

# 2019夏季インターン最終発表

## 2019/9/12(木)

同志社大学大学院  
理工学研究科 数理環境科学専攻 博士前期課程1年  
統計ファイナンス研究室  
林 大地

# 自己紹介

## 林 大地

同志社大学大学院 理工学研究科 数理環境科学専攻  
統計ファイナンス研究室

専攻：統計的学習理論，統計的機械学習の応用(異常検知)

使用言語: Python, R, Julia

- Keras, Pytorch

趣味: ジャズ，ピアノ曲を聴く，おいしいものを食べる，読書

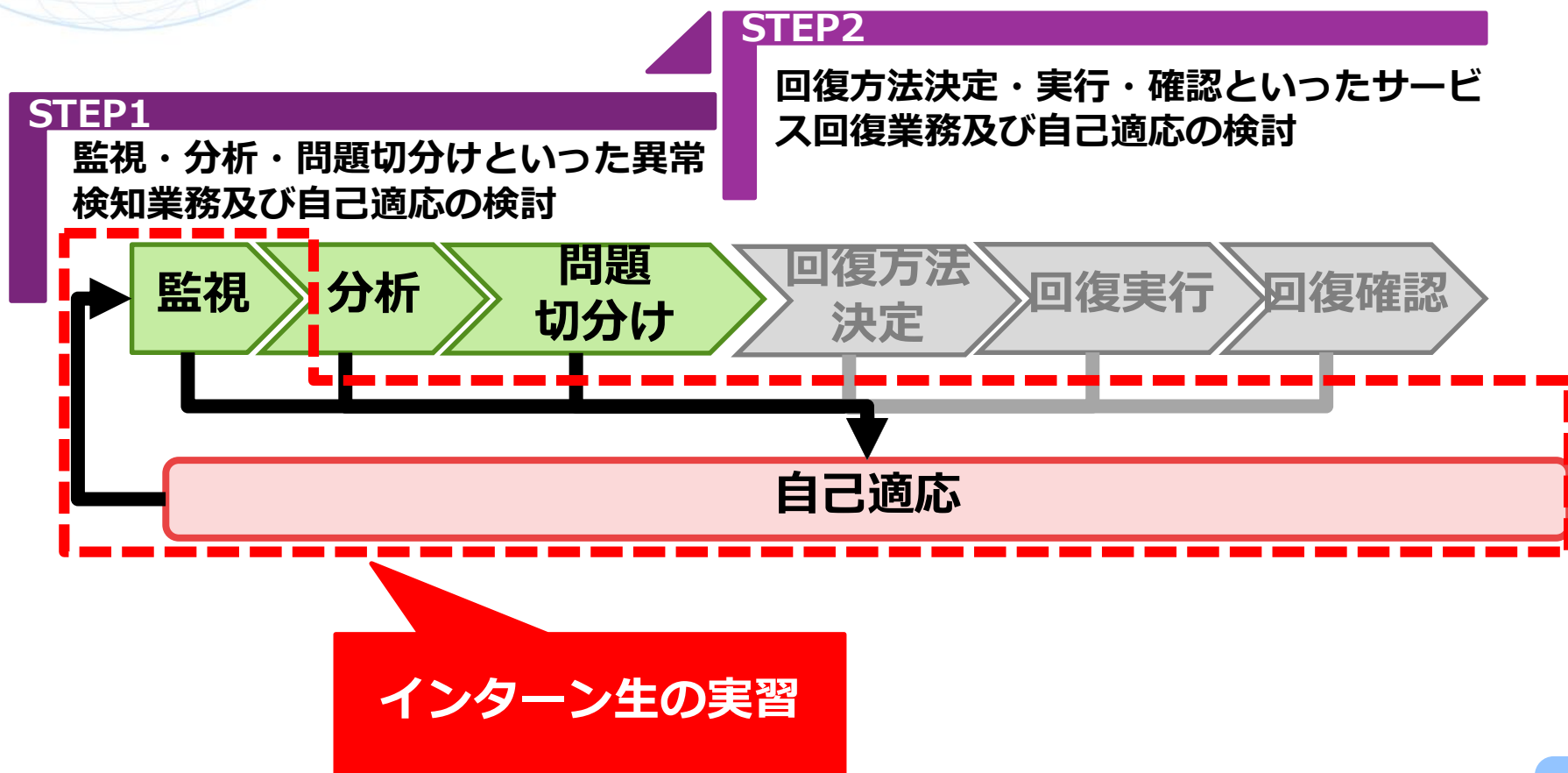
京都のRistという企業で学生インターンとしてDeep Learningや機械学習を用いた事業のお手伝いをしています。

(11月の勉強会(@京都)で発表者として登壇する予定です。資料もUPされるので，興味があれば見て下さい)

- 【実習内容】  
ネットワーク、クラウド、アプリケーション、デバイスをつなぐ環境下における障害検知手法の研究

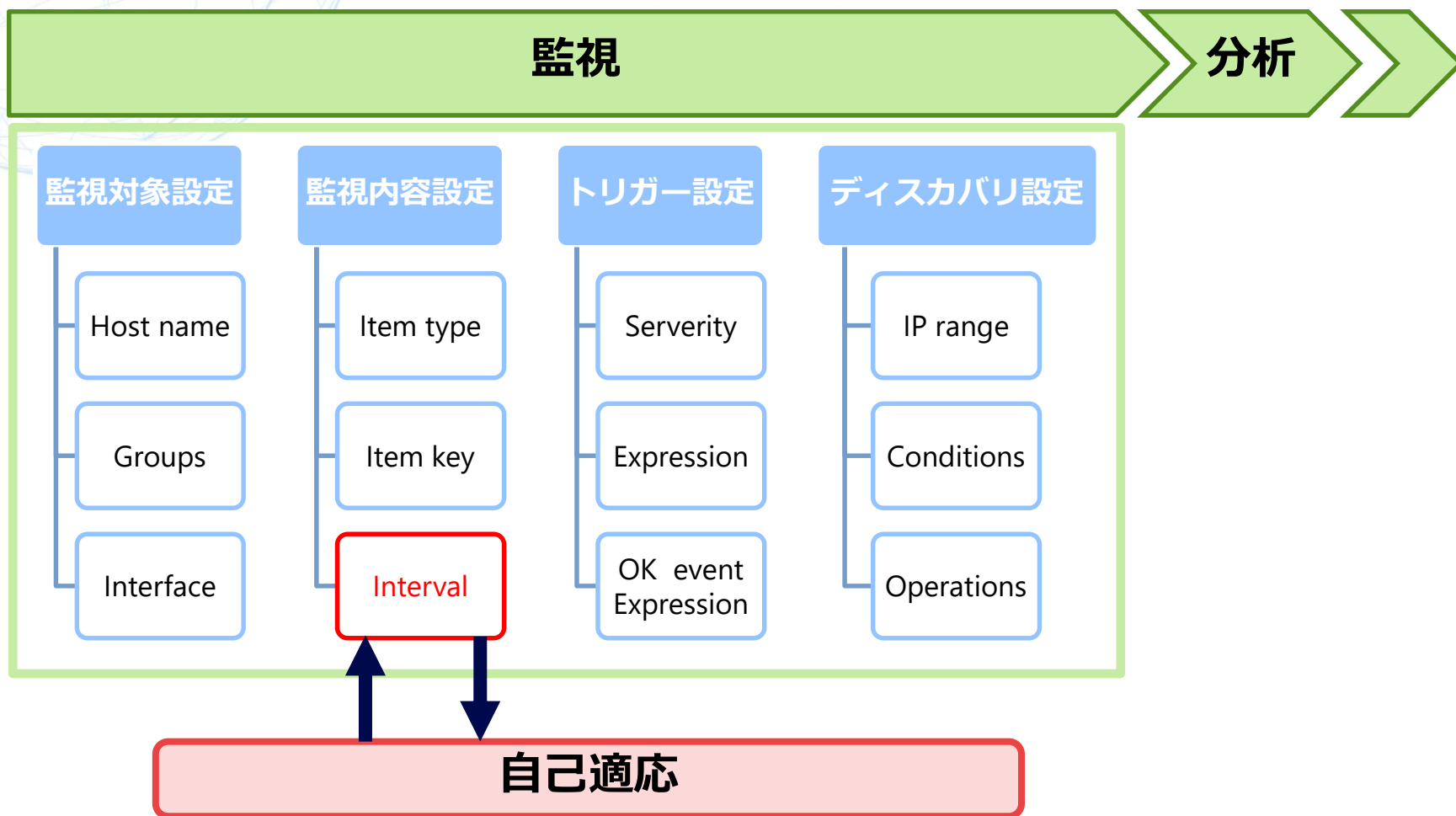
# インターン生実習の位置づけ

- ネ基Dでは現在、保全系技術高度化に注力している。
- 保全系技術高度化に属する研究テーマとして、保全プロセスに対する自己適応機能の検討に取り組んでいる。
- 本実習は、オペレーションプロセスに対する自己適応機能の検討の一環として行う。



# 取り組み概要

- サービス故障の検知・切り分けに役立つ監視を行うためには、様々な設定を適切に行う必要がある。
- 本実習では、その中でポーリング間隔(Interval)に対する自己適応機能を検討する。



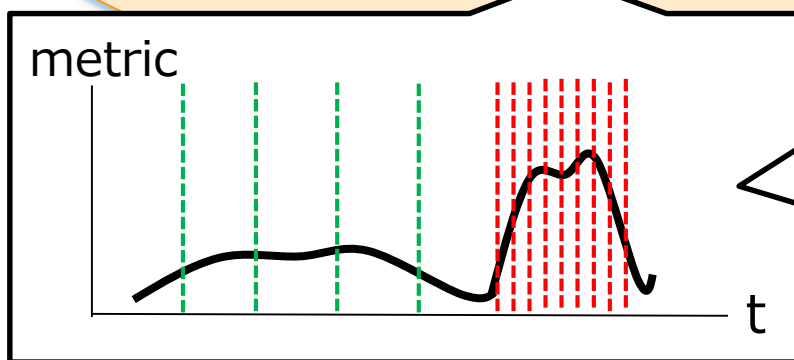
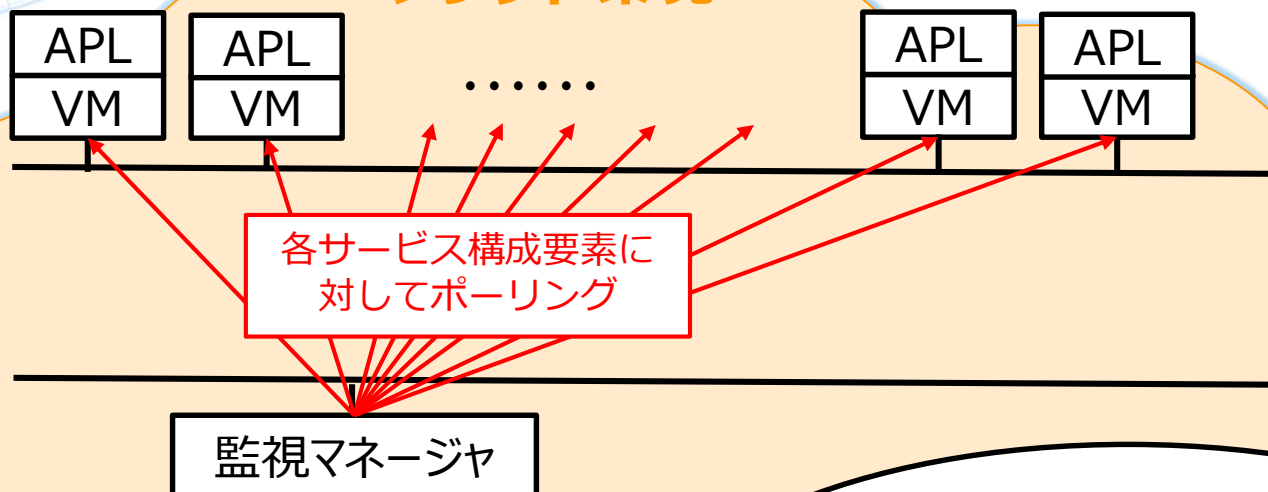
※設定内容の参考

Zabbix Documentation 4.4 : <https://www.zabbix.com/documentation/4.4/manual>

# 取り組みの目的

- マイクロサービス環境下では監視プロセスがサービスや監視マネージャに与える負担の大きさが問題となり得る。
- メトリクス取得の間隔をデータの変化に合わせて自動調節することで、人手による調整によらずに、監視の精度を保ちつつ、監視のオーバーヘッドを最小限に抑える。

## クラウド環境



### ■ 計測データの変化の激しさに応じて サンプリング間隔を動的に調整する

- 監視のオーバーヘッドの削減
- 変化のパターンをきめ細かく認識

# 今回の実習でやりたいこと

監視データの推移を見て、ポーリング間隔の自動調整

# インターンシップ中のスケジュール

- 1~2週目の実習内容
  - 論文読み解き
  - 論文にかかれたアルゴリズムの実装&実験
- ➡ 先行研究手法はノイズに対して弱い.
- 3~4週目の実習内容
  - 変化点検知手法と融合させる事による解決
  - オリジナル手法の提案&実装.
  - 資料作成



先行研究: 生データ + 変化点検知 + ポーリング間隔調整

林手法: 前処理 + 変化点検知 (new) + ポーリング間隔調整 (new)



# 先行研究の紹介

- 先行研究

- G. Tangari, D. Tuncer, M. Charalambides et al., "Self-Adaptive Decentralized Monitoring in Software-Defined Networks", IEEE Transactions on Network and Service Management , 2018

- 内容

- 今までの測定値の動きを見て，測定間隔を逐次自動調整する.
- 測定値の変化の度合いを測定し，変化が激しいときに測定の頻度を上げるといった考え方.
- これまでの手法の課題であった複数のハイパーパラメータのチューニングを最小限にとどめ，かつ，以前のモデルよりも良い精度を出すことに成功.

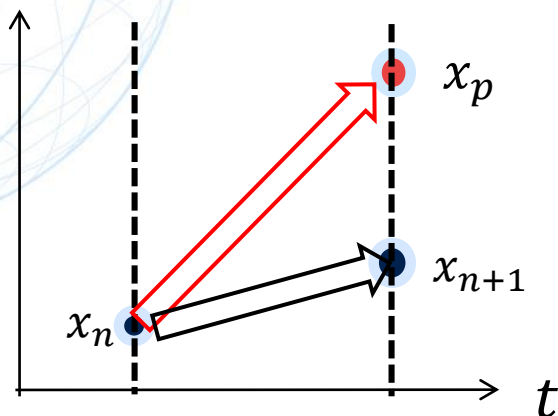


先行研究: 生データ + 変化点検知 + ポーリング間隔調整

林手法: 前処理 + 変化点検知 + ポーリング間隔調整  
(new) (new)

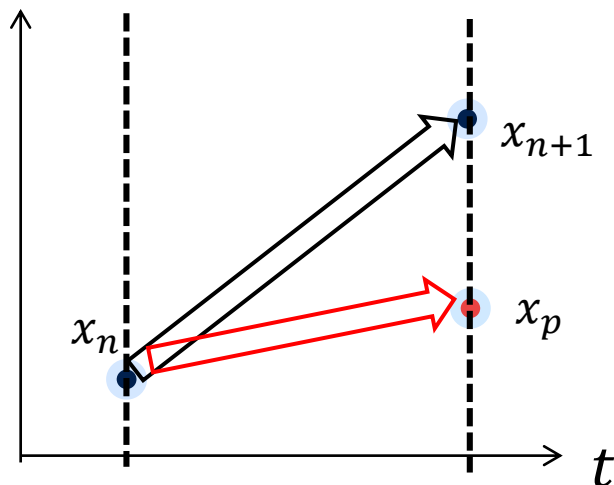
# どのような時にポーリング間隔は変化するか

- 予測よりもデータの変動が小さいとき
  - ポーリング間隔を延ばす



$x_n$  : 現在受け取った実測値  
 $x_p$  : 次回のポーリング時刻での予測値  
 $x_{n+1}$  : 予測した時刻での実測値

- 予測よりもデータの変動が大きいとき
  - ポーリング間隔を縮める



# 主要アルゴリズムの概要

- ・ **Step1** : 予測値  $x_p$  を以下の式で計算

$$x_p = x_n + \frac{t_n - t_{n-1}}{N - 1} \sum_{i=1}^{N-1} \frac{x_{i+1} - x_i}{t_{i+1} - t_i}$$

これまでのデータの変動から予測値を算出

$x_i$ :  $i$  番目にポーリングした時の実観測値  
 $N$ : ウインドサイズ(過去のデータを参照する数)  
 $t_i$ :  $i$  番目にポーリングした時の時刻

- ・ **Step2** : 実測値 ( $x_{n+1}$ ) を受け取り, 以下を計算

$$D = \frac{x_p - x_{n+1}}{x_{n+1} - x_n}$$

$D$  はポーリング間隔を調整するために用いる評価式

この  $D$  を用いて次のポーリング間隔  $T_{new}$  を更新する  
ただし、 $T_{old}$  は前回のポーリング間隔、 $T_{min}, T_{max}$  はそれぞれポーリング間隔の下限と上限

If  $D < 0$ :

$$T_{new} = \max\{T_{min}, T_{old} + D \cdot T_{old}\}$$

else:

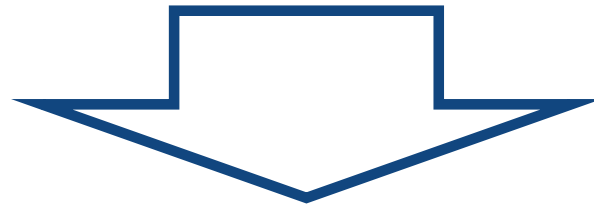
$$T_{new} = \min\{T_{max}, T_{old} + D \cdot T_{old}\}$$

さらに、以下で、ウインドサイズを更新する(AIMD アルゴリズム).

$$N = \begin{cases} N + 1 & T_{new} > T_{old} \\ \frac{N}{2} & (Otherwise) \end{cases}$$

# 論文を読んだだけでは分からないこと

- 論文著者が用いたデータが明示されていない...
- ポーリング間隔がどのように変移しているか明示されていない...



- やりたいこと
  - **先行手法が実データに対して，適用可能かどうかを試したい。**



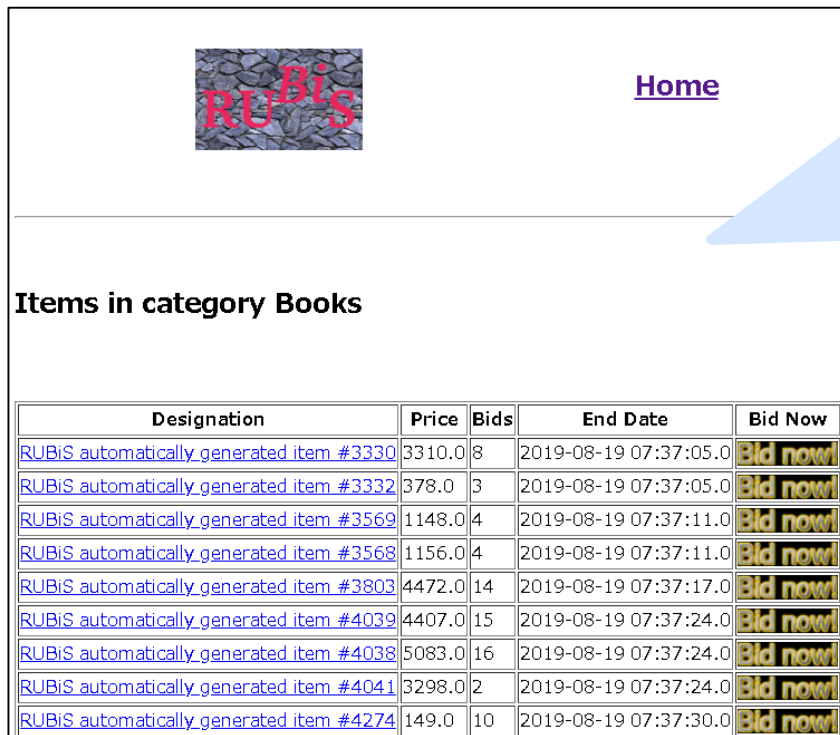
# 実データ作成&取得



# 題材アプリケーション

- RUBiS
  - <https://rubis.ow2.org/>
  - RUBiSはクラウド上での負荷分散や異常検知手法の実験台として、アカデミックな領域でよく使われているwebアプリケーション。

## RUBiS画面（カテゴリ別商品一覧）



The screenshot shows the RUBiS website interface. At the top left is the RUBiS logo, and at the top right is a 'Home' link. Below the header, the category 'Books' is displayed. A table lists items with columns for Designation, Price, Bids, End Date, and Bid Now. The table contains 10 rows of automatically generated items.

Designation	Price	Bids	End Date	Bid Now
<a href="#">RUBiS automatically generated item #3330</a>	3310.0	8	2019-08-19 07:37:05.0	<a href="#">Bid now!</a>
<a href="#">RUBiS automatically generated item #3332</a>	378.0	3	2019-08-19 07:37:05.0	<a href="#">Bid now!</a>
<a href="#">RUBiS automatically generated item #3569</a>	1148.0	4	2019-08-19 07:37:11.0	<a href="#">Bid now!</a>
<a href="#">RUBiS automatically generated item #3568</a>	1156.0	4	2019-08-19 07:37:11.0	<a href="#">Bid now!</a>
<a href="#">RUBiS automatically generated item #3803</a>	4472.0	14	2019-08-19 07:37:17.0	<a href="#">Bid now!</a>
<a href="#">RUBiS automatically generated item #4039</a>	4407.0	15	2019-08-19 07:37:24.0	<a href="#">Bid now!</a>
<a href="#">RUBiS automatically generated item #4038</a>	5083.0	16	2019-08-19 07:37:24.0	<a href="#">Bid now!</a>
<a href="#">RUBiS automatically generated item #4041</a>	3298.0	2	2019-08-19 07:37:24.0	<a href="#">Bid now!</a>
<a href="#">RUBiS automatically generated item #4274</a>	149.0	10	2019-08-19 07:37:30.0	<a href="#">Bid now!</a>

## RUBiS のサイトでできること

- ✓ ユーザ登録
- ✓ 商品の閲覧
- ✓ 商品への入札
- ✓ 商品の出品
- ✓ ユーザのコメント書き込み
- ✓ 商品のコメント書き込み
- • • など

# データ取得方法

 : dockerホスト

 : dockerコンテナ

時系列データ (5秒間隔)

- CPU使用率
- トラフィック量
- メモリ使用量



私たち

**AWS**

RUBiSに定常的な  
負荷を与える

Monitoring-Subnet

prometheus

grafana

監視サーバ

クライアントサーバ

rubisweb

WEBサーバ

rubisservlet

APサーバ

dockerfiles-  
rubisdb-01

DBサーバ

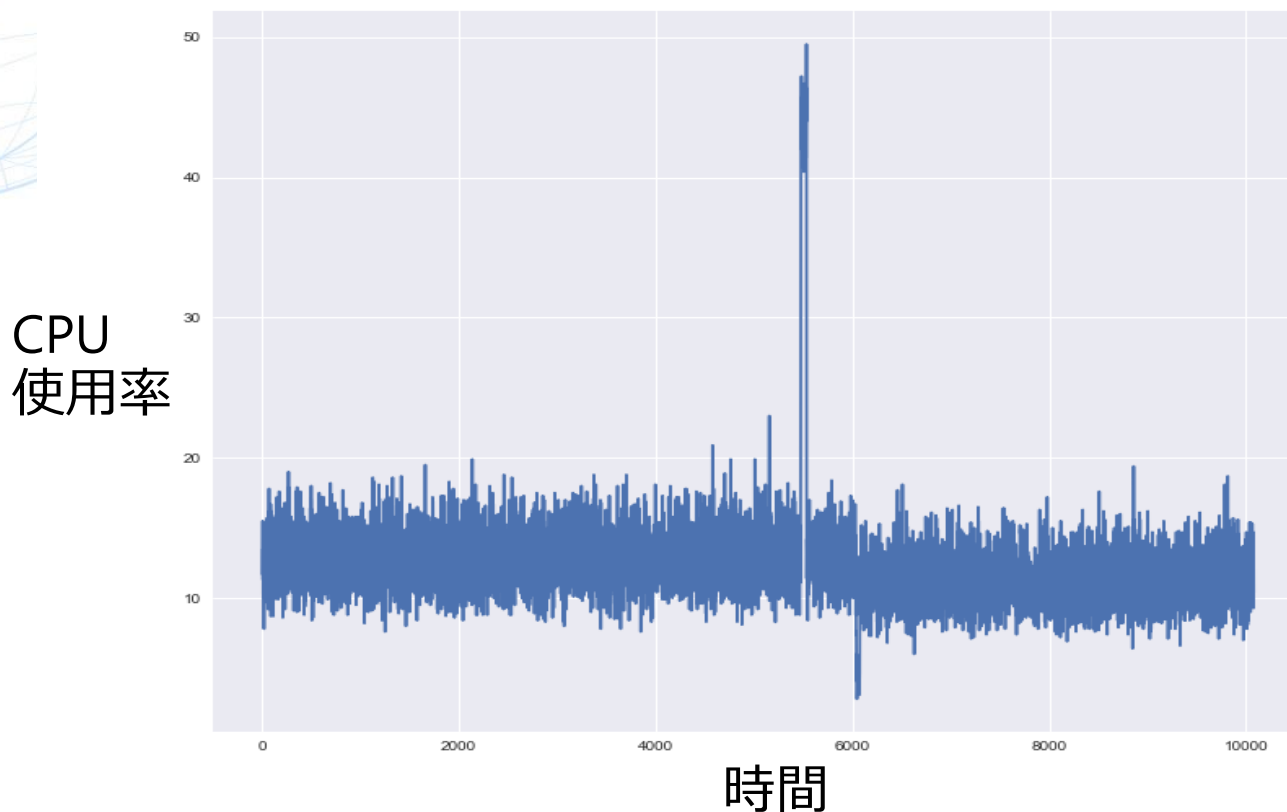
**RUBiS**

Frontend-Subnet

Backend-Subnet

# 生データの全体のプロット

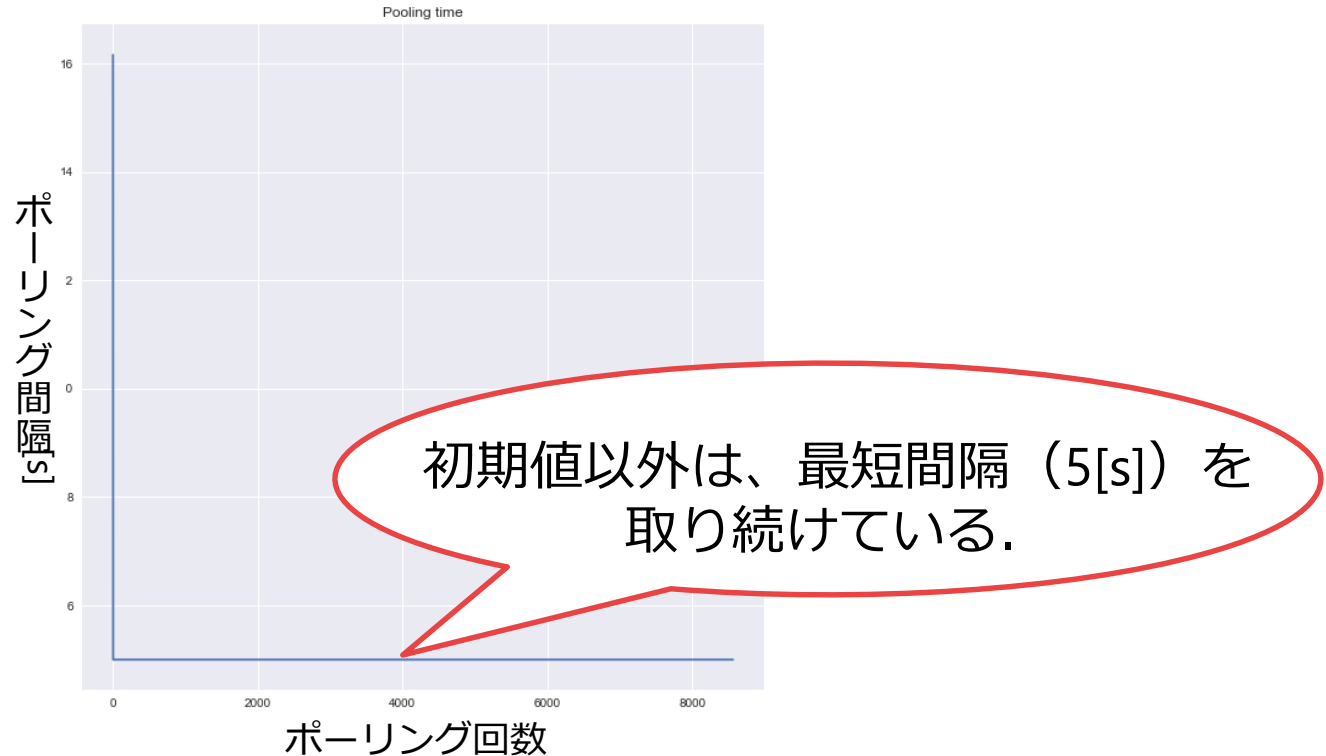
- 元の生データ全体をプロットしたのが以下の図である.



- 長期的なデータの変動は少ないことがわかる.

# 先行研究手法を適用した結果

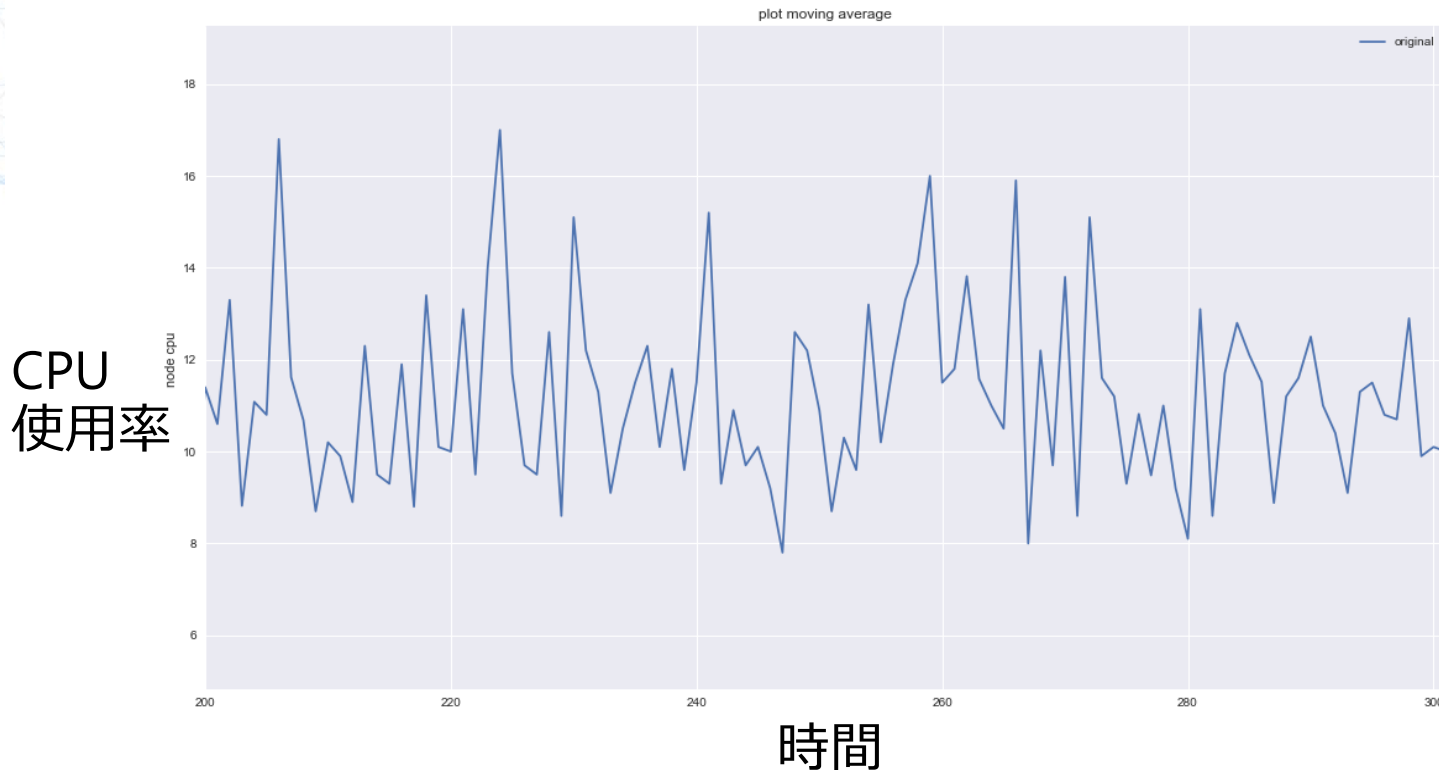
- 先行手法を適用したときのポーリング間隔の推移を図にしたのが以下の図である。



- 生データに対して動かしてみると、最短のポーリング間隔を取り続けている

# 生データを局所的に見てみる

- データを近くで見ると



- データの細かい増減に反応してサンプリング間隔を常に短くしようとしてしまう.



**平滑化が必要がある**

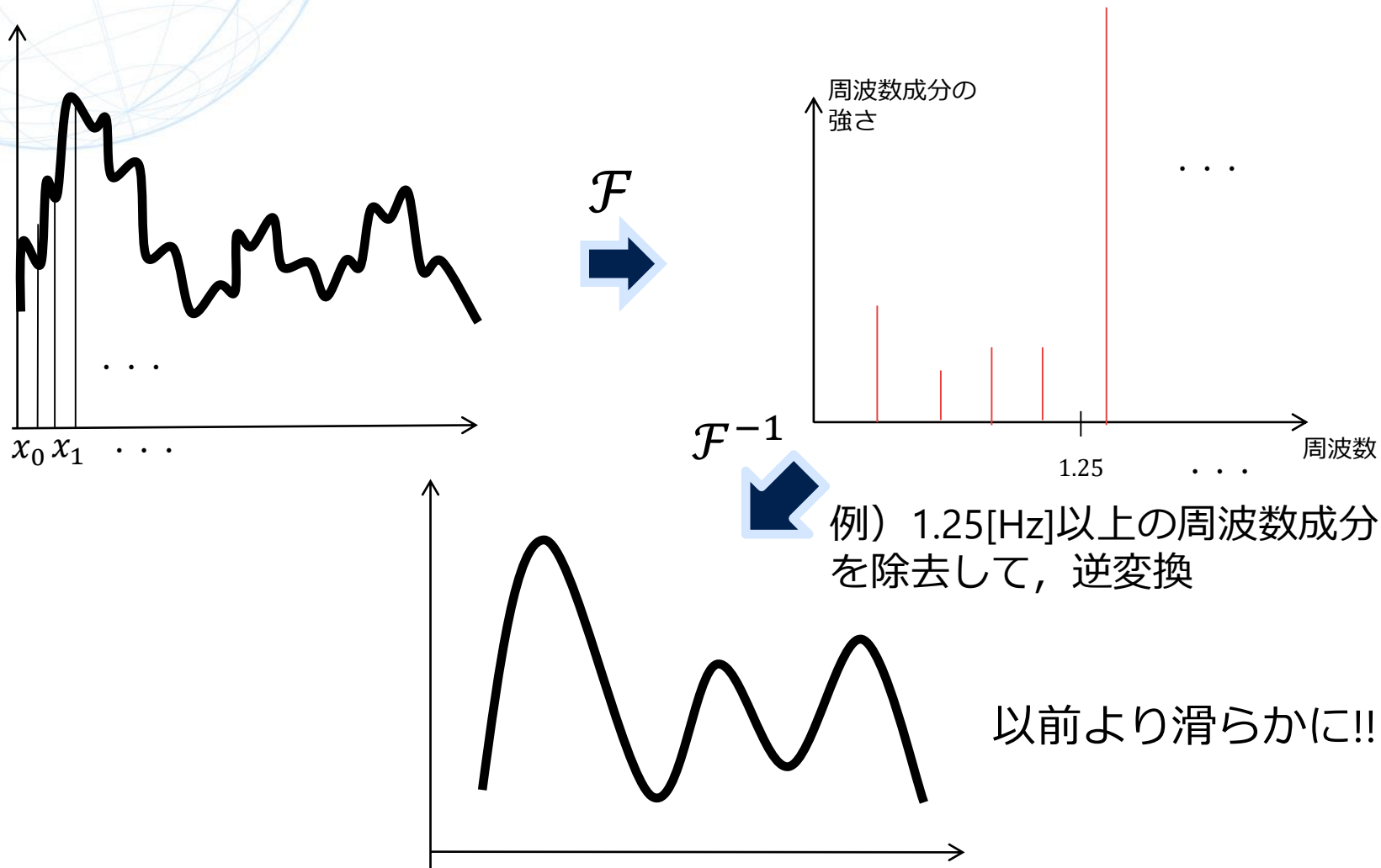


先行研究: 生データ + 変化点検知 + ポーリング間隔調整

林手法: 前処理 + 変化点検知 + ポーリング間隔調整  
(new) (new)

# 用いた平滑化手法

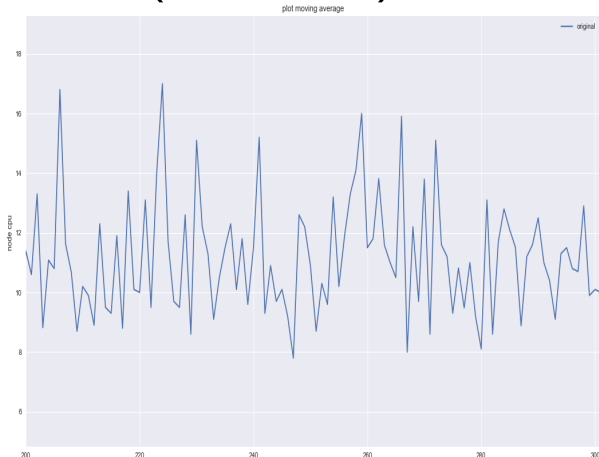
- ローパス フィルターによる平滑化
  - Fourier解析を用いる.



# 平滑化の結果

## 元データ(平滑化前)

CPU  
使用率

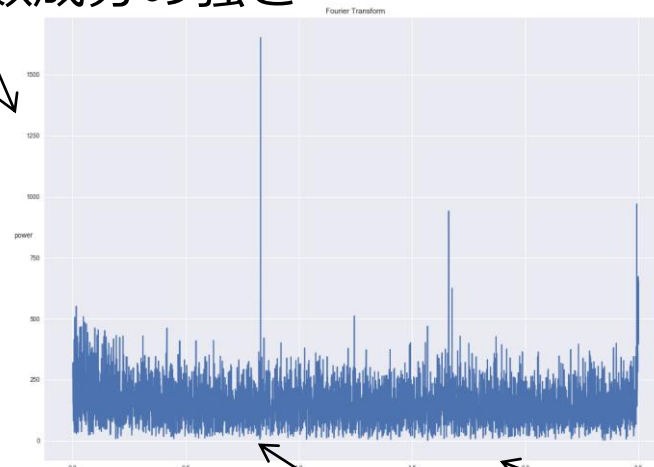


時間

各周波数成分の強さ

Fourier  
変換

$\mathcal{F}$



$\mathcal{F}^{-1}$



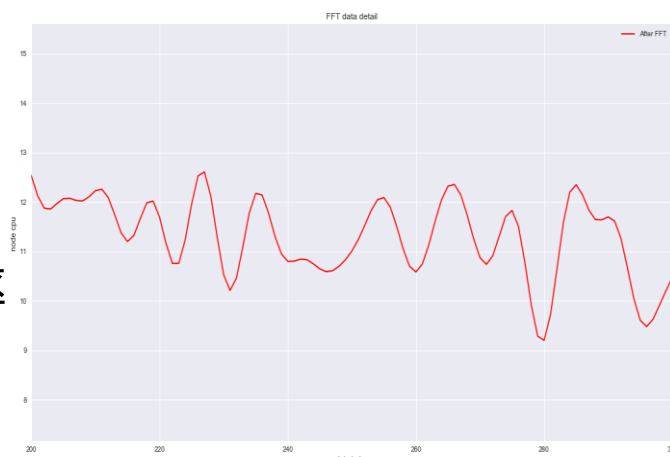
逆変換

0.7

周波数[Hz]

0.7以上の周波  
数成分を取り除  
いた

CPU  
使用率



時間

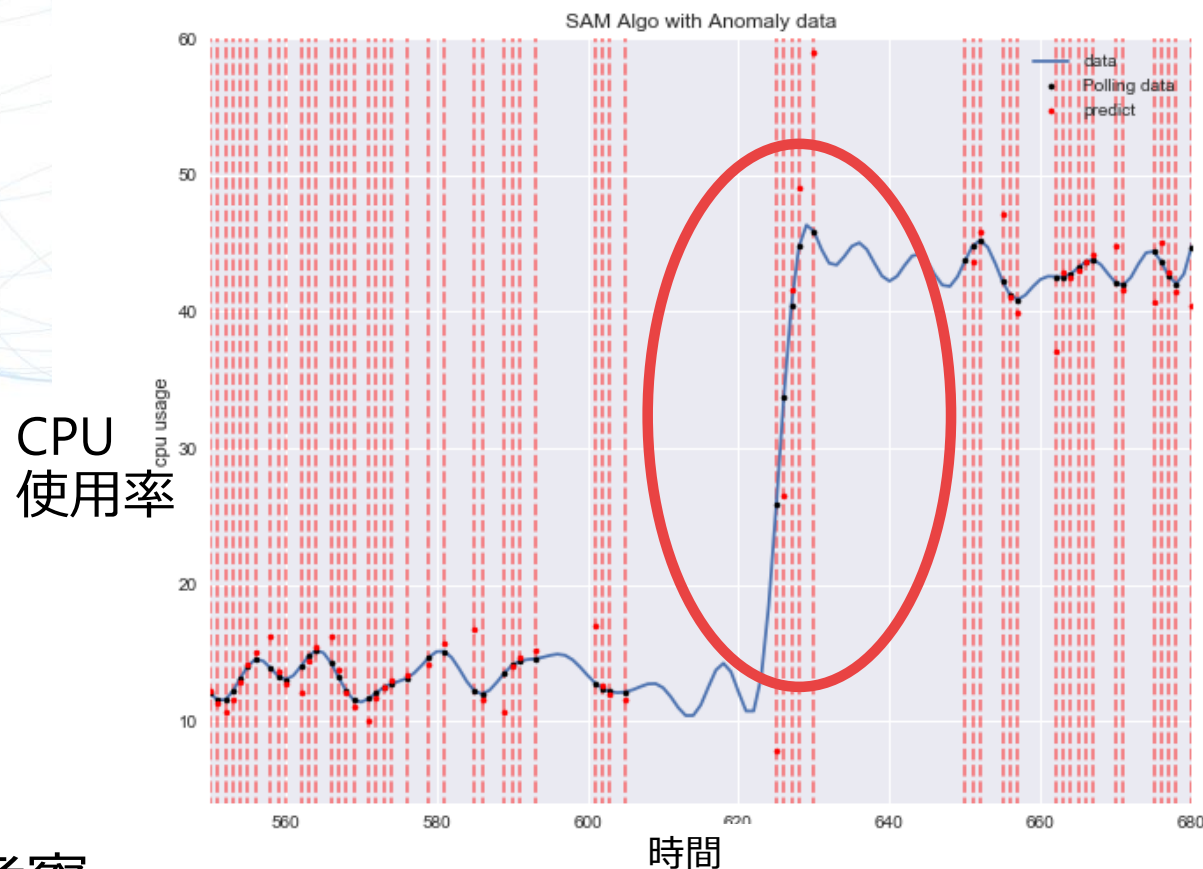
Fourier変換による  
平滑化データ





# いざ実験

- 用いたデータ
  - RUBiSをデプロイしたVMのCPU使用率 (13時間分)
- 用いた方法
  - 平滑化+ 先行手法
- 初期条件
  - ポーリングの上限を100[s], 下限を5[s]に設定

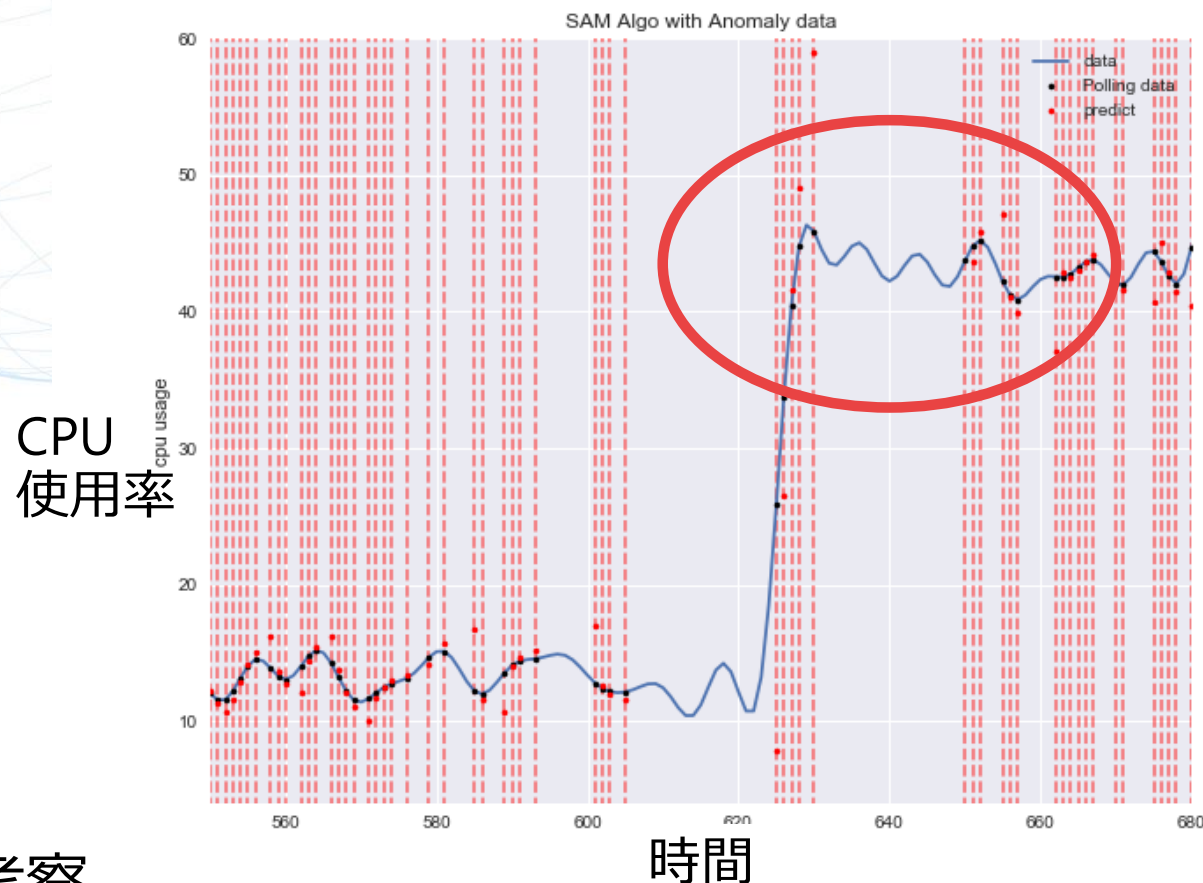


— CPU使用率

ポollingした場所

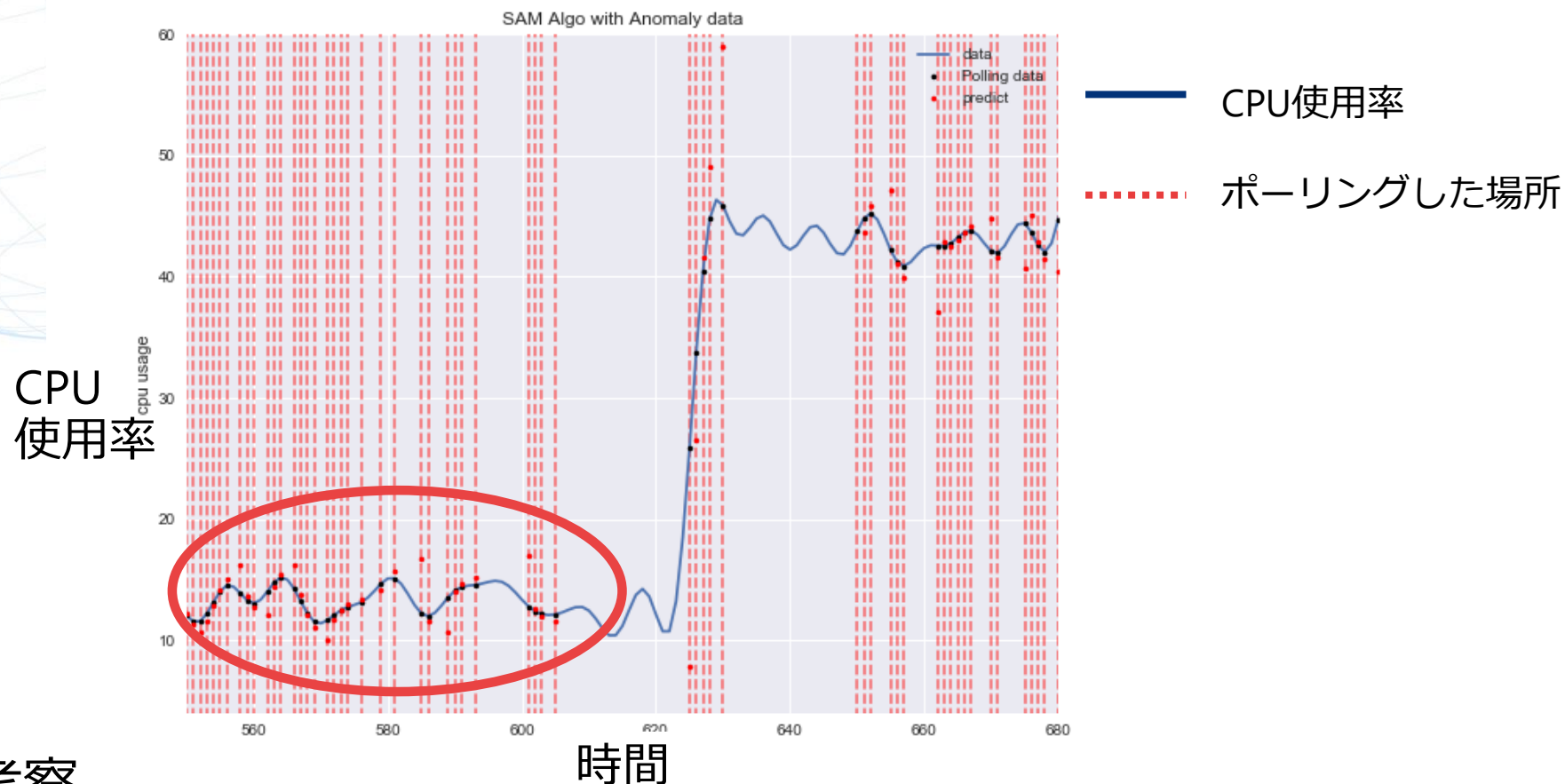
## ・ 考察

- CPU使用率が急上昇している異常部分については、ポolling間隔を縮めることがわかる.



## ・ 考察

- 値の上昇がおさまった部分でポーリング間隔を延ばしていることがわかる。



## ・ 考察

- しかし、正常部分ではうまくポーリング間隔を延ばせていない

- ・手法の課題

- ・通常時の細かい値の上下にも敏感に反応して、サンプリング間隔を縮めてしまう.

- ・中間発表以降の方針

- ・変化点検知手法を工夫することで、微小な揺らぎには過敏に反応しないようにできるのでは?

先行研究:生データ+変化点検知+ポーリング間隔調整

林手法: 前処理 + **変化点検知** + ポーリング間隔調整  
(new) (new)

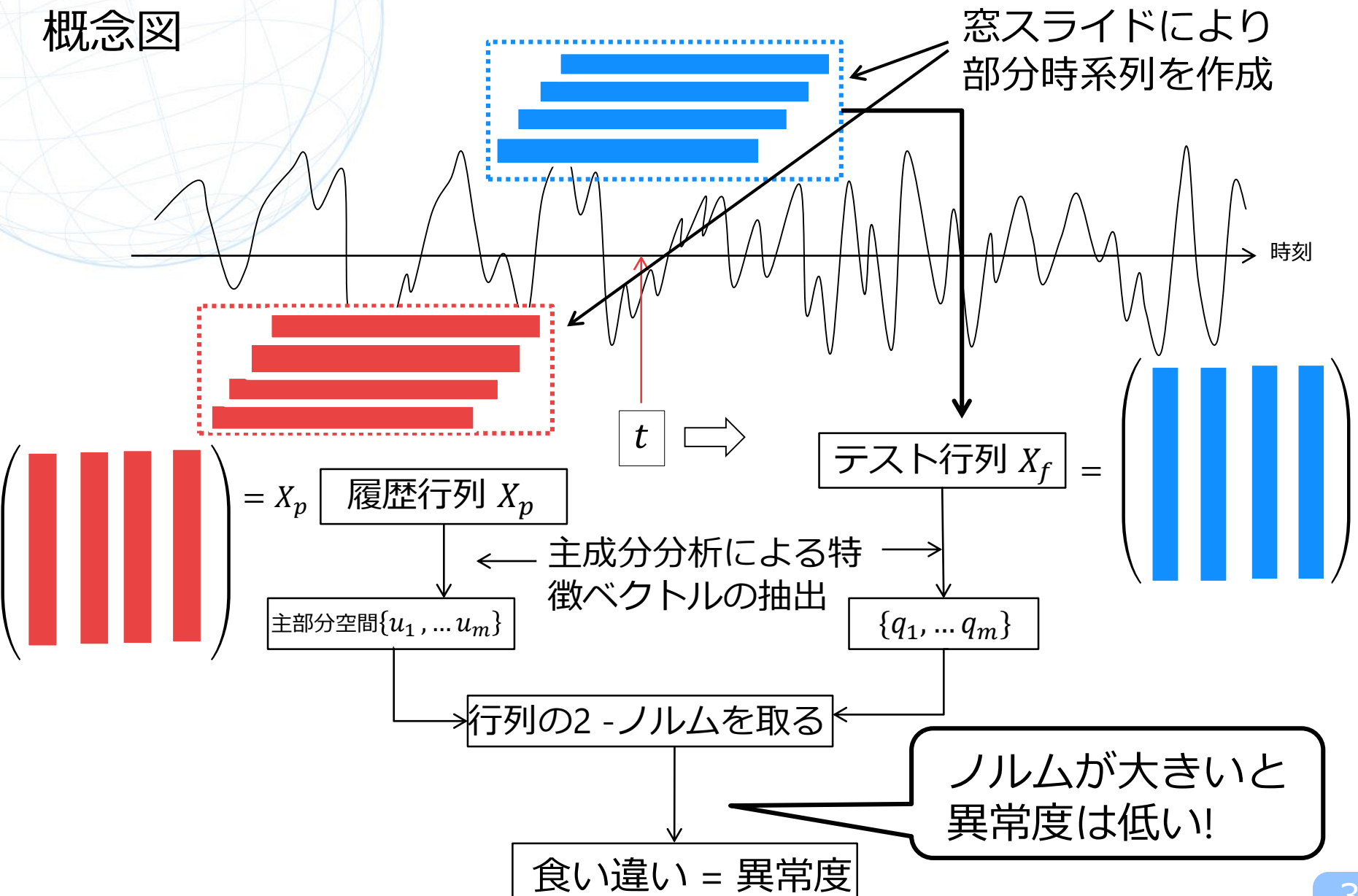


前処理を施すだけでは、ノイズへの過敏な反応を防ぎきれなくポーリング間隔調整が難しかったので、変化点検知手法を工夫する

- 特異スペクトル変換
  - 過去パターンと現在パターンの推移を表す特徴ベクトルを主成分分析により算出し, その食い違いを見る.
  - 幅広く使われる手法
- Change Finder
  - 二段階学習により, 時系列データの本質的な動きを検出する.
- コサイン類似度による手法
  - 次のデータ点の予測ベクトルと実測ベクトルの間でコサイン類似度を算出する

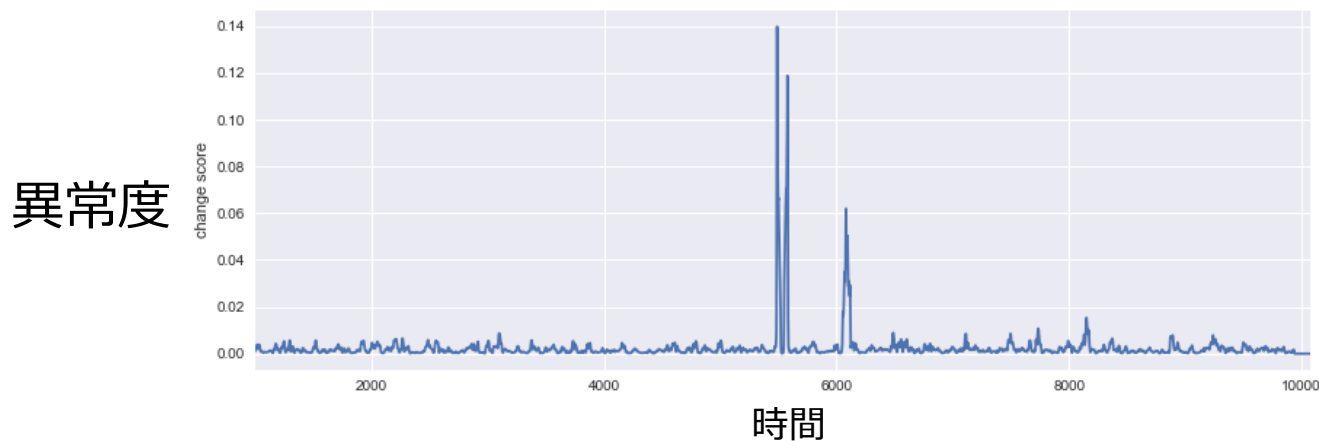
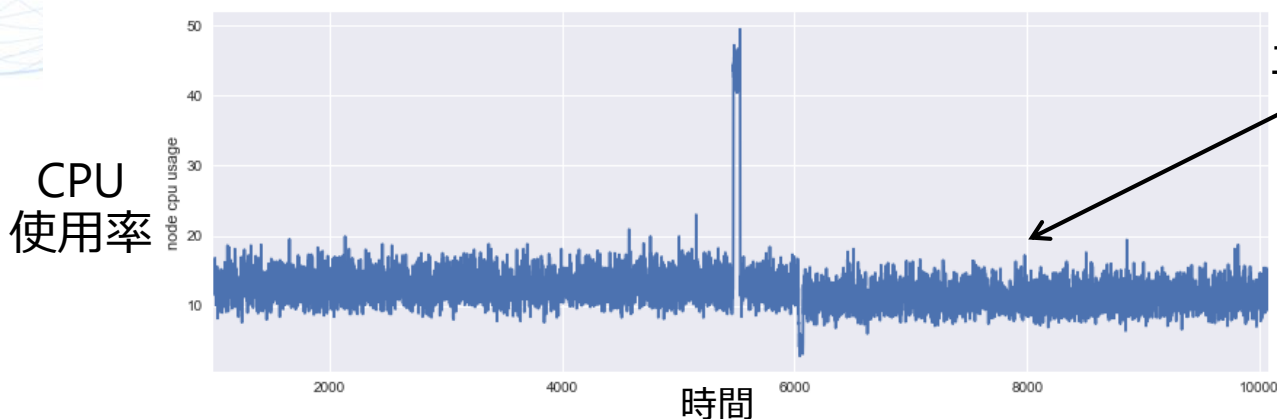


## 概念図



# 特異スペクトル変換

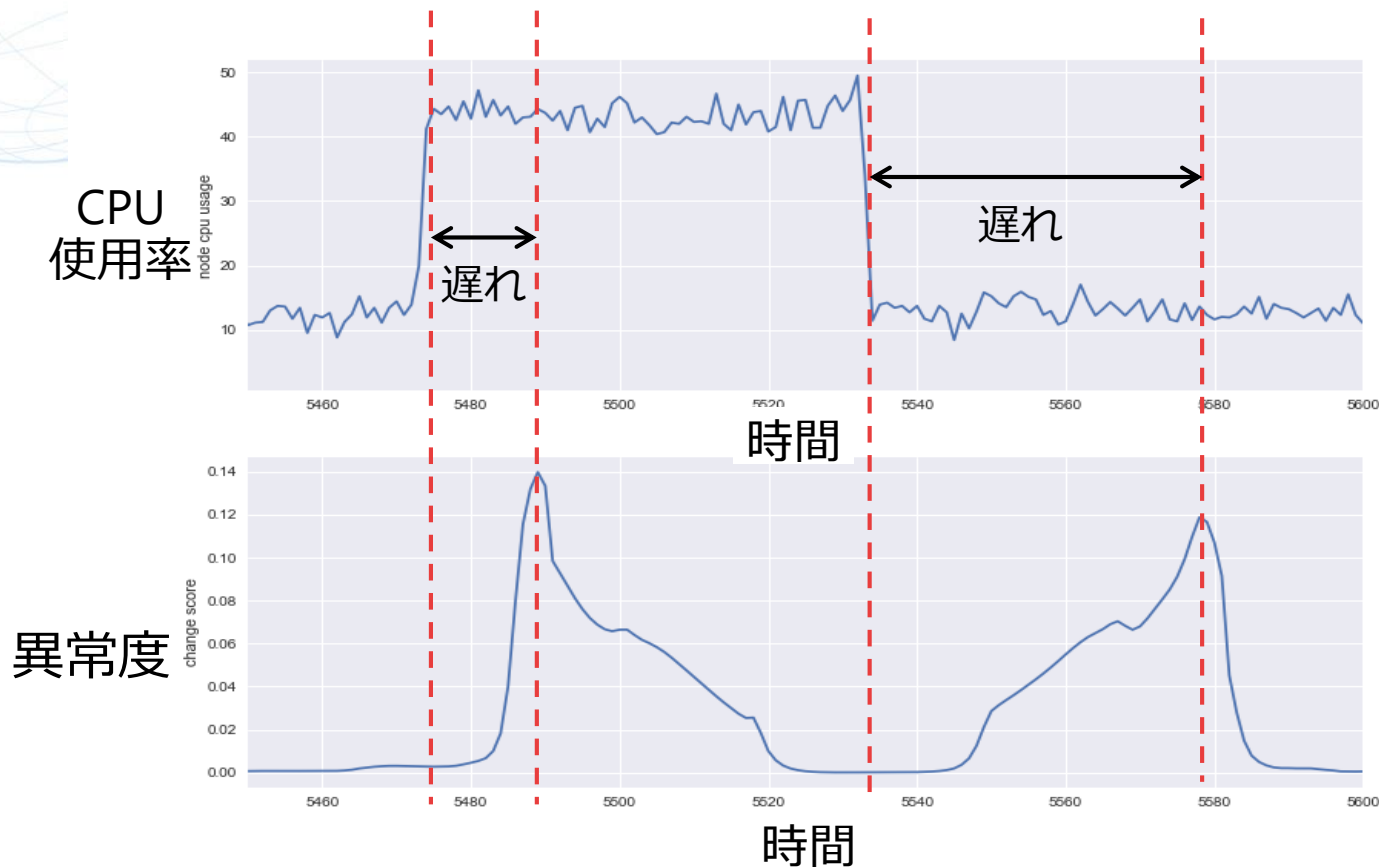
- ・ 良い点
  - 平滑化の必要がない.
  - ノイズに対して非常にロバスト



# 特異スペクトル変換

- ・ 問題点

- オンライン型対応だと、変化検知のタイミングが遅れる。



- Change-FinderはTakeuchi and Yamanishi[6]により提案された.
  - 機械学習の概念が異常検知の分野に初めて適応された有名な手法
- 概要
  - Change-Finderは2段階学習と忘却学習アルゴリズム(SDAR)により異常度を算出する.
  - Step 1
    - SDARアルゴリズムにより, 確率密度関数 $p_{t-1}(x_1, \dots, x_{t-1} | \theta)$ を推定.
    - $score(x_t) = -\log p_{t-1}(x_t | x_1, \dots, x_{t-1})$ により, 外れ値スコアを算出
    - 一定時間ごとに, 移動平均による平滑化を行う.
  - Step 2
    - Step1 で平滑化した外れ値スコアに対し, 再度SDARアルゴリズムにかける. この際, 算出される値を時系列の異常度とする.

**Remark :**

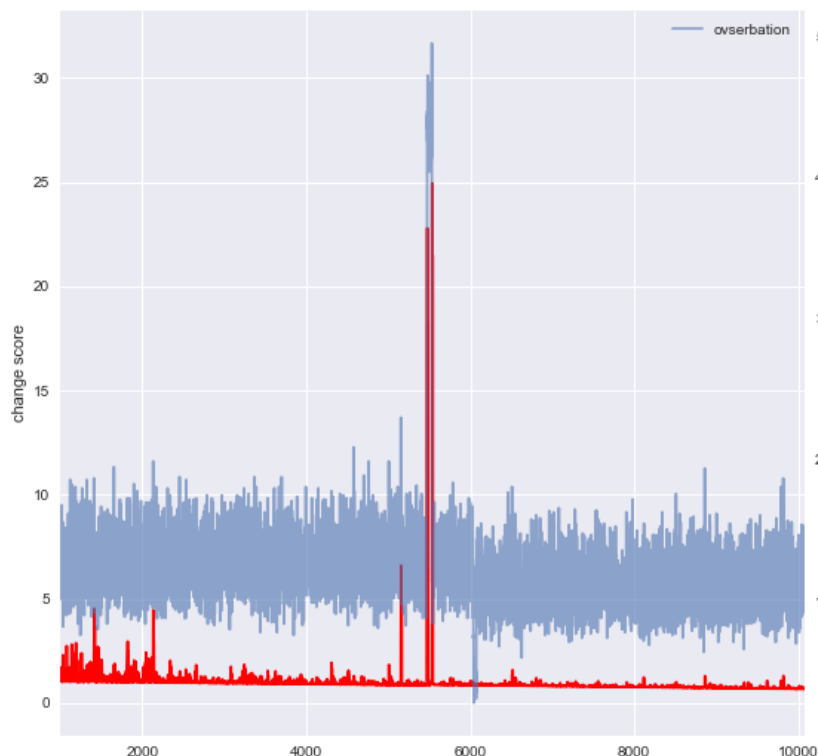
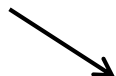
平滑化を行うことにより, 時系列データの本質的な変化を見ることができる.

# Change-Finder

- ・ 良い点

- ノイズに対してロバスト
- 平滑化の必要なし
- 高速かつ、検知能力が高い(オンライン型向き)

異常度の軸



— 実測値  
— 異常度

CPU使用率の軸

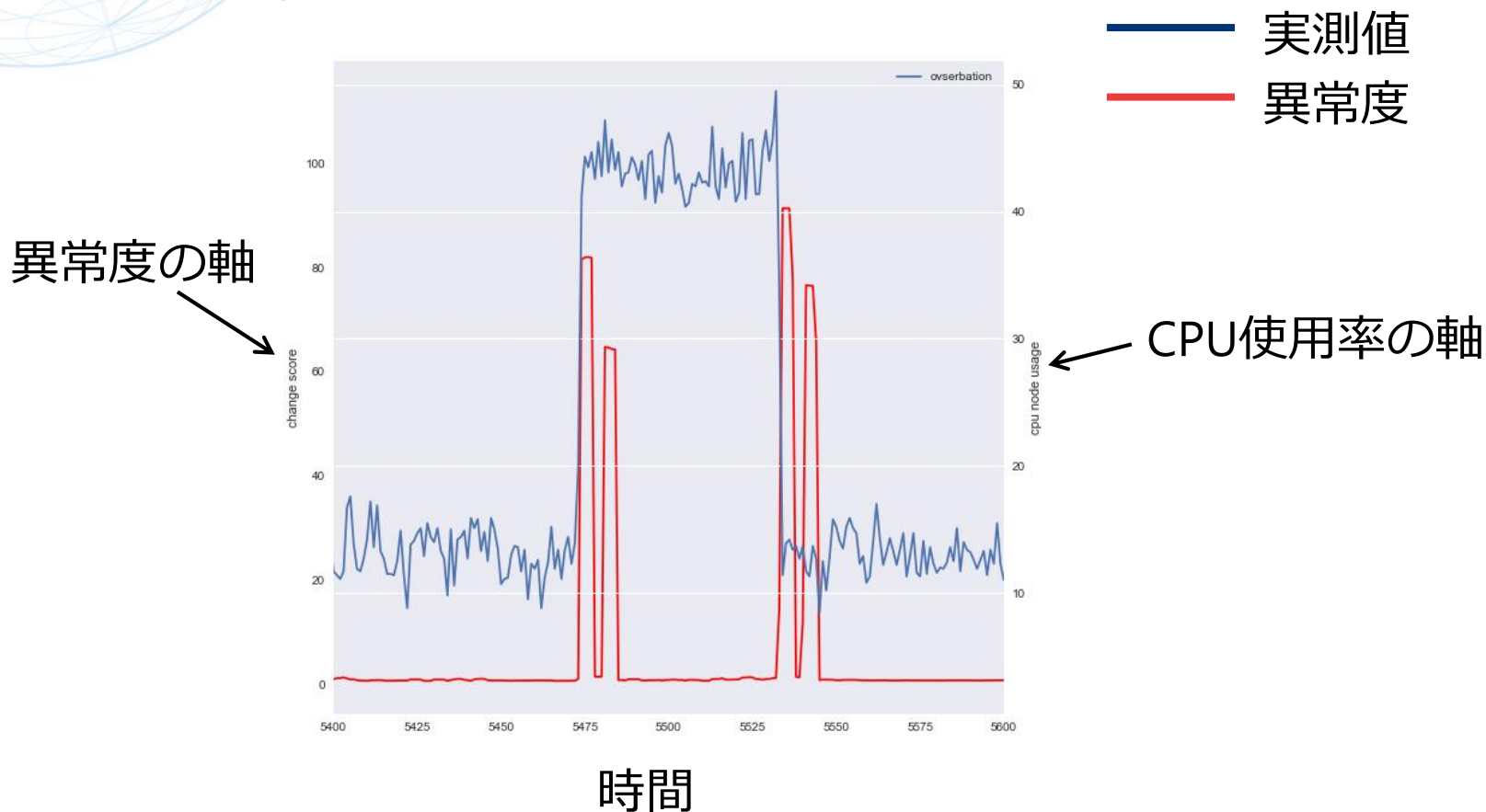


時間

# Change-Finder

- 良い点

- ノイズに対してロバスト
- 平滑化の必要なし
- 高速かつ、検知能力が高い(オンライン型向き)



# ハイパーパラメータ設定の難しさ

- ・ 問題点

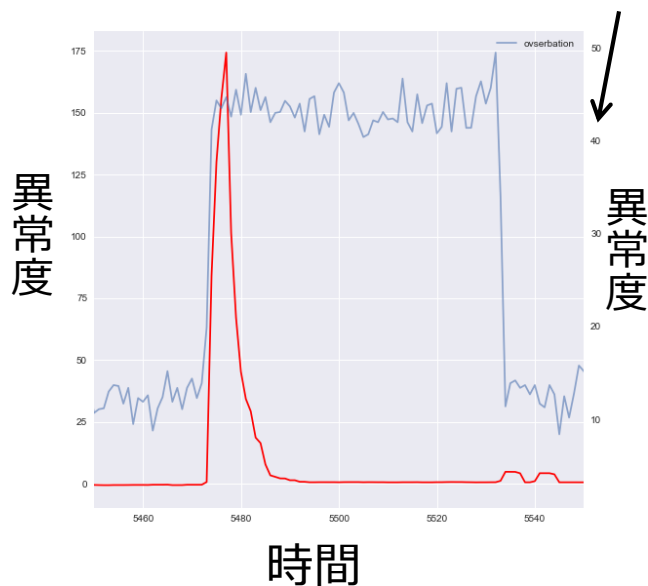
- ハイパーパラメータの設定が非常に神経質

— CPU使用率

— 異常度

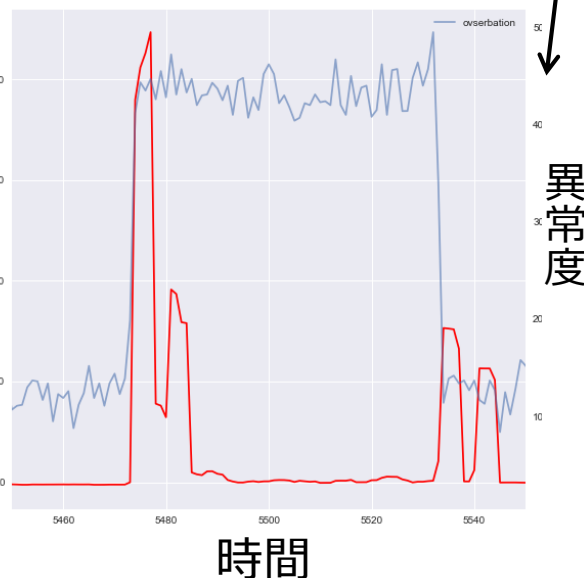
$r$ : 忘却率

$r = 0.001$



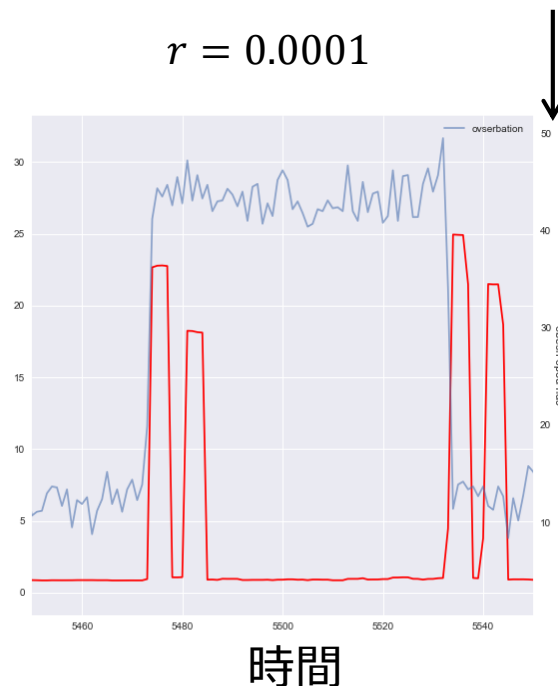
CPU使用率

$r = 0.0005$



CPU使用率

$r = 0.0001$



- ・ 忘却率と呼ばれるパラメータに関して、非常に神経質であることが分かる。

# コサイン類似度による手法

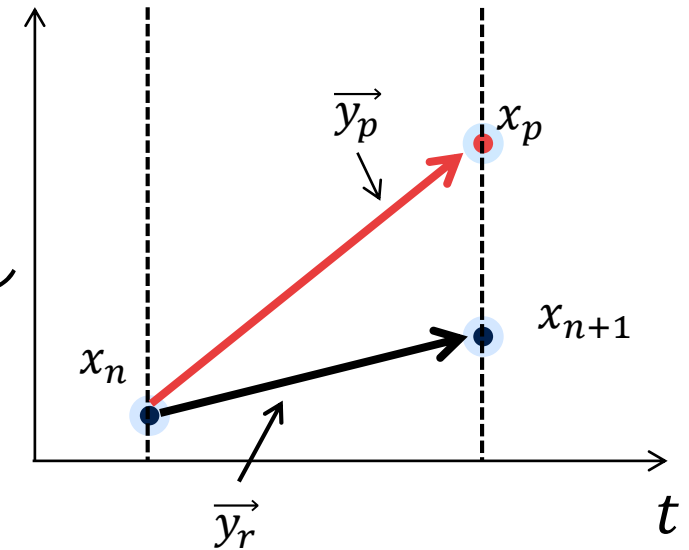
## 概要

- 予測値を算出するまでの流れは，先行研究と同じ．以下が変更点
- $x_n$  : 現在受け取った実測値  
 $x_p$  : 予測値  
 $x_{n+1}$  : 予測した地点の実測値  
 $\vec{y_p}$  :  $x_n$ と $x_p$ を結ぶ予測ベクトル  
 $\vec{y_r}$  :  $x_n$ と $x_{n+1}$ を結ぶ実測値ベクトル

$\vec{y_p}, \vec{y_r}$ のコサイン類似度:

$$S(\vec{y_p}, \vec{y_r}) = \frac{\langle \vec{y_p}, \vec{y_r} \rangle}{\|\vec{y_p}\| \cdot \|\vec{y_r}\|}$$

- 以下で，異常度を計算;  
if  $S(\vec{y_p}, \vec{y_r}) \geq 0$ :  
     $score = 1 - S(\vec{y_p}, \vec{y_r})$   
else:  
     $score = 1$



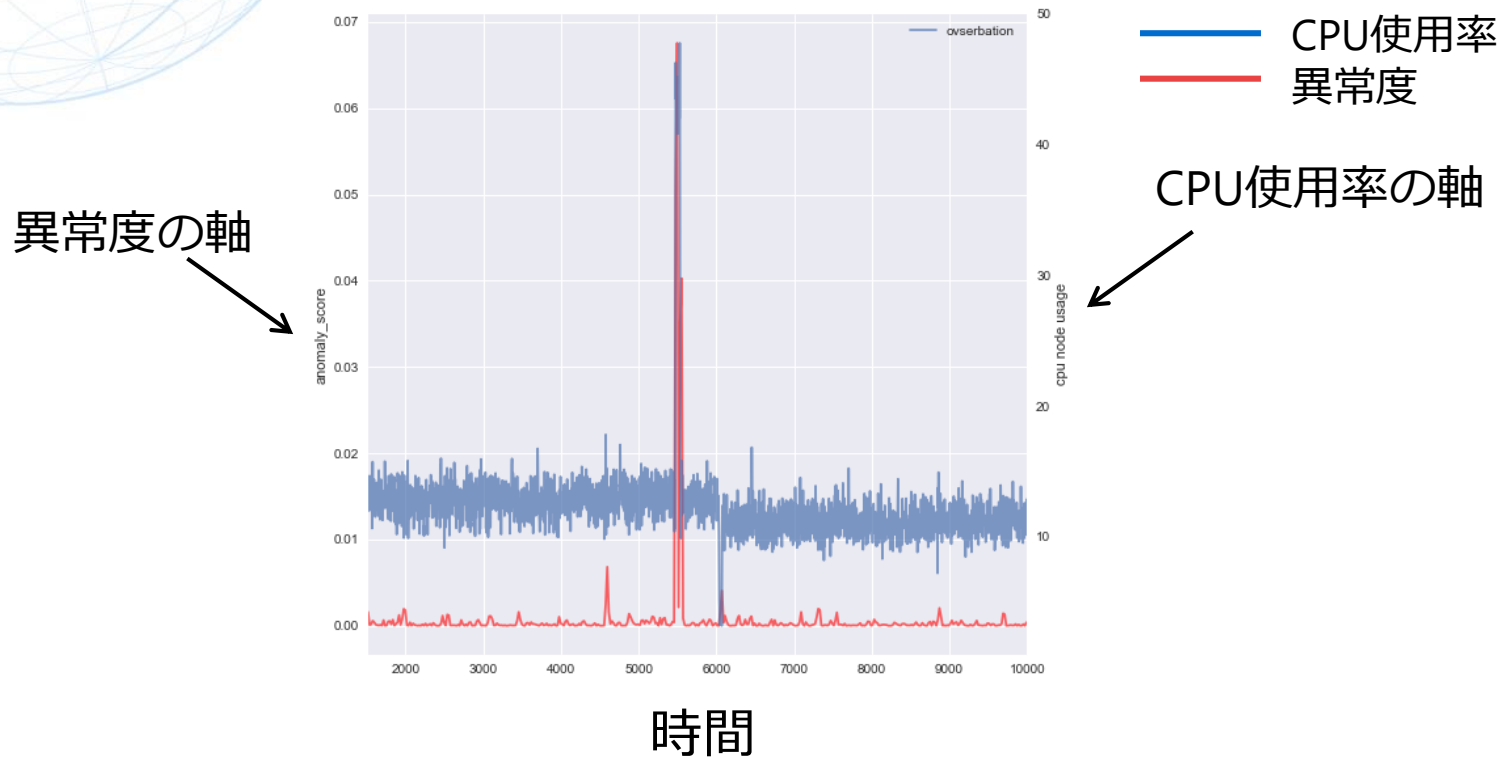
角度で異常度を  
計算している



# コサイン類似度による手法

- ・ 良い点

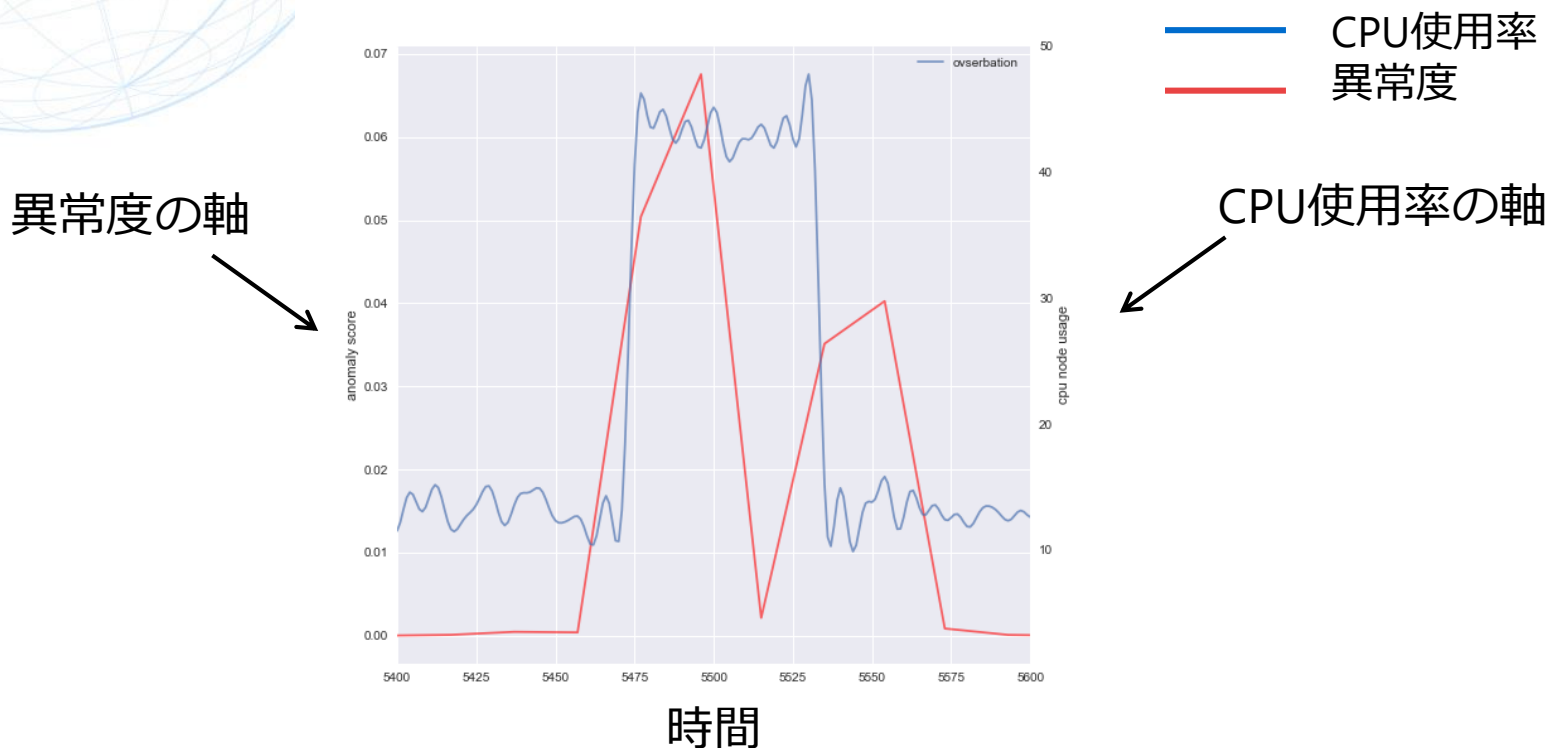
- ノイズにロバスト.
- 少ないハイパーパラメータの調整で異常度を算出できる.
- データ変化から遅れがなく検知できる.



# コサイン類似度による手法

- ・ 良い点

- ノイズにロバスト.
- 少ないハイパーパラメータの調整で異常度を算出できる.
- データ変化から遅れがなく検知できる.



# 採用した変化点検知手法

	ロバスト性	オンライン	パラメータ調整
先行研究手法	×	○	○
特異スペクトル変換	○	×	△
Change-Finder	○	○	×
コサイン類似度	○	○	○



コサイン類似度手法を採用



先行研究:生データ+変化点検知+ポーリング間隔調整

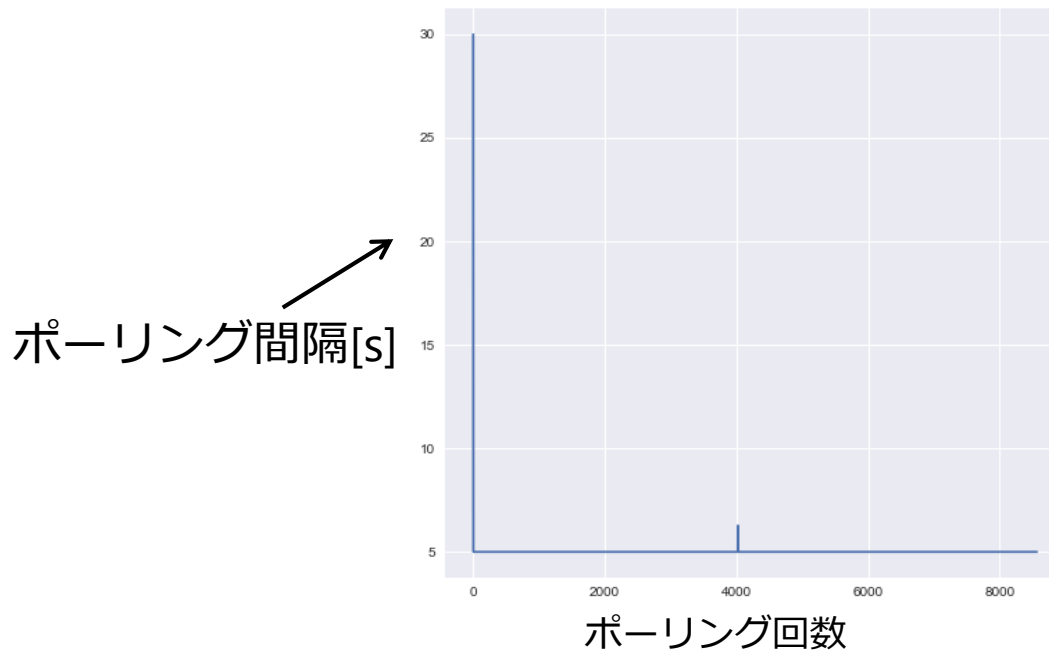
林手法: 前処理 + 変化点検知 (new!) + ポーリング間隔調整 (new!)

# 先行研究のポーリング間隔調整を流用した場合 NTT

- まず単純に思いつくのは、先行研究のアルゴリズムの流れはそのまま異常度をコサイン類似度手法で算出し、先行手法中のDに適用することが考えられる。
- 先行手法中のアルゴリズムに適用するために、Dとして以下を採用する。

$$D = (\text{異常度}) \times 2 - 1$$

- 以上の仮定のもと、算出したポーリング間隔の推移を以下に示す。



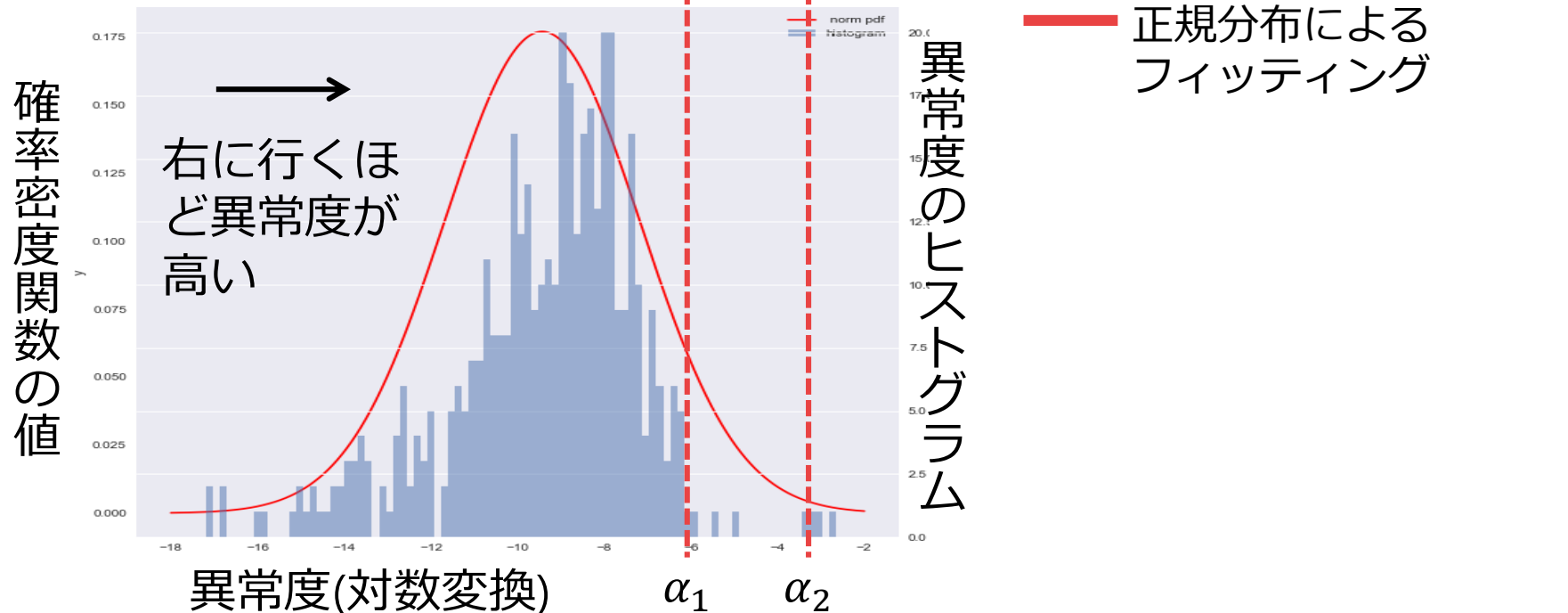
Dを負の値に拡張  
するためにする  
操作。

ポーリング間隔はほとんど最短間隔を取り続けている。

➡ 異常度のスケールがそもそも小さいことが原因。

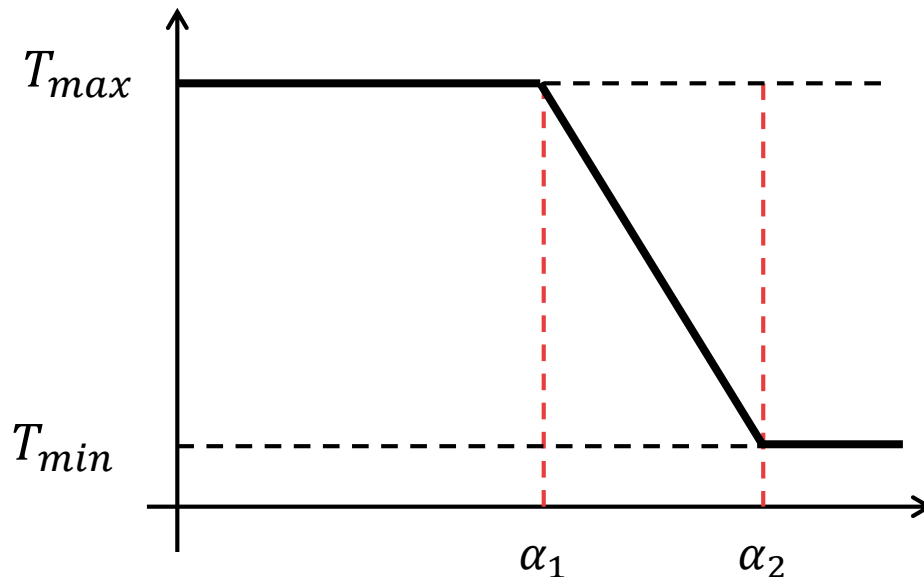
# 確率分布を用いたポーリング間隔調整

- ・ 今回用いたポーリング間隔調整手法.
  - ①正常時のデータ(訓練データ)から異常度を算出
  - ②異常度をある確率分布から生じたデータだと考え, パラメトリック推定を行う.
  - ③ ②で導いた確率分布の $\alpha\%$ 点を二つ取り, それぞれ $\alpha_1$ (第1閾値)  
 $\alpha_2$ (第2閾値)とする.



# 確率分布を用いたポーリング間隔調整

- ・今回用いたポーリング間隔調整手法.
  - ①正常時のデータ(訓練データ)から異常度を算出
  - ②異常度のある確率分布から生じたデータだと考え, パラメトリック推定を行う.
  - ③ ②で導いた確率分布の $\alpha\%$ 点を二つ取り, それぞれ $\alpha_1$ (第1閾値)  
 $\alpha_2$ (第2閾値)とする.
  - ④ 以下の式で, 次のポーリング間隔 $T_{new}$ を決める



$T_{new}$ の更新式;  $T_{max}, T_{min}$ はそれぞれポーリング間隔の上限と下限.

$$\text{if } a(x) \geq \alpha_1 : \\ T_{new} := T_{max}$$

$$\text{if } \alpha_1 \leq a(x) \leq \alpha_2 : \\ T_{new} := \frac{T_{max} - T_{min}}{\alpha_1 - \alpha_2} (a(x) - \alpha_1) + T_{max}$$

$$\text{if } a(x) \leq \alpha_2 : \\ T_{new} := T_{min}$$

- 評価対象
  - 先行手法
  - 林手法
- 評価指標
  - RMSE (Root Mean Squared Error)
    - 小さい方が良い

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

$n$  : データ点数  
 $y_i$  : 実観測値  
 $\hat{y}_i$  : 線形補間により予測したデータ点

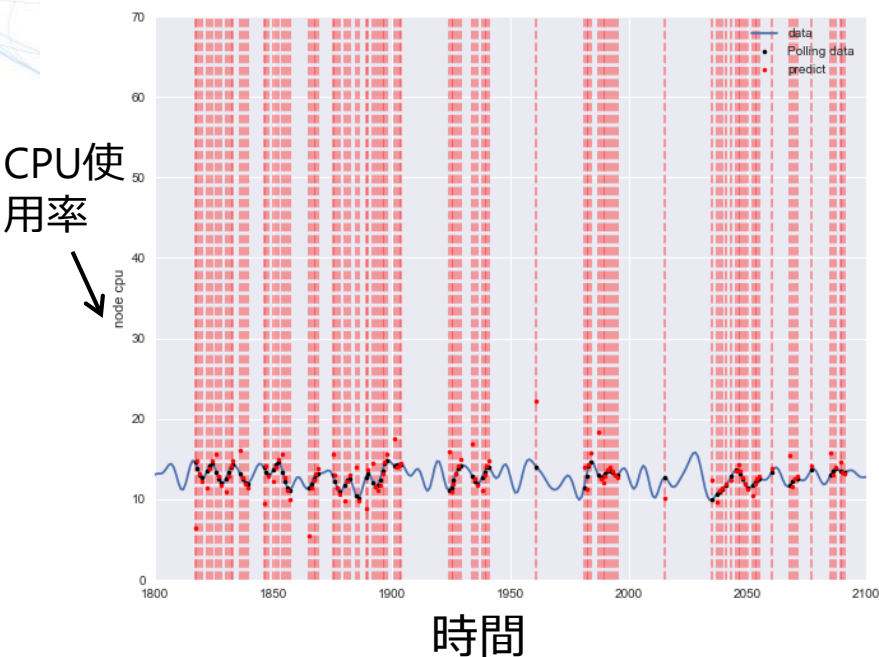
- ポーリング回数
    - 少ない方が良い
- 用いたデータ
  - Fourier解析により, 平滑化したデータ(異常部分を含む)を使用.
- 初期条件
  - ①ポーリング時間の下限を5[s], 上限を100[s]に設定.
  - ②初期のウィンドウサイズを100に設定.



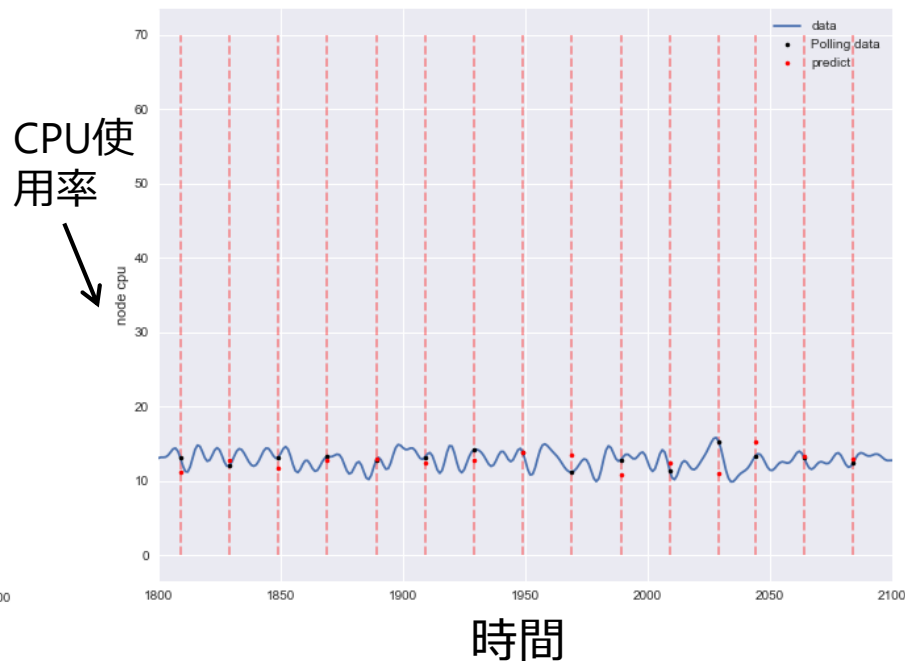
# 実験結果

- 同じ箇所(正常部分)に対するポーリング間隔の推移を表したのが以下の図である

先行手法



林手法  
CPU使用率  
ポーリングした場所



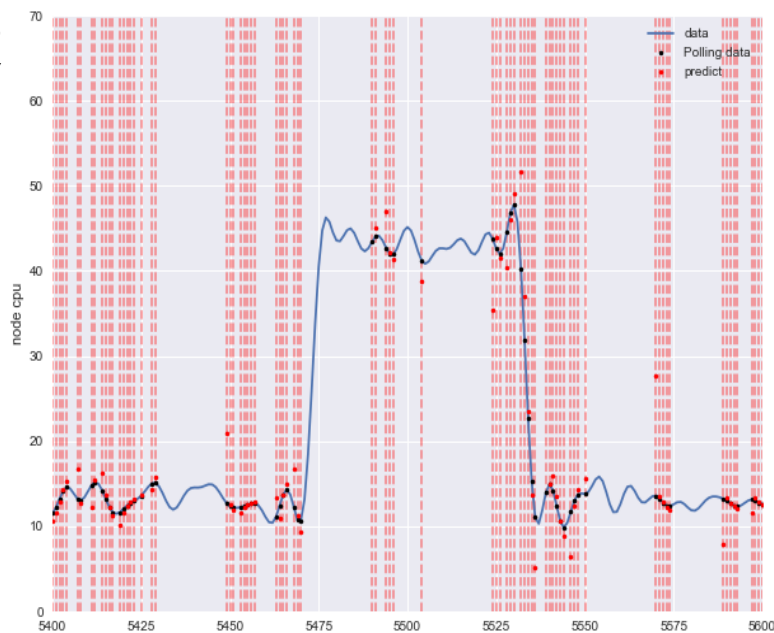
- 林手法では、正常部分でしっかりとポーリング間隔を延ばせていることが分かる

# 実験結果

- 同じ箇所(異常部分)に対するポーリング間隔をプロットしたものが以下の図である.

先行手法

CPU使用率



時間

CPU使用率

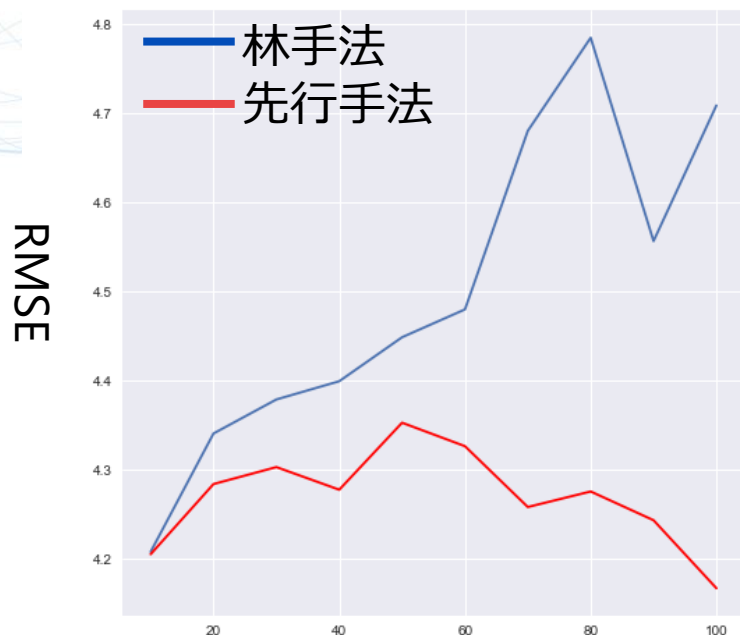


時間

- 異常部分に引っかけると、しっかりとポーリング間隔を縮めていることが分かる.

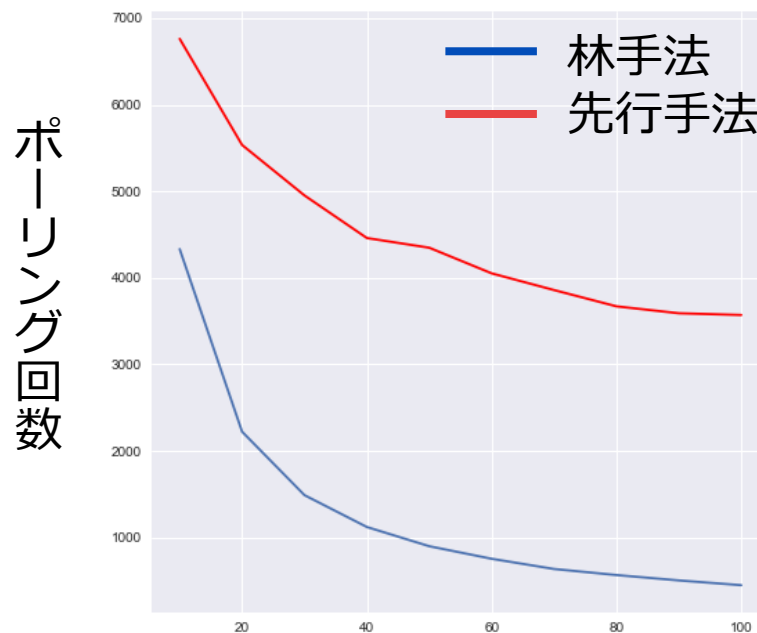
ポーリング間隔の上限を10刻みで、10~100[s]まで変化させた場合のRMSEとポーリング回数の推移を表したのが以下の図である。

## RMSEの推移



ポーリング間隔の上限

## ポーリング回数の推移



ポーリング間隔の上限

- ポーリング回数の上昇に伴い、RMSEが減少している。
- ポーリング回数は林手法が常に少ない。
- RMSEは先行手法が常に勝っている。

## ・考察

- 通常の変化に強く、ポーリング回数が先行手法と比べると、大幅に減少した.
- 異常時にポーリング間隔をしっかりと短くできている.
- 異常度を確率分布と絡めて、ポーリング調整をすることは微小な揺らぎの影響を受けにくくするために有効だと思われる.
- ポーリング回数とRMSEはトレードオフ関係にある.

## ・できたこと

- 先行研究提案のアルゴリズムの実装.
- いくつかの変化点検知手法の実装
- 先行研究の課題の発見&オリジナル手法による解決
- 異常度の確率分布を用いたポーリング間隔の自動調整.

## ・課題

- 異常と判断した後, ポーリング間隔を広げる時にもっと緩やかポーリング間隔が伸びるようにしたい.
- スケール調整がデータにより必要.
- ポーリング間隔自動調整手法の評価指標として, RMSEに代わる指標の検討.

## ・展望

- 隠れマルコフモデルなどの潜在変数を考慮したモデルによる変化点検知手法と絡める (カルマンフィルタや粒子フィルタ)
- 深層異常検知手法による集団型異常検知.

- 業務内容に関して

- ＜良かったこと＞

- 大学で学んだ知識が役に立ってうれしかった.
    - インターン後に自身が学ぶべき分野(課題)が分かった.
    - パワポでの資料作成技術が身についた(普段はBeamer)
    - RUBiSやPrometheusなどを用いた監視サーバの構築に興味をもった.
    - Dockerに興味を持った.

- ＜反省点＞

- 実装力
      - もっと高い実装力を持っていればデバックスの時間も少なくなり,もっと色々やりたいことができた.
    - プレゼンテーション力
    - わかりやすい資料の作成力.

## ・職場の感想

### <良かったと感じたこと>

- 大学での研究内容をしっかりと聞いてくれた.
- みなさん親切でとても働きやすかった.
- 議論とかできて楽しかった.
- イヤホンで音楽を聞きながら働いていても何も言われない.
- 9:30出社がとてもよかった.
- 書籍が充実している.
- 有給をしっかりと取るように上司が指示するところ

### <改善が必要と感じたこと>

- 社内ハッカソンが近年盛り上がり欠けている(と聞きました)
- 有志で集まる勉強会みたいなのがもっと活発にあってもよいのでは？と感じました.
- 日給2000円
  - 僕はそこまで気にしませんでしたでしたが、周りでもテーマは魅力的だが日給2000円は少なすぎるので、行きたくないという人が多かったです.

- [1] G. Tangari, D. Tuncer, M. Charalambides et al.: "Self-Adaptive Decentralized Monitoring in Software-Defined Networks", IEEE Transactions on Network and Service Management , 2018
- [2] 井出 剛: 入門 機械学習による異常検知 – Rによる実勢ガイド, コロナ社, 2015
- [3] 井出 剛: 杉山 将 異常検知と変化検知, 講談社, 2015
- [4] 北川源四郎: 時系列解析入門, 岩波書店, 2005
- [5] ビショップ, C.: パターン認識と機械学習(上・下), 丸善出版, 2012
- [6] Takeuchi, J. and Yamanishi, K.: A Unifying Framework for Detecting Outliers and Change Points from Time Series, IEEE Transactions on Knowledge and Data Engineering, Vol. 18, No. 4, pp. 482–492 (2006).
- [7] 山西健司, データマイニングによる異常検知, 共立出版, 2009





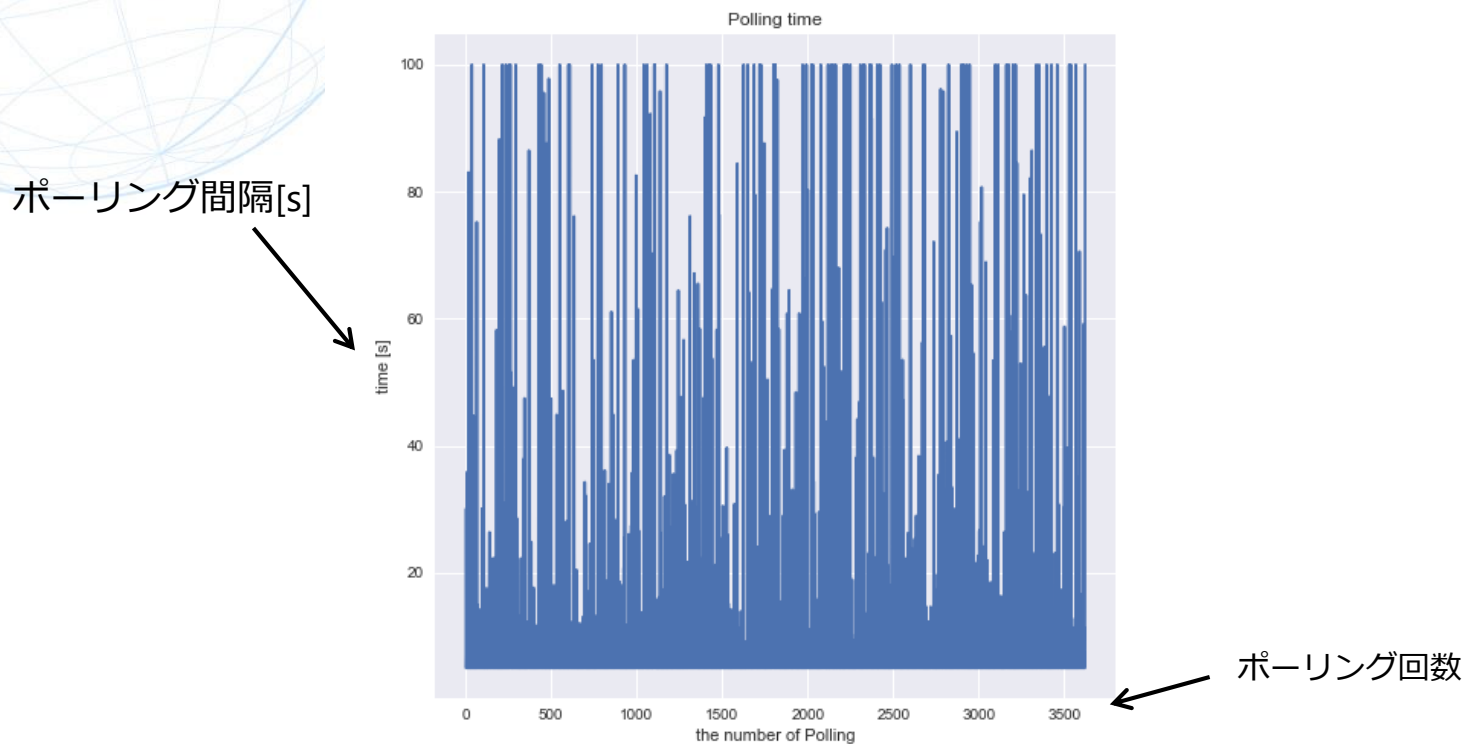
**ご清聴ありがとうございました**



# 補足資料

# 実験結果

- ・ポーリング間隔の推移を表した図を以下に示す.



- ・前処理を施すことにより, ポーリング間隔が変動するようになった.

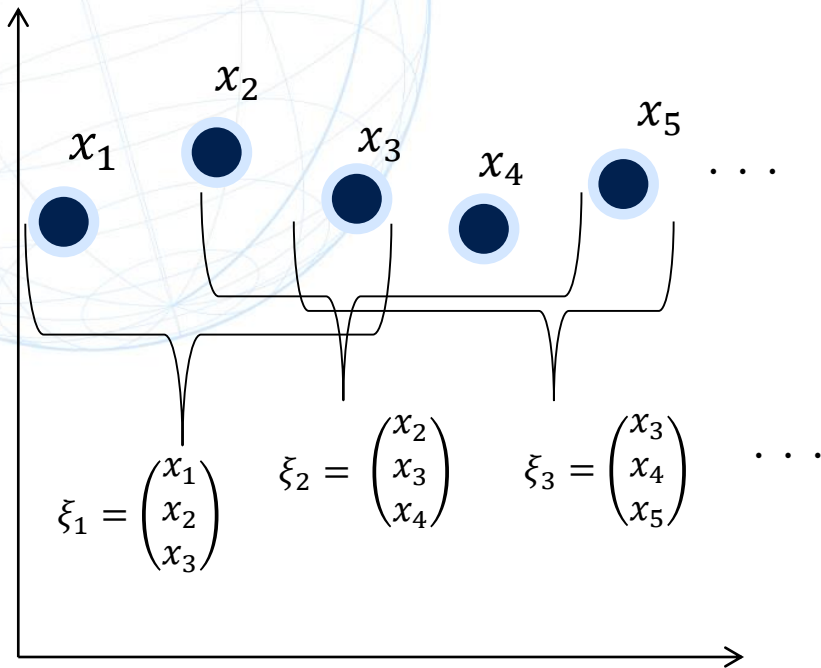
# 特異スペクトル変換

ふあ

# 実装上の注意点

- ・ 論文の主要アルゴリズムの疑似コード中の式にそもそも間違いがあったので、その修正
- ・ 疑似コード中に、突然現れるAIMDアルゴリズムの実装
- ・ ウィンドウサイズNの更新式をそのまま実装すると、ゼロ割り算が発生するので、その回避

# 移動平均による平滑化の考え方



$N = 3$ のとき

- ・スライジングして取ってきた各 $\xi_i$ について、平均を取るという考え方.
- ・よく使われる.
- ・ $\xi_i$ のことを部分時系列データと呼ぶ文献も多い.

## 用いたプログラミング言語 & ツール

- Python (ver3.7.3)
- Jupyter notebook

### Pythonとは？

- MITリリースの科学技術計算を得意とするプログラミング言語
- 近年、非常に人気が高まっている！



- 書きやすい & 読みやすい
- 科学技術計算用のモジュールがたくさん用意されている
- 近年流行の、機械学習をはじめとしたデータ解析を行いやすい環境

### • Jupyter notebookとは？

- Pythonを使っている人は、たいていこのnotebookを使って解析をしている.
- 対話型かつ、コードブロックごとに実行できるので使いやすい.
- Markdownも使えるので、ちょっとしたメモを取りつつコードがかけやすい.

# どのような時にポーリング間隔は変化するか NTT

$D \geq 0$  のとき(増える)

$$D = \frac{x_p - x_{n+1}}{x_{n+1} - x_n}$$

$D < 0$  のとき(減る)

$x_n$  : 現在受け取った実測値  
 $x_p$  : 次回のポーリング時刻での予測値  
 $x_{n+1}$  : 予測した時刻での実測値

