

# 情報特別演習最終レポート

筑波大学情報学群情報科学類 (201611350) 江畑 拓哉

February 9, 2018

## Contents

<b>1</b>	<b>概要</b>	<b>3</b>
<b>2</b>	<b>序論</b>	<b>3</b>
<b>3</b>	<b>時系列データベースの比較と OpenTSDB の利用法</b>	<b>4</b>
3.1	InfluxDB . . . . .	4
3.2	Graphite . . . . .	5
3.3	Datomic . . . . .	6
3.4	OpenTSDB . . . . .	9
3.4.1	HBase とその周辺知識 . . . . .	9
3.4.2	HBase . . . . .	9
3.4.3	Hadoop . . . . .	10
3.4.4	Zookeeper . . . . .	10
3.4.5	OpenTSDB . . . . .	12
3.4.6	OpenTSDB の HTTP API . . . . .	13
3.4.7	Grafana . . . . .	15
<b>4</b>	<b>Clojure を用いた JVM における高速計算技法</b>	<b>16</b>
4.1	Clojure 自身の高速化手法 . . . . .	16
4.2	ClojureCL . . . . .	17
4.3	Neanderthal . . . . .	17
4.4	Clojure.core.matrix . . . . .	20
4.4.1	vectorz-clj . . . . .	20
4.4.2	clatrix . . . . .	20
4.5	ACM3 (Apache Common Math 3) . . . . .	21
<b>5</b>	<b>ARIMA モデルによる時系列分析</b>	<b>21</b>
5.1	BackShift 記法 . . . . .	21
5.2	単位根検定 . . . . .	22
5.2.1	定常性の性質 . . . . .	22
5.2.2	ADF 検定 . . . . .	25

5.2.3	KPSS 検定	26
5.3	AR モデル	27
5.4	MA モデル	27
5.5	ACF と PACF	28
5.5.1	ACF	28
5.5.2	PACF	28
5.5.3	p, q の決定法	29
5.6	パラメータ推定	32
5.6.1	対数尤度関数	32
5.6.2	アメーバ法	32
5.6.3	AIC	34
5.7	SARIMA モデルについて	34
<b>6</b>	<b>Clojure/ClojureScript を用いた Web 開発</b>	<b>35</b>
6.1	Clojure によるバックエンド開発	35
6.1.1	Luminus Framework	35
6.1.2	Swagger UI	36
6.2	ClojureScript によるフロントエンド開発	38
6.2.1	基本的な開発	38
6.2.2	Reagent	38
<b>7</b>	<b>MLLK の開発</b>	<b>40</b>
7.1	ARIMA モデルライブラリ	40
7.2	データベース設計	40
7.2.1	Postgresql	41
7.2.2	OpenTSDB	42
7.3	バックエンド開発	42
7.4	フロントエンド開発	42
<b>8</b>	<b>ARIMA 推定 と Random Forest による予測</b>	<b>45</b>
8.1	概要	46
8.2	実験方法	46
8.3	実験結果	47
8.4	考察	47
<b>9</b>	<b>まとめと今後の課題</b>	<b>47</b>
<b>10</b>	<b>謝辞</b>	<b>48</b>
<b>11</b>	<b>付録</b>	<b>48</b>
11.1	このレポートにおける数式について	48
11.1.1	独立同時分布と変数の補足	48
11.1.2	標準誤差	49
11.2	出力結果	51
11.3	実験結果	54

## 1 概要

今情報特別演習において私は、機械学習を初学者が学ぶための Web アプリ開発を行った。当初は大規模データベースを用いた機械学習 API を作るという目標であったが、機械学習を学んでいくにあたり、これらの中身を理解するための初学者向けの解説の供給が少ないと感じたため、開発目標をやや変更した。

今回学習した内容は、① 大規模データベースの、特に時系列データベースの比較とその利用法 ② 機械学習等を実装する際に重要となる高速計算を JVM 言語内で行う手法 ③ 代表的な時系列分析の一つである (S)ARIMA モデル ④ 開発言語として取り上げた Clojure/ClojureScript の、言語自体・これを用いた Web 開発手法、の 4 分野である。

結果として 時系列データベースとして OpenTSDB を採用し ARIMA モデルとそれに付随する ADF 検定などを実装・解説を作成した。更に JVM 上で高速計算を行うために、*jblas* や *Intel<sup>®</sup> MATH KERNEL LIBRARY* を用いた GPU 演算、OpenCL の利用例を調べ比較し、一部を Web アプリの実装に活用した。そして *Clojure* という言語を身に着け、Clojure/ClojureScript を用いて JavaScript のライブラリである *React.js* などを利用する手法についてまとめ、これらを利用して Web アプリの概形を実装した。

## 2 序論

この情報特別演習の初期テーマの決定はグループメンバーからの提案が元であった。その概要は、大規模データベースである *Apache HBase<sup>™</sup>* (以降 “HBase” と呼称する) を用いて入力されたデータの因果関係を分析、予測する機械学習 API を作成するというものである。ここから因果関係と相関関係の違いについて学習し、機械学習手法について吟味した結果、① ニューラルネットを用いた学習手法と、② (S)ARIMA モデルを用いた時系列データ予測と Random Forest を用いた欠損値補完を組み合わせたものの、2 つの手法が議題に上がったが、後者を選択することになりこれを研究することになった。その中で機械学習を学ぶ際にその内部を知る必要があり、学んでいく際にその資料の供給が少ないことを感じた。そしてその資料も兼ね備えたいと考え、また機械学習結果を視覚的にわかりやすく伝える目的も合わせて、Web アプリという形で開発を行うことに目標を定めた。

機械学習という名前は世間に流布しており、それを用いた API として例えば Microsoft Azure の *ComputerVisionAPI* 等を上げることができるが、この中身が解説されることはその必要性や機密性の問題から非常に少ない。また Python や R といった言語やそのライブラリ等に付属している統計処理、機械学習の関数などもその殆どが簡便化されており、例えば *auto.arima()* 関数などはその中身に踏み込むことなく実行、モデルの比較を行うことができる。これは機械学習の利用者という立場からすれば素晴らしい進歩であると考えられるが、その反面機械学習を学ぶ立場になった場合、その中身を知らない状態で API や関数を利用することができるため、手放しに機械学習を理解できたと誤解してしまう可能性がある。

このため主要な機械学習についてその中身を学習できるツールの作成は、統計や機械学習を学びたいと志している、或いは先述のようなツールの中身について興味を持った者に

対して、一定の需要があるのではないかと考えている。

### 3 時系列データベースの比較と OpenTSDB の利用法

一般的に時系列データのような、単調増加する要素を持つデータを通常の大規模データベースに保存することはデータの分散という点から問題が発生する。[14] このため時系列データを扱うためのデータベースを考える必要がある。幸いなことに、初期案にあった HBase に対して時系列データを扱うことができるように拡張した *OpenTSDB* というデータベースがある。OpenTSDB は HTTP API としての操作が許されており、また保存されているデータを確認することが容易であるように設計されている。今回は HBase を完全分散モードで利用する、というチームメンバーの目標に沿ってこちらのデータベースを採用した。

他の時系列データベースの提案として考えられるものに、① *InfluxDB*、② *Graphite*、③ *Datomic*などを挙げることができ、これらを比較した。

この章ではそれぞれのデータベースの特徴と利用法を簡潔にまとめる。

#### 3.1 InfluxDB

InfluxDB は InfluxData が開発を行っているデータベースであり、高機能なクラウドシステムを有料で使うことができるほか、オープンソースソフトウェアとしての利用も可能である。このデータベースの利点の一つに、利用方法が簡単であることが挙げられる。シングルノードでの利用に関してのみに焦点を絞れば、2017 年 12 月において *Arch Linux*でのインストールは、パッケージのインストールとサービスの起動の 2つのコマンドのみである。またデータベースの読み書きに関しては、SQL に近い記法を用いた HTTP API を用いて行うことができ、今回の演習における開発言語である Clojure で必要な部分に関するラッパーを書くことは非常に簡単であった。また TICK stack と呼ばれる InfluxDB を含む時系列データを扱うための環境を追加でセットアップすればデータのモニタリング、収集、リアルタイム処理をより効率的に行うことができる。本演習の初期目標ではデータの入力をユーザが行うことができる設定になっていたため、悪意あるデータを監視することが容易であるという点、データベースの外側の分野までの広いサポート環境があるという点からこのデータベースは非常に魅力的である。

このデータベースが扱うデータモデルの概要を以下に示す。

Table 1: InfluxDB data model

name(required)		
timestamp (required)	fields (required)	tags (optional)
⋮	⋮	⋮

それぞれの用語についてその意味と例を挙げると以下ようになる。

- name データの名前 (ex. 日経平均株価)  
データの名前であり、何に関してのデータであるかを表す。
- timestamp 時刻データ (ex. 2018-01-27T00:00:00Z)  
時刻データであり、いつのデータであるのかを示す。この場合の“いつのデータ”とは、データの登録日時ではなく、そのデータの発生日時である。
- fields 測定値群 (ex. (終値: 12000) (始値: 11000))  
そのデータが持つ値を示す。いくつかの属性に従って複数の値を格納することができるが、ここに登録されるデータは索引付けされるべきものではないという点でタグ群と意味が異なる。
- tags タグ群 (ex. (記録者: A) (ソース: 東京株式市場))  
そのデータの持つ属性や追加情報を示す。ここに登録されるデータは索引付けされており、データの絞り込みを行う目的に用いられる。

## 3.2 Graphite

Graphite は Python を中心にして書かれた時系列データベースであり、同じく Python の Web フレームワークである Django と組み合わせることが、Graphite 自身の Web UI コンポーネントが Django であるという点もあり、非常に容易である。同時に Python は機械学習に関する API・ライブラリ が豊富に存在しているため、本演習が純粋に“Web API の作成”のみの目標であったならばこちらを用いて開発を行っていただろう。またデータベースの導入自体も、Python のパッケージ管理システムである pip を用いて行うことができることから、純粋に Python のみによってすべてを解決することができる。更に Graphite のドキュメントは豊富に存在しており、例えば Monitoring with Graphite [3] を挙げることができる。

Graphite の内部について簡単に説明を行うと、主に 4 つのコンポーネント、Carbon、Whisper、Cario、Django を中心に展開する。

- Carbon は、後述するデータベースそのものと言える Whisper にデータを登録する役割を担っており、メトリクス<sup>1</sup>のバッファリングを行ったり他のデータベースにメトリクスをリレーさせたりすることができる。
- Whisper は、入手したデータをファイルシステムに書き込み・読み出しを行う役割を担っており、この部分は Ceres と呼ばれるコンポーネントに置き換えることがで

---

<sup>1</sup>metrics: 入手したデータを分析して数値化したもの

きる。両者の違いは、Whisper が保存領域を固定サイズとして確保するのに対して、Ceres は任意のサイズで保存領域を確保できるということにある。

- Cario は、Graphite のグラフィックエンジンを担当しており、保存されているデータを視覚化する上で非常に重要な役割を果たしている。
- Django は、Cario によって出力されたデータを表示する役割を担っており、データを扱う開発者はこの部分を見てデータを確認することになる。

このデータベースが扱うデータモデルは階層構造を取っており、一例を紹介すると以下のようなになる。

“stock\_price.nikkei\_index.close\_price 12000 1517055464”

上の文字列を送信することによって、stock\_price の中の nikkei\_index の中にある close\_price という階層に 12000 という値を Unix 時間である 1517055464 のデータとして登録している。つまりこのデータは以下のような形に保存されたと考える。

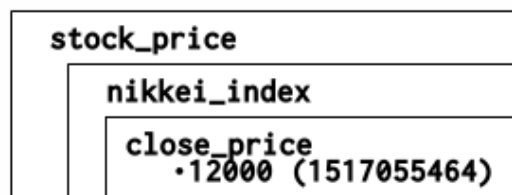


Figure 1: Graphite example

### 3.3 Datomic

Datomic は他のデータベースとはかけ離れた設計が行われた新しい世代の分散型データベースである。Clojure の作者である Rich Hickey 氏らが作成し、有料でメンテナンスとアップデートが付属されたクラウドシステムを使うことができる。また一年に限っては無料でこの機能を利用することもできる。これとは別に存在する無料版に関しては分散できるピア数などの制限がかかる。

Datomic には2つの目標「情報を時間によって紐付け蓄積する」「データベースアプリケーションのモデルをリモートアクセスするものからそれぞれのプログラムの中にあるものとする」<sup>1</sup> がある。この考え方によって得られた大きな2つの特徴に、① Append-Only

<sup>1</sup><http://endot.org/notes/2014-01-10-using-datomic-with-riak/>

②データベースに独立したクエリーエンジンがある。

Append-Only とはその名の通り、追加のみという意味で、言い換えれば変更ができないということを意味する。これは情報を時間に紐付けることによって最新の情報を見ることができるため、情報を“書き換える”必要がなくなったためにできたことであり、トランザクション処理などのデータの管理を容易にすることができる。

データベースに独立したクエリーエンジンとは、アプリケーション側でトランザクションやクエリ処理を実行するという意味を示しており、データベースに HTTP API などを用いてクエリを投げデータベース側がそのクエリを処理して結果を送信していたものをアプリケーション側に移す、ということになる。その意味で Datomic はアプリケーション側をピア<sup>2</sup>と呼称する。

ピアが扱うデータはデータベースではなくピア側のキャッシュに Read Only な形で LRU<sup>3</sup>形式で保持される。データベースは書き込まれたデータを保存し、更新があればそれぞれのピアが持っている、データベースに対して常に開いているノードに告知し、アプリケーション側から要求されるデータ群をそのまま返すことになる。これによってピア側のメモリキャッシュを疑似データベースとして貪欲に使うことができ、データベースのボトルネックを解消することができるようになっている。更にピア側のキャッシュ上のデータベースは実質ゼロコストで用いることができるため、LRU が最適であるような目的のアプリケーションにこのデータベースを適用した場合、データへのアクセスという点において他のデータベースに性能で劣ることはない。またクエリ処理を分散しているため、多くのクエリ処理をこなさなければならないピアが増えたとしても、キャッシュ上のデータを使っている限りはその処理によってデータベースに負荷がかかることもない。データベースの更新をピアに告知しなければならないという点でデータベースへの書き込みがネックになる可能性もあるが、論理的に分かれているデータごとにデータベースそのものを分割することで解決することができる。

データベースのアクセス方法は Datalog と呼ばれる Clojure らしいシステムによって扱われるため、SQL に慣れている場合には苦勞する可能性があるが、アプリケーションに柔軟に組み込むことができる。これはデータがキャッシュ上に Read-Only な形で存在しているという特性と、Clojure が関数型言語の側面を持っているという点を考えれば、データベース上のデータを手元にあるデータであるかのように利用することができるということを意味している。また保存しているデータは必ず Datom という最小単位に分割されており、これを元にして様々な形にデータを変形させることができる。

このデータベースが扱うデータ例を以下に示す。

```
{:nikkei-index/type "close-price"  
  :nikkei-index/value 12000  
  :nikkei-index/timestamp 1517055464}
```

データは nikkei-index/type に対する値として “close-price” が格納されている。nikkei-index に “close-price” が含まれているわけではない。

---

<sup>2</sup>peer

<sup>3</sup>Least Recently Used

h

Table 2: Datomic の特徴

目指すもの	<ul style="list-style-type: none"><li>・ 情報は時間によって紐付ける</li><li>・ データベースアプリケーションのモデルをそれぞれのプログラム内に移動する</li></ul>
大きな特徴	<ul style="list-style-type: none"><li>・ Append-Only データベース</li><li>・ データベース側ではなくアプリケーション側にクエリ処理エンジンがある</li></ul>

# Datomic Architecture

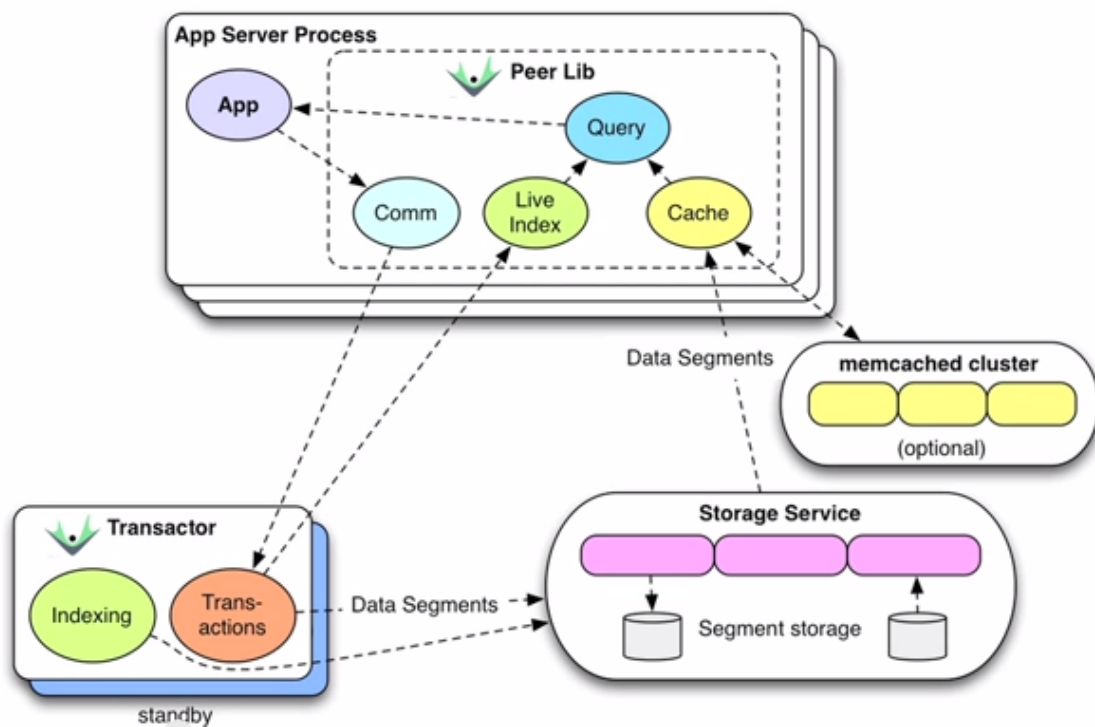


Figure 2: Talk Notes: Using Datomic With Riak より



## 3.4 OpenTSDB

OpenTSDB の特徴の説明、セットアップや利用方法に関して説明を行う前に、その基盤である HBase とその周辺知識について簡単にまとめ、その後 OpenTSDB についての説明を行う。また OpenTSDB を用いる際に強力な可視化ツールとなる Grafana についてもデータベースの中身にデータが入っていることを示すために触れておく。

### 3.4.1 HBase とその周辺知識

HBase とは *Apache<sup>TM</sup> Hadoop<sup>®</sup>* (以降 “Hadoop” と呼称する) と呼ばれる、大規模データの分散処理フレームワークのためのデータベースである。そして Hadoop の分散サービスを形成するために *Apache Zookeeper<sup>TM</sup>* (以降 “Zookeeper” と呼称する) という管理ツールが使われる。

### 3.4.2 HBase

HBase は NoSQL の一つである。NoSQL は大別して、①キーバリュー型②ワイドカラム型③ドキュメント型④グラフ型、があり HBase はワイドカラム型<sup>4</sup>に属している。

Table 3: ワイドカラム型の例 (Name 列を取り出すこと等を得意とする)

ID	Name	Email	Birthday	Authorization
001	Bob	bob @ foo.com	1998/01/02	true
002	John	john @ bar.com	1987/02/01	false
⋮	⋮	⋮	⋮	⋮

Hadoop の HDFS (Hadoop Distributed File System) の補完を担っており、複数台のマシンのディスクを一台のディスクであるかのように扱うことができる。全体のデータは Region という単位で分割されており、これをそれぞれのディスクに 1 つ以上割り振っていくことで分散を行う。

続いて HBase の論理データモデルについて説明を行う。最上位概念は Namespace と呼ばれるもので、この中には Table と呼ばれるデータを表形式で保持している概念を 1 個以上含んでいる。Table は一つ以上の RowKey、一つ以上の ColumnFamily で構成されている。そして ColumnFamily には一つ以上の ColumnQualifier が存在している。行キーである ColumnQualifier と列キーである RowKey の交差点にはそれぞれ Cell と呼ばれる領域があり、ここにデータが格納されることになる。データは Timestamp とともに保存されており、Cell にはそのデータが重ねて保存される。つまり Cell には Timestamp に紐付けられたデータが複数存在することになる。また、ワイドカラム型であるという特性上、Table は Rowkey でソートされた状態で保存されることになる。

HBase の物理モデルの Table の構造はキーバリュー形式で保存されている。物理モデ

<sup>4</sup>簡単に説明するとデータを行ごとではなく列に対して管理しており特定の列を取り出して処理することに最適化されており、高いパフォーマンスやスケーラビリティを持っている。

ルの詳細はデータの分散などの説明も必要となるが、これ以上の内容は本演習で理解することができなかったため説明を省略する。

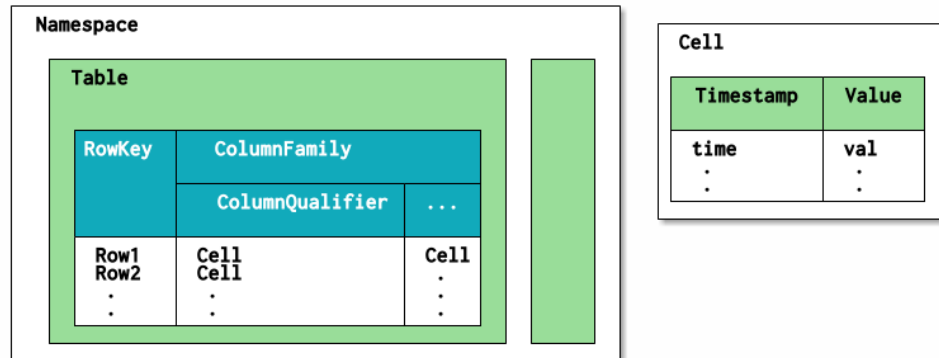


Figure 3: HBase の論理データモデル

### 3.4.3 Hadoop

Hadoop は大規模データセットの分散処理フレームワークである。Hadoop はモジュール化されているため、そのコンポーネントの殆どを別のソフトウェアに入れ替えることもできる柔軟な設計がされている。今演習では標準的な Hadoop の構成に付随してインストールされる、①Hadoop Common ②Hadoop YARN<sup>5</sup> ③Hadoop MapReduce ④Hadoop Distributed File System (HDFS) をそのまま利用している。

Common は他のモジュールに利用される基本的なライブラリ群である。YARN は Hadoop のリソース管理やスケジューリングを行い、MapReduce は分散処理のためのフレームワークである。HDFS は分散ファイルシステムで、大容量ファイルを扱うことができる。HDFS は大量の小さなデータを高速に扱うことを不得手としているので、HBase がこの補完を行っている。

### 3.4.4 Zookeeper

Zookeeper は Hadoop などにおける、構成情報の管理、分散処理の提供、またグループサービスの提供なども行う、分散アプリケーション全体を管理するツールである。使用用途は多岐にわたり、例えば Hadoop などにおける構成管理、Apache Storm<sup>TM</sup> 6 における処理の同期などに用いられる。ツリー状の階層化された名前空間を持ち、ノードと呼ばれる要素にサーバなどを割り当てている。高速処理や高い信頼性があるにもかかわらず、非常に簡単な API を持っていることが特徴である。ベンチマークとしては Zookeeper 3.4

<sup>5</sup>Yet Another Resource Negotiator

<sup>6</sup>リアルタイム高速分散処理フレームワーク

Documentation に記載されている。

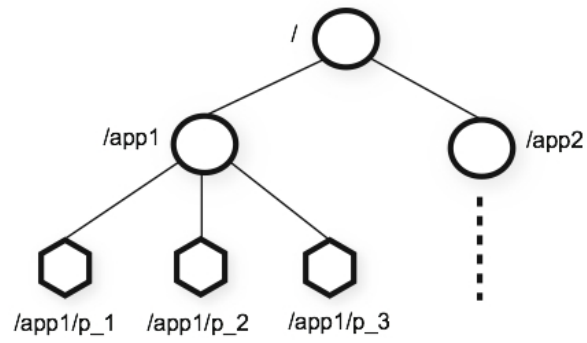


Figure 4: Zookeeper の階層構造

### 3.4.5 OpenTSDB

OpenTSDB とは HBase をホストとした<sup>7</sup> 時系列データベースで、その構成は① 時系列デーモン (以降 TSD と呼称する) ② コマンドラインユーティリティ、の2つである。特徴としては TSD にマスター・スレーブといった上下関係がないこと、HBase などのホストに各アプリケーションが直接触れる必要がないこと、標準的に保存されているデータをブラウザから視覚的に確認することができることなどが挙げられる。

APACHE HBASE					
Home Table Details Procedures Local Logs Log Level Debug Dump Metrics Dump HBase Configuration					
ServerName Start time La:					
localhost.localdomain,16201,1517818226215 Mon Feb 05 17:10:26 JST 2018 1 s					
Total:1					

### Backup Masters

ServerName	Port
Total:0	

### Tables

User Tables System Tables Snapshots

4 table(s) in set. [Details]

Namespace	Table Name	Online Regions	Offline Regions	Failed Regions	Split Regi
default	tsdb	1	0	0	0
default	tsdb-meta	1	0	0	0
default	tsdb-tree	1	0	0	0
default	tsdb-uid	1	0	0	0

Figure 5: HBase 上の OpenTSDB (tsdb という名前のテーブルがあることが確認できる)

これによって得られる恩恵として、アプリケーションをチームで開発・維持する際に OpenTSDB を軸にしてデータベース側とアプリケーション側に分割することができるということが考えられる。例えばアプリケーション側はデータベース側の分散等の開発が終わる前に仮設置の HBase に対して OpenTSDB を適用し、アプリケーションをほぼ本環境と同じように動かすことができる。またデータベースの分散数を増やしたい場合は、データベース側にのみ視点を当てて変更を行うことができる。

<sup>7</sup>正確には Google の BigTable もホストとなりうる

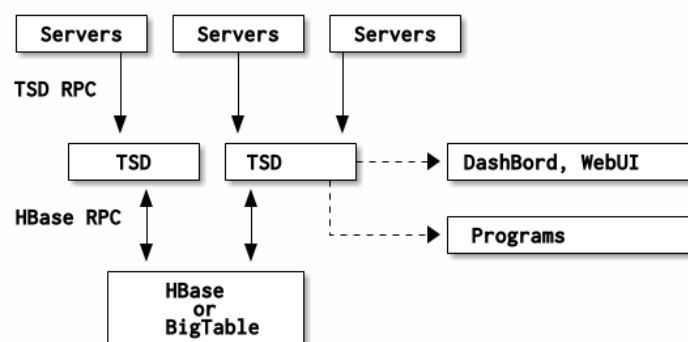


Figure 6: OpenTSDB の概略図

OpenTSDB の論理モデルは Metric と呼ばれるその時系列データのタイトルとも言える概念が最も外側に位置しており、この中にテーブルに近い構造が一つ含まれていると考えることが出来る。このテーブルの行キーはソートされたタイムスタンプであり、時系列データベースの要である。テーブルの列キーはタグと呼ばれるキーバリュー形式の識別子が0以上割り当てられており、これによって欲しいデータの絞り込みを行うことができる。

Metric			
Timestamp	Tag		...
	key	value	...
Time	Value		
.	.		
.	.		
.	.		

Figure 7: OpenTSDB の論理モデル

OpenTSDB はそのアクセスを HTTP API を用いて行うことができる。以降にその概要をまとめる。

### 3.4.6 OpenTSDB の HTTP API

OpenTSDB を利用するにあたって重要な要素に HTTP API の習得がある。このクエリによってアプリケーション開発者はデータの取得や送信を行うことになる。尚、HTTP

API を使わずに Telnet を用いる手段もあるが、どちらも機能として同等であるためここでは HTTP API についての説明のみに留める。

API は、データの取得に関してはクエリ文字列とボディ部の両方の手段をサポートしており、ボディ部を用いる場合はクエリ文字列を用いるよりも詳細な検索をかけることができる。対してデータの送信は PUT メソッドによるボディ部を用いた手段のみが利用できる。それぞれの具体例を示すと以下のようになる。

Table 4: OpenTSDB におけるクエリ例①

前提条件 クエリ内容	<ul style="list-style-type: none"> <li>・ http: //localhost:4242 に対して OpenTSDB が開いている</li> <li>1 年前から現在までの Metric nikkei-index における</li> <li>タグについて key が “type” 、 value が “close-price”</li> <li>であるデータを要求する</li> </ul>
クエリ文字列	<ul style="list-style-type: none"> <li>・ http: //localhost:4242/api/query \</li> <li>?start=1y-ago&amp;m=avg:nikkei-index{type=close_price}</li> </ul>
ボディコンテンツ	<ul style="list-style-type: none"> <li>・ http: //localhost:4242/api/query</li> <li>・ Content-Type JSON</li> <li>・ Body</li> <li>{ “start” : 1y-ago,</li> <li>  “queries” :</li> <li>    [{“aggregator” : “sum”,</li> <li>      “metric” : “nikkei-index”,</li> <li>      “tags” :</li> <li>        {“type” : “close-price”}</li> <li>    }]</li> <li>}</li> </ul>

Table 5: OpenTSDB におけるクエリ例②

前提条件 クエリ内容	<ul style="list-style-type: none"> <li>・ http: //localhost:4242 に対して OpenTSDB が開いている</li> <li>・ Metric “nikkei-index” の、 タグが、key は “type”、value は “close-price” である UnixTime が 1717055464 である時間に、 12000 という値を保存する</li> </ul>
	<ul style="list-style-type: none"> <li>・ http: //localhost:4242/api/put</li> <li>・ Content-Type JSON</li> <li>・ Body  <pre>{ "metric" : "nikkei-index",   "timestamp" : 1717055464,   "value" : 12000,   "tags" :     { "type" : "close-price" } }</pre> </li> </ul>

### 3.4.7 Grafana

Grafana は InfluxDB や Graphite 、 OpenTSDB などのデータベースに対してデータを視覚化させるためのツールで、 Web ブラウザから Grafana が開けることになるポート<sup>8</sup>にアクセスすることで、ユーザーフレンドリーなクエリ処理機構を用いてデータベースと通信を行い、得られた結果をグラフや表にすることができる。以下にその例を示す。

---

<sup>8</sup>標準では 3000 番

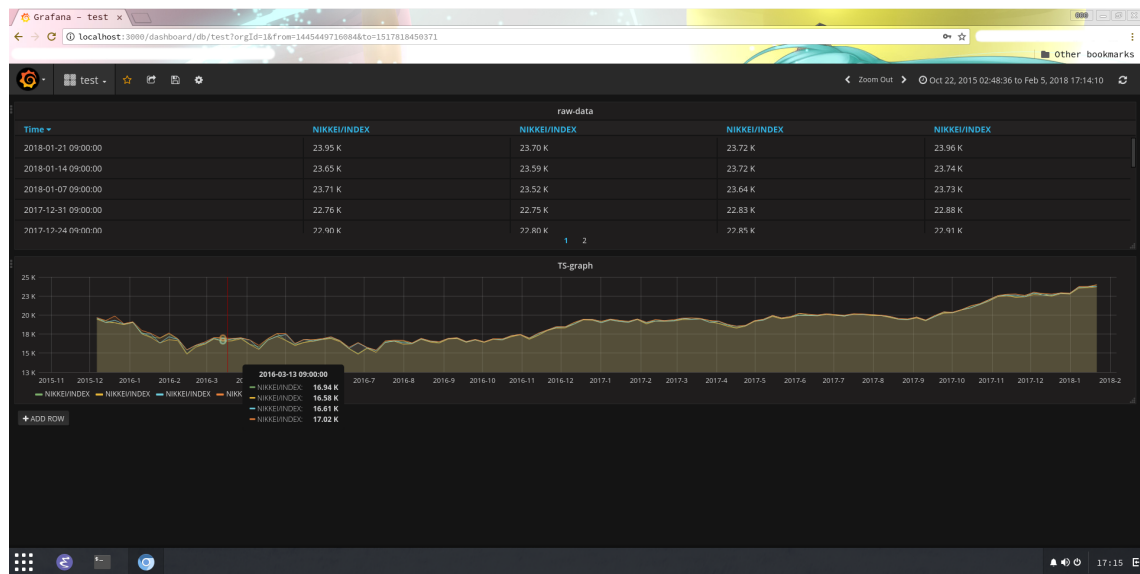


Figure 8: Grafana を用いた視覚化の例

## 4 Clojure を用いた JVM における高速計算技法

本演習の開始時、自分のこれまでのプログラム言語学習経歴<sup>9</sup>から、Lispの影響を受けた言語を選択することが最も演習に適していると考えており、更にHBaseを活用することが決定していたため、Javaに近くLispに近い言語としてJVM<sup>10</sup>上で動作するClojureを採用した。また演習を勧めていく上でフロントエンドの開発も行う必要が出てきたため、同様のシンタックスを用いるClojureScriptも採用し、この両方の言語を中心に学習した。

この章ではその内のClojureにおける高速計算手法についての学習成果を完結にまとめる。

### 4.1 Clojure 自身の高速化手法

ClojureにGPUライブラリ等を適用する以前に純粋なClojureで最適化されたコードを書くことが高速計算を行う際に重要であることは言うまでもない。本演習ではClojure自身の学習も兼ねClojure for the Brave and True [6]、Clojure High Performance Programming [8]を教材にClojureの最適化手法を学習した。具体的な学習内容としてはプログラム設計の見直しや基本的なシンタックスの見直し、効率の良いスレッド化・並列処理、プログラム全体のパフォーマンス測定法(プロファイリング法)などである。成果としてどこまでの性能向上が認められたかを具体的に比較することは難しいが、性格の良いプログラムを書くことが出来るようになったのではないかと考えている。

<sup>9</sup> 昨年の情報特別演習においてはPythonを中心に利用し、授業外でCommon Lispをある程度習得した

<sup>10</sup> Java virtual machine



## 4.2 ClojureCL

ClojureCL とは Clojure で OpenCL を用いるためのライブラリで C 言語で書かれる OpenCL のコードよりも簡潔なシンタックスで書くことが主張されている。このライブラリは JVM 上で OpenCL を動作させるため、JNI<sup>11</sup> を基盤としたライブラリである jocl を用いており、非常に低レベルな部分で OpenCL とリンクしているため、OpenCL そのものの知識が必要となるものの、その速度を十分に体感することが出来る。本演習ではドキュメントに記載されたソースコードを移し、自分の環境においてそれを体験するまでを行った。2018 年 2 月 1 日においてはより深い理解を行うために、OpenCL in Action [11] を学習している。

## 4.3 Neanderthal

Neanderthal は、Intel<sup>®</sup> MKL を用いた高速行列演算・線形代数のためのライブラリである。その速度は GPU を利用するモードの場合には大規模サイズの行列演算に関しては、Clojure / Java ライブラリに対しても大凡 3000 倍の高速化を達成し、CPU を利用する場合においても純粋な Java よりも 100 倍の高速化を達成している。このライブラリは後述する Clojure.core.matrix 系ライブラリに対してやや扱いが難しいが、その分大幅な高速化が望むことが出来る。またこのライブラリの依存関係は intel MKL のライブラリを含むことに意味があるため、intel MKL のライブラリ<sup>12</sup> をアプリケーション内に含んでしまえば標準的な環境で動作させることが出来る。

Single precision floating point (vs jBlas single precision):

Neanderthal and jBlas run on 4 cores, Vectorz doesn't have parallelization.

Since Clatrix does not support single-precision floating point numbers, I did this comparison with jBlas directly for reference (Neanderthal is still considerably faster :), but keep in mind that you can't use that from core.matrix.

Matrix Dimensions	Neanderthal	jBLAS	Vectorz	Neanderthal vs jBLAS	Neanderthal vs Vectorz
2x2	232.36 ns	362.00 ns	61.36 ns	1.56	0.26
4x4	237.72 ns	369.99 ns	129.34 ns	1.56	0.54
8x8	253.22 ns	476.57 ns	568.02 ns	1.88	2.24
16x16	372.30 ns	598.43 ns	3.45 µs	1.61	9.27
32x32	903.14 ns	1.37 µs	23.44 µs	1.52	25.96
64x64	2.80 µs	7.52 µs	218.64 µs	2.69	78.21
128x128	16.30 µs	31.48 µs	1.55 ms	1.93	94.85
256x256	126.25 µs	191.15 µs	12.28 ms	1.51	97.24
512x512	1.07 ms	1.25 ms	96.94 ms	1.16	90.21
1024x1024	7.93 ms	10.63 ms	778.46 ms	1.34	98.12
2048x2048	57.47 ms	104.95 ms	6.22 sec	1.83	108.16
4096x4096	470.12 ms	568.46 ms	50.06 sec	1.21	106.49
8192x8192	3.76 sec	4.85 sec	6.68 min	1.29	106.56

Figure 9: ベンチマーク Neanderthal Benchmarks より

<sup>11</sup>Java Native Interface

<sup>12</sup>Arch Linux においては /opt/intel/lib,/opt/intel/mkl/lib

残念ながら本演習では成果物を稼働させるサーバをどのように扱うかについて協議が不足しており、更に必要とされるライブラリがその環境で利用可能であるか不明であったため、実装に組み込ませることができなかった。しかし積極的にこちらを用いて開発を行いたいと考えている。ここで実行例の一部を紹介する。

Listing 1: test-Neanderthal.clj

```

1  (ns test-neanderthal.core
2    (:require
3      [uncomplicate.neanderthal.core :refer :all]
4      [uncomplicate.neanderthal.native :refer :all]
5      [uncomplicate.neanderthal.linalg :refer :all]))
6
7  ;; -----
8  ;; sample1
9  (def a (dge 2 3 [1 2 3 4 5 6]))
10 ;; #RealGEMatrix[double, m:n:2x3, layout:column, offset:0]
11 ;;           ↓           ↓           ↓           ↗
12 ;; →         1.00      3.00      5.00
13 ;; →         2.00      4.00      6.00
14 ;; ↙         ↘           ↘           ↘         ↘
15
16 (def b (dge 3 2 [1 3 5 7 9 11]))
17 ;; #RealGEMatrix[double, m:n:3x2, layout:column, offset:0]
18 ;;           ↓           ↓           ↗
19 ;; →         1.00      7.00
20 ;; →         3.00      9.00
21 ;; →         5.00     11.00
22 ;; ↙         ↘           ↘           ↘         ↘
23
24 (mm a b)
25 ;; #RealGEMatrix[double, m:n:2x2, layout:column, offset:0]
26 ;;           ↓           ↓           ↗
27 ;; →        35.00     89.00
28 ;; →        44.00    116.00
29 ;; ↙         ↘           ↘           ↘         ↘
30
31 ;; -----
32 ;; sample2
33 (def A (dge 3 2 [1 0 1 1 1 2]))
34
35 (def or (qrfp A))
36 ;; #RealGEMatrix[double, m:n:3x2, layout:column, offset:0]

```

```

37 ;;           ↓           ↓           ↵
38 ;; →         1.41      2.12
39 ;; →        -0.00      1.22
40 ;; →        -2.41      3.15
41 ;; └──────────────────┐
42
43 (def r (dge 2 2 (:or or)))
44 ;; #RealGEMatrix[double, m:n:2x2, layout:column, offset:0]
45 ;;           ↓           ↓           ↵
46 ;; →         1.41      2.12
47 ;; →        -0.00      1.22
48 ;; └──────────────────┐
49
50 (def q (org or))
51 ;; #RealGEMatrix[double, m:n:3x2, layout:column, offset:0]
52 ;;           ↓           ↓           ↵
53 ;; →         0.71     -0.41
54 ;; →         0.00      0.82
55 ;; →         0.71      0.41
56 ;; └──────────────────┐
57
58 (def b (dge 3 1 [1 0 -2]))
59
60 (def x (mm (tri (trf r)) (trans q) b))
61 ;; #RealGEMatrix[double, m:n:2x1, layout:column, offset:0]
62 ;;           ↓           ↵
63 ;; →         1.00
64 ;; →        -1.00
65 ;; └──────────┐
66
67 ;; -----
68 ;; sample2 ~another solution~
69 (def A (dge 3 2 [1 0 1 1 1 2]))
70
71 (def b (dge 3 1 [1 0 -2]))
72
73 (def x_ (dge 2 1 (ls A b)))
74 ;; #RealGEMatrix[double, m:n:2x1, layout:column, offset:0]
75 ;;           ↓           ↵
76 ;; →         1.00
77 ;; →        -1.00
78 ;; └──────────┐

```

---

5 行目までの内容は依存関係の解決である。 Sample1 において単純な行列の掛け算を

行っており、Sample2 は QR 分解を用いて  $Ax = b$  の解を求めている。そして Sample2 ~ another solution ~ はこれを存在しているライブラリ関数を用いて解いたものである。両者の速度差はこのサイズの行列演算であればほぼないが、大規模サイズの行列であった場合は後者のほうが圧倒的に速い。後者も前者もほぼ直接 Fortran のライブラリである LAPACK<sup>13</sup> を触っているため、計算途中で結果を取り出している前者のほうが効率が悪いのである。

このコードからわかるように、このライブラリが返す値は必ずしも求めている/求まった解答の形を示していない。この理由は Intel MKL 内のソースコードが与えられたデータのメモリに解答を書き込む性質があるためである。この破壊的代入を行う性質は高速化に大きな貢献をしているとともに、高い副作用と難解さを招いている原因であると考えられるが、このライブラリを利用するためには Intel MKL のドキュメントを精読することや、内部の Fortran による実装を眺める他にない。

## 4.4 Clojure.core.matrix

先に紹介した2つに対してこちらは非常におとなしいライブラリであり、Clojure の標準的な算術関数のラップや行列演算に関するライブラリの基盤を開発している。ライブラリの基盤というのは、Java などのオブジェクト指向言語におけるインターフェースのようなもので、実装すべき関数を先に示しておくことで、それを様々な手法によって実装・更新されていくことで長期的にそのライブラリ群を使うことが出来るという利点がある。本演習では、高速さが持ち味である vectorz-clj や、jblas を用いて実装されており更に関数が充実している clatrix の2つを検討し、その両方を利用した。

### 4.4.1 vectorz-clj

Vectorz-clj は純粋に JVM で動作する高速な行列計算ライブラリを掲げており、導入にかかるコストの低さが魅力的である。問題としては行列の結合・切り出しに関する関数のいくつかの挙動が不自然であることで、その点を除いては後述する clatrix よりも概ね高速に動作する。

### 4.4.2 clatrix

clatrix は jblas をラップしたライブラリであり、行列計算において必要とされる関数をほぼすべて網羅しており、先述のライブラリで不足した部分を補完するために利用した。このライブラリを利用するためには jblas がインストールされていることが必要であるため、標準的な環境にこれを用いたアプリケーションを実行したとしても正常に動作しない。不足する関数を自力で補完することでこのライブラリを使用しないという選択肢もあるため、よりアルゴリズムの能力を磨いて自力に必要な関数を補完したいと考えている。

---

<sup>13</sup>Linear Algebra PACKage

## 4.5 ACM3 (Apache Common Math 3)

計算速度そのものの向上という意味ではこのカテゴリからはやや離れるが、良質なアルゴリズムを用いている様々な数学に関するライブラリとして ACM3 がある。本演習ではそのうちの、アメーバ法に関する関数を ARMA モデルにおけるパラメータ推定のために利用した。

## 5 ARIMA モデルによる時系列分析

(S)ARIMA モデルは時系列分析手法の一つであり、本演習の要とも言える機械学習手法である。本演習ではこのモデルとその周辺手法を実装した。

ARIMA モデルは 正確には “Autoregressive integrated moving average model” と呼ばれ、概要は ① I ② AR ③ の 3 要素によって構成されており、この適用できるデータは「非定常過程が見られる」時系列データ<sup>14</sup>である。定常過程と非定常過程の違いについては後述する 5.2 単位根検定 で説明を行うが、時系列データには非定常過程を持っている場合が少なからずあり、更に ARIMA モデルを ARMA モデルに変換することは非常に容易であるため ARIMA モデルを実装することによって実質的に定常過程、非定常過程両方の性質を持った時系列データを分析することが出来る。尚 ARIMA モデルの分析対象はある時系列データ内の時点間の関係である、自己相関<sup>15</sup>にある。

また ARIMA モデルの発展として季節階差を削除することを目的とした SARIMA モデルもあるが、こちらは 5.7 SARIMA モデルについて にある理由により開発を中断した。

以降にこのモデルの実装において必要になる知識を紹介する。尚ここで使用する式の書式に関しては 11 付録に記載する。

### 5.1 BackShift 記法

BackShift 記法とは、記号 “B” という演算子を用いた時系列データを表現するための手法であり、以下のような使われ方をする。[10]

$$(1 - B)Y_t = 1Y_t - BY_t = Y_t - Y_{t-1} \quad (1)$$

$$(1 - B)^2Y_t = (1 - B)(1 - B)Y_t \quad (2)$$

$$\begin{aligned} (1 - B^k)Y_t &= Y_t - B^kY_t \\ &= Y_t - Y_{t-k} \end{aligned} \quad (3)$$

但し  $Y_t$  は時系列データを表しており、また今後の説明のため、 $t$  が大きいほど最近のデータであるものとする。

式 (1) は一次階差を表しており、後述する AR モデルでは AR(1) の場合に用いられる。式 (2) は二次階差を表しており、同様に AR(2) の場合に用いられる。式 (3) はある区間を開けて階差を取っており、これは季節階差を取る際等に用いられる。季節階差という考

<sup>14</sup> 「定常過程を持っている」時系列データは ARMA (Autoregressive moving average) モデルでの推定となる

<sup>15</sup> 系列相関ともいう

え方から一旦離れてわかりやすい例を挙げるとすれば、時系列データが月単位のデータであった場合、昨年と今年の差分を取る場合には、 $(1 - B^{12})Y_t$  という形をとることになる。

この記法を用いることで  $n$  次階差や季節階差を表しやすくなり、また関数型言語などにおいてはその実装の手がかりを得ることが出来る<sup>16</sup>。

## 5.2 単位根検定

実世界に存在する多くの時系列データは非定常過程を持っていることが示唆されている。この示唆について Jackknifing multiple-window spectra [15] から有名な一説を引用すると以下ようになる。

Experience with real-world data, however, soon convinces one that both stationarity and Gaussianity are fairy tales invented for the amusement of undergraduates.

- Thomson, 1994

ARIMA モデルでは「非定常な」時系列データを 階差 を取ることによって「定常な」時系列データに変換し ARMA モデルに適用する。ARIMA モデルにおける I 部分には  $d$  次の階差を取るという意味が含まれている。一般にこの階差は一次であるらしいが、実装側では以降に紹介する ADF 検定 をおこなうことで定常性を判定した。またこの他にも、KPSS 検定や Ljung-Box 検定などがあるが、単位根検定という観点から KPSS 検定のみを追加で紹介することとする。

以降における  $y_t$  について定義する。 $Y_t$  を議論の時系列データとして、

$$y_t = Y_t - E(Y_t) = Y_t - \mu - \mu_1 t \quad (4)$$

正確ではないが、ここにおける  $\mu$  は  $t = 0$  における時系列データの値、 $\mu_t$  は 時系列データの傾きと考えることが出来る。

### 5.2.1 定常性の性質

ここまで“定常性”という言葉を用いてきたが、この定常性について簡単に触れておく。定常性とは同時分布が時間を通じて変わらないこと<sup>17</sup>を意味しており、以下のような性質を持っている。

<sup>16</sup> $(1 - B^n)$  という意味を持つ関数を定義することで理論上 ARIMA モデルに必要な階差に関する関数は満足することが出来る

<sup>17</sup>この同時分布が同一であることを持つ時系列過程を特に“強定常である”という

- $E(Y_t) = \mu$   
母平均 (population mean) は時点  $t$  に依存しない。
- $Var(Y_t) = \gamma_0$   
分散 (variance) は 時点  $t$  に依存しない
- $Cov(Y_t, Y_{t-j}) = \gamma_j$   
共分散 (covariance) は 時点  $t$  に依存しない<sup>18</sup>

更に  $E(Y_t) = \mu \wedge Cov(Y_t, Y_{t-j}) = \gamma_j$  のみである場合を特に “弱定常である” という。

逆に非定常過程の時系列データに目を向けたとき、経済学上重要な要素に以下のようなものがある。

- 確定的トレンド (deterministic trend)  
 $Y_t = \beta t + \epsilon_t$  where  $\epsilon_t \sim iid(0, \sigma^2)$  と表され、 $E_{DT}(Y_t) = \beta t$ 、 $Var_{DT}(Y_t) = \sigma^2$  である。こちらは直ちにトレンド定常 (trend stationarity) という形に変形することが出来る。
- 確率的トレンド (stochastic trend) 又は単位根過程  
 $(1-B)Y_t = \beta + \epsilon_t$  where  $\epsilon_t \sim iid(0, \sigma^2)$  と表され、 $E_{ST}(Y_t) = \beta t$ 、 $Var_{ST}(Y_t) = t\sigma^2$  である。  
確定的トレンドに比べこちらは時間が経過する程に大きな影響を及ぼすことになる。こちらは後述する単位根検定として利用できる ADF 検定や KPSS 検定を行うことで発見することが出来る。
- 構造変化  
その時系列が予想しない変化 (経済データであるならば、例えば突発的な戦争や飢餓) を受けた際に起こる。本来はこれに対する検定も用意するべきであったが、どのような条件をフラグとして検定が行われるべきであるかが理解できなかったため実装することができなかった。

---

<sup>18</sup> $\sigma_{Y_t Y_{t-j}}$  と表すこともある

## Simulating DT and ST time series

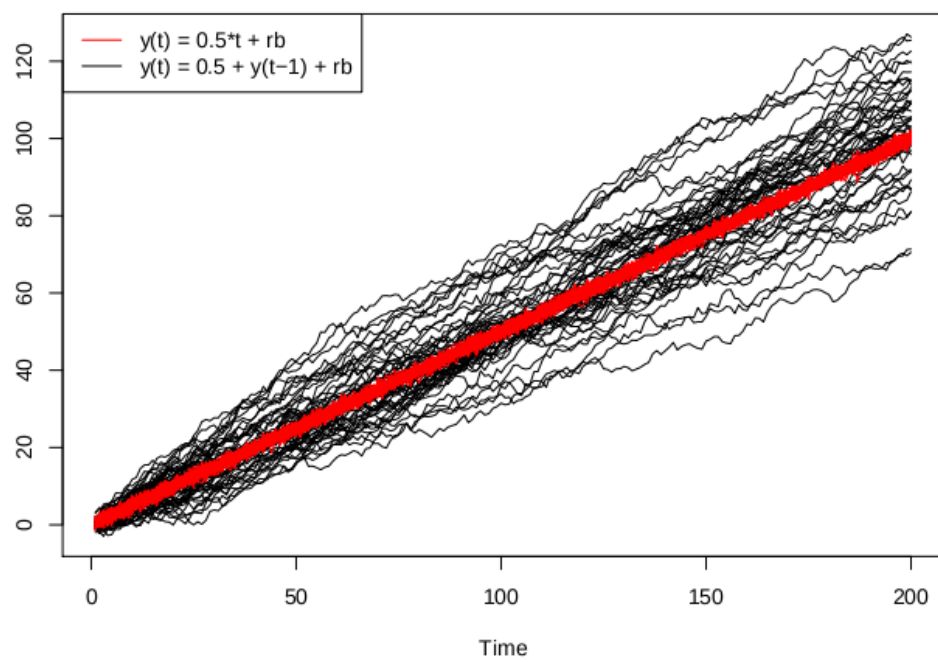


Figure 10: difference between Deterministic and Stochastic trend [5]



### 5.2.2 ADF 検定

ADF 検定 (Augumented Dickey-Fuller test) は DF 検定 (Dickey-Fuller test) の拡張である。

DF 検定とは  $Y_t$  について自己回帰モデル AR(1) を作成し以下の条件を用いて仮説検定を行う手法である。

AR(1) モデル  $Y_t = \theta Y_{t-1} + \epsilon_t$  where  $\epsilon_t \sim iid(0, \sigma^2)$  とする。(AR モデルそのものについては 5.3 AR モデル で解説を行う)

単位根を持っていることを帰無仮説とし、定常であることを対立仮説であるとする。

つまり、帰無仮説  $H_0$  と対立仮説  $H_A$  は以下のように定義できる

$$H_0 : \theta = 1 \quad (5)$$

$$H_A : \theta < 1 \quad (6)$$

ここからモデルを変形し、以下の式を立てる。

$$\Delta Y_t = (\theta - 1)y_{t-1} + \epsilon_t \quad (7)$$

上式において、 $\pi = (\theta - 1)$  と置換した場合、帰無仮説と対立仮説は以下のように更新される。

$$H_0 : \pi = 0 \quad (8)$$

$$H_A : \pi < 0 \quad (9)$$

この検定値は簡単な t 検定によって求めることが出来、以下の式によって得られる。

$$\begin{aligned} \tilde{\tau} &= (\hat{\theta} - 1)/se(\hat{\theta}) \\ &= \hat{\pi}/se(\hat{\pi}) \end{aligned} \quad (10)$$

これを自ら指定した有意水準 (5% 又は 1% であることが多い) において検定する。この検定手法では対立仮説が成り立つならば定常性を認めることが可能である。混乱を招かないために強調するが、この検定における帰無仮説は、“単位根を持っている” ことである。これが棄却されれば “定常である” ことを認めることができる。

また上の場合において、元データの一次階差と取っていることが明らかであるが、この検定で定常であると認められた場合、このデータは 1 次の単位根 があるという。同様に d 次の単位根がある とは、d - 1 次までの単位根検定においてすべて 非定常である と判断され、d 次において初めて 定常である と判断されたことを示している。

DF 検定が AR(1) モデルに対する検定であることにたいして、ADF 検定は AR(n) モデルにまで対象を拡大したものであり、一般式は難解であるため省くが、例えば AR(3) モデルは以下のように示すことが出来る。

$$Y_t = \theta_1 Y_{t-1} + \theta_2 Y_{t-2} + \theta_3 Y_{t-3} + \epsilon_t \text{ where } \epsilon_t \sim iid(0, \sigma^2) \quad (11)$$

$$Y_t - Y_{t-1} = (\theta_1 - 1)Y_{t-1} + \theta_2 Y_{t-2} + \theta_3 Y_{t-3} + \epsilon_t \quad (12)$$

$$\begin{aligned} \Delta Y_t &= (\theta_1 + \theta_2 + \theta_3 - 1)Y_{t-1} \\ &\quad + (\theta_2 + \theta_3)(Y_{t-2} - Y_{t-1}) + \theta_3(Y_{t-3} - Y_{t-2}) + \epsilon_t \\ &= (\theta_1 + \theta_2 + \theta_3 - 1)Y_{t-1} + (\theta_2 + \theta_3) * \Delta Y_{t-1} + \theta_3 \Delta Y_{t-2} + \epsilon_t \end{aligned} \quad (13)$$

これより  $\pi = (\theta_1 + \theta_2 + \theta_3 - 1)$  において DF 検定と同様に t 検定を行う。

尚この検定にはいくつかの追加要素として、定常過程にある時系列データの平均値を考えるパターンや、時系列データに傾きがある場合を考慮したパターンがある。これらは人為的にデータを確認することで決定するが、R 言語や Python などの ADF 検定ではすべてのパターンを一度に実行している場合がある。これは ADF 検定そのものは計算コストが低いため、すべてのパターンを網羅しても問題がないためである。

### 5.2.3 KPSS 検定

KPSS 検定 (Kwiatkowski-Phillips-Schmidt-Shin test) とは先述の ADF 検定に対して帰無仮説と対立仮説を反転させたものとイメージすることが出来る。この検定においては以下の式を中心に展開する。

$$\begin{aligned} Y_t &= \xi_t + \epsilon_t \\ \text{where } \xi_t &= \xi_{t-1} + v_t \\ v_t &\sim iid(0, \sigma_v^2) \\ \epsilon_t &\sim iid(0, \sigma^2) \end{aligned} \quad (14)$$

この式における  $\xi_t$  はランダムウォークを示している。尚ランダムウォークとは次に現れる値が確率的にランダムであることを示す。また、 $\epsilon_t$  はその性質から定常過程を示している。

仮に  $\sigma_v^2 = 0$  であるとしたとき、 $\xi_t = \xi_0$  であることから上式に影響を加える要素は  $\epsilon_t$  のみとなり、つまり  $Y_t$  は定常であるとみなすことが出来る。これを用いて上式を変形すると以下ようになる。

$$Y_t = \hat{\mu} + \hat{\epsilon}_t \quad (15)$$

ここで仮説検定を行う。帰無仮説は定常過程を示している式 (15) であり、つまりは式 (14) における  $\sigma_v^2 = 0$  である。対立仮説はこの逆であり、非定常であること、つまり  $\sigma_v^2 > 0$  である。

$$H_0 : \sigma_v^2 = 0 \quad (16)$$

$$H_0 : \sigma_v^2 > 0 \quad (17)$$

検定は以下の式を用いて行う。

$$KPSS = 1/T^2 (\sum_{t=1}^T S_t^2) / \hat{\sigma}_\infty^2$$

$$\text{where } S_t = \sum_{s=1}^t \hat{e}_s \quad (18)$$

また上式における  $\hat{\sigma}_\infty^2$  とは  $\epsilon_t$  の長期変動に関する HAC 推定量<sup>19</sup> である。  
以下に  $\sigma_\infty^2$  の例を示す。[7]

$$\sigma_\infty^2 = \lim_{T \rightarrow \infty} (1/TE((\sum_{t=1}^T \epsilon_t)^2)) \quad (19)$$

ADF 検定あったようにこちらにもいくつかのパターンがあり、トレンド定常の場合などの場合に合わせた形に式(14)が存在し、それに伴って仮説検定の内容に多少の変化がある。

実装においては HAC 推定量を理解・実装することができなかったため、KPSS 検定も実装することができなかった。

### 5.3 AR モデル

AR(p) モデル (自己回帰モデル) は以下の式で表すことが出来るモデルである。重回帰モデルが説明変数の線型結合を用いて関心のある変数を予測していることに対して、自己回帰モデルは説明変数を過去の観測値に置き換えたものであると言える。

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \epsilon_t$$

$$= c + \sum_{i=1}^p (\phi_i Y_{t-i}) + \epsilon_t$$

$$\text{where } c \text{ is constant}$$

$$\epsilon \sim iid(0, \sigma^2) \quad (20)$$

式からわかるように  $c$  は定数項である。

### 5.4 MA モデル

MA(q) モデル (移動平均モデル) は以下の式で表すことが出来るモデルである。同様に重回帰モデルの説明変数を過去のノイズの重みに置き換えたものであると言える。これとよく似たものに移動平均という手法があるが、移動平均は過去の値のサイクルを推定するために用いられ、MA(q) モデルは将来の値を予測するために用いられる。<sup>20</sup>移動平均と MA モデルは正確には別のものであり、混同されるべきではない。MA モデル自体は有限インパルス応答に近い発想である。[16]

<sup>19</sup>特に Newey-West 推定量を用いられることが多い

<sup>20</sup>{

$$\begin{aligned}
Y_t &= c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q} \\
&= c + \epsilon_t + \sum_{i=1}^q (\theta_i \epsilon_{t-i})
\end{aligned} \tag{21}$$

## 5.5 ACF と PACF

ARMA モデルにおいて<sup>21</sup>、AR(p), MA(q) の p, q の値を決定する際の指標となるものに、ACF、PACF がある。この 2 つはグラフによって視覚化され、この 2 つのグラフから p, q の値は手動で決定される。<sup>22</sup> [13]

### 5.5.1 ACF

ACF (autocorrelation function) は自己相関係数とも呼ばれ、元データとある時点分だけずらしたデータとの相関係数を計算する。つまり以下の式を計算することになる。尚  $t-s$  はそのずらした量 (ラグと呼ぶ) である。またこの場合に求まる値を、 $t-s$  次の ACF と呼ぶことがある。

$$\begin{aligned}
\rho(s, t) &= (E((Y_t - \mu_t)(Y_s - \mu_s)))/(\sigma_{Y_t} \sigma_{Y_s}) \\
&= (E((Y_t - \mu_t)(Y_s - \mu_s)))/\sigma^2 \\
&= Cov(Y_t, Y_s)/\sigma^2
\end{aligned} \tag{22}$$

上式では  $\sigma_t \sigma_s = \sigma^2$  が成り立っている。これは定常過程を持つ時系列データの分散が時点に依存しないためである。

尚 ACF と PACF には閾値があり、この閾値を超えた値を元に p, q を考えていく。以下にその境界値を求める式を示す。但し n はデータサイズである。

$$threshold = \pm 2/\sqrt{n} \tag{23}$$

### 5.5.2 PACF

PACF (partial autocorrelation function) は偏自己相関とも呼ばれ、ある時点とそこからある時点分だけ離れた時点の二点間の、その間の存在の影響も考慮した場合における相関関係である。一般式は複雑であるため、2 次、3 次の 偏自己相関 の式を示す。尚、1 次の PACF は 1 次の ACF に等しい。計算の都合上、k 次の 偏自己相関 を  $\phi(k, k)$  と表す。

$$\phi(2, 2) = Cov(Y_t, Y_{t-2}|Y_{t-1})/(\sigma_{Y_t|Y_{t-1}} \sigma_{Y_{t-2}|Y_{t-1}}) \tag{24}$$

$$\phi(3, 3) = Cov(Y_t, Y_{t-3}|Y_{t-1}, Y_{t-2})/(\sigma_{Y_t|Y_{t-1}, Y_{t-2}} \sigma_{Y_{t-3}|Y_{t-1}, Y_{t-2}}) \tag{25}$$

<sup>21</sup> ACF, PACF は定常過程であるデータに適用される

<sup>22</sup> R 言語における auto.arima 関数などはこのような手間なしでモデルを決定することが出来るが、これは p, q などの値を元データにかかわらず適当に複数定めモデルを作成し、AIC などを用いて最も良いモデルを選択しているからである。

このように式がラグごとに異なるため、実装上計算には以下の行列式を用いた Durbin-Levinson recursion が用いられることが多い。尚 以下の式における  $\rho(s)$  とは 自己相関  $\rho(s, t)$  を示している。

$$\begin{pmatrix} \rho(0) & \rho(1) & \rho(2) & \cdots & \rho(k-1) \\ \rho(1) & \rho(0) & \rho(1) & \cdots & \rho(k-2) \\ \rho(2) & \rho(1) & \rho(0) & \cdots & \rho(k-3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho(k-1) & \rho(k-2) & \rho(k-3) & \cdots & \rho(0) \end{pmatrix} \begin{pmatrix} \phi(k, 1) \\ \phi(k, 2) \\ \phi(k, 3) \\ \vdots \\ \phi(k, k) \end{pmatrix} = \begin{pmatrix} \rho(1) \\ \rho(2) \\ \rho(3) \\ \vdots \\ \rho(k) \end{pmatrix} \quad (26)$$

偏自己相関 の値  $\phi(i, j)$  は AR モデルに似た回帰式を得られる。以下の場合 は  $\phi(2, 2)$  についての例である。

$$\begin{aligned} \hat{y}_t &= \phi(2, 1)y_{t-1} + \phi(2, 2)y_{t-2} + \epsilon_t \\ \text{where } \hat{y}_t &= Y_t - \mu \end{aligned} \quad (27)$$

### 5.5.3 p, q の決定法

AR(p)、MA(q) モデルの p, q の値を考える際には先述の通り ACF と PACF を用いることがある。この 2 つのグラフと p, q の関連について簡単に説明する。説明のために具体的なグラフ A, B を以下に示す。

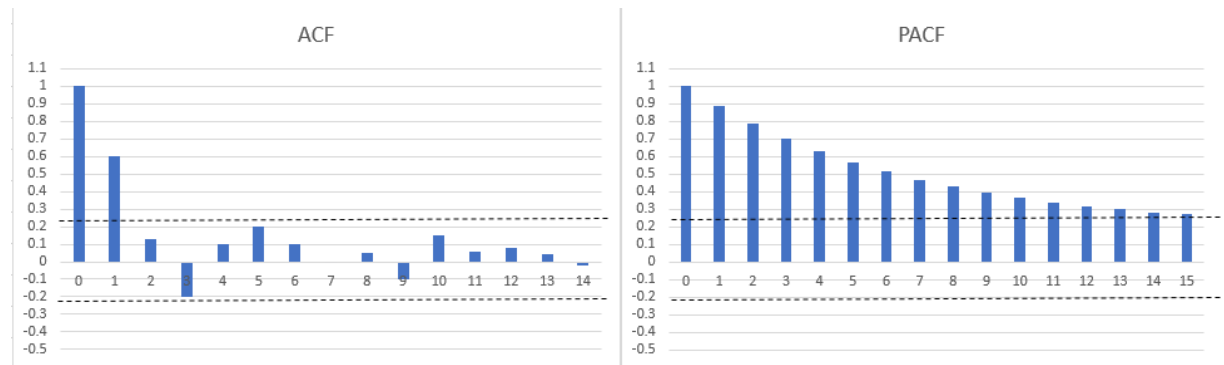


Figure 11: グラフ A

簡潔に言うならば ACF と q、PACF と p が強く関係している。

グラフ A に着目すると、自己相関 の 1 次までの値が閾値を大きく超えていることがわかる。自己相関 はラグ k の共分散であり、偏自己相関 と違って 自己相関 はラグ k-1 やそれ未満の値の影響を考慮していない。よってノイズの重みを示している MA(q) モデルの、q の値が決まると考えることができ、この例で言うならば  $q = 1$  であると考えることが出来る。

同様にグラフ B に着目すると、偏自己相関 の 2 次までの値が閾値を大きく超えている

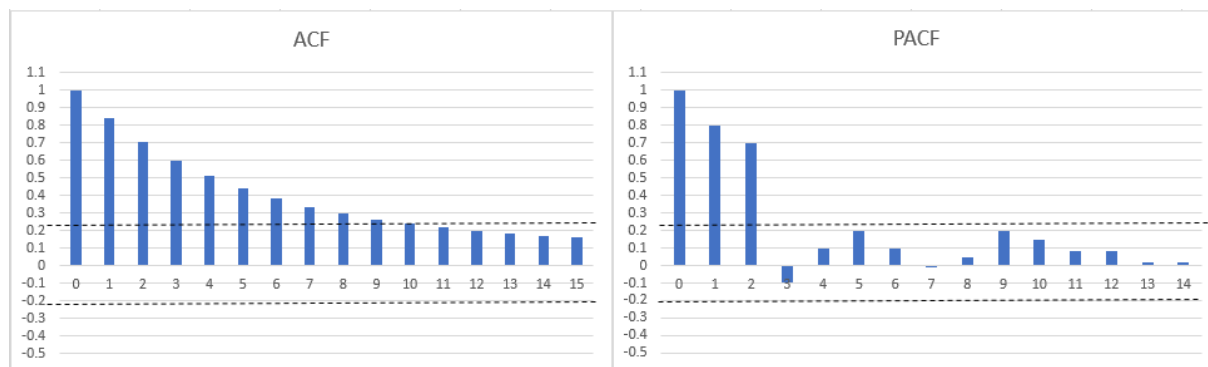


Figure 12: グラフ B

ことがわかる。偏自己相関 で説明したように、偏自己相関 の値は AR モデルに非常に近い形で用いることができ、このことから AR(p) モデルの  $p$  の値が決まると考えることができることがわかり、 $p = 2$  がふさわしいことがわかるだろう。<sup>23</sup>

尚この  $p, q$  の値は観測者によって異なることがあるため、完全な正解があるわけではない。

MA モデル (グラフ A) における PACF、AR モデル (グラフ B) における ACF のグラフは先細りの形を示している。これは上に示した 2 つのグラフが極めて理想的な MA / AR モデルを示していることを意味している。ARMA モデルの場合は ACF、PACF は共に先細りの形ではないため、この 2 つのグラフを見て係数  $p, q$  を決定しなければならない。ところが AR モデルと MA モデルを同時に用いる場合、お互いの影響を打ち消し合ってしまう可能性があり、この係数決定は慎重に行わなければならない。ここで階差の次数  $d$  も合わせた詳しい係数の決定基準と性質を以下に示す。[9] 尚 ここで用いる“カットオフ”とは、グラフ A の ACF グラフにおける 0 ~ 2 次の自己相関 から 3 次の自己相関 への変化を意味する。

1. その系列の 自己相関 が高い場合にはより高い階差を取る必要がある
2. 1 次の 自己相関 が 0 以下の場合、或いは 自己相関 がすべて小さくパターンが見られない場合には、それ以上の階差を取る必要はない  
1 次の 自己相関 が -0.5 以下の値を取っているならば、階差を取りすぎている可能性がある
3. 階差の最もふさわしい次数は、多くの場合、標準偏差が最も低い階差の次数になる<sup>24</sup>

<sup>23</sup> 詳しい証明について調べたものの、発見ができなかったため 引用を行ったサイト [13] に書かれた内容を元に記述した

<sup>24</sup> これは階差は一般的には一次であるらしいという事実に反するが、階差が一般的に一次であるらしいという根拠が見つからなかったため検証を行うことができなかった

4. 階差を取らない、つまり 0 次階差を取るようになったモデルは元の時系列データが定常であることを示している。  
1 次階差を取るようになったモデルは元の時系列データが何らかの傾向（ランダムウォークや Simple Exponential Smoothing type model<sup>25</sup>）を持っていると考えられる。  
2 次階差を取るようになったモデルは元の時系列データが time-varying trend <sup>26</sup> (random trend や Linear Exponential Smoothing type model<sup>27</sup>) を持っていると考えられる。
5. 階差を取らないモデルは通常定数項を持っている。2 次階差を持つモデルは通常定数項を持たない。  
1 次階差を取るモデルで、元の時系列が 0 でない傾きを持っているならば定数項を考慮する必要がある。
6. 差分を取った時系列の PACF グラフが鋭いカットオフを示している場合、或いは更に 1 次の自己相関が正である場合にはモデルに AR モデルを追加することを検討する必要がある。  
AR(p) モデルの p の値はカットオフを示している次数である。(グラフ B ならば  $p = 2$ )
7. 差分を取った時系列の ACF グラフが鋭いカットオフを示している場合、或いは更に 1 次の自己相関が負である場合にはモデルに MA モデルを追加することを検討する必要がある。  
MA(q) モデルの q の値はカットオフを示している次数である。(グラフ A ならば  $q = 1$ )
8. ARMA モデルが差分を取った時系列に適合する場合には p, q の値をそれぞれ 1 下げた場合のモデルを検証する必要がある。  
ARMA(p, q) ならば ARMA(p - 1, q)、ARMA(p, q - 1)  
特に ARMA モデルのパラメータ推定値が収束するために 10 回以上の反復を必要とした場合にはこれを行うべきである。
9. AR モデルに単位根が含まれている場合 <sup>28</sup>、p の値を 1 下げて階差の次数を 1 上げるべきである。

<sup>25</sup>単純指数平滑法モデルとも言う

<sup>26</sup>翻訳が見つけることができなかったため意識をすると、時間的変動傾向

<sup>27</sup>Brown の線形指数平滑法モデルとも言う

<sup>28</sup>AR モデル部分のパラメータの合計値が大凡 1 であるとき

10. MA モデルに単位根が含まれている場合<sup>29</sup>、 $q$  の値を 1 下げて階差の次数を 1 下げるべきである。
11. 長期予測が奇妙な動きを見せている場合には、AR モデルか MA モデルに単位根が存在している可能性がある。

## 5.6 パラメータ推定

ここまでにおいて ARIMA モデルの係数  $p$ ,  $d$ ,  $q$  を決定することが出来る。ここから AR モデル、MA モデルそれぞれにおける、 $\phi$ 、 $\theta$  の値を推測する手法を説明する。このパラメータ推定は最尤推定 (maximum likelihood) や CSS (conditional sum of squares) が用いられるようである。本演習では対数尤度関数に対してアメーバ法を適用してパラメータ推定を行った。これに際しアメーバ法の適用や、対数尤度関数の実装などについては Clojure for Data Science [4] を活用した。

### 5.6.1 対数尤度関数

本演習では計算の都合上対数尤度関数を用いて最尤推定を行った。対数尤度関数に関しては既存のコードを読んで実装し、実装後に資料 [1] を用いて学習を行った。しかし理解が曖昧な部分が多いため本演習では詳細を示すことができなかった。

### 5.6.2 アメーバ法

アメーバ法とは Nelder-Mead 法とも呼ばれるパラメータ推定などに用いられる手法であり、本演習では対数尤度関数に対してこれを適用した。

アメーバ法の特徴に目的となる関数の導関数が不要であることがある。これは本演習における対数尤度関数などの複雑な関数に対して有効なアルゴリズムである。アメーバ法の基本的なアルゴリズムは、反射、収縮、膨張の三種類であり、最適解の存在する位置に向かってアメーバのように近づいていくことからこの名前がついている。

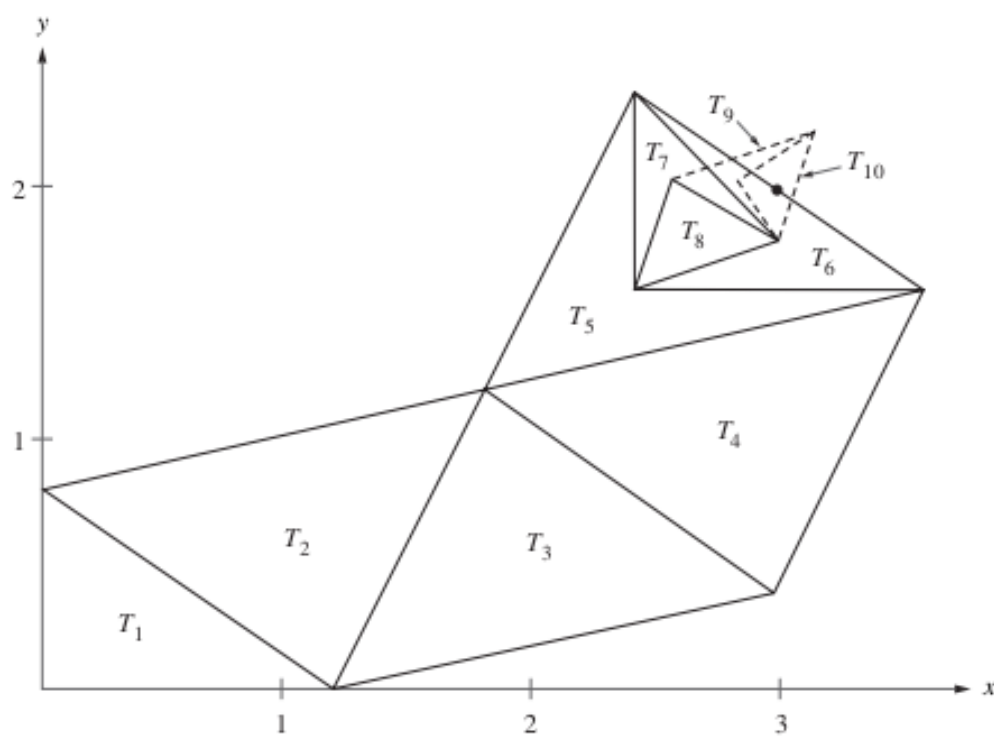
具体的な動きを二次元の図で表したものを引用 [2] すると以下のようになる。 $T_i$  はアメーバの領域を示しており、これは議論の次元数に対して 1 つ多い頂点を持っている (この場合では 2 次元に対して 3 頂点)。番号  $i$  は動きの順番を示しており、反射<sup>30</sup>、縮小を行いながら収束していることがわかる。拡大についてはこの図では見られない動きであるが、頂点を動かした先がより求めたい最大/最小に近いと判断できた場合に、そのまま頂点を引き伸ばす様子を考えることができる。

本演習では詳しいアルゴリズムの実装については調べず、ACM3 に含まれている関数をそのまま用いた。

<sup>29</sup>MA モデル部分のパラメータの合計値が大凡 1 であるとき

<sup>30</sup>反転、或いは移動と考えることも出来る





**Figure 8.10** The sequence of triangles  $\{T_k\}$  converging to the point  $(3, 2)$  for the Nelder-Mead method.

Figure 13: アメーバ法の概略図

### 5.6.3 AIC

AIC (赤池情報量) はモデルの過適合などを判定するための指標であり、求めることができた対数尤度の適合度に対して  $p, q$  の値を考慮して計算される。具体的には以下の式を用いた。

$$AIC = 2 * (p + q + 1) - 2 * \log L \quad (28)$$

尚  $L$  とは対数尤度である。AIC を最小化させることがモデルの適合度を上げることになる。

## 5.7 SARIMA モデルについて

本演習においては SARIMA (seasonal arima) モデルを作成することが目標であったが、現実のデータにおいて等間隔で周期を持ったものが少ないことが実装中に判明した。例えば、月単位のデータは 1 年周期 (=12ヶ月周期) で階差を取ることが出来るが、週単位のデータは 1 年周期 (= 52.1429) で階差を取ることができない。週単位のデータを近似的に 1 年周期で階差を取る、つまり数年ごとに調整を入れる案も考え実験をしたものの、予測を行った際における調整した分の補完が困難を極めたため実装を中断した。

しかしながら季節階差を取る、という発想自体は間違いではない。具体的にどのような式に基づいて SARIMA モデルが作成されているかを  $SARIMA(1, 1, 1)(1, 1, 1)_{12}$  を用いて示す。尚説明の簡略化のため定数項は無視している。<sup>31</sup>

$$(1 - \phi_1 B)(1 - \phi'_1 B^{12})(1 - B)(1 - B^{12})Y_t = (1 + \theta_1 B)(1 + \theta'_1 B^{12})\epsilon_t \quad (29)$$

- $(1 - \phi_1 B)$   
非季節部分の AR(1) モデルの項
- $(1 - \phi'_1 B^{12})$   
季節部分の AR(1) モデルの項
- $(1 - B)$   
非季節部分の階差
- $(1 - B^{12})$   
季節部分の階差
- $(1 + \theta_1 B)$   
非季節部分の MA(1) モデルの項

---

<sup>31</sup>SARIMA(非季節部分のパラメータ  $p, d, q$ )(季節部分のパラメータ  $P, D, Q$ ) 季節階差のラグ

- $(1 + \theta_1' B^{12})$

季節部分の MA(1) モデルの項

この式からわかるように SARIMA モデルは ARIMA モデルに対して単純な乗算を行ったのみとなっている。これは ARIMA モデルのアプローチを大きく逸脱していないことを意味している。実際 ACF や PACF に関しては ARIMA モデルと同様に SARIMA モデルにもほとんど同じように適用することが出来る。<sup>32</sup>

## 6 Clojure/ClojureScript を用いた Web 開発

本演習では開発言語として Clojure/ClojureScript を取り上げた。この 2 言語は Java/JavaScript をカバーした言語であるが、Java/JavaScript とは違い 2 つの言語に大きな差はない。またこの言語は Web 開発に適した言語であり、その WebSocket の質などは極めて優秀であるという比較<sup>33</sup>もある。更に Lisp であるという特性から HTML に代表されるマークアップ言語、SQL などのドメイン特化言語を記述することが容易である。また 4 先述のように高速計算も可能であることから ARIMA モデルに関するライブラリ作成、OpenTSDB への接続、バックエンド、フロントエンドを実質一つのシンタックスで開発することができる。

この章では実際に Web 開発で用いたライブラリなどを簡潔に示す。

### 6.1 Clojure によるバックエンド開発

バックエンド開発は Luminus Framework を用いて行った。その際 Web API としての動作を確認するために Swagger UI を用いた。

#### 6.1.1 Luminus Framework

Luminus Framework とは Clojure の Web 開発に関するライブラリを統合したフレームワークである。ライブラリへのアクセスが極めて単純であり、それぞれのライブラリを好きなライブラリへ置換することも容易であるため、極めて自由度の高いフレームワークとなっている。そういった意味では Luminus はフレームワークと言うよりも推奨ライブラリのライブラリと言った方が近いのかも知れない。また様々なデータベースへのアクセスもサポート<sup>34</sup>されており、本演習では SQL として PostgreSQL を用いた。また Luminus には教科書とも言える書籍 [12] があり、これをチュートリアルとして活用することが出来る。

<sup>32</sup>但し季節階差を取ったものに対して ACF/PACF を示した後、ARIMA モデルで用いた意味での ACF/PACF を適用していくという順番を負う

<sup>33</sup>Websocket Shootout: Clojure, C++, Elixir, Go, NodeJS, and Ruby

<sup>34</sup>OpenTSDB への接続に関するサポートはなかったため、この部分の一部を自作する必要があった

### 6.1.2 Swagger UI

Swagger UI とは Swagger Tools と呼ばれる RESTful API を作成するための支援ツール群の一つであり、ブラウザから API の動作を確認することが出来る。本演習では Swagger Tools の一部である Swagger UI のみを用いているが、これは ① 関数型言語の特性の一つとして、データの源泉からデータの変換が終了するまでをひと続きに記述することが容易であるということから、サーバとクライアントのやり取りを行う部分のみを独立して記述することが効率的でないと判断したため ② Clojure の REPL<sup>35</sup> 上で開発を行っているためブラウザから API の確認を行う以上の高機能を求めているため ③ Luminus に含まれるライブラリである Compojure-api のサンプルでは Swagger UI を API の確認以上の目的で使用していないためである。

以下に実際に実装で用いた画面例を示す。

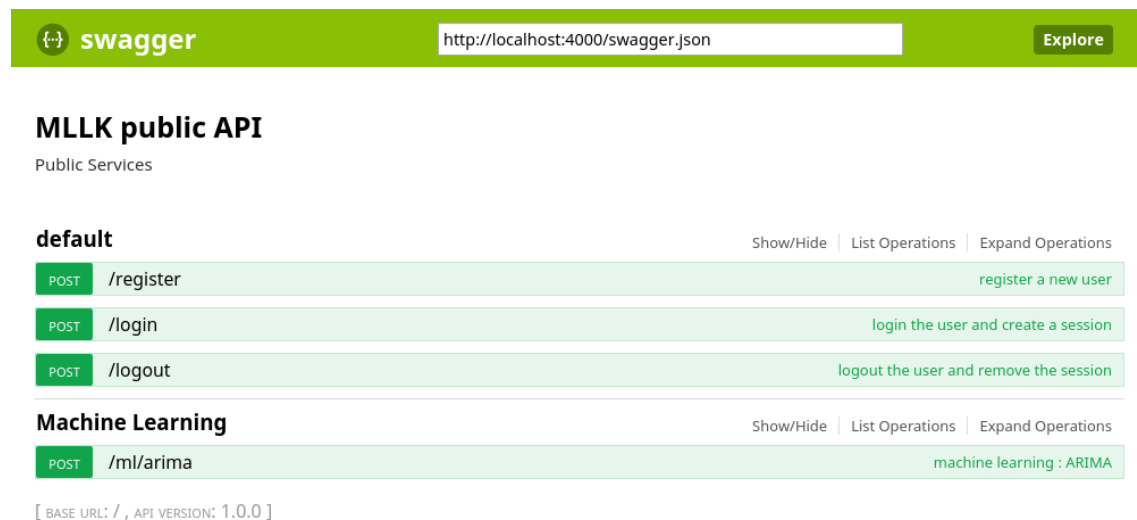



Figure 14: Swagger UI の利用例①

default という部分に含まれる POST メソッドは上から順に①ユーザ登録②ログイン認証③ログアウト、を意味している。これらの項目をクリックすることで必要なクエリやリクエストボディを入力することが出来る欄を開くことが出来る。開発者はここにテストとしてクエリを書き込むことでコードの妥当性を確かめることが出来る。

<sup>35</sup>Read-Eval-Print Loop 書いたコードを評価することでそれを反映することが出来る


**swagger**

Explore

## MLLK public API

Public Services

**default** Show/Hide List Operations Expand Operations

POST

/register

register a new user

**Response Class (Status 200)**

Model

Example Value

```

{
  "result": "string",
  "message": "string"
}

```

Response Content Type

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
<b>UserRegistration</b>	(required)		body	<div> <div>Model</div> <div>Example Value</div> </div> <pre> {   "name": "string",   "pass": "string",   "pass-confirm": "string" } </pre>

Parameter content type:

Try it out!

Figure 15: Swagger UI の利用例②

## 6.2 ClojureScript によるフロントエンド開発

フロントエンド開発に関しては Luminus Framework ある機能に加え、React.js の ClojureScript ラッパーである Reagent、React に特化された CSS フレームワークである Material UI を用いて作成した。

### 6.2.1 基本的な開発

Luminus Framework には Reagent を追加するオプションがあり、これを中心にフロントエンド側の開発を行った。またリクエスト処理などは Ajax を用いて実装した。

### 6.2.2 Reagent

Reagent は React.js の ClojureScript ラッパーである。React.js とは UI を作成するための JavaScript ライブラリであり、ボタンやスライダーなどのコンポーネントを簡単に作成することが出来る。Reagent では HTML との連携を非常に簡単に行えるようになっており非常に見やすいシンタックスを提供する。以下に実際に実装に用いた例を紹介する。

Listing 2: login-form 関数

```
1 (defn login-form []
2   (let [fields (atom {}) ;; 局所変数
3         error (atom nil)]
4     (fn [] ;; 描画時に呼び出される無名関数として定義
5       [c/modal ;; 自作関数であるモーダルウィンドウのコンポーネント呼び出し
6        [:div "MLLK Login"] ;; モーダルのタイトル <div> 要素として記述
7        [:div ;; モーダルの内容
8         [:div.well.well-sm ;; well well-sm は bootstrap のクラス
9          [:strong "* required fields"]]
10         [c/text-input "name" :name "enter a pass user name" fields]
11         ;; 自作のテキストフィールドの呼び出し
12         (when-let [error- (:name @error)]
13           [:div.alert.alert-danger error-]) ;; 名前に関するエラーを表示
14         [c/password-input "password" :pass "enter a password" fields]
15         (when-let [error- (:pass @error)]
16           [:div.alert.alert-danger error-])
17         (when-let [error- (:server-error @error)]
18           [:div.alert.alert-danger error-])]
19        [:div ;; <div><button type="button" class="btn btn-primary" ...>
20         ;; ...</button><button ...>...</button></div>
21        [:button.btn.btn-primary
22         {:on-click #(login! fields error)}]
```

```

23     ;; ボタンクリック時の動作を指定 (自作の関数 login! を呼び出している)
24     "Login"]
25     [:button.btn.btn-danger
26     {:on-click #(session/remove! :modal)}
27     "Cancel"]]])))

```

一見すると非常に入り組んだ構造に見えるかも知れないが、それは HTML を記述している中に JavaScript を直接埋め込んでいるような記述をしているためである。しかし HTML と JavaScript をまとめて記述することで どこに何があるのか という疑問を持たずに済む。また細かいパーツに関数として分割することが出来るため<sup>36</sup>、上例のようにログインフォームのみを独立して記述をすれば、全体として大きな構造が必要となったとしても小さな構造体に問題を分割することが出来る。

以下がこのソースコードを中心にして作成された login-form である。

Figure 16: login-form

<sup>36</sup>正確には関数が評価されるタイミングを考慮する必要がある

## 7 MLLK の開発

本演習では紆余曲折があったものの最終的には MLLK (Machine Learning Learning Kit) という Web アプリケーションのベータ版を開発した。完成まで導くことができなかった原因は、公開に足る性能を達成できなかったこと、アプリケーションの安全性が確立できなかったこと、データベースサーバなどの環境が特別であったため公開可能なサーバを調達できなかったことである。

### 7.1 ARIMA モデルライブラリ

5 ARIMA モデルによる時系列分析 に従ってそれぞれの関数を実装した。

### 7.2 データベース設計

以下の図に従ってデータベースを作成した。以降にそれぞれのテーブルについて説明する。

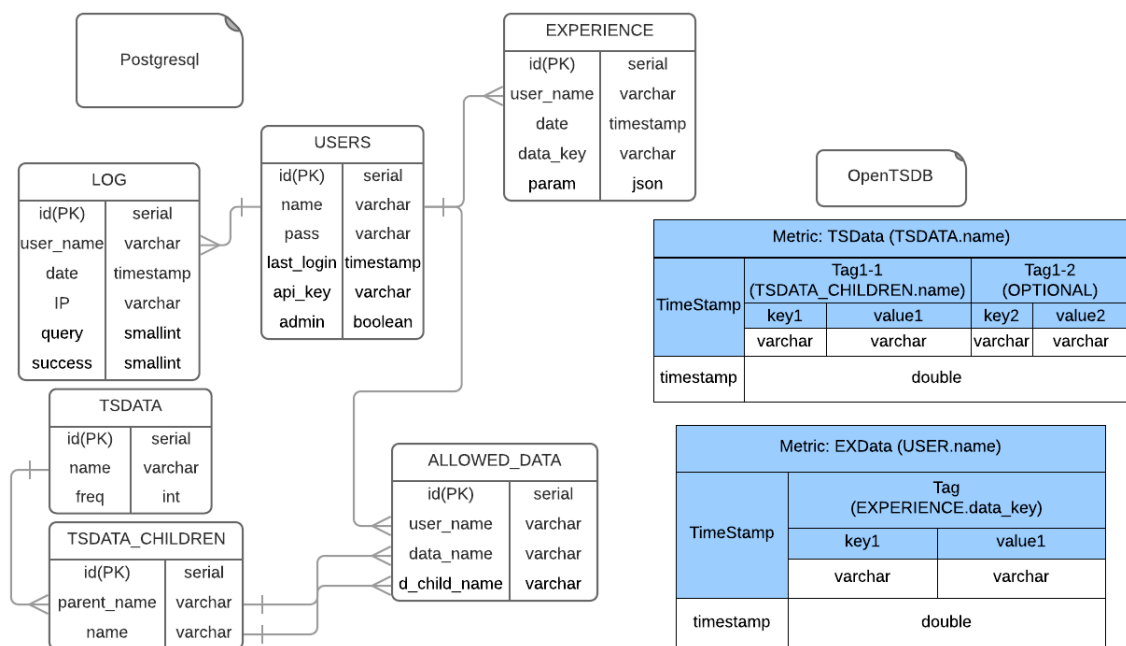


Figure 17: データベースの設計図



### 7.2.1 Postgresql

SQL サーバとしては Postgresql を採用した。

- **USERS**

ユーザに関するデータを保存している。name はユーザの名前、pass は暗号化済みのパスワード、last\_login は最後のログイン時間、api\_key は ユーザが持つ Quandl の API キーである。admin は管理者権限で本演習ではすべてのユーザが偽にセットされている。id は主キーであり、これは他のテーブルにも共通している。

- **LOG**

ユーザが行った情報をまとめるためのテーブルであるが、今回は実際にテーブルの作成のみで実際にテーブルを動かすことはできなかった。query と success はログインなどのクエリと対応付けられた 番号 が、date にはそのクエリが送られた日時が当てはまる。IP にはユーザがアクセスした端末の IP アドレスが当てはまる。

- **TSDATA**

例えば “NIKKEI/INDEX” のような時系列データの名称を保存するためのテーブルである。name はその時系列データの名称である。freq はデータの時間的頻度を指定しているが、ARIMA モデルの規格化などもあり今回はすべて週ごとを示す “weekly” がセットされている。

- **TSDATA\_CHILDREN**

“close-price” のような TSDATA テーブル で扱うべきではない細かいデータ区分を保存するためのテーブルである。name はその区分を示す名前である。ここにある name と parent\_name を用いて OpenTSDB の Metric TSDData に保存された時系列データへのアクセスを行う。<sup>37</sup>

- **ALLOWED\_DATA**

ユーザごとに扱えるデータに制限をかけるためのテーブルである。これは Quandl が API キーを登録せずともある上限まではデータを入手できるという機能に即するために作成した。

- **EXPERIENCE**

ユーザの実験データを保存するが、本演習では機能の実装が完了できなかったためテーブルのみの実装になった。date は実行日時、data\_key は OpenTSDB の Metric EXData にある タグの value の値である。param には行った機械学習の種類やパラメータが含まれた JSON が当てはまる。

---

<sup>37</sup>TSDATA の Metric 名は TSDATA テーブルの name , タグの value が TSDATA\_CHILDREN テーブルの name に対応している

### 7.2.2 OpenTSDB

時系列データに関しては 5 ARIMA モデルによる時系列分析 にあるとおり、OpenTSDB を用いた。

- TSDData

実際の時系列データを保存するためのデータベースである。本演習ではデータの入手元は [Quandl.com](https://www.quandl.com/) のみであった。Metrics には “NIKKEI/INDEX” のような時系列データの名称が当てはまり、タグの key は本演習では混乱を避けるために “type” に統一されており、そしてタグの value は “close-price” などの時系列データ内のカテゴリが当てはめられている。

- EXData

本演習では実装が終わらず実際に稼働させることはできなかった。Metric には Postgresql にある USERS テーブルの name が当てはまり、タグの key には “key” が、value には Postgresql の EXPERIENCE の data\_key が当てはまる。

## 7.3 バックエンド開発

バックエンドは 6 にあるように、Luminus Framework を中心に開発を行った。作成した ARIMA モデル やそれに関するライブラリや OpenTSDB を用いるための簡易的な API などは別個依存関係に加えた。フロントエンドも含めた<sup>38</sup>具体的な依存関係は 開発を行っている [git@github:MokkeMeguru/MLLK](https://github.com/MokkeMeguru/MLLK) にある `project.clj` にある。

## 7.4 フロントエンド開発

フロントエンドの開発もバックエンド同様 Luminus を用いて行った。以下に実際の画面をユーザが利用する場面を想定して説明する。

まずユーザはホームページにアクセスし、ユーザ登録画面へと遷移する。

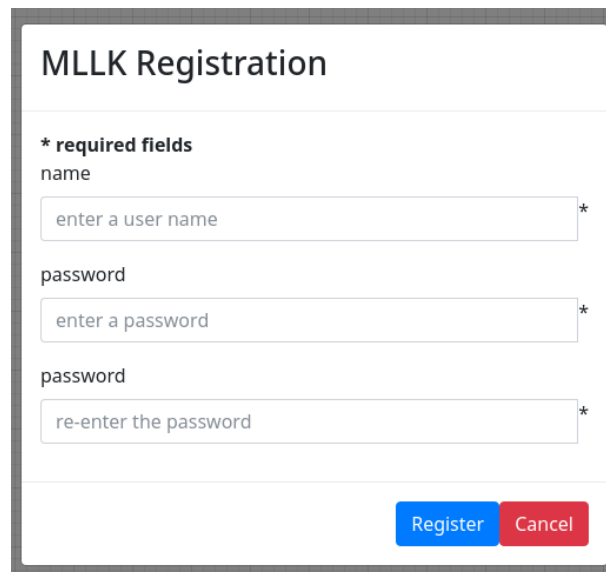
或いは既にユーザ登録が済んでいるユーザは名前とパスワードを入力しログインすることが出来る。この画面は既に 6.2 ClojureScript によるフロントエンド開発 で紹介済みであるため省略する。

次にユーザは実際に ARIMA モデルを (或いは開発が終了した場合にはそれらの手法を) 実行するためにこれらを行う画面へ遷移する。

ここで機械学習手法を青枠右側にあるメニューアイコンから呼び出し選択を行う。次に青枠左側にあるメニューアイコンを選択すると以下ようになる。

---

<sup>38</sup>バックエンドとフロントエンドは同一のプロジェクトとして作成されている



**MLLK Registration**

**\* required fields**

name  
 \*

password  
 \*

password  
 \*

[Register](#) [Cancel](#)

Figure 18: ユーザ登録画面

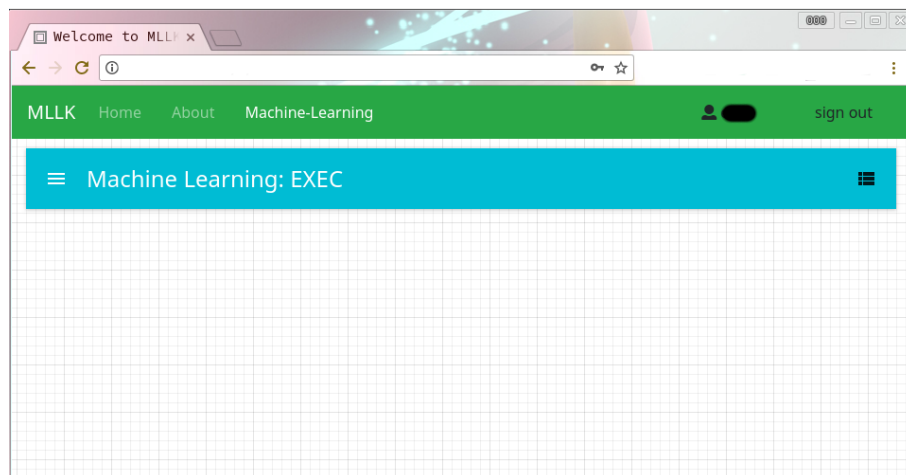


Figure 19: 統計・機械学習手法選択画面

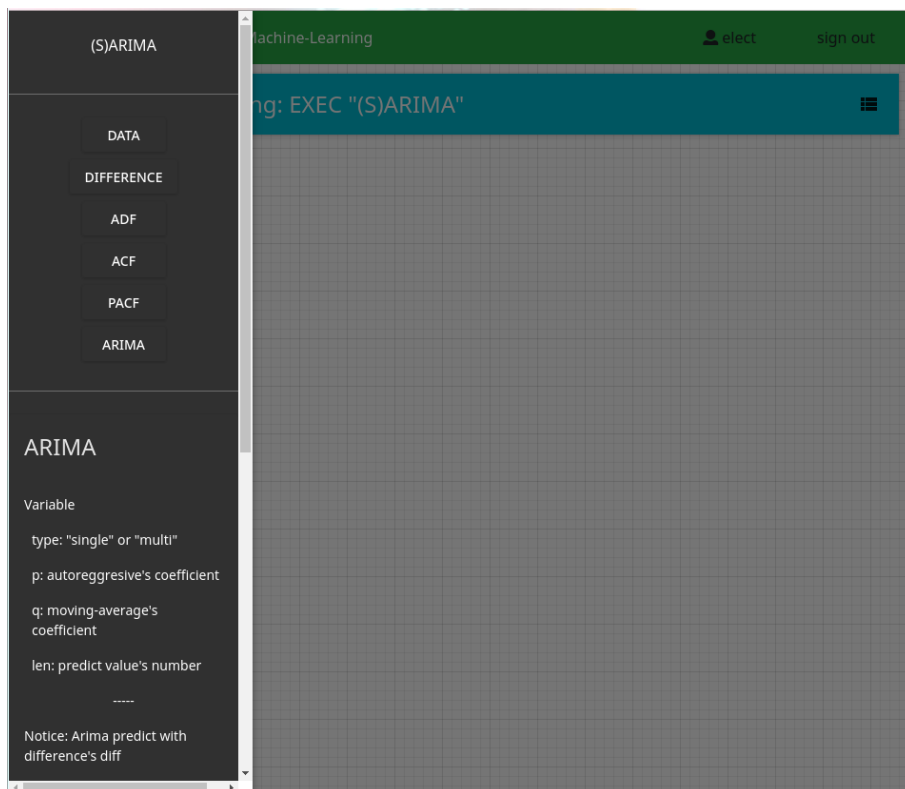


Figure 20: パラメータ設定画面

パラメータを選択する画面である。上にある“DATA”などと書かれているボタンはそれぞれのパラメータを入力する画面へ遷移する（現在は“ARIMA”ボタンを選択した状態）。はじめに大まかな説明がつき、その下に詳しい説明へのリンク（正確な理論が書き上がり次第組み込まれる）、そしてパラメータの入力欄が続いていく。ユーザはこれらの必要な項目の入力が済ませ実行ボタンである“run”ボタンを押す。

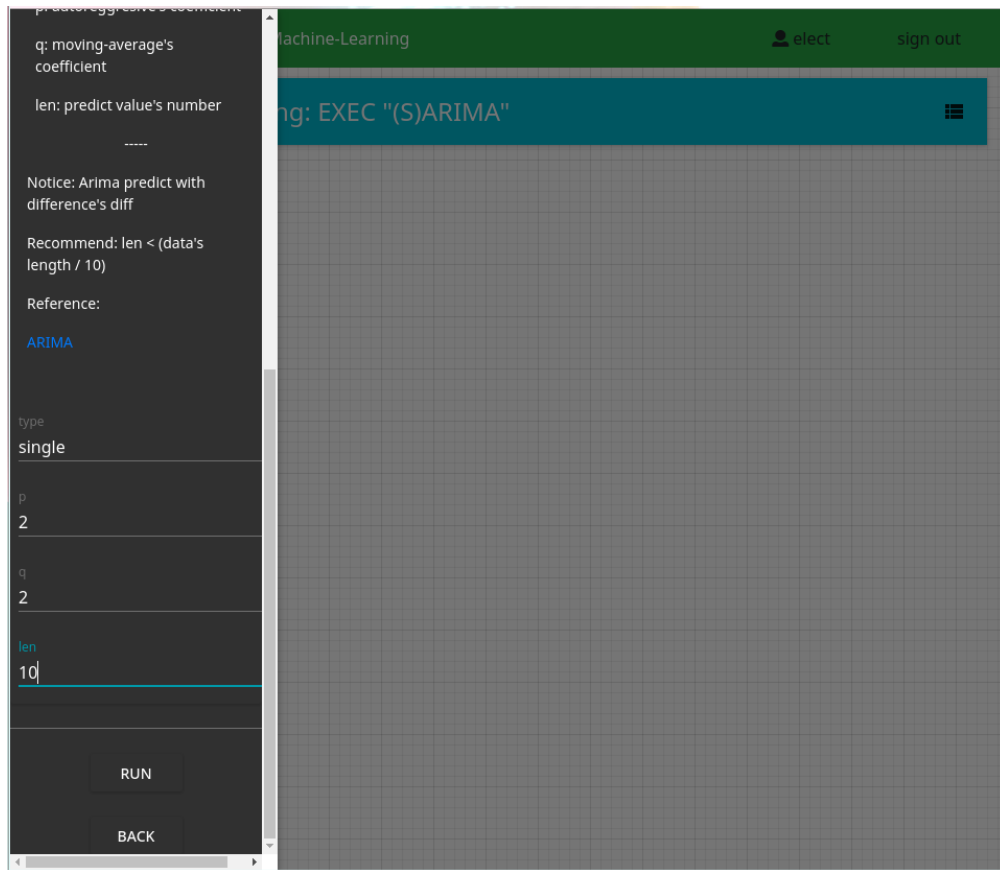


Figure 21: パラメータ入力欄と実行ボタン、説明ページへのリンク

実行することでユーザは以下のような画面を得る。

尚 2018 年 1 月 18 日時点 で出力可能な内容を印刷した pdf を 11 付録に追記する。

## 8 ARIMA 推定 と Random Forest による予測

この章では 2 序論で言及した (S)ARIMA モデルを用いた時系列データ予測と Random Forest を用いた欠損値補完を組み合わせた手法の検証実験について示す。

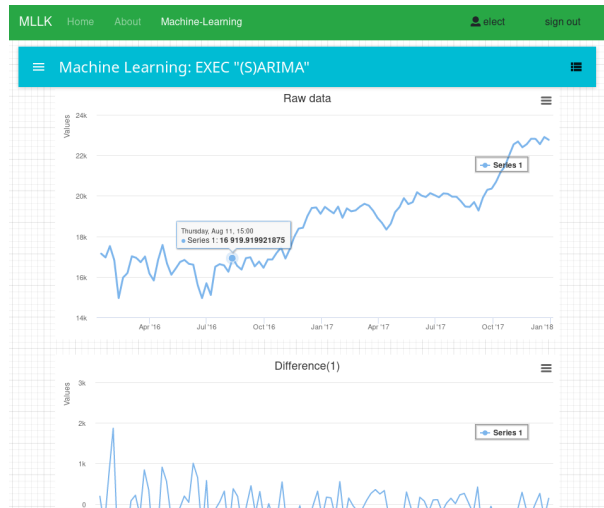


Figure 22: 実行結果画面

## 8.1 概要

ARIMA 推定は 単一の時系列データに対してモデル推定・予測を行うが、これにある予測 を 他のデータと関連させて行う、という立場から予測を行いたいと考えた。つまりこの実験の目的は複数の時系列データに対して、相関 を使って予測を行うというアプローチの妥当性について調査したい、ということである。

## 8.2 実験方法

Emacs IPython Notebook を用いて Python の既存である関数を用いて実験を行う。

実験データは Quandl から入手し、使用するデータは ① 日経平均株価 ② アサヒグループホールディングス ③ JR 東日本 ④ トヨタ自動車 ⑤ アステラス製薬 ⑥ ソニー の6つの終値である。

実験手順は 以下のとおりである。

- ② から ⑥ の 2015 年 1 月 から 2017 年 4 月 までの 週次 データに対して ARIMA モデルを適用し、2017 年 11 月頭までの予測を行う。
- ① から ⑥ における 2015 年 1 月 から 2017 年 4 月 までの時系列データに対して Random Forest を用いて回帰分析を行う。  
目的変数は ① であり、説明変数は ② から ⑥ である。
- 得られた Random Forest のモデルを用いて ① について 2017 年 11 月頭までの予測を行う。

4. 実際の ① の値と比較を行う。

5. Random Forest のみによる回帰分析、ARIMA モデル、SARIMA モデルのそれぞれを ① に適用し、3. と精度比較を行う。

### 8.3 実験結果

膨大な実験データになったため、付録に解説を含めて添付する。

### 8.4 考察

提案した手法は実験結果からわかるように良い精度を得られなかった。原因として考えられるものに、ARIMA モデルそのものが精度を必ずしも保証できないということ、説明変数の選択が不適切であったことが考えられる。

実際に実験中に用いた ARIMA モデルは予測精度に難があり、SARIMA モデルを使わなければならなかった。そしてその判定は AIC 等を用いて求める他に、それぞれの実行結果を目で確認する必要があるのではないかと考えている。季節階差などは ACF や PACF のグラフの閾値内にも隠れている可能性が十分にあることは今回の実験から明らかである。

説明変数の選択については、この実験は複数回実行されておりそれぞれ 4 ~ 5 個の説明変数を用いて行ったものの、この実験結果とほとんど変わらない精度であったことから、非常に難しい問題であると考えている。特に株価のような様々な要因が複雑に重なり合っ  
て初めて決定されると考えられるデータに対して最適な説明変数を選択することは、相当に株式について知見を持った観測者でなければ出来ないと考えている。但し、例えば仮想的に小さな世界を作成し、その中でのみ相互作用が起こる場合にこの手法を適用した場合にどのような結果が出るのか、という疑問が生まれた。この検証については今演習では時間や能力が足りず実験を行うことが出来なかったため、来年度以降の課題としたい。

## 9 まとめと今後の課題

本演習で学んだ内容は非常に多岐に渡っており、来年度以降の研究題材を豊富に得ることができたと考えている。

例えば本演習中に GPU 計算について学ぶ際に GPU を用いた開発の花形の一つとも言えるゲームエンジン・グラフィックエンジンの開発について触れる場面があった。現在一部のゲームを除いて殆どのゲームは C++ をベースに作成されているが、JVM 言語も GPU を用いた高速計算が可能である。このため Clojure を用いて高速なゲームエンジンを作成してみる、更にはそこに何かしらのゲーム AI を組み込んでみる、ということは需要については議論の余地があるとしても非常に興味深い内容になると考えている。

また機械学習について調べていく途中で RNN<sup>39</sup> を見る機会、黒板アーキテクチャ、当然のことながら Random Forest について学ぶ機会があり、様々な特徴を持ったデータベ

---

<sup>39</sup>Reccurent Neural Network

スについて触れる機会も得たため、これらを組み合わせてみた場合にどのようなことが出来るのかを検討・調査してみたいと考えている。

本年度の情報特別演習はこなさなければならない課題は多かったが、その分学ぶ部分も多く存在した。残念でならない点は、量が増えたためかそれぞれの成果がやや雑に纏まってしまったということである。来年度は本年度以上に課題を多く解くと共に、その課題一つ一つに真摯に取り組んでいこうと考えている。

## 10 謝辞

本演習においてはグループメンバーや担当教員である 筑波大学 システム情報系 情報工学域 日野 英逸 准教授に多大なご指導を頂いた。具体的には、グループメンバーには様々な課題を賜り充実した演習を行うことが出来、准教授には実験のみならずレポートの作成などについてもご指導を頂いた。協力に心からの感謝を申し上げます。

## 11 付録

### 11.1 このレポートにおける数式について

このレポートにおける数式表現といくつかの基本的な用語の定義を以下に例と共に示す。

#### 11.1.1 独立同時分布と変数の補足

$$Y_t = \beta t + \epsilon_t \text{ where } \epsilon_t \sim iid(0, \sigma^2) \quad (30)$$

この式における *where* とは左式における変数の補足を行うことを意味しており、この場合で  $\epsilon_t$  の意味を補足している。 $iid(\mu, \sigma^2)$  とは 独立同時分布 (independent and identically distributed) を意味しており、この確率変数は他の確率変数と同一の分布を持ち、且つそれぞれが独立していることを示している。独立同時分布において共分散、相関係数は 0 である。本レポートにおいてこの式は、平均  $\mu$ 、分散  $\sigma^2$  に従う<sup>40</sup> 独立同時分布という意味を持っている。

これと同様の概念にホワイトノイズというものがあるため、混乱を避けるためこちらも補足を行う。

ホワイトノイズ  $\epsilon_t \sim W.N(0, \sigma^2)$  は以下の性質を持っている。

- $E(E_t) = 0$   
平均は 0

---

<sup>40</sup> $a \sim b$  とは、 $a$  は  $b$  という分布に従う という意味である



- $Var(E_t) < infinity$   
分散は発散しない
- $Cor(E_t, E_s) = 0$   
相関 (correlation) 関係はない <sup>41</sup>

ホワイトノイズと独立同時分布の関係は、ホワイトノイズには必ずしも独立性があるわけではないという意味で、独立同時分布のほうがより“強固”であると言える。

### 11.1.2 標準誤差

$$\hat{\pi}/se(\hat{\pi}) \quad (31)$$

この式における  $se(\hat{\pi})$  とは  $\hat{\pi}$  についての標準誤差を示している。ここで用いる標準誤差とは最小二乗法で求められる解ベクトル  $\hat{\beta}$  の要素  $\hat{\beta}_i = \hat{\pi}$  についての誤差、最小二乗推定量  $\hat{\beta}_i$  に関する標準誤差である。これは回帰の標準誤差とは異なることに注意しなければならない。以下に回帰に関する標準誤差を示し、その後最小二乗推定量  $\hat{\beta}_i$  に関する標準誤差を示す。

例として以下の式を真のモデルとする。但し  $u$  は誤差項であり、平均 0 分散  $\sigma^2$  である独立同時分布に従う要素を持つベクトルであるとする。<sup>42</sup>

$$\begin{aligned} \mathbf{y} &= \mathbf{X}\hat{\beta} + \mathbf{u} \\ \text{where } \mathbf{y}, \mathbf{u} &\text{ is } n * 1 \text{ vector} \\ \mathbf{X} &\text{ is } n * p \text{ matrix} \\ \hat{\beta} &\text{ is } p * 1 \text{ vector} \\ \mathbf{u} &\sim iid(0, \sigma^2 \mathbf{I}) \\ \mathbf{I} &\text{ is } n * n \text{ identity matrix} \end{aligned} \quad (32)$$

推定量  $\hat{\beta}$  について求まる残差  $\epsilon$  は以下のように表すことが出来る。

$$\epsilon = \mathbf{y} - \mathbf{X}\hat{\beta} \quad (33)$$

これを用いて残差平方和 SSR<sup>43</sup> は、

$$\begin{aligned} SSR &= \epsilon' \epsilon \\ &= \mathbf{y}' \mathbf{y} - 2\hat{\beta}' \mathbf{X}' \mathbf{y} + \hat{\beta}' \mathbf{X}' \mathbf{X} \hat{\beta} \end{aligned} \quad (34)$$

<sup>41</sup> $r_{e_t, e_s}$  と表されることもある

<sup>42</sup> $\mathbf{X}, \hat{\beta}$  に定数項は含まれているものとする

<sup>43</sup>sum of squared residuals

と表すことが出来る。これを用いて回帰の標準誤差<sup>44</sup>は、

$$s^2 = SSR/(n - p) \quad (35)$$

となる。さらにこれを用いることで、最小二乗推定量  $\hat{\beta}_i$  に関する標準誤差は、

$$\begin{aligned} s.e.(\hat{\beta}_i) &= \sqrt{s^2/S_{XX}^j} \\ \text{where } S_{xx}^j &= ((X'X)^{-1})_{jj} \end{aligned} \quad (36)$$

となる。尚  $S_{xx}^j$  とは  $x_j$  から他の変数の影響を取り除いた  $x_j$  に固有の平方和である。

---

<sup>44</sup>standard error of regression

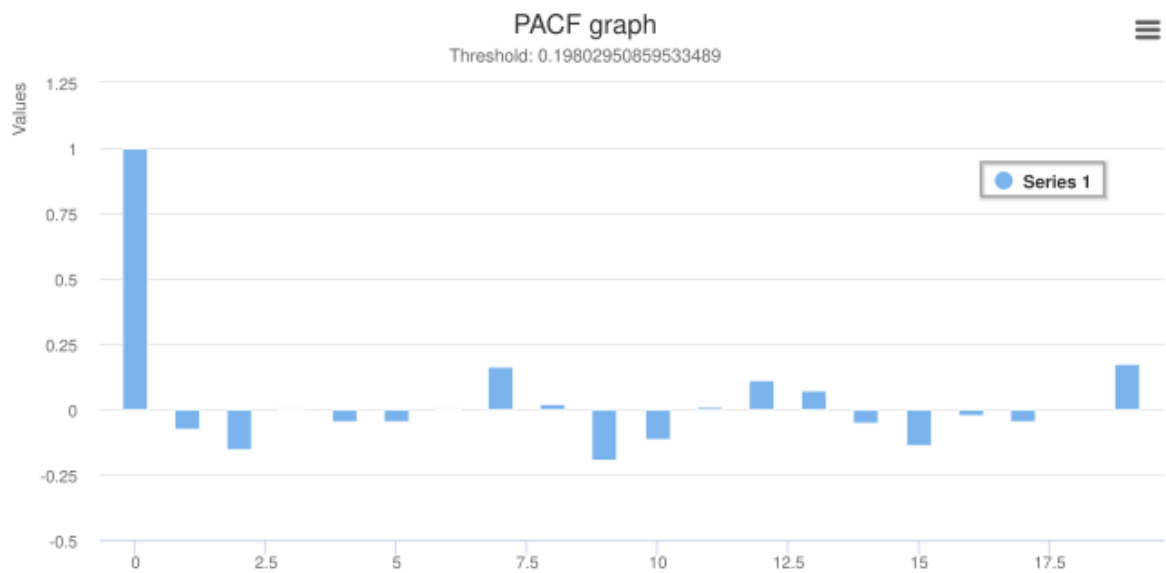
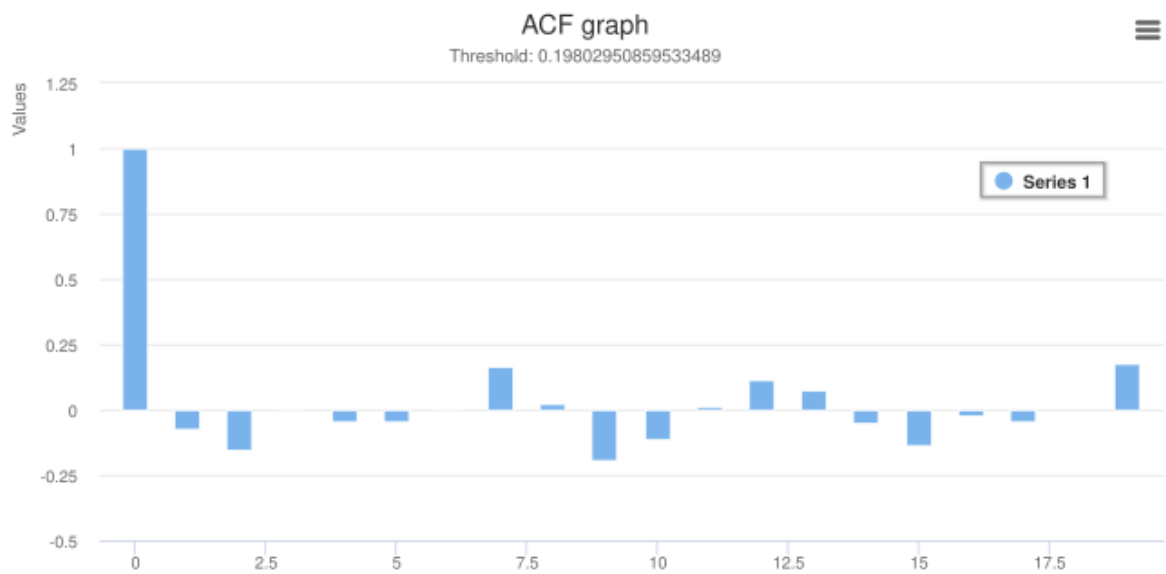
## 11.2 出力結果

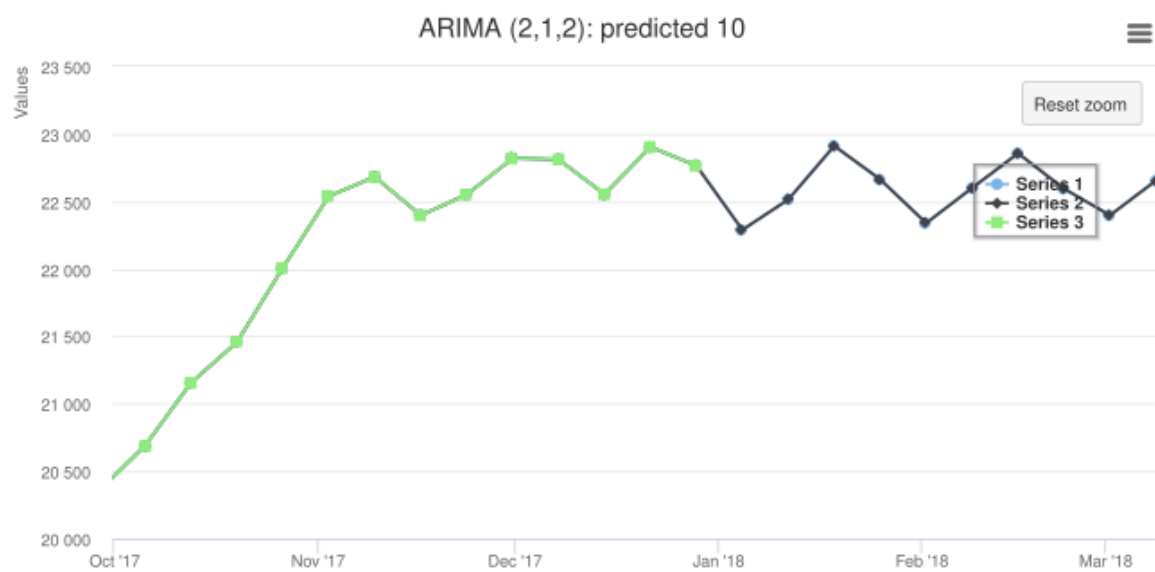


### ADF test with 1 difference

ADF value : -0.0692956621564448

Rejected? : False





ARIMA (2,1,2): predicted 10

p : [-0.1200378291226889 -0.8906776451180309]

q : [0.06819048033145769 0.9932921110318496]

aic : 1536.2608441051316

黒線とやや隠れてしまっているが青線の間がモンテカルロ法によって求めた<sup>45</sup>予測の95%信頼区間である。

<sup>45</sup>モデルのシミュレート を 1000 回行い、平均と標準偏差を求めている

### 11.3 実験結果

まず依存環境を解決する。

---

```
1 from pprint import pprint
2 import warnings
3 warnings.filterwarnings('ignore')
4 import numpy as np
5 import pandas as pd
6 from scipy import stats
7 import quandl
8 from sklearn.ensemble import RandomForestRegressor
9 import statsmodels.api as sm
10 from statsmodels.tsa.arima_model import ARIMA, ARMA
11 from matplotlib import pylab as plt
12
13 %matplotlib inline
```

---

次に、データを入手するための関数を定義する。データは 2015 年 1 月から 2017 年 11 月頭までの週次のものである。

---

```
1 def get_dataframe (name) :
2     return quandl.get(name, start_date='2015-1-1', end_date='2017-11-1',
3                       collapse='weekly')
```

---

これを用いてデータを入手する。dataframe の後ろにつく数字は、8.2 実験方法に記載されている数字と一致している。

---

```
1 dataframe1 = get_dataframe("NIKKEI/INDEX.4")
2 dataframe2 = get_dataframe("TSE/2502.4")
3 dataframe3 = get_dataframe("TSE/9020.4")
4 dataframe4 = get_dataframe("TSE/7203.4")
5 dataframe5 = get_dataframe("TSE/4503.4")
6 dataframe6 = get_dataframe("TSE/6758.4")
```

---

以下は入手したデータの整形である。

---

```
1 ts1 = dataframe1['Close Price'].rename('Close')
2 ts2 = dataframe2['Close'][:120]
```

---

```

3 ts3 = dataframe3['Close'][:120]
4 ts4 = dataframe4['Close'][:120]
5 ts5 = dataframe5['Close'][:120]
6 ts6 = dataframe6['Close'][:120]

```

---

以降は ADF 検定を用いた定常性判定であり、最小となる階差の次数における分析結果が返される。返り値の 1 つめの要素と最後の Map の中身の値を比較して仮説検定が行われ、その際に用いられる次数、つまり最もふさわしい次数が 3 つめの要素である。今回は最も ADF の値が小さくなるように、定数と傾きに関する項を回帰に含めている。

```

1 pprint(sm.tsa.stattools.adfuller(ts2, regression='ctt', autolag = 'AIC'))
2 pprint(sm.tsa.stattools.adfuller(ts3, regression='ct', autolag = 'AIC'))
3 pprint(sm.tsa.stattools.adfuller(ts4, regression='ctt', autolag = 'AIC'))
4 pprint(sm.tsa.stattools.adfuller(ts5, regression='ct', autolag = 'AIC'))
5 pprint(sm.tsa.stattools.adfuller(ts6, regression='ctt', autolag = 'AIC'))

```

---

```

(-3.4243993894646616,
 0.13326170365782197,
 0,
 119,
 {'1%': -4.471237472469957,
  '10%': -3.584424431130831,
  '5%': -3.882969922547368},
 1307.5889250985836)
(-2.904793565308496,
 0.16067015581233307,
 0,
 119,
 {'1%': -4.036933565633866,
  '10%': -3.1490681814297643,
  '5%': -3.4480491338265407},
 1537.811468467928)
(-2.6037424086219305,
 0.5096036701299048,
 0,
 119,
 {'1%': -4.471237472469957,
  '10%': -3.584424431130831,
  '5%': -3.882969922547368},
 1449.4672932660785)
(-3.4174043810519117,
 0.04910564863187634,

```

```

0,
119,
{'1%': -4.036933565633866,
 '10%': -3.1490681814297643,
 '5%': -3.4480491338265407},
1143.8350636832258)
(-4.137776413232316,
 0.02100681603520161,
 1,
118,
{'1%': -4.472110860871852,
 '10%': -3.584692378310343,
 '5%': -3.883407308731662},
1337.068933005826)

```

次に ② から ⑥ の ARIMA モデルの係数  $p, q$  の値を、Python の関数を用いて半自動的に求める。

---

```

1 pprint(sm.tsa.arma_order_select_ic(ts2, ic='aic'))
2 pprint(sm.tsa.arma_order_select_ic(ts3, ic='aic'))
3 pprint(sm.tsa.arma_order_select_ic(ts4, ic='aic'))
4 pprint(sm.tsa.arma_order_select_ic(ts5, ic='aic'))
5 pprint(sm.tsa.arma_order_select_ic((ts6 - ts6.shift(1)).dropna(), ic='aic'))

```

---

```

{'aic':          0          1          2
0  1679.536657  1577.743635  1763.332522
1  1488.936634  1490.614433  1488.660757
2  1490.733981  1490.708809  1490.657758
3  1489.374765  1491.192556  1492.459610
4  1491.350072  1493.041001  1494.459551,
 'aic_min_order': (1, 2)}
{'aic':          0          1          2
0  1979.585644  1869.081432  1809.600127
1  1737.428598  1738.866964  1739.981652
2  1738.990991  1740.494877  1741.980017
3  1739.657859  1741.646606  1735.021824
4  1741.609854  1743.301390      NaN,
 'aic_min_order': (3, 2)}
{'aic':          0          1          2
0  2002.003998  1868.465319  1964.683360
1  1640.157010  1641.641261  1643.141242
2  1641.711249  1642.628899  1644.606973

```



```

3  1643.150901  1644.607821  1647.420408
4  1645.148941  1646.595370  1648.894668,
  'aic_min_order': (1, 0)}
{'aic':          0          1          2
0  1545.805878  1436.081643  1388.521781
1  1302.593986  1303.591920  1304.847993
2  1303.801149  1305.297028  1306.731610
3  1304.443329  1306.432100  1292.806527
4  1306.403755  1310.297566  1294.654617,
  'aic_min_order': (3, 2)}
{'aic':          0          1          2
0  1525.435271  1522.249768  1523.248768
1  1521.388158  1522.796172  1524.758002
2  1522.815016  1524.769149         NaN
3  1524.725640  1526.723993         NaN
4  1526.710486  1528.435723         NaN,
  'aic_min_order': (1, 0)}

```

ここまでで ARIMA モデルにおける  $p, d, q$  の係数が定まったため、それらを適用してモデルを作成する。尚 階差  $d$  が 0 である場合は、この条件においては同義となる ARMA モデルを適用している。

---

```

1 ts2_arima = ARMA(ts2, order = (1, 2)).fit()
2 ts3_arima = ARMA(ts3, order = (3, 2)).fit()
3 ts4_arima = ARMA(ts4, order = (1, 0)).fit()
4 ts5_arima = ARMA(ts5, order = (3, 2)).fit()
5 ts6_arima = ARIMA(ts6, order = (1, 1, 0)).fit()

```

---

次に Random Forest モデルを作成する。目的変数を

---

```

1 r_forest = RandomForestRegressor(n_estimators = 1000, criterion = 'mse',
2                                 random_state = 1, n_jobs = 1)
3 r_forest.fit(np.vstack((ts2.values,ts3.values,ts4.values,ts5.values,ts6.values)).T,
4              ts1[:120].values)

```

---

Random Forest のモデルを作成したので、ARIMA モデルを適用した ② から ⑥ を用いて予測値を作成する。

---

```

1 ts2_predict = ts2_arima.predict('2015-1-18','2017-11-5')
2 ts3_predict = ts3_arima.predict('2015-1-18','2017-11-5')

```

---

---

```

3 ts4_predict = ts4_arima.predict('2015-1-18', '2017-11-5')
4 ts5_predict = ts5_arima.predict('2015-1-18', '2017-11-5')
5 ts6_predict = ts6_arima.predict('2015-1-18', '2017-11-5')

```

---

必要な要素が揃ったため、ここで問題となる回帰予測をおこなった。

---

```

1 ts1_pred_with_RF = r_forest.predict(
2     np.vstack((ts2_predict, ts3_predict, ts4_predict, ts5_predict, ts6_predict)).T)
3 ts1_pred_with_RF_dataframe = \
4     pd.DataFrame({'Close': ts1_pred_with_RF.tolist()}, index = \
5         pd.DatetimeIndex(periods = 147, freq = 'W', start = '2015-1-18'))
6 plt.hold(True)
7 plt.plot(ts1_pred_with_RF_dataframe)
8 plt.plot(ts1)

```

---

[<matplotlib.lines.Line2D at 0x7fbda6504400>]



この図を見るとこの手法はあまり精度が出ていないことがわかる。

次に比較として、単純に ① に ARIMA モデルを適用した場合の時系列予測を行う。行っていることは ② 等と変わらないため解説は省く。

---

```

1 sm.tsa.stattools.adfuller(ts1[:120], regression='ctt', autolag = 'AIC')

```

---

```
(-2.8812138487142485,
0.3547039460728244,
0,
119,
{'1%': -4.471237472469957,
'10%': -3.584424431130831,
'5%': -3.882969922547368},
1624.5165654066914)
```

---

```
1 sm.tsa.arma_order_select_ic(ts1[:120], ic='aic')
```

---

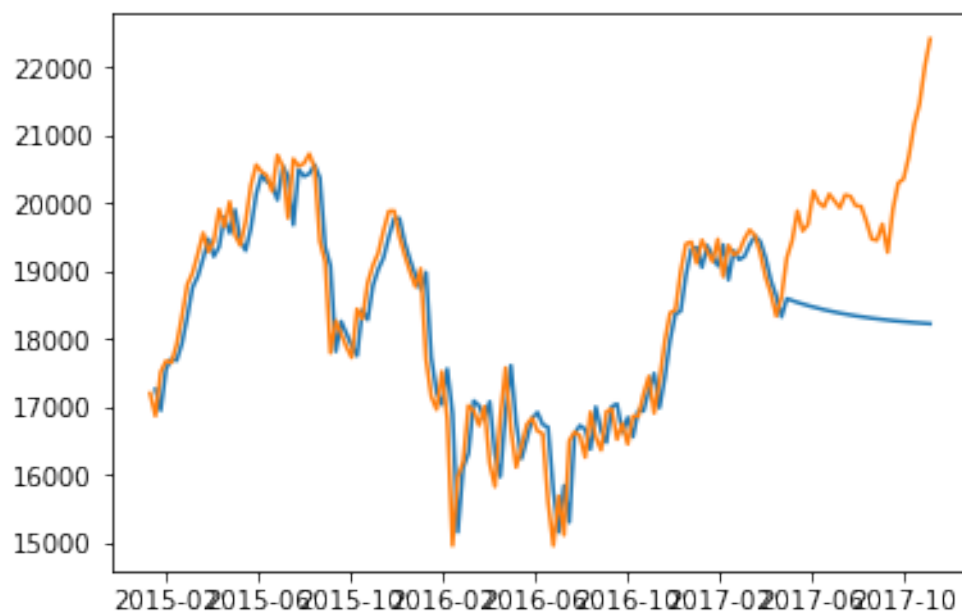
```
{'aic':          0          1          2
0  2096.943059  1986.989400  1930.176824
1  1837.682455  1839.576223  1841.481132
2  1839.583519  1841.552211  1843.455817
3  1841.446417  1843.432328  1843.027764
4  1843.368379          NaN  1845.121876, 'aic_min_order': (1, 0)}
```

---

```
1 ts1_arma = ARMA(ts1[:120], order = (1, 0)).fit()
2 ts1_pred_with_ARMA_dataframe = ts1_arma.predict('2015-1-18', '2017-11-5')
3 plt.hold(True)
4 plt.plot(ts1_pred_with_ARMA_dataframe)
5 plt.plot(ts1)
```

---

```
[<matplotlib.lines.Line2D at 0x7fbda60c4320>]
```



この図を見るとモデルは与えられたデータに対してかなり適合していることがわかるが、予測の段階で非常に大きな誤差が生まれていることがわかる。5.5.3 p, q の決定法 に書かれている規則を考えるとこのモデルには単位根が残っている可能性が考えられる。これについては次に行う Random Forest を用いた回帰予測とこれらのモデルの精度比較を行った後、SARIMA モデルと合わせて検討を行う。

同様に比較として Random Forest 回帰を用いた予測を行う。今回は ① における ARIMA モデル の係数 p の値に従い <sup>46</sup> 説明変数に過去の値の一つ取ってモデルを作成した。

---

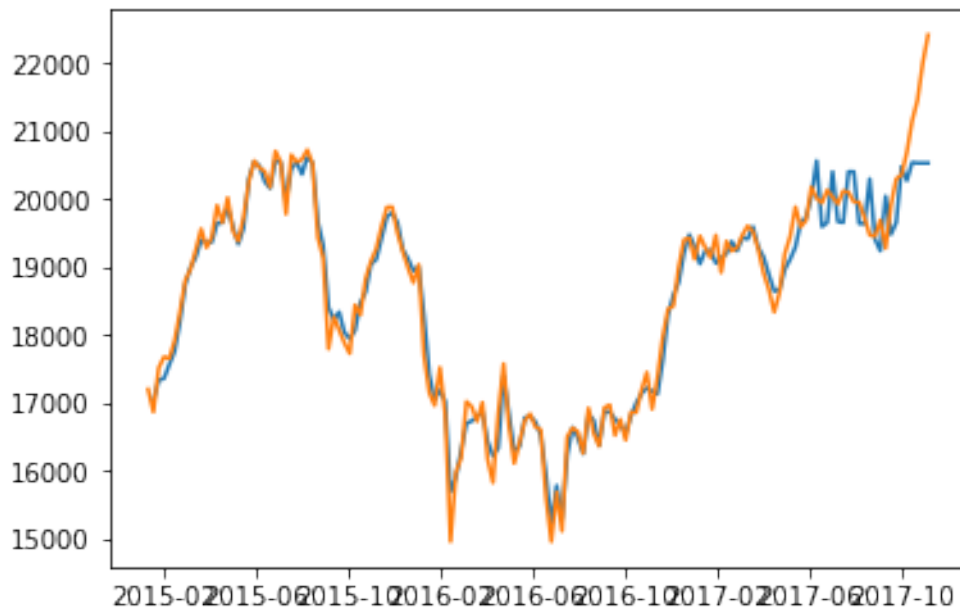
```
1 ts1_s1 = ts1.shift(1).dropna()[:120]
2 ts1_y = ts1[1:][:120]
3 r_forest2 = RandomForestRegressor(n_estimators = 1000, criterion = 'mse',
4                                   random_state = 1, n_jobs = 1)
5 r_forest.fit(np.vstack((ts1_s1.values)), ts1_y.values)
6 ts1_pred_with_RF2 = r_forest.predict(np.vstack((ts1.shift(1).dropna().values)))
7 ts1_pred_with_RF2_dataframe = \
8     pd.DataFrame({'Close': ts1_pred_with_RF2.T.tolist()},
9                  index = pd.DatetimeIndex(
10                      periods = 147,
11                      freq = 'W', start = '2015-1-18'))
12 plt.plot(ts1_pred_with_RF2_dataframe)
13 plt.plot(ts1)
```

---

[<matplotlib.lines.Line2D at 0x7fbda609c278>]

---

<sup>46</sup>少なくとも過去の値一つと比較した際に何らかの相関があると考えられるためである。



このグラフを見ると、ARIMA モデルを用いた場合に対してもより精度が良くなっているように見える。予測部分に関してもそれなりに近い動きを示しているが、やや不安定な動きを見せている。

次にモデルとの適合具合を調べるために MSE (二乗平均誤差)<sup>47</sup> を取る。

---

```

1 train_with_RF_resid = ts1[1:][:120] - ts1_pred_with_RF[:120]
2 train_with_ARMA_resid = ts1[1:][:120] - ts1_pred_with_ARMA_dataframe.tolist()[:120]
3 train_with_RF2_resid = ts1[1:][:120] - ts1_pred_with_RF2[:120]
4 MSE_train_RF = np.array([(elem * elem) for elem in train_with_RF_resid]).mean()
5 MSE_train_ARMA = np.array([(elem * elem) for elem in train_with_ARMA_resid]).mean()
6 MSE_train_RF2 = np.array([(elem * elem) for elem in train_with_RF2_resid]).mean()
7 print('train: MSE')
8 print('RandomForest and ARIMA model: ', MSE_train_RF)
9 print('      AR(I)MA model          : ', MSE_train_ARMA)
10 print('      Random Forest           : ', MSE_train_RF2)

```

---

```

train: MSE
RandomForest and ARIMA model: 1210139.138746533
AR(I)MA model : 247038.67430585716
Random Forest : 42615.21572554316

```

---

<sup>47</sup>Mean Squared Error

この結果から、提案した手法が元データを学習する面で良い精度を得られたなかったことは明らかである。また、ARIMA モデルの精度が低いように見えるが、これは ARIMA モデルが元データに対してずれた結果を示しているためであると考えられる。実施に求めた時系列データの一つずらすことで以下の結果を得ることが出来る。

---

```

1 ts1_pred_with_ARMA_dataframe2 = ts1_arima.predict('2015-1-18', '2017-11-5')
2 train_with_ARMA_resid2 = ts1[:120] - ts1_pred_with_ARMA_dataframe2.tolist()[120:]
3 MSE_train_ARMA2 = np.array([(elem * elem) for elem in train_with_ARMA_resid2]).mean()
4 print('train: MSE')
5 print('      AR(I)MA model?      : ', MSE_train_ARMA2)

```

---

```

train: MSE
AR(I)MA model? : 8580.270341196769

```

この結果は Random Forest に比べてもかなり良い結果であることがわかる。  
次に予測に関する精度を比較する。こちらも同様に MSE を用いた比較を行っている。

---

```

1 pred_with_RF_resid = ts1[1:][120:] - ts1_pred_with_RF[120:]
2 pred_with_ARMA_resid = ts1[1:][120:] - ts1_pred_with_ARMA_dataframe.tolist()[120:]
3 pred_with_RF2_resid = ts1[1:][120:] - ts1_pred_with_RF2[120:]
4 MSE_pred_RF = np.array([(elem * elem) for elem in pred_with_RF_resid]).mean()
5 MSE_pred_ARMA = np.array([(elem * elem) for elem in pred_with_ARMA_resid]).mean()
6 MSE_pred_RF2 = np.array([(elem * elem) for elem in pred_with_RF2_resid]).mean()
7 print('predict: MSE')
8 print('RandomForest and ARIMA model: ', MSE_pred_RF)
9 print('      AR(I)MA model      : ', MSE_pred_ARMA)
10 print('      Random Forest      : ', MSE_pred_RF2)
11
12 pred_with_ARMA_resid2 = ts1[1:][120:] - ts1_pred_with_ARMA_dataframe2.tolist()[120:]
13 MSE_pred_ARMA2 = np.array([(elem * elem) for elem in pred_with_ARMA_resid2]).mean()
14 print('      AR(I)MA model?      : ', MSE_pred_ARMA2)

```

---

```

predict: MSE
RandomForest and ARIMA model: 7288573.59721489
AR(I)MA model : 4002781.362739798
Random Forest : 415257.4516347963
AR(I)MA model? : 4002781.362739798

```

この結果からはそれぞれの図で確認したとおり、Random Forest が最も良い精度を持っていることがわかる。また ARIMA モデルは、モデルの元データとの適合度においては求まったデータをずらすことで精度が大幅に向上したが、予測の観点ではずらしても良い

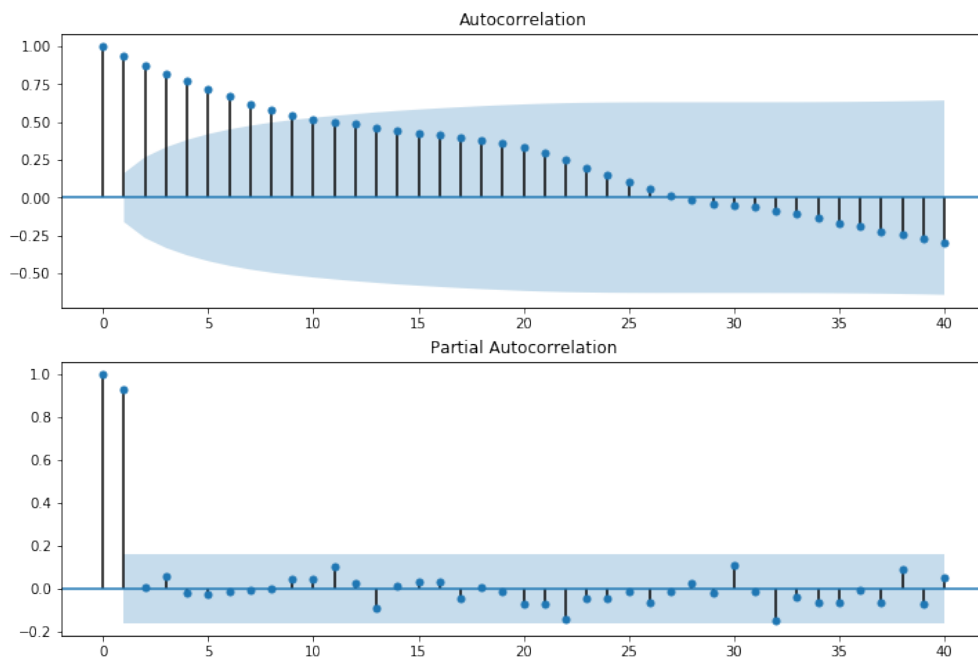
精度を得られていないことがわかる。

今回の実験における ARIMA モデルの精度について確認が出来たところで SARIMA モデルを用いた検討を行う。まず SARIMA モデルを適用するべきであるのかを確認するために ACF, PACF のグラフを確認する。

---

```
1 from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
2 fig = plt.figure(figsize = (12,8))
3 ax1 = fig.add_subplot(211)
4 fig = sm.graphics.tsa.plot_acf(ts1, lags=40, ax = ax1)
5 ax2 = fig.add_subplot(212)
6 fig = sm.graphics.tsa.plot_pacf(ts1, lags=40, ax = ax2)
```

---



この結果を注視すると、PACF のグラフに微かに周期性を見ることが出来る。よって 季節階差<sup>48</sup> を取ることが出来ると考えられる。

実際に SARIMA モデルを適用すると以下ようになる。尚 実験の結果<sup>49</sup>、このデータに対して季節成分の係数 (P,D,Q) は (1, 1, 0) を設定している。

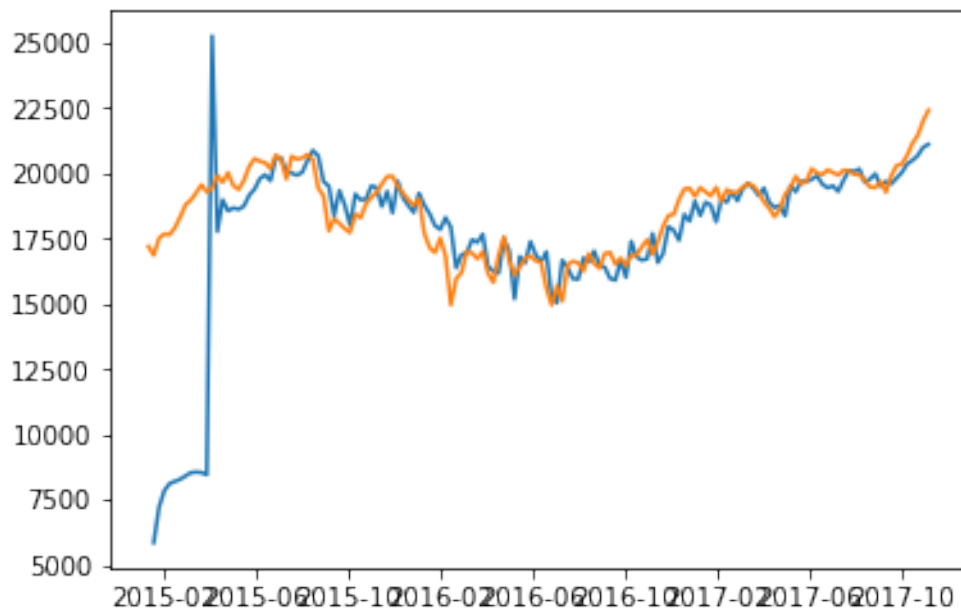
---

<sup>48</sup> 正確にはこのデータは週次ごとのデータなので、“季節” 差分を取っているわけではなく、周期性に基づいて階差を取っている。

<sup>49</sup> 様々なモデルを作成し、最も AIC の小さいモデルを選択した

```
1 ts1_SARIMA = sm.tsa.SARIMAX(ts1, order=(1,0,0), seasonal_order=(1,1,0,12)).fit()
2 ts1_pred_with_SARIMA_dataframe = ts1_SARIMA.predict('2015-1-18', '2017-11-5')
3
4 plt.hold(True)
5 plt.plot(ts1_pred_with_SARIMA_dataframe)
6 plt.plot(ts1)
```

[<matplotlib.lines.Line2D at 0x7fbda3f561d0>]



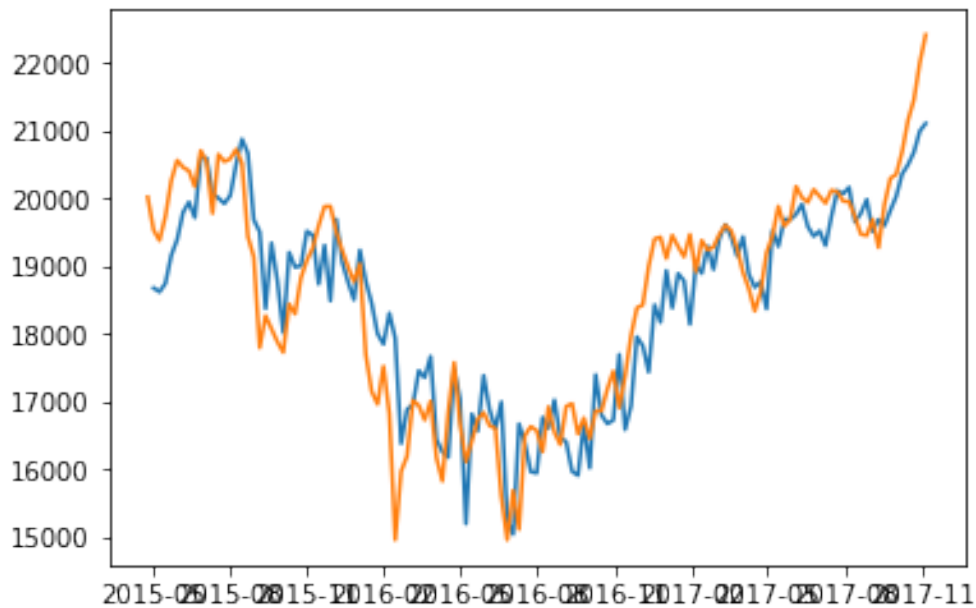
結果より時系列的に始めに近い部分の値が不安定ではあるが、それ以外の値に関しては元データに即した値を求めると考えられ、予測も比較的理想的に近い値を取っている。

また、不安定な部分を除いて表示を行うと以下ようになる。

```
1 plt.plot(ts1_pred_with_SARIMA_dataframe[15:])
2 plt.plot(ts1[15:])
```

[<matplotlib.lines.Line2D at 0x7fbda3ebd710>]





次にこの不安定な部分を取り除いた場合の SARIMA モデルについて MSE を求める。始めに近い値を取り除くことが出来ると考えた理由は、季節差分を取った際に初期のデータは比較される側のみであり、それ以降のデータとは扱いが異なるためである。

---

```

1 pred_with_SARIMA_resid2 = ts1[15:][1:][105:] - ts1_pred_with_SARIMA_dataframe.tolist()[15:]
2 MSE_pred_SARIMA = np.array([(elem * elem) for elem in pred_with_SARIMA_resid2]).mean()
3 print('train: MSE')
4 print('      SARIMA model?      : ', MSE_pred_SARIMA)
5 pred_with_SARIMA_resid2 = ts1[15:][1:][:105] - ts1_pred_with_SARIMA_dataframe.tolist()[15:]
6 MSE_pred_SARIMA = np.array([(elem * elem) for elem in pred_with_SARIMA_resid2]).mean()
7 print('predict: MSE')
8 print('      SARIMA model?      : ', MSE_pred_SARIMA)

```

---

```

train: MSE
SARIMA model? : 256194.51962546672
predict: MSE
SARIMA model? : 571242.1110044359

```

この結果から、少なくとも ARIMA モデルに比べると SARIMA モデルが精度面で高い性能を示していると考えることが出来る。

## References

- [1] 6.3.3 *Parameter Estimation*. URL: [http://www.maths.qmul.ac.uk/~bb/TimeSeries/TS\\_Chapter6\\_3\\_3.pdf](http://www.maths.qmul.ac.uk/~bb/TimeSeries/TS_Chapter6_3_3.pdf).
- [2] Jason Cantarella. *Nelder-Mead Method*. URL: <http://www.jasoncantarella.com/downloads/NelderMeadProof.pdf>.
- [3] Jason Dixon. *Monitoring with Graphite. Tracking Dynamic Host and Application Metrics at Scale*. Vol. 290. O'Reilly Media, 2017.
- [4] Henry Garner. *Clojure for Data Science*.
- [5] PhD Hedibert Freitas Lopes. URL: <http://hedibert.org/wp-content/uploads/2015/04/DT-or-ST.pdf>.
- [6] Daniel Higginbotham. *Clojure for the Brave and True. Learn the Ultimate Language and Become a Better Programmer*. No Starch Press.
- [7] Bart Hobijn, Philip Hans Franses, and Marius Ooms. “Generalizations of the KPSS-test for stationarity”. In: *Statistica Neerlandica* 58.4 (2004), pp. 483–502. ISSN: 1467-9574. DOI: 10.1111/j.1467-9574.2004.00272.x. URL: <http://dx.doi.org/10.1111/j.1467-9574.2004.00272.x>.
- [8] Shantau Kumar. *Clojure High Performance Programing*. Packt Publishing.
- [9] Robert Nau. *Identifying the numbers of AR or MA terms in an ARIMA model*. URL: <http://people.duke.edu/~rnau/411home.htm>.
- [10] George Athanasopoulos Rob J Hyndman. *Forecasting: principles and practice*. OTexts(October 17, 2013).
- [11] Matthew Scarpino. *OpenCL in Action. How to Accelerate Graphics and Computation*. Manning Pubns Co, Nov. 2011.
- [12] Dmitri Sotnikov. *Web Development with Clojure, Second Edition. Build Bulletproof Web Apps with Less Code*. Pragmatic Bookshelf.
- [13] *STAT 510*. URL: <https://onlinecourses.science.psu.edu/stat510/node/33>.
- [14] *The Apache HBase<sup>TM</sup> Reference Guide*. Revision 0.94.27. Copyright © 2012 Apache Software Foundation. Dec. 2015.
- [15] D. J. Thomson. “Jackknifing multiple-window spectra”. In: *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*. Vol. vi. Apr. 1994, VI/73–VI/76 vol.6. DOI: 10.1109/ICASSP.1994.389899.
- [16] カルマンフィルタの基礎. 2012. ISBN: 9784501960902. URL: <https://books.google.co.jp/books?id=A02GjwEACAAJ>.

Emacs 26.0.91 (Org mode 9.1.6)