**Getting Started with the FNNR-ABM-Primate Project**

Note: The FNNR-ABM-Primate project is J. Mak’s Thesis, involving modeling population dynamics and movement. The FNNR-ABM project, led by the PES project team (http://complexities.org/pes/), involves household PES enrollment.

Glossary

FNNR – Fanjingshan National Nature Reserve, our study site from which we have collected data (http://complexities.org/pes/research/recent-updates/).  
ABM – Agent-based Model

IDE – Integrated Development Environment

OS – Operating System

Table of Contents

(Press Ctrl+F and type in section keyword in capital letters to jump to section)

1. INSTALLATION - Have Python 3+ installed on your computer.

2. IDE – (optional but recommended) Download a Python IDE.

3. LIBRARY – Download the Python libraries needed for this project (Mesa, openpyxl, matplotlib).

4. GITHUB – Download and unzip the FNNR-ABM project files from Github.

5. READCODE - Understand the project goals, Python code, and imported libraries.

6. RUNCODE – Run the model, and understand which variables to edit.

7. THANKS – Contact project developers for more information.

[Note that Sections 1 through 4 are prepared for ABM/ Python beginners; experienced modelers can jump to Section 5]:

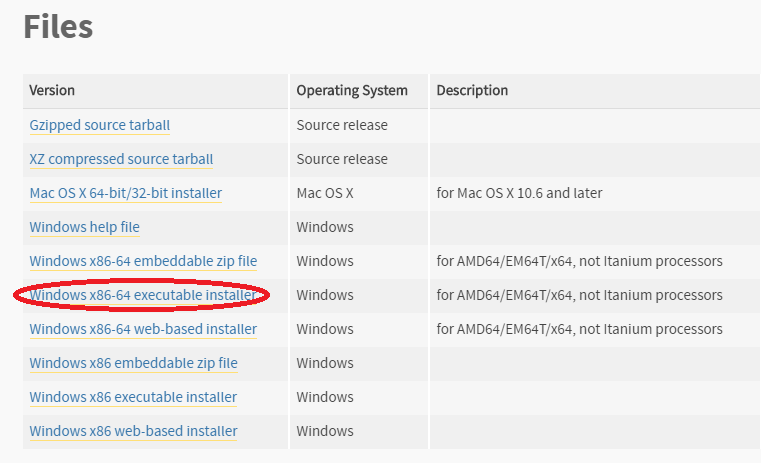
1. INSTALLATION - Have Python 3+ installed on your computer.

To download the latest version of Python, visit <https://www.python.org/>. At time of writing, [Python 3.6.1](https://www.python.org/downloads/release/python-361/) is the latest version, though again, any version of Python 3.X.X should work. Python 2.X.X is more stable for use with older systems, but it differs in syntax from Python 3.X.X, so it is not compatible with code from the imported libraries we will use here (such as Mesa).

On the Python download page, scroll to the bottom and select the option that is best for you. For the most common configuration, refer to Figure 1.1; however, it may not apply to you. First, find out if you have a Mac, Linux or Windows OS, then figure out if your OS is 32-bit (x86) or 64-bit (x64). To find this out, view your computer properties (on Windows 10, search or find ‘This PC’ in File Explorer, right-click, and select ‘Properties’ from the menu; other versions of Windows might need you to right-click ‘My Computer’). Most standard newer computers will have the 64-bit version of Windows.

Once you download and run the installer (or configure the zip file/tarball; the installer is recommended), follow the installation steps to install Python 3.X.X onto your computer. If you are not sure what options to pick, do not change the default options. Keep note of where Python is installed on your computer. If it is convenient and fast to do so, restart your computer afterwards.

Figure 1.1 – The most common option. This option may not be right for you if you are not using a 64-bit version of Windows.



2. IDE - (optional but recommended) Download a Python IDE.

There are many different software programs that will run your Python code. IDEs are optional to download because Python comes with a default one named IDLE (and for shorter python functions, one can even run code straight from the command line). However, downloading a more sophisticated IDE will handle different versions of Python and different libraries more seamlessly, as well as provide debugging/testing tools and more detailed error messages. They may also provide other tools such as a built-in file system to manage multiple Python modules (files) more easily, the ability to open non-Python files, and more.

Once you have found an IDE (google to find different ones available; the one used in this tutorial is PyCharm), follow installation instructions, unzipping/extracting any files with 7zip (a free program) or Winzip as needed. If the IDE you downloaded is PyCharm, make sure to create a new Python Project (only do this once, not every time you run the code) and place all of the FNNR-ABM-Private files inside the main Project Package.

3. LIBRARY - Download the Python libraries needed for the project (Mesa, openpyxl, possibly matplotlib).

Python has many built-in frameworks and libraries (collections of pre-written functions and modules) that save users time and effort, as well as many more libraries available on the web to download; most common projects will use at least one external library (as opposed to being coded entirely from scratch). The two libraries we must download for the project are:  
  
Mesa – a Python 3+ framework for working with agent-based models

It “allows users to quickly create agent-based models using built-in core components (such as spatial grids and agent schedulers) or customized implementations; visualize them using a browser-based interface; and analyze their results using Python’s data analysis tools. Its goal is to be the Python 3-based counterpart to NetLogo, Repast, or MASON.”

Documentation can be found online at https://mesa.readthedocs.io/en/master/

Or by downloading the .pdf at https://media.readthedocs.org/pdf/mesa/latest/mesa.pdf

If you install Mesa through pip (covered later here), it will come installed along with the other libraries it depends on, such as Tornado (web framework), Pandas (data structure library), Numpy (for a variety of numerical expressions or generations), Six (for wrapping over differences between Python 2 and 3), Tqdm (progress meter/Jupyter Notebook-related), Matplotlib (for plotting, and more). The user will likely not directly access these libraries when working with Mesa, but they should be aware of what the libraries do.

Matplotlib – this may come with Mesa, but just in case you get an error related to this, you may need to download matplotlib separately, since it is used explicitly and separately from Mesa in the model. It helps build graphs to display Python data.

The most common (and Pythonic) way to install external libraries is to open the Command Prompt on Windows (cmd.exe, which is a terminal/shell for users to run administrative system commands. and type in:

pip install mesa

Before you do this, read the troubleshooting section to see if you have set up your environment correctly; if you have, your current directory should not matter. If you are using conda or miniconda (or another environment/package manager like PyCharm), replace ‘pip’ with ‘conda’ in the above commands. Ensure that you are in the right directory while running commands.

**Troubleshooting**

There are a number of possible error messages you can get. The instructions below diagnose them based on Windows 10.

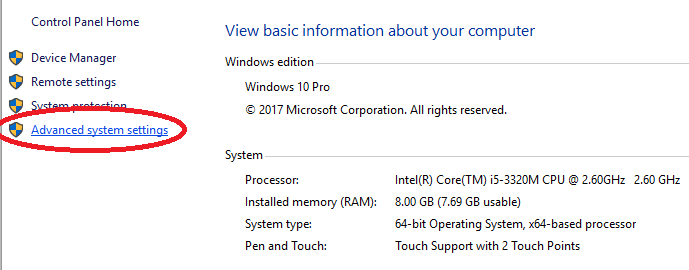
*If ‘pip’ is not recognized in the command window:*

1. Set the Environment Path.

a. This PC or My Computer > Right-Click > Properties

b. Click on ‘Advanced system settings’ on the left tab.

Figure 3.1 – Setting System Environments Before Using Pip



c. Select ‘Environment Variables...’.

Figure 3.2 – Advanced System Settings

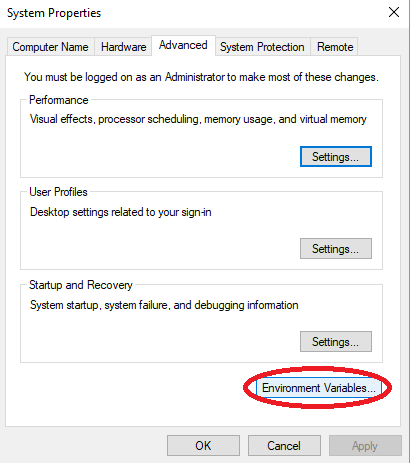
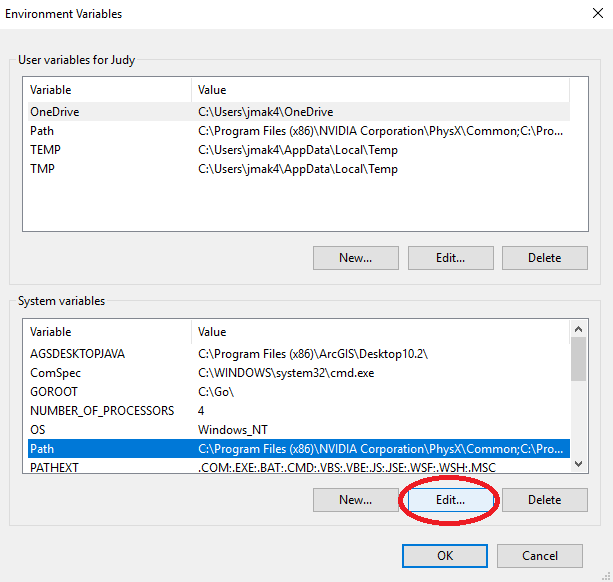
d. Select ‘Path’ under ‘System Variables’ (near bottom of the window, not the first ‘Path’ near the top), then ‘Edit...’.

Figure 3.3 – System Variables

e. Select ‘New’, then ‘Browse...’ to find where your Python installation is. Common file paths to add here include (depending on where you’ve installed Python):

C:\Python27 ← likely

C:\Users\<YOUR USERNAME>

C:\Users\<YOUR USERNAME>\Downloads

C:\Users\<YOUR USERNAME>\AppData\Local\Programs\Python\Python36 ← most likely

Basically, add the directory that contains the same version of python.exe that you want to run, and maybe others, such as your Downloads folder, to be safe. Make sure that when adding new file paths, you do not overwrite or delete old ones in the current list.

\*NOTE: If you have multiple versions of Python installed, make sure that the Python version you want is moved up above the other version(s). To do this, select the ‘Move Up’ button in the Path > Edit… window. Now you should be able to run pip in the command line to install the necessary libraries.

2. Change the CMD Directory.

‘cd’ is a command that changes directories to what the user types.

Once you are in the correct parent directory, your system should take care of the rest.

For example, if you have Python installed under C:\Users\<YOUR USERNAME>\AppData\Local\Programs\Python\Python36, then in cmd.exe, you may want to type:

cd C:\Users\

in order for cmd.exe to change its directory and look for pip in the right drive.

Another useful shell command is dir, which shows folders/files in current directory

*If it installs successfully in the wrong directory, or if your IDE does not recognize the library after installation:*

3. If you are using Anaconda/Miniconda and the library has installed the library in the wrong environment (such as one for a version of Python 2.X.X), set up a new environment; otherwise, skip to Step 4.

To set up a new environment, type the following into the Command window:

conda create --name 3point6 python 3.6

Note: ‘3point6’ here can be any name you wish, and ‘3.6’ can be changed to another version of Python.

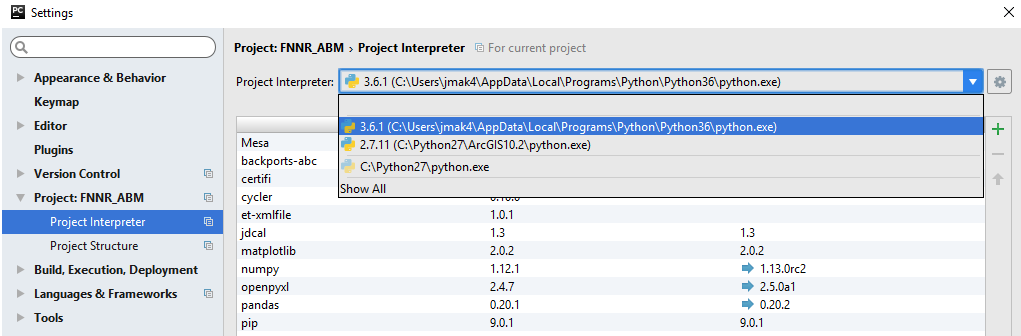
Then activate the env in the Command window:

activate 3point6 (or whatever you named it)

You should be able to use the pip command to install the needed libraries under this new environment now. After you do so in the command line, proceed to Step 4.

4. Change your IDE/project interpreter.

Figure 3.4 – Setting Project Interpreters in PyCharm



This varies per IDE, but in PyCharm, you will want to go to File > Settings to set the Project Interpreter. The Project Interpreter should either be wherever you have the desired version of Python installed, or in a Conda environment that has the desired version of Python installed (see Step 3). You will know you have selected the correct environment when:

- the Python version shown is 3.X.X, and

- the libraries shown in the table under the Interpreter selection dropdown box include pip, matplotlib, openpyxl, pandas, Mesa, and more (assuming that you have successfully used pip or conda in the command line to install the libraries at this point).

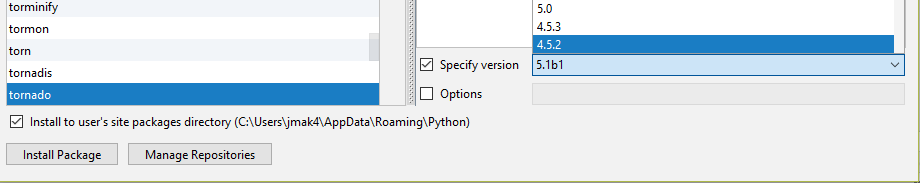
**Updating Libraries**

There are multiple ways to ensure you have a specific version of a library. Pip and conda are two ways, but I prefer to use PyCharm’s Project Interpreter under File > Settings (see Figure 3.4) to install libraries.

**Note: tornado should be version 4.5.2, not the latest version!**

Double-click a library name in the Project Interpreter to install, update, or revert a library. In the case of a new user, they will probably want to revert their tornado library. The screenshot below shows how to select version 4.5.2—even if you have a later version—and install that package in PyCharm.

Figure 3.5 – Reverting to tornardo version 4.5.2



Finally, if your libraries appear to have successfully installed to the same directory that runs the desired version of Python as configured in your IDE, but you still get import errors, you may need to restart your computer.

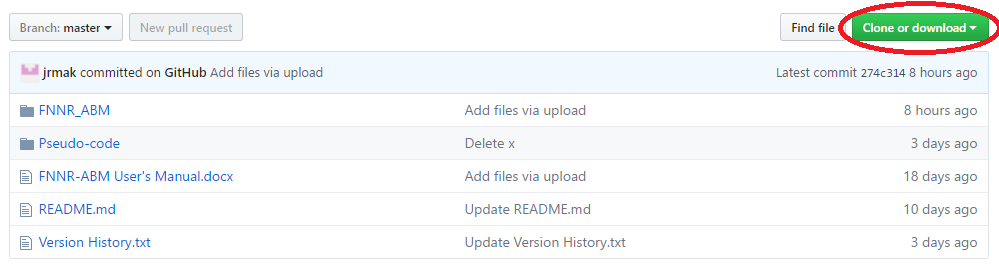
4. GITHUB – Download and unzip the FNNR-ABM-Primate project files from Github.

Github is a file hosting, sharing, and version-control website and tool, particularly for code files of various computer languages. It functions similarly to Google Drive, but for code. For the purposes of this project, it is useful for sharing and storing different versions of .py files in an easily-readable format for others to download, edit, share, and track version histories of file changes. In that respect, it resembles an advanced version of Google Drive or Google Docs, but specifically meant to be used for collaborative code sharing. While Github is a powerful suite that comes with its own commands and software, named Git, we will only cover how to navigate the website’s basic functions here.

This project’s Github repository is located at <https://github.com/jrmak/FNNR-ABM>-Primate.

Once you navigate to the page, the files from the repository will be shown; the organization is similar to Window’s file system. To download the files, find the green button to the right that says “Clone or download.” This can only be done from the main repository page, and not individual files, though one may also copy and paste code directly from viewed raw files.

Figure 4.1 – Downloading Github Files

Change the above figure (using a graph from your site at <https://github.com/jrmak/FNNR-ABM>-Primate)

Shown above is an example from another project, but the green ‘download’ button should be in the same place.

Make sure you are downloading the correct branch/version of the code (usually ‘master’). In this example, the master branch (see the leftmost grey button at the top of the image) is selected by default; it is usually the latest stable version of the project available. Once you click the green button, select the option on the right (download as .zip), unless you have Github for desktop installed and specifically want to edit the files from there (not be covered in this tutorial). The repository’s files will then be compiled into a .zip file; unzip it using WinZip or 7zip. The name of the folder should resemble ‘FNNR-ABM-Primate-master.’

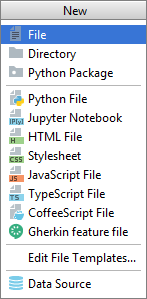
Once these files are downloaded to your computer (you probably only need the code files within FNNR\_ABM-Primate) and unzipped, move them to the appropriate location on your hard drive. If you are running a very basic setup with IDLE (the default Python IDE) it is possible to keep these files in your Downloads folder (in Windows), but the steps below are recommended.

**You only need to do the following the first time you download the files:**

Using PyCharm, create a new PyCharm project by File > New Project… in the upper left corner. Name the project whatever you want (I used ‘FNNR\_ABM\_Primate’) and place the project wherever you want (I prefer to keep the default under PycharmProjects, then once the project has been created, create the Python package using File > New… as shown in the figure below (Select ‘Python Package’ from the drop-down list). Name the Python Package FNNR\_ABM\_Primate, even if you have already named your project that.

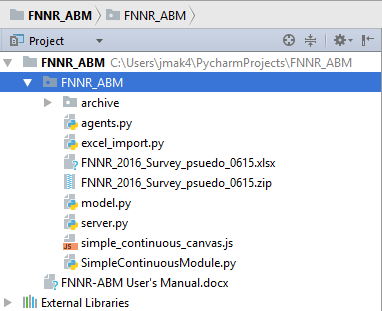
Note that in general, if you would like to start a new .py file to code and save, select ‘Python File’ and not the generic ‘File’ type at the top of the menu (see below).

Figure 4.2 – New ‘Python File’ (not the selected ‘File’)



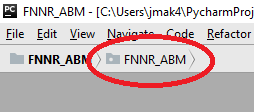
From the FNNR-ABM-Primate file you have on your computer, copy and paste your code (.py) files into PyCharm under your Python Package folder directory. Also paste the latest copy of the unzipped .xlsx Excel data file. The result on the left-hand window should look similar to this:

Figure 4.3 – Project Structure (Only create a new project once)

(Again, replace ‘FNNR-ABM’ in Figure 4.3 with ‘FNNR-ABM-Primate’)

Note that if this window is not visible, you may need to toggle it to show within PyCharm. One way is to double-click the Python package’s tab:

Figure 4.4 – Showing Visible Windows

Note: even though the example in Figure 4.4 shows ‘FNNR-ABM’, you would actually be showing the ‘FNNR-ABM-Primate’ project.

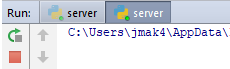
The code should now be visible whenever a module is opened from the left-hand side directory. To run the code, open server.py (it must be this module for this project specifically) and press Shift + F10 or run it manually under the Run… option at the top toolbar. Afterwards, server or graph should be the default module set to run next to the green buttons (Fig. xx) on the top right corner, as pictured below; you can also click the green triangle to run.

Figure 4.5 – Run ‘server’ (Despite the screenshot, it should not say ‘server (1)’)



\*Once you run the code, make sure you exit the instance afterwards. You can do this by either closing any windows the code opens (whether it is a web browser tab or all matplotlib graph windows) or by clicking the red rectangle (symbolizing ‘STOP’) in PyCharm next to the bottom-left or top-right portions of the screen. Make sure you end all instances of “server” running. When in doubt, you may also press Ctrl + F2 to terminate all processes running. If you are looking at a graph or running the visualization in the web browser, this will close the windows.

Figure 4.6 – Stopping All Processes



5. READCODE - Understand the project goals, Python code, and imported libraries.

The goals of the FNNR-ABM-Primate project are as follows:

- Simulate and record changes to monkey population structures and dynamics through slider inputs; and

- Simulate and compare agent movement in a visualization given different input environmental grids.

- Understand monkey habitat use behavior from a bottom-up approach

This section is comprised of a detailed description of each module, its functions, and how the Mesa framework supports the code.

There are two parts to the model: the visualization of agent movement and a demographic population structure model. Both parts run at the same time, and share the same agents, but only one result (either visualization or population graphs) can be calculated and shown. The other modules are shared between them.

The model is run in steps; every step represents five days in a year. During each step, a multitude of functions execute. The function parameters are determined based on real-world data (Yang, Lei, and Yang 2002); they also utilize some random generators in limited situations where accurate information is not available or uncertainty exists.

This model runs on the Mesa framework, which is specifically designed to use Python to create agent-based models. Mesa contains the following tools (as well as more not covered at this time):

- Defined Agent and Model superclasses that allow an ABM-style relationship between the two, as illustrated by time-steps

- Visualization and data-graphing components (web simulation, data collector)

Below are the modules that comprise the model’s code.

**model.py** – contains the main model structure (movement class); initializes agents.

First, monkey families are randomly generated, then members within the families are generated. The starting population structure models the pseudocode rules uploaded to GitHub based on probability categories (for example, monkeys aged 7-10 comprise 20% of the population, so if a random number generator from 0 to 1 brings up a number between 0.42 and 0.62—a range of 0.20—the resulting age is a random number from 7 to 10; Yang, Lei, and Yang 2002).

After the families are generated, they are placed on the grid to move in the visualization model. The grid values themselves (attributes of these grids such as elevation, slope)—which will be weighted to determine movement direction each step—are read from a .txt file in the same directory.

Every step, the model runs, and each agent (family agents for the visualization and monkey agents for the population model) follows their rules as set in agents.py.

**agents.py** – contains the Family and Monkey agent classes (as well as defining how the agents react to the environmental grid (which in turn represents variables such as elevation, vegetation, slope, etc.)—in the current version of the model, shown as the different elevation categories as defined by colors; in later versions, an imported Maxent grid). The Family agent class determines the behavior of the pixels in the visualization; the Monkey agent class determines the behavior of each individual (e.g., birth, death, growth) in the population / demographic structure model.

Each agent goes through the following events, in addition to aging in each time step:

1) Both male and female – birth or death; death occurs according to mortality probabilities for each age category, as set in the pseudocode with original data from Yang, Lei, and Yang (2002);

2) Female only – possibility of giving birth every 3 years in April or September, or giving birth in the year when a child has died recently;

3) Male only – possibility of breaking off into an all-male group when enough males over the age of 10 are flagged.

This module agent.py may be broken up into shorter parts later.

**maxent.py** – contains the environmental grid class, which contributes trivially to understanding the model, but is necessary code-wise.

**model\_for\_graph.py** – contains redundant functions that would otherwise take up space in model.py and contribute trivially to understanding the model, but are necessary code-wise.

**excel\_export\_summary.py** – exports results of the population sub-model to an Excel (.csv) file when graph.py is run.

**aggregated\_dem.txt** – the ASCII file that contains the environmental data the monkeys move upon. Will be replaced later with maxent.txt.

7. RUNCODE – Run the model, and understand which variables to edit.

The code can be run in multiple ways, depending on what data you need to access. You may execute (run) one of the following modules:

**server.py** – run this module if you would like to see a visualization of monkeys’ family (small-group in terms of Yang, Lei, and Yang 2002) movement through the FNNR in a given year (73 time-steps).

**graph.py** – run this module to see graphs of the population changes over years, including births, deaths, and at the end of the model run, the demographic structure. This also updates (or creates) a .csv file listing the age and sex structure of the monkeys in the reserve.

8. THANKS – Contact project developers for more information.

If you run into an error while running the code or have other questions beyond this document, contact project developers for more information.

Project Faculty Supervisor – Dr. Li An

Project Developer – J. Mak (email via GitHub)

You may also directly comment on the Github repository at <https://github.com/jrmak/FNNR-ABM>-Primate.

Special thanks to all involved in the FNNR Project (available at http://complexities.org/pes/people/).

References:

Yang, Y. Q., X. P. Lei, and C. D. Yang. 2002. *Fanjingshan Research: Ecology of the wild Guizhou snub-nosed monkey (Rhinopithecus bieti)*. Guiyang: Guizhou Science Press.