Towards a real-time predictions using Emulators of Agent-Based Models

MK,JW,ED,NM

November 22, 2019

Abstract

1 Introduction

Agent-based modelling is a field that excels in its ability to simulate complex systems (Bonabeau 2002). Instead of deriving aggregated equations of system dynamics, an Agent-Based Model (ABM) encapsulates system-wide characteristics from the behaviours and interactions of individual agents, for instance, humans, animals or vehicles. ABM has traditionally been used to understand the dynamics of a system, and has emerged as an important tool for evaluations of 'what-if' scenarios in planning of urban traffic (Balmer et al. 2009), emergency evacuations (Schoenharl and Madey 2011) and smart cities (Batty 2007). However, ABM has rarely been used in real-time applications to take advantage of their explanatory power and flexibility. Such modelling may be of great importance in practice. For instance, ABMs can model a crowd of people and provide indexes of the level of crowdiness and speed in real time, which is helpful for crowd management and emergency evacuations. Similar applications can be found for traffic flow modelling to deal with traffic congestion.

There are three major technological barriers to the development of ABMs for real-time applications. First, agent-based models are often developed and calibrated using historical data. When operating in real time, these models often diverge from reality, as they often do not 'know' the current state of the reality. There is currently no systematic method to incorporate new data into an object-oriented ABM that often has many unstructured variables. For instance, in order to update the location of an agent in an ABM, modellers will have to locate the agent's object first, and then update its 'location' parameters. There are some recent studies on updating ABMs in real time, but are still limited in scale and complexity (Wang and Hu 2015; Ward et al. 2016). Second, often being used to model complex systems, ABMs are necessarily complex and computationally expensive. For instance, MATSim Singapore, a transportation ABM of Singapore that aims to model people's movements across multiple transportation modes in Singapore, would take 2 days to implement a scenario even in a cluster of 4 super computers (Anda 2017). Third, available data in real time are often highly aggregated, which makes it very difficult to implement individual-based models such as ABMs.

This paper aims to develop a framework that would enable researchers and practitioners to benefit from agent-based modelling in real time. The idea is to use an *emulator*, also referred as a *surrogate model* or *meta-model*, which is a stand-alone model, to directly deal with aggregated input data and aim to produce similar outputs to an agent-based model in a fraction of its computation time. The emulator is a fundamental concept in the physical sciences and is commonly used in fields such as climate modelling (Krasnopolsky et al. 2005) and biogeochemistry (Conti and O'Hagan 2010) where simulation models are often too costly for real-time implementation. Recent efforts have shown benefits of emulators for complex simulation models (Rasouli and Timmermans 2013; Krasnopolsky et al. 2005), and have also extended to dynamic (Conti et al. 2009; Conti and O'Hagan 2010) and stochastic simulation models (Moutoussamy et al. 2015; Baker et al. 2019). There have been a very limited number of attempts in the state-of-the-art for emulating ABMs (Rasouli and Timmermans 2013; Bijak et al. 2013; Heard 2014; Hilton 2017;

Oyebamiji et al. 2017). Among them, the studies that focus on emulating a *real-time* ABMs are very limited, because of two major challenges. First, ABMs usually have a very high number of dimensions. Large scale ABMs are composed of thousands to millions of agents, each with multiple possible actions. Emulating an ABM would most likely require solving a high dimensional problem, as the system being modelled by the ABM for real-time applications is evolving over time. Second, emulating an ABM is also very challenging because the input and output data in ABMs are often unstructured. Instead of a numerical format, which can be easily represented by any statistical regression models, many ABMs have input and output variables being text or even objects in object-oriented programming languages.

This paper will be one of the first studies to develop a framework towards real-time implementation of ABMs. In this paper, we will describe the developed framework and compare several statistical and machine learning emulators to represent an ABM in real time.

The remainder of this manuscript is structured as follows. Section 2 reviews the literature of emulators for computer codes and ABMs. Section 3 outlines the developed agent-based models and emulation methods in this research. Section 4 shows the numerical experiment in several case studies, to show the benefits of the proposed method in a real-time application, and compares several methods to each other. Finally, Section **??** concludes this study and shows several directions for future research.

2 Literature Review

An emulator is a statistical representation of a large scale model or simulator, where the simulator is considered as an unknown function (Bastos and O'Hagan 2009). The emulator then depicts the relationship between the input and output variables of the simulator (Rasouli and Timmermans 2013). This section reviews the existing approaches in emulators for complex simulators, including Agent-Based Models (ABMs).

2.1 Analytical emulators

Analytical emulators of large scale simulators aim to find a parametric smoothing function depicting the relationship between input and output variables. Rasouli and Timmermans (2013) aims to emulate the daily distance travelled per person in a microsimulation model and an ABM of traffic flow. Regression was the adopted approach, in particular main effects plus first order interaction effects regression model. The authors also explored the impact of the number of simulation runs on the performance of the emulator. The results suggest that the accuracy of the emulator increases with the number of simulation runs. Lafuerza et al. (2016) developed an analytically tractable emulator of an ABM of social interaction, which allows mathematical analysis to be performed. The emulator clarifies the elements of the ABM, including social influence, heterogeneity and noise.

Analytical emulators are tractable and have a nice parametric formulation, that helps understanding the relationship between the input and output variables of the main simulator. However, a parametric formulation may be limited to the current dynamics of the system, and may need to be revised as the system under study changes over time.

2.2 Meta-modelling approaches

Another approach to emulation of complex simulator is to adopt a statistical or machine learning approach to learn the mapping between input and output variables. Emulators such as these are also referred as 'meta-models' or 'surrogate' models.

There are a significant number of studies in literature developing statistical emulators of complex computer simulators. Gaussian Process (also referred Kriging in some other literature) is among the most popular techniques for developing an emulator for computer codes (Bastos and O'Hagan 2009). Compared with other methods to learn a mapping between input and output variables, that have also been used

as surrogate models, such as Neural Network (Shrestha et al. 2009), Gaussian Process (GP) is flexible, can fit complex surfaces with limited number of data points. More importantly, GPs do not just offer an estimate, such as in Neural Networks, but also provide error statistics for the quality of prediction. (Oakley and O'Hagan 2002) proposed a univariate Gaussian process to emulate computationally large simulators with uncertain inputs. The univariate Gaussian process emulator was extended to a multivariate emulator in several studies, such as Higdon et al. (2008), to allow more flexibility in dealing with the size and multivariate nature of the data. Principal component analysis (PCA) was also used to reduce the dimensionality of the problem and speed up the computation required for the experiment. Bastos and O'Hagan (2009) develops several numerical and graphical diagnostics to validate and evaluate the suitability of Gaussian Process emulator as a surrogate model for a simulator.

The literature has recently extended to emulators for dynamic simulators (Conti et al. 2009; Conti and O'Hagan 2010) and stochastic simulators (Moutoussamy et al. 2015; Baker et al. 2019). Stochastic simulators model systems that behave differently under the same conditions. In other words, given the same inputs, stochastic simulators provide outputs that are random for each simulation. Moutoussamy et al. (2015) developed two emulators of the probability density function of the outputs from a stochastic simulator: a kernel regression and a decomposition of the density of outputs. Baker et al. (2019) diagnosed the complexities and challenges when developing an emulator for stochastic computer codes using both deterministic and stochastic tests. Dynamic simulator models systems that are evolving over time, usually over fixed time steps. The simulator iterates step-by-step from an initial system state to a final state. (Bhattacharya 2007) was one of the first studies that aims to develop emulators for dynamic simulators. The authors introduced a grid within the range of outputs from the simulator, where the entire dynamic outputs are expected. Conti et al. (2009) discussed two approaches for emulating a dynamic simulator: (1) emulate a complete multi-step run of the simulator from the first to the last time step, and (2) emulate only a single-step simulator, then run the emulator iteratively. The authors explored the second approach, by using a GP to emulate a dynamic simulator. Conti and O'Hagan (2010) extends the work in Conti et al. (2009) by developing multi-output, single-output and time input emulators for the problem of emulating dynamic computer simulators. Shrestha et al. (2009) developed a Neural Network to emulate a computationally expensive Monte Carlo simulation for the analysis of model parametric uncertainty. The authors suggested that the proposed method can be also useful in real time applications, when it is too expensive to implement a large number of simulations for hydrological models. Farah et al. (2014) developed a Bayesian framework to calibrate the parameters of a large dynamic epidemic model. The authors adopted GP to emulate the outputs from a deterministic simulator and dynamic linear model to model their evolution over time.

2.3 Emulators of ABMs

The described emulators can emulate any computer codes, including ABMs (Lee et al. 2015). Gaussian Process (GP) models have been used in literature of ABMs for sensitivity analysis and model calibration (Dosi et al. 2018; Bijak et al. 2013; Heard 2014; Hilton 2017). Bijak et al. (2013) presented a Semi-Artificial Model of Population, which is a fusion of demographic micro-simulation and agent-based models in the problem of modelling population. A GP emulator is developed to analyse the impacts of seleted parameters on population size and share of married agents. Dosi et al. (2018) analysed policy impacts using ABMs, where GPs were used as part of their sensitivity analysis on key variables and parameters. The benefits of GPs in the model calibration and statistical inference problem of ABMs have also been explored in Heard (2014), where GP emulators combine observed data with outputs from ABM simulations to fit and calibrate Gaussian-process approximations. In Hilton (2017), GP emulator was used to quantify the uncertainty in the outputs of ABMs and also to calibrate an ABM against empirical observations. Lamperti et al. (2018) developed an extreme gradient boosted decision tree algorithm (XGBoost) to calibrate and explore the parameter space of ABMs. Recently, Oyebamiji et al. (2017) and Oyebamiji et al. (2019) developed GP regression models to emulate dynamic and stochastic individual-based models. The

GP model is extended by combining with multivariate dynamic linear models to calibrate the individual-based simulator. The individual-based simulator was a stochastic and dynamic simulator of microbial communities.

Talks about the challenges in using Emulators for real-time applications here: Now if we want to use these existing emulators of ABMs in real-time, what would be the problem?

The emulators that aim for representing real-time ABMs are challenging, and so far very limited in literature. This is challenging because ABMs are often unstructured and high-dimension, as described in the Introduction. This paper will be one of the first studies in literature that aim to emulate ABMs for real-time applications.

3 Methodology

3.1 A framework towards real-time implications of agent-based models

Consider a model f with input x from some input space $\mathcal{X}^p \subseteq \mathbb{R}$ and output $y \in \mathbb{R}^q$, such that $y = f(x)$. Here p and q are the dimension of the input and output vector, respectively. We are particularly interested in the case where $f(\cdot)$ is an ABM. For instance, for a simulation of a train station, while x can be the number of people walking across a corridor, the output y can be the average level of crowdedness or walking speed.

In real time, researchers and policy makers may be interested in the current or near future output y. If the ABM $f(\cdot)$ can be run very efficiently, then the real-time applications of $f(\cdot)$ is a straight forward run of the ABM to process the input x to get some expected real-time output $\bar{y}$. However, most ABMs are complex and computationally expensive due to the heterogeneous behaviours and interactions of multiple agents. Some complex ABMs, such as the MATSIM Singapore, takes 2 days to complete a simulation scenario, even on a cluster of 4 supercomputers (Anda 2017). It is then very challenging to find sufficient computing power to run ABMs fast enough for real-time applications.

It is also currently very challenging to let ABM represent the current system conditions in real time and predict the near-future state of the system. There is currently no systematic method to incorporate real-time data into ABMs (Ward et al. 2016). Even if there are recent studies that aim to incorporate real-time data into very simple ABMs on a limited scale (Ward et al. 2016; Wang and Hu 2015), it is still very challenging to update an unstructured ABM with new data. To model an individual (e.g. a person) in the real world, a representative agent must be created in the simulation environment, with similar characteristics with the real individual. In real time, to update the states of that individual (e.g. location, speed or direction), the real individual must be tracked. The associated agent must then be located, and its parameters must be updated accordingly. This step is not just complex and expensive, but also require tracking of individuals. Most of sensors that can give real-time data are still aggregated, e.g. they provide the counts of people walking pass a gate rather the trajectories of individuals. This is because of the current limitation in available sensors, data storage and especially the concerns regarding the recent General Data Protection Regulation (GDPR).

This paper aims to solve both of these problems by proposing a novel bi-level framework to enable researchers and practitioners to benefit from the modelling capabilities of ABMs in real time. The idea is to adapt a central concept in physical sciences, called *Emulators* to represent a complex Agent-based model in real time and provide similar outputs directly from aggregated input from sensors. We develop and evaluate this framework on a simple case study that somewhat similar to a real case where the framework would be useful in practice. Figure 1 shows a corridor with a door in and a door out, similar to a typical pedestrian corridor at a train station or a shopping centre. The corridor is also narrowing down, showing at congestion management is necessary in high demand because this reduction in space may lead to overcrowding. We assume a set of pedestrian counters that are equally spaced along the corridor to provide the number of pedestrian walk through the sensor within a time period. This is fair assumption in practice because pedestrian, people or footfall counters are widely available in many public

places.

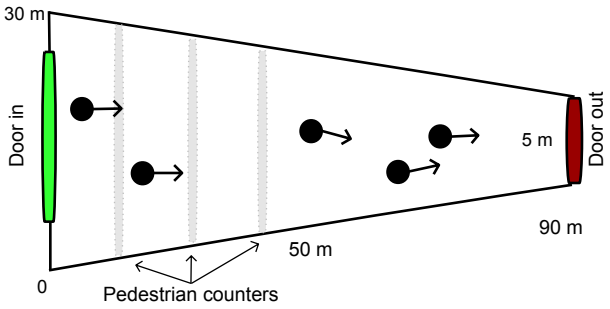


Figure 1: Case study: A narrowing corridor

We plan to use an Agent-based model to simulate pedestrians walking through the corridor in Figure 1. This is a typical research problem where ABMs excel in, because problem involve the simulation of individuals with heterogeneous behaviour and with system states (e.g. congestion) emerge from the interactions between individuals. We will adopt the Helbing's Social Force model (Helbing and Molnár 1995; Helbing et al. 2000, 2005), a very popular model of pedestrian simulation and crowd dynamics to develop this ABM.

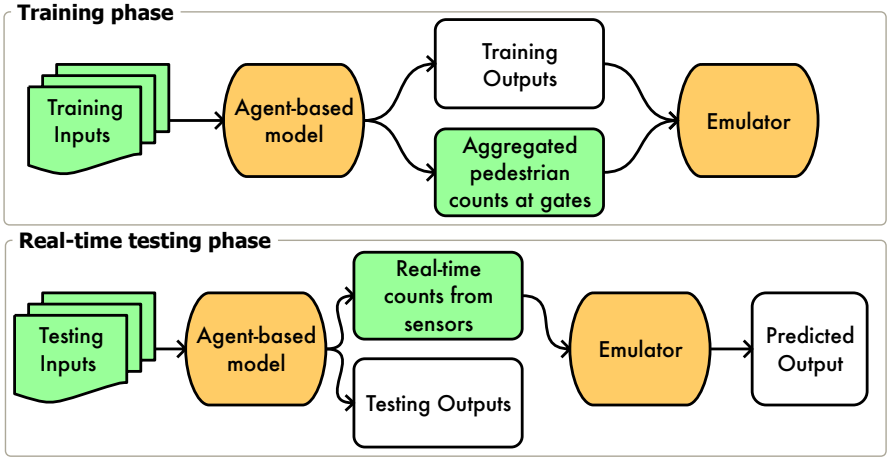


Figure 2: Study framework

The bi-phase framework is illustrated in Figure 2. The final phase, *Real-time testing phase*, involves the use of an Emulator to process real-time pedestrian counts from pedestrian counters into *Predicted Outputs*. The Predicted output here is a proxy for a future level of crowdiness in the corridor. Practitioners may use this value to decide a timely and suitable crowd management policies before congestion occurs in the corridor.

In the first phase of this framework, some *Training inputs* are fed into the Agent-based model to provide *Training outputs* and *Aggregated pedestrian counts from sensors*. Both of these are fed into an *Emulator* that we will develop, where the Training outputs are similar to the Predicted output, and the Aggregated pedestrian counts from sensors are the inputs of the Emulator. The Emulator, in this set up, is similar to a Regression model in statistic and machine learning.

The second phase of this framework follows a pseudo-truth approach, where the Agent-based model is again used to generate a new set of data (*Testing outputs* and *Real-time counts from sensors*). Because we will develop a stochastic Agent-based model, these data are different to the data that have been used to train the Emulator. The Emulator will be used to provide *Predicted Output*, that we will compare

to the Testing Output to evaluate the accuracy of the Emulator. We will also give the Emulator some completely 'out-of-sample' unseen data in the Real-time counts from sensors to see if the Emulator is able to generalise and provide good Predicted Output or not.

In the following, we will describe the developed stochastic Agent-based model and the Emulators.

3.2 Agent-based modelling phase: the Social Force Model

This section first describes the developed agent-based model to represent the pedestrian dynamics in several simulation environments. Each agent in this model is a pedestrian who is walking from a predefined entrance to an exit within a simulation environment. The movements of each agent inside a human crowd is modelled using the Social Force Model (Helbing and Molnár 1995; Helbing et al. 2000), a very popular model of pedestrian dynamics. The Social Force Model (SFM) models pedestrian movements in interaction with each other. Each of N pedestrians i with mass m_i adapt their velocity as a result of an actual force F_a:

$$F_a = F_p + F_{int} = m_i \frac{dv_i}{dt} \tag{1}$$

Here F_a consists of the personal desire force F_p and the interaction force F_{int}. The personal desire force F_p measures the desire of the pedestrian to reach the desired direction and velocity v_i^p from the current direction and velocity v_i. v_i^p is different to v_i because of interactions with other pedestrian (crowding) that limit their movements. We can calculate the desire force F_p as follows:

$$F_p = \frac{m_i}{\tau}(v_i^p - v_i) \tag{2}$$

where τ is the relaxation parameter.

The interaction force F_{int} is the sum of the repulsive and attraction force F_{ped}, and the environmental force F_w, as follows:

$$F_{int} = F_{ped} + F_W = \sum_{j \neq i} \mathbf{f}_{ij} + \sum_{j \neq i} \mathbf{f}_{iW} \tag{3}$$

where F_{ped} measures the psychological tendency to keep a distance to other pedestrian, so $\mathbf{f}_{ij}$ measures the tendency that pedestrian i would keep away from pedestrian j. F_W measures the force to avoid hitting walls and obstacles, so $\mathbf{f}_{iW}$ measures the force for pedestrian i to avoid obstacles.

Helbing et al. (2000) proposed the formulas to calculate $\mathbf{f}_{ij}$, as follows:

$$\mathbf{f}_{ij} = \{A_i \exp[(r_{ij} - d_{ij})/B_i] + kg(r_{ij} - d_{ij})\}\mathbf{n}_{ij} + \kappa g(r_{ij} - d_{ij})\Delta v_{ij}^t \mathbf{t}_{ij} \tag{4}$$

here the psychological tendency for pedestrian i and j to keep a distance with each other is described by a repulsive interaction force $A_i \exp[(r_{ij} - d_{ij})/B_i]\mathbf{n}_{ij}$, where A_i and B_i are constant. d_{ij} measures the distance between the pedestrian centres of mass, so $d_{ij} = ||\mathbf{r}_i - \mathbf{r}_j||$. $\mathbf{n}_{ij} = (\mathbf{r}_i - \mathbf{r}_j)/d_{ij}$ is the normalised vector pointing from pedestrian j to i.

The pedestrian 'touch' each other if their distance d_{ij} is smaller than the sum of their radius $(r_i + r_j)$. Helbing et al. (2000) assumes two other interacting forces in this case to model close-distance interactions: (1) a 'body force' $k(r_{ij} - d_{ij})n_{ij}$ and (2) a 'sliding friction force' $\kappa(r_{ij} - d_{ij})\Delta v_{ij}^t \mathbf{t}_{ij}$. These two forces are integrated through a binary function $g(x)$, which is zero if the pedestrians do not touch each other, and is equal to x otherwise.

Similarly, the interactions with the walls or obstacles can also be calculated as follows:

$$\mathbf{f}_{iW} = \{A_i \exp[(r_i - d_{iW})/B_i] + kg(r_i - d_{iW})\}\mathbf{n}_{iW} + \kappa g(r_i - d_{iW})(\mathbf{v}_i \cdot \mathbf{t}_{iW})\mathbf{t}_{iW} \tag{5}$$

The SFM is a very popular model of pedestrian dynamics. We develop an agent-based model of pedestrian simulation using the SFM (Helbing et al. 2000), with the following fundamental characteristic of an agent-based model:

- individual heterogeneity,
- agent interactions,
- emergence.

Individual heterogeneity is represented through by modelling agents of with random desired speed and parameters. The agent also interact with each other through the aforementioned interactions, that govern their behaviours. When many agents occupy a limited space, crowding – an emergent property, also arises as a result of agents' behaviours.

3.3 Gaussian process emulator of Agent-based models

As one approach to emulate the ouput of the true ABM simulator, we use Gaussian Process (GP) regression to construct an approximate model that will enable real-time prediction of the simulator output. Formally, a GP is a collection of random variables, any subset of which are described by a multivariate normal distribution. It can also be thought of as a distribution over functions, where each of the random variables is drawn from the underlying distribution of functions at a particular value. A GP is uniquely defined by a mean function and a covariance function, which determine the distribution of the random variables comprising the GP. In this work, we assume that the mean function is zero, in that we do not provide any prior constraints on the shape of the output of the ABM simulator, though it is a straightforward modification to include a nonzero mean function. Given a set of known values of the simulator, we can condition the GP on those inputs in order to estimate the output values of the function at an arbitrary point. Informally, a GP interpolates between known values of the simulator in high dimensional space, and provides estimates of the output function values and its corresponding uncertainties about those outputs at arbitrary points. Two additional practical advantages of constructing a GP emulator is that the process of fitting a GP relies only on standard numerical linear algebra routines, and provides a straightforward way to estimate the values of hyperparameters. Due to these pertinent features, GP emulation is widely used across a number of fields in order to construct approximate models in a wide variety of contexts.

In this work, we assume that the input vectors $\mathbf{x}$ are q-dimensional, and the simulator output is a single variable. Multiple outputs can be handled by fitting an individual emulator independently to each output. We use the standard squared exponential covariance function to describe the GP:

$$k(\mathbf{x}, \mathbf{x}^*) = \sigma \exp\left[-\frac{1}{2}(\mathbf{x} - \mathbf{x}^*)^T \beta (\mathbf{x} - \mathbf{x}^*)\right] \tag{6}$$

where the covariance function introduces a set of hyperparameters that must be estimated. These include an overall covariance σ, and a vector of weights β of length q describing the inverse squared correlation lengths of each input dimension of the emulator. The overall covariance coefficient determines the range of values expected for the function, while the inverse squared correlation lengths determine the length scale over which the function varies in a particular input dimension. The procedure for estimating the correlation lengths from the data is often referred to as Automatic Relevance Detection, and allows unscaled inputs to be used directly in the GP emulator.

Given a set of hyperparameter values, fitting the GP emulator requires inversion of the $n \times n$ covariance matrix $K(X, X)$. Here, X is a $n \times q$ array holding all of the inputs on which the emulator will be trained, with this notation indicating that the matrix entries are the values of the covariance function evaluated for all possible pairs of training points. $K(X, X)$ is a dense matrix, which requires $\emptyset(n^3)$ operations to invert, where n is the number of training points, using standard Cholesky decomposition to take advantage of the symmetry of the covariance matrix. This is the main computational expense in the GP fitting, and in practice limits the size of the number of training points that can be used to fit the emulator.

Once the covariance matrix is inverted, making predictions at m unknown points X^* (represented as an $m \times q$ matrix) involves computing the mean $K(X^*, X)K^{-1}(X, X)Y$, where Y is the vector of known

values on which we are training, and covariance $K(X^*, X^*) - K(X^*, X)K^{-1}(X, X)K(X, X^*)$. For a single prediction, this can be done in $\emptyset(n)$ operations for the mean and $\emptyset(n^2)$ operations for the covariance once the inverse of $K(X, X)$ is pre-computed and stored. Note that this also means that predictions become more expensive as more points are used to fit the emulator, and in particular the fact that predictions will be made on many more points than is used to train the model, the predictions can become a large computational expense that limits the ability to apply emulators in real time.

Hyperparameter estimation for a GP can be done in a robust way, as the log-likelihood marginalized over the function values, as well as its first and second derivatives, can be computed in closed form with little computational expense beyond inversion of the covariance matrix. In particular, gradient-based optimization methods are commonly used for performing maximum-likelihood estimation for GPs in order to estimate the hyperparameters. In practice, all hyperparameters are constrained to be positive, and so for numerical stability all computations are based on the logarithm of the hyperparameter values, leading to an unconstrained optimization problem to estimate the hyperparameter values given a set of inputs and targets on which the emulator will be trained.

(Need to describe more specifically the approach taken here for ABMs – what are the target values? How does the emulator account for uncertainty? Many runs for a single set of parameters?)

3.4 Random Forest emulators of Agent-based models

The Random Forest (RF) is one of the most powerful machine learning techniques for the classification and regression tasks in predictive analytics. The RF model is an *ensemble* method that make predictions by combining decisions from multiple base models, such as:

$$g(x) = f_0(x) + f_1(x) + f_2(x) + ... \tag{7}$$

here, each base model is a simple Decision tree (Breiman 2001), built upon a statistical technique called *bagging*. A RF does not just average the predictions from trees, it uses two key concepts that gave it the name Random Forest:

- Each tree is built and learnt from a random sampling of training observations. This is to reduce the overall variance in the entire RF model but not at the cost of increasing the bias.
- Splitting nodes in each tree using random subsets. This is to reduce the potential overfitting problem, because each tree only sees a subset of all training features when deciding to split a node.

There are in general 3 main steps to train a Random Forest Regression model:

1. Take random samples from the training data, where each data point has the same probability of getting selected, and all the sample set has the same size as the original training data. This is generally referred as bootstrapping in statistics, which means that samples are drawn with replacement and some samples may be used multiple times.
2. A Decision Tree Regressor is trained at each sampled dataset from the previous step, and its prediction is recorded.
3. An ensemble prediction is predicted by averaging the predictions of all the trees

Interested readers may refer to Breiman (2001) for more detailed and formal descriptions of the Random Forest.

4 Numerical experiments

This section shows the experiment results, using the synthetic data from the agent-based simulation model and the emulator methods described in Section 3.

4.1 Simulation environment and synthetic data

This section describes the Agent-based simulation case study. The study area is a narrowing down corridor, as illustrated in Figure 1. The agents' movements are governed by the Helbing's Social Force model, as described in the Section 3.2. All agents has the same goal to reach the exit as soon as possible.

This ABM is stochastic because each agent has a random maximum speed, that is generated randomly from a truncated normal distribution. It is used as an individual parameter to govern the maximum walking speed of the agent. Figure 3 shows the distribution of this maximum speed parameter, versus the actual walking speed of the agent in a simulation.

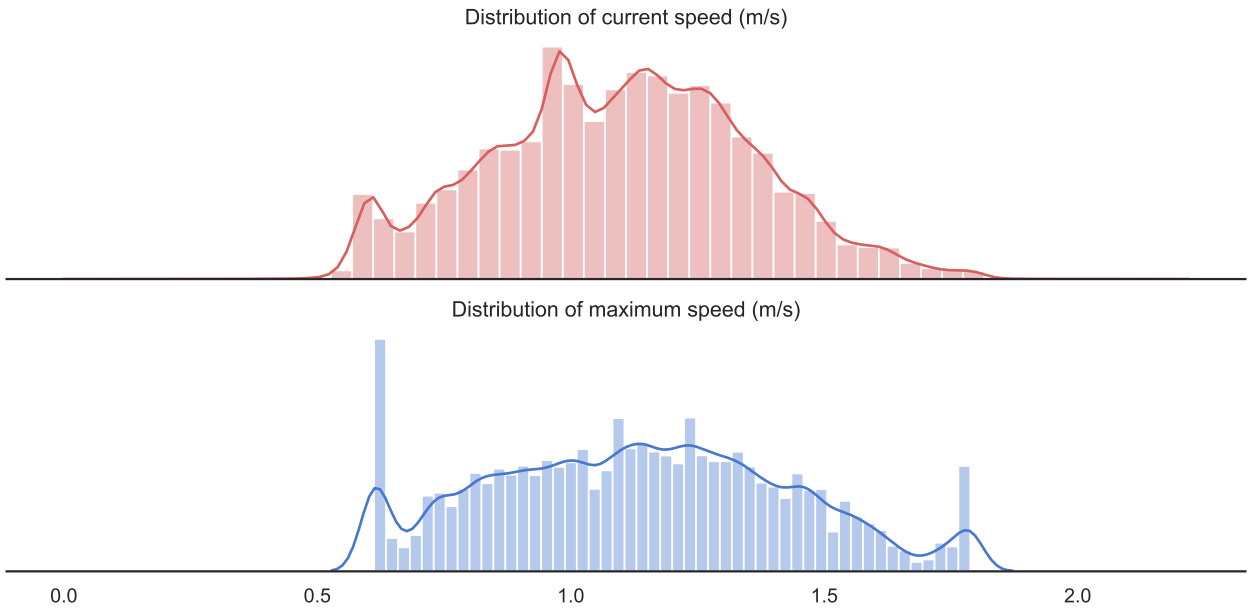


Figure 3: Distributions of current and maximum speed (m/s)

The narrowing down corridor (Figure 1) has been deliberately chosen as the case study to create areas of higher density, because more agents will gather in a limited space and congestion will be more likely.

4.2 Mid-term average prediction

data at 120s in, then predict the average social force at 1000 - 2000s

Show both cases: Seen demand versus Unseen demand for both Random Forest and Gaussian Process

4.3 Short-term prediction

Predict the next minute for both seen and unseen demand

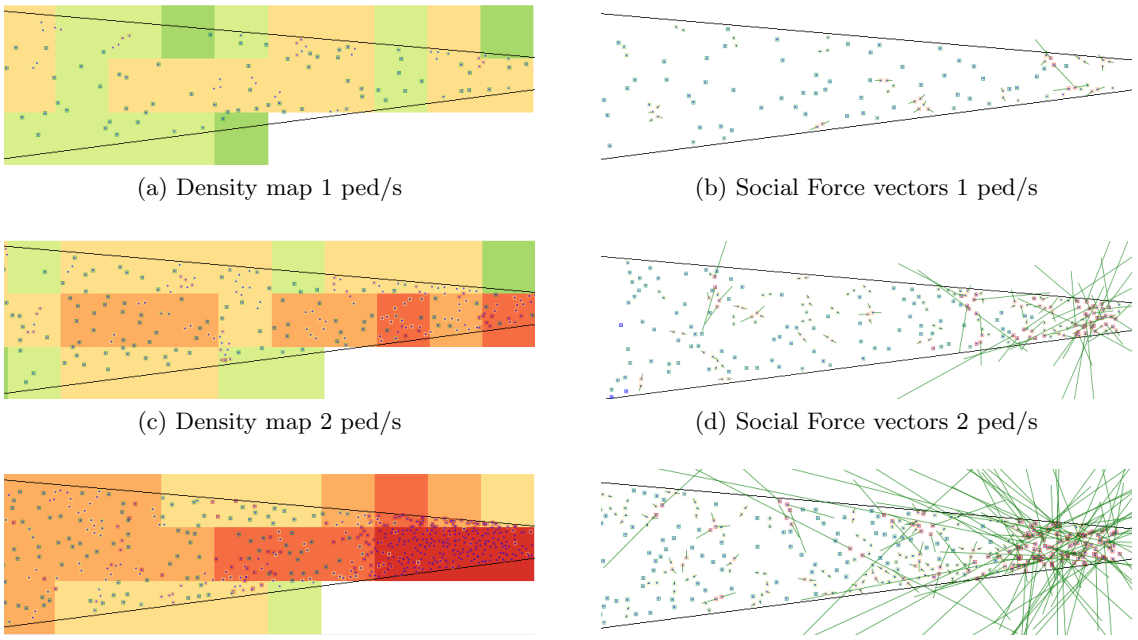
4.4 Multi-output short-term prediction

Predict the social force in each cell

References

References

Anda, C. (2017). OD MATSim: Aggregated mobile phone data transport simulations. IVT, ETH Zurich.



(a) Density map 1 ped/s

(b) Social Force vectors 1 ped/s

(c) Density map 2 ped/s

(d) Social Force vectors 2 ped/s

(e) Density map 3 ped/s

(f) Social Force vectors 3 ped/s

Figure 4: Density map and Social Force Vectors

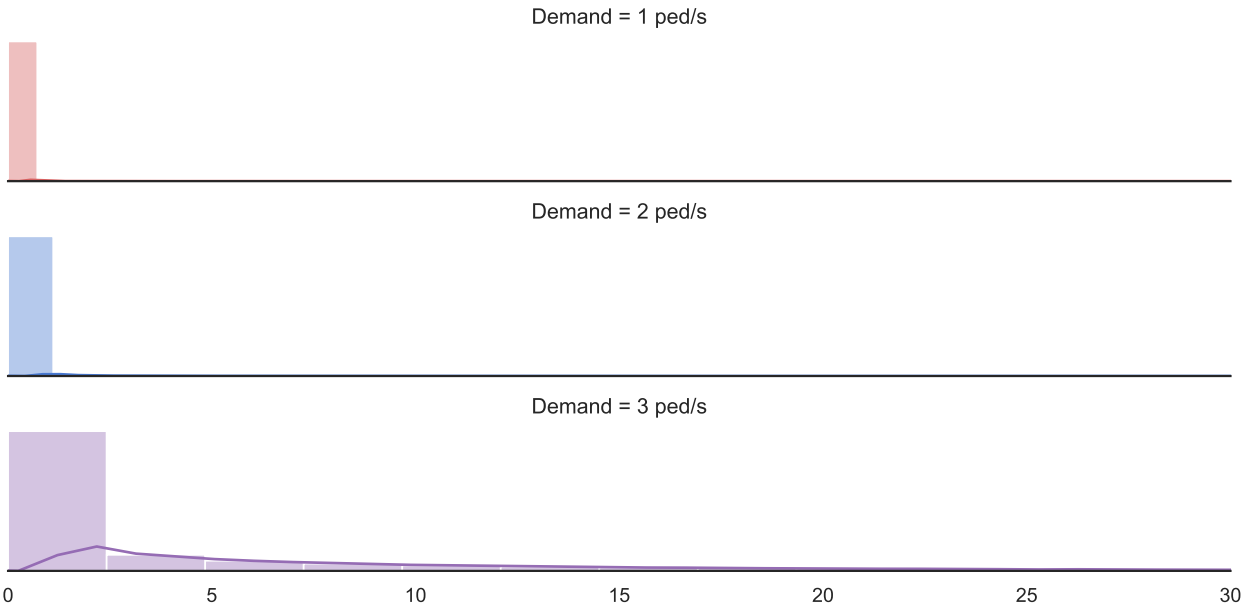
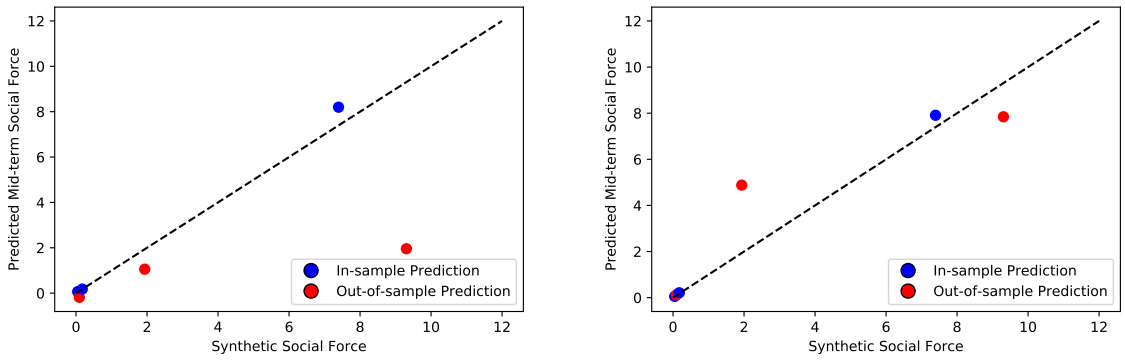


Figure 5: Social force distribution at different pedestrian demand

Baker, E., Challenor, P., and Eames, M. (2019). Diagnostics for Stochastic Emulators. *arXiv:1902.01289 [stat]*. arXiv: 1902.01289.

Balmer, M., Rieser, M., Meister, K., Charypar, D., Lefebvre, N., and Nagel, K. (2009). Matsim-t: Architecture and simulation times. In *Multi-agent systems for traffic and transportation engineering*, pages 57–78. IGI Global.

Bastos, L. S. and O'Hagan, A. (2009). Diagnostics for Gaussian Process Emulators. *Technometrics*,



(a) Gaussian Process Emulator (b) Random Forest Emulator

Figure 6: Mid-term prediction of social force

51(4):425–438.

Batty, M. (2007). *Cities and Complexity: Understanding Cities with Cellular Automata, Agent-Based Models, and Fractals*. MIT Press, Cambridge, Mass.

Bhattacharya, S. (2007). A simulation approach to Bayesian emulation of complex dynamic computer models. *Bayesian Analysis*, 2(4):783–815.

Bijak, J., Hilton, J., Silverman, E., and Cao, V. D. (2013). From agent-based models to statistical emulators. In *Joint Eurostat/UNECE Work Session on Demographic Projections*, page 12.

Bonabeau, E. (2002). Agent based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(90003):7280–7287.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Conti, S., Gosling, J. P., Oakley, J. E., and O'Hagan, A. (2009). Gaussian process emulation of dynamic computer codes. *Biometrika*, 96(3):663–676.

Conti, S. and O'Hagan, A. (2010). Bayesian emulation of complex multi-output and dynamic computer models. *Journal of Statistical Planning and Inference*, 140(3):640–651.

Dosi, G., Pereira, M. C., Roventini, A., and Virgillito, M. E. (2018). The effects of labour market reforms upon unemployment and income inequalities: an agent-based model. *Socio-Economic Review*, 16(4):687–720.

Farah, M., Birrell, P., Conti, S., and Angelis, D. D. (2014). Bayesian Emulation and Calibration of a Dynamic Epidemic Model for A/H1n1 Influenza. *Journal of the American Statistical Association*, 109(508):1398–1411.

Heard, D. (2014). *Statistical Inference Utilizing Agent Based Models*. PhD thesis, Duke University.

Helbing, D., Buzna, L., Johansson, A., and Werner, T. (2005). Self-Organized Pedestrian Crowd Dynamics: Experiments, Simulations, and Design Solutions. *Transportation Science*, 39(1):1–24.

Helbing, D., Farkas, I., and Vicsek, T. (2000). Simulating dynamical features of escape panic. *Nature*, 407:487 EP–.

Helbing, D. and Molnár, P. (1995). Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286.

Higdon, D., Gattiker, J., Williams, B., and Rightley, M. (2008). Computer model calibration using high-dimensional output. *Journal of the American Statistical Association*, 103(482):570–583.

Hilton, J. (2017). *Managing Uncertainty in Agent-Based Demographic Models*. PhD thesis, University of Southamton.

Krasnopolsky, V. M., Fox-Rabinovitz, M. S., and Chalikov, D. V. (2005). New Approach to Calculation of Atmospheric Model Physics: Accurate and Fast Neural Network Emulation of Longwave Radiation in a Climate Model. *Monthly Weather Review*, 133(5):1370–1383.

Lafuerza, L. F., Dyson, L., Edmonds, B., and McKane, A. J. (2016). Simplification and analysis of a model of social interaction in voting. *The European Physical Journal B*, 89(7):159.

Lamperti, F., Roventini, A., and Sani, A. (2018). Agent-based model calibration using machine learning surrogates. *Journal of Economic Dynamics and Control*, 90:366–389.

Lee, J.-S., Filatova, T., Ligmann-Zielinska, A., Hassani-Mahmooei, B., Stonedahl, F., Lorscheid, I., Voinov, A., Polhill, J. G., Sun, Z., and Parker, D. C. (2015). The Complexities of Agent-Based Modeling Output Analysis. *Journal of Artificial Societies and Social Simulation*, 18(4):4.

Moutoussamy, V., Nanty, S., and Pauwels, B. (2015). Emulators for stochastic simulation codes. *ESAIM: Proceedings and Surveys*, 48:116–155.

Oakley, J. and O'Hagan, A. (2002). Bayesian inference for the uncertainty distribution of computer model outputs. *Biometrika*, 89(4):769–784.

Oyebamiji, O. K., Wilkinson, D. J., Jayathilake, P. G., Curtis, T. P., Rushton, S. P., Li, B., and Gupta, P. (2017). Gaussian process emulation of an individual-based model simulation of microbial communities. *Journal of Computational Science*, 22:69–84.

Oyebamiji, O. K., Wilkinson, D. J., Li, B., Jayathilake, P. G., Zuliani, P., and Curtis, T. P. (2019). Bayesian emulation and calibration of an individual-based model of microbial communities. *Journal of Computational Science*, 30:194–208.

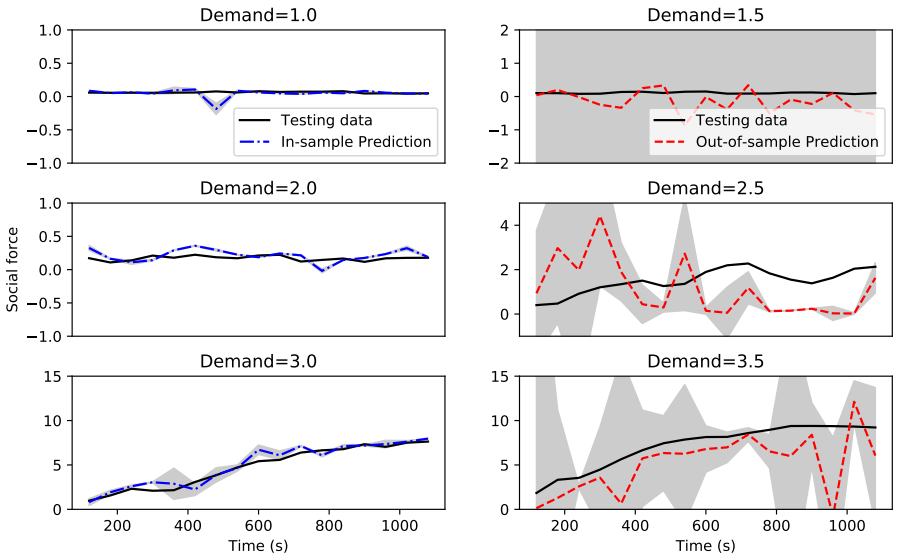
Rasouli, S. and Timmermans, H. (2013). Using emulators to approximate predicted performance indicators of complex microsimulation and multiagent models of travel demand. *Transportation Letters*, 5(2):96–103.

Schoenharl, T. and Madey, G. (2011). Design and Implementation of An Agent-Based Simulation for Emergency Response and Crisis Management. *Journal of Algorithms & Computational Technology*, 5(4):601–622.

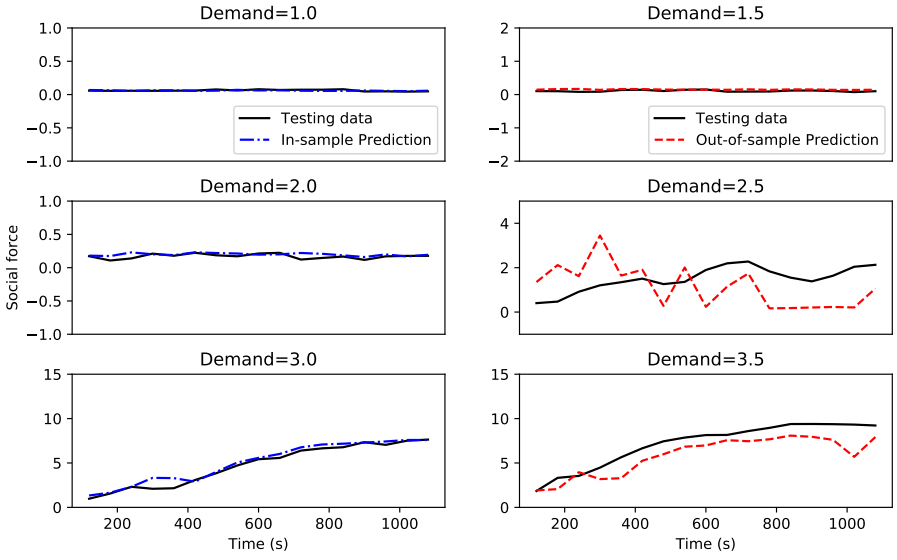
Shrestha, D. L., Kayastha, N., and Solomatine, D. P. (2009). A novel approach to parameter uncertainty analysis of hydrological models using neural networks. *Hydrology and Earth System Sciences*, 13(7):1235–1248.

Wang, M. and Hu, X. (2015). Data assimilation in agent based simulation of smart environments using particle filters. *Simulation Modelling Practice and Theory*, 56:36–54.

Ward, J. A., Evans, A. J., and Malleson, N. S. (2016). Dynamic calibration of agent-based models using data assimilation. *Royal Society Open Science*, 3(4).

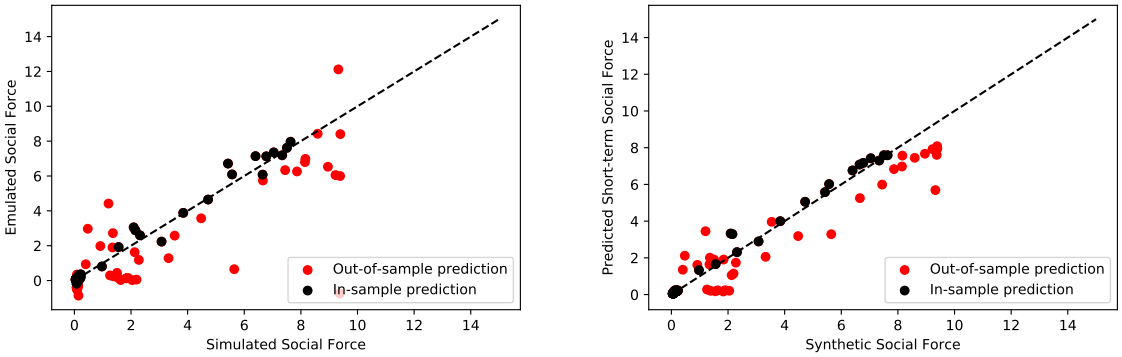


(a) Gaussian Process Emulator



(b) Random Forest Emulator

Figure 7: Short-term prediction of social force



(a) Gaussian Process Emulator (b) Random Forest Emulator

Figure 8: Scatter plot of short-term prediction

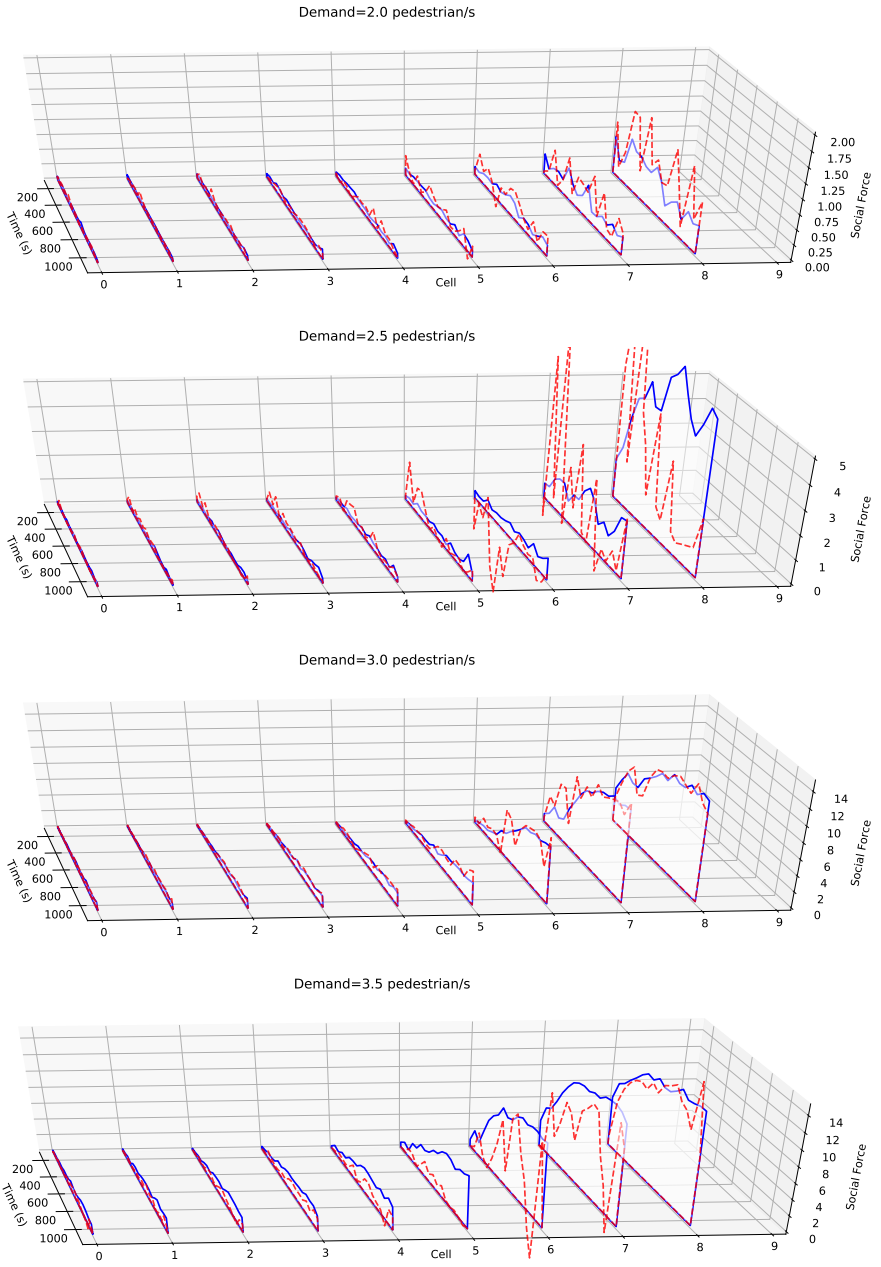


Figure 9: Prediction of social force on each cell using Gaussian Process (Prediction: Red dashed line, Synthetic data: Blue solid line)

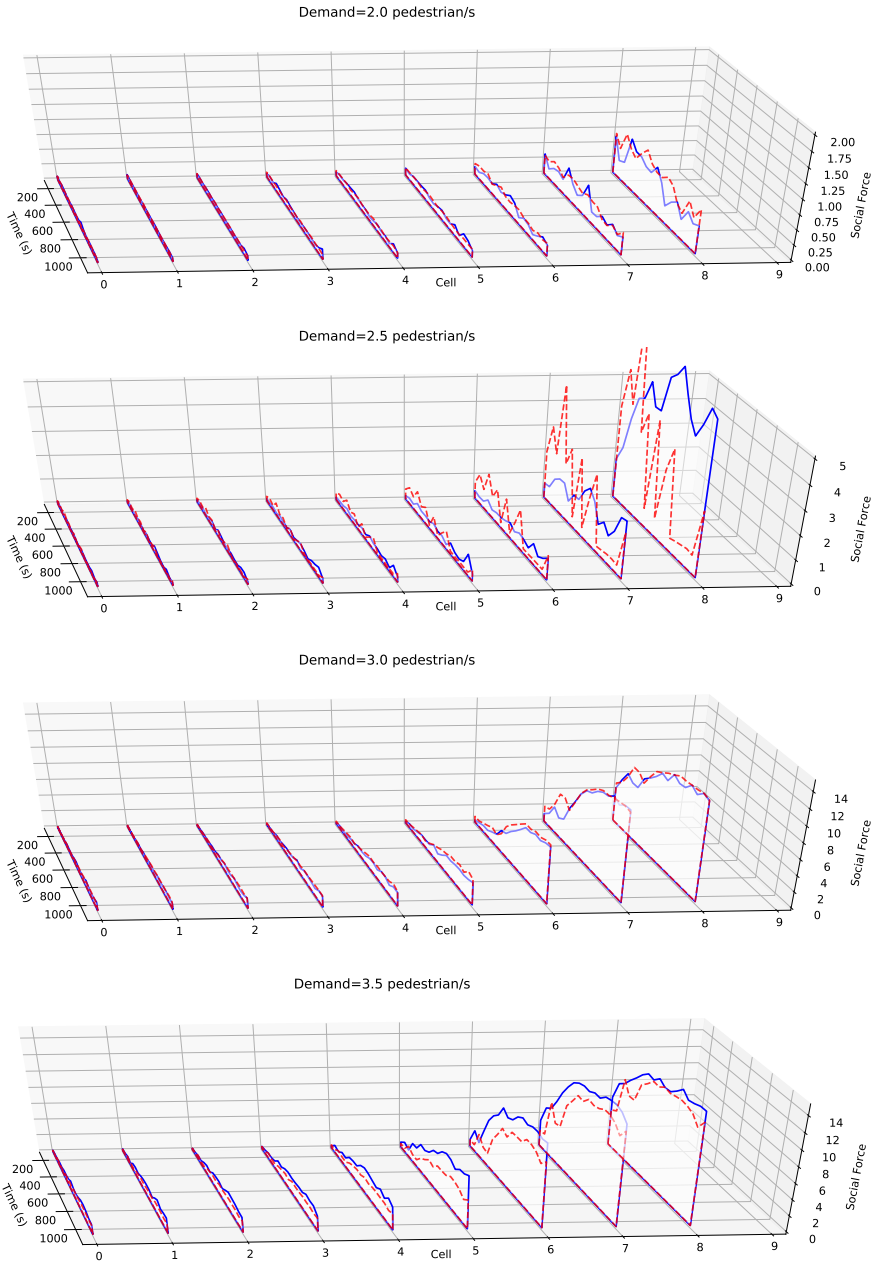


Figure 10: Prediction of social force on each cell using Random Forest (Prediction: Red dashed line, Synthetic data: Blue solid line)