App developers and exclusive dealing: an overview of the operating system market.

Author:
J.C. Rueda
Anr: 679403

Supervisor:
C. Fiedler

School of Economics and Management

BSc Economics

Bachelor Thesis

June, 2017

Word count: 7683

Contents

Abstract

Nowadays, a large number of markets involve platforms. These markets are known as multi-sided markets and are characterized by the presence of Network Externalities. The Operating System market for smartphones is one of the well-known examples of this type of market. The paper simulates four different competition structures for App Developers in the two-sided market of Operating Systems for Smartphones: no App Developers, a Monopoly Developer, Perfect Competition between Developers and Exclusive Dealing of one Developer with an Operating System. It shows that the market structure that gives the highest welfare benefits is the one where App Developers compete in prices at the perfect competition level, while Operating Systems compete in a differentiated product Bertrand Competition game. Also, it proves that a market with App Developers gives higher Consumer Welfare.

1 Introduction

Ever since the release of the first iPhone in 2007, the smartphone market has grown rapidly, reaching a revenue of 52.9 billion (US Dollars) for the quarter ending on April 1, 2017, compared to a revenue of 50.6 billion (US Dollars) in the same quarter a year ago (Apple, 2017). Conversely, in 2009, the first Samsung smartphone was released[1]. It also shows a revenue of 44.72 billion US Dollars[2] for the quarter ending on March 31, 2017 (Samsung, 2017). With the smartphone industry's increase, important regulators started to pay close attention to the possible violation of competition laws.

On April 20th, 2016 the European Commission started an antitrust investigation into Google and its Android operating system. The investigation found evidence that "Google obliges manufacturers, who wish to pre-install Google's app store for Android, Play Store, on their devices, to also pre-install Google Search, and set it as the default search provider on those devices" (Comission, 2016). It is important to note that this is not the first and only investigation regarding operating system competition. This fact altogether with the growth rate of the industry in the last few years, makes competition analysis in this market more interesting.

An important concept in this field is network externalities. These externalities are the ones related to the effect on the valuation of a good, given by the usage of another user or agent (Martin, 2010). There are two types of network externalities: direct and indirect. The first one is associated with the cases where the value of the product increases with the number of consumers using the product. A good example of this type of externality is LinkedIn. As more companies become involve in its network, the value for job searchers increases. Indirect network externalities are more linked with goods or services that are not valuable without complements. For example, a Blu-ray disc player is only useful if there are products in the market recorded in Blu-ray format. In general,

[1] Although the first smartphone, the Dream, incorporated the Linux-based android operating system, in this analysis I consider the present day Android OS smartphones, such as the popular Samsung S series, as they are the direct competitors of Apple nowadays.

[2] 50.55 KRW trillions.

Operating Systems are a clear example of network externalities: the wiliness to pay of consumers increases as the number of available apps supported by the platform increases.

Multi-sided markets or multi-sided platforms, are markets where network externalities play a fundamental role. These type of markets are usually characterized by the presence of two or more sides, whose benefits are driven by the interaction through a platform (Rochet & Tirole, 2003). Hagiu and Wright (2015) defined that "The most common approach to date has focused on the presence of important cross-group or indirect networks effects between the two or more customers groups participating on the platform".

As Hagiu (2004) stated "A market is said to be two-sided if firms serve two distinct types of customers, who depend on each other in some important way, and whose joint participation makes platforms more valuable to each"[3]. A classic example of two-sided markets are videogame platforms such as Sony Play Station, Nintendo Wii and Microsoft Xbox. In this case, the platform is in charge of making the interaction between videogame developers and gamers easier. Interestingly enough, this type of platform has an additional characteristic: videogame developers are effectively dependent on the platform owners in order to retail their product to consumers[4].

In this paper I consider a two-sided model to analyze differences in competition models of the smartphone industry, specifically on operating systems such as iOS and Android. The operating system industry, in particular for smartphones, is one of the best examples in this file: OS connects mobile phone manufacturers, end-users, app developers, advertisement companies and more.

It is important to recall that most of the studies about two-sided markets are based on comparisons between competition models between the platforms, instead of one of the branches for the multi-sided market. In other words, literature tries to explain different competition models in the core of the market instead of one of the sides. Therefore, even with big contributions to the literature related to multi-sided platforms, as the one made by Jean-Charles Rochet and Jean Tirole through their papers [5] called "Platform Competition in Two-Sided Markets" and "Two-Sided Markets: A Progress Report", the gap in the literature regarding the analysis of the competition on one of the sides of the market remains.

This paper aims to analyse different market structures, in a context of operating systems, for app developers and answer which is welfare enhancing. Therefore, the main objective of this paper is to analyze the two-sided market of Operating Systems for mobile phones. I will study a model of interaction between

[3]This definition is alligned with the definitions used by other authors. For example, Rochet and Tirole (2003) defined two-sided markets as "markets in which one or several platforms enable interactions between end-users, and try to get the two(or multiple) sides "on board" by appropriately charging each side".

[4]As it was explained in Rochet and Tirole (2006) there are a lot more examples of two-sided markets such as portals, TV networks, newspapers, payment card systems and operating systems (Microsoft, Apple, Novell, IBM, Sun, etc...).

[5]Both papers were written by the two authors named.

three economic agents: App Developers [6], Operating Systems, and Consumers or customers. I will compare four different competition situations:

- A market without app developers. Only Operating Systems and consumers.
- A monopolistic app developer that develops apps for both platforms.
- Two app developers, each serving both markets (Perfect Competition).
- One monopolistic app developer that serves exclusively OS_1 (Exclusive Dealing).

For this purpose, a python simulation and a theoretical model will be used.

This paper is organized as follows: Section 2 gives a literature review about multi-sided (two-sided) market analysis. Section 3 presents the modelling framework with the general assumptions. It has 4 subsections, each corresponding to one of the competition models explained above. Section 4 explains how all three models were solved and programmed in Python together with the final results. Finally, Section 5 summarizes my main conclusions with respect to the results found under Section 3 and Section 4.

[6] In this paper they will be called Developers.

2 Literature Review

This paper belongs to a quickly growing economics literature on two-sided platforms model, initiated by Parker and Van Alstyne, Rochet and Tirole, Hagiu and Armstrong. In this section, I am going to go deep into the main studies about multi-sided markets and how this paper differs from the already available literature.

Parker and Van Alstyne were pioneers in the analysis of network effects to explain the behaviour of Software Platforms. They showed that contrary to the expected behaviours of firms avoiding Bertrand competition, it is possible to find a set of optimizing prices with one of them below marginal cost, because it will increase demand across markets[7] (Parker & Van Alstyne, 2000). One of their main conclusions is that the level of externality benefit is crucial to the determination of the price structure. Under high levels, "[...] the market that contributes more to demand for its complement is the market to provide with a free good." (Parker & Van Alstyne, 2000)[p. 1503], for lower levels, firms will keep one price low but both positive[8].

The model proposed by Rochet and Tirole focuses on the credit card market. The authors stated the importance of a non-traditional analysis for multi-sided markets. They claimed that one of the main differences and focus of their paper is the fact that multi-sided platforms do not choose price levels, instead, they choose price structures. With respect to that price structure, they emphasized on the fact that most of the multi-sided platforms based their business models on getting profits from one side of the market and leaving some losses on the other. Indeed, in Hagiu (2004) the author based on an empirical survey by Evans, Hagiu, and Schmalensee, stated that software platforms choose a price structure that allows them to "subsidize or earn little if any profits on the developer side of the market", and charge unreasonably high prices to users in order to get all of their profits. Rochet and Tirole (2003) also stated the importance of multihoming[9] on one side of the market in order to define competitive prices: "multihoming on one side intensifies price competition on the other side..." (Rochet & Tirole, 2003).

In the two-sided market literature, we can find two types of externalities: usage and membership externalities. The first of these arises from usage decisions. In the smartphone market context, users derive utility from using the operating system[10] on their smartphone rather than a PC, and developers also get utility from the fact that users are linked to the operating system. On the other hand, membership externalities refer to the situation when users on one side of the market derive strictly positive surplus from the interaction of an extra member

[7]The term "demand across markets" refers to the situation of the interactive demands in a multi-sided platform. However, they didn't use the term multi-sided markets during their paper. In contrast, they refer as inter and intra-market externalities.

[8]Rochet and Tirole also concluded that if attracting one side, through lower prices, generates externalities on the other side, then it is a good strategy(Rochet & Tirole, 2006).

[9]In smartphones market, multihoming implies that users can get access to more than one operating system. In general, a clear example of multihoming users is the credit card market: users are multihoming when they hold more than one type of credit card (Visa and Maestro).

[10]In this case, operating system is considered the platform.

on the other side of the market. In this paper, I am only focusing on usage externalities.

It is interesting to recall that while Rochet and Tirole (2003) limit their analysis to a mathematical model with pure usage externalities, Rochet and Tirole (2006) augmented their initial model considering both usage and membership externalities. The structure used in both papers to develop the mathematical model is similar. In both papers, authors focused exclusively on the effect of variation in competition model or market structure for the platforms. Nevertheless, the way in which the authors approach the variations are different: the first paper considers the case of a monopoly platform, competing platforms, and linear demands, while the second paper adopts a more complicated approach considering a basic model with or without payments between end-users.

Hagiu (2004) took a different approach. He chose to study how the optimal price structure appears for a two-sided market with a continuous and large number of app developers. He did this by looking at how the interaction between buyers and sellers was affected under a strong preference for variety, by buyers, and competition between sellers. His model concluded that when users have strong preferences for diversity, the optimal price structure is the one that allows the firm to make more profits on third-part producers instead of users[11].

The results of this paper rely on three main assumptions: Firstly, there is not vertical differentiation. Secondly, all applications are solely differentiated by fixed costs of production and lastly, there are not variables fees charged by the platform. He defined the timing for the pricing game as follows: First, the platform sets a price for users and developers simultaneously. Subsequently, both sides of the market make their adoption decision and following, developers set a price for consumers and they decide which app to buy. He also analysed different platform pricing structures for different types of two-sided markets, placing emphasis on the fact that depending on the market the optimal price structure can be radically different[12]. Finally, Hagiu investigated the effect of proprietary platforms against open platforms in the context of a social efficiency analysis. He concluded that the inefficiency of an open platform on developers' side, depends entirely on business stealing and product diversity effects(Hagiu, 2004).

A more recent paper in this field is Armstrong (2006). He considered three models: a monopoly platform, platform competition with single-homing users, and a competitive bottlenecks model[13]. According to Armstrong there are three main determinants of equilibrium prices: the magnitude of the cross-group externalities, the condition on the fees[14] and the condition of single or multi-homing for users.

[11] Hagiu makes an interesting remark, concluding that this optimal pricing structure is more applicable in video games market than in software markets.

[12] He considered 3 markets: software platforms, video games, and digital media. The first one, according to the data presented by the authors chooses to charge 0 on developers' side and get all their profits from users. The Second one is the other way around and the last one is something in between the other 2 strategies.

[13] This is the case of users with the possibility of multi-home and also users deriving more utility from network benefits than considering the cost they have to pay.

[14] Levied or lump-sum.

Cross-group externalities are defined as the situations where the "benefit enjoyed by a member of the group, depends upon how well the platform does in attracting custom from the other group" (Armstrong, 2006). Therefore, the first price determinant can be analysed as follows: if the platform is in a situation where a member of one group gets a large utility from each member on the other side of the market, the platform should get more profits from that group. The second condition can be summarized as the fact that cross-group externalities become weaker if the platform is charging per-transaction charge[15]. The last condition focuses on the fact that for example, platforms under single-homing users' environment have more incentives to compete and therefore profits are lower.

Each of these papers contribute to the theoretical framework used in this paper. The innovation with respect to the papers by Rochet and Tirole, Armstrong and Parker and Van Alstyne is the introduction of an analyzed focus on the competition of one side of the market instead of the platforms itself. Further, I consider a simple Bertrand model between 2 platforms and variations in the App Developers competition. The innovation of my model in relation to the paper by Hagiu is more complicated. Hagiu also focuses his analysis on competition on one side, however, he considers consumers with the same valuation for the good, while I consider different valuations randomly selected for each consumer. Another innovation of this model is in the price timing: this model considers a timing where consumers first decide which operating system to buy, and then they decide if they buy or not the app offered on the other side of the market. The last innovation of the model developed in this paper is the variable quality of operating systems. They are not only competing to set a price in the market but also need to set up an optimal quality level that will attract more users.

[15]This is because a proportion of the benefit that users are getting from the interaction is being reduced by the extra payment.

3 Modeling framework

The general theoretical model considers 2 Operating Systems[16]. This assumption is based on the fact that market share of Operating Systems for mobile phones is reported to be in the first quarter of 2017, 86.1% for Android, 13.7% for iOS and 0.2% for others (Statista, 2017). This gives us an insight that the Operating Systems market is mainly shared between two big firms. In fact, as explained before, we look specifically at Apple OS iOS and Google OS Android. We study a large finite number of consumers on one side of the market and 1 or 2 Developers, depending on the model, on the other side of the market. For simplicity, I assume single-homing consumers[17].

We are interested in modeling a two-sided market for Operating Systems. For the purpose of analyzing different competition models, as stated in the research question, we will model 4 different scenarios for competition in this market. The theoretical model combined with the Python Simulation, allow us to find results related to optimal decisions, giving enough tools to compare different scenarios and make conclusions about welfare optimality. Each model will be explained in its respective subsection in this paper. However, we will start with the general assumptions common to all models.

The theoretical model is based in Hagiu (2004). It is important to consider that given the specific market studied in this paper, variety effect does not play an interesting role, as Hagiu stated "... there are good reasons to believe that user demand for application variety is higher for video games than for productivity-oriented or professional software (for computers, PDAs, smartphones or other electronic devices)".

Consumers are represented in the equations by the subindex $i \in [1, n]$ with a total number of consumers equal to n. Considering only two Operating Systems, they are represented by $j \in (1, 2)$. Finally, the subindex k shows Developers equations[18].

With the purpose of keeping the analysis tractable and simple, the functional form of the utility for consumer i purchasing operating system j and app produced by Developer k is given by:

$$U_i(q_j^{os}, p_j^{os}, p_k^a) = \theta_i + \theta_i q_j^{os} - p_j^{os} + max(0, \alpha_i - p_k^a) \tag{1}$$

Where

$$\theta \sim \mathcal{U}(0, 1)$$

$$\alpha \sim \mathcal{U}(0, 1)$$

[16] In this paper we are going to refer to Operating Systems as OS.

[17] It means that consumers can exclusively use one platform, in other words, each consumer can buy one OS solely.

[18] The number of Developers will vary between models, therefore the limits for subindex k will be specified in each subsection.

and consumer i has a θ_i[19] and α_i that parametrizes its wiliness to pay for the OS and the app respectively[20]. Furthermore, q_j^{os} shows the quality level of the OS j. The price charged by OS j is represented according to p_j^{os}. Finally, the last term on the utility function of consumers represents the utility derived from the consumption of the app, when the consumer buys it from Developer k.

Utility function specified in this model satisfies feasible features for the Operating System's market. If consumer 1 has a willingness to pay of θ_1 higher than the willingness to pay of consumer 2 θ_2, for a given level of quality consumer 1 is willing to pay a higher price. It is not misleading to assume that Consumers' utility increases with higher quality levels set by OS[21]. A consumer with a low willingness to pay does not valuate the quality as a consumer with a high willingness to pay[22]. Therefore, the return of an increase in OS quality is mathematically multiplied by the willingness to pay of each consumer: higher θ implies higher returns of higher levels of quality. Finally, since prices are a cost for consumers, their utility decreases with both prices. It is important to specify that in the market, consumers cannot buy the app without buying one OS. I assume that consumers incur no fixed usage cost.

Developers face fixed cost F related to the cost incurred in the app development process. Without loss of generality, for simplicity I normalize the variable costs for Developers to 0. Developers, even in the model of monopoly, do not have the market power needed to charge a fee to the OS it is linked with[23]. Developers do not take into account the change in composition of the demand given by the Operating Systems. It means that they cannot change the price set after seen the price and demand for Operating Systems[24].

Operating Systems are assumed to have symmetric costs structure defined according to

$$c(q_j^{os}) = \frac{1}{2}(q_j^{os})^2 \tag{2}$$

where q_j^{os} is the quality level set by OS j. $c(q_j)$ is increasing and convex. It means that even when a firm has the incentive to set a higher quality level, to attract more customers, each increase in the quality level increases $\frac{1}{2}(q_j^{os})^2$ the costs of the firm j. I assume that Operating Systems do not face any cost from the interaction between Developers and Consumers. Apple, for example, indirectly forces consumers to get new phones in order to update their OS and therefore to have access to new platforms. In this sense, Apple does not need

[19] In fact, if the optimal decision for consumer 1 with θ_1 is to purchase the OS, independently on which of the two, all consumers with $\theta_i \geq \theta_1$ with $i \neq 1$ will purchase from one of the two OS.

[20] θ and α are independently distributed.

[21] A higher level of quality in the Operating System market, implies better features in consumers' phones. Therefore consumers get more utility from phones with better features.

[22] We can consider consumers with low levels of θ as people that do not care about technology, they want an OS but they are not willing to spend a lot of money in such a good despite the quality level.

[23] It does not mean that the Developer cannot charge a price above the costs.

[24] If developers do not behave in this way, they should consider that consumers with higher α_i are more likely to purchase the app and then charge a higher price. In general, they should set a different price as the composition of the α ex post is different given the condition of pre-purchase of an OS in order to buy the app.

to incur in any maintenance costs from updating a platform.

OS and Developers have a standard profit function that will vary between models, and therefore will be explained in each subsection.

Competition model between Operating Systems holds for all models, with variations in the number of Developers associated with each OS. In general, the decision of consumers of purchasing OS_1 or OS_2 is related to the utility each OS generates to the consumer. First of all, consumers only purchase an OS if it gives them a positive utility, therefore:

$$U_i(q_j^{os}, p_j^{os}, p_k^a) \geq 0 \tag{3}$$

$$j = 1, 2$$

$$k = 1, 2$$

where p_k^a is the price of the app that will variate in each model, and q_j^{os}, p_j^{os} are the quality and price associated to OS_j with $j = 1, 2$. If equation (3) is satisfied for both Operating Systems , consumers' purchasing decision is made by a comparison between utilities. I assume a tie-breaking rule of consumers buying from OS1 in a case of a tie. Then the purchasing decision can be expressed as:
Purchasing OS_1 if

$$U_i(q_1^{os}, p_1^{os}, p_k^a) \geq U_i(q_2^{os}, p_2^{os}, p_k^a) \tag{4}$$

Purchasing OS_2 if

$$U_i(q_2^{os}, p_2^{os}, p_k^a) > U_i(q_1^{os}, p_1^{os}, p_k^a) \tag{5}$$

It is important to consider that this purchasing decision is the base of the demand function. In each subsection I will define demand function of consumers for the correspondent case. Demand changes as a result of different competition games in the model. It is important to consider that demand functions define in this section, are the total demand that each firm should consider to set an optimal price.

Finally, we define welfare of the economy as the sum of:

$$W = \sum_{j=1}^{2} \pi_j^{os} + \sum_{k=1}^{2} \pi_k^a + \sum_{i=1}^{N} (U_i = \theta_i + \theta_i q_j^{os} - p_j^{os} + max(0, \alpha_i - p_k^a)) \tag{6}$$

where N is the total number of consumers in the market, π_j^{os} is the profit for Operating System j and π_k^a is the profit for Developer k. As explained before, the number of Developer changes in some models, with a maximum number of 2 Developers.

All models consist of 5 stages:

1. Operating Systems set quality.
2. Operating Systems set prices for consumers.
3. Consumers decide to buy OS.

4. Developer set price for consumers.

5. Consumers decide to buy App, given the previous purchase of the OS.

Operating systems compete in a Bertrand game during stage 1 and 2. Given the condition that Consumers can only buy the app if they acquired an OS in advanced, Developer's price decision is established after Consumers purchasing decision to buy the OS. In some models, Developers compete in prices during stage 4. Under section 4, we can find the solution of this game given by a backward induction procedure programmed in Python.

3.1 Model 1: No app Developers.

For the first model, we consider the case of 2 Operating Systems and Consumers. In this case, there are not Developers in the market. Figure 1 shows the willingness to pay for OS θ_i. The line in figure 1, represents all consumers in the

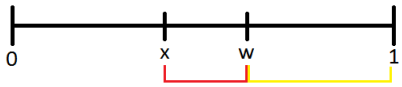

Figure 1: Demand for Model 1

market. Consumers demand when the utility attained from any of the OS is positive as stated in equation (3). The decision of buying from OS_1 or OS_2 is determined by equation (4) or (5). Based on the market analysis, we know that demand for OS_1 is given by

$$D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = 1 - \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}}$$

Demand for OS_2 is given by

$$D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}} - \frac{p_2^{os}}{1 + q_2^{os}}$$

The proof is found under Appendix A-1.

As explained under general modelling framework, Operating Systems face standardized profit function. Operating Systems' profits are defined as follows

$$\pi_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os})(p_1^{os}) - \frac{1}{2}(q_1^{os})^2 - F \tag{7}$$

$$\pi_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os})(p_2^{os}) - \frac{1}{2}(q_2^{os})^2 - F \tag{8}$$

On the other hand, Operating Systems have a cost structure that only depends on the quality level. As explained in general modelling framework, Operating Systems do not face any cost derived from the interaction between Developers

and Consumers.

Since in this model we do not consider any Developer, the timing of the model is reduced to only the first 3 stages.

3.2 Model 2: Monopoly.

In this model, we consider the entrance of a third agent in the market: one Developer, holding the assumptions established before for OS and Consumers. Figure 2 shows the interaction between both wiliness to pay of each consumer: α_i and θ_i.

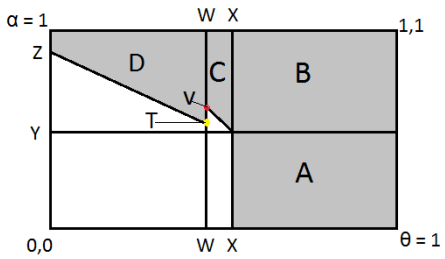

Figure 2: Demand for Model 2

Total demand of the market for App and Operating Systems is represented by the areas A+B+C+D. Areas are defined as follows

$$
\begin{aligned}
Area(A) &= (1-x)(y) \\
Area(B) &= (1-x)(1-y) \\
Area(C) &= (x-w)(1-v) + \frac{1}{2}(x-w)(v-y) \\
Area(D) &= \frac{1}{2}(z-T)(w) + (1-z)(w)
\end{aligned}
$$

Demand for Developer is given by

$$
\begin{aligned}
D^a(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = &\left(\frac{p_1^{os}}{1+q_1^{os}} - \frac{p_1^{os}-p_2^{os}}{q_1^{os}-q_2^{os}}\right)\left(1-\frac{1}{2}\left(2p^a + p_1^{os} - \frac{(p_1^{os}-p_2^{os})(1+q_1^{os})}{q_1^{os}-q_2^{os}}\right)\right) \\
&+ \frac{p_1^{os}-p_2^{os}}{q_1^{os}-q_2^{os}}\left(1-\frac{1}{2}\left(2p^a + 2p_2^{os} - \frac{(p_1^{os}-p_2^{os})(1+q_2^{os})}{q_1^{os}-q_2^{os}}\right)\right) + \left(1-\frac{p_1^{os}(1-p^a)}{1+q_1^{os}}\right)
\end{aligned}
$$

Demand for OS_1 is given by

$$
D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = 1 - \frac{p_1^{os}}{1+q_1^{os}} + \left(\frac{p_1^{os}}{1+q_1^{os}} - \frac{p_1^{os}-p_2^{os}}{q_1^{os}-q_2^{os}}\right)\left(1-\frac{1}{2}\left(2p^a + p_1^{os} - \frac{(p_1^{os}-p_2^{os})(1+q_1^{os})}{q_1^{os}-q_2^{os}}\right)\right)
$$

Demand for OS_2 is given by

$$
D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = \frac{p_1^{os}-p_2^{os}}{q_1^{os}-q_2^{os}}\left(1-\frac{1}{2}\left(2p^a + 2p_2^{os} - \frac{(p_1^{os}-p_2^{os})(1+q_2^{os})}{q_1^{os}-q_2^{os}}\right)\right)
$$

The proof is found under Appendix A-2.

It is important to consider that Developer is not constraint by consumers' decision of purchasing OS[25]. As explained under general modelling framework, Developer and Operating Systems face standardized profit function. Developer's and Operating Systems' profits are respectively defined as follows

$$\pi^a(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = D^a(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a)(p^a) - F \tag{9}$$

$$\pi_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_1^{os}, p^a)(p_1^{os}) - \frac{1}{2}(q_1^{os})^2 \tag{10}$$

$$\pi_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a)(p_2^{os}) - \frac{1}{2}(q_2^{os})^2 \tag{11}$$

In this case, Developer in the market only faces fixed cost of developing the app, and since there is only one Developer in the market, it does not have to pay any price to the OS in order to get access to its platform. For both Operating Systems is convenient to give free access to the Developer in the market, because some consumers have really strong preferences for the App that even with high prices of the OS, they are willing to buy the OS only for the App.

On the other hand, Operating Systems have a cost structure that only depends on the quality level. As explained in general modelling framework, Operating Systems do not face any cost derived from the interaction between Developers and Consumers.

3.3 Model 3: Perfect Competition.

For the third model, we consider the case of two Developers holding the assumptions established before for OS and Consumers. Developers compete in a Bertrand game and both apps are assumed to be homogeneous goods. Both Developers can be linked to both Operating Systems. However, it is important to remember that consumers are single-homing in all goods in the market, therefore it is not possible for them to buy both apps at the same time. In figure 3 we can see the interaction between both wiliness to pay of each consumer: α_i and θ_i.

Total demand of the market for App and Operating Systems is represented by the areas A+B+C. Since price for App developers is equal to zero, given the Bertrand Competition with homogeneous goods, for simplicity, we do not consider p^a in demand functions. Areas are defined as follows

$$Area(A) = (1-x)(1)$$
$$Area(B) = (x-w)(1-v) + \frac{v}{2}(x-w)$$
$$Area(C) = (1-z)(w) + \frac{w}{2}(z-v)$$

[25]It can also be seen in the backward induction of the game.

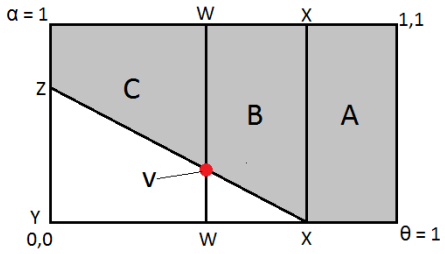

Figure 3: Demand for Model 3

Demand for OS_1 is given by

$$D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = 1 - \frac{p_1^{os}}{1+q_1^{os}} + \left(\frac{p_1^{os}}{1+q_1^{os}} - \frac{p_1^{os}-p_2^{os}}{q_1^{os}-q_2^{os}}\right)\left(1 - \frac{1}{2}\left(\frac{p_2^{os}(1+q_1^{os}) - p_1^{os}(1+q_2^{os})}{q_1^{os}-q_2^{os}}\right)\right)$$

Demand for OS_2 is given by

$$D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = \frac{p_1^{os}-p_2^{os}}{q_1^{os}-q_2^{os}}\left(1 - \frac{1}{2}\left(\frac{p_2^{os}(1+2q_1^{os}-q_2^{os}) - p_1^{os}(1+q_2^{os})}{q_1^{os}-q_2^{os}}\right)\right)$$

Demand for App Developers is given by

$$\begin{aligned} D^a(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = 1 - \frac{p_1^{os}}{1+q_1^{os}} + \left(\frac{p_1^{os}}{1+q_1^{os}} - \frac{p_1^{os}-p_2^{os}}{q_1^{os}-q_2^{os}}\right)\left(1 - \frac{1}{2}\left(\frac{p_2^{os}(1+q_1^{os}) - p_1^{os}(1+q_2^{os})}{q_1^{os}-q_2^{os}}\right)\right) \\ + \frac{p_1^{os}-p_2^{os}}{q_1^{os}-q_2^{os}}\left(1 - \frac{1}{2}\left(\frac{p_2^{os}(1+2q_1^{os}-q_2^{os}) - p_1^{os}(1+q_2^{os})}{q_1^{os}-q_2^{os}}\right)\right) \end{aligned}$$

The proof is found under Appendix A-3.

As explained under general modelling framework, Developer and Operating Systems face standardized profit function. Developer's and Operating Systems' profits are respectively defined as follows

$$\pi^a(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = D^a(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os})(p^a) - F \tag{12}$$

$$\pi_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_1^{os})(p_1^{os}) - \frac{1}{2}(q_1^{os})^2 \tag{13}$$

$$\pi_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os})(p_2^{os}) - \frac{1}{2}(q_2^{os})^2 \tag{14}$$

3.4 Model 4: Exclusive Dealing.

Finally, in the last model, we consider the case of only one Developer, holding the assumptions established before for OS and Consumers. The only Developer in the market has an exclusive contract with OS_1. Figure 4 shows the interaction between both wiliness to pay of each consumer: α_i and θ_i. Total demand of the

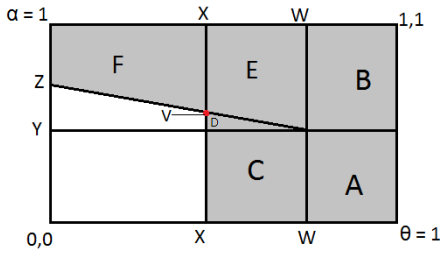

Figure 4: Demand for Model 4

market for App and Operating Systems is represented by the areas A + B +C + D + E + F. We can define Areas as

$$Area(A) = (1-w)(y)$$
$$Area(B) = (1-w)(1-y)$$
$$Area(C) = (w-x)(y)$$
$$Area(D) = \frac{1}{2}(w-x)(v-y)$$
$$Area(E) = (w-x)(1-v) + \frac{1}{2}(w-x)(v-y)$$
$$Area(F) = x(1-z) + \frac{1}{2}(z-v)x$$

Demand for OS_1 is given by

$$D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = 1 - \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}} \left(p^a + \frac{p_1^{os}}{2} \right)$$

Demand for OS_2 is given by

$$D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = \frac{1}{2} \left(\frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}} - \frac{p_2^{os}}{1+q_2^{os}} \right) \left(2p^a + p_1^{os} - \frac{p_2^{os}(1+q_1^{os})}{1+q_2^{os}} \right)$$

Demand for App is given by

$$D^a(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = 1 - p^a + \frac{1}{2} \left(\frac{p_2^{os}}{1+q_2^{os}} \right) \left(\frac{p_2^{os}(1+q_1^{os})}{1+q_2^{os}} - 2p_1^{os} \right)$$

The proof can be found under Appendix A-4.

As explained under general modelling framework, Developer and Operating Systems face standardized profit function. Developers' and Operating Systems' profits are defined according to equations (9), (10) and (11).

4 Python Simulation and results

4.1 Python Simulation

Each model explained in section 3, was simulated in Python. Every code is different from the other since the competition conditions change through the

models. Coding for every model can be found under Appendix A.

In general terms, each simulation was run with 1000 consumers. For models 2, 3 and 4, each consumer was generated by a randomization process according to a uniform distribution between 0 and 1, which gave a vector of 2 entries: the first one refers to θ and the second one to α. For model 1, this vector only has one entry since there is not any Developer in the market. α and θ are independent.

Buy function was defined as a vector of 2 entries in most of the cases. Each entry could only take the value of 0 or 1 since the product is discrete[26]. Demand function was normalized to a number between 0 and 1, given the values of the parameters θ and α. Therefore demand function shows the proportion of total customers that buy from each firm.

4.2 Results

Since the code generates different numbers for θ and α each time the code is run, readers who run the code might get different numbers to the ones presented in this subsection. However, differences are not significant and lead to the same conclusions.

Table 1 presents the results obtained in the Python Simulation[27] for each model. Cells with "-" imply that for the corresponding model, that cell does not take any value[28]. C_W = Consumer Welfare and $Welfare$ = Total welfare in the economy defined by equation (6). Profits for Developers presented in the table do not include the Fixed Cost that firms must incur when developing the app. Therefore, to be more exact each π_k^a is equal to the value reported in the table minus F.

In the first model, we can see that quality and price for both firms are relatively close. However, given the tie-breaking rule in favor of OS_1, we expect that firm 1 had a bigger demand as indeed happened. This is also consistent with the fact that OS_1 charges a higher price and quality than OS_2. Consequently, given this results π_1^{os} is larger than π_2^{os}. Since both firms have the same cost of investment in quality, given a higher demand for OS_1, it has higher incentives to invest in quality than OS_2. This is mainly due to the fact that for a given price, increases in quality attracts more consumers, and also the fact that OS_1 can use its demand advantage to charge a larger price. However, despite the difference in quality, we can see that profits for both firms are almost the same.

From Model 2 onwards, we have one new agent in the market: App Devel-

[26] Each consumer can only buy 1 OS or 1 App, never half OS or half App, etc...

[27] Due to problems with the code, π_2^{os} were calculated with the equations presented under section 3 instead of Python. The validity of this method is tested and corroborated in Appendix C.

[28] In fact, Model 1 does not have any app in the market. Model 2 and 4 only have one app, and it is results are called according to p_1^a and π_1^a.

Table 1: Results Python Simulation

Variable	Model 1	Model 2	Model 3	Model 4
D_1^a	-	0.4490	0.4655	0.3944
D_2^a	-	-	0.4655	-
D_1^{os}	0.3699	0.6423	0.9116	0.6752
D_2^{os}	0.3222	0.1087	0.0195	0.0296
p_1^a	-	0.4743	0.000	0.4986
p_2^a	-	-	0.000	-
q_1^{os}	0.4243	0.1995	0.2352	0.2531
q_2^{os}	0.3159	0.1615	0.0128	0.0171
p_1^{os}	0.4734	0.4356	0.4129	0.4885
p_2^{os}	0.4051	0.4222	0.4058	0.3853
π_1^a	-	0.2523	0.000	0.2648
π_2^a	-	-	0.000	-
π_1^{os}	0.0875	0.2598	0.3453	0.2996
π_2^{os}	0.0806	0.0329	0.0078	0.0112
C_W	0.2623	0.3334	0.7153	0.3173
$Welfare$	0.4304	0.6267	1.0684	0.6281

opers. Developers, as explained before, set a price in a myopic way. In model 2 we only consider one Developer and therefore, we expected a $p^a = 0.5$ which is consistent with the result obtained[29] $p^a = 0.4743$.

Model 2 represents the first model with 3 agents in the market: Developers, Operating Systems, and Consumers. We can see that adding an extra agent in the market, makes quality levels dropped drastically. The quality level in this model is around one-half of the quality set in Model 1. This is due to the fact that Consumers purchasing decision is now based on the consumption of two goods: OS and App. Therefore Operating Systems have less incentive to invest in quality levels: increases in quality levels are not directly related to increases in demand in this model given the complementary good App. In this model, we can also see an increase in consumer welfare mainly due to the new agent in the market.

Given the cost structure for Developers in model 3 and the Bertrand Competition model with homogenous goods, prices for both developers, as expected, are equal to marginal cost, in this case, 0. This leaves Developers with 0 prof-

[29]Since the market condition for the app exclusively its limited to

$$\alpha \geq p^a$$

Developer assume that his profits are given by

$$D^a(p^a) = (1 - p^a)p^a - F$$

Optimizing Developers' price leads to $p^a = 0.5$. It is important to consider that this demand is different to the one defined under section 3 for Model 2 because here we only consider the demand the Developer sees. This is the case because in Model 2 we define demand considering the price and quality for the Operating Systems, or in other words, we define demand considering the demand for Operating Systems.

its[30].

In model 3 we can see that the increase of a second Developer, creates an incentive for OS_1 to invest more in quality levels than in the model before, however it creates the opposite incentive for OS_2 which continue dropping its quality level. As explained before, Developers charge a price of 0 which means that given the tie-breaking rule, despite consumers preferences, most of them are demanding from OS_1. This means that all consumers with positive α_i are going to demand the app and therefore, as in Model 1, increases in quality levels are directly related to increases in demand. The asymmetric demand makes this effect only significant for OS_1.

Operating system 1 obtains it highest profits under this model because the competition situation allows OS_1 to get the maximum market power possible with respect to the 4 models presented in the table. At the same time, Operating System 2 gets the lowest profits possible because of big advantage that the tie-breaking rule is giving to OS_1 and the low market power that Developers have.

In the last model, we can see that most of the results found in the other 3 models hold. First, the price of the only Developer in the market is again close to 0.5, as it was predicted and explained by model 2. The developer is getting approximately the same level of profits as under model 2, even though in this case it has an exclusive dealing with OS_1. This is because of the myopic assumption for Developers; they always optimize its demand in the same way despite the competition model they are facing. Therefore we can expect that models with only one Developer will lead to similar prices and profits. As a result, Developer's profit in this model is 0.01 points higher than in model 2 with a price 0.0243 points higher than model 2.

With respect to consumer welfare, we can see that in models with more economic agents, consumers are always better. In fact, under model 3 the one with the highest number of economic agents, we consider 2 App Developers and 2 Operating Systems, Consumers get the highest Welfare possible. When comparing model 1 and 2, the majority of consumers who have low willingness to pay for the app, let's say $\alpha_i < p^a$, are worse off in the scenario where App is available. This can be seen by comparing the utility these consumers will get under both model 1 and model 2. According to the utility function and assuming $q^{os}_{1,m1}$ and $p^{os}_{1,m1}$ as the quality and price for OS_1 in model 1, comparing with model 2 we know that all consumers with

$$\theta > \frac{p^{os}_{1,m1} - p^{os}_{1,m2}}{q^{os}_{1,m1} - q^{os}_{1,m2}}$$

are worse off in model 2.

Market conditions for Operating Systems are asymmetric even when both firms are ex-ante symmetric. The asymmetry of the market and the slight advantage of OS_1 makes OS_1 to be the one with the highest quality level in the market in

[30]Even negative considering the fixed cost F.

all models.

It is interesting to notice, that in Model 1 we have the highest quality level but only the second highest price. This is due to the fact that under Model 4, the one with the highest price, OS_1 has more market power in the way that it has an exclusive contract with the App Developer. OS_1 takes advantage of the extra demand and charges a higher price in the market, even with one of the lowest quality levels set by OS_1. In Model 4, OS_1 does not need to invest in a high-quality level to attract more consumers, because he already has the tie-breaking rule and also the exclusive dealing to put him in an advantageous position with respect to his competitor OS_2.

5 Conclusions

It is important to recall that the simple framework implemented in this paper, without loss of generality, can help us better understand a more complex multi-sided market. Therefore, all analysis and conclusions made in this paper can be implemented into multi-sided platforms.

The highest prices charged in the market are set under exclusive contract (Model 4) by OS_1. Since in this case Operating System 1 has an exclusive dealing with the only Developer in the market, it allows him to take advantage of the demand for the app and therefore set a higher price. OS_1, in this case, has an extra demand given by the consumers with high willingness to pay for the app that is willing to buy the OS only for the utility attained from the app.

Consumers, in general, get the same surplus under monopoly (Model 2) and exclusive contracts (Model 4). App Developers also get the same level of profits under model 2 and 4. This can be explained because both models consider similar competition structures: in both, there is only one Developer and it's setting its price p^a under the same competition model.

Economic theory suggests that perfect competition leads to the highest consumer surplus and the lowest profits for firms. Results for perfect competition model (Model 3) support this theory. Since App developers are setting prices equal to marginal cost, as in perfect competition model, they get the lowest profit possible $\pi_k^a = 0$ and Consumers at the same time get the highest surplus possible. Under the same theory, model 1 shows the lowest surplus for Consumers. This result can be due to the fact that in model 1 there is less competition in the market. Since there are not Developers, Operating Systems have more market power.

For the same conditions explained before, under models 2 and 4 Developers get more profits than under Model 3. This is because, under models 2 and 4, Developers have some market power that allows them to charge a price larger than their marginal cost. These prices allow them to get strictly positive profits. However, it is interesting to notice that the myopic condition for App Developers, always makes them overestimate their demand and therefore they expect higher profits that the ones thy actually get.

The analysis of the best competition model for Operating Systems is the interest in this paper. Operating System 1 gets always an advantage over Operating System 2 given the tie-breaking rule defined for the simulation. In fact, profits for Operating System 1 are always larger than profits for Operating System 2.

In model 3, where Developers are in perfect competition scheme, Operating System 1 gets the highest profits possible. However, these profits do not show a big difference with respect to the ones obtained under model 4. The tie-breaking rule combined with $p^a = 0$ creates the scenario where OS_1 have more market power. A $p^a = 0$ itself makes attracting consumers for OS_1 easier. The exclusive dealing contract in model 4, also allows OS_1 to get market power.

Operating System 2, gets its highest profits under the model without app developers (Model 1), where both firms hold market power, and there are fewer competitors in the market, no app developers. Once more the tie-breaking rule in favor of 1, does not allow Operating System 2 enjoy the market power at the same level that OS_1 does. Therefore, in the model that there is not untie rule, OS_2 gets the maximum profits available.

Prices for Operating Systems are relatively stable, but always in favor of OS_1, through the models. However, quality drops drastically with the introduction of a new agent in the market because it reduces Operating Systems' incentives to invest in higher quality levels. As can be seen in Table 1, after App Developers are incorporated in the market quality levels set by both OS drops drastically. This is due to the fact that the purchasing decision is not only based on the utility derived from OS but also in the utility from the App. Then, increasing quality does not necessarily mean more customers.

After all the analysis made before, we find that the existence of separate app developers with no market power leads to the best Social Welfare. This result can be explained by the economic theory that states that there is higher social welfare as there is less market power. In fact, model 3 is the only model presented where we consider one of the sides of the market, setting prices as if firms on that side of the market where in perfect competition. In this model, the Developers' side. We also find that the introduction of App Developers to the market increases consumers' welfare and total welfare.

References

Apple. (2017). *Apple reports second quarter results.* Retrieved 2017-05-18, from `https://www.apple.com/newsroom/2017/05/apple-reports-second-quarter-results/`

Armstrong, M. (2006). Competition in two-sided markets. *The RAND Journal of Economics*, *37*(3), 668–691.

Comission, E. (2016). *Antitrust: Commission sends statement of objections to google on android operating system and applications – factsheet.* Retrieved 2017-03-17, from `http://europa.eu/rapid/search.htm/`

Evans, D. S., Hagiu, A., & Schmalensee, R. (2005). A survey of the economic role of software platforms in computer-based industries. *CESifo Economic Studies*, *51*(2-3), 189–224.

Hagiu, A. (2004). Two-sided platforms: Pricing and social efficiency.

Hagiu, A., & Wright, J. (2015). Multi-sided platforms. *International Journal of Industrial Organization*, *43*, 162–174.

Martin, S. (2010). *Industrial organization in context.* Oxford University Press.

Parker, G. G., & Van Alstyne, M. W. (2000). Internetwork externalities and free information goods. In *Proceedings of the 2nd acm conference on electronic commerce* (pp. 107–116).

Rochet, J.-C., & Tirole, J. (2003). Platform competition in two-sided markets. *Journal of the european economic association*, *1*(4), 990–1029.

Rochet, J.-C., & Tirole, J. (2006). Two-sided markets: a progress report. *The RAND journal of economics*, *37*(3), 645–667.

Samsung. (2017). *Samsung electronics announces first quarter results.* Retrieved 2017-05-20, from `https://news.samsung.com/`

Statista. (2017). *Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 1st quarter 2017.* Retrieved 2017-06-02, from `https://www.statista.com/`

A Demand Functions

A.1 Model 1

First, to calculate the demand function of consumers we need to recall equations (3), (4), (5) and Figure 1. In this model, we first need to define the minimum value of θ_i that a consumer needs to have in order to buy any OS. This point is given by equation (3). The results from the Python simulation suggest that point x is given by equation (3) evaluated for OS_2

$$x = \frac{p_2^{os}}{1 + q_2^{os}}$$

Point w is the minimum θ_i for which consumers are purchasing from OS_1 exclusively. Then, implementing equation (4) and the condition of no app developers in the market ($\alpha = 0$) we get

$$w = \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}}$$

Given the points defined above, the yellow area is the demand for OS_1 and the red area is the demand for OS_2.

$$D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = 1 - \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}}$$

Demand for OS_2 is given by

$$D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}} - \frac{p_2^{os}}{1 + q_2^{os}}$$

A.2 Model 2

First, to calculate the demand function of consumers we need to recall equations (3), (4), (5) and Figure 2. In the case of the purchasing decision with respect to the App, point y, is defined as the case when $\theta = 0$ and therefore utility function (1) is reduced to

$$U_i = max(0, \alpha_i – p^a)$$

which means that the purchasing condition of the app for consumers is defined as

$$\alpha_i – p^a \geq 0$$

Therefore, point y is defined as

$$y = p^a \tag{15}$$

Points w and x are the cases where $\alpha = 0$. Therefore utility function (1) is reduced to

$$U_i = \theta_i + \theta_i q_j^{os} - p_j^{os} \tag{16}$$

Point x defines the minimum wiliness to pay of consumers for which consumers purchase any of the Operating Systems. From equation (16) we know that the condition for a consumer to demand OS_j is

$$\theta \geq \frac{p_j^{os}}{1 + q_j^{os}}$$

According to python results explained under section 4, we know that

$$\frac{p_2^{os}}{1 + q_2^{os}} > \frac{p_1^{os}}{1 + q_1^{os}}$$

it implies that the minimum θ_i that a consumer needs to have in order to buy any of the Operating Systems without considering the app is given by

$$\theta \geq \frac{p_1^{os}}{1 + q_1^{os}}$$

which means that the minimum θ_i for which consumers are purchasing any OS is given by the point x and it is defined as

$$x = \frac{p_1^{os}}{1 + q_1^{os}} \tag{17}$$

Point w is the minimum θ_i for which consumers are purchasing from OS_1 exclusively. Given the reduced utility function specified by equation (16), with (4) and (16) we get

$$w = \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}} \tag{18}$$

Based on Python results, we know that in this case $w < x$.

Point z is the point where consumers have a high enough α that they are willing to buy the OS even when they get negative utility from buying only OS because they get positive utility in total counting with the utility attained from the App. Then, the point z is the minimum value of α_i that given $\theta_i = 0$ makes equation (1) positive. Since all consumers to the left of point w get more utility from OS_2 than OS_1 , point z is only considering quality and price of OS_2. From equation (1) and the explanation above, we get that z is defined as

$$z = p^a + p_2^{os} \tag{19}$$

Finally, point v and T are given by the minimum value of α_i that given a

$$\theta_i = \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}}$$

makes equation (1) positive for OS_1 and OS_2 respectively. Therefore,

$$v = p^a + p_1^{os} - \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}}(1 + q_1^{os}) \tag{20}$$

$$T = p^a + p_2^{os} - \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}}(1 + q_2^{os}) \tag{21}$$

Recalling figure 2 and equations (15), (17), (18), (19), (20) and (21) we can calculate the demand for each firm in the market. Areas are defined as follows

$$Area(A) = (1 - x)(y)$$

$$Area(B) = (1 - x)(1 - y)$$

$$Area(C) = (x - w)(1 - v) + \frac{1}{2}(x - w)(v - y)$$

$$Area(D) = \frac{1}{2}(z - T)(w) + (1 - z)(w)$$

Demand for Developer is given by

$$D^a(q_2^{os}, p_2^{os}, p^a) = Area(B) + Area(C) + Area(D)$$

$$D^a(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = \left(\frac{p_1^{os}}{1 + q_1^{os}} - \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}}\right)\left(1 - \frac{1}{2}\left(2p^a + p_1^{os} - \frac{(p_1^{os} - p_2^{os})(1 + q_1^{os})}{q_1^{os} - q_2^{os}}\right)\right)$$
$$+\frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}}\left(1 - \frac{1}{2}\left(2p^a + 2p_2^{os} - \frac{(p_1^{os} - p_2^{os})(1 + q_2^{os})}{q_1^{os} - q_2^{os}}\right)\right) + \left(1 - \frac{p_1^{os}(1 - p^a)}{1 + q_1^{os}}\right)$$

Demand for OS_1 is given by

$$D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = Area(A) + Area(B) + Area(C)$$

$$D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = 1 - \frac{p_1^{os}}{1 + q_1^{os}} + \left(\frac{p_1^{os}}{1 + q_1^{os}} - \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}}\right)\left(1 - \frac{1}{2}\left(2p^a + p_1^{os} - \frac{(p_1^{os} - p_2^{os})(1 + q_1^{os})}{q_1^{os} - q_2^{os}}\right)\right)$$

Demand for OS_2 is given by

$$D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = Area(D)$$

$$D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}}\left(1 - \frac{1}{2}\left(2p^a + 2p_2^{os} - \frac{(p_1^{os} - p_2^{os})(1 + q_2^{os})}{q_1^{os} - q_2^{os}}\right)\right)$$

A.3 Model 3

To calculate the demand function of consumers we need to recall equations (3), (4), (5) and figure 3. The purchasing decision with respect to the App, is reflected in point $y = 0$. This point shows that given the Bertrand Competition in Developers and the cost structure for Developers, we have that

$$y = p_a = 0 \tag{22}$$

Therefore, all consumers are willing to buy any app despite the θ level.

In this case, given the results obtained in Python, w is defined according to equation (18)

$$w = \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}} \tag{18}$$

In this case, the market condition needed to have demand is given by price and quantities of firm 1, which implies that equation (17) holds also in this model. In other words,

$$x = \frac{p_1^{os}}{1 + q_1^{os}} \tag{17}$$

Point z is calculated in the same way as it was calculated for model 1, now adding one more agent in the economy. Given equation (22) we get

$$z = p_2^{os} \tag{23}$$

Point v is calculated in the same was as it was explained for model 2. Therefore combining equation (20) and (22) we get

$$v = p_1^{os} - \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}}(1 + q_1^{os}) \tag{24}$$

Recalling figure 3 we can calculate the demand for each firm in the market. Areas are defined as follows

$$Area(A) = (1 - x)(1)$$

$$Area(B) = (x - w)(1 - v) + \frac{v}{2}(x - w)$$

$$Area(C) = (1 - z)(w) + \frac{w}{2}(z - v)$$

Demand for OS_1 is given by

$$D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = Area(A) + Area(B)$$

$$D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = 1 - \frac{p_1^{os}}{1 + q_1^{os}} + \left(\frac{p_1^{os}}{1 + q_1^{os}} - \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}} \right) \left(1 - \frac{1}{2} \left(\frac{p_2^{os}(1 + q_1^{os}) - p_1^{os}(1 + q_2^{os})}{q_1^{os} - q_2^{os}} \right) \right)$$

Demand for OS_2 is given by

$$D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = Area(C)$$

$$D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}} \left(1 - \frac{1}{2} \left(\frac{p_2^{os}(1 + 2q_1^{os} - q_2^{os}) - p_1^{os}(1 + q_2^{os})}{q_1^{os} - q_2^{os}} \right) \right)$$

Demand for Developers is given by

$$D^a(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = Area(A) + Area(B) + Area(C)$$

$$\begin{aligned} D^a(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = {} & 1 - \frac{p_1^{os}}{1 + q_1^{os}} + \left(\frac{p_1^{os}}{1 + q_1^{os}} - \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}} \right) \left(1 - \frac{1}{2} \left(\frac{p_2^{os}(1 + q_1^{os}) - p_1^{os}(1 + q_2^{os})}{q_1^{os} - q_2^{os}} \right) \right) \\ & + \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}} \left(1 - \frac{1}{2} \left(\frac{p_2^{os}(1 + 2q_1^{os} - q_2^{os}) - p_1^{os}(1 + q_2^{os})}{q_1^{os} - q_2^{os}} \right) \right) \end{aligned}$$

A.4 Model 4

First, to calculate the demand function of consumers we need to recall equations (3), (4), (5) and Figure 4. In the case of the purchasing decision with respect to the App, point y, is calculated as it was explained in Appendix B. Recalling equation (15) we have

$$y = p^a \tag{15}$$

Points x is defined according to equation (17) but in this case, for OS_2 therefore, we have

$$x = \frac{p_2^{os}}{1 + q_2^{os}} \tag{25}$$

Point w is defined according to equation (18). Then we have

$$w = \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}} \tag{18}$$

Given the exclusive contract between Developer and OS_1, point z is given by

$$z = p^a + p_1^{os} \tag{26}$$

Point v is the value of α_i under which consumer with a $\theta_i = x$ still gets a positive utility from buying OS_1 and app. We can define this point as

$$v = p_1^{os} + p^a - \frac{p_2^{os}}{1 + q_2^{os}}(1 + q_1^{os}) \tag{27}$$

We can define Areas as

$$Area(A) = (1 - w)(y)$$
$$Area(B) = (1 - w)(1 - y)$$
$$Area(C) = (w - x)(y)$$
$$Area(D) = \frac{1}{2}(w - x)(v - y)$$
$$Area(E) = (w - x)(1 - v) + \frac{1}{2}(w - x)(v - y)$$
$$Area(F) = x(1 - z) + \frac{1}{2}(z - v)x$$

Therefore, demand for OS_1 is given by

$$D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = Area(A) + Area(B) + Area(E) + Area(F)$$

$$D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = 1 - \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}} \left(p^a + \frac{p_1^{os}}{2} \right)$$

Demand for OS_2 is given by

$$D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = Area(C) + Area(D)$$

$$D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = \frac{1}{2} \left(\frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}} - \frac{p_2^{os}}{1 + q_2^{os}} \right) \left(2p^a + p_1^{os} - \frac{p_2^{os}(1 + q_1^{os})}{1 + q_2^{os}} \right)$$

Finally, demand for App is given by

$$D^a(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = Area(B) + Area(E) + Area(F)$$

$$D^a(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}, p^a) = 1 - p^a + \frac{1}{2} \left(\frac{p_2^{os}}{1 + q_2^{os}} \right) \left(\frac{p_2^{os}(1 + q_1^{os})}{1 + q_2^{os}} - 2p_1^{os} \right)$$

B Python codes

B.1 Model 1

```
from scipy import optimize,arange
from numpy import array
from numpy import argmax
from scipy import stats
from numpy import linspace
from fractions import Fraction

import numpy as np
import random

number_customers = 1000
customers=[]
for i in range(number_customers):
    customers.append(round(random.uniform(0,1),5))

print ('Max theta:',max(customers),'Min theta:',min(customers))

def u_os(p,q,theta):
    u_os = theta + theta*q - p

    return u_os

def buy_os(p1,p2,q1,q2,theta):

    d1 = 0
    d2 = 0

    uos1=u_os(p1,q1,theta)
    uos2=u_os(p2,q2,theta)

    if uos1 >= 0:
        if uos1 >= uos2:

            d1 = 1

    elif uos2 > 0:
        if uos2 > uos1 :

            d2 = 1


    return [d1,d2]
```

```
def demand_os(p1,p2,q1,q2,customers):
    d1_sum=0
    d2_sum=0
    long=len(customers)

    for i in customers:

        purchasing=buy_os(p1,p2,q1,q2,i)

        if purchasing[0]==1:
            d1_sum+=purchasing[0]
        if purchasing[1]==1:
            d2_sum+=purchasing[1]

    totaldemand = [d1_sum/long, d2_sum/long]

    return totaldemand

def profit_os1_2nd(p1,p2,q1,q2,customers):
    return demand_os(p1,p2,q1,q2,customers)[0]*p1

def profit_os2_2nd(p1,p2,q1,q2,customers):
    return demand_os(p1,p2,q1,q2,customers)[1]*p2

def reaction(p1,p2,q1,q2,customers):
    p1 = optimize.fminbound(lambda x: -profit_os1_2nd(x,p2,q1, q2, customers),0,1)
    p2 = optimize.fminbound(lambda x: -profit_os2_2nd(p1,x,q1, q2, customers),0,1)

    return [p1,p2]

def fixed_point(p,q,customers):

    vector_reaction = reaction(p[0],p[1],q[0],q[1],customers)

    return  [p[0] - vector_reaction[0],p[1]-vector_reaction[1]]

def equilibrium_2nd_stage(q,customers):

    x0 = [0.1,0.1]

    equil = optimize.root(fixed_point, x0, args=(q,customers))
    return equil.x

def c(q):
    return (q**2)/2
```

```
def profit_os1_1st(q1,q2,customers):

    vector_p = equilibrium_2nd_stage([q1,q2],customers)

    return profit_os1_2nd(vector_p[0],vector_p[1],q1,q2,customers)-c(q1)

def profit_os2_1st(q1,q2,customers):

    vector_p = equilibrium_2nd_stage([q1,q2],customers)

    return profit_os2_2nd(vector_p[0],vector_p[1],q1,q2,customers)-c(q2)


def reaction_q(q1,q2,customers):

    q1 = optimize.fminbound(lambda x: -profit_os1_1st(x,q2,customers),0,1)
    q2 = optimize.fminbound(lambda x: -profit_os2_1st(q1,x,customers),0,1)

    return [q1,q2]

def fixed_point_q(q,customers):

    vector_reaction = reaction_q(q[0],q[1],customers)

    return  [q[0]-vector_reaction[0],q[1]-vector_reaction[1]]

def equilibrium_1st_stage(customers):

    x0 = [0.1,0.1]

    equilib = optimize.root(fixed_point_q, x0, args=(customers))

    return equilib.x

q_os_1=equilibrium_1st_stage(customers)[0]
q_os_2=equilibrium_1st_stage(customers)[1]

print("q_1= {:.5f}, q_2= {:.5f}".format(q_os_1,q_os_2))

p_os_1=equilibrium_2nd_stage([q_os_1,q_os_2],customers)[0]
p_os_2=equilibrium_2nd_stage([q_os_1,q_os_2],customers)[1]

print("p_os_1= {:.5f}, p_os_2= {:.5f}".format(p_os_1,p_os_2))
```

```
Max theta: 0.99847 Min theta: 0.0015
q_1= 0.42427, q_2= 0.31587
p_os_1= 0.47340, p_os_2= 0.40510
```

```
def Consumers_Utility(p,q,customers):
    result=[]
    utility=[]
    for i in range (number_customers):

        uos1=u_os(p_os_1,q_os_1,customers[i])
        uos2=u_os(p_os_2,q_os_2,customers[i])
        utility.append([uos1,uos2])
        if uos1>0:
            if uos1>=uos2:

                result.append(uos1)

        elif uos2>0:
            if uos2>uos1:

                result.append(uos2)

    return(sum(result))

print (Consumers_Utility([p_os_1,p_os_2],[q_os_1,q_os_2],customers)/1000)
```

```
0.262310025537
```

```
print (profit_os1_1st(q_os_1,q_os_2,customers))
```

```
0.0875249313062
```

B.2 Model 2

```
from scipy import optimize,arange
from numpy import array
from numpy import argmax
from scipy import stats
from numpy import linspace
from fractions import Fraction

import numpy as np
import random

number_customers = 1000
customers=[]
theta_l=[]
alpha_l=[]
for i in range(number_customers):
    customers.append([round(random.uniform(0,1),5),round(random.uniform(0,1),5)])
    theta_l.append(customers[i][0])
    alpha_l.append(customers[i][1])
    # The vector is define as [theta,alpha]
print('Max alpha:',max(alpha_l),'Min alpha:',min(alpha_l),'Max theta:',max(theta_l),'Min theta:',min(t
def u_a(pa,alpha):
    return alpha - pa

def buy_a(pa, alpha):
    if u_a(pa,alpha)>= 0:
        return 1
    else:
        return 0

def demand_a(pa, customers):
    num=0
    for i in customers:
        purchasing=buy_a(pa,i[1])
        if purchasing==1:
            num+=1
    return num / len(customers)

def profita(pa, customers):
    return (demand_a(pa, customers)*pa)

p_a_optimal = optimize.fminbound(lambda pa: -profita(pa, customers),0,1)

print("p_a {:.4f}".format(p_a_optimal))
```

```python
def u_os1(p1,q1,p_a, theta, alpha):

    if u_a(p_a,alpha)>=0:
        u_os1 = theta + theta*q1 - p1 + alpha - p_a
    else:
        u_os1 = theta + theta*q1 - p1
    return u_os1

def u_os2(p2,q2,p_a, theta, alpha):
    if u_a(p_a,alpha)>=0:
        u_os2 = theta + theta*q2 - p2 + alpha - p_a
    else:
        u_os2 = theta + theta*q2 - p2
    return u_os2

def buy_os(p1,p2,q1,q2,p_a,theta,alpha):

    d1 = 0
    d2 = 0

    if u_os1(p1,q1,p_a,theta,alpha) >= 0:
        if u_os1(p1,q1,p_a,theta,alpha) >= u_os2(p2,q2,p_a,theta,alpha):

            d1 = 1

    elif u_os2(p2,q2,p_a,theta,alpha) > 0:
        if u_os2(p2,q2,p_a,theta,alpha) > u_os1(p1,q1,p_a,theta,alpha) :

            d2 = 1


    return [d1,d2]

def demand_os(p1,p2,q1,q2,p_a,customers):
    d1_sum=0
    d2_sum=0
    long=len(customers)

    for i in customers:
        purchasing=buy_os(p1,p2,q1,q2,p_a,i[0],i[1])
        if purchasing[0]==1:
            d1_sum+=purchasing[0]
        if purchasing[1]==1:
            d2_sum+=purchasing[1]

    totaldemand = [d1_sum / long, d2_sum / long]

    return totaldemand
```

```
def profit_os1_2nd(p1,p2,q1,q2, p_a,customers):
    return demand_os(p1,p2,q1,q2,p_a,customers)[0]*p1

def profit_os2_2nd(p1,p2,q1,q2, p_a,customers):
    return demand_os(p1,p2,q1,q2,p_a,customers)[1]*p2

def reaction(p1,p2,q1,q2,p_a,customers):
    p1 = optimize.fminbound(lambda x: -profit_os1_2nd(x,p2,q1, q2, p_a,customers),0,1)
    p2 = optimize.fminbound(lambda x: -profit_os2_2nd(p1,x,q1, q2, p_a,customers),0,1)

    return [p1,p2]

def fixed_point(p,q,p_a,customers):

    vector_reaction = reaction(p[0],p[1],q[0],q[1],p_a,customers)

    return  [p[0] - vector_reaction[0],p[1]-vector_reaction[1]]

def equilibrium_2nd_stage(q,p_a,customers):

    x0 = [0.1,0.1]

    equil = optimize.root(fixed_point, x0, args=(q,p_a,customers))
    return equil.x

def c(q):
    return (q**2)/2

def profit_os1_1st(q1,q2,p_a,customers):

    vector_p = equilibrium_2nd_stage([q1,q2],p_a,customers)
    return profit_os1_2nd(vector_p[0],vector_p[1],q1,q2,p_a,customers)-c(q1)

def profit_os2_1st(q1,q2,p_a,customers):

    vector_p = equilibrium_2nd_stage([q1,q2],p_a,customers)
    return profit_os2_2nd(vector_p[0],vector_p[1],q1,q2,p_a,customers)-c(q2)


def reaction_q(q1,q2,p_a,customers):

    q1 = optimize.fminbound(lambda x: -profit_os1_1st(x,q2,p_a,customers),0,1)
    q2 = optimize.fminbound(lambda x: -profit_os2_1st(q1,x,p_a,customers),0,1)

    return [q1,q2]
```

```
def fixed_point_q(q,p_a,customers):

    vector_reaction = reaction_q(q[0],q[1],p_a,customers)

    return  [q[0]-vector_reaction[0],q[1]-vector_reaction[1]]

def equilibrium_1st_stage(p_a,customers):

    x0 = [0.1,0.1]

    equilib = optimize.root(fixed_point_q, x0, args=(p_a,customers))

    return equilib.x

q_os_1=equilibrium_1st_stage(p_a_optimal, customers)[0]
q_os_2=equilibrium_1st_stage(p_a_optimal, customers)[1]

print("q_1= {:.4f}, q_2= {:.4f}".format(q_os_1,q_os_2))

p_os_1=equilibrium_2nd_stage([q_os_1,q_os_2],p_a_optimal,customers)[0]
p_os_2=equilibrium_2nd_stage([q_os_1,q_os_2],p_a_optimal,customers)[1]

print("p_os_1= {:.4f}, p_os_2= {:.4f}".format(p_os_1,p_os_2))
```

```
Max alpha: 0.99797 Min alpha: 0.0012 Max theta: 0.99977 Min theta: 0.00153
p_a 0.4743
q_1= 0.1995, q_2= 0.1615
p_os_1= 0.4356, p_os_2= 0.4222
```

```
def Consumers_Utility(p,q,pa,customers):
    result=[]
    for i in range (number_customers):

        uos1=u_os1(p_os_1,q_os_1,p_a_optimal,customers[i][0],customers[i][1])
        uos2=u_os2(p_os_2,q_os_2,p_a_optimal,customers[i][0],customers[i][1])
        if uos1>0:
            if uos1>=uos2:

                result.append(uos1)

        elif uos2>0:
            if uos2>uos1:

                result.append(uos2)
    return(sum(result))

print (Consumers_Utility([p_os_1,p_os_2],[q_os_1,q_os_2],p_a_optimal,customers)/1000)
```

```
0.33336095712
```

```
print ( profit_os1_1st(q_os_1,q_os_2,p_a_optimal,customers))
```

```
0.259763562355
```

```
print ( profita(p_a_optimal, customers))
```

```
0.252331705322
```

B.3 Model 3

```
from scipy import optimize,arange
from numpy import array
from numpy import argmax
from scipy import stats
from numpy import linspace
from fractions import Fraction

import numpy as np
import random

number_customers = 1000
customers=[]

for i in range(number_customers):
    customers.append([round(random.uniform(0,1),5),round(random.uniform(0,1),5)])
    # The vector is define as [theta,alpha]

def u_a(p_a,alpha):
    return alpha-p_a

def buy_a(p_a,alpha):
    if u_a(p_a,alpha) >= 0:
        buy = 1.0
    else:
        buy = 0.0
    return buy

def total_demand_a(p_a):
    demand_vector=[]
    for alpha in customers:
        demand_vector+=buy_a(p_a,alpha)/number_customers
    return sum(demand_vector)

def profit(p_a1,p_a2):
    if p_a1 > p_a2:
        profits = 0
    elif p_a1 == p_a2:
        profits = 0.5*total_demand_a(p_a1)*(p_a1)
    else:
        profits = total_demand_a(p_a1)*(p_a1)
    return profits
```

```
def reaction_a(p_a2):
    if p_a2 > 0:
        reaction = 0.8*(p_a2)
    else:
        reaction = 0
    return reaction

def vector_reaction_a(p):
    return array(p)-array([reaction_a(p[1]),reaction_a(p[0])])

p0 = [0.5, 0.5]

ans = optimize.fsolve(vector_reaction_a, p0)

p_a_1= ans[0]
p_a_2= ans[1]

print("p_a_1= {:.5f}, p_a_2= {:.5f}".format(p_a_1,p_a_2))

def u_os1(p1,q1,p_a_1,p_a_2, theta, alpha):

    u_a1=u_a(p_a_1,alpha)
    u_a2=u_a(p_a_2,alpha)

    if u_a1 >= 0:
        if u_a1 >= u_a2:

            u_os1= theta + theta*q1 - p1 + alpha - p_a_1

    elif u_a2 >= 0:
        if u_a2 > u_a1:

            u_os1= theta + theta*q1 - p1 + alpha - p_a_2
    else:
        u_os1= theta + theta*q1 - p1

    return u_os1

def u_os2(p2,q2,p_a_1,p_a_2, theta, alpha):

    u_a1=u_a(p_a_1,alpha)
    u_a2=u_a(p_a_2,alpha)

    if u_a1 >= 0:
        if u_a1 >= u_a2:

            u_os2 = theta + theta*q2 - p2 + alpha - p_a_1
```

```
    elif u_a2 >= 0:
        if u_a2 > u_a1:

            u_os2 = theta + theta*q2 - p2 + alpha - p_a_2
    else:
        u_os2 = theta + theta*q2 - p2
    return u_os2

def buy_os(p1,p2,q1,q2,p_a_1,p_a_2,theta,alpha):

    d1 = 0
    d2 = 0

    if u_os1(p1,q1,p_a_1,p_a_2,theta,alpha) >= 0:
        if u_os1(p1,q1,p_a_1,p_a_2,theta,alpha) >= u_os2(p2,q2,p_a_1,p_a_2,theta,alpha):

            d1 = 1

    elif u_os2(p2,q2,p_a_1,p_a_2,theta,alpha) > 0:
        if u_os2(p2,q2,p_a_1,p_a_2,theta,alpha) > u_os1(p1,q1,p_a_1,p_a_2,theta,alpha) :

            d2 = 1

    return [d1,d2]

def demand_os(p1,p2,q1,q2,p_a_1,p_a_2,customers):
    d1_sum=0
    d2_sum=0
    long=len(customers)

    for i in customers:
        purchasing=buy_os(p1,p2,q1,q2,p_a_1,p_a_2,i[0],i[1])
        if purchasing[0]==1:
            d1_sum+=purchasing[0]
        if purchasing[1]==1:
            d2_sum+=purchasing[1]

    totaldemandOS = [d1_sum / long, d2_sum / long]

    return totaldemandOS

def profit_os1_2nd(p1,p2,q1,q2,p_a_1,p_a_2,customers):
    return demand_os(p1,p2,q1,q2,p_a_1, p_a_2,customers)[0]*p1

def profit_os2_2nd(p1,p2,q1,q2,p_a_1,p_a_2,customers):
    return demand_os(p1,p2,q1,q2,p_a_1,p_a_2,customers)[1]*p2
```

```
def reaction(p1,p2,q1,q2,p_a_1,p_a_2,customers):
    p1 = optimize.fminbound(lambda x: -profit_os1_2nd(x,p2,q1,q2,p_a_1,p_a_2,customers),0,1)
    p2 = optimize.fminbound(lambda x: -profit_os2_2nd(p1,x,q1,q2,p_a_1,p_a_2,customers),0,1)

    return [p1,p2]

def fixed_point_p(p,q,p_a_1,p_a_2,customers):

    vector_reaction = reaction(p[0],p[1],q[0],q[1],p_a_1,p_a_2,customers)

    return  [p[0] - vector_reaction[0],p[1]-vector_reaction[1]]

def equilibrium_2nd_stage(q1,q2, p_a_1, p_a_2, customers):

    x0 = [0.1,0.1]

    equil = optimize.root(fixed_point_p, x0, args=([q1,q2],p_a_1,p_a_2,customers))
    return equil.x

def c(q):
    return (q**2)/2

def profit_os1_1st(q1,q2,p_a_1,p_a_2,customers):

    vector_p = equilibrium_2nd_stage(q1,q2,p_a_1,p_a_2,customers)

    return profit_os1_2nd(vector_p[0],vector_p[1],q1,q2,p_a_1,p_a_2,customers)-c(q1)

def profit_os2_1st(q1,q2,p_a_1,p_a_2,customers):

    vector_p = equilibrium_2nd_stage(q1,q2,p_a_1,p_a_2,customers)

    return profit_os2_2nd(vector_p[0],vector_p[1],q1,q2,p_a_1,p_a_2,customers)-c(q2)

def reaction_q(q1,q2,p_a_1,p_a_2,customers):

    q1 = optimize.fminbound(lambda x: -profit_os1_1st(x,q2,p_a_1,p_a_2,customers),0,1)
    q2 = optimize.fminbound(lambda x: -profit_os2_1st(q1,x,p_a_1,p_a_2,customers),0,1)

    return [q1,q2]

def fixed_point_q(q,p_a_1,p_a_2,customers):

    vector_reaction = reaction_q(q[0],q[1],p_a_1,p_a_2,customers)

    return  [q[0]-vector_reaction[0],q[1]-vector_reaction[1]]
```

```
def equilibrium_1st_stage(p_a_1,p_a_2,customers):

    x0 = [0.1,0.1]

    equilib = optimize.root(fixed_point_q, x0, args=(p_a_1,p_a_2,customers))


    return equilib.x

q_os_1=equilibrium_1st_stage(p_a_1,p_a_2, customers)[0]
q_os_2=equilibrium_1st_stage(p_a_1,p_a_2, customers)[1]

print("q_os_1= {:.4f}, q_os_2= {:.4f}".format(q_os_1,q_os_2))

p_os_1=equilibrium_2nd_stage(q_os_1,q_os_2,p_a_1,p_a_2,customers)[0]
p_os_2=equilibrium_2nd_stage(q_os_1,q_os_2,p_a_1,p_a_2,customers)[1]

print("p_os_1= {:.4f}, p_os_2= {:.4f}".format(p_os_1,p_os_2))
```

```
p_a_1= 0.00000, p_a_2= 0.00000
q_os_1= 0.2352, q_os_2= 0.0128
p_os_1= 0.4129, p_os_2= 0.4058
```

```
def Consumers_Utility(p,q,pa,customers):
    result=[]
    for i in range (number_customers):

        uos1=u_os1(p_os_1,q_os_1,p_a_1,p_a_2,customers[i][0],customers[i][1])
        uos2=u_os2(p_os_2,q_os_2,p_a_1,p_a_2,customers[i][0],customers[i][1])
        if uos1>0:
            if uos1>=uos2:

                result.append(uos1)

        elif uos2>0:
            if uos2>uos1:

                result.append(uos2)
    return(sum(result))
print (Consumers_Utility([p_os_1,p_os_2],[q_os_1,q_os_2],[p_a_1,p_a_2],customers)/1000)
```

```
0.715292719529
```

```
print (profit_os1_1st(q_os_1,q_os_2,p_a_1,p_a_2,customers))
```

```
0.347222011631
```

B.4 Model 4

```
from scipy import optimize,arange
from numpy import array
from numpy import argmax
from scipy import stats
from numpy import linspace
from fractions import Fraction

import numpy as np
import random

number_customers = 1000
customers=[]
theta_l=[]
alpha_l=[]
for i in range(number_customers):
    customers.append([round(random.uniform(0,1),5),round(random.uniform(0,1),5)])
    theta_l.append(customers[i][0])
    alpha_l.append(customers[i][1])
    # The vector is define as [theta,alpha]
print('Max alpha:',max(alpha_l),'Min alpha:',min(alpha_l),'Max theta:',max(theta_l),'Min theta:',min(t
def u_a(pa,alpha):
    return alpha - pa

def buy_a(pa, alpha):
    if u_a(pa,alpha)>= 0:
        return 1
    else:
        return 0

def demand_a(pa, customers):
    num=0
    for i in customers:
        purchasing=buy_a(pa,i[1])
        if purchasing==1:
            num+=1
    return num / len(customers)

def profita(pa, customers):
    return (demand_a(pa, customers)*pa)

p_a_optimal = optimize.fminbound(lambda pa: -profita(pa, customers),0,1)

print("p_a {:.4f}".format(p_a_optimal))
```

```
def u_os1(p1,q1,p_a, theta, alpha):

    if u_a(p_a,alpha)>=0:
        u_os1 = theta + theta*q1 - p1 + alpha - p_a
    else:
        u_os1 = theta + theta*q1 - p1
    return u_os1

def u_os2(p2,q2,theta):
    u_os2 = theta + theta*q2 - p2
    return u_os2

def buy_os(p1,p2,q1,q2,p_a,theta,alpha):

    d1 = 0
    d2 = 0

    if u_os1(p1,q1,p_a,theta,alpha) >= 0:
        if u_os1(p1,q1,p_a,theta,alpha) >= u_os2(p2,q2,theta):

            d1 = 1

    elif u_os2(p2,q2,theta) > 0:
        if u_os2(p2,q2,theta) > u_os1(p1,q1,p_a,theta,alpha) :

            d2 = 1


    return [d1,d2]

def demand_os(p1,p2,q1,q2,p_a,customers):
    d1_sum=0
    d2_sum=0
    long=len(customers)

    for i in customers:
        purchasing=buy_os(p1,p2,q1,q2,p_a,i[0],i[1])
        if purchasing[0]==1:
            d1_sum+=purchasing[0]
        if purchasing[1]==1:
            d2_sum+=purchasing[1]

    totaldemand = [d1_sum / long, d2_sum / long]

    return totaldemand
```

```
def profit_os1_2nd(p1,p2,q1,q2, p_a,customers):
    return demand_os(p1,p2,q1,q2,p_a,customers)[0]*p1

def profit_os2_2nd(p1,p2,q1,q2, p_a,customers):
    return demand_os(p1,p2,q1,q2,p_a,customers)[1]*p2

def reaction(p1,p2,q1,q2,p_a,customers):
    p1 = optimize.fminbound(lambda x: -profit_os1_2nd(x,p2,q1, q2, p_a,customers),0,1)
    p2 = optimize.fminbound(lambda x: -profit_os2_2nd(p1,x,q1, q2, p_a,customers),0,1)

    return [p1,p2]

def fixed_point(p,q,p_a,customers):

    vector_reaction = reaction(p[0],p[1],q[0],q[1],p_a,customers)

    return  [p[0] - vector_reaction[0],p[1]-vector_reaction[1]]

def equilibrium_2nd_stage(q,p_a,customers):

    x0 = [0.1,0.1]

    equil = optimize.root(fixed_point, x0, args=(q,p_a,customers))
    return equil.x

def c(q):
    return (q**2)/2

def profit_os1_1st(q1,q2,p_a,customers):

    vector_p = equilibrium_2nd_stage([q1,q2],p_a,customers)
    return profit_os1_2nd(vector_p[0],vector_p[1],q1,q2,p_a,customers)-c(q1)

def profit_os2_1st(q1,q2,p_a,customers):

    vector_p = equilibrium_2nd_stage([q1,q2],p_a,customers)
    return profit_os2_2nd(vector_p[0],vector_p[1],q1,q2,p_a,customers)-c(q2)


def reaction_q(q1,q2,p_a,customers):

    q1 = optimize.fminbound(lambda x: -profit_os1_1st(x,q2,p_a,customers),0,1)
    q2 = optimize.fminbound(lambda x: -profit_os2_1st(q1,x,p_a,customers),0,1)

    return [q1,q2]
```

```
def fixed_point_q(q,p_a,customers):

    vector_reaction = reaction_q(q[0],q[1],p_a,customers)

    return  [q[0]-vector_reaction[0],q[1]-vector_reaction[1]]

def equilibrium_1st_stage(p_a,customers):

    x0 = [0.1,0.1]

    equilib = optimize.root(fixed_point_q, x0, args=(p_a,customers))

    return equilib.x

q_os_1=equilibrium_1st_stage(p_a_optimal, customers)[0]
q_os_2=equilibrium_1st_stage(p_a_optimal, customers)[1]

print("q_1= {:.4f}, q_2= {:.4f}".format(q_os_1,q_os_2))

p_os_1=equilibrium_2nd_stage([q_os_1,q_os_2],p_a_optimal,customers)[0]
p_os_2=equilibrium_2nd_stage([q_os_1,q_os_2],p_a_optimal,customers)[1]

print("p_os_1= {:.4f}, p_os_2= {:.4f}".format(p_os_1,p_os_2))
```

```
Max alpha: 0.99871 Min alpha: 0.00083 Max theta: 0.99954 Min theta: 0.00153
p_a 0.4986
q_1= 0.2531, q_2= 0.0171
p_os_1= 0.4885, p_os_2= 0.3853
```

```
def Consumers_Utility(p,q,pa,customers):
    result=[]
    for i in range (number_customers):

        uos1=u_os1(p_os_1,q_os_1,p_a_optimal,customers[i][0],customers[i][1])
        uos2=u_os2(p_os_2,q_os_2,customers[i][0])
        if uos1>0:
            if uos1>=uos2:

                result.append(uos1)

        elif uos2>0:
            if uos2>uos1:

                result.append(uos2)
    return(sum(result))

print (Consumers_Utility([p_os_1,p_os_2],[q_os_1,q_os_2],p_a_optimal,customers)/1000)
```

```
0.317302506241
```

```
print ( profit_os1_1st(q_os_1,q_os_2,p_a_optimal,customers))
```

```
0.299635376133
```

```
print ( profita(p_a_optimal, customers))
```

```
0.264782355584
```

C Mathematical proof of profits

This appendix proofs that profits for Operating System 1 found under Python simulation and the ones found calculating demands by hand, as it was done under Appendix A, leads to the same values. Due to some problems with the code, profits for Operating System 2 were calculated only by hand as its shown under this Appendix.

C.1 Model 1

Given the equations defined under Appendix A-1 and the values presented in Table 1 in section 4, we can calculate

$$D_1^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = 1 - \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}}$$

$$D_1^{os} = 0.36993$$

$$D_2^{os}(q_1^{os}, q_2^{os}, p_1^{os}, p_2^{os}) = \frac{p_1^{os} - p_2^{os}}{q_1^{os} - q_2^{os}} - \frac{p_2^{os}}{1 + q_2^{os}}$$

$$D_2^{os} = 0.322213$$

Then, recalling equation (7) and (8), we can calculate profits as

$$\pi_1^{os} = 0.36993 * 0.4734–0.5(0.42227)^2 = 0.85122$$

$$\pi_2^{os} = 0.322213 * 0.40510–0.5(0.31587)^2 = 0.0806415$$

C.2 Model 2

First we need to calculate point v and T. Plugging in the values from Table 1, in equations (20) and (21), we get

$$v = 0.4869563$$

$$T = 0.4869551$$

Point x, w , y and z are given by equations (15), (17), (18) and (19). Plugging in the results from table 1 we get:

$$x = 0.363151$$

$$w = 0.3526$$

$$y = 0.4743$$

$$z = 0.8965$$

Now, with the equation for each area given under Appendix A-2, we can calculate

$$Area(A) + Area(B) = 1 - 0.363151 = 0.636849$$

$$Area(C) = (1-0.4869563)(0.010551)+\frac{1}{2}(0.010551)(0.4869563-0.4743) = 0.005479892$$

$$Area(D) = (1 - 0.8965)(0.3526) + \frac{1}{2}(0.3526)(0.8965 - 0.4869551) = 0.1086969$$

Therefore we have:

$$D_1^{os} = 0.642328892$$

$$D_2^{os} = 0.1086968659$$

And finally we can calculate profits according to equations (10) and (11)

$$\pi_1^{os} = (0.642328892)(0.4356) - \frac{1}{2}(0.1995)^2 = 0.2598$$

$$\pi_2^{os} = (0.1086968659)(0.4222) - \frac{1}{2}(0.1615)^2 = 0.0329$$

C.3 Model 3

First we calculate point v according to the values in Table 1 and equation (24)

$$v = 0.373466907$$

Point x, w , y and z are given by equations (17), (18), (22) and (23). Plugging in the results from table 1 we get:

$$x = 0.3342778497$$

$$w = 0.03192446$$

$$y = 0$$

$$z = 0.4058$$

Now, with the equation for each area given under Appendix A-3, we can calculate

$$Area(A) = (1 - 0.3342778497) = 0.66572215$$

$$Area(B) = (0.33428-0.0319)(1-0.37347)+\frac{1}{2}(0.33428-0.0319)(0.3735) = 0.24589$$

$$Area(C) = (0.3735)(1 - 0.4058) + \frac{1}{2}(0.3735)(0.4058 - 0.3735) = 0.0194853$$

Therefore we have

$$D_1^{os} = 0.911616$$

$$D_2^{os} = 0.0194853$$

And finally we can calculate profits according to equations (13) and (14)

$$\pi_1^{os} = (0.911616)(0.4129) - \frac{1}{2}(0.2352)^2 = 0.348747$$

$$\pi_2^{os} = (0.0194853)(0.4058) - \frac{1}{2}(0.0128)^2 = 0.007825$$

C.4 Model 4

First we calculate point v according to the values in Table 1 and equation (26)

$$v = 0.512398$$

Point x, w , y and z are given by equations (15), (18), (25) and (26). Plugging in the results from table 1 we get:

$$x = 0.378822$$

$$w = 0.437288$$

$$y = 0.4986$$

$$z = 0.9871$$

Now, with the equation for each area given under Appendix A-3, we can calculate

$$Area(A) + Area(B) = (1 - 0.437288) = 0.562712$$

$$Area(C) = (0.437288 - 0.378822)(0.4986) = 0.0291511$$

$$Area(D) = \frac{1}{2}(0.437288 - 0.378822)(0.512398 - 0.4986) = 0.000403356$$

$$Area(E)+Area(F) = (0.437288)(1-0.9871)+\frac{1}{2}(0.437288)(0.4885) = 0.1124486$$

Therefore we have

$$D_1^{os} = 0.6751606$$

$$D_2^{os} = 0.029554$$

And finally we can calculate profits according to equations (10) and (11)

$$\pi_1^{os} = (0.6751606)(0.4885) - \frac{1}{2}(0.2531)^2 = 0.297786$$

$$\pi_2^{os} = (0.029554)(0.3853) - \frac{1}{2}(0.0171)^2 = 0.0112411$$