tobuild{01} = '\begin{tabular}{l|c|c|c|} \cline{2-4}';
96  tobuild{02} = '& \textbf{Model 1} & \textbf{Model 2a} & \
        textbf{Model 2b}\\ \hline';
97  tobuild{03} = ['\multicolumn{1}{|l|}{\textbf{Active Links
        }} & $',num2str(accept5matrix(1,1)),'$ & $',num2str(
        accept5matrix(1,2)),'$ & $',num2str(accept5matrix(1,3)
        ),'$\\ \hline'];
98  tobuild{04} = ['\multicolumn{1}{|l|}{\textbf{Active Nodes
        }} & $',num2str(accept5matrix(2,1)),'$ & $',num2str(
        accept5matrix(2,2)),'$ & $',num2str(accept5matrix(2,3)
        ),'$\\ \hline'];
99  tobuild{05} = ['\multicolumn{1}{|l|}{\textbf{Node
        Activity Potential}} & $',num2str(accept5matrix(3,1)),
        '$ & $',num2str(accept5matrix(3,2)),'$ & $',num2str(
        accept5matrix(3,3)),'$\\ \hline'];
100 tobuild{06} = ['\multicolumn{1}{|l|}{\textbf{Global
        Clustering Coefficient}} & $',num2str(accept5matrix
        (4,1)),'$ & $',num2str(accept5matrix(4,2)),'$ & $',
        num2str(accept5matrix(4,3)),'$\\ \hline'];
101 tobuild{07} = ['\multicolumn{1}{|l|}{\textbf{Interaction
        Time}} & $',num2str(accept5matrix(5,1)),'$ & $',
        num2str(accept5matrix(5,2)),'$ & $',num2str(
        accept5matrix(5,3)),'$\\ \hline'];
102 tobuild{08} = ['\multicolumn{1}{|l|}{\textbf{Time Between
         Contacts}} & $',num2str(accept5matrix(6,1)),'$ & $',
        num2str(accept5matrix(6,2)),'$ & $',num2str(
        accept5matrix(6,3)),'$\\ \hline'];
103 tobuild{09} = ['\multicolumn{1}{|l|}{\textbf{Component
        Count}} & $',num2str(accept5matrix(7,1)),'$ & $',
        num2str(accept5matrix(7,2)),'$ & $',num2str(
```

```
   accept5matrix(7,3)),'$\\ \hline'];
tobuild{10} = ['\multicolumn{1}{|l|}{\textbf{Links per
   Component}} & $',num2str(accept5matrix(8,1)),'$ & $',
   num2str(accept5matrix(8,2)),'$ & $',num2str(
   accept5matrix(8,3)),'$\\ \hline'];
tobuild{11} = ['\multicolumn{1}{|l|}{\textbf{Nodes per
   Component}} & $',num2str(accept5matrix(9,1)),'$ & $',
   num2str(accept5matrix(9,2)),'$ & $',num2str(
   accept5matrix(9,3)),'$\\ \hline'];
tobuild{12} = ['\multicolumn{1}{|l|}{\textbf{Triangle
   Count}} & $',num2str(accept5matrix(10,1)),'$ & $',
   num2str(accept5matrix(10,2)),'$ & $',num2str(
   accept5matrix(10,3)),'$\\ \hline'];
tobuild{13} = '\end{tabular}';

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

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

List of Figures

List of Tables

Index of Names

Index of Subjects