Time Series Analysis in Python

Wenyan Gong, Zongxi Li, Cong Ma,
Qingcan Wang, Zhuoran Yang, Hao Zhang

January 15, 2017

1 Project Objective

Time series analysis comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data. It is widely used in signal processing, pattern recognition, mathematical finance, weather forecasting, earthquake prediction, control engineering, and largely in any domain of applied science and engineering which involves temporal measurements. As in Figure 1, we plot the USD to CNY exchange rate.

In this project, we will play a game with time series in finance. It has gained its popularity in Wall Street recently, since it is fundamental to most promising quantitative investment strategies. We develop a system that can predict future prices of stocks using various kinds of methods for time series analysis.

Figure 1: USD to CNY exchange rate. Jan 12 2016 - Jan 12 2017. Source, Yahoo finance.

Given input of a stock price series, our system will first fit some powerful and popular time series models, such as the autoregressive (AR) model and the moving average (MA) model. This procedure will give you the estimator of the parameters in these models.

The most important ingredient in estimation is the optimization procedure. Users can select one of optimization methods in the system based on their preference. The optimization method includes but not limited to gradient descent and gradient descent with momentum.

After model fitting and estimation, our system provides a fast way to do statistical inference. Users can do different kinds of statistical test as well as obtain confidence intervals. Moreover, with fitted model, future price prediction is made and it's compared with real price data. Moreover, we provide different methods to assess the prediction accuracy, which will be

visualized afterwards. With the identified model, we can further consider trading strategy and option pricing.

More interestingly, if users input multiple stock prices, we can divide these stocks into different groups, which is called clustering. Among each group, stocks share certain degree of similarity. Different clusters will also be visualized. With collection of stocks (e.g., S&P 500), we can exploit the correlations within them and build a reduced order model for price prediction. For example, principal component analysis can be used to extract dominant features in the financial market.

2 Design Process

Given the tasks, there are many factors to consider. First, we need to specify the programming language to use. We decide to use python, since it's free, and widely distributed. Moreover, there are many powerful packages, e.g., `numpy` (linear algebra), `scipy` (scientific computation), `matplotlib` (plotting library), etc.

We make use of existing packages like `numpy` and `matplotlib`, but largely other parts are made by our group. For instance, we implemented various optimization methods like the (stochastic) gradient descent (GD), and GD with momentum.

`Github` is used to track the progress of this project, and streamlines the collaboration within our group. `Github` is a powerful and convenient tool that really makes a difference in group development of codes.

We make use of `Doxygen` to generate documentation for our project. The users can view various classes of this project in an interactive way via a web browser.

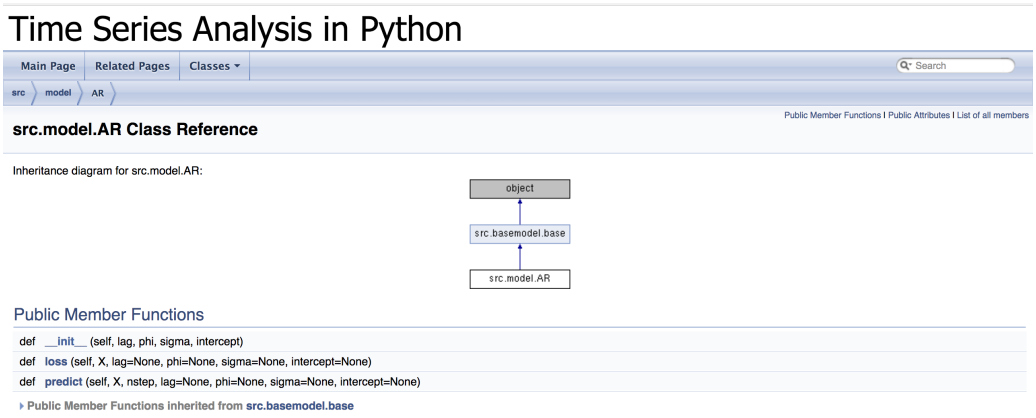


Figure 2: Doxygen documentation

Along the way, we have adopted the philosophy of object oriented programming (OOP) by carefully designing classes which include relevant functions. For instance, in the Reduction class, we implemented various model reduction methods including principal component analysis, independent component analysis, and dynamic mode decomposition.

Part of our code are tested using Python unittest, since there are analytical solutions. For more complicated functions, they are tested using canonical examples, and by comparing with the outputs of standard machine learning package scikit-learn.

We have a few milestones for this project. (1) Prototype, Dec 15. In this release, we had at least one implementation for each step. Then given a time series data, we could apply at least one method from each class to do the analysis. (2) Alpha version, Jan 1. In this release, we completed most methods in each class. (3) Final version, Jan 12. In this release, we finalized

the project. Finish all the testing using various data including simulated data and real financial data.

3 Architecture

The high-level program structure is shown below. The division of work is pretty even, and there are some minor work that are too trivial to mention extensively here.

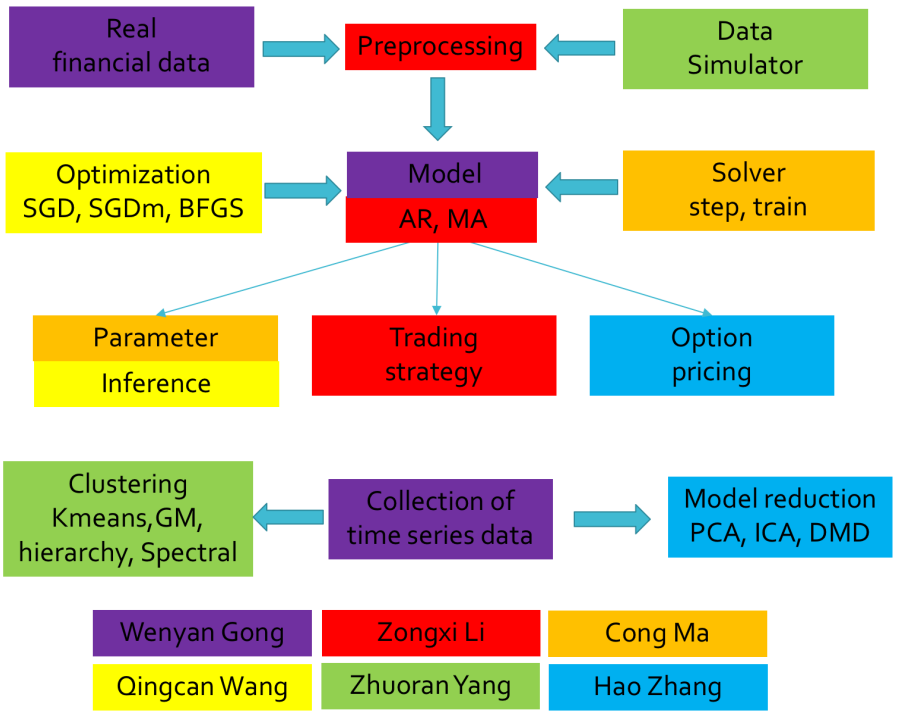


Figure 3: Program structure and division of work

Overall, the program consists of two parts. The first part deal with a single time series, and exploits the time correlation within the time series. There is data preprocessing module that takes the raw data and get the right data format for later analysis. Optimization, model, and solver are used all together to identify the models. Finally, there is a post processing module that makes use of the information from model. The post processing module includes parameter inference, trading strategy, and option pricing. The second part deals with a collection of time series of data, basically it exploits the correlation between different time series. In this way, we can gain more insights into the financial market. While these insights are impossible for a single time series.

3.1 Data Preparation and Preprocessing

In this package, we provide methods to simulate data from specified time series models and a real-world dataset, thus enabling both theoretical research and applications.

For simulated data, we provide functions that samples from the `AR(p)`, `MA(q)`, `ARMA(p,q)`, `GARCH(p,q)` models. Given the parameters of the model, we obtain a $n \times T$ matrix storing n independent realizations of the time series with time T. Since the `for` loop in `Python` is slow, we integrate `c` code with python using `Cython`.

For real data, we provide `S&P 500` dataset, which consists of the prices of the Standard & Poor stocks in a consecutive 490 days. In addition, we also preprocess the raw data, thus

enabling users to directly apply various methods on the dataset. More specifically, we provide functions that can transfer the price time series to return time series and do the reverse transferring. For instance, if the user inputs daily price of `Google` stock, then we can output the daily return of `Google` stock. If the user inputs the daily return, we are able to output the daily price. Moreover, we can detect the peak and trough for a given time series, which will be used for developing trading strategy. For example, if you input `Google` stock price series, we can detect the highest price and the lowest price as well as their location in the time series.

3.2 Model

For a given time series, there are two kinds of model class that can be used to fit the time series. One is the autoregressive (AR) model. The other is the moving average (MA) model. For details, see [3]. Inside each class, we provide three functions, initialization, loss function and prediction function.

To be specific, we can first initialize the model with given parameters. Secondly, the loss function is able to calculate the current loss as well as the gradient of negative loglikelihood function, based on the current parameters in the model. This is crucial, because the loss and gradient will be the input of the optimization method, which will in turn return the updated parameters to the model to calculate new loss and gradient. Finally, the prediction function is able to predict the future time series based on historical time series and fitted parameters.

3.3 Optimization

After the model type being fixed, there would be certain loss function regarding the selected model. The key of model fitting is to decide the parameters, which are derived by minimizing the loss function. Therefore, certain optimization techniques should be applied during the fitting.

In `optim.py`, we provide several optimization methods including stochastic gradient descent, stochastic gradient descent with momentum and Broyden–Fletcher–Goldfarb–Shanno (BFGS) method [2]. The methods will be called by the `Solver` class, which take the current iteration point, the gradient value and the configuration parameters as input and return the next iteration point. The user can choose the optimization methods according to the scale of the problem.

3.4 Solver

We develop a solver class, which serves as a bridge between the model class and the optimization methods. Specifically, given the loss function and the gradient functions specified by the model and the optimization methods, the solver uses this method to minimize the loss until it converges.

3.5 Inference

In addition to parameter estimation, we also provide inferential methods that enables users to access the uncertainty of the estimated model. After all, once the model is estimated, it is still not clear how reliable our estimation is. In fact, even for the pure white noise, it is possible to estimate a statistical model, even though that model is not meaningful.

In specific, we provide functions that estimates the auto-covariance auto-correlation functions (ACF) and perform the Box-Ljung test that checks if the ACF is significant. See [3] for a more systematic treatments of statistical inference for time series models. In addition, we provide functions that select the order of `AR(p)` models, estimate the parameters, and test whether the parameters are significant.

3.6 Trading Strategy

The software is able to generate trading signal based on predicted price. To be specific, it can detect which time point to buy and which to sell. Please see function `signal_generation()`. Moreover, it can calculate the profit and loss based on existing trading signal, which can be realized by `profit_loss()`

The most important function for the trading is `rolltrade()` , which can automatically do the trading with historical data and fitted model. This function will call other functions from the "model" as well as the functions that generate signals and calculate the profit. It will finally return the predicted price, trading signal and profit time series. Moreover, there are many options for the trading, for instance, you can specify the trading frequency, the prediction length and the initial wealth.

3.7 Option Pricing

In finance, an option is a contract which gives the buyer (the owner or holder of the option) the right, but not the obligation, to buy or sell an underlying asset or instrument at a specified strike price on a specified date. Option pricing is an important issue in financial market.

In the software, we develop a `OptionPricing` class to calculate the option price. The class takes the strike price K, the expiry time T, the risk-free interest rate r and the volatility (standard deviation of the stock's returns) σ as parameters to generate an object. Then the class provides an method to solve the Black-Scholes differential equations, and returns the option price $V(S, t)$ given the current time t and underlying stock price S.

3.8 Clustering

In order to handle the cope with heterogeneous datasets, i.e., datasets with various characteristics, we provide clustering methods that enable users to perform data analysis with more accuracy. By utilizing clustering tools, users may discover hidden structures shared by a small subgroup of the dataset that are buried due to the large scale of the hole dataset.

To better understand the importance of clustering, let us consider the example of stock market, which consists of stocks that belong to various sectors. Although there is a global trend of the market as a whole, various sectors may exhibit different, even converse movements. Thus, global information can be too crude to perform more fine-grained analysis; it would cause huge error if the user predicts the trend of energy stocks using data coming from the whole market.

In the package, we provide a variety of commonly used clustering methods, which include k-means, hierarchical clustering, spectral clustering, and Gaussian mixture modeling. See [4] and [1] for introduction of clustering methods. Our methods yield results comparable with other machine learning packages in `python`. In specific, in `demo_clustering.ipynb` we showcase our method and compare with the `Scikit-Learn` package using the `S&P 500` dataset, which consists of the prices of 470 stocks in a period of 490 days. A concise illustration is provided in the next section.

3.9 Model Reduction

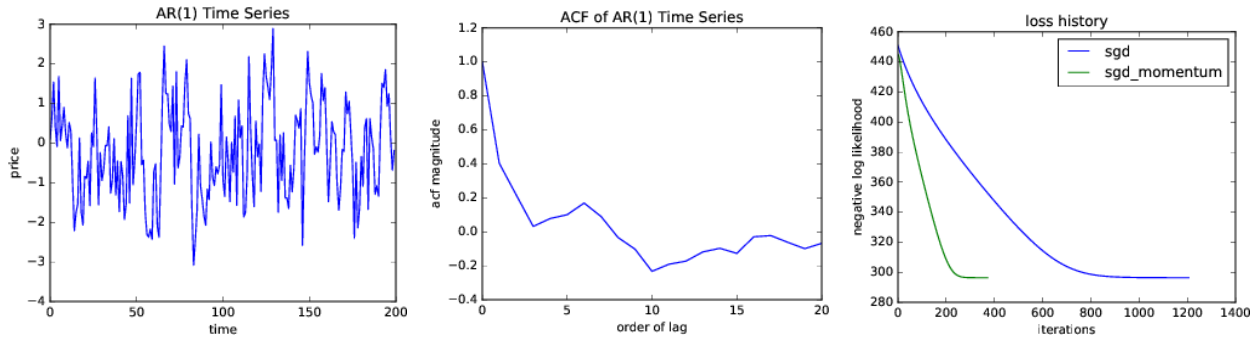
In order to exploit the correlation between various stocks, we develop a model reduction module. As we know, tech companies are expected to have the same stock price trend. Google and Apple might have the same long term behavior. If we view the stock prices at a given time instant as a vector, then as time progresses, the vector changes with time. Principal component

analysis (PCA) [6] finds dominant structures in the stock market, and the principal component provides a good basis for low dimensional description. Independent component analysis (ICA)[5] finds the hidden independent signals that generate the stock data. The hidden signals are independent in the sense that they are most non-gaussian. Dynamic mode decomposition (DMD) [6][7] fits a linear model that transfers the current state to the next state. Given this linear operator (a matrix), we can find its eigenvectors (DMD modes) and eigenvalues (DMD eigenvalues). If we project the initial condition onto leading DMD modes, then each mode evolves with a given frequency (DMD eigenvalue), and this gives the temporal evolution of states.

4 Demos of results

4.1 Modal validation

In this demo, we use simulated data to validate the models we wrote. In the first place, we generate a time series of length 200 from the `AR(1)` model with parameters $\phi = 0.5$, $\sigma^2 = 1$ and intercept $= 0$. The plot of this time series can be seen in Figure 4(a). After generating the time series, we plot the auto-correlation coefficients with respect to the order of lag as in Figure 4(b). As we can see, the estimated order is around 2. To test the solver we wrote for the model, we specify the lag to be 1, which is the same as the parameters for the simulated data. Then we use the solver to conduct maximum likelihood estimation on the simulated data. Figure 4(c) shows the loss history for two different optimization methods. As we can see, the loss steadily decreases for both methods and converges to the same value. The difference is that with momentum updates, the loss function converges faster. And this matches the theory that gradient descent with momentum term has better convergence rate. In the end, the solver returns the parameters of the fitted `AR(1)` model: $\phi = 0.49$, $\sigma^2 = 1.02$ and intercept $= 0.02$.



(a) Time Series (b) Auto-correlation Coefficient (c) Loss History

Figure 4: (a) shows the time series generate from the AR(1) model with parameters $\phi = 0.5$, $\sigma^2 = 1$ and intercept $= 0$. (b) shows the auto-correlations with respect to the order of the lag. (c) shows the loss history of two different optimization methods.

4.2 Financial data

With our package, we would be able to play with real financial data: to fit a suitable model and make highly reasonable predictions. In this section, we take the stock return of Google from year 2015-2016 as an example. There are altogether 490 trading days during the time. The stock prices and returns are shown below:

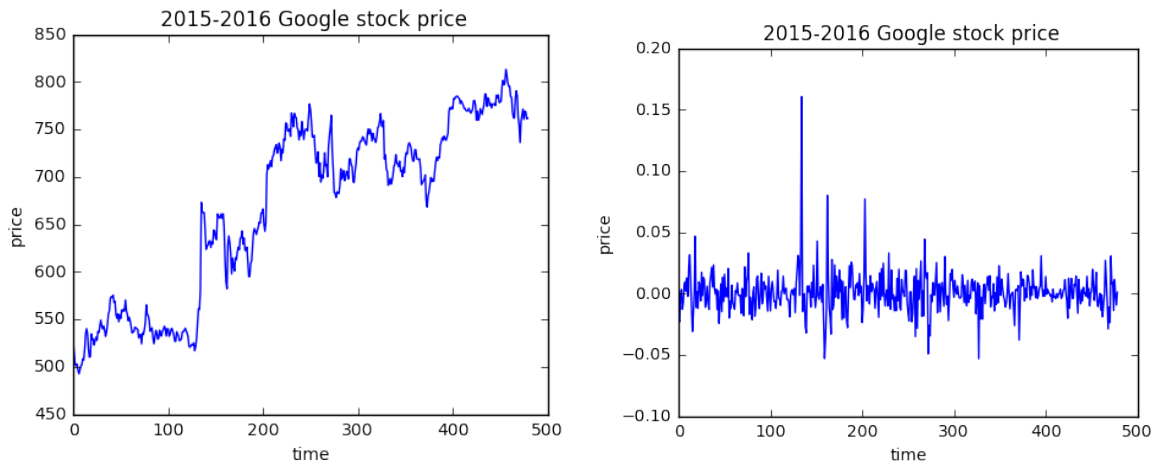


Figure 5: Left, Google stock pirce. Right, Google stock return.

We may fit an `AR(5)` model according to the return in the first 480 trading days. Here, the lag value is picked in a moderate size to balance the complexity and validity. By choosing the optimization method to be stochastic gradient descent with momentum update, a fitted model is quickly returned. With the fitted model, we would be able to predict the future return as well as the future stock price for the following days. Here, we make a prediction for 5 days.

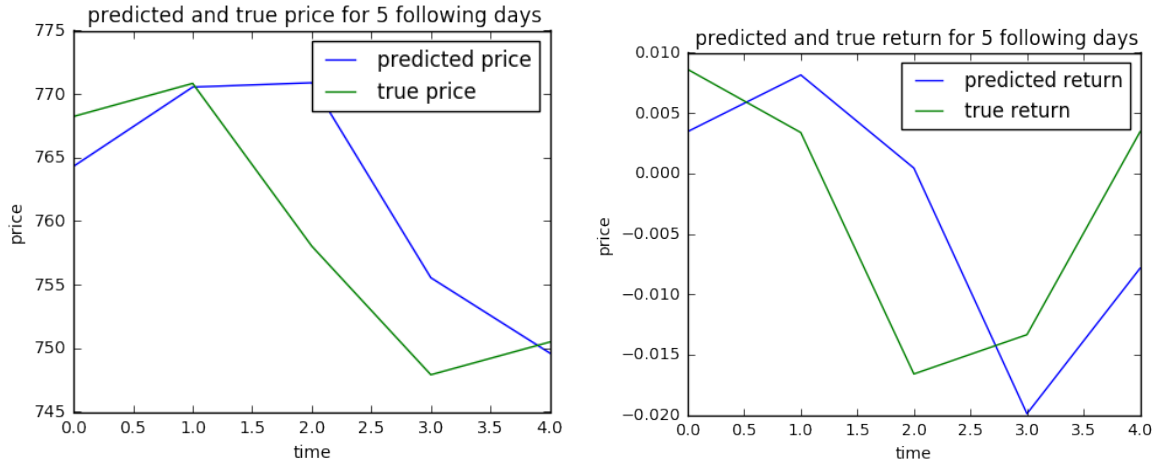
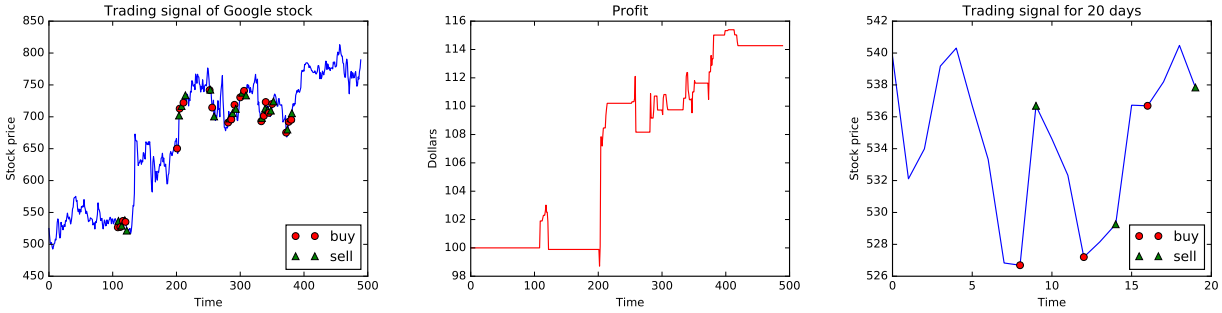


Figure 6: Left, predicted Google stock price. Right, predicted Google stock return.

As we can see from the figure, we correctly predict the trend of the stock price. Moreover, the predicted price and the true price are very close. This lays a good foundation for the following trading section.

4.3 Trading strategy

In this section, we demonstrate how we can use our code to do the trading strategy. We pick the `Google` stock price from 01/02/2015 to 12/09/2016, which contains 490 trading days data. To begin with, we made use of the first 100 data points to fit `AR(5)` model, and then predicted future prices. This part is similar to the above section, so we will omit the details, including the optimization procedure. After getting the fitted model, we input it as well as some parameters into trading package. There, we specify the parameters as follows. The training length is `l` $= 100$. The prediction step is `nstep` $= 20$. The length of trading window is `window` $= 5$. And the initial money to invest is `money` $= 100$.



(a) Trading signal (b) Profit curve (c) 20 days trading signal

Figure 7: (a) shows the trading signals over 490 days. (b) shows the profit over 490 days. (c) shows the trading signals over 20 days.

After running the package, we can generate trading signals and profit curve, which are shown in Figure 7. As we can see from the first picture, the red ball is the buying signal and the green triangle is the selling signal. Our signal is very effective as is shown in the third picture in Figure 7. We always buy at relative low price and sell at high price. In addition, our strategy is really successful, which can be seen in the second picture. Our final profit is about 15%. We can do the trading with other model estimation methods, but we don't show demo here due to the limit of pages.

4.4 Option pricing

This demo shows how to use the `OptionPricing` class to calculate the price of an option. We choose `Google` stock and use 490 daily price to calculate the volatility. We set the strike price $K = 800$ and expiry time $T = 90$ days. Based on the the Black-Scholes model, we can calculate the call option price $V(S, t)$ given current time t and stock price S, showed in Fig 8. When viewing V as the function of S, there is a sharp corner at $S = 800$ when $t = T$, and the function is smooth at $t = 0$, which is the same as the real situation in the market.

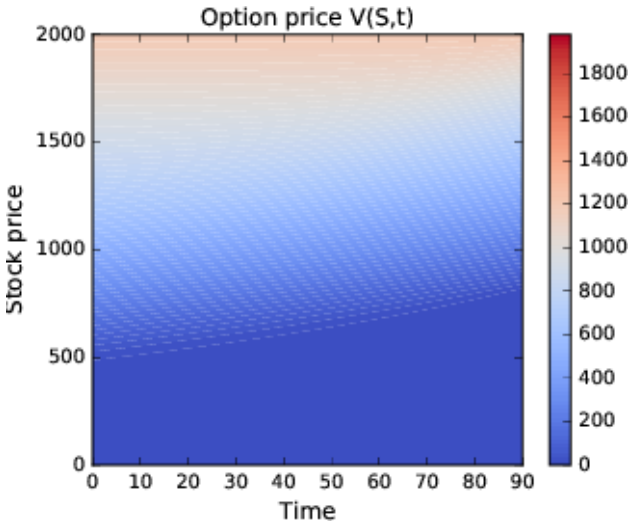


Figure 8: Option Pricing

4.5 Clustering

To illustrate our clustering methods, we apply our methods on the `S&P 500` dataset and compare with clustering methods provided in the `Scikit-Learn` package. We set the number of clusters to 5 and record the amount of time needed for each of the methods, which is reported in Table-1. In addition, to visualize the clustering results, we plot all the 470 stocks with labels in different colors. As shown in Figures 9–11, our methods are comparable with `Scikit-Learn`.

	K-means	Spectral Clustering	Hierarchical Clustering
Our Package	0.4423 s	0.4749 s	0.1838 s
`Scikit-Learn`	0.1308 s	0.4022 s	0.1288 s

Table 1: Running time of Clustering Algorithms implemented by our package and `Scikit-Learn`. As shown in the table, our package provide comparable implementations

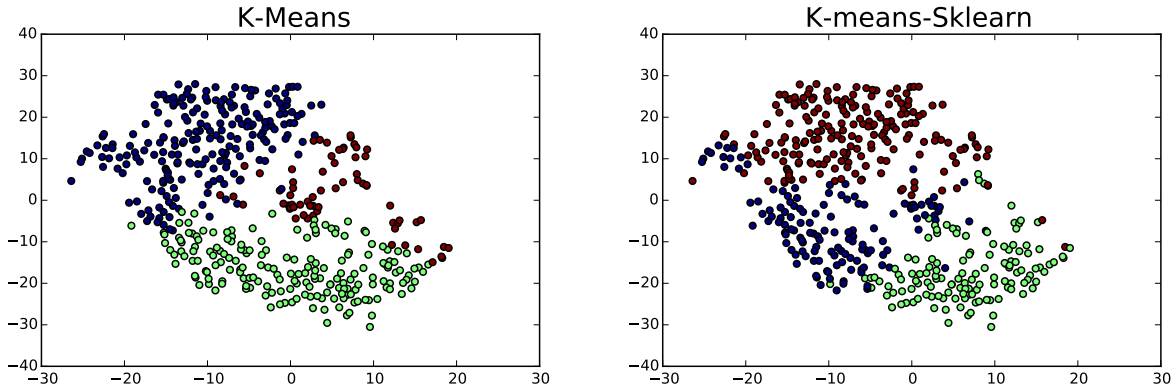


Figure 9: K-means clustering for the `S&P 500` dataset using our package and `Scikit-Learn`.The plot of our clustering result is on the left.

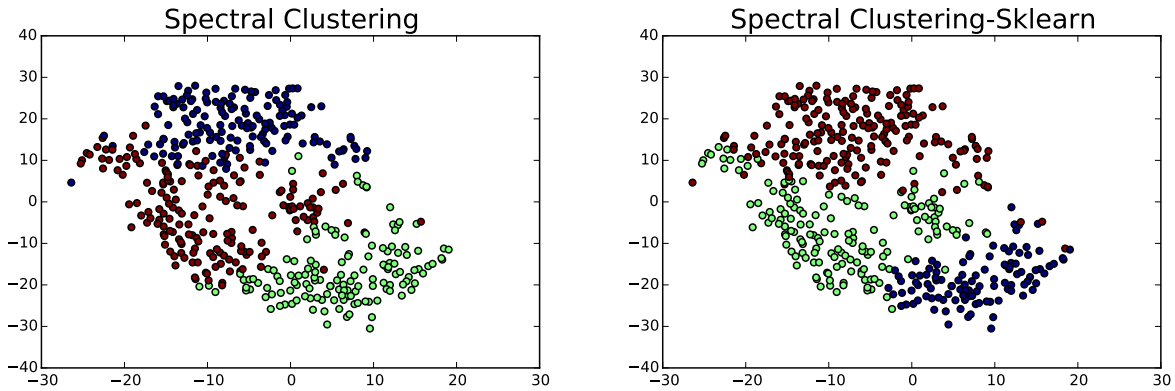


Figure 10: Spectral clustering for the `S&P 500` dataset using our package and `Scikit-Learn`. The plot of our clustering result is on the left.

4.6 Model reduction

In order to demonstrate the use of model reduction in financial market, we will take `S&P 500` data as an example. The data we use consists of 470 stocks, and 490 transition days. So the

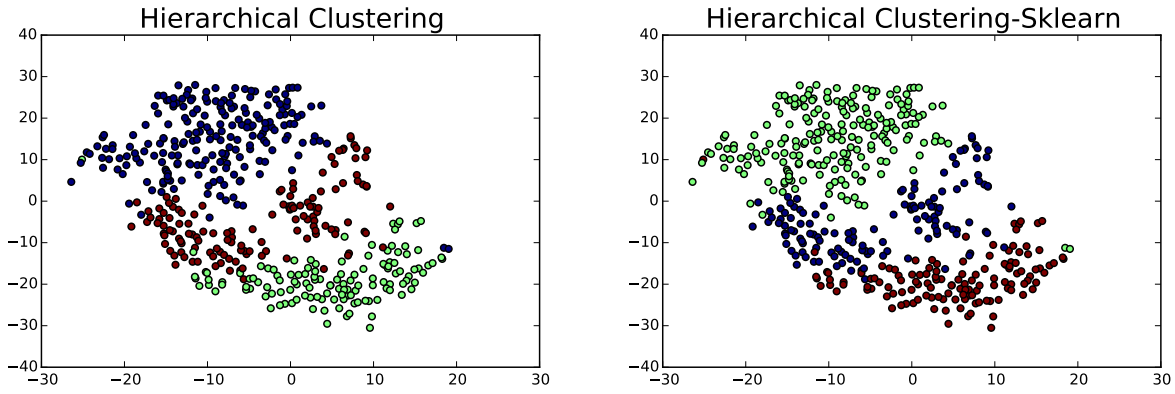


Figure 11: Hierarchical clustering for the `S&P 500` dataset using our package and `Scikit-Learn`. The plot of our clustering result is on the left.

state dimension is 470. ICA does not work well in this case. We will apply PCA and DMD. Using the PCA and DMD outputs, we predict the stock price for the first stock in `S&P 500`.

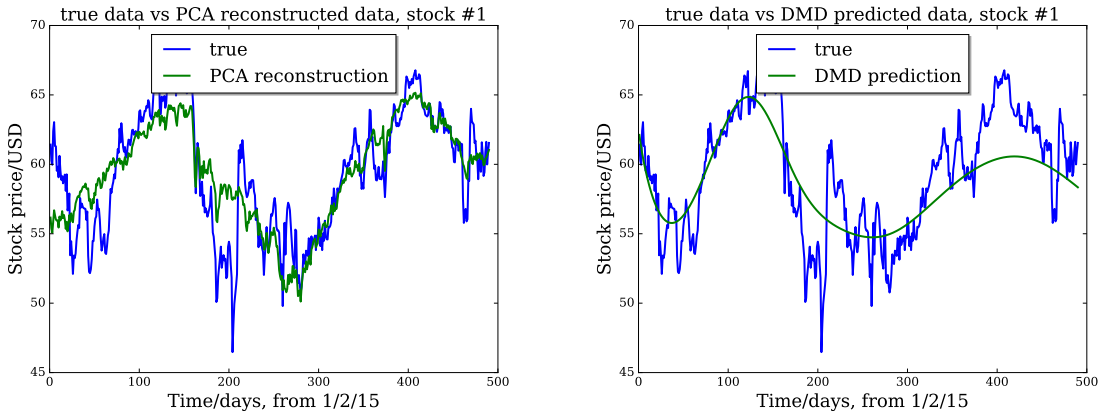


Figure 12: Left, PCA. Right, DMD.

It is observed that with only 5 components, PCA reconstructs the real data pretty well. 5 leading components contains about 97% of the total energy. This is to say, there is strong correlation between different stocks. In the financial market, there is clearly coherent structures. In order to get a good prediction, DMD needs 20 modes. DMD gives fixed DMD eigenvalues, so the prediction is very smooth. For both PCA and DMD, the trend of the stock price are captured pretty well.

5 Lessons learned

The project allows us to play with various languages and tools. Most files are written in python, which makes us more familiar with common packages like `numpy` and `matplotlib.pyplot`. Some of us try using `Cython` for the first time to realize functions efficiently. Also, we start to truly rely on `Github` and feel the importance of version control after confronting mistakes. We were once new to these tools, but now we are familiar with the basic usages that might be helpful in our future research work.

Besides language, we also make progress in object-oriented programming. We learned how to use classes and inheritance to make the program more ordered and easier to deal with.

Coding, from solving homework problems to dealing with research, used to be an individual task for most of us. This final project allows us to experience coding in a brand new way. This time, we have to pay much attention to interface design ahead of time to make future collaboration easy. Also, comments are now essential for us to understand each other's work. We develop good coding habits through our operation, which is an important lesson we learned from this course.

References

[1] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128, 2006.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[3] Jianqing Fan and Qiwei Yao. *The Elements of Financial Econometrics*. Science Press, 2015.

[4] Trevor Hastie, Robert John Tibshirani, and Jerome Friedman. *The elements of statistical learning : data mining, inference, and prediction*. Springer series in statistics. Springer, 2009.

[5] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4):411–430, 2000.

[6] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[7] Clarence W Rowley, Igor Mezić, Shervin Bagheri, Philipp Schlatter, and Dan S Henningson. Spectral analysis of nonlinear flows. *Journal of fluid mechanics*, 641:115–127, 2009.