Independent Component Analysis for Financial Time Series

Xi Chen (NetID: xc93) | Zhechang Yang (NetID: zy51)

April 25, 2018

Contents

Abstract

The Independent Component Analysis (ICA) is a method for separating multivariate signals into statistically independent components. In this paper, the definition of ICA and the FastICA algorithm are briefly introduced. The FastICA algorithm is firstly implemented in plain Python and then it is optimized by rewriting critical functions in C++ and using Numba package. Next, ICA is applied to simulated data and real world financial data. For the simulated data, ICA method succeeds in separating the mixed signals and the ICs obtained by ICA are the same to the true source code. As for the real data of eight stocks, our code gets the same result compared to that obtained by scikit-learn package. Also, ICA is applied to time series predictions. However, ICA doesn't bring significant improvement on MSE compared with that obtained by AR model directly. Finally, we do a comparative analysis for ICA and PCA.

Keywords: Independent component analysis; FastICA algorithm; Source signals; Financial time series

GitHub Link: `https://github.com/Xi-Ronnie-Chen/ICA_in_Finance.git`

1 Introduction

In this paper, we are mainly referring to "A. Hyvarinen and E. Oja: **Independent component analysis: algorithms and applications.** *Neural Networks* 13 (2000): 411 - 430".

Independent Component Analysis (ICA) is a method for solving a long-standing statistical problem: how to find a suitable representation of multivariate data. Such representation should reveal essential structures in the data set. ICA handles the task by finding components that are statistically independent and nongaussian. Rigorously, ICA defines a generative model, in which the observed multivariate data is assumed to be generated by a linear or non-linear combination[1] of several latent random variables. In this setting, both the latent variables and the mixture weights are unknown. Estimating these unknown variables and parameters becomes the main task in our analysis. (more details about of the ICA model formulation can be found in section 2.1)

Independence is the guiding principle in ICA estimation. To put it in one sentence, we are searching for a linear combination of the observable random variables such that transformed variables (the components) are non-Gaussian and are as independent as possible. An important intuition behind ICA estimation is that: according to the Central Limit Theorem, a sum of two independent random variables has a distribution that is closer to Gaussian than any one of the original random variables. Such intuition leads to the basic principle of ICA estimation: maximizing non-Gaussianity gives us independent components. (more details about of the ICA estimation principles can be found in section 2.2)

Independent component analysis has great potentials because its assumptions are simple and realistic. Hyvarinen, Karhunen, and Oja (2001) shows that the ICA problem is well-defined (estimable) if and only if the latent components are nongaussian. Such important conclusion tells us that non-Gaussianity is essentially the only assumption needed to make ICA applicable in practice. Besides, non-Gaussian distributions are often what we have in hand in many situations. Especially in finance, fat-tailed distributions[2] are widely seen, which makes ICA more preferable than other classical Gaussian-based methods.

Independent component analysis has a wide range of applications. One common feature of the data analyzed by ICA is that the observed data points are generated by a set of parallel signals or time series. One typical example is the cocktail-party problem, in which we try to reconstruct the original speech signals based only on the observed mixture of the original signals. Other common applications include: brain waves analysis, econometrics, feature extraction etc.

In our research, we apply independent component analysis on financial times series data and try to improve the prediction performances. This idea is well-founded because of the evidence from Malaroiu, Kiviluoto, Oja's research project report[3]. They show that the ICA transformation tends to produce components that are more structured and regular. Such result motivates us to predict stock returns (time series signals) by doing prediction with AR models in the ICA space, and then transform back to the original time series space to reconstruct the returns.

[1] We will focus on linear combination in this paper because its interpretations are more straightforward.

[2] More information about fat-tailed distribution: `https://en.wikipedia.org/wiki/Fat-tailed_distribution`.

[3] The original report can be found at: `http://research.ics.aalto.fi/ica/biennial2001-3.pdf`

2 Independent Component Analysis (ICA)

2.1 Definition of ICA

- The ICA model

$$\mathbf{x}_{n\times 1} = A_{n\times n}\mathbf{s}_{n\times 1}$$

where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \quad \mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix}$$

  - $\mathbf{x}$ contains the observable data points.
  - A is the unknown mixing weights.
  - $\mathbf{s}$ represents the underlying independent non-Gaussian components, which are supposed to capture the essential structures in the data set.

- Assumptions
  - (1) $s_1, s_2, \cdots, s_n$ are statistically independent, $E[s_i] = 0$, $\mathrm{Var}(s_i) = 1$
  - (2) independent components $(s_1, s_2, \cdots, s_n)$ have non-Gaussian distributions[4]
  - (3) the unknown mixing matrix (A) is square (can sometimes be relaxed)
  - (4) no noise terms

2.2 Estimation Principle

Key: "Non-Gaussian is Independent"

Since both the mixing weights (A) and the underlying components $\mathbf{s}$ are unknown, we need to reformulate the original linear system $\mathbf{x} = A\mathbf{s}$ as $\mathbf{s} = W\mathbf{x}$ where $W = A^{-1}$. As we can tell from this reformulated linear system, the underlying independent non-Gaussian components can be constructed by finding appropriate linear combinations of the observed data set. Our task in ICA estimation is to find n different weight vectors ($\mathbf{w}$ – rows of matrix W) such that $\hat{s} = \mathbf{w}^T\mathbf{x}$ equals one of the independent components.

To see how the Central Limit Theorem (CLT) can help us find such appropriate linear combinations, we need to keep the following important fact in mind: according to CLT, a sum of two independent random variables usually has a distribution that is closer to Gaussian than any one of the original random variables. As $\hat{\mathbf{s}} = \mathbf{w}^T\mathbf{x} = \left(\mathbf{w}^T A\right)\mathbf{s} \doteq \mathbf{z}^T\mathbf{s}$, $\hat{\mathbf{s}}$ can be seen as a linear combination of $\{s_i\}$ with weights given by $\{z_i\}$. According to our insights from CLT, in most cases, $\mathbf{z}^T\mathbf{s}$ is more Gaussian than any of the $\{s_i\}$. $\mathbf{z}^T\mathbf{s}$ becomes the least Gaussian only when it equals to one of the $\{s_i\}$. Therefore, our original problem of finding appropriate linear combinations can be transformed as an optimization problem: choosing vectors $\mathbf{w}$ such that the non-Gaussianity of $\mathbf{w}^T\mathbf{x}$ is maximized.

The above discussions lead us to the guiding principle of ICA estimation: maximizing non-Gaussianity. Concretely, the optimization problem is solved efficiently by the FastICA algorithm.

[4]Reason: any orthogonal transformation of independent Gaussian (S_1, S_2) has exactly the same distribution as (S_1, S_2).

2.3 The FastICA Algorithm

2.3.1 Preprocessing

- Centering:

$$\mathbf{x} \leftarrow \mathbf{x} - E[\mathbf{x}]$$

- Whitening:

$$\mathbf{x} \leftarrow ED^{-1/2}E^T\mathbf{x}$$

  where E and D come from the eigenvalue decomposition of $E\left[\mathbf{x}\mathbf{x}^T\right] = EDE^T$.

2.3.2 The Algorithm

In FastICA, non-Gaussianity is measured by negentropy[5], which is defined as:

$$J(\mathbf{y}) = \text{entropy}(\mathbf{y}_{\text{gauss}}) - \text{entropy}(\mathbf{y})$$

However, such direct estimation turns of to be computationally expensive, largely due to the estimation of X's pdf. In practice, some approximations have to be used to get the numerical values of $J(X)$. Hyvarinen and Oja (2000) shows that the following approximation is conceptually simple, fast to compute, and have appealing statistical properties:

$$J(y) \propto (E[G(y)] - E[G(v)])^2$$

where v is a Gaussian random variable with zero mean and unit variance. Two useful choices of G are:

$$G_1(u) = \frac{1}{a_1}\log(\cosh a_1 u) \qquad G_2(u) = -\exp\left(-u^2/2\right)$$

where $a_1 \in [1, 2]$ is a constant to be determined. The derivative of function G is denoted as g:

$$g_1(u) = \tanh(a_1 u) \qquad g_2(u) = u\exp\left(-u^2/2\right)$$

Given the objective function $J\left(\mathbf{w}^T\mathbf{x}\right)$ as above, FastICA conducts fixed-point iterations to find a maximum of the non-Gaussianity of $\mathbf{w}^T\mathbf{x}$. In plain english, the algorithm is trying to find n local maxima of the objective function.

To prevent different vectors $\{\mathbf{w}_i\}$ from converging to the same maxima, we need to decorrelate $\mathbf{w}_1^T\mathbf{x}, \cdots, \mathbf{w}_n^T\mathbf{x}$ after each update step. There are two commonly used decorrelation schemes: deflation and symmetric decorrelation. Deflation is Gram–Schmidt-like process, which means that we estimate independent components one by one and then subtract the newly estimated weight vector $\mathbf{w}_i$ from its projections on all previously estimated weights $\mathbf{w}_1, \cdots, \mathbf{w}_{i-1}$. Symmetric decorrelation treats all weight vectors as the same. It achieves the goal of decorrelation by symmetric orthogonalization of the matrix $W = (\mathbf{w}_1, \cdots, \mathbf{w}_n)^T$: $W = \left(WW^T\right)^{-1/2} W$. Both of these two methods are summarized in Algorithm 1 and Algorithm 2 correspondingly.

[5]Negentropy provides a reasonable estimation of non-Gaussianity because of the following fundamental result of information theory: a Gaussian variable has the largest entropy among all random variables of equal variance.

Algorithm 1 FastICA (deflation scheme)

$\mathbf{w}_1, \cdots, \mathbf{w}_n \leftarrow$ unit norm random vectors ▷ initialization

for $i = 1 \cdots n$ **do** ▷ n independent components

while $\left| \left\langle \mathbf{w}_i^{(\text{old})}, \mathbf{w}_i^{(\text{new})} \right\rangle - 1 \right| > \epsilon$ **do** ▷ convergence criterion

$\mathbf{w}_i^{(\text{new})} \leftarrow E\left[\mathbf{x} g\left(\left\langle \mathbf{w}_i^{(\text{old})}, \mathbf{x} \right\rangle\right)\right] - E\left[g'\left(\left\langle \mathbf{w}_i^{(\text{old})}, \mathbf{x} \right\rangle\right)\right] \mathbf{w}_i^{(\text{old})}$ ▷ udpate step

$\mathbf{w}_i^{(\text{new})} \leftarrow \textsc{Decorrelation}(\mathbf{w}_i^{(\text{new})}, \mathbf{w}_1 \cdots \mathbf{w}_{i-1})$ ▷ Gram-Schmidt-like decorrelation

$\mathbf{w}_i^{(\text{new})} \leftarrow \mathbf{w}_i^{(\text{new})} / \left\| \mathbf{w}_i^{(\text{new})} \right\|$ ▷ normalization

end while

end for

function $\textsc{Decorrelation}(\mathbf{w}_i, \mathbf{w}_1 \cdots \mathbf{w}_{i-1})$

$\mathbf{w}_i \leftarrow \mathbf{w}_i - \sum_{j=1}^{i-1} \langle \mathbf{w}_i, \mathbf{w}_j \rangle \mathbf{w}_j$ ▷ subtract $\mathbf{w}_i$ from its projections

$\mathbf{w}_i \leftarrow \mathbf{w}_i / \|\mathbf{w}_i\|$ ▷ normalization

return $\mathbf{w}_i$

end function

Algorithm 2 FastICA (symmetric decorrelation)

$\mathbf{w}_1, \cdots, \mathbf{w}_n \leftarrow$ unit norm random vectors ▷ initialization

$W \leftarrow (\mathbf{w}_1, \cdots, \mathbf{w}_n)^T$ ▷ initialization

$W \leftarrow \left(W W^T\right)^{-1/2} W$ ▷ symmetric orthogonalization

while $\max \left| \left\langle \mathbf{w}_i^{(\text{old})}, \mathbf{w}_i^{(\text{new})} \right\rangle - 1 \right| > \epsilon$ **do** ▷ convergence criterion

$\forall i : \mathbf{w}_i^{(\text{new})} \leftarrow E\left[\mathbf{x} g\left(\left\langle \mathbf{w}_i^{(\text{old})}, \mathbf{x} \right\rangle\right)\right] - E\left[g'\left(\left\langle \mathbf{w}_i^{(\text{old})}, \mathbf{x} \right\rangle\right)\right] \mathbf{w}_i^{(\text{old})}$ ▷ done in parallel

$W \leftarrow \left(\mathbf{w}_1^{(\text{new})}, \cdots, \mathbf{w}_n^{(\text{new})}\right)^T$

$W \leftarrow \left(W W^T\right)^{-1/2} W$ ▷ symmetric orthogonalization

end while

```
Thu Apr 26 00:45:13 2018    ica.prof

         917 function calls in 0.016 seconds

   Random listing order was used

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       13    0.001    0.000    0.007    0.001 <ipython-input-2-cdf4e7e6c6b1>:25(decorrelation)
       13    0.002    0.000    0.004    0.000 <ipython-input-2-cdf4e7e6c6b1>:10(_logcosh)
        1    0.001    0.001    0.016    0.016 <ipython-input-2-cdf4e7e6c6b1>:30(fastICA)
       13    0.001    0.000    0.006    0.000 /opt/conda/lib/python3.6/site-packages/scipy/linalg/decomp.py:240(eigh)
        1    0.000    0.000    0.001    0.001 /opt/conda/lib/python3.6/site-packages/scipy/linalg/decomp_svd.py:16(svd)
       14    0.001    0.000    0.001    0.000 /opt/conda/lib/python3.6/site-packages/scipy/linalg/misc.py:169(_datacopied)
       14    0.000    0.000    0.001    0.000 /opt/conda/lib/python3.6/site-packages/scipy/linalg/blas.py:218(find_best_blas_type)
       14    0.001    0.000    0.002    0.000 /opt/conda/lib/python3.6/site-packages/scipy/linalg/blas.py:279(_get_funcs)
       14    0.000    0.000    0.002    0.000 /opt/conda/lib/python3.6/site-packages/scipy/linalg/lapack.py:496(get_lapack_funcs)
       14    0.000    0.000    0.001    0.000 /opt/conda/lib/python3.6/site-packages/scipy/_lib/_util.py:192(_asarray_validated)
        1    0.001    0.001    0.001    0.001 {method 'normal' of 'mtrand.RandomState' objects}
       29    0.001    0.000    0.001    0.000 {method 'reduce' of 'numpy.ufunc' objects}
        1    0.000    0.000    0.016    0.016 <string>:1(<module>)
       13    0.000    0.000    0.001    0.000 /opt/conda/lib/python3.6/site-packages/numpy/lib/twodim_base.py:197(diag)
       14    0.000    0.000    0.001    0.000 /opt/conda/lib/python3.6/site-packages/numpy/lib/function_base.py:1170(asarray_chkfinite)
       14    0.000    0.000    0.001    0.000 /opt/conda/lib/python3.6/site-packages/numpy/core/numerictypes.py:962(find_common_type)
       14    0.001    0.000    0.001    0.000 /opt/conda/lib/python3.6/site-packages/numpy/core/_methods.py:53(_mean)
       14    0.000    0.000    0.002    0.000 {method 'mean' of 'numpy.ndarray' objects}
       81    0.001    0.000    0.001    0.000 {built-in method numpy.core.multiarray.dot}
        1    0.000    0.000    0.002    0.002 <ipython-input-2-cdf4e7e6c6b1>:2(whiten)
        1    0.000    0.000    0.016    0.016 {built-in method builtins.exec}
       69    0.001    0.000    0.001    0.000 {built-in method builtins.len}
```

Figure 1: Profile

Naive implementation

```
%%timeit
V,W,S = fastICA(matrix_return,n_components=4)
```

```
3.32 ms ± 185 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Optimization with `jit`

```
%%timeit
V,W,S = fastICA2(matrix_return,n_components=4)
```

```
3.89 ms ± 765 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Optimization with C++

```
%%timeit
V,W,S = ica.fastICA(matrix_return, "logcosh", 4, 200, 1e-04)
```

```
1.16 ms ± 50.2 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Figure 2: Time Efficiency Comparisons

3 Optimization for Performance

As we can tell from Figure 1, the function `fastICA()` consumes most of the time efficiency[6]. Therefore, it is worthwhile to optimize this core function. In our numerical experiments, we tried two different optimization approaches: (1) `jit` and (2) C++. Using the brute force `jit` "magic", the result is by no means satisfactory. `fastICA()`'s time cost remains at the level of 3 ms (no speed up at all). Our explanation for this "horrible" result is that: most of the numerical calculations in our pure python implementation are conducted through the `numpy` package. No enough space is left for `jit` to show its power. Trying to make our code run faster, we then reimplement the whole function in C++. This time, our efforts pay off. We successfully speed up the `fastICA()` function by three folds (from 3 ms to 1 ms). Details about the time efficiency comparisons between different implementations can be found in Figure 2.

[6]To make the results prettier, I omit the function calls with cumtime less than 0.001.

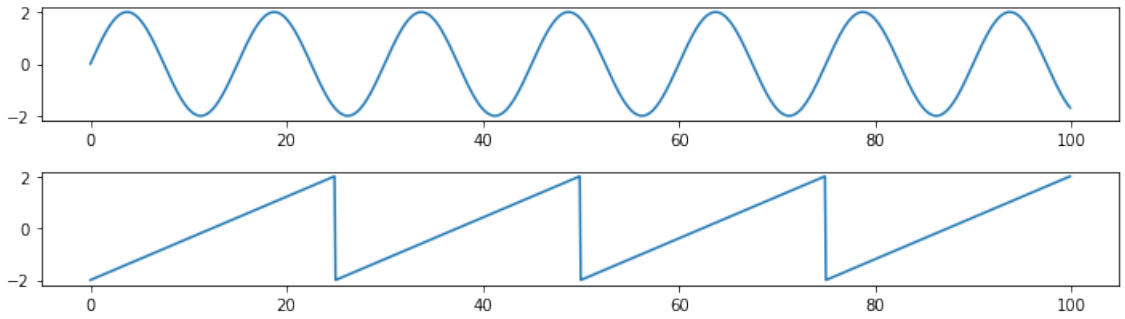

Figure 3: Source Signals: sine and triangle wave

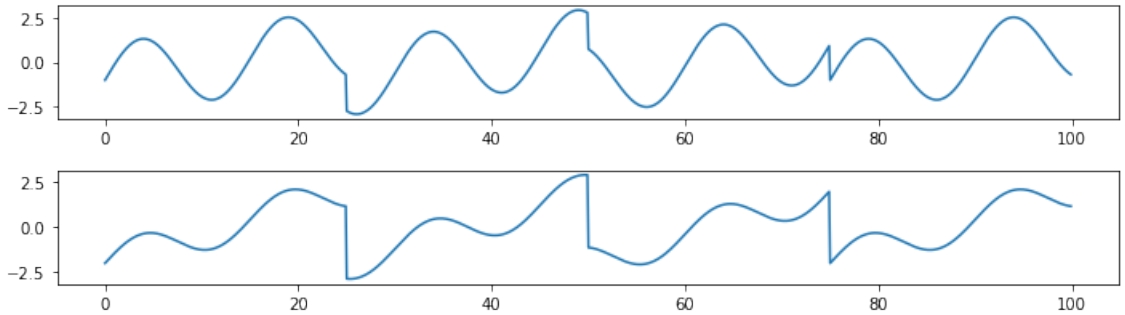

Figure 4: Mixtures of Source Signals

4 Application: Simulated Data

To investigate application of ICA, we generate two source signals. As seen in the Figure 3, the first source signal is the sine wave and the second one is the triangle wave.

Then we try to mix these two signals to get the mixtures of signals. Assume that the distance between two signals is 0.5. Figure 4 shows the observed mixtures of the source signals.

Next, we apply ICA to the mixed signals. The code of FastICA used here is implemented by ourselves. Figure 5 provides the result of recovered source code. As can be seen, the recovered signals are really close to the true source signals, although there are some slightly differences.

Also, to make sure that our code is correct, we apply ICA again by using scikit-learn package. As seen in the Figure 6, the result is the same as ours.

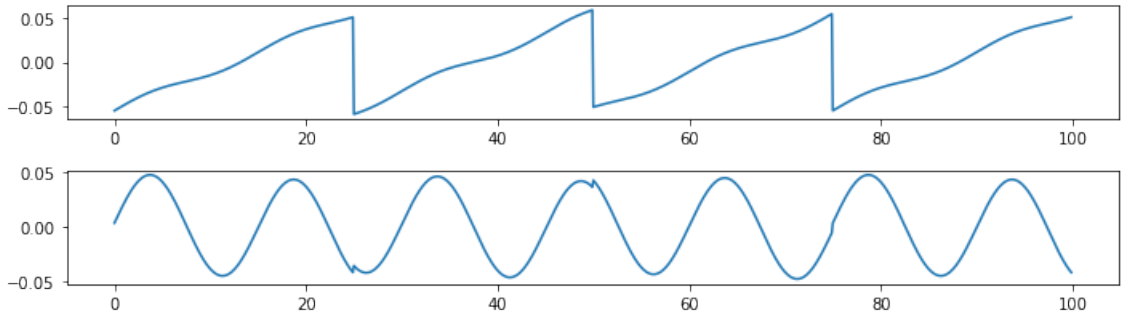

Figure 5: ICA on Mixed Signals

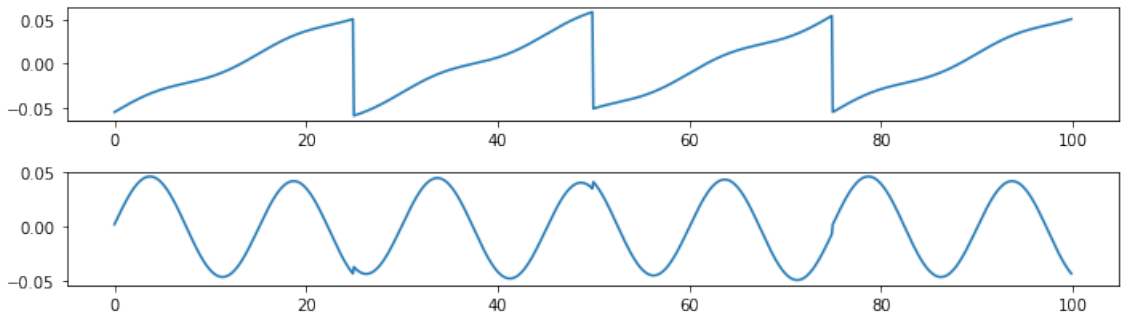

Figure 6: ICA from scikit-learn

5 Application: ICA for Financial Time Series

5.1 Reasons to Explore ICA in Finance

ICA provides a method to separate multivariate signals into statistically independent components. In the stock market, the stock returns can be regarded as signals influenced by many factors. Such factors could include news, financial condition of the company, government policies and so on. However, these factors tend to be correlated. For example, newspaper would report new policies when the policies are released. It causes the collinearity problem when we do regressions. Thus, independent components derived by ICA could be useful to alleviate this problem. Also, we hope that ICA could bring new ways of forecasting financial time series.

5.2 Data Description

We apply ICA to data from the New York Stock Exchange to test the effectiveness of ICA. In specific, we use daily close prices from Jan. 1st, 2016 to Dec. 31st, 2017 of eight large tech companies (Apple, AMD, Cisco, NVIDIA, Adobe, eBay, Intel). They are all included in NASDAQ composite. Since ICA requires the stationarity of the data, we transform the non-stationary close prices to log returns, $\log(p_t) - \log(p_{t-1})$. Figure 7 shows the prices of six companies between 2016 to 2017. As can be seen, these stocks all have the ascending trend in 2016 and 2017. In particular, NVIDIA rose fourfold in two years. Figure 8 shows the log returns of these six companies between 2016 to 2017. As you can see, the returns vibrate in a narrow range at most time but have spikes sometimes.

5.3 Finding Hidden Factors in Financial Data

Before we apply ICA, we need to center and whiten the returns data to project the original signal vectors to a new space so that the new signal vectors are white, i.e. their components are uncorrelated, and their variances are equal to 1. In other words, assume the new vectors are $\mathbf{x}^*$:

$$E\left[(\mathbf{x}^*)(\mathbf{x}^*)^T\right] = I$$

Then we could use FastICA algorithm to get the independent components from the returns vectors. Figure 9 provides the eight independent components calculated by FastICA algorithm.

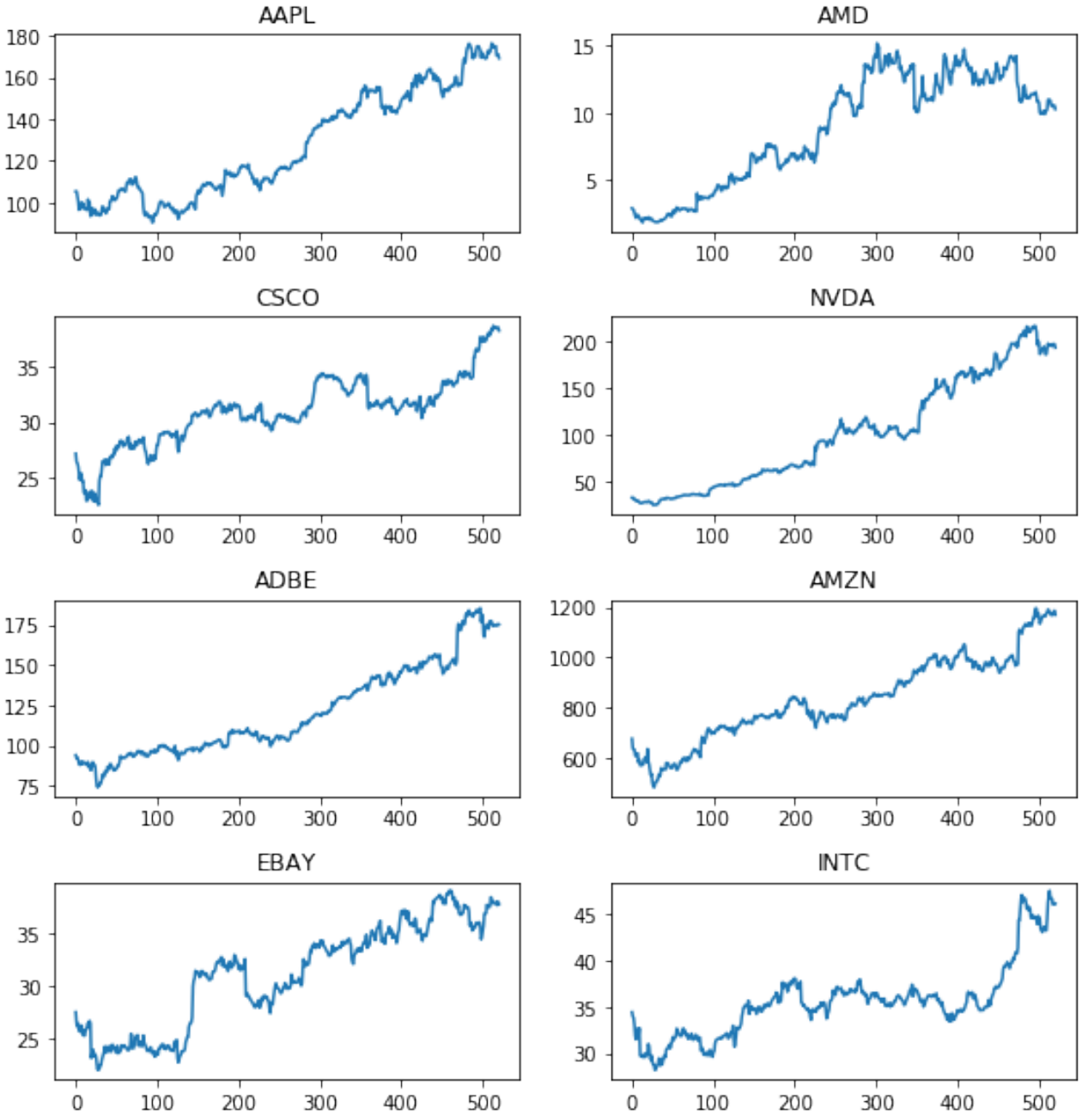

Figure 7: Daily Close Prices of Apple, AMD, Cisco, NVIDIA, Adobe, eBay, and Intel from Jan. 1st, 2016 to Dec. 31st, 2017

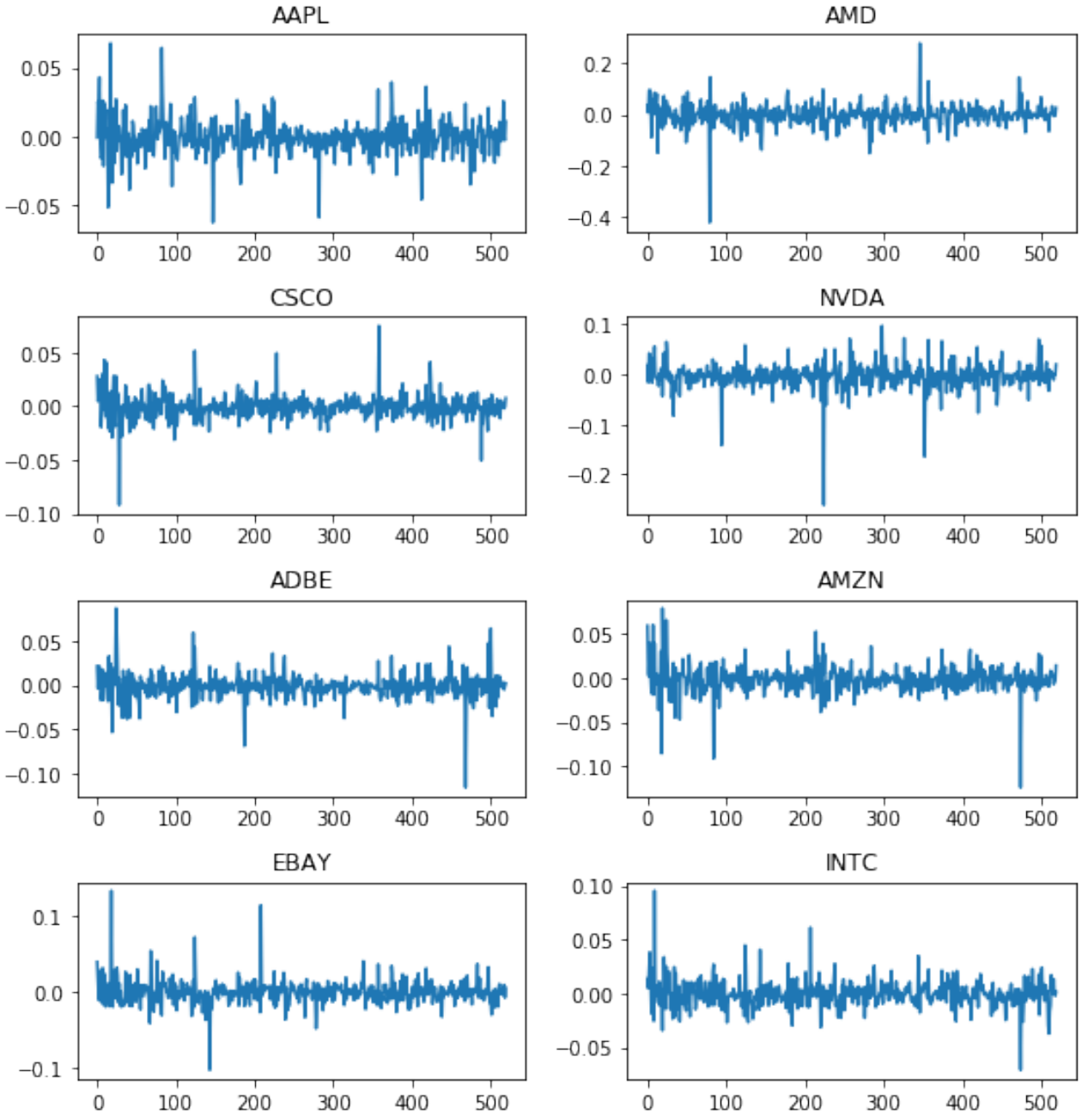

Figure 8: Log Returns of Apple, AMD, Cisco, NVIDIA, Adobe, eBay, and Intel from Jan. 1st, 2016 to Dec. 31st, 2017

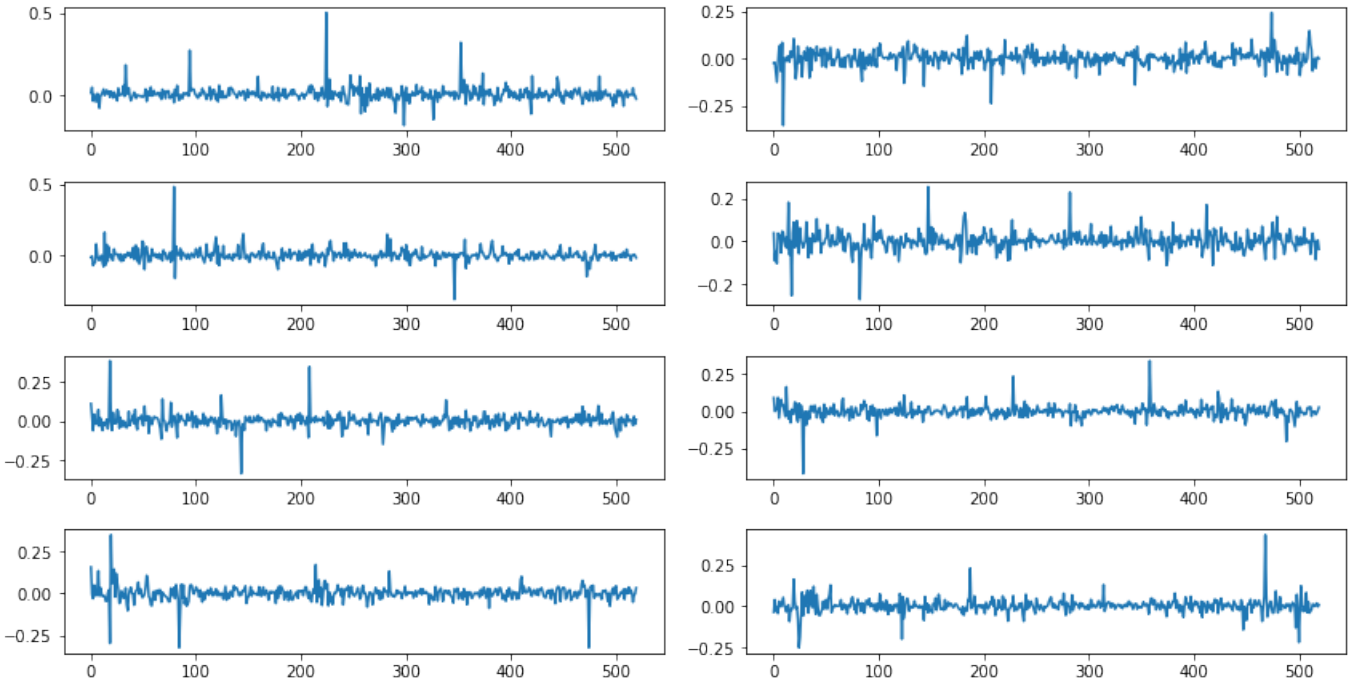

Figure 9: Independent Components from Log Returns

Since it is difficult for us to get the true source signals, we could not say whether these ICs are close to the true source signals. However, we could test if signals recovered by ICs are close to the original signals to make sure our code is correct:

$$X' = AS \qquad A = W^{-1}$$

We take the AMD stock returns for example. Figure 10 presents the original stock return (blue line) and the return recovered by ICs (black line). As you can see, they are almost the same.

Also, we could choose number of ICs less than the original signal number. For example, first four ICs could be chosen although there are eight stock returns. Thus, we also apply FastICA to the data where parameter "n_components" is equal to four. Figure 11 shows the four ICs calculated by our code.

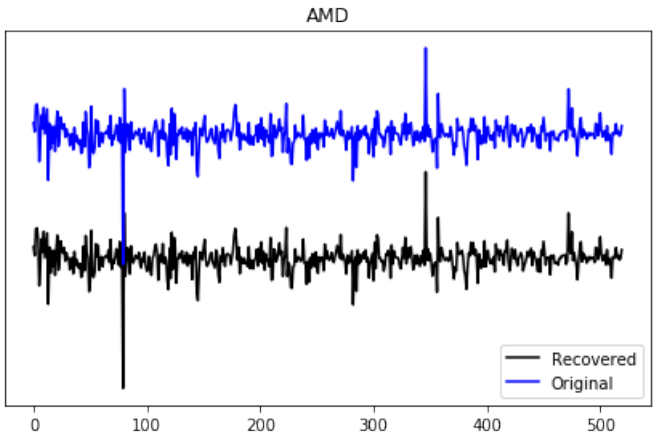

Figure 10: AMD Stock Returns

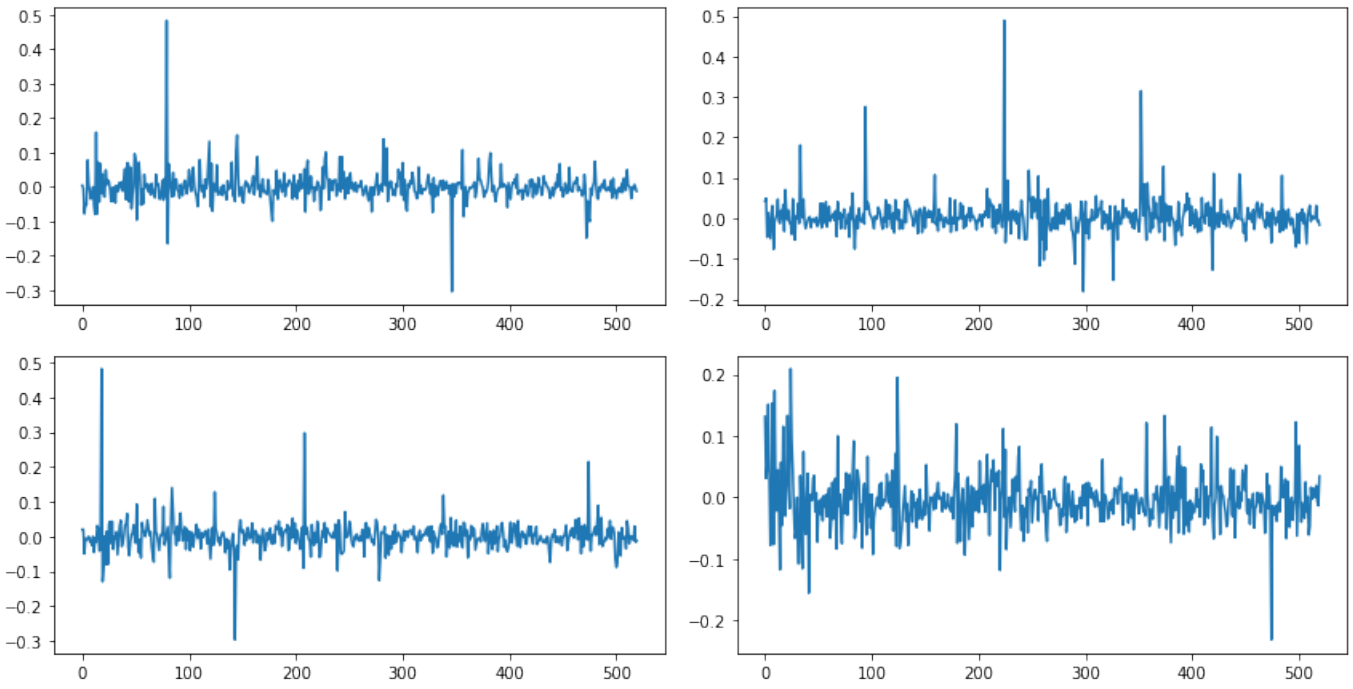

Figure 11: Four ICs

In order to check the FastICA algorithm implemented by us is correct, we also perform ICA to data by using the sklearn package. Similarly, we set "n_components" to four. Figure 12 shows the first four ICs, resulting from sklearn package. Although the order of ICs changes, the results are exactly the same.

5.4 Time Series Prediction by ICA

As we've talked in the paper, we could choose number of ICs less than the number of original signals. Thus, ICA could produce component signals that can be compressed with fewer bits than the original signals. So, ICs are more structured and regular than original ones. Instead of doing AR prediction directly to original time series data, we could first project the data to ICA space, then doing AR prediction there, finally projecting back it back to the original space. In this paper, we try to investigate whether this method outperforms the direct method. In specific, our procedure is as follows:

1. Whiten the returns data.

2. Apply FastICA algorithm implemented by us to the whitened data. Set the number of ICs as four.

3. Predict each IC separately, i.e. using AR model to each IC. The prediction equation is:

$$s^p(t+1) = g\left[s^s(t), s^s(t-1), \cdots, s^s(t-q)\right]$$

4. Combine the AR predictions of ICs together and transform them back to the original space, then we could get the predictions for the original time series:

$$x^p(t) = As^p(t)$$

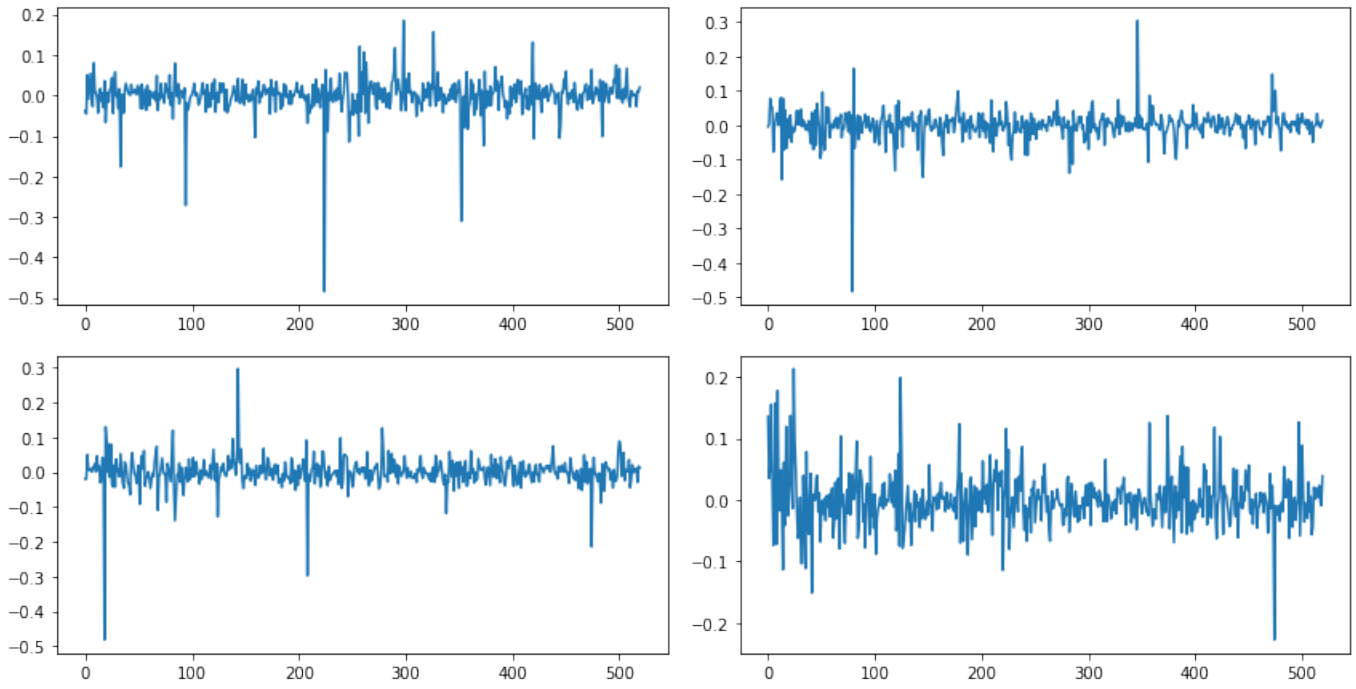

Figure 12: Four ICs from sklearn

For the autoregressive model, the order of AR would influence the result of the mean squared error. Higher order of AR could reduce the mean squared error. Thus, to investigate time series AR predictions by ICA, we should test different orders of AR. In this paper, we choose AR(1) to AR(5) and AR(20) according to trading rule of stock markets (5 trading days per week and 20 trading days per month).

Table 1 shows a comparison of errors obtained by classical AR prediction and ICA-AR method we introduce above. The errors are in units of 0.0001. As can be seen, from AR(1) to AR(5) and AR(20), applying ICA first then doing AR predictions doesn't have significant improvement on mean squared error. To be specific, for AR(1), mean squared errors of AMD, NVDA, AMZN applying ICA are smaller than those doing AR predictions directly, while mean squared errors of AAPL, ADBE, EBAY, INTC applying ICA are larger. Similarly to AR(2) to AR(20), there are no significant advantages by applying ICA, the mean squared errors of two methods are really close to each other.

Table 1: Comparison of Errors Obtained by AR and ICA-AR

Order of AR	Method	AAPL	AMD	CSCO	NVDA	ADBE	AMZN	EBAY	INTC
AR(1)	ICA	1.6473	18.5558	1.3728	6.8173	2.0532	2.4198	2.4797	1.5573
	Directly	1.6416	18.6018	1.3728	6.8202	2.0508	2.4249	2.4772	1.5554
AR(2)	ICA	1.6370	18.4342	1.3731	6.8274	2.0567	2.4204	2.4642	1.5617
	Directly	1.6311	18.4369	1.3688	6.8227	2.0478	2.4275	2.4728	1.5569
AR(3)	ICA	1.6317	18.2462	1.3671	6.7542	2.0428	2.4269	2.4532	1.5535
	Directly	1.6195	18.2893	1.3528	6.7560	2.0444	2.4321	2.4571	1.5492
AR(4)	ICA	1.5962	18.0547	1.3634	6.7099	2.0342	2.3867	2.4532	1.5283
	Directly	1.5847	18.0972	1.3444	6.6975	2.0393	2.4007	2.4470	1.5223
AR(5)	ICA	1.5981	17.9164	1.3542	6.7031	2.0322	2.3827	2.4575	1.5271
	Directly	1.5870	17.9677	1.3292	6.6928	2.0251	2.4049	2.4501	1.5223
AR(6)	ICA	1.4045	17.0800	1.2226	6.5574	1.9765	1.9739	2.0812	1.3189
	Directly	1.3818	17.0975	1.2038	6.5390	1.9233	1.9435	2.0347	1.2843

6 Comparative Analysis

6.1 ICA vs. PCA

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables. This technique is commonly used in machine learning to reduce the dimensions of high-dimensional datasets. Also, PCA is a well-established tool in finance. Applications range from Arbitrage Pricing Theory and factor models to input selection for multi-currency portfolios (Utans et al., 1997). In this section, we will compare the difference between PCA and ICA.

We could use Singular value decomposition (SVD) to obtain the principal components. Let X denote the m $\times$ n data matrix, where m is the number of observed vectors, and each vector has n components. The data matrix can be decomposed into:

$$X = USV^T$$

The principal components are given by $XV = US$, i.e. the vectors of the orthonormal columns in U, weighted by the singular values from S. In order to better compare PCA with ICA, we draw each PC and IC in the same plot. Figure 13 shows the first four PCs and ICs. In the first subplot of this figure, signals obtained by PCA and ICA are almost the same perhaps because this source signal have strong impact on the mixture of signals. As for the third and the fourth subplots, sudden changes of the two components happen in the same time but the amplitude looks different. However, the second subplot shows that two components are quite different either of frequency or amplitude.

As mentioned above in the paper, the first four ICs obtained by ICA could occur different orders. However, PCA method provides the same order of PCs if we sort the PCs by singular values, thus we could get the same order all the time, which makes PCA a better dimension reducing tool than ICA. The

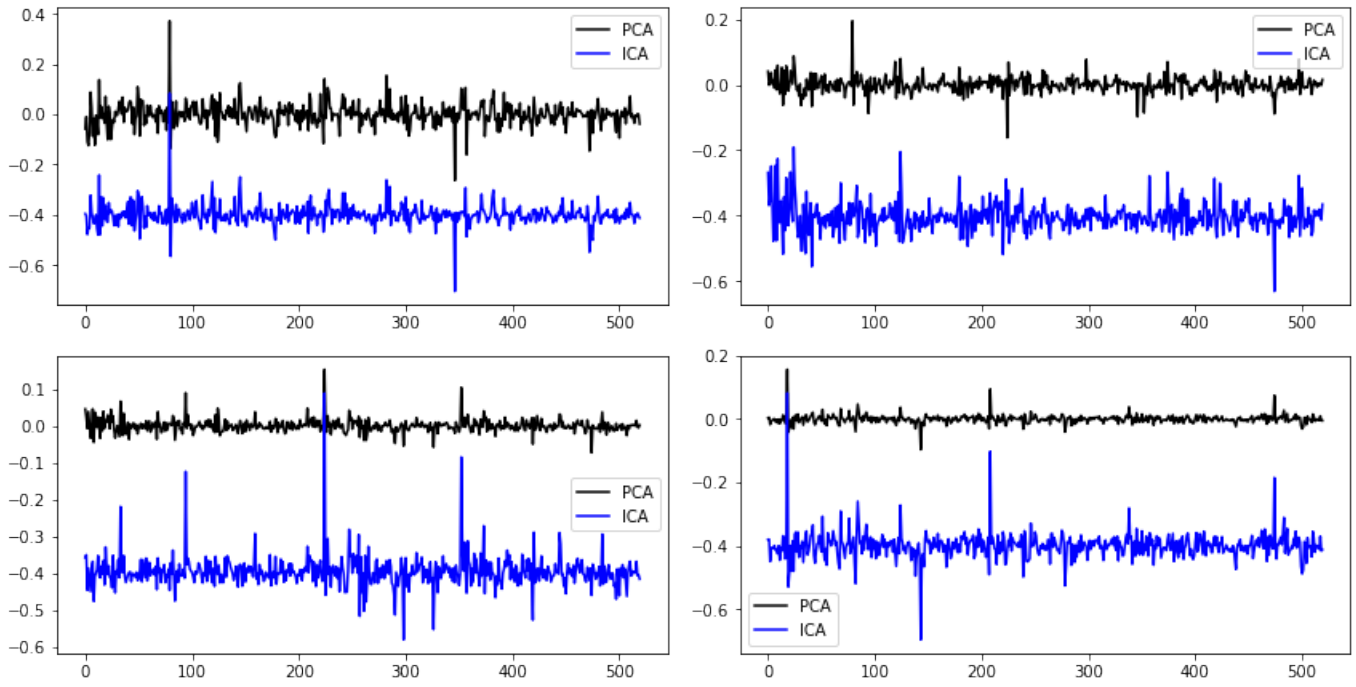

Figure 13: ICA vs. PCA

```
%%timeit
V,W,S = sk_fastica(matrix_return,n_components=4, algorithm='parallel')
```

```
4.06 ms ± 189 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
%%timeit
V,W,S = sk_fastica(matrix_return,n_components=4, algorithm='deflation')
```

```
104 ms ± 7.52 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Figure 14: "Deflation" vs. "Parallel"

advantage of ICA is that ICs possess the higher order independence while PCs only satisfy the condition of uncorrelatedness (linear independence). Therefore, ICA is a better tool to separate multivariate signals.

6.2 Deflation Version of FastICA vs. Parallel Version

When we talk about the FastICA algorithm, we mention that there are two versions. One is deflation algorithm, which obtains the optimal weight vectors one by one then construct the optimal weight matrix. And the other one is parallel algorithm which initializes the weight matrix and then optimizes it by doing iterations. Here we try to investigate which algorithm is faster. Figure 14 shows the time that is used to run two versions of FastICA. As can be seen, the parallel algorithm is much faster than the deflation one.

7 Conclusions

In this paper, Independent Component Analysis (ICA) is applied to simulated data and two years daily data of eight stocks from New York Stock Exchange. For the simulated data, ICA method succeeds in separating the mixed signals and the ICs obtained by ICA are the same to the true source code. And for the simulated data and the real data of eight stocks, our code gets the same result compared to that obtained by scikit-learn package. Then we do time series predictions by ICA. However, ICA doesn't bring significant improvement on MSE compared with that obtained by AR model directly. Finally, we do a comparative analysis for ICA and PCA. ICA is a computational method which obtains components having higher order independence. Although PCs are only linearly independent, PCA is a better dimension reduction tool for high dimension data since PCs could be sorted by singular values. Many papers have shown that ICA and PCA are both useful tool for analyzing financial. And there will be clearly more avenues in which ICA and PCA can be used to finance.

8 References

1. A. Hyvarinen, K. Karhunen, and E. Oja: **Independent Component Analysis.** John Wiley & Sons, Inc (2001).

2. A. Hyvarinen and E. Oja: **Independent component analysis: algorithms and applications.** *Neural Networks* 13 (2000): 411 - 430.

3. E. Oja, K. Kiviluoto, and S. Malaroiu: **Independent component analysis for financial time series.** *Proc. IEEE 2000 Symp. on Adapt. Systems for Signal Proc., Comm. and Control AS-SPCC*, Oct. 1 - 4, 2000, Lake Louise, Canada, pp. 111 - 116 (2000).

4. S. Malaroiu, K. Kiviluoto, and E. Oja: **ICA preprocessing for time series prediction.** *Proc. 2nd Int. Workshop on Indep. Comp. Anal. and Blind Source Separation*, June 19 - 22, 2000, Helsinki, Finland, pp. 453 - 457 (2000).

5. S. Malaroiu, K. Kiviluoto, and E. Oja: **Time series prediction with Independent Component Analysis.** *Proc. Int. Conf. on Advanced Investment Technology*, December 19 - 21, 1999, Queensland, Australia (1999).

Appendix

- **Intuition for independence**
  - Random variables X and Y are said to be independent if information on the value of X does not give any information on the value of Y, and vice versa.
- **Definition of independence**

$$X \text{ and } Y \text{ are independent} \Leftrightarrow f(x,y) = f_X(x) f_Y(y)$$

  where

$$f_X(x) = \int f(x,y)\, dy \qquad f_Y(y) = \int f(x,y)\, dx$$

- **Important properties of independence**
  - If X and Y are independent, given two functions h_1 and h_2, we always have:

$$E[h_1(X) h_2(Y)] = E_X[h_1(X)] E_Y[h_2(Y)]$$

  - If X and Y are independent, then they are uncorrelated, which means that:

$$\operatorname{cov}(X,Y) = E[XY] - E[X] E[Y] = 0$$

    However, uncorrelatedness does not imply independence.
  - If X and Y are independent, then we have:

$$\operatorname{Kurt}(X+Y) = \operatorname{Kurt}(X) + \operatorname{Kurt}(Y)$$

- **Intuition for entropy**
  - The entropy of a random variable can be interpreted as the degree of information that the observation of the variable gives. The more "random", i.e. unpredictable and unstructured the variable is, the larger its entropy.
- **Definition of entropy**

$$H(X) = -\int f_X(x) \log f_X(x)\, dx$$