放電加工機の異常検知（状態空間モデル）

１．はじめに

以前より、放電加工機の異常検知の検討をしていました。近々に、1 年分のデータが採取できるので、データの加工の見直し、異常検知手法の再検討を行い、再トライします。

この技術報告書では、異常検知の手法として状態空間モデルについて、概略を報告します。状態空間モデルは、一定間隔でない時系列データにも適用可とあり、装置関連のデータ解析に有効ではないかと考えました。

最後の項で、Python で実行するプログラムを紹介します。

放電加工機の 1 年分のデータ解析は、次回で紹介できたらと思います。

1.1 時系列データにおける異常検知手法の色々

代表的な手法の利点と欠点は以下です。

動的時間伸縮法（近傍法に近い）　利点：シンプル　欠点：ノイズに弱い

特異スペクトル変換　　利点：ノイズに頑健　欠点：計算コストが高い

自己回帰モデル　利点：周期性があるデータに強い。

欠点：次数が決めうちのため状態が一定なものでないと適用が難しい

状態空間モデル　利点：状態が変化しても対応可能かつノイズに強い。

欠点：周期性があり安定しているデータには対応しづらい

いままでの検証結果によれば、上の二つは異常検知できたようですが、

下の二つの結果は、思い通りにできてません。

自己回帰モデル は、放電加工機は周期性のあるデータでは無いので、理解できますが、

状態空間モデル　は、こちらの取組不足の点もあるので、プログラムも更新して、見直そうと考えています。

1.2 時系列データと点過程データについて

時系列（time series）データと点過程（point process）データがあります。

時系列データとは、ある時間間隔で観測した結果のデータです。

例えば、月ごとの受注件数、1 日ごとの売上の合計、1 時間ごとの平均気温などです。

点過程データとは、事象の発生を、発生した時刻とともに記録したデータです。

例えば、装置のエラーの発生とその時刻などです。

多くは、点過程データから時系列データを作り、時系列データに対しデータ分析を実施するのが主流とのことです。問題は、どうやって点過程データを集計し時系列データにするのか、になります。

放電加工機の異常検知において、1 回の稼働時間は長くて 3～4 時間程度です。故に、ある時間間隔の観測値を、長期に渡ってデータ採取する事はできません。

出来そうな事は以下の 2 点です。

・1 稼働の単位で、観測値を所定の時間間隔でデータ採取し稼働単位の異常検知を行う。

・1 日の稼働の平均値を、 1 日ごとの観測値と扱い、長期間で、異常検知を行う。

また、観測値については、以下の 2 点があげられます

・放電加工時の出力値（例えば、電流値、電圧値）を測定して、観測値として扱う。

（上記外でも、加工機の状態に関連する観測値は、ワイヤを駆動させるモーターの負荷変動とか、メーカの協力が必要ですが、色々と想定できます。）

・放電加工時、コーションを含むメッセージが発信されます。メッセージを数値化して、コーションの度合いによりメッセージを数値化して、データとする。

メッセージは点過程データですので、1 日毎の平均点で、観測値にします。

今回は、このメッセージを採取することで、時系列データとして扱おうとしています。

1.3　引用・参考　文献、WEB

異常検知というか機械学習は、線形代数と統計学の知識が必要です。一方、Python 等でのプロムグラミングは、ライブラリーというブラックボックスの関数で処理できるので、関数の詳細は知らなくても進められます。

この報告書では、いまさら数学・統計の勉強を深堀りできないし、されとて、ブラックボックスの活用のみではつまらないという方に、数式だけでなく、文章で、概要を伝えられないという思いで進めています。

状態空間モデルの線形代数と統計学からの裏付けは、以下の井出先生の本を参照ください。　Python のプログラムは　【時系列データ】カルマンフィルターで異常検知　を参照しました。

「入門　機械学習による異常検知―R による実践ガイドー」　井出剛　著　コロナ社

・時系列データにおける異常検知

https://www.kabuku.co.jp/developers/time_series_anomaly_detect

・第 279 話｜点過程データと時系列データ

https://www.salesanalytics.co.jp/column/no00279/

状態空間モデルの意味と 4 つの分類（線形性、正規性、時変性、連続性）

https://mathwords.net/zyotaikukan

・【時系列データ】カルマンフィルターで異常検知

https://qiita.com/shinmura0/items/8c1e83636291f10e0840

特異値分解について勉強する（ざっくり理解する編）

https://cha-kabu.hatenablog.com/entry/2020/10/31/215515

最尤推定法・尤度関数の直感的な考え方｜データから分布を推定

https://math-note.xyz/statistics/maximum-likelihood-estimation/

2. 状態空間モデルについて

2.1 状態空間モデルの基本

状態空間モデルの説明は、線形代数と統計学の数式の羅列になります。

そのような事は、私は出来ませんし、ここまで読んだ人も期待してないと思うので、

概要のみの説明にします。

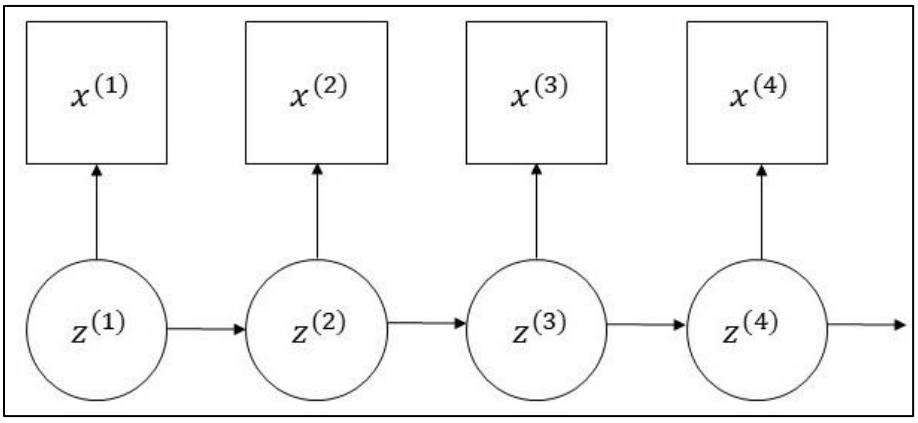
右の数式の意味は、理解ください。　　$P(A \mid B) = N(A \mid \mu, \sigma^2)$

左辺は、「条件 B の下での A 確率」

右辺は、「A は、平均 μ　分散　σ^2　の正規分布」　　の意味です。

また、今回扱う状態空間モデルは、線形・正規・時不変・離散型の状態空間モデルです。

以下の図は、状態空間モデルの解説の冒頭に必ず出てきます。



図中で、$z^{(t)}$　は 時刻 t における状態を m 次元ベクトルで表し、

$x^{(t)}$ は 時刻 t における観測量をM次元ベクトルで表します。

言葉で説明すると、「ある時点の状態は、その前時点の状態に依存し、観測値はその時点の状態にのみ依存する」という意味です、具体的な表現で言うと、池の魚の数で、その日に見つけた数が x 匹なので、池全部には z 匹いるだろう、次の日は z からの増減だろうという事です。　　さらに　数式で表すと、

$x^{(t)} \approx C\, z^{(t)}$　　C　は　M x m　行列　　(2.1)

$z^{(t)} \approx A\, z^{(t-1)}$　　A　は　m x m　行列　　(2.2)　　となります。

変化は、正規分布に従って確率的におこなわれるので、確率のモデルは以下です。

$p(x^{(t)} \mid z^{(t)}) = N(x^{(t)} \mid Cz^{(t)}, R)$ (2.2)

$p(z^{(t)} \mid z^{(t-1)}) = N(z^{(t)} \mid Az^{(t-1)}, Q)$ (2.3)

Q は　状態の、遷移のばらつきを表す m x m　の共分散行列

R は　観測量のばらつきを表す　M x M　の共分散行列

加えて　始点の　t=1 に対応するため、$p(z^{(1)}) = N(z^{(1)} \mid z_0, Q_0)$　を仮定します。

やることは以下の 2 点です。

①　未知パラメータ　A, C, Q, R　の推定

②　観測データから、状態変数系列の逐次推定

手法は以下となります。

①　部分空間同定法、特異点分解、最尤推定という手法で行います

②　カルマンフィルタを用いた手法で行います

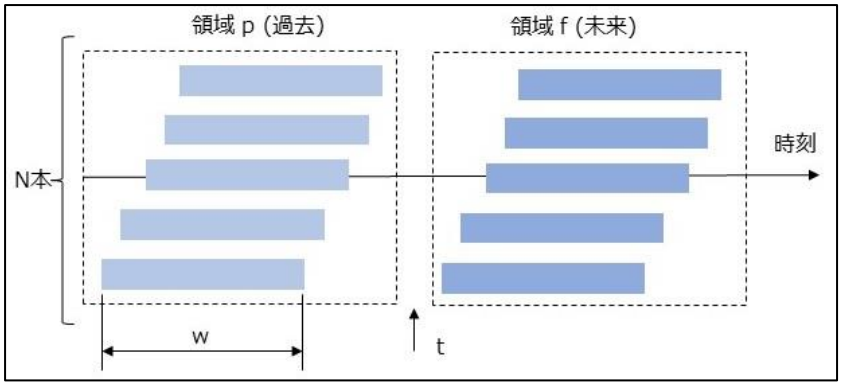
最後に、異常度を算出します。

以降は　キーワードについて解説します。

2.2　部分空間同定法

時系列データの部分系列から形成される部分空間を想定して、過去と未来の部分空間の距離を近くするパラメータを求める手法です。

これでは、何の事か分からないので、下の図で、追加説明します。



横軸上に、M 次元の観測量が T 個　$x^{(1)}$　……　$x^{(T)}$　のデータがあります。時系列を長さ w の窓幅で切り取り、領域 p と f に N 本の部分時系列のベクトルを作ります。

時刻 t を先頭とする部分時系列のベクトルは以下となります。

$$X^{(t)} \equiv \begin{bmatrix} x^{(t)} \\ x^{(t+1)} \\ : \\ x^{(t+w-1)} \end{bmatrix} \qquad (2.4)$$

それぞれの部分時系列ベクトルは　Mw次元の長いベクトルになります。

部分時系列のベクトルを列ベクトルとする行列X_p X_f を次のように定義します。

$$X_p \equiv [X^{(t-N)} \cdots\cdots X^{(t-2)}, X^{(t-1)}] \qquad (2.5)$$

$$X_f \equiv [X^{(t)}, X^{(t+1)}, \cdots\cdots, X^{(t+N-1)}] \qquad (2.6)$$

式 (2.1),(2.2)を適用すると、以下となります。

$$X^{(t)} \approx \begin{bmatrix} \mathrm{C}\ z^{(t)} \\ \mathrm{C}\ z^{(t+1)} \\ : \\ \mathrm{C}\ z^{(t+w-1)} \end{bmatrix} \approx \begin{bmatrix} \mathrm{C}\ z^{(t)} \\ \mathrm{CA}\ z^{(t)} \\ : \\ \mathrm{C}A^{w-1}\ z^{(t)} \end{bmatrix} = \begin{bmatrix} \mathrm{C} \\ \mathrm{CA} \\ \\ \mathrm{C}A^{w-1} \end{bmatrix} z^{(t)} \equiv \Gamma\ z^{(t)} \qquad (2.7)$$

再右辺は、Mw x m の　行列　Γ　の定義式で、可観測性行列と呼ばれます。

次に、状態変数をまとめた行列を以下とします。

$$Z_p \equiv [z^{(t-N)} \cdots\cdots z^{(t-2)}, z^{(t-1)}] \qquad (2.8)$$

$$Z_f \equiv [z^{(t)}, z^{(t+1)}, \cdots\cdots, z^{(t+N-1)}] \qquad (2.9)$$

とすると　以下の関係式となります。

$$X_p \approx \Gamma Z_p \qquad X_f \approx \Gamma Z_f \qquad (2.10)$$

Z_f は　Z_p　の時刻を N ステップ進めたものですので、

$Z_f \approx A^N Z_p$　となり、転置行列にすると、　$Z_f^T \approx Z_p^T\ A^{NT}$ となります。　(2.11)

最後の式から、状態の未来を示す行列（空間）と状態の過去を示す行列（空間）の関係が拘束されおり、大きな特性の変化が無ければ、互いの空間が近いと考えます。

よって、　Z_f^T　と Z_p^Tの列空間の距離が最小となるパラメータを推定します。

この推定は、特異点分解という手法で行います。

2.3　特異点分解と最尤推定

　特異値分解とは次元削減手法のひとつで、任意の行列を 3 つの行列の積に分解し、重要度の低い余分な列ベクトルを削ることで元の行列に近似した低次元の行列を作成します。そして、特異値が大きい順に m 本の左右特異ベクトルを求めます。

　行列 A　を　A　=　$U\Sigma V^T$ 分解します。

U は A の左特異ベクトル　と　V^Tは右特異ベクトルと呼ばれます。

Σ　は対角行列で、対角成分は、(　σ_1 , σ_2　, … ,　σ_n) で　A の特異値と呼びます。

$\sigma_1 \geq \sigma_2 \geq, \cdots, \geq \sigma_n \geq 0$　の順に並びます。

A を展開すると

$A = u_1\sigma_1 v_1^T + u_2\sigma_2 v_2^T + \cdots + u_m\sigma_m v_m^T$

となり、　σ_n　は、左の方が大きいので、影響が大きく、重要度は高くなります。

つまり、右側の方は、無視できるということで、次元の削減ができます。

ここまでが、特異点分解の概論で、次に、状態空間モデルへの適用の話になります。

　2.2 項　の末尾にあった　「Z_f^T　と Z_p^Tの列空間の距離が最小となる　」　を

「Z_f^T　と Z_p^Tの列空間の相関係数を最大化する　」　に読み替えます。

次に、式（2.10）から　$Z_p^T = X_p^T$α　,　$Z_f^T = X_f^T$β　が導かれ、

「X_p^Tα　と　X_f^Tβ　との相関関係を最大化する　αとβ　を算出すること」になります。

X_p^Tα　と　X_f^Tβ　の行列を、$U\Sigma V^T$　に分解して、α、βの値を、最適化します。

　まだ、X と Z　の関係が明示されたところまでで、A,C,Q,R のパラメータの算出は、これからで、これを、最尤推定で、求めます。

最尤推定を簡単に言うと、X と Z のデータが入手できたので、この手元のデータを、実現する確率が最大となる、パラメータを算出する　ということです。

2.1 項にある式を以下に再掲し、補足します。

$$p(x^{(t)} \mid z^{(t)}) = N(x^{(t)} \mid Cz^{(t)}, R) \qquad (2.2)$$

$$p(z^{(t)} \mid z^{(t-1)}) = N(z^{(t)} \mid Az^{(t-1)}, Q) \qquad (2.3)$$

これらの式から、対数尤度関数を求め、微分して最大値を求めます。

詳細の数式は参考文献を参照ください。

一般論として　平均　μ　分散　σ　正規分布の対数尤度関数は

M（μ、σ）＝ log L（μ、σ）

$= - n/2 \log 2\pi - n\log\sigma - [(x_1-\mu)^2 + \cdots\cdots + (x_n-\mu)^2] /2\sigma^2$

となります。

次に、微分を用いて、最大値を求めます。

$$\frac{dM}{d\mu}(\mu,\sigma) = \frac{dM}{d\sigma}(\mu,\sigma) = 0$$

を満たす時　最大値となり、A,C,Q,R を求めることになります。

2.2.3　カルマンフィルタ（カルマンゲイン）

　上記で、パラメータを決めました。これで、事前の推定値と実際の観測値は入手できるので、以下の式を用いて、逐次に、「状態」を推定していきます。

補正後の状態　＝

補正前の状態 ＋ カルマンゲイン ×（本物の観測値－予測された観測値）

「本物の観測値」と「予測された観測値」が大きく離れていれば、大きく補正さます。

カルマンゲインは、分母に(予測値の分散)^{2}と(観測値の分散)^{2}の合計、分子に

(予測値の分散)^2　ですので、予測値の分散が大きいと、カルマンゲインが大きくなり、大きく補正されます。逆に分散が小さいとカルマンゲインが小さくなり、補正が入らないことになります。

　数式では以下となります。

$p(z^{(t)} \mid X_t) = N(z^{(t)} \mid \mu_t, V_t)$　の分布で、

A, C, Q, R　は推定済です。

$\mu_t = A\,\mu_{t-1} + K_t(x^{(t)} - CA\mu_{t-1})$

$V_t = (I_m - K_t C)Q_{t-1}$

$K_t = Q_{t-1}\,C^t\,(R + CQ_{t-1}\,C^t)^{-1}$　　-→ カルマンゲイン

これにより、各 t　における　逐次状態を推定します。

2.2.4　異常度の算出

　今回のデータは、正規分布であるので、μ（平均）と　λ（分散）が決まります。

値 x の異常度は、

$$a(x) = \left(\frac{x-\mu}{\sigma}\right)^2$$

で　算出します。

平均値からの離れ具合を分散（式では標準偏差）で割って 2 乗する　ということです。

3　プログラムでの実践

データロード

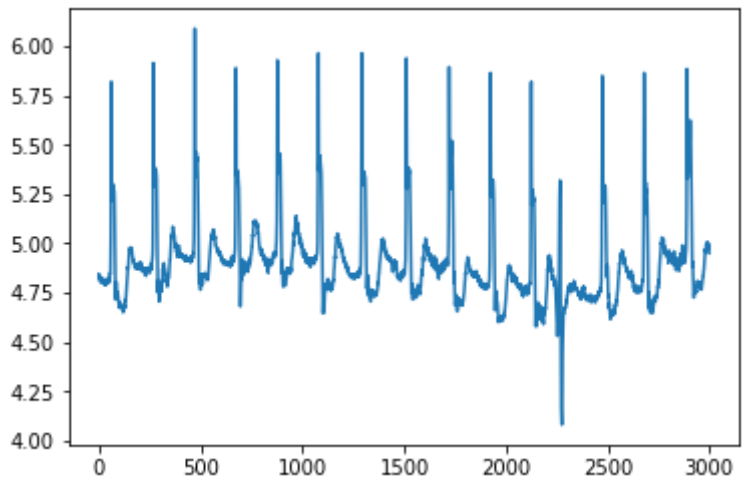
```
import urllib.request
import numpy as np
from numpy.linalg import svd, inv, matrix_rank
import matplotlib.pyplot as plt
from scipy.linalg import sqrtm

url =  "http://www.cs.ucr.edu/~eamonn/discords/qtdbsel102.txt"
f_name = "data.txt"

# ダウンロードを実行
urllib.request.urlretrieve(url, f_name)

# データ抽出
data = np.loadtxt(f_name, delimiter="¥t")
data = data[2000:5000,1]

plt.plot(data)
plt.show()
```

ライブリーを紹介します。

Numpy：効率的な数値計算を行うための型付きの多次元配列をサポート

Scipy：数値解析ソフトウェア・ライブラリ、linalg: 線形代数ルーチン、sqrtm:平方根

今回、試すのは心拍数のデータです。（よくチュートリアルで使用されます）



データ作成

Xp は時間的に前のデータを格納した行列、Xf は後のデータを格納した行列です。

```
w = 50 # 抽出窓の幅
m = 2  # z の次元
T = 1000 # 学習区間

x, x_after = [], []

for i in range(w, T):
    x.append(data[i-w: i])
    x_after.append(data[i: i+w])

x_p = np.array(x).T
x_f = np.array(x_after).T
```

x は （0 から 50 番目）------- (950 から 1000 番目)

x_after は （50 から 100 番目）------- (1000 から 1050 番目) のデータです。

部分空間同定法

パラメータ A,Q,C,R を部分空間同定法で求めます。

まずは、W を求めます。 X_p^T の T は転置行列の意味です。

$S_{pf} = X_p X_f^T$

$S_{pp} = X_p X_p^T$

$S_{ff} = X_f X_p^T$

W = $S_{pp}^{-1/2} S_{pf} S_{ff}^{-1/2}$

$X^{-1/2}$ は $X^{1/2}$ の逆行列を表します。

@ は 行列積、.T は 転置、 sqrtm は行列の平方根、 inv は逆行列 を求めます。

```
s_pf = x_p @ x_f.T
s_pp = x_p @ x_p.T
s_ff = x_f @ x_f.T

# 行列 W
W = inv(sqrtm(s_pp)) @ s_pf @ inv(sqrtm(s_ff))
```

次に、W について特異値を求めます。

そして、特異値が大きい順に m 本の左右特異ベクトルを求めます。

```
# 左右特異ベクトル
left, _, right = svd(W)
```

左特異ベクトルを特異値の大きい順に $\tilde{\alpha}^1,\ldots,\tilde{\alpha}^m$

右特異ベクトルを特異値の大きい順に $\tilde{\beta}^1,\ldots,\tilde{\beta}^m$ とし、 $\tilde{\alpha}^i$ $\tilde{\beta}^i$を求めます。

$\alpha^i = S_{pp}^{-1/2}\ \tilde{\alpha}^i$

$\beta^i = S_{ff}^{-1/2}\ \tilde{\beta}^i$

（～は不偏推定量を意味します）

```
# 特異ベクトル m 本抽出
alpha, beta = [], []
for i in range(m):
    alpha.append(inv(sqrtm(s_pp)) @ left[:,i])
    beta.append(inv(sqrtm(s_ff)) @ right[i,:])
```

次に Z_p , Z_fを求めます。

$Z_p = [\ \alpha^1\ ,\cdots,\ \alpha^m\ ]^T\ X_p$

$Z_f = [\ \beta^1\ ,\cdots,\ \beta^m\ ]^T\ X_f$

```
# z を算出
z_p = np.array(alpha) @ x_p
z_f = np.array(beta) @ x_f
```

ωを検出窓の大きさとして、以下の値を求めます。

X = $[x^{(1)}, \cdots, x^{(T-w)}]$

Z = $[z^{(1)}, \cdots, z^{(T-w)}]$

$Z_+ = [z^{(2)}, \cdots, z^{(T-w)}]$

$Z_- = [z^{(1)}, \cdots, z^{(T-w-1)}]$

最後に A,Q,C,R を求めます。

A = $(Z_+ Z_-^T)\ (Z_- Z_-^T)^{-1}$

Q = $\frac{1}{T-w-1}(Z_+ - AZ_-)\ (Z_+ - AZ_-)^T$

C = $(XZ^T)(ZZ^T)^{-1}$

R = $\frac{1}{T-w}$(X-CZ) $(X - CZ)^T$

```
# A, Q, C, R
X = np.copy(x_p)
Z = np.copy(z_p)
Z_plus = z_p[:,1:]
Z_minus = z_p[:,:-1]

A = (Z_plus @ Z_minus.T) @ inv(Z_minus @ Z_minus.T)
Q = ((Z_plus - A @ Z_minus) @ (Z_plus - A @ Z_minus).T)/(T-w-1)
C = (X @ Z.T) @ inv(Z @ Z.T)
R = ((X - C @ Z) @ (X - C @ Z).T)/(T-w)
```

カルマンフィルター

z(t−1)を以下の式で置き換え、

p($z^{(t-1)}$ | X_{t-1}) = N($z^{(t-1)}$ | μ_{t-1} ,V_{t-1})

次の漸化式で μt,Qt を推定します。

K_t = Q_{t-1} C^T (R + CQ_{t-1} C^T)$^{-1}$

μ_t = Aμ_{t-1} + K_t($x^{(t)}$ - C Aμ_{t-1})

V_t = (I_m - K_tC)Q_{t-1}

Q_t = Q + AV_t A^T

```
def update(A, Q, C, R, q, x, mu, m):
    k_t = q @ C.T @ inv(R + C @ q @ C.T)
    mu_t = A @ mu + k_t @ (x - C @ A @ mu)
    v_t = (np.eye(m) - k_t @ C) @ q
    q_t = q + A @ v_t @ A.T
   return mu_t, q_t
```

ただし、μ0,Q0 の初期値は適当に与えます。

```
q = np.eye(m)
mu = np.zeros(m)
```

異常値

異常値 a は以下の式で定義されます。

a($x^{(t)}$) = ($x^{(t)}$ - CAμ_{t-1})^T $\sum_t^{-1}$($x^{(t)}$ - CAμ_{t-1})

$\sum_t$= R + CQ_{t-1} C^T

```
def anomaly_score(A, C, R, q, x, mu):
    sigma = R + C @ q @ C.T
    predict = C @ A @ mu
    score = (x - predict).T @ sigma @ (x - predict)
    return [score, predict, sigma]
```

結果

最後に異常検知を行います。

for i in range(2000):　　—>　0から1999まで

```
result_score, result_predict, result_sigma = [], [], []
begin = 950
end = 1000

for i in range(2000):
    # t-1
    x = data[begin + i: end + i]
    mu, q = update(A, Q, C, R, q, x, mu, m)

    # t
    x = data[begin + i + 1: end + i + 1]
    result = anomaly_score(A, C, R, q, x, mu)

    result_score.append(result[0])
    result_predict.append(result[1])

    sigma_predict = np.array(result[2])
    sigma_predict = sigma_predict[-1, -1] - sigma_predict[-1,:-1] @ 文字列続く
                        inv(sigma_predict[:-1,:-1]) @ sigma_predict[-1,:-1].T
    result_sigma.append(np.max([sigma_predict, 0]))
```

グラフを表示します。

```
result_score = np.array(result_score)
result_predict = np.array(result_predict)
result_sigma = np.array(result_sigma)

#boundary_upper = result_predict[:,-1] + 2*np.sqrt(result_sigma)
#boundary_lower = result_predict[:,-1] - 2*np.sqrt(result_sigma)

plt.figure(figsize=(12,12))
plt.subplot(2,1,1)
plt.plot(data[1000:3000], label="true")
#plt.fill_between(np.arange(2000), boundary_upper, boundary_lower,
facecolor='y',alpha=0.3)
plt.plot(result_predict[:,-1], label ="predict")
#plt.xlim(1300,1400)
plt.legend()

plt.subplot(2,1,2)
plt.plot(result_score, label="score")
#plt.xlim(1300,1400)
plt.legend()
plt.show()
```