Technology Adoption In A Hierarchical Network: A Structural Empirical Approach

Xintong Han[*]and Lei Xu[†]

**Preliminary and Incomplete.
Please do not circulate.
Comments welcome.**

This Version: 2018/03/17

Click here for the latest version.

Abstract

We study the question of technology adoption in the setting of Python programming language, and model the adoption process from Python 2 to Python 3. Python 3 provides more advanced but incompatible features. The adoption of Python 3 faces not only the classical chicken-and-egg problem due to network effects, but also adoption costs due to a hierarchical network through dependency requirements. With a complete dataset of package characteristics for all historical releases and user downloads, we build and estimate a dynamic model of technology adoption where each package developer makes an irreversible decision to adopt Python 3. The rich dataset allows us to draw the complete hierarchical structure of the packages, and we group packages into various layers based on the dependency relationship. Then we test the effect of various policies on the diffusion of Python 3 adoption through the whole network.

[*]Concordia University and CIREQ, Montreal, Canada. Email: xintong.han@concordia.ca
[†]Toulouse School of Economics, Toulouse, France. Email: lei.xu@tse-fr.eu

1 Introduction

New technologies are being developed faster than ever. But why do new and better technologies often fail to attract quick and widespread adoption? In many cases, a fast adoption of new technologies can be socially beneficial: slow adoption often leads to long time of incompatible products, and a more rapid adoption enables consumers to better enjoy the convenience brought by the latest technology. We study this question in the setting of Python programming language, in particular, an on-going transition process from Python 2 to Python 3.

Python 3 provides more advanced but incompatible features compared to Python 2. The adoption of Python 3 faces not only the classical chicken-and-egg problem due to network effects, but also adoption costs due to (1) existing code base and (2) hierarchical network through dependency requirements.

The network of packages is formed through dependencies. One package (downstream) often use other packages (upstream) as dependencies. The decision to adopt Python 3 by a downstream package often depends on the decisions of upstream packages. If the upstream packages lack Python 3 support, then the downstream packages would have to experience a higher cost of adoption in order to adopt Python 3 before its dependencies.

The motivations of developers' contribution are multi-fold, and the model abstracts the diverse motivations by assuming a maximization of total number of downloads. For any of the most commonly-known motivations (such as altruism, career, ego gratification, etc.), having more downloads is always better for the developer.

We build a dynamic model of technology adoption where each package developer makes an irreversible decision to adopt Python 3, i.e. making the package available for other Python 3 users or packages. The intertemporal tradeoff comes mainly from higher benefits (more user downloads), lower switching costs due to the Python 3 support status of the dependencies, and higher switching costs due to a growing codebase over time.

With a complete dataset of package characteristics for all historical releases and user downloads, we estimate the model using methods developed in the literature of dynamic games.[1] The computation difficulty mainly comes from the hierarchal network of dependencies. Let $U_{i,t}$ be the set of dependencies of package i at time t. When a dependency

[1]All the data come from the Python Package Index project, which is the largest repository for Python packages, and it records historical downloads information for more than 113,000 packages from 2005 until now.

$j \in U_{i,t}$ lacks Python 3 support at time t, then package i faces intertemporal tradeoff between adopting Python 3 at time t and later at time $t+\tau$ where $\tau \geq 1$. The calculation of the tradeoff depends on the knowledge of the probability of its dependency j adopting Python 3 in future time periods $t+\tau$. Most packages have more than one dependencies, therefore for each set of parameter values, for each package i at each time period t, we have to first calculate the probability of Python 3 adoption for each of the dependencies $j \in U_{i,t}$. With that information, we then build a new transition matrix of state variables, solve the value function, and calculate the adoption probability for package i at each time period t.

We simplify the estimation by taking advantage of an important feature of the hierarchical network, namely, unidirectionality. Our rich dataset allows us to draw the complete hierarchical structure of the packages, and we group packages into various layers based on the dependency relationship. Layer 0 includes packages without any dependencies; layer 1 includes packages only using packages in layer 0 as dependencies; layer 2 includes packages only using packages in layer 0 & 1 as dependencies; etc. Then we solve the choice probabilities for all the packages by each layer, starting from layer 0, then layer 1, etc. With the adoption probabilities of upstream packages, we make several assumptions on the belief of the downstream packages regarding the future adoption status of their respective dependencies.

Through the dependency network, we can also group packages into various sub-communities, such as web development, data analysis, etc. We measure both the local (within sub-community) and global (across communities) network effects. Then we test the effect of various policies on the diffusion of Python 3 adoption through the whole network.

2 Literature Review

There are two main separate streams of literature that our research touches upon. One is the literature of social networks, and the other one is the network effects literature in the field of industrial organization.

The literature of social networks allows for very complex network structure. Every agent can be linked to another in a different fashion (See Jackson (2010) for a detailed literature review and Bandiera and Rasul (2006)). However, dynamic effects have been largely ignored due to the difficulties with multiple equilibria and estimation.

On the other hand, the network effects literature in industrial organization has developed very complex dynamics systems that allows for multiple sides and firm competition (See Aguirregabiria (2011) for a detailed literature review). However, the network structure has remained simple, also for estimation purposes.

Our research is based upon existing dynamic models developed in IO and extends towards a more complex network structure on one side of a new technology, namely the package networks in the setting of Python.

Please note that the literature review of this draft is far from complete. More papers and discussion will be added over time.

Early literature of technology adoption focused mostly on the theoretical properties, which paved a solid theoretical foundation for later empirical research after much data has become widely available.

The seminars work by Katz and Shapiro (1985) show how and which technology are adopted with a sponsor, namely, some entity to promote such a technology, through a simple two-period model. It has provided predictions that has proved to be true in many settings, including Python.

More recent works on technology adoption are mainly empirical. Gowrisankaran and Stavins (2004) study banks' adoption decisions of automated clearinghouse electronic payment systems. In particular, they study how the decisions are affected by network effects and competition. Ryan and Tucker (2011) is among the first to use a structural dynamic model to study technology adoption. With detailed adoption and usage data, they show that employees have huge heterogeneous benefits and costs when adopting a new video calling technology. Our research goes more in depth and strive to explain the sources of such heterogeneous benefits and costs through user characteristics and dependency network.

Our model and setting also bear some resemblance to those by Lee (2013). Lee (2013) models the way consumers purchase game consoles and game titles over time. He also models how developers choose platforms to develop games. The focus is mainly on the demand estimation, allowing for switching behavior and purchasing multiple game titles over time. Our research, on the contrary, focus more on the developer site where packages

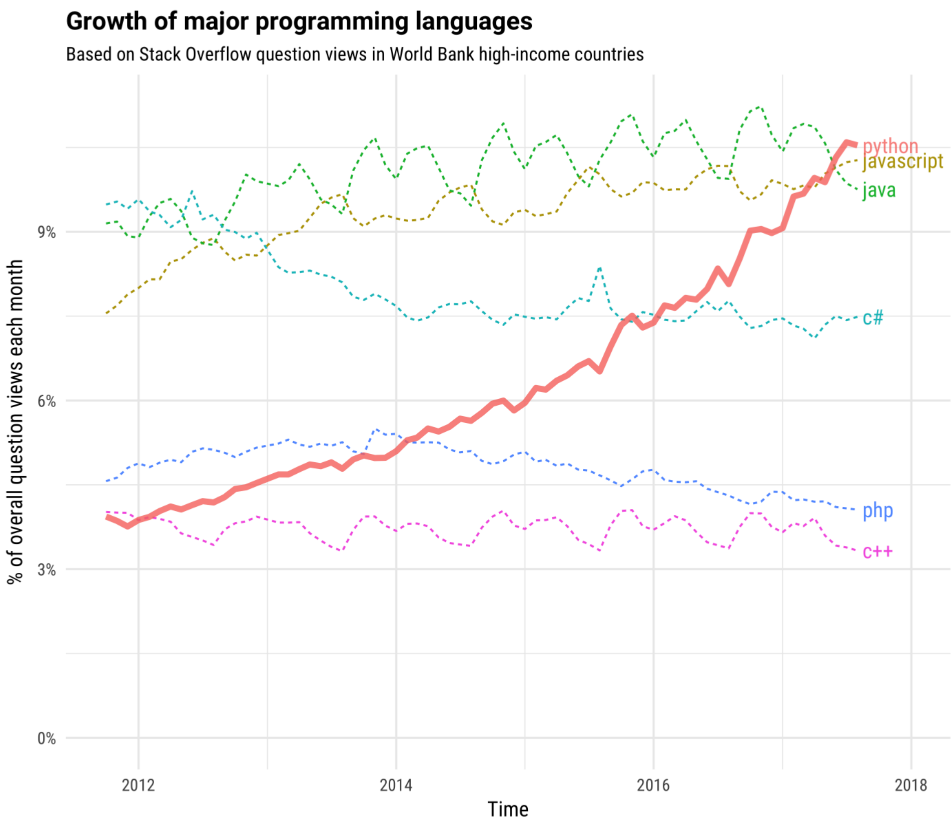

Figure 1

dynamically decide whether to adopt a new technology given a complex network structure and predicted future cost and benefit.

3 Industry Background

Python is an interpreted, object-oriented, high-level programming language. It has a syntax that allows users to express concepts in fewer lines of code compared to most other languages.

The first version was released in 1989, but it did not gain much popularity until late 2000s. Python 2 was related in 2000, and then Python 3 in 2008.

During the past few years, Python has become the fastest-growing major programming language. Figure 1 plots the numbers of visits to questions related to a particular programming language on Stack Overflow, the largest Q&A website for programming-related matters. Python has grown to be the number one based on this measure.

Backward compatibility has been a widely disputed topic in the software industry. The tradeoff is very clear: easier user transition to the new technology vs. higher cost

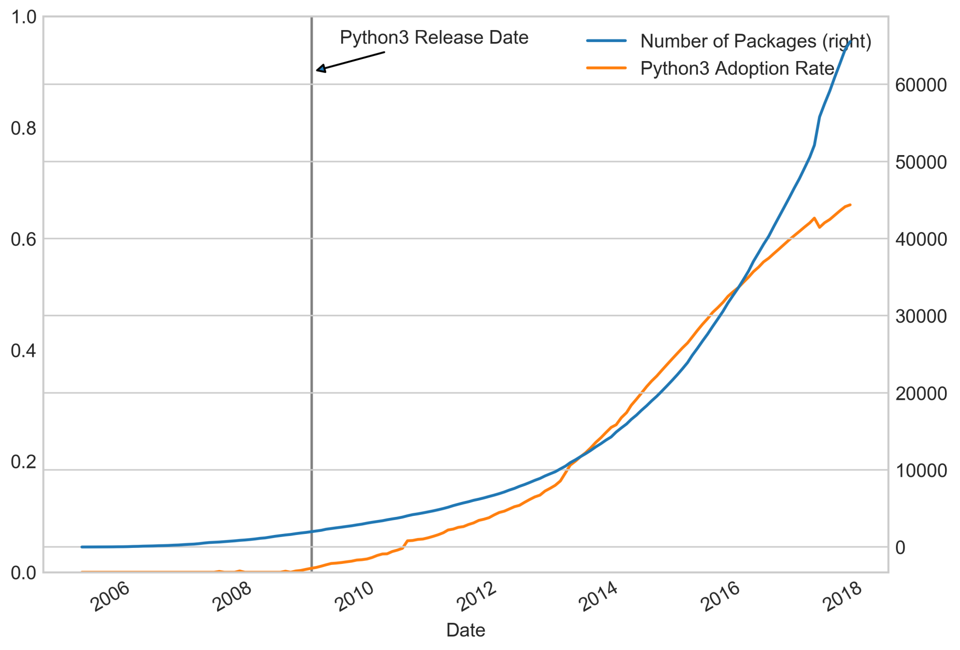

Figure 2: Python Packages with Python 3 Support

of development and slowing innovation. In order to introduce several key features to Python, the core teams decided to break the backward compatibility.[2] Users and package developers who would like to run their code on Python 3 would have to modify their old code.[3]

The transition process has indeed been longer than many had expected. Due to a large existing Python 2 user base, many packages are reluctant to provide Python 3 support. For those who did, they usually provide two versions for each release: one for Python 2, and another for Python 3 users.

Figure 2 plots the proportion of packages that provide Python 3 support. It has experienced a steady growth, during which timothy total number of packages has also been growing exponentially, indicating a growing popularity of Python programming language in general.

[2]Some of the major new features include newer classes, unicode encoding, and float division. Please refer to `http://python.org/` for more detailed information.

[3]Some packages are developed to help users to transit to Python 3 easier automatically. But in most cases, users and package developers would still have to test and manually modify much code.

Table 1: Package Characteristics

name	pandas
license	BSD
summary	Powerful data structures for data analysis
home_page	http://pandas.pydata.org
author	The PyData Development Team
author_email	pydata@googlegroups.com
version	0.10.1
requires_dist	numpy (≥1.9.0)
	pytz (≥2011k)
	python-dateutil
classifiers	Intended Audience :: Science/Research
	Programming Language :: Python :: 2
	Programming Language :: Python :: 3
	Topic :: Scientific/Engineering

Table 2: Downloads Statistics (Before 2015)

Version 1 (for Python 2)	
upload_time	2013-01-22T05:42:07
python_version	2.7
downloads	41564
filename	pandas-0.10.1.win-py2.exe
size	2041220

Version 2 (for Python 3)	
upload_time	2013-01-22T05:54:10
python_version	3.2
downloads	50892
filename	pandas-0.10.1.win-py3.exe
size	1866691

4 Data

All of our data come from Python Package Index (PyPI). It is a repository of software for the Python programming language. In another word, it is a website where Python packages can be stored and downloaded by others. It provides some easy search functions for users to find and install packages they would like to use.

When uploading packages to PyPI, package developers usually provide various information related to each release, such as the description fo the package, contact information of owner, whether they support for Python 2 or Python 3, what other packages are required as dependencies, etc. Table 1 lists the typical information available for a package. Before 2015, PyPI also records cumulative downloads statistics for each file on it. Af-

Table 3: Downloads Statistics (After 2015)

timestamp	2018-03-16 10:15:13.000 UTC
country_code	FR
filename	pandas-0.10.1-cp32-macosx.whl
project	pandas
version	0.22.0
python	3.2
system	Mac OS

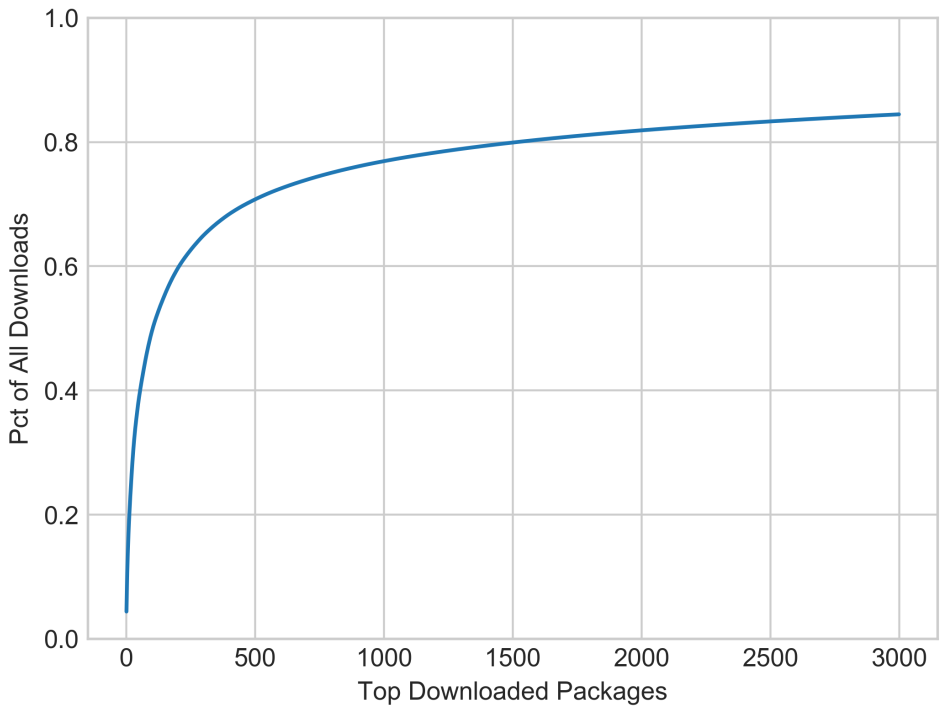

Figure 3: Total Downloads of Top Packages as Percentage of All Downloads

ter 2015, the downloads statistics system stopped working for a few month, and in late 2015, they provided a new system which provided more detailed information for each download. Table 2 and 3 show the sample data for the old and new systems. The new system provides much more detailed data over the old system. However, so far it stores only about two years of data. In this draft, we use data from the old system, i.e. until 2015. We do plan to take advantage of the rich information available in the new system in future versions of our research.

The official statistics show that it currently holds more than 130k packages and 900k releases. Like any other such platform, there are a long tail effect where most downloads come from a handful of packages and most packages are not regularly maintained. Our analysis would require a further selection of more actively maintained packages.

Figure 3 plots the total downloads of top packages as percentage of all downloads. Top 100 packages account for about 50% of all downloads; top 500 packages account for 70% of all downloads; etc.

Most packages on PyPI are small packages and are not widely used by users. They are developed probably just as a hobby for personal use. For example, 40% of all the packages have only 1 or 2 releases. We select the best-maintained and most important packages on PyPI for our estimation, based on the following criteria:

- Time Duration (Last Release - First Release Date) $\geq$ 1 Year: 12.9%
- Downloads Per year $\geq$ 2000: 30.8%
- Total Number of Releases $\geq$ 5: 38.9%
- Total Releases / Time Duration $\geq$ 1: 92.4%
- Some Python 2/3 Support Info Available: 59.9%

This selection criteria give us 3102 packages and 13056 observations for our analysis. Other selection measures will also be used for robustness checks.

5 Model Setting

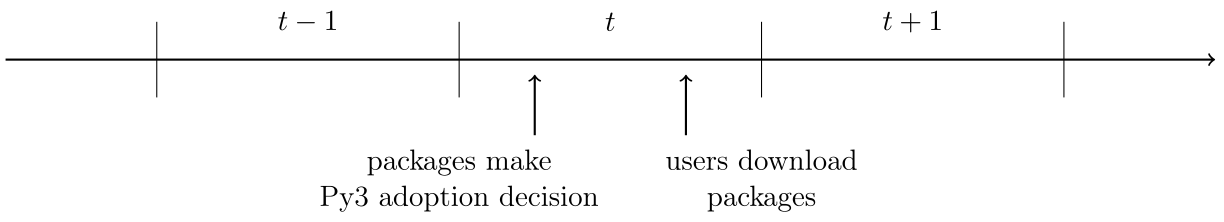

Figure 4: Timeline of Model

A package i decides at time t whether to add support to the latest Python version (i.e. Python 3) for its new release. If she adopt, she will provide online a compatible version that can be used for under Python 2 and Python 3. Once the decision is made, it is hard for her to switch back. Although it is not theoretically impossible to drop Python 3 support later, in reality, it is a very rare case. Most packages support both Python 2 and Python 3 after adding Python 3 support. Some packages just started dropping Python 2 support recently, and we expect many more will do so later. In our paper, we focus on the decision of adding Python 3 support to existing Python 2 support only. More examples of irreversible decisions as well as discussion can be found in Rust and Phelan (1997) and Aguirregabiria and Mira (2010). Let $d_{i,t}$ be the a binary irreversible decision that she made, we have

$$d_{i,t} = \begin{cases} 1 & \text{if package } i \text{ adopts Py3} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

5.1 Vertical Network with an Hierarchical Representation

One particular characteristic of our model is the existence of the vertical network. Each package i has some linked with some upstream packages and downstream packages. Let use $\mathcal{G} \in \{0,1\}^n$ a $n \times n$ matrix to present the relationship between all packages. For each element $g_{i,j} \in \mathcal{G}$, if package j is a direct upstream package that has been cited in package i's report when releasing and appears in package i's source code, we would observe $g_{i,j} = 1$. We denote $\mathcal{U}_i = \{j;\ g_{i,j} \in j \; and \; g_{i,j} = 1\}$ the set of upstream firms. We assume in our paper that $\mathcal{G}$ is time-invariant and is commonly known by packages. Furthermore, elements in $\mathcal{G}$ are assumed to be arranged in row by their hierarchy, and there are in total L hierarchies. One can use block matrices to define $\mathcal{G}$:

$$\mathcal{G} = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ H_{2,1} & 0 & \cdots & \cdots & 0 \\ H_{3,1} & H_{3,2} & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 & 0 \\ H_{L,1} & H_{L,2} & \cdots & H_{L,L-1} & 0 \end{bmatrix}$$

Where diagonal blocks and upper triangular part of the matrix $\mathcal{G}$ are zeros, block matrices $H_{i,j}$ for $i > j$ and $j = 2, ..., L$ represents the dependencies between packages in hierarchy i and its upper hierarchy j. We further denote $\mathcal{H} = \{\mathcal{H}_1, ..., \mathcal{H}_L\}$ a set indicating the element's hierarchy level where $\mathcal{H}_\ell$ contains all packages belonging to hierarchy ℓ. For instance, if the network structure is defined as in Figure **??**, we have:

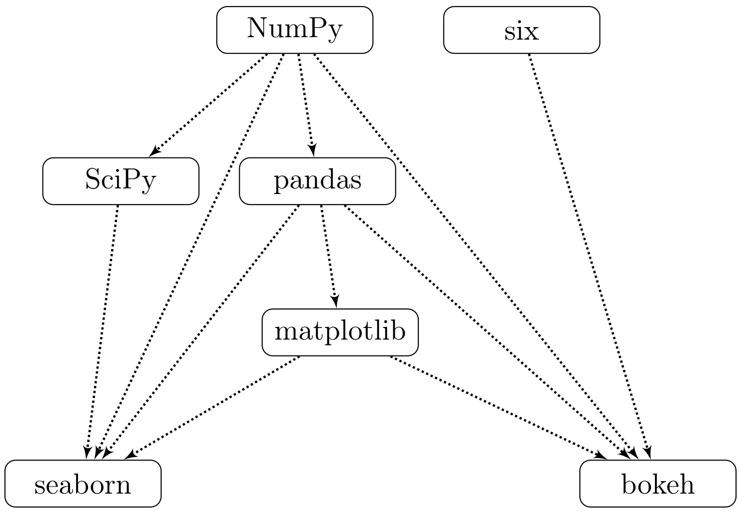

Figure 5: Example of Hierarchical Network

$$\mathcal{G} = \begin{bmatrix} 0 & 0 & 0 \\ H_{2,1} & 0 & 0 \\ H_{3,1} & H_{3,2} & 0 \end{bmatrix} = \begin{array}{c} a \\ b \\ c \\ d \end{array} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$\mathcal{H} = \{\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3\}$$

where $H_{2,1} = (1, 1)$ and $H_{3,1} = (0, 1)$, $H_{3,2} = 1$ and $\mathcal{H}_1 = \{A, B\}$, $\mathcal{H}_2 = \{C\}$, $\mathcal{H}_3 = \{D\}$. Furthermore, we have $\mathcal{U}_A = \mathcal{U}_B = \emptyset$, $\mathcal{U}_C = \{A, B\}$ and $\mathcal{U}_D = \{B, C\}$.

5.2 Adoption Cost

We model the cost function as a function of agent's decision $d_{i,t}$, the number of coding lines (internal cost) $code_i$ and the network constraint $\mu_{i,t}$ that represents the proportion of the upstream packages adopting the newest version of Python at time t. The switching cost is thereby defined as below:

$$C_{i,t} = \begin{cases} AC_0 + \alpha^{\mu}\,\mu_{i,t} & \text{if package } i \text{ adopts Py3} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

it is easy to see that if package i adopts at time t, we have $d_{i,t} = 1 \; and \; d_{i,t-1} = 0$. Otherwise, we should observe $d_{i,t} = 1 \; and \; d_{i,t-1} = 1, d_{i,t} = 0 \; and \; d_{i,t-1} = 0$, or $d_{i,t} =$

0 *and* $d_{i,t-1} = 1$ (the latest situation is in fact impossible).

The equation above admits an equivalent representation that is

$$C_{i,t} = \mathbf{1}_{\{d_{i,t}=1 \; and \; d_{i,t-1}=0\}}\{AC_0 + \alpha^{\mu}\, \mu_{i,t}\},$$

and we further assume that $\mu_{i,t}$ admits the following representation:

$$\mu_{i,t} = \sum_{j \in \mathcal{U}_i} \mathbf{1}\{d_{j,t} = 0\} \tag{3}$$

by construction the cost related to the network constraint is between 0 and 1.

Assumption 1. *At time t, a package i observes Python 3 adoption decisions made by its dependencies, namely, $d_{i,t}$ for all $j \in U_{i,t}$.*

This is an assumption on the information set available to a package when making decisions. In particular, the question is that if a dependency $j \in U_{i,t}$ makes adoption decision at time t, would package i know it or not? If not, then it's more appropriate to model the game as a simultaneous-move game where the decisions are based on information from the previous period. If yes (as stated in Assumption 1), then it's more appropriate to model it as a sequential-move game, and the downstream packages have a better ideas of their adoption costs.

We prefer the latter setup for the following reasons: First, at any moment in time, a upstream package tend to have more public information available regarding the plans for future releases. A upstream package is usually much more popular compared to its downstream counterparts (in terms of downloads). They tend to pre-announce their plans for future releases, including Python 3 adoption decisions. Second, on the other hand, for a downstream package who wants to adopt Python 3 but faces incompatible dependencies, it's likely that it would pay more attention to such decision by its dependencies. In such a case, it is more likely that the Python 3 adoption decisions of upstream packages are available to downstream packages as soon as they are made.

Assumption 1 implies a sequential-move game where upstream packages make decisions first, followed by the downstream packages. The exact order of play in the model is defined based on the network structures, and it will be specified in detail in later sections.

Assumption 1 simplifies our model estimation only slightly. Since the adoption cost comes from the dependencies without Python 3 support, at time t, a package i has to weigh the tradeoff between adopting today vs later, taking into consideration of the adoption probability of each of the Python 3-incompatible dependencies. With this assumption, the set of dependencies without Python 3 support at time t is weakly smaller, which can alleviate some computational burden in certain cases.

We do not think that a simultaneous-move game (without Assumption 1) and a sequential one (with Assumption 1) would produce significantly different results. The only observations that can give rise to different estimates are cases when a package and its dependencies adopt Python 3 in the same time period, which are rare occasions in the data.

If package i observes $d_{j,t} = 1$, then the perceived AC is much smaller than the case without observing $d_{j,t}$. In this case, the cost parameter with a simultaneous-move model is underestimated compared to a sequential one.

5.3 Payoff Function

The motivations of developers' contribution are multi-fold, and the model abstracts the diverse motivations by assuming a maximization of total number of downloads. There are a large literature on the motivations behind private contributions to public goods such as Open Source Software. All the packages in our study are open source, which is also free of charge for anyone to use.[4] For any of the most common motivations (such as altruism, career, ego gratification etc), more downloads are always better for the developer. Therefore, we assume that package developers benefit from more downloads of their packages.

Let $x_{i,t} = log(DL_{i,t})$, which is the logrithm of total number of times a package i is downloaded by users at time t. Combining with the adoption cost described in the previous sub-section, the payoff function can be written in the following way:

$$u_{i,t} = x_{i,t} - C_{i,t} + \nu^{d}_{i,t} \tag{4}$$

[4]For very limited number of packages, though anyone can download it free of charge, a certain fee has to be paid in order to use it.

where $\nu^d_{i,t}$ is a choice specific idiosyncratic shock which follows type I extreme value distribution, we denote that $\nu^d_{i,t}$ is only observed by developer but not by the econometrician. We use $\mathcal{R}_{i,t}$ to represent the package i's decision restriction. Once a package i has decided to adopt the new technology, which means $d_{i,t-1} = 1$ (he already provided a compatible version of Python 23). This package cannot switch back from Python 2/3 or 3 to its older version. Taking $\mathcal{R}_{i,t} = \infty \times \mathbf{1}\{d_{i,t} = 0, d_{i,t-1} = 1\}$, we ensure that the aforementioned situation will never happen. Thus, $d^*_{i,t-1} = 1$ implies that $d^*_{i,t+\tau} = 1$, for $\tau = 1, ..., \infty$.

The evolution of $x_{i,t}$ relies on the package's decision of the past period $d_{i,t-1}$, and we model it to follow an $\mathcal{AR}(1)$ process:

$$x_{i,t} = \varrho_{d,0} + \varrho_{d,1} x_{i,t-1} + (z_i' \varrho_{d,2}) + \varepsilon^d_{i,t} \quad with \tag{5}$$

$$d = \begin{cases} 1 & \sum_{\tau \le t} d_{i,\tau} = 1 \\ 0 & \sum_{\tau \le t} d_{i,\tau} = 0 \end{cases}. \tag{6}$$

where d indicates if the package i has adopted before $t+1$, z_i is a time-invariant vector capturing the package i's specific characteristics, $\varepsilon^0_{i,t}$ and $\varepsilon^1_{i,t}$ are two white noises that are normally and interdependently distributed with mean 0 and variances $\sigma^2_{d_0}$ and $\sigma^2_{d_1}$. After controlling a rich and abundant set of covariates z_i and along with $x_{i,t-1}$, one can assumptionume there is no unobserved variables in the equation of popularity that affects both $\varepsilon_{d,i,t}$ and $(x_{i,t-1}, z_i)$,

$$\mathbf{E}(\varepsilon_{d,i,t} | x_{i,t-1}, z_i) = 0,$$

and the average effect of adoption is captured by

$$\mathbf{E}(x^1_{i,t} - x^0_{i,t} | x_{i,t-1}, z_i, d_t = 1) = \tag{7}$$

$$\varrho_{\Delta,0} + \varrho_{\Delta,1} x_{i,t} + z_i' \varrho_{\Delta,2}, \tag{8}$$

where $\varrho_{\Delta,k} = \varrho_{1,k} - \varrho_{0,k}$ for $k = 0, 1, 2$.

5.4 State Markov Chain

We define $\mathcal{S}_{i,t}$ as a nested set of state variables containing $(x_{i,t-1}, d_{i,t-1}, \{d_{j,t}, S_{j,t}\}_{j\in U_{i,t}})$, where $U_{i,t}$ is the set of dependencies of package i at time t. $\mathcal{S}_{i,t}$ has a Markov representation, and by construction, we have $\mathbf{P}(\mathcal{S}_{i,t+1}|\mathcal{S}_{i,0}, ..., \mathcal{S}_{i,t}) = \mathbf{P}(\mathcal{S}_{i,t+1}|\mathcal{S}_{i,t})$. We model an irreversible adoption decision process, which implies that $\mathbf{P}(d_t = 1|d_{t-1} = 1) = 1$. The law of motion of $x_{i,t}$ is described as an AR1 process (equation 5).

As mentioned in the previous sections, one important component of the adoption cost comes from the depenedency network when the dependency packages lack Python 3 support, which is represented by the last component of the state variables $\{d_{j,t}, S_{j,t}\}_{j\in U_{i,t}})$. We model a dynamic model of sequential decisions where upstream packages make adoption decisions before downstream packages, and downstream packages observe the decisions made by upstream packages at time t, namely $d_{j,t}$.[5] We define the number of package i's dependencies who haven't adopted Python 3 support at time t by $\mu_{i,t} \equiv \sum_{j\in U_{i,t}} \mathbb{1}(d_{j,t} = 0)$ [6]

Package i cares about $\{d_{j,t}, S_{j,t}\}_{j\in U_{i,t}})$ in so far as it cares about its own adoption cost (a function of $U_{i,t}$), as well as its future evolution based given the current states. The law of motion of $\mu_{i,t}$ can be computed in the following way:

$$\begin{aligned} & \mathbf{P}(\mu_{i,t+1} = \mu'|\mu_{i,t} = \mu, \{d_{j,t}, \mathcal{S}_{j,t}\}_{j\in\mathcal{U}_{i,t}}; \theta) \\ = \ & \mathbf{P}(\mu' - \mu = \Delta\mu|\{d_{j,t}, \mathcal{S}_{j,t}\}_{j\in\mathcal{U}_{i,t}}; \theta) \end{aligned} \tag{9}$$

where $\Delta\mu$ is the change of the number of packages lacking Python 3 support from time t to $t+1$ and $\Delta\mu \in \{0, -1, -2, ..., -\mu\}$, i.e. adoption cost in future periods might be lower. Again, $\{d_{j,t}, S_{j,t}\}_{j\in U_{i,t}})$ is important in predicting dependency j's probability of Python 3 adoption at a future date. Denote package i's belief that its dependency j will adopt Python 3 at time $t+1$ as $\widehat{p}^1_{j,t+1} = \mathbf{P}(d_{j,t+1} = 1|d_{j,t} = 0, \mathcal{S}_{j,t}; \theta)$, and $\widehat{p}^0_{j,t+1} = \mathbf{P}(d_{j,t+1} = 0|d_{j,t} = 0, \mathcal{S}_{j,t}; \theta)$.

Then we can write equation 9 as:

$$\mathbf{P}(\mu' - \mu = \Delta\mu|\{d_{j,t}, \widehat{p}_{j,t+1}\}_{j\in\mathcal{U}_{i,t}}; \theta) \tag{10}$$

[5]More details of the model can be found in Section xxx.

[6]Future versions will include other package characteristics such as lines of code, e.g. $\mu_{i,t} \equiv \sum_{j\in U_{i,t}} \mathbb{1}(d_{j,t} = 0) log(code_{j,t})$

With $\mu_{i,t} \equiv \sum_{j \in U_{i,t}} \mathbb{1}(d_{j,t} = 0)$, we can further simplify 10 by denoting the set of dependencies without Python 3 support as $\Omega_{i,t} \equiv \{j \in U_{i,t} | d_{j,t} = 0\}$, thus $\mu_{i,t} = |\Omega_{i,t}|$.

Then equation 10 can be written in the following way:

$$\mathbf{P}(\mu' - \mu = \Delta\mu | \{\widehat{p}_{j,t+1}\}_{j \in \Omega_{i,t}}; \theta) \tag{11}$$

The adoption decisions by different packages in $\Omega_{i,t}$ can lead to the same value of $\Delta\mu$. Define $\mathcal{O}_{i,t}$ as the power set of $\Omega_{i,t}$ that contains all possible subsets of $\Omega_{i,t}$. Further we denote $\mathcal{O}^k_{i,t} = \{o \in \mathcal{O}_{i,t} || o| = k\}$, i.e. all the elements of set $\mathcal{O}_{i,t}$ with the same cardinality of k.

Then equation 11 can be written as:

$$\sum_{o' \in \mathcal{O}^{\mu'}_{i,t}} \mathbf{P}(o' | \Omega_{i,t}, \{\widehat{p}_{j,t+1}\}_{j \in \Omega_{i,t}}; \theta) \tag{12}$$

$$= \sum_{o' \in \mathcal{O}^{\mu'}_{i,t}} \left(\prod_{j \in o'} \widehat{p}^0_{j,t+1} \prod_{j \in \Omega_{i,t} \backslash o'} \widehat{p}^1_{j,t+1} \right) \tag{13}$$

The burdensome mathmatical notation can be easily understood with a simple example. Suppose that at time t, package i has 3 dependencies $U_{i,t} = \{A, B, C\}$, and $d_{A,t} = 0$, $d_{B,t} = 0$, $d_{C,t} = 1$, leaving the set of dependency without Python 3 support being $\Omega_{i,t} = \{A, B\}$. Suppose package i's belief that each of the dependency will adopt Python 3 at time $t+1$ with the following probability: $\widehat{p}^1_{A,t+1} = a$, $\widehat{p}^0_{A,t+1} = 1-a$, $\widehat{p}^1_{B,t+1} = b$, $\widehat{p}^0_{B,t+1} = 1 - b$. The powerset of $\Omega_{i,t}$ is $\mathcal{O}_{i,t} = \{\varnothing, \{A\}, \{B\}, \{A, B\}\}$, and $\mathcal{O}^0_{i,t} = \{\varnothing\}$, $\mathcal{O}^1_{i,t} = \{\{A\}, \{B\}\}$, $\mathcal{O}^2_{i,t} = \{\{A, B\}\}$. Following equation 13, the transition from $\mu = 2$ to $\mu' = 1$ can be calculated as:

$$\sum_{o' \in \mathcal{O}^1_{i,t} = \{\{A\},\{B\}\}} \left(\prod_{j \in o'} \widehat{p}^0_{j,t+1} \prod_{j \in \Omega_{i,t} \backslash o'} \widehat{p}^1_{j,t+1} \right) \tag{14}$$

$$= \prod_{j \in o'} \widehat{p}^0_{j,t+1} \prod_{j \in \Omega_{i,t} \backslash o'} \widehat{p}^1_{j,t+1} \tag{15}$$

$$= \prod_{j \in \{A\}} \widehat{p}^0_{j,t+1} \prod_{j \in \{A,B\} \backslash \{A\}} \widehat{p}^1_{j,t+1} + \prod_{j \in \{B\}} \widehat{p}^0_{j,t+1} \prod_{j \in \{A,B\} \backslash \{B\}} \widehat{p}^1_{j,t+1} \tag{16}$$

$$= \widehat{p}^0_{A,t+1} \cdot \widehat{p}^1_{B,t+1} + \widehat{p}^0_{B,t+1} \cdot \widehat{p}^1_{A,t+1} \tag{17}$$

$$= \; (1-a)\,b + a\,(1-b) \tag{18}$$

(Later make the assumption that $\widehat{p}^1_{j,t+\tau} = \widehat{p}^1_{j,t}, \forall \tau \geq 0$)

New TP:

$$P(o_{i,t} \in O_{i,t-1} | \Omega_{i,t-1}) = \prod_{j \in o_{i,t}} p^0_j \prod_{j \in \Omega_{i,t-1} \backslash o_{i,t}} p^1_j \tag{19}$$

5.5 Decision Rules and Bellman Equation

Note that the state variable is denoted as $S_{i,t} \equiv (x_{i,t}, d_{i,t-1}, \{d_{j,t}, S_{j,t}\}_{j \in U_{i,t}})$.

The flow utility of package t at time t is calculated as the following:

$$u_{i,t}(S_{i,t}, d_{i,t}; \theta) = \alpha^x \cdot x_{i,t}(x_{i,t-1}, d_{i,t}) + C_{i,t}(d_{i,t}, d_{i,t-1}, \{d_{j,t}\}_{j \in U_{i,t}}; \theta) \tag{20}$$

As explained in section (x), the law of motion of logrithm of downloads follows an AR1 process $x_{i,t}(x_{i,t-1}, d_{i,t}) = \rho_{d_{i,t},0} + \rho_{d_{i,t},1} \cdot x_{i,t-1}$. The adoption cost of Python 3 $C_{i,t}$ is summarized as:

$$C_{i,t} = \begin{cases} AC_0 + \alpha^\mu \, \mu_{i,t}(\{d_{j,t}\}_{j \in U_{i,t}}) & \text{if } d_{i,t-1} = 0 \text{ and } d_{i,t} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

In the current version of our model, $\mu_{i,t}(\{d_{j,t}\}_{j \in U_{i,t}}) = \sum_{j \in U_{i,t}} (d_{j,t} = 0)$.

At each period t, the package i considers the following dynamic problem:

$$\max_{\{d_{i,t+\tau}\}_{\tau=0}^{\infty}} \mathbf{E}_t \{ \sum_{\tau=0}^{\infty} \beta^\tau u_{i,t+\tau}(S_{i,t+\tau}, d_{i,t+\tau}) | \mathcal{S}_{i,t}, d_{i,t}; \theta \} \tag{22}$$

The Bellmen equation implies at each period t, the ex ante value function, conditional on $d_{t-1} = 0$, can be represented by:

$$V(\mathcal{S}_{i,t}, d_{t-1} = 0, v_{i,t}; \theta) \tag{23}$$

$$= \max_{\{d_{i,t+\tau}\}_{\tau=0}^{\infty}} \mathbf{E}_t \{ \sum_{\tau=0}^{\infty} \beta^\tau u_{i,t+\tau}(S_{i,t+\tau}, d_{i,t+\tau}) | \mathcal{S}_{i,t}, d_{i,t}; \theta \} \tag{24}$$

$$= \max_{d_{i,t} \in \{0,1\}} u_{i,t}(S_{i,t}, d_{i,t}; \theta) + v^{d_{i,t}}_{i,t} + \beta \mathbf{E}_t V(\mathcal{S}_{i,t+1}, v_{i,t+1} | \mathcal{S}_{i,t}, v_{i,t}, d_{i,t}; \theta) \tag{25}$$

$$= \max\{v(\mathcal{S}_{i,t}, v_{i,t}, d_{i,t} = 0; \theta), v(\mathcal{S}_{i,t}, v_{i,t}, d_{i,t} = 1; \theta)\} \tag{26}$$

$$= \max\{\alpha^x (\rho_{0,0} + \rho_{0,1} x_{i,t}) + v^0_{i,t} + \beta \mathbf{E}_t V(\mathcal{S}_{i,t+1}, v_{i,t+1} | \mathcal{S}_{i,t}, v_{i,t}, d_{i,t} = 0; \theta), \tag{27}$$

$$\alpha^x (\rho_{1,0} + \rho_{1,1} x_{i,t}) + AC_0 + \alpha^\mu \mu_{i,t} + v^1_{i,t} + \beta \mathbf{E}_t V(\mathcal{S}_{i,t+1}, v_{i,t+1} | \mathcal{S}_{i,t}, v_{i,t}, d_{i,t} = 1; \theta)\} \tag{28}$$

Following the arguments of Magnac and Thesmar (2002), both β and the distribution of ν cannot be identified, thus we assume that $v^{d_{i,t}}_{i,t}$ are iid errors that follows Extreme Value Distribution (Type I). Then $EV(\mathcal{S}_{i,t}, d_{i,t} = 0; \theta)$ can be calculated using the following equation:

$$EV(\mathcal{S}, d = 0; \theta) = \int_{S'} \log\{ \sum_{d' \in \{0,1\}} \exp(u(S', d'; \theta) + \beta EV(\mathcal{S}', d'; \theta))\} d\mathbf{P}_{\mathcal{S}'|\mathcal{S}} \tag{29}$$

The irreversible decision implies that $P(d_{i,t+\tau} = 1 | d_{i,t} = 1) = 1$ for all $\tau \in \mathbb{N}$. With this condition, $EV(\mathcal{S}_{i,t}, d_{i,t} = 1; \theta)$ can be calculated with a closed-form express:

$$EV(S, d = 1; \theta) = \frac{1 + \rho_{1,1} (1 - \beta)}{(1 - \beta)(1 - \beta \rho_{1,1})} \rho_{1,0} + \frac{1}{1 - \beta \rho_{1,1}} \rho_{1,1} x \tag{30}$$

For the convergence of the expected value function of adopting Python 3, we need an additional assumption that $\beta \varrho_{1,1} \in (-1, 1)$.

Finally, the probability of adopting Python 3 is calculated using the logit property:

$$P(d_{i,t} = 1 | \mathcal{S}_{i,t}, d_{t-1} = 0, v_{i,t}; \theta) = \frac{exp\{v(\mathcal{S}_{i,t}, v_{i,t}, d_{i,t} = 1; \theta)\}}{\sum_{d' \in \{0,1\}} exp\{v(\mathcal{S}_{i,t}, v_{i,t}, d'; \theta)\}} \tag{31}$$

$$P(d_{i,t} = 0 | \mathcal{S}_{i,t}, d_{t-1} = 0, v_{i,t}; \theta) = 1 - P(d_{i,t} = 0 | \mathcal{S}_{i,t}, d_{t-1} = 0, v_{i,t}; \theta) \tag{32}$$

$$P(d_{i,t} = 1 | \mathcal{S}_{i,t}, d_{t-1} = 1, v_{i,t}; \theta) = 1 \tag{33}$$

5.6 Transition Matrix

The calculation of EV in equation 29 depends crucially on the specification of the transition matrix, or $\mathbf{P}_{\mathcal{S}'|\mathcal{S}}$ in equation 29.

In our model, the utility function is mainly governed by two variables, namely, $x_{i,t}$, the measure of downloads or popularity, and $\mu_{i,t}$, the measure of adoption cost due to dependencies. The construction of the transition matrix depends on the join law of motion of $x_{i,t}$ and $\mu_{i,t}$.

The law of motion of $x_{i,t}$ is relatively easy. We assume that it follows an AR1 process as specified in equation 5, with the parameter values estimated outside the value function iteration.

On the other hand, the law of motion of $\mu_{i,t}$ is much more difficult. $\mu_{i,t} \equiv \Omega_{i,t} \equiv \sum_{j \in U_{i,t}} \mathbb{1}(d_{j,t} = 0)$, i.e. the number of dependencies of package i without Python 3 support at time t.

One of the most important tradeoff for package i to adopt Python 3 today at t vs. future periods $t + \tau$ is the decreasing adoption cost due to the decreasing number of dependencies without Python 3 support over time. Therefore, the solution to the dynamic adoption model depends on the calculation of future adoption probabilities for each of the dependencies.

The calculation is a formidable task due to the nature of the nested dependency network. The computational difficulty can be illustrated by forecasting the Python 3 adoption probability for each of package i's dependency $j \in U_{i,t}$ at time $t+1$.

$$\widehat{p^1_{j,t+1}} = \int_{\mathcal{S}_{j,t+1}} \mathbf{P}(d_{j,t+1} = 1 | \mathcal{S}_{j,t+1}, d_{j,t} = 0, z_j; \theta) d\mathbf{P}(\mathcal{S}_{j,t+1} | \mathcal{S}_{j,t}, d_{j,t} = 0, z_j; \theta) \tag{34}$$

The integral can then be computed through simulation of future states $\mathcal{S}_{j,t+1}$. Let $\mathcal{S}^m_{j,t+1}$ be the mth simulation, than the value of $\widehat{p^1_{j,t+1}}$ can be obtained by:

$$\widehat{p^1_{j,t+1}} = \frac{1}{M} \sum_{m=1}^{M} \mathbf{P}(d_{j,t+1} = 1 | \mathcal{S}^m_{j,t+1}, d_{j,t} = 0, z_j; \theta) \tag{35}$$

This simulation method can be very computationally intensive. Note that the nested states are expressed as $(x_{i,t-1}, d_{i,t-1}, \{d_{j,t}, S_{j,t}\}_{j \in U_{i,t}})$. The simulation of the states of package j requires the simulation of the states of j's dependency $k \in U_{j,t+1}$, as well as the states of k's dependency, and so on. Any slight changes in any of the linked dependencies, or the dependencies of each dependency, can affect $p_{j,t}$. In this way, the full solution approach by Rust (1987) is no longer feasible due to the curse of dimensionality problem. In fact, with the 3102 packages and 13056 observations, it is simpler to build the transition matrix dynamically for each package i at each time period t. In later sections (x) we list the detailed steps regarding how to compute $p_{j,t}$ for $j \in \Omega_{i,t}$ given the hierarchical network.

A transition matrix used for the dynamic programming problem includes the transition probability not only from the current state to the next, but also from each of the possible states to all other states. Given the state $S_{i,t}$, package i calculates $\widehat{p}^1_{j,t}$ for each of $j \in \Omega_{i,t}$. To construct the transition matrix, the belief of $\widehat{p}^1_{j,t+\tau}$ for $\tau \in \mathbb{N}$ is also needed.

Assumption 2. *Package i holds myopic expectations regarding the future Python 3 adoption probabilities by its dependencies, i.e.* $\widehat{p}^1_{j,t+\tau} = \widehat{p}^1_{j,t}$ *for all* $\tau \in \mathbb{N}$.

This assumption implies that although package i can correctly calculate the adoption probability of its dependencies at time t, the ability to forecast future probabilities is limited.

Assumption 2 greatly simplifies the model estimation. In a way, it bears certain similarity to the assumptions of inclusive value function, which is assumed to follow an AR1 process, in many previous papers (see Lee (2013) and Gowrisankaran and Rysman (2012)). In future version of this paper, we also plan to explore the possibility to allow the forecast of the adoption probability to follow such an AR1 process.

Combining the transition probability specified in equation 13 and Assumption 2, the full TM needed to calculate $EV(S, d = 0; \theta)$ can be specified as the following:

$$P(o' \in O_{i,t} | o \in O_{i,t}) = \begin{cases} 0 & \text{if } o \not\subseteq o' \\ \prod_{j \in o'} \widehat{p}^0_{j,t} \prod_{j \in o \setminus o'} \widehat{p}^1_{j,t} & \text{if } o \subseteq o' \end{cases} \tag{36}$$

The calculation of TM, as specified in equation 36, can be illustrated using the same example in section 5.4. Recall that the set of dependencies without Python 3 support is $\Omega_{i,t} = \{A, B\}$. The adoption probability of A and B at time t can be calculated by package i. Assume that $\widehat{p}^1_{A,t} = a$ and $\widehat{p}^1_{B,t} = b$. Assumption 2 implies that $\widehat{p}^1_{A,t+\tau} = a$ and $\widehat{p}^1_{B,t+\tau} = b$ for all $\tau \in \mathbb{N}$. The powerset of $\Omega_{i,t}$ is $\mathcal{O}_{i,t} = \{\varnothing, \{A\}, \{B\}, \{A, B\}\}$, and

$\mathcal{O}^0_{i,t} = \{\varnothing\}$. Therefore, the transition matrix of μ can be calculated using equation 36:

$$TM(\mu_{i,t}) = \begin{array}{c|cccc} & \{A,B\} & \{A\} & \{B\} & \varnothing \\ \hline \{A,B\} & (1-a)(1-b) & a(1-b) & (1-a)b & ab \\ \{A\} & 0 & 1-a & 0 & a \\ \{B\} & 0 & 0 & 1-b & b \\ \varnothing & 0 & 0 & 0 & 1 \end{array}$$

The corresponding value of μ for each row (or column) is 2,1,1,0, respectively. By construction, the transition matrix takes into account both the number and the identities of dependencies that are without Python 3 support. For example, the situation with dependency A being the only one without Python 3 support is different from the situation when B is the only one. Both cases have the same value of μ which is 1, thus having the same adoption cost, but the value of waiting is different because A and B have different likelihood of adoption in future periods. The TM calculated using equation 36 takes cares of all such cases.

6 Estimation

Our model of technology adoption is estimated using Maximum Likelihood Estimation (MLE). The likelihood function is defined as: $l(\theta) = \prod_{i=1}^{N}\prod_{t=1}^{T} \widehat{p}^{0}_{i,t}{}^{\mathbb{1}\{d_{i,t}=0\}} \widehat{p}^{1}_{i,t}{}^{\mathbb{1}\{d_{i,t}=1\}}$

where $\widehat{p}^1_{i,t} \equiv \widehat{p}^1_{i,t}(S_{i,t};\theta)$, which is defined in equation 31.

In this section, we provide detailed steps of model estimation.

Step 0: Model & Data Primitives The discount factor in dynamic discrete choice models is typically unidentified in the dynamic environment (see Magnac and Thesmar (2002)). In the current version, we fix the discount factor to 0.9.

The actual release time of each package is continuous time. For our model with choices at discrete time, we divide the time into half-year intervals, e.g. 2013/01/01 to 2013/06/30 is one period, and 2013/07/01 to 2013/12/31 is another.

Step 1: User Downloads (AR1 Process) In the current draft, the demand system is assumed to follow a simple AR1 process specified according to equation 5. The parameter

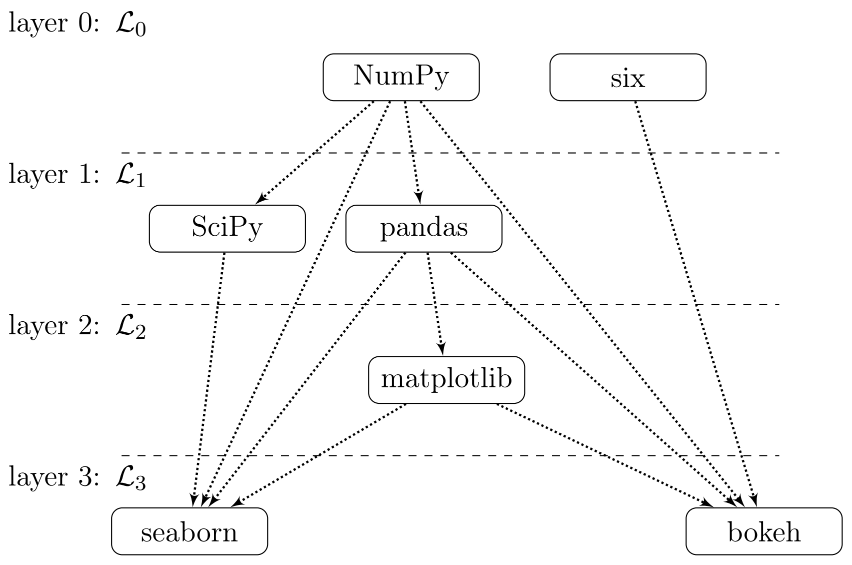

Figure 6: Layered Hierarchical Network

estimates of the AR1 process will be used as inputs to the technology adoption model.

Step 2: Hierarchical Network Through the dependency requirement, we group packages into different layers: $0, 1, 2, \cdots, L$. Figure 6 shows an example of the layered network. Each layer contains a set of packages with dependencies in the upstream layers. They can be gourd using the following iteration:

$$\begin{aligned}
\mathcal{L}_0 =& \{i \mid |U_i| = 0\} \\
\mathcal{L}_1 =& \{i \mid U_i \subseteq \mathcal{L}_0\} \\
\mathcal{L}_2 =& \{i \mid U_i \subseteq \mathcal{L}_{0,1}\} \\
& \vdots \\
\mathcal{L}_l =& \{i \mid U_i \subseteq \mathcal{L}_{0,1,\cdots,l-1}\} \\
& \vdots \\
\mathcal{L}_L =& \{i \mid U_i \subseteq \mathcal{L}_{0,1,\cdots,L-1}\}
\end{aligned}$$

where $\mathcal{L}_{0,1,\cdots,l} = \mathcal{L}_0 + \mathcal{L}_1 + \cdots + \mathcal{L}_l$

Step 3: Estimation of the Model of Technology Adoption

- Given initial parameter values $\theta^0 = \{\alpha^x, \alpha^\mu, AC_0\}$
  - for $i \in \mathcal{L}_0$, build transition matrix for each i, t and calculate $\widehat{p}^1_{i,t}(\mathcal{S}_{i,t}; \theta)$

  - for $i \in \mathcal{L}_1$, given $\widehat{p}^1_{j,t}(\mathcal{S}_{j,t};\theta) \quad \forall j \in \mathcal{L}_0$, build transition matrix for each i,t and calculate $\widehat{p}^1_{i,t}(\mathcal{S}_{i,t};\theta)$
  - for $i \in \mathcal{L}_2$, given $\widehat{p}^1_{j,t}(\mathcal{S}_{j,t};\theta) \quad \forall j \in \mathcal{L}_{0,1}$, build transition matrix for each i,t and calculate $\widehat{p}^1_{i,t}(\mathcal{S}_{i,t};\theta)$

    $\vdots$
  - for $i \in \mathcal{L}_l$, given $\widehat{p}^1_{j,t}(\mathcal{S}_{j,t};\theta) \quad \forall j \in \mathcal{L}_{0,1,\cdots,l-1}$, build transition matrix for each i,t and calculate $\widehat{p}^1_{i,t}(\mathcal{S}_{i,t};\theta)$

    $\vdots$
  - for $i \in \mathcal{L}_L$, given $\widehat{p}^1_{j,t}(\mathcal{S}_{j,t};\theta) \quad \forall j \in \mathcal{L}_{0,1,\cdots,L-1}$, build transition matrix for each i,t and calculate $\widehat{p}^1_{i,t}(\mathcal{S}_{i,t};\theta)$

- Calculate likelihood fuction $l(\theta)$
- Update θ until $l(\theta)$ is maximized

7 Results

Table 4: Parameter Estimates for Downloads AR1 Process

	(1)
	$x_{i,t}$
$x_{i,t-1}$	0.796***
	(0.01)
$\mathbb{1}(d_{i,t}=1)$	-0.997***
	(0.08)
$\mathbb{1}(d_{i,t}=1)\cdot x_{i,t-1}$	0.129***
	(0.01)
Constant	1.909***
	(0.06)
N	13056
R^2	0.711

Table 4 summarizes the parameter estimates of the AR1 process of user downloads, specified as in equation 5. The results show that once adopting Python 3, users downloads would jump from one process to another one with a higher stationary value.

In the estimation of our technology adoption model, we used two sets of data as inputs to the likelihood function. The first one is the 3102 packages after the selection

Table 5: Estimated Parameters of Adoption Model

		Full Sample	Level≥1
Nonlinear	α^x	0.078***	0.056***
Parameters (θ)		(0.005)	(0.002)
	α^μ	-0.578***	-1.061***
		(0.091)	(0.021)
	AC_0	-2.289***	-0.586***
		(0.074)	(0.174)
Log Likelihood		-3998.874	-876.32
Number of observations		13056	3167
Number of packages		3102	642

criteria detailed in Section 4; the second one also used the same data to calculate all the probability, but with the likelihood function using data excluding packages without any dependencies.

The rationale is that within the 3102 packages, 2460 (or 79%) of them do not report any dependencies. We manually checked some of the source code and found many do indeed have dependencies. The dependency information we collected are reported by the developers to PyPI when they upload the packages. So there is no guarantee that they are without measurement error. In the next version, we plan to further clean up the data by collecting the dependency information directly from the source code.

Assuming that developers do not misreport dependency information conditional on providing it, then packages in $\mathcal{L}_l$ where $l \geq 1$ depicts a much more accurate picture of the network.

Now coming to the regression table 5. Note again that our model has three parameters α^x, α^μ, and AC_0. α^x is the coefficient of the logged number of downloads. The result shows that more popular packages, i.e. those with higher user downloads, are more likely to adopt Python 3. AC_0 and α^μ measures the fixed and variable cost of adopting Python 3.

The estimation results show that packages do benefit from more user downloads. In order to adopt Python 3, each package has to go through a fixed cost AC_0; each additional Python3-incompatible dependency posts an additional cost α^μ. The two sets of estimates give similar estimates for the benefit from user downloads but very different results for adoption costs. The result using the full sample gives us an AC_0 four times as large as α^μ; whereas the sample without level0-packages says that each additional dependency has an impact twice as large as the fixed cost AC_0. The differences between the estimates from the two samples are consistent to the problem of measurement errors in the full sample. If many packages in $\mathcal{L}_0$ indeed have dependencies but fail to report them, then for these packages, the data shows a slow Python 3 adoption despite having zero dependencies. In that case, the fixed adoption cost AC_0 would seem much more important than the effect of an additional incompatible dependency.

This is just our baseline result. In future versions, more results will be provided and robustness checks conducted.

8 Counterfactuals

This section is still incomplete. Here are some of the rationales on what types of counterfactuals that we would like to run.

Katz and Shapiro (1985) predict that the success of a new technology and the speed of technology adoption depends highly on the "sponsorship". A sponsor is an entity that is willing to make investment to promote it. Python is controlled by the Python Foundation, but is distributed free of charge as an OSS. The Python Foundation has been promoting for a faster transition to Python 3. However, given the limited, probably exogenous amount of effort, it is key to understand how the effect can be spent most effectively in order to help the whole industry to switch to Python 3 at a fast pace.

In this section, we strive to gain more insights on the question of how to target certain packages in order to facilitate a fast transition to Python 3. We assume that the Python Foundation has limited effort to spent for the promoting activity, which has heterogeneous effect on the adoption decisions of packages based on characteristics of each package. We assume that the objective is to maximize the overall adoption rate of Python 3 by a certain time, say 2020. The Python Foundation maximizes that objective given limited amount of resources.

Intuitively, a good target for promotion is one that satisfies the following criteria: (1) low promotion cost (2) promotion is effective, i.e. the difference in adoption probability with and without promotion is large (3) key players in the network, namely, decisions of the targeted package can affect the decisions of many others.

9 Conclusion

Economic theory has long proposed the barrier to adopt new technology due to existing network effects. Our research explores the ways technology adoption can be affected due to the internal links amongst players. We find strong evidence that the adoption decisions of downstream players are significantly affected by their upstream counterparts.

This paper also developed a framework to measure the impact of such network on technology adoption. We extend existing dynamic choice models to incorporate the hierarchical network among Python packages. Taking advantage of the unidirectional property of this network, we detailed and estimated a computationally tractable model that allow each agent to anticipate future actions of others. We believe that the structural framework can be applied to analyze other industries with such a vertical network structure.

References

Aguirregabiria, Victor. 2011. "Empirical Industrial Organization: Models, Methods, and Applications." .

Aguirregabiria, Victor and Pedro Mira. 2010. "Dynamic discrete choice structural models: A survey." *Journal of Econometrics* 156 (1):38–67.

Bandiera, Oriana and Imran Rasul. 2006. "Social Networks and Technology Adoption in Northern Mozambique." *The Economic Journal* 116 (514):869–902.

Gowrisankaran, Gautam and Marc Rysman. 2012. "Dynamics of Consumer Demand for New Durable Goods." *Journal of Political Economy* 120 (6):1173–1219.

Gowrisankaran, Gautam and Joanna Stavins. 2004. "Network Externalities and Technology Adoption: Lessons from Electronic Payments." *The RAND Journal of Economics* 35 (2):260–17.

Jackson, Matthew O. 2010. *Social and economic networks*. Princeton university press.

Katz, Michael L and Carl Shapiro. 1985. "Network externalities, competition, and compatibility." *American Economic Review* 75 (3):424–440.

Lee, Robin S. 2013. "Vertical Integration and Exclusivity in Two-Sided Markets." *American Economic Review* 103 (7):2960–3000.

Magnac, Thierry and David Thesmar. 2002. "Identifying Dynamic Discrete Decision Processes." *Econometrica* 70 (2):801–816.

Rust, John. 1987. "Optimal Replacement of GMC Bus Engines: An Empirical Model of Harold Zurcher." *Econometrica* 55 (5):999–1033.

Rust, John and Christopher Phelan. 1997. "How Social Security and Medicare Affect Retirement Behavior In a World of Incomplete Markets." *Econometrica* 65 (4):781–51.

Ryan, Stephen P and Catherine Tucker. 2011. "Heterogeneity and the dynamics of technology adoption." *Quantitative Marketing and Economics* 10 (1):63–109.