Imperial College London

DEPARTMENT OF COMPUTING

Parallel Parameter Estimation for Gilli-Winker Model using Multi-core CPUs

Author:
Pengyu Guo

Supervisor:
Luk Wayne

Independent Study Option Report for the MSc degree of Imperial College London

May 2020

Abstract

Agent-based modelling is a powerful tool that is widely used to model the global financial systems. When the parameters of the model are appropriate, the price time series generated by the model exhibit marked similarities with actual financial time series and even reproduce some of their statistical characteristics.

By using Kirman's Ant model as a prototype, this report systematically explored Gilli and Winker's parameter optimization method. In view of some limitations of this method, this report promoted some improvements, including a local-restart strategy to enhance the convergence ability of the original optimization method, as well as incorporate Simulated Annealing into the original method to help the algorithm escape from local optimums. Furthermore, since the parameter optimization of agent based modelling tends to be very time-consuming, an acceleration method was also proposed to speed up this procedure. In the end, the presented methods have been validated with the EUR/USD exchange rate.

Acknowledgments

I would like to express my sincere thanks to my supervisor Prof Wayne Luk, for his guidance and encouragement throughout the project. I am also extremely grateful to Dr Ce Guo, for his technical guidance, and also for the huge amount of confidence he placed in me to help me finish this project successfully.

I am also extremely thankful for my supportive parents, who gives me the opportunity to study at this great university and for always being there for me.

Contents

1 Introduction **1**

2 Background and related work **3**

2.1 Agent based models . . . 3

2.2 Parameter estimation method . . . 4

2.3 Estimation acceleration . . . 8

3 Contribution **11**

3.1 Local restart Nelder-Mead . . . 11

3.2 Simulated Annealing . . . 13

3.3 Acceleration of the estimation procedure . . . 15

3.3.1 Parallel execute multiple simulations . . . 15

3.3.2 Avoid unnecessary redundant computations . . . 16

4 Evaluation **18**

4.1 Simulated time series . . . 18

4.2 Model accuracy . . . 19

4.3 Parameter estimation . . . 21

4.3.1 Effect of the estimation . . . 21

4.3.2 Effect of the acceleration of the estimation . . . 22

5 Conclusion and future work **27**

Chapter 1

Introduction

The global financial system that emerged in the late 19^{th} century during the first modern wave is one of the most complex systems created by humanity. There are a lot of analytical tools that tried to model this financial system, and many of them apply general equilibrium theory in the representative agent framework. That is, each agent within this framework is assumed to make its decision with full rationality and knowledge. While these approaches are a great attempt, they have two serious problems:

- When used with financial markets, they always fail to produce a simulation that matches the facts.
- Their financial simulation is very time consuming.

The main reason for the first problem is that these approaches depend heavily on many unrealistic premises, such as total rationality, market convergence to equilibrium prices, and so on. Thus, they overlook the agents' diverse strategies and the emergent characteristics of interactions between them. This makes them lack a sound micro-level foundation, as well as the incredible results of agents' interactions. An agent based model, on the other hand, effectively solves this problem.

Different from the general equilibrium analysis, in the agent based model, the behaviours of agents are both autonomous and heterogeneous. Every behaviour follows the agent's own rules and interacts with all others autonomously within a virtual market. For this reason, ABMs are able to link investors' behaviour at the micro-level with the macro behaviour of asset prices in the actual market, thus making a more comprehensive simulation of the financial market.

In addition to this, agent based modelling has also had some other advantages over the general equilibrium theory. First, using agent based modelling allows us to solve the second problem raised earlier. Since there are always a large number of independent operations in the simulation of agent based modelling, the parallelism techniques can therefore be used to improve the simulation speed. Furthermore, under ABM, agents' behaviour is artificially and thus avoiding the potential inaccuracy of mathematical equations derived under the general equilibrium theory. More so, the use of ABM makes agent behaviour and environment parameters easier to be controlled. Consequently, a basic model can spin off variant models through this operation.

Such simplicity and flexibility have helped agent based modelling explore new avenues of economic research and model the financial markets with precision. However, there are still challenges. There is evidence[1] to suggest that the parameters of the model are among the most critical factors for generating predictions similar to the real price flows. Still, since agent based modelling remains complicated in most cases, it is difficult to find significant parameters governing agents' unique behaviour. Therefore, in this article, we attempt to mitigate this potential problem by more systematically exploring Gilli and Winker's[2] nonlinear parameter optimization technique using the Kirman ant model[3] for an exploratory case study.

In view of the problems existing in the Gilli-Winker method, three solutions are introduced in this report to improve it in terms of its performance and speed:

- A local restart strategy is proposed to solve the problem of insufficient convergence ability of the original optimization strategy.
- The Simulated Annealing algorithm is incorporated into the original optimization strategy to help it escape from local minimums.
- An acceleration method was proposed to reduce the execution time of the original optimization algorithm.

The remaining part of the report proceeds as follows: Section 2 begins by discussing the structure of Kirman's ant model and looks at how Gilli and winker estimate the parameters of this model. Section 3 presents how we improve the estimation method proposed by Gilli and Winker and the detail of our acceleration method. The fourth section presents the evaluations of our research, focusing on the performance of our model, estimation method, and acceleration strategy. As a result, an average accuracy of 52% and a speed up of 14 was achieved.

Chapter 2

Background and related work

2.1 Agent based models

The ant colony model proposed by Kirman[3] in 1993 is one of the most typical models in the filed of agent based modelling. This model tries to explain ants' social herding behaviour by using a recruitment framework consisting of two types of agents. There have been numerous models that are built upon this model. Alfarano et al.[4] developed a herding model that embeds the Kirman mechanism. The main difference is that this model describes the time evolution series directly from the population state transition probability, rather than simulating the interaction at the agent level. Westerhoff and Franke[5] proposed a model that also adopts the Kirman model's idea of herding. They express the population state in a different way so as to facilitate the exposition of the herding mechanism in the transition probabilities.

In this project, we take the agent based model introduced by Kirman as a prototype to try to reproduce some stylized facts of the foreign exchange market. The first reason for this specific selection is that our goal in this report is to explore more possibilities of agent based model's parameter optimization. Therefore we stick to this basic model and leave the other complex models for future research. The second consideration is that Kirman's model has already been calibrated by several pieces of literature, which provides a sound basis for our study.

The Kirman's Ant model consists of two different groups of individuals that behave differently, which ensures model heterogeneity. The members of the first group act on fundamentals, who function on the assumption that the exchange rate S_t will return to its fundamental value $\bar{S}$ gradually. Consequently, fundamentalists' expected exchange rate at the next time step is:

$$E^f_{t+1} = S_t + v(\bar{S} - S_t) \tag{2.1}$$

Where v is the level of change in anticipation of the fundamentalists. It is worth noting that, in the original Kirman's ant model, a constant fundamental value is shared by all fundamentalists. Whereas in our project, in order to increase the diversity of the model, each member of fundamentalists has its own fundamental value, which will change over time. The fundamental value at time t + 1 is generated from a normal distribution $(\bar{S}_t, \sigma)$, where σ is the innovation constant.

The second group of individuals follows a chartist rule that is assumed to extrapolate the last period return. That is, members in this group expect the exchange rate at the next time step to be:

$$E^c_{t+1} = S_t + (S_t - S_{t-1}) \tag{2.2}$$

An interesting aspect of this model is that as the simulation proceeds, the type of an agent will keep changing. There are two possible reasons: an individual is recruited by a different type of individual, or an individual spontaneously mutates. As a result, the proportion of chartists and fundamentalists in the total population will keep changing. Recent evidences(4) suggests that this helps the basic model to generate complex econometric behaviour such as excess kurtosis and decreasing leptokurtosis.

Another interesting feature of this model is the way it uses to simulate the time series of the exchange rate. The following formula is used to do so, where the market price at each time step is calculated as a weighted mean of market expectation:

$$S_{t+1} = W_t E^f_{t+1} + (1 - w_t) E^c_{t+1} \tag{2.3}$$

Instead of using the share of fundamentalists directly, Kirman defined W_t as the proportion of agents who expect fundamentalists to dominate the current population. Consequently, the proportion W_i is calculated as follows.

$$W_i = P(q_i > 0.5) \tag{2.4}$$

Where q_i is a normal distribution has: mean $= \frac{\text{number of fundamentalists}}{\text{size of population}}$

The benefit of this is that the agents' judgment of which type of agent dominates the market is subject to some error, this is similar to human beings in the real financial market as humans are not perfect as well. Therefore by doing so, agents in the Ant model can behave more similarly to the actual investors in the financial market and thus produce more realistic results.

2.2 Parameter estimation method

As mentioned before, it is now well established that the parameters of the agent based model can highly influence model's simulation results. If a proper parameter value is given, then the model is likely to generate results showing certain statistical similarities with real time series for the financial market. Existing methods that optimize the parameters of the agent based models can be roughly categorised into two types: the historical data approach and the indirect optimization approach. The historical data approach works by splitting the data into two sets: a validation set, which is responsible for checking the result, and a modelling set that test the model.

The estimation method proposed by Recchioni et al. [6] is a representative one of this kind of approach. In their approach, a basic gradient-based algorithm was used, and the model was evaluated based on its prediction errors on the validation set. The indirect estimation approaches use typical moments of real price series to evaluate the performance of model outputs. In this project, parameters of the Kirman's model is estimated using an indirect estimation method proposed by Gilli and Winker [2]. The indirect estimation method is more preferred because compared to the historical data method, this method is generally less expensive to run, in hardware resources and time. Therefore, this method is more suitable for this project.

Algorithm 1 shows the main steps of Gilli and Winker's indirect estimation method.

Algorithm 1 Gilli and Winker's parameter optimization method

```
1: while not converge do
2:    Nelder-Mead optimization algorithm $\longrightarrow$ successive vectors x
3:    Evaluate objective function f(x)
4: end while
```

Line 3 assesses whether a particular parameter vector can result in appropriate model output. To do so, some metrics which allow one to quantify how well the simulation result matches the real price flow need to be established. In this method, the degree of similarity between them is evaluated using two moments: the Arch(1) effect and the empirical kurtosis. These two moments were chosen because they are able to capture some characteristic features of the daily return series, for example the time varying volatility. Since these characteristic features are believed to be robust and significant, these two moments are expected to be able to describe the daily return series comprehensively and accurately [7].

Consequently, the goal of the estimation procedure is to find an appropriate combination of parameters, which can generate simulation series that minimize the following objective function:

$$f = |k^{ag} - k^{emp}| + \lambda|\alpha_1^{ag} - \alpha_1^{emp}| \tag{2.5}$$

Where α_1^{ag} and k^{ag} represent the Arch(1) effect and Kurtosis of the time series generated by the agent based model respectively, α_1^{emp} and k^{emp} denotes that of the real data. λ is a constant that was used to balance these two moments' relative magnitude.

Algorithm 2 shows in more detail how we evaluate a parameter combination with these two moments. That is, how the objective function is approximated in line 3 of Algorithm 1.

Algorithm 2 Approximation of the objective function

```
1: for i = 1:number of repetitions do
2:    SimulationResult = []
3:    for j = 1:number of interactions do
4:       Generate an exchange rate $X^{temp}$
5:       SimulationResult = SimulationResult + $X^{temp}$
6:    end for
7:    Compute $\alpha_1^i$ and $k^i$ of SimulationResult
8: end for
9: Truncate the first and last 10% of $\alpha_1$ and $k$ 's distribution
10: $\alpha_1^{ag}$ = mean($\alpha_1^1$,...,$\alpha_1^i$)
11: $k^{ag}$ = mean($k^1$,...,$k^i$)
12: Objective function$f = |k^{ag} - k^{emp}| + \lambda|\alpha_1^{ag} - \alpha_1^{emp}|$
```

There are two for loops in the algorithm above, where the inner loop uses the given parameter combination to conduct the agent interaction and produce the simulation result. The outer loop is responsible for doing Monte Carlo repetition for each parameter combination to reduce errors due to the model's stochastic effects.

The optimization strategy for selecting the successive vectors in Step 2 of algorithm 1 is another crucial ingredient besides the objective function. In their implementation, Gilli and Winker use the Nelder-Mead simplex search method in this step. This method uses a geometry called simplex as the 'vehicle' of its search in the parameter space. At every iteration, it tries out several modifications to the current shape of the simplex, and choose one that moves the current simplex towards a better region. Ideally, the last few iterations would be a continual shrinking of the simplex toward the best point inside it.

In more detail, the Nelder-Mead method works as follows. In order to generate a better simplex based on the current simplex shape, this method starts by sorting all the vertices of the current simplex by the objective function f. If we are dealing with an 2-dimensional parameter space, then each simplex has three vertices x_h, x_s and x_l. Let's assume that $f(x_l) < f(x_s) < f(x_h)$ (point l is the best point and h is the worst one). Then, the next step of this method is to calculate the reflection point x_r according to the following equation:

$$x_r = c + \alpha(c - x_h) \tag{2.6}$$

Where α is the reflection constant, c stands for the centroid of the current simplex, which considers all points except the worst one. The performance of x_r is then evaluated to determine the shape of the new simplex. This leads to four cases:

1. If the performance of x_r is somewhere between that of the two best points of the current simplex (i.e. $f(x_l) < f(x_r) < f(x_s)$), then the worst point x_h will be replaced by x_r. By doing so, the Nelder-Mead is trying to move away from where the worst point is located. As a result, the new simplex consists of three vertices: x_s, x_r, x_l. As shown in figure 2.1(a).

2. Figure 2.1(b) shows the second case. If the point of reflection performs no worse than the best point in the current simplex($f(x_r) <= f(x_l)$), then the algorithm will try to find an even better “expanded point” by moving a little bit more along the direction of r. The expanded point is calculated as follow:

   $$x_e = c + \gamma(x_r - c) \tag{2.7}$$

   Then, the better one in x_r and x_e will be used to replace x_h to produce the new simplex.

3. In the worst case, if x_r performs worse than the second-worst point x_s, then that may mean that moving in the direction described by r may not be the best choice, therefore, in this case, we contracting our simplex:

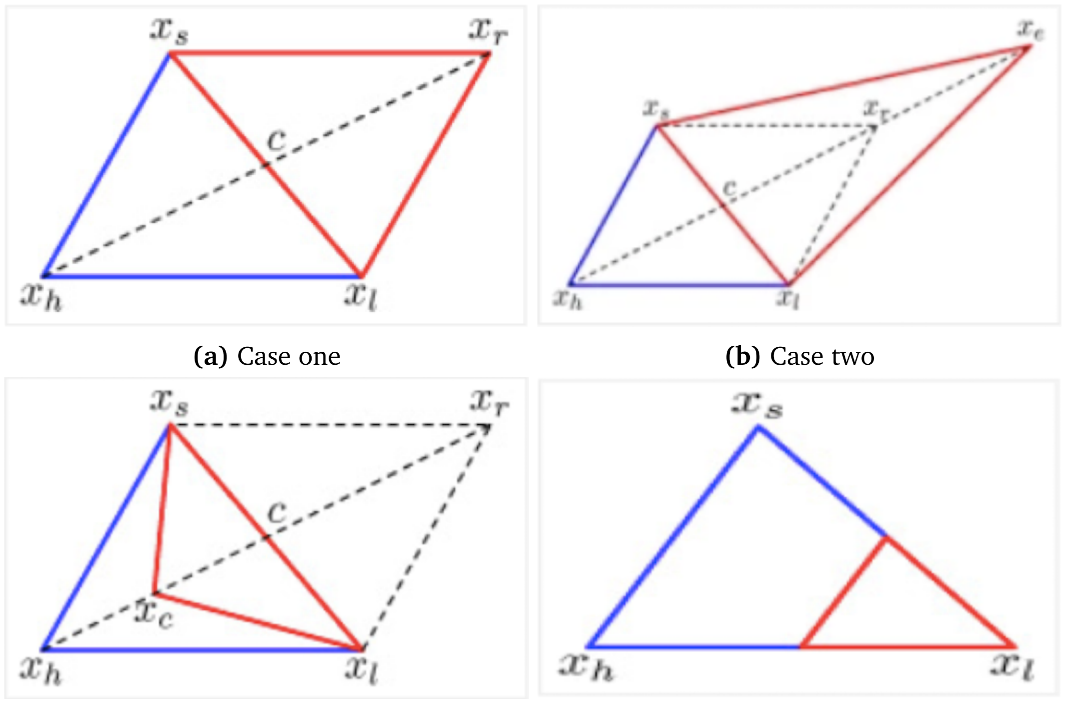
   $$x_c = c + \beta(x_h - c) \tag{2.8}$$

   If the performance of x_c is better than the worst point x_h, then x_h is replaced with x_c, and the new simplex is generated accordingly, as shown in figure 2.1(c). However, if not, then we conduct the fourth transformation: the shrink contraction, for which we redefine the entire simplex.

4. Figure 2.1(d) shows the shrink contraction: in this case, only the best point x_l is kept, the other two points of the new simplex are produced with respect to it and the previous points:

   $$x_j = x_l + \delta(x_j - x_l) \tag{2.9}$$

   By doing so, every point in the simplex is pushed towards the current best point, aiming to converge to the best neighbourhood.



(a) Case one **(b)** Case two

(c) Case three **(d)** Case four

Figure 2.1: Different cases of Nelder-Mead

2.3 Estimation acceleration

The most significant advantage of Nelder-Mead method is that it is able to choose search steps efficiently so that the objective function can be optimized very fast. In our project, this means that when doing parameter optimization, this method can minimize the number of parameter evaluations and thus reduce the time required for estimation. However, due to the optimization strategy itself, a single evaluation still requires multiple repetitions of the simulation, and therefore the running time of Gilli and Winker's parameter optimization algorithm is still very long. According to Gilli and Winker's suggestion, in order to produce a sufficiently stable result, each evaluation requires at least 200 repetitions of Monte Carlo simulation, and each repetition requires at least 10,000 agent interactions. Using this setting, it took them three days to estimate the optimal parameter combination that contains three parameters, which is too time consuming. Therefore, this section mainly focuses on the methods proposed in the past literature that speed up the parameters optimization of agent based model, and discusses how these methods enlighten our acceleration strategies.

A large and growing body of literature[8] has proved that hardware platforms such as multi-core CPU and GPU can be used to achieve a considerable acceleration effect on the parameter estimation speed of agent based models. Existing approaches of the acceleration can be roughly characterised as two types: reasonable hardware assignment and Maximisation of Parallelism.

Reasonable hardware assignment refers to finding the most appropriate allocation of a series of tasks to available hardware so that the utilization of the hardware platform can be maximized. For example, some complex operations may better be executed on GPUs, such as a large number of independent floating point calculations, while highly data-dependent control flow segments are recommended to run on CPU.

Among them, the most common approach is to use a CPU-GPU scheme, in which the CPU controls the progress of the simulation and calls the GPU for specific tasks(e.g., Complex matrix or float point calculation). Pavlov and muller's research [9] concludes that, compared to using CPU or GPU only, the CPU-GPU method is more promising. However, some other scholars(4) argue that there is a potential problem in the CPU-GPU approach: it often requires the transmission of data between the CPU and GPU. Whereas the overhead of this transmission significantly reduces the speedups this approach brings to us and may even make the estimation procedure longer than before. In order to solve this problem, Li et al. [10] proposed a method that manually setting the low-latency segment in memory as shared memory to improve the speed of GPU accessing the main memory. In agent based modelling, agents are often affected by their direct neighbours. Li et al. take advantage of this fact; they put the data of a neighbour of an agent into the low-latency shared memory in the corresponding memory of this agent, to reduce the total memory access time when updating the agent. This approach has gained great success, however, it does not applies to our project. This is because in our model, the interaction between agents is entirely random, which means that it is difficult to decide which agent's data should be put into which part of the memory.

In addition to the method of finding a suitable hardware assignment, in the literature, another way to speed up the estimation process is to maximize the degree of parallelism. This mainly benefits from the fact that simulations in ABM often requires a lot of independent computations, which provides a good condition for parallel computations. Working on GPU, Laville et al. [11] implemented an agent based simulation of microorganisms in the soil. In their method, each GPU thread executes the simulation of one agent each time, and therefore multiple simulation instances are able to be executed simultaneously on the same graphics card. However, this leads to another problem: if a simulation involves many calculations that rely on the calculation result of another thread, then the overhead of communication may reduce the benefits that parallelization brings to us. A qualitative study given by Li et al.[12] proposes a mechanism to mitigate this problem: in the case of the estimation procedure consists of multiple repeated simulation instances, computations common to multiple instances are performed only once. This avoids a lot of unnecessary redundant computations. The acceleration method proposed in this report combined the idea of the two methods mentioned above[11][12] and made some modifications to fit our model.

Overall, we found that due to GPU's mature programming framework and reliable performance, the vast majority of literature on ABS using hardware accelerators has focused on GPUs, including the previous two studies[11][12]. However, the evidence presented so far seems to suggest that GPU acceleration is not suitable for our project; the reasons are summarized as follows:

- Most current GPUs are connected to their host CPU via a specific data transmission bus, which means that the GPU can not directly access the main memory. Nevertheless, in our model, data needs to be transferred from the main memory to graphical memory and back again for every epoch of the simulation. Therefore, it is impossible to avoid the data transfer between the CPU and GPU by putting all the agent data on the GPU. This, coupled with the previously mentioned problem of not being able to optimize data transfer speed using shared memory, using GPU acceleration in our project may even be counterproductive.

- GPU is made for highly parallel simple tasks such as multiplying big matrices and complex float point calculations. This is also one of the reasons why most of the literature on agent based simulations has focused on GPU - the model they use always contains convoluted agents' behaviours and decisions, which requires a lot of calculations that can be accelerated by using GPU. In contrast, our basic model only includes a few this kind of calculations, and therefore the acceleration effect of GPU on our model may not be satisfactory.

- High performance on a GPU requires the given task to be expressed in a way that fits the GPU's hardware properties. The main requirements include the possibility of achieving coalesced memory access as well as a common control flow within a warp between the threads. Therefore, unlike using multi-core CPUs, GPU programming requires a deep understanding of the GPU architecture to maximize its performance. This increases the difficulty of using it for acceleration.

For the above reasons, this project uses multi-core CPUs to conduct parallel acceleration for the estimation process. More details will be discussed in section 3.3.

Chapter 3

Contribution

This chapter presents the contributions of this report, focusing on three methods that improve upon Gilli and Winker's agent based modelling parameter estimation method - local restart Nelder-Mead, Simulated Annealing, and the Multi-core CPU acceleration strategy.

3.1 Local restart Nelder-Mead

This section starts by explaining in more detail why the Nelder-Mead method is so important for estimating Kirman's Ant model. Although in their paper, Gilli and Winker did not elaborate on why they chose the Nelder Mead method as the optimization strategy. However, according to the specificity of this optimization problem, the potential reason for choosing the Nelder-Mead method may include:

- As pointed out previously, the Nelder-Mead method tends to optimize the objective function reasonably fast and in an efficient way. That is, Nelder-Mead allows the estimation algorithm to choose efficiently search steps and thus reduce the number of evaluations as much as possible. This is very important to Gilli and Winker's estimation algorithm as in their method, the evaluation of each parameter combination is computationally expensive (each evaluation requires $5 * 10^7$ operations).

- The objective function of their method is continuous but non-differentiable due to Monte Carlo variance. As one of the derivative-free optimization methods, the execution of the Nelder-Mead method is based on comparisons of function values only and does not need any information on derivatives.

- The Nelder-Mead method can effectively deal with a very high number of variables, which benefits the potential future work of estimates more parameters simultaneously.

According to what we have said before, it can conclude that the Nelder-Mead method is indispensable for estimating Kirman's Ant model. However, it has been proved by several researchers[13] that the Nelder-Mead method may converge to a non-optimal local solution or fail to converge at all – even if we start from a feasible initial solution. This is mainly due to the deterioration or insufficient decrease of the simplex geometry in the original Nelder-Mead process.

Therefore, in order to make Nelder-Mead's performance more stable and enhance its convergence ability, our project uses a local restart strategy to conduct the estimation, which can be formally defined as:

Algorithm 3 Nelder-Mead with local restart

```
1: $\theta^1$ = Initial parameter value
2: acc = 1
3: i = 0
4: while $acc > \epsilon$ do
5:     i++
6:     $\theta^i = Nelder - Mead(\theta^{i-1})$
7:     $acc = |f(\theta^{i-1})/f(\theta^i) - 1|$
8: end while
```

Where ϵ is the stopping accuracy required for acc. The original Gilli and Winker's estimation method only calls the Nelder-Mead function once per iteration. However, this single call transforms the simplex in the parameter space many times (usually more than 400 times) until the shape of the simplex stays more or less the same. In contrast, our local restart strategy sets the number of evaluations per call to a low value. When this call is over, a new Nelder-mead optimization is restarted, and this new optimization's initial simplex is constructed around the solution obtained in the preceding phase. Restarts are repeated until the result of several continuous Nelder-Mead optimization remain stable.

The advantage of this strategy over the original optimization strategy is that restarting the Nelder-Mead regenerates its search simplex, and in the end, many search directions are covered. This avoids the problem of finding a nonoptimal solution due to simplex degradation. In fact, It has already been proved in the literature that restarting the Nelder-Mead for several times is beneficial[14].

It is also worth noting that the most prevalent local restart strategies do not change the number of evaluations per call. In other words, in order to enhance the convergence ability of an algorithm, the ordinary restart strategy will only call the algorithm several times without changing the implementation details of the algorithm. However, in our implementation, we manually reduce the Nelder-Mead algorithm's maximum number of evaluations. This is because the Nelder-Mead method frequently produces important changes in the first few iterations and provides satisfactory results rapidly[15], so reducing the number of evaluations per call can help improve the running speed while obtaining good results.

3.2 Simulated Annealing

In addition to the weak convergence ability we discussed in the previous section, the Nelder-Mead method has another severe limitation: it tends to get stuck in local optima. The reason is fairly simple: when the simplex enters a local optima area in the parameter space, according to the algorithm, it will keep contracting or shrinking without exploring other parts of the parameter space and therefore miss some better areas. Therefore, even though we have already strengthened the Nelder-Mead's ability of convergence in section 3.1, a global optimum is not guaranteed to be found.

Figure 3.1 shows the objective function approximation against the two parameters ϵ and σ. The other parameters are kept fixed, as shown in the table below.

Parameter List		
Symbol	Interpretation	Value
σ	probability for successful recruitment	wait for estimation
ϵ	probability for self mutation	wait for estimation
N_a	size of population	100
N_i	number of interactions	50000
V	the speed of adjustment	0.05

Table 3.1: Table of parameter

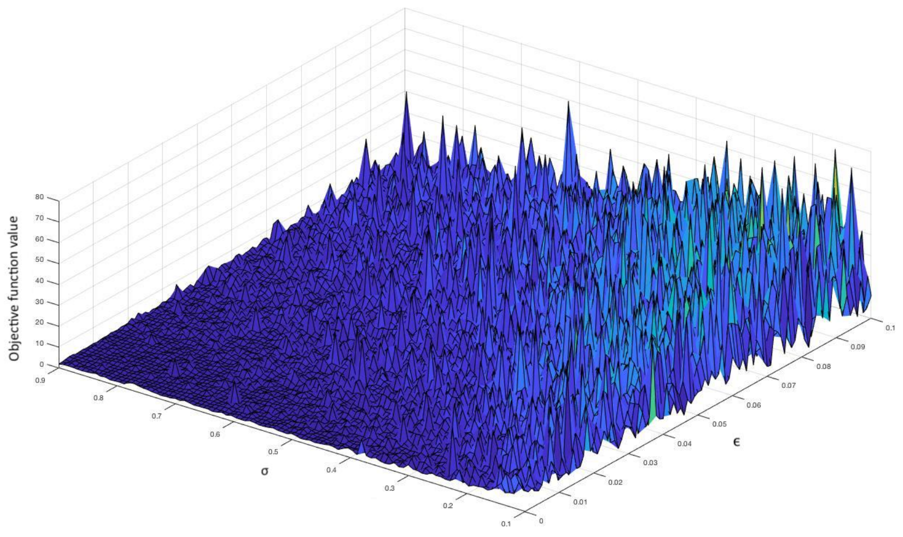


Figure 3.1: Parameter space of the objective function

In the figure, the objective function value of each parameter vector is the average result of 400 Monte Carlo repetitions, each of which conducts 10000 agent interactions. The result shows that the objective function itself does not appear to be globally convex. Even though the plot may become smoother as the number of repetitions increasing, its computational requirement will be quite high at that time[1].

Therefore, in order to avoid the algorithm get trapped into local optimums this parameter space has, it was decided to combine the local restart Nelder-Mead with the Simulated Annealing algorithm. The successive parameter vectors will be a joint decision from both of them.

Simulated Annealing is a search strategy that avoids being trapped in local optimums by frequently accepting worse solutions that are no more than a given threshold. By incorporating it into our optimization strategy, our algorithm can perform an ”uphill move” at an appropriate time to escape from the local optimum. Algorithm 4 shows the details of the new optimization strategy.

Algorithm 4 New optimization strategy: Simulated Annealing + Nelder-Mead

```
1: for each search step do
2:     $x_{new}$ = randomshift($x_{old}$)
3:     if $f(x_{new}) < f(x_{old}) + threshold$ then
           accept $x_{new}$
4:     else
           $x_{new}$ = Nelder-Mead($x_{old}$)
5:     end if
6: end for
```

It is worth noting that there is a gradual reduction of the threshold as our algorithm proceed. The advantage is that as the search proceeds, the solution should be closer to the global optimum, and a smaller threshold at this time can help the algorithm to reduce the extent of its search and converge to the global optimum.

In addition to helping the algorithm escape from the local optimum, the new strategy also takes into account the unavoidable simulation variance. For example, because of the use of the threshold, the modified algorithm becomes less likely to fall into an area that is actually bad but performs well in a particular simulation.

One of the interesting points in our implementation is that: different from the common Simulated Annealing strategy, the threshold in the simulated annealing part of our algorithm will not be reduced to zero in the later stage of the search, but stop decreasing at a low value. By doing so, we aim to make sure that there will still be opportunities to find the global optimum point even if the search direction is wrong in the early stage of the search. Later experiments show that this little modification does increase the algorithm’s probability of finding global optimum point for our

[1]Without any acceleration techniques, it takes us 20 hours to run the 400 Monte Carlo repetitions

specific parameter space.

3.3 Acceleration of the estimation procedure

As mentioned before, since the estimation procedure is time consuming, a method that accelerates this procedure is needed to be developed. In our estimation strategy, the time required to simulate the model dominates the total estimation time, and therefore we focused on optimizing the simulation speed of our model. As suggested by[16], one of the most effective ways to accelerate agent based simulations is parallel computing techniques. This is because agents are autonomous and always carry out independent operations. Besides, due to the stochastic nature of ABS, the simulation of a given scenario is usually repeated multiple times in order to generate meaningful results. This also provides an excellent opportunity for using parallel computing techniques.

Furthermore, as pointed out in section2, due to the particularity of our model, this project does not adopt GPUs, which is widely used in the acceleration of agent based simulation, but use multi-core CPUs instead. A major benefit of multi-core CPUs over GPUs is that they are able to execute a number of unmodified code explicitly written for standard CPUs. This makes migration to these platforms easier. Moreover, because the individual CPU processors allow out-of-order executions, and are able to access some fast caches, the need of adapting an algorithm's control flow to multi-core CPU is easier than with GPUs.

In fact, it has already been proved that it is feasible to use multi-core CPUs for agent based simulations. A recent systematic literature review [17] concluded that multi-core CPUs could effectively accelerate simulation tasks, and if a sufficient number of processors are provided, using the multi-core CPU based accelerator is able to have a similar performance to that of using the GPU-based acceleration.

In view of all that has been discussed in section 2 and this section, this report proposes two acceleration methods that can run simultaneously.

3.3.1 Parallel execute multiple simulations

Before discussing how the first acceleration method works, it is important to review some of the main steps of our parameter estimation strategy. Given an input parameter vector, the Simulated Annealing and Nelder-Mead will co-determine the next parameter vector. In order to do so, they need to evaluate the performance of some different candidate parameter vectors. During this process, because of the existence of simulation variance, several repeated simulations are conducted on the evaluation of each parameter vector. Then the average score across all simulations is taken as the final score of this parameter vector.

Since each simulation is independent of the others, they are able to run simultaneously. Therefore, the first acceleration strategy works by running repeated Monte Carlo simulations for each parameter vector in parallel, as shown in Algorithm 5.

Algorithm 5 Acceleration strategy one - parallel execution

```
1: parfor i = 1:number of repetitions do
2:     Generate a time series of the exchange rate
3:     Compute $\alpha_1^i$ and $k^i$ of this time series.
4: end parfor
5: Truncate the first and last 10% of $\alpha_1$ and $k$ 's distribution
6: Objective function$f = |k^{ag} - k^{emp}| + \lambda|\alpha_1^{ag} - \alpha_1^{emp}|$
```

Where parfor is a Matlab command that executes for loop in parallel.

Furthermore, since the agents in our model often perform independent operations, we have also tried another parallelization method in the early stage of our project: each processor manages the simulation for a subset of agents. However, through experiments, it is found that this method did not provide us with the expected speedup. A possible reason for this could be: the model used in this project is only a basic model such that the consumption of agent state updating and other agents' behaviours accounted for only a small proportion of the total consumption. Thus, the benefits of parallelization in this case are limited.

Consequently, this method is abandoned, and the previous one is adopted. But this abandoned method still provides the future work with a good idea, for which we will use a more complex model.

3.3.2 Avoid unnecessary redundant computations

In addition to exploiting the parallelism across multiple simulation instances, we have also adopted another acceleration mechanism. In a parameter estimation procedure that consists of multiple repeated simulation instances, duplicate computations of multiple instances are performed only once. That is, the population composition(Number of fundamentalists/number of chartists) at each time step of a single simulation and the corresponding predicted market price are recorded. If there is a new simulation that has a population composition which has appeared in other simulation's history, then the new simulation's predicted market price at this time point will no longer be generated by complex calculations, but be directly calculated using the following formula:

$$S_t = S_{pre} + noise \tag{3.1}$$

Where S_{pre} represents the predicted market price corresponding to the same population composition that has appeared in another simulation's history. The noise is supposed to be normally distributed; noise $\sim N(0, \sigma^2)$

The logical justification of this design is that in our model, the biggest factor that affecting price prediction is the composition of the population. Other factors, for example fundamental value, are essentially some randomly generated Gaussian distributions. Therefore, by using the noise term to take into account all these other factors, the simulation time can be greatly reduced while does not affect its result too much.

Chapter 4

Evaluation

This section evaluates the performance of our model, as well as analyze the capability of the modified estimation method and the effect of the acceleration algorithm. The data used in this section is the EUR/USD exchange rate of January 2019. Furthermore, it is noteworthy that the parameter used by the model in Section 4.1 and 4.2 is the optimal parameter found by our estimation algorithm.

Before presenting the details of each evaluation, the results of them are first summarized here:

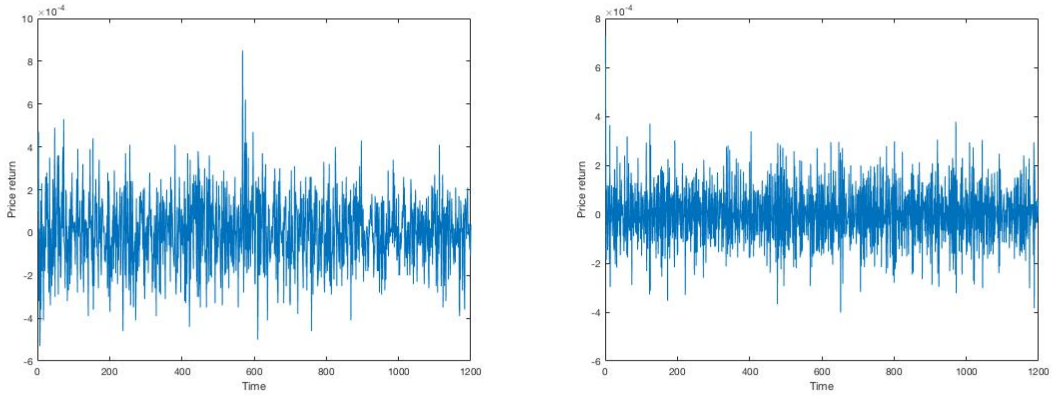
- Our model is able to generate high-quality results that reflect some critical characteristics of the real data. Besides, our model can predict the exchange rate with an acceptable accuracy - the average prediction accuracy is around 52%. More details are discussed in Section 4.1 and Section 4.2.

- The optimization strategy proposed in this report generates satisfactory solutions that are close to the global minimum point. Section 4.3 presents more details about this.

- As shown in Section 4.4, the use of the acceleration strategy provides a significant speed-up for parameter optimization. It can accelerate the original optimization algorithm to 14 times at most.

4.1 Simulated time series

Figure 4.1 shows the daily returns of the EUR/USD exchange rate and a typical simulated returns series. For the real data, the estimated ARCH(1) effect amounts to 0.1485, while it is 0.2007 for the simulated series. The frequency of large changes is also high for both series leading to a kurtosis of 1.9562 for the actual data and 2.4364 for the simulated series.



(a) Actual return **(b)** Simulated return

Figure 4.1: Daily return of the exchange rate

A first look indicates that both series exhibit similar patterns, such as volatility clustering and heavy tails. Moreover, there is a close resemblance between the simulated series and the real data series concerning ARCH(1) effect and kurtosis. This proves that the design of our objective function and the model are rational. Further analysis shows that the simulated time series also shows some interesting patterns, such as the symmetric triangle pattern and ascending/descending channels. It is worth noting that those patterns do not frequently appear with simple simulation models like the random walk.

Overall, these results indicate that some characteristics of the real data are reflected in the simulated series. This proves that even though only a basic model is used in our project, one of the most important aims of this project is accomplished - model the financial markets and be able to reproduce some of the typical behaviours of the real price flow.

4.2 Model accuracy

In addition to testing whether the characteristic of the real data can be reproduced, the performance of the model is also evaluated numerically.

In the EUR/USD dataset, the data during the period of 8 am - 8 pm every day is used as the experimental data. Data beyond this period was generated in the non-active period of exchange rate trading and thus they are regarded as low-quality information. The daily data in the 8 am - 8 pm period is split into 12 segments. Our model tries to predict the relative trend of the exchange rate in each segment compared with the previous one (e.g., Is the exchange rate at 9 am higher or lower than that of 8 am?), thereby transforming the discrete prediction results into a binary classification problem.

Two metrics are used to see how similar the predicted results were to the real data:

F_1 score and accuracy. They are calculated as follows:

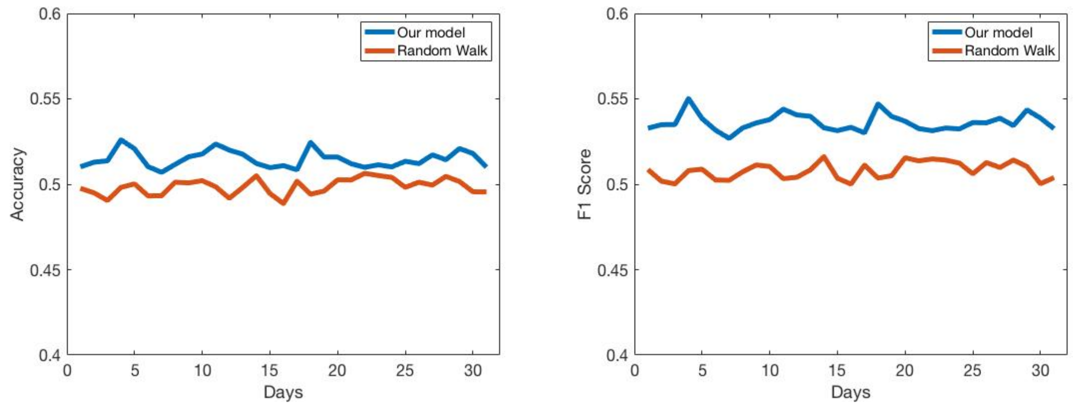
$$F_1 = \frac{2 * \text{true positive}}{2 * \text{true positive} + \text{false positive} + \text{false negative}} \tag{4.1}$$

Where $F_1 \in (0, 1)$, the larger F_1 is, the better the result.

$$acc = \frac{\text{true negative} + \text{true positive}}{\text{true negative} + \text{true positive} + \text{false negative} + \text{false positive}} \tag{4.2}$$

It is worth mention that in our dataset, the ratio of positive/negative labels is 50.32%. This indicates that the data is balanced and therefore it is appropriate to use accuracy as one of the metrics.

The figure below presents our model's daily prediction F_1 score and accuracy for all 31 days in January 2019. To make the result more convincing, the prediction result of a random walk simulation is also included in the figure, which predict the exchange rate at the next moment using a simple Gaussian distribution. Furthermore, in order to reduce experiment errors, the daily F_1 score and accuracy given in the figure are the average results of 1000 simulations.



(a) Accuracy **(b)** F1 score

Figure 4.2: Daily accuracy and f1

It can be seen that the random walk model poorly predict the exchange rate return - its accuracy has always been around 50%. This is expected because the model is making random predictions about the trend of exchange rates. In contrast, the numerical experiment shows that our agent based model yields more accurate predictions, as its f1 score and accuracy are always higher than the random model. This is consistent with our conclusions in the previous section. Since the simulated time series generated by our model exhibit behaviours similar to that of the real data, our model's result quality is also higher than that of the random guess.

In summary, this experiment justifies our model's reliability in terms of predicting reasonably accurate results for unseen situations. The correctness of the estimation algorithm is also indirectly proved because the parameters used in this experiment are the optimal parameters it found. However, since the model adopted in this project is only a very basic model that simulates a very limited number of investor behaviours, there seems to be a lot of room for improvement in terms of reproducing real data behaviour and prediction accuracy.

4.3 Parameter estimation

4.3.1 Effect of the estimation

Figure 4.3 shows the grid plot of the objective function in the ϵ - σ subspace. The two routes respectively represent the parameter search path generated using the Nelder-Mead only and using the combined strategy that consists of Nelder-Mead and Simulated Annealing.

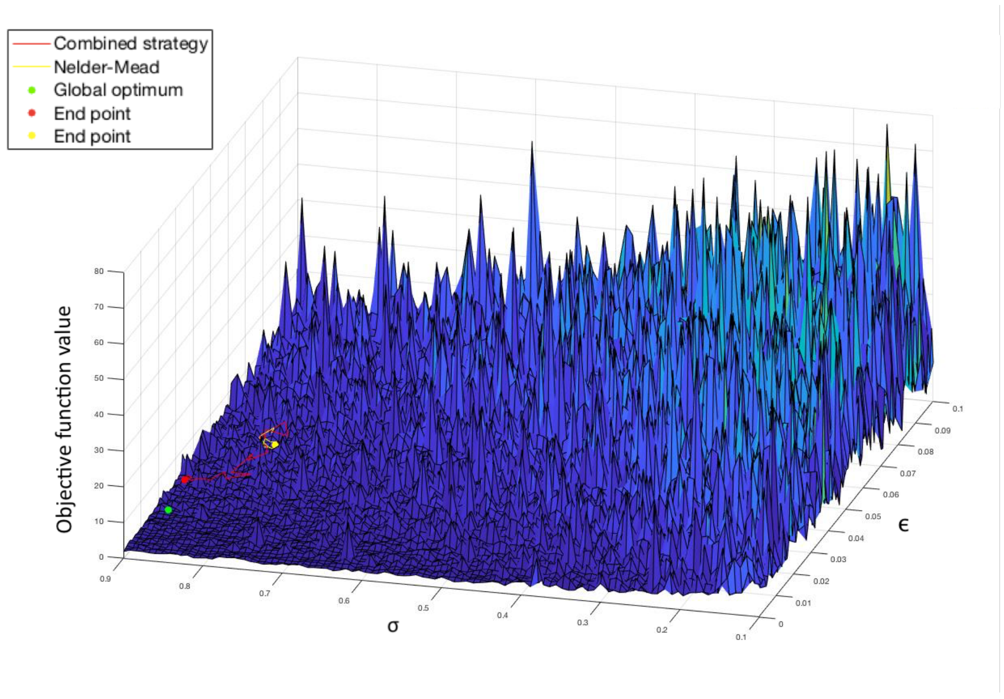


Figure 4.3: Figure shows different strategy's search path

A first look indicates that when using Nelder-Mead only, the estimation algorithm is trapped in a local optimum close to the starting point. Whereas when using our strategy that combined Nelder-Mead and Simulated Annealing, the algorithm has gone into a number of local minimum and successfully escapes them. This is expected because the aim of incorporating the Simulated Annealing into our optimization strategy is to solve the problem of local optimum. The results in this section indicate that the design of this strategy is reasonable.

As pointed out in Section 3.2, we approximated the objective function for all combinations of ϵ and σ, and the global optimum of this parameter space was found to be: (0.015,0.876). Experimental results show that the point where the algorithm ended up searching is fairly close to that of this global optimum point. Therefore it is proven that, in addition to escaping from the local optimum, our strategy is also able to approximate the global optimal solution in the case of the objective function is not globally convex.

In this project, the estimation problem mainly focuses on the two-dimensional problem: ϵ - σ. Therefore, this experiment only evaluates the performance of our estimation method on this parameter combination. In this case, running a brute-force search to try all combinations for a predefined granularity may yield more accurate results within a reasonable time as the parameter vector contains only two parameters. However, considering that in the future work, more parameters of the agent based model will be included in the estimation problem, using brute force search all the time may not be the best choice because of the curse of dimensional problem. This once again proves the necessity of the research we have done.

The most striking result to emerge from the experiment is that: the starting point of the search is extremely important. That is, the estimation algorithm is guaranteed to converge only if the initial value of ϵ is set to a small value. Whereas if it is initialized with a large value, then it will just keep increasing and exceeds one rapidly(remember this parameter stands for probability). A possible reason for this phenomena could be: one can find that the terrain of the right half of the parameter space (when the value of ϵ is large) is fairly rough, and sometimes making a small change to the parameter combination value may lead to a big step in the objective function value. When searching in this type of terrain, the simplex used by Nelder-Mead would be elongate indefinitely, and their shape goes to infinity in this space, then start growing randomly and aimlessly.

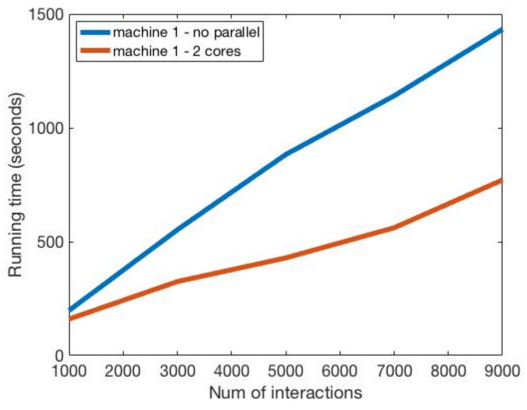
Through experiments, our model's safe range of this parameter is suggested to be: $\epsilon \in (0, 0.03)$. If the initial value of ϵ is set beyond this range, the convergence of our estimation algorithm will be seriously affected.

Overall, these results indicate that our modifications to the original estimation algorithm are successful. Compared with the old strategy, the current optimization strategy is more likely to find the global optimal solution. However, strictly speaking, it did not fully achieve the original goal of the project, because the estimation algorithm failed to converge when starting with an inappropriate initial point. Of course, future research will try to improve the capability of our estimation algorithm to ensure its convergence in the case of a bad initial point is given.

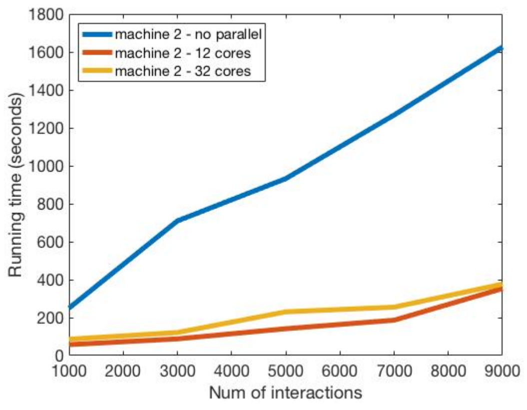
4.3.2 Effect of the acceleration of the estimation

To test the effect of our acceleration method for CPUs with a different number of processors, this section's evaluation uses two machines that have 2 and 32 processors respectively. The following plots and tables present the time required for these two machines when running the parameter optimization algorithm with different computational complexity. In each table, the speedup is calculated as:

$$\frac{\text{execution time on 1 core}}{\text{execution on K cores}} \tag{4.3}$$



(a) machine 1

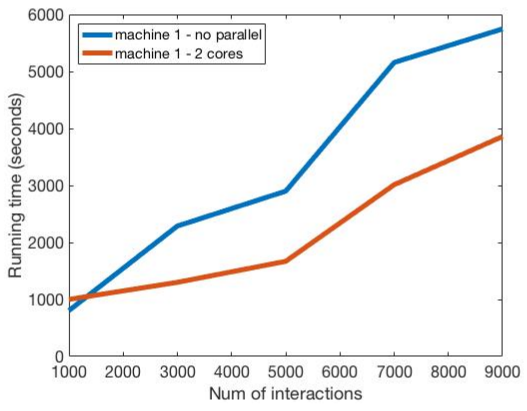


(b) machine 2

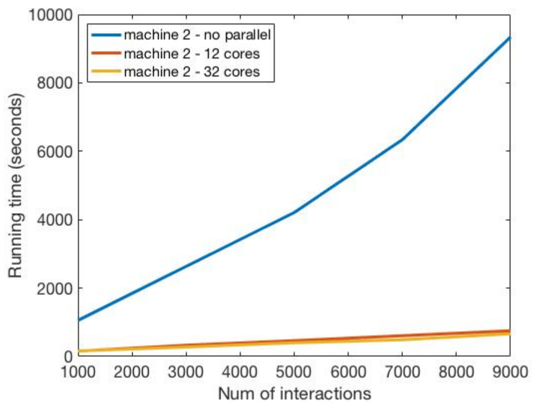
Figure 4.4: execution time for 10 epochs' optimization

		1000	3000	5000	7000	9000
Machine 1	no parallel execution time	197.541	552.174	882.677	1140.478	1432.919
	2 cores execution time	159.988	324.832	428.703	561.112	771.307
	2 cores speed up	1.2	1.7	2.1	2.0	1.9
Machine 2	no parallel execution time	249.737	709.224	932.228	1267.519	1625.767
	12 cores execution time	57.416	87.881	141.459	186.304	353.138
	12 cores speed up	4.3	8.1	6.6	6.8	4.6
	32 cores execution time	85.913	121.308	230.234	255.007	376.399
	32 cores speed up	2.9	5.8	4.0	4.9	4.3

Table 4.1: execution time and speed up for 10 epochs' optimization



(a) machine 1

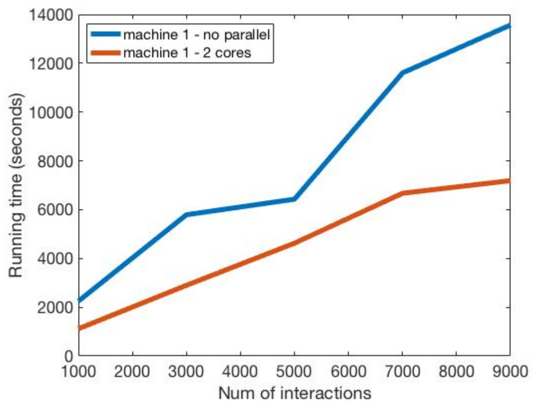


(b) machine 2

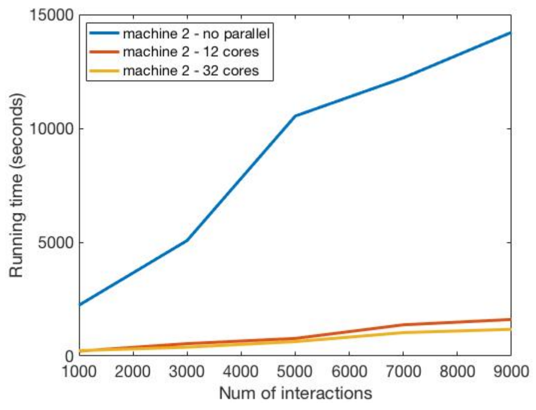
Figure 4.5: execution time for 50 epochs' optimization

		1000	3000	5000	7000	9000
Machine 1	no parallel execution time	805.448	2288.134	2900.681	5158.269	5746.729
	2 cores execution time	1001.318	1300.849	1669.681	3013.165	3859.309
	2 cores speed up	0.8	1.8	1.7	1.7	1.5
Machine 2	no parallel execution time	1054.796	2632.190	4209.715	6339.005	9340.687
	12 cores execution time	156.253	328.846	462.391	607.277	751.746
	12 cores speed up	6.8	8.0	9.1	10.4	12.4
	32 cores execution time	161.647	276.692	401.529	489.519	664.138
	32 cores speed up	6.5	9.5	10.5	12.9	14.1

Table 4.2: execution time and speed up for 50 epochs' optimization



(a) machine 1



(b) machine 2

Figure 4.6: execution time for 100 epochs' optimization

		1000	3000	5000	7000	9000
Machine 1	no parallel execution time	2249.848	5790.534	6429.422	11605.109	13569.261
	2 cores execution time	1118.562	2898.222	4625.037	6671.737	7190.364
	2 cores speed up	2.0	1.9	1.4	1.7	1.9
Machine 2	no parallel execution time	2231.647	5071.616	10537.742	12216.580	14215.568
	12 cores execution time	217.457	544.944	774.158	1372.292	1605.976
	12 cores speed up	10.3	9.3	13.6	8.9	8.9
	32 cores execution time	238.992	389.169	633.526	1029.927	1174.310
	32 cores speed up	9.3	13.0	16.6	11.9	12.1

Table 4.3: execution time and speed up for 100 epochs' optimization

The tables and plots above illustrate some of the main characteristics of our acceleration strategy. First, it can be observed that the number of CPU processors and the speed of the estimation are not increased by the same proportion. (i.e., the estimation speed is not 32 times faster for a 32-processors CPU). In some cases, we can even find that as we increase the number of processors, on the contrary, the estimation time gets longer. For example, when an optimization instance of 50 repetitions and 1000 interactions (Table 4.2) is considered, the execution time is 805s when no acceleration method is applied on machine 1, while the execution time increased to 1001s after using parallel acceleration with dual cores.

In fact, this is consistent with what figure 4.7 tells us, for which we evaluate the speedup of a different number of cores on machine 2. It shows that when the number of interactions and epochs are fixed, the effect of increasing the number of cores decreases as the number of cores increases. The reason for this is that when there are too many processors running simultaneously, the overhead associated with create and distribute the parallel tasks dramatically slows down the estimation algorithm. More processors run simultaneously means more overhead, too many processors can offset the benefits that parallelization bring to us. Therefore it can be seen that in figure 4.5(b) and 4.6(b), running with 12 threads took almost the same amount of time as running with 32 threads.

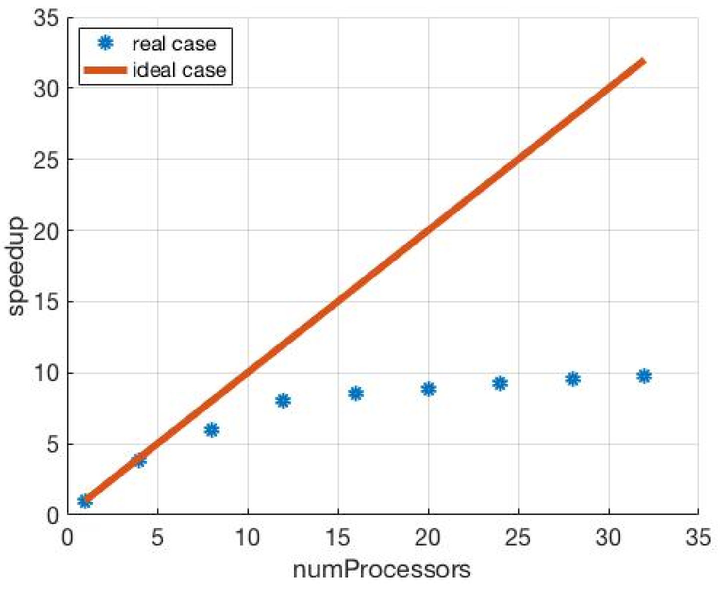


Figure 4.7: Speed up effect on machine 2

Interestingly, there was another counter-intuitive phenomenon: when the same amount of processors are used, the larger the computational complexity(e.g., more epochs and interactions), the higher the gain of the parallelization. For example, in the experiment, an optimization of 10 epochs with 1000 interactions per epoch took the 2-core CPU 160 seconds. Whereas using the same CPU, a 100 epochs optimization run for 1118 seconds instead of 1600 seconds(160 * (100/10)). The reason for this is the same as in the previous one - running multiple threads has overhead on its own, increasing the computational complexity can increase the revenue and thus indirectly reduces the consumption due to increased computational complexity.

Summarizing the findings for running our acceleration method on different machines, it can be stated that the acceleration method proposed in this report can effectively reduce the time required for Agent Based model's parameter optimization. However, the acceleration effect is greatly affected by the problem of inter-processor communication consumption we discussed earlier. Therefore, future research will also study on approaches that reduce this kind of consumption.

Chapter 5

Conclusion and future work

This report systematically explores Gilli and Winker's nonlinear parameter optimization technique, discusses some limitations of their method, and puts forward corresponding solutions.

To conclude, the major changes we made to Gilli and Winker's parameter estimation method include:

- Using a local restart strategy, this report improves the convergence ability of the original optimization method.
- The Simulated Annealing algorithm was incorporated into the original optimization strategy to help it escape from the many local optimum.
- This report proposed a multi-core CPU parallel acceleration method, which dramatically reduces the execution time of the original optimization algorithm.

The experiment shows that our agent based model is able to yield reasonable result. Our modifications to Gilli and winker's optimization strategy were also proved to work as expected - the new strategy is able to efficiently overcome local optimum and obtain close approximations of the global optimum.

However, the presented optimization strategy is restricted by the initial parameter value. Further research will concentrate on this problem to enhance the convergence ability of the optimization algorithm when a bad initial point is given.

At the same time, given that only a basic model is used in this project, we will also try to add some more complex behaviours to this model. For example, instead of using a constant value, the probability σ of convincing another individual could be made dependent on past success. By doing so, we aim to produce simulations that are closer to the real data, as well as more accurate predictions.

Furthermore, in addition to CPU and GPU, FPGA is another hardware platform that is frequently used in the acceleration of highly parallelisable tasks. Therefore, it may also be a reasonable attempt to try to use FPGA instead or combine it with the multi-core CPUs.

Bibliography

[1] Calvez B, Hutzler G. Parameter Space Exploration of Agent-Based Models. In: Khosla R, Howlett RJ, Jain LC, editors. Knowledge-Based Intelligent Information and Engineering Systems. Berlin, Heidelberg: Springer Berlin Heidelberg; 2005. p. 633–639. pages 2

[2] Winker P, Gilli M. Indirect Estimation of the Parameters of Agent Based Models of Financial Markets. FAME Research Paper 35. 2001 05;. pages 2

[3] Kirman A. Ants, Rationality, and Recruitment. The Quarterly Journal of Economics. 1993 02;108:137–56. pages 2, 3

[4] Lux T, Alfarano S, Wagner F. Estimation of Agent-Based Models: The Case of an Asymmetric Herding Model. Computational Economics. 2005 02;26:19–49. pages 3

[5] Franke R, Westerhoff F. Estimation of a Structural Stochastic Volatility Model of Asset Pricing. Computational Economics. 2011 06;38:53–83. pages 3

[6] Recchioni M, Tedeschi G, Gallegati M. A calibration procedure for analyzing stock price dynamics in an agent-based framework. Journal of Economic Dynamics and Control. 2015 08;60:1–25. pages 5

[7] Barde S. Direct comparison of agent-based models of herding in financial markets. Journal of Economic Dynamics and Control. 2016 10;73. pages 5

[8] Bauer D, McMahon M, Page E. An approach for the effective utilization of GP-GPUS in parallel combined simulation; 2008. p. 695–702. pages 8

[9] Pavlov R, Müller J. Multi-Agent Systems Meet GPU: Deploying Agent-Based Architectures on Graphics Processors. vol. 394; 2013. p. 115–122. pages 9

[10] Li X, Cai W, Turner S. Efficient Neighbor Searching for Agent-Based Simulation on GPU; 2014. p. 87–96. pages 9

[11] Laville G, Lang C, Marilleau N, Mazouzi K, Philippe L. Using GPU for Multi-agent Soil Simulation; 2015. pages 9, 10

[12] Li X, Cai W, Turner S. Cloning Agent-based Simulation on GPU; 2015. p. 173–182. pages 9, 10

[13] Lagarias J, Reeds J, Wright M, Wright P. Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions. SIAM Journal on Optimization. 1998 12;9:112–147. pages 11

[14] Kolda T, Lewis R, Torczon V. T.G. Kolda, R.M. Lewis, V. Torczon: Optimization by direct search: New perspectives on some classical and modern methods. SIAM Review 45, 385-482. SIAM Review. 2003 09;45:385–482. pages 12

[15] Singer S, Nelder J. Nelder-Mead algorithm. Scholarpedia. 2009 01;4:2928. pages 12

[16] Xiao J, Andelfinger P, Eckhoff D, Cai W, Knoll A. A Survey on Agent-based Simulation Using Hardware Accelerators. ACM Computing Surveys. 2019 01;51:1–35. pages 15

[17] Williams B, Ponomarev D, Abu-Ghazaleh N, Wilsey P. Performance Characterization of Parallel Discrete Event Simulation on Knights Landing Processor; 2017. p. 121–132. pages 15