Python と時系列分析

１．はじめに

Python チュートリアルに参加する機会がありました。時系列分析を、今までは、R でプログラミングしてましたが、 今回、Python でプログラミングすることを考えました。

本報告書は、時系列分析を例に、Python を紹介するという意図も含めて、作成しています。

（同種の報告書は、継続して、作成するつもりです。）

1.1 Python について

導入として、 先駆者の教えをまとめました。（チュートリアルで耳にした内容です）

・「何ができるか」に重点を置いて、「どうやってできるか」は考えない。

外国語の取得ではないので、やってるうちに、基本的文法は覚える。

・検索力をつける。（プログラミングで非常に重要）

自力の考えで、解決できる問題はわずかしかない。

・Coogle Brain(人工知能チーム)の 15 分ルール

最初の 15 分は自分自身で解決を試みる。 （他人の時間を無駄にするので）

15 分後解決していなかったら必ず人に聞く。 （自分の時間を無駄にするので）

＊初学者は、とりあえず、すぐ聞く。

・ある程度、勉強したら、

作りたいもの、やりたいことを、まず決める。

一つの目的に役立つ技術を習得する。やみくもな勉強はしない。

以上 含蓄ある教えであります。

さて、今回 紹介することは、以下です。

「Python は豊富なライブラリを備えている」と言いますが、

構成として右の様に、考えられています。

ライブラリ（**標準ライブラリ**、**外部ライブリ**）

→ パッケージ

→ モジュール

→ クラス、**関数**

今回 ライブラリ と 関数 について、時系列分析のプログラムの中で、紹介します。

1.2 時系列分析　（状態空間モデル）

以前の技術報告書で「放電加工機のメッセージデータで異常検知を行う」という内容を報告しました。今回も題材は、同じデータを使用します。

今回は、状態空間モデルという手法で、異常検知を行います。この手法は、前回の Box-Jenkins 法をベースにした ARMA モデルとは異なり、モデル化の自由度が増えますが、反面、分析ルールは定まっていないので、その分は難解です。また、差分をとる前処理が不要、欠損値があっても分析可能等のメリットもあります。

手法のステップは以下です。

①状態空間モデルの表現

状態　＝　前時点の状態を用いた予測値　＋　過程誤差　　（状態方程式）

観測値　＝　状態　＋　観測誤差　　（観測方程式）

観測値を入手して、状態を推定することを目指します。

（池から 1 日毎の釣った魚の数から、池にいる魚の総量の増減を予測するみたいなやり方です）

数式にすると

$x(t) = T(t)x(t-1) + R(t)w(t)$　$w(t)\sim N(0,Q(t))$

$y(t) = Z(t)x(t) + v(t)$　　$v(t)\sim N(0,H(t))$

で、状態空間モデルの基本形です。

単純な構造として、ローカルレベルモデルと呼ばれるものがあり、今回はこのモデルで解析します。

$T(t)=Z(t)=R(t)=1$ として、モデルを単純化しています。

$x(t) = x(t-1) + w(t)$　　$w(t)\sim N(0,\sigma(w)^2)$

$y(t) =x(t) + v(t)$　　$v(t)\sim N(0,\sigma(v)^2)$

② 過程誤差の分散$\sigma(w)^2$、 観測誤差の分散$\sigma(v)^2$ の推定

観測データから、対数尤度を求めて、最適化した値を求めます。

最適化する作業として、scipy.optimize というライブラリを使用します。

ライブラリはブラックボックスですので、どういう計算、アルゴリズムかは、プログラム内では不明です。

（scipy は数値解析のソフトで、ユーザーガイドは、web で確認できます）

（optimize は、最適化のアルゴリズムを提供します）

③ 過程誤差の分散と観測誤差の分散の推定値を用いる

観測データから、再度計算して、フィルタ化推定量と呼ぶ、状態量を算出します。

④ 平滑化

ノイズの影響を減ずるため、現在の観測値から、過去の状態を推定します。

以上です。 概略すぎて、この記述では理解できないと思います。詳細は別の機会にします。

以上の手順でプログラムを作成するのですが、 2 種類のプログラムを紹介します。

計算式をベースにプログラムしたもの と、この計算を全て包含したライブラリを使用したもの で、当然短いプログラム構文で終わるものです。（164 行のものが 33 行で終わります！）

後者のライブラリは、stasmodels.api というもので、統計モデルの推定、検定、探索ができる Python ライブラリです。その中の tsa.UnobservedComponents が 状態空間モデルを扱うライブラリになります。

この後の報告書の内容は、プログラムの説明と解析結果のグラフになります。

この報告書の Python 関連で、言いたかったことは以下です。

・Python の特徴は 豊富なライブラリにあるということ。

・ライブラリは、ブラックボックスですので、簡単ではありますが、内容の理解は別物となります。

・最近、悪意のあるライブラリも出現してるので、各種の情報を気にした方が良いようです。

（多分 web 関連のライブラリではと推察しますが）

引用・参考 文献、WEB

・時系列分析と状態空間モデルの実践 R と Stan で学ぶ理論と実際

馬場真哉 プレアデス出版

・ローカルレベルモデルを用いた時系列データに対する異常検知

https://qiita.com/hrkz_szk/items/ea082ca07460ab8b8813

・産業機械の異常状態をローカルレベルモデルで使って検知してみた

https://qiita.com/hrkz_szk/items/0b4b93367533f74e8cb9

２．プログラムの説明と解析結果

2.1 計算式ベースのもの（最適化についてはライブラリを使う）

----------プログラム----------

```
#基本ライブラリ
import numpy as np
import pandas as pd
#図形描画ライブラリ
import matplotlib.pyplot as plt
plt.style.use('seaborn-darkgrid')
from matplotlib.pylab import rcParams
#  統計モデル（最尤推定で使用）
from scipy.optimize import minimize
```

＃の後は、コメント文でプログラムではないです

import　〇〇〇　でライブラリをインポートします。
numpy, pandas ,matplotlib は標準的なものです。
scipy.optimize は　特殊かも

```
df = np.loadtxt("data0620.csv",delimiter=',')
```

data0620.csv というデータを読み込んでいます

```
print(df)
#データ（numpy の array にすること）
data_series = np.array(df)
print(data_series)
N = len(data_series)
print(N)
```

def で関数を定義してます。Kf_LocalLevel という名前で 括弧内が引数です。
引数を渡すと、関数を実行して、戻り値を得ます（return result のところ）。
関数は、プログラム内で使用します。

```
def Kf_LocalLevel(y, mu_pre, P_pre, sigma_w, sigma_v):
  #step1: forecast
  mu_forecast = mu_pre
  P_forecast = P_pre + sigma_w
  y_forecast = mu_forecast
  F = P_forecast + sigma_v
  #step2: filtering
  K = P_forecast / (P_forecast + sigma_v)
  y_residual = y - y_forecast
  mu_filter = mu_forecast + K * y_residual
  P_filter = (1-K) * P_forecast
  #store the result
  result = {
      'mu_filter' : mu_filter,
      'P_filter' : P_filter,
      'y_residual' : y_residual,
      'F' : F,
      'K' : K
  }
  return result
```

この関数は、観測値、前期の状態、前期状態の予測誤差の分散、過程誤差の分散、観測誤差の分散 を引数として、補正後の状態（フィルタ化推定量）、フィルタ化推定量の分散、観測値の予測残差、観測値の予測誤差の分散、カルマンゲインを戻り値にしています。

この関数は、過程誤差の分散、観測誤差の分散、観測値データ　を引数として、
対数尤度を戻り値にしています。

```
def cal_LogLik_LocalLevel(sigma,data_series=data_series):
  data_series = np.array(data_series)
  sigma_w = np.exp(sigma[0])
  sigma_v = np.exp(sigma[1])
  #sample size
  N = len(data_series)
  #状態の推定量
  mu_zero = 0
  mu_filter = np.hstack((mu_zero,np.zeros(N)))
  P_zero = 10000000
  P_filter = np.hstack((P_zero,np.zeros(N)))
  y_residual =   np.zeros(N)
  F =   np.zeros(N)
  K =   np.zeros(N)
  for i in range(0,N):
    result = Kf_LocalLevel(y=data_series[i],
                           mu_pre=mu_filter[i],
                           P_pre=P_filter[i],
                           sigma_w=sigma_w,
                           sigma_v=sigma_v)
    mu_filter[i+1] = result['mu_filter']
    P_filter[i+1] = result['P_filter']
    y_residual[i] = result['y_residual']
    F[i] = result['F']
    K[i] = result['K']

  LogLik = 1/2 * np.sum( np.log(F) + y_residual**2 / F )
  return LogLik
```

この関数は、ライブラリ scipy.optimize の minimize を使っています。
最適化しているのですが、内容はプログラム内ではみられません。
関数(cal_loglik_LocalLeve)を引数としています。

```
def output_sigma(initial_value=list((1,1))):
  opt_result = minimize(fun=cal_LogLik_LocalLevel, x0=initial_value, method='l-bfgs-b')
  return np.exp(opt_result.x)
```

この関数は、平滑化を行う関数で、フィルタ化推定量、フィルタ化推定量の分散、状態平滑化漸化式のパラメタ（１時点前)、状態分散平滑化漸化式のパラメタ（１時点前)、観測値の予測誤差の分散、観測値の予測残差、カルマンゲインを引数として、平滑化状態、平滑化状態分散、状態平滑化漸化式のパラメタ、状態分散平滑化漸化式のパラメタを、戻り値にしています。

```
  def smooth_LocalLevel バックスラッシュ
  (mu_filtered, P_filtered, r_post, s_post, F_post, y_residual_post, K_post):
    r = y_residual_post / F_post + (1-K_post) * r_post
    mu_smooth = mu_filtered + P_filtered * r
    s = 1/F_post + (1-K_post)**2 * s_post
    P_smooth = P_filtered - P_filtered**2 * s
    #store the result
    result = {
        'mu_smooth' : mu_smooth,
        'P_smooth' : P_smooth,
        'r' : r,
        's' : s
    }
    return result
```

ここまでで、四つの関数(def)を定義しました。
ここから、プログラム実行が開始しされます。

```
#----------過程誤差と観測誤差の推定----------#
#状態の推定量
mu_zero = 0
mu_filter = np.hstack((mu_zero,np.zeros(N)))
#状態の分散
P_zero = 10000000
P_filter = np.hstack((P_zero,np.zeros(N)))
#観測値の予測残差
y_residual =  np.zeros(N)
#観測値の予測残差の分散
F =  np.zeros(N)
#カルマンゲイン
K =  np.zeros(N)
#過程誤差の分散
sigma_w = 1000
#観測誤差の分散
sigma_v = 10000
#最尤推定
print(output_sigma())
#過程誤差の最適な分散
sigma_w = output_sigma()[0]
#観測誤差の最適な分散
sigma_v = output_sigma()[1]
#----------状態の推定----------#
for i in range(0,N):
  result = Kf_LocalLevel(y=data_series[i],
                         mu_pre=mu_filter[i],
                         P_pre=P_filter[i],
                         sigma_w=sigma_w,
                         sigma_v=sigma_v)
  mu_filter[i+1] = result['mu_filter']
  P_filter[i+1] = result['P_filter']
  y_residual[i] = result['y_residual']
  F[i] = result['F']
  K[i] = result['K']
```

状態のフィルタ化推定量の入れ物を初期値 0 で作ります。

フィルタ化推定量分散の入れ物を
初期値 1000000 で作ります。

観測値の予測残差、残差、カルマン
ゲインの入れ物を空で作ります。

仮定誤差の分散、観測誤差の分散を
1000 と 10000 に暫定します。

1 ページ前にある関数 output_sigma()
で計算を行い、sigma_w ,sigma_v を算出します。

関数 Kf_LocalLevel で再計算します

```
#----------平準化----------#
# 平滑化状態
mu_smooth = np.zeros(N + 1)
# 平滑化状態分散
P_smooth = np.zeros(N + 1)
# 漸化式のパラメタ（初期値は 0 のままでよい）
r = np.zeros(N)
s = np.zeros(N)
# 最後のデータは、フィルタリングの結果とスムージングの結果が一致する
mu_smooth[-1] = mu_filter[-1]
P_smooth[-1] = P_filter[-1]
# 逆順でループ
for i in range(N-1,-1,-1):
  result = smooth_LocalLevel(
    mu_filter[i],P_filter[i],r[i], s[i], F[i], y_residual[i], K[i]
  )
  mu_smooth[i] = result['mu_smooth']
  P_smooth[i] = result['P_smooth']
  r[i - 1] = result['r']
  s[i - 1] = result['s']
    ########################
rcParams['figure.figsize'] = 15,5
plt.plot(df, label='Observed')
plt.plot(mu_smooth, label='Predicted')
plt.legend(loc='upper right', borderaxespad=0, fontsize=15)
plt.show()
##################################
sigma = P_smooth[1:] + F
anomaly_detection = np.zeros(N-1)
for i in range(1,N-1):
  anomaly_detection[i] = (df[i] - mu_smooth[i-1])*sigma[i]

plt.plot(anomaly_detection)
plt.title('anomaly detection')
plt.show()
```

平滑化します。

関数 smooth_Locallevel で計算を行い、平滑化します。

グラフ表示のプログラムです。
観測値データと円滑化したフィルタ化推定量のグラフを表示します。

異常検知と
そのグラフ表示のプログラムです。

2.2 ライブラリ tsa.UnobservedComponents を利用したもの

プログラム６ページ分が１ページになります。

プログラムは以下です。

```
#基本ライブラリ
import numpy as np
import pandas as pd
#図形描画ライブラリ
import matplotlib.pyplot as plt
plt.style.use('seaborn-darkgrid')
from matplotlib.pylab import rcParams
import statsmodels.api as sm
data = np.loadtxt("data0620.csv",delimiter=',')
print(data)

mod_local_level = sm.tsa.UnobservedComponents(data, 'local level')
res_local_level = mod_local_level.fit()
print(res_local_level.summary())

####################################
rcParams['figure.figsize'] = 15,5
plt.plot(data, label='Observed')
plt.plot(res_local_level.fittedvalues[1:], label='Predicted')
plt.legend(loc='lower right', borderaxespad=0, fontsize=15)
plt.show()
##############################
sigma = res_local_level.params[0] + res_local_level.params[1]

anomaly_detection = (data - res_local_level.fittedvalues[0:])*sigma

plt.plot(anomaly_detection)
plt.title('anomaly detection')
plt.show()
```

Import stasmodels.api as sm
が tsa.UnobservedComponents を
含む ライブラリ です。

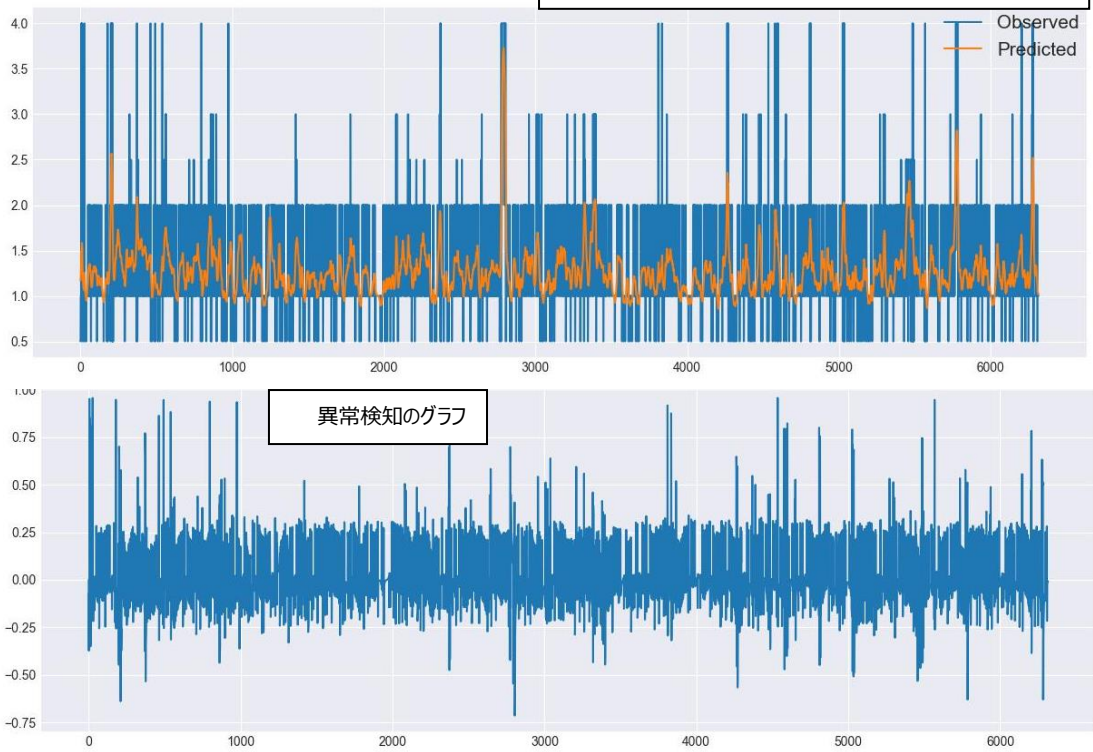
この３行で計算は終わりです。

グラフ表示と異常検知を行ってます

3,グラフ表示

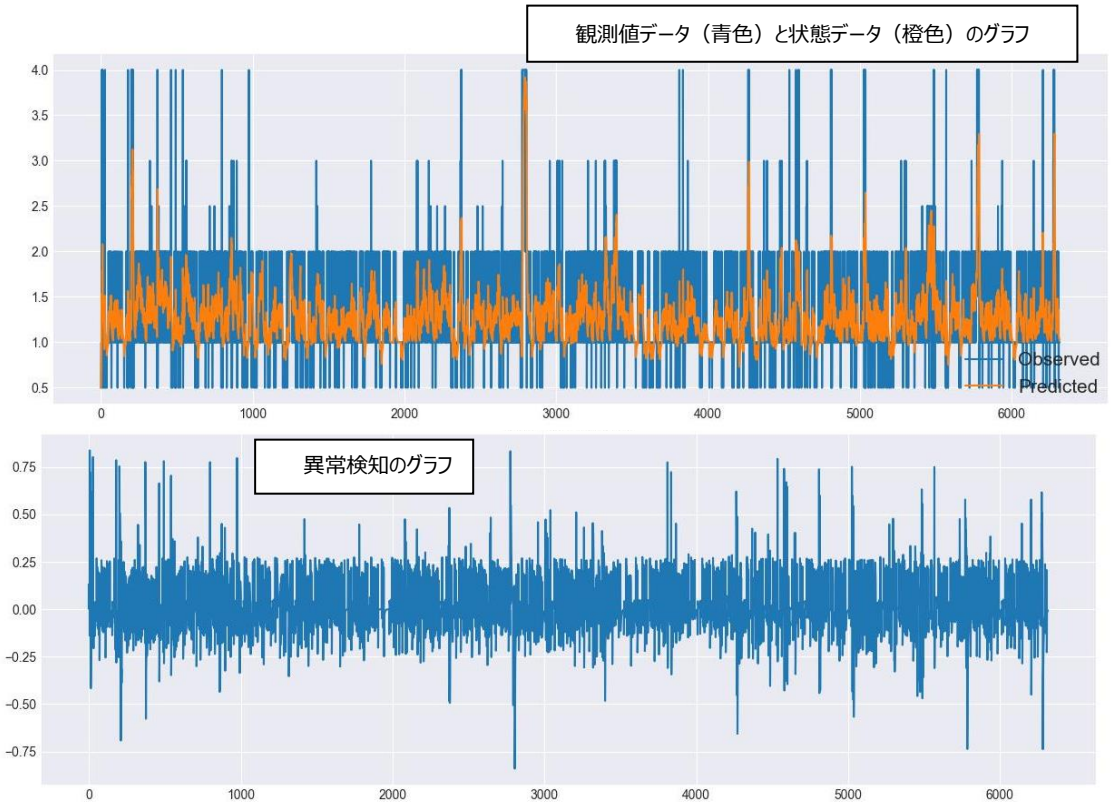
3.1 計算式ベースのプログラムで作成したグラフ

観測値データ（青色）と状態データ（橙色）のグラフ



3.2 ライブラリ　tsa.UnobservedComponents　を利用したプログラムのグラフ

観測値データ（青色）と状態データ（橙色）のグラフ



3　グラフの評価と今後の取組

ライブライの違う二つのプログラムで実行しました。グラフは同等のものがプロットされました。

ただ、平滑化したつもり状態データは、平滑化されたようには見えないし

異常検知の突出したレベルが検出されていません。（異常無しと判定されたのかもしれませんが）

観測データを、何かしら処理すべきなかもしれませんし、簡略化されたローカルレベルモデルでは、

無理であり、モデル化を見直す必要があるかもしれません。

状態空間モデルを更に学んで、使えるものしようと考えています。

（適切なモデル化　とか　異常検知の手法について深堀りが必要なようです。）

おまけ　その 1 ですが、pykalman というカルマンフィルタライブライを使っても、

フィルタ化推定量　を　算出できるようです。

おまけ　その 2 ですが、Python は　クラス　という型があるので、紹介します。

クラスは、同じ変数や関数を共有するオブジェクトを作るのに使います。

関数がクラスに含まれていると　メソッド　と呼び、変数が含まれていると、属性　と呼びます。

（Python　は　オブジェクト指向のプログラミングといわれますが、クラスはその典型です。）

（オブジェクトは‘対象’くらいの意味合いかと思っています）

簡単なプログラムを以下に書きます。

```
class Cat(object):
    def __init__(self, name, weight):
        self.name = name
        self.weight = weight
mike = Cat("MIKE", 4.5)

print(mike.weight)
  #   4.5 と出力されます
print(mike.name)
   #   MIKE  と出力されます
```

Cat というクラスを定義しました。

名前と体重を入力して、mike とします

Mike の体重を出力しました。

Mike の名前を出力しました。