KFAS ライブラリの利用法を示す.

R の全コードは 1-kfas-sample.R を，サンプルデータは ssm-sample-data.csv を参照のこと．なお，本資料におけるデータはすべてシミュレーションで生成された模擬データである．また，本資料では説明の簡単のため 12 時点で 1 周期と想定してデータを分析する．

## ライブラリ読み込み

ライブラリを読み込む．本資料において灰色の網掛け部は，実装コードを表す.

KFAS ライブラリは状態空間モデルを推定するためのライブラリであり，本研究と同じ分析を再現するために必須である．以下は必須ではないが利用することを推奨する．dplyr はデータの変換などの処理を，lubridate は日付処理を，forecast は予測精度の評価などを行うためのライブラリである．

```
library(KFAS)
library(dplyr)
library(lubridate)
library(forecast)
```

本資料で用いたライブラリのバージョンは以下の通りである．本資料において白背景のコードはコンソールへの出力を表す.

```
> packageVersion("KFAS")
[1] ‘1.5.1’
> packageVersion("dplyr")
[1] ‘1.1.4’
> packageVersion("lubridate")
[1] ‘1.9.3’
> packageVersion("forecast")
[1] ‘8.21.1’
```

## データ読み込み

サンプルデータを読み込む.

```
> # CSV ファイルの読み込み(シミュレーションで生成された模擬データ)
> sample_data <- read.csv(file = "ssm-sample-data.csv")
>
> # time      ：日付(論文では 0.5 か月だが，本サンプルコードでは月単位データとした)
> # temp      ：水温
```

```
> # distance  ：黒潮流軸までの距離
> # kuroshio_a：黒潮流路が A 型のときに 1 をとるフラグ
> head(sample_data)
       time  temp distance kuroshio_a
1 1998-01-01 16.19      116         0
2 1998-02-01 13.82      148         0
3 1998-03-01 16.44       49         0
4 1998-04-01 16.49      131         0
5 1998-05-01 19.70       45         0
6 1998-06-01 21.83       40         0
```

秋冬の黒潮 A フラグ（論文中の$WinterA_t$に該当する）を作成する．

```
> # 秋冬の黒潮 A フラグの追加
> sample_data <-
+   sample_data %>%
+   mutate(winter = as.numeric(month(as.Date(time)) %in% c(11, 12, 1, 2, 3)),
+          winter_a = winter * kuroshio_a)
> # winter  ：秋冬に 1 をとるフラグ
> # winter_a：秋冬かつ，黒潮が A であるときに 1 をとるフラグ
> head(sample_data)
       time  temp distance kuroshio_a winter winter_a
1 1998-01-01 16.19      116         0      1        0
2 1998-02-01 13.82      148         0      1        0
3 1998-03-01 16.44       49         0      1        0
4 1998-04-01 16.49      131         0      0        0
5 1998-05-01 19.70       45         0      0        0
6 1998-06-01 21.83       40         0      0        0
```

ts 型に変換する．必須ではないが，KFAS ライブラリを利用する場合は ts 型にするのが簡便である．

```
> # ts 型に変換
> # 0.5 か月単位データの場合は frequency = 24 と指定する
> sample_ts <-
+   sample_data %>%
+   select(-time, -winter) %>%            # 日付列と秋冬フラグを削除
+   ts(start = c(1998, 1), frequency = 12) # ts 型に変換(1998 年 1 月開始．12 か月 1 周期)
```

```
>
> # 1998年12月までのデータを取得
> window(sample_ts, end = c(1998, 12))
         temp distance kuroshio_a winter_a
Jan 1998 16.19      116         0       0
Feb 1998 13.82      148         0       0
Mar 1998 16.44       49         0       0
Apr 1998 16.49      131         0       0
May 1998 19.70       45         0       0
Jun 1998 21.83       40         0       0
Jul 1998 21.91      101         0       0
Aug 1998 24.34      119         0       0
Sep 1998 20.12      137         0       0
Oct 1998 19.77       27         0       0
Nov 1998 18.89       97         0       0
Dec 1998 17.80      149         0       0
```

`window(sample_ts, end = c(1998, 12))`とすることで，1998年12月までのデータを取得できる．結果は略するが，`autoplot(sample_ts, facets = TRUE)`とすることで，時系列折れ線グラフを描くことができる．

## 訓練データと検証データに分割

2021年12月までを訓練データと，2022年1月以降を検証データとする．

```
> # 訓練データ
> train <- window(sample_ts, end = c(2021, 12))
>
> # 検証データ
> validate <- window(sample_ts, start = c(2022, 1))
```

## モデル化を行う関数の作成

モデル化の方法はHelske(2017)及び馬場(2018)，野村(2016)に詳しい．

`SSModel`関数を用いて，モデルの構造を決定する．

推定すべきパラメータ，すなわち観測誤差`H`，過程誤差`Q`には`NA`を指定する．

`temp ~`と指定することで，海水温`temp`を予測するモデルとなる．

`SSMtrend`関数を用いることでトレンドの構造を指定する．本資料では平滑化トレンドモ

デル（2 階差分のトレンド）を指定した．

SSMseasonal 関数を用いることで，季節成分の構造を指定する．本資料ではダミー変数を用いる確率的季節成分を指定した．論文中では 24 時点で 1 周期であるが，本資料では 12 時点で 1 周期とする．周期を変更する場合は SSMseasonal の引数 period を変更する

SSMarima 関数を用いることで，定常 AR 成分の構造を指定する．本資料では 2 次の AR 成分とした．d は和分の次数の指定であり，定常 AR 成分の場合は 0 を指定する．

最後に+ distance + kuroshio_a + winter_a とすることで，外生変数をモデルに加えている．

訓練データは data = ts_data のように指定する．以下はサンプルコードである．

```
build_ssm <- SSModel(
  H = NA,
  temp ~
    SSMtrend(degree = 2,                  # 平滑化トレンドモデル
             Q = c(list(0), list(NA))) +
    SSMseasonal(
      sea.type = "dummy", # ダミー変数を利用した季節成分
      period = 12,        # 周期は 12 とする
      Q = NA
    ) +
    SSMarima(
      ar = c(0, 0),       # 2 次の AR 成分
      d = 0,
      Q = 0
    ) + distance + kuroshio_a + winter_a, # 外生変数
  data = ts_data
)
```

定常 AR 成分を持つモデルの場合は，AR モデルのパラメータの推定に工夫が必要である．例えば 1 次の自己回帰項のパラメータの絶対値が 1 以上だと非定常な AR 過程となってしまう．KFAS ライブラリでは artransform という関数が提供されているため，これを利用する．この場合は，以下のように update_func を用意する．

update_func はほぼモデルの構造を指定するコードと同じであるが，分散の推定結果が負にならないように exp を適用したり，自己回帰項のパラメータに artransform を適用したりしている．

```
update_func <- function(pars, model) {
  model <- SSModel(
    H = exp(pars[6]),
    temp ~
      SSMtrend(degree = 2,
               Q = c(list(0), list(exp(pars[1])))) +
      SSMseasonal(
        sea.type = "dummy",
        period = 12,
        Q = exp(pars[2])
      ) +
      SSMarima(
        ar = artransform(pars[3:4]),
        d = 0,
        Q = exp(pars[5])
      ) + distance + kuroshio_a + winter_a,
    data = ts_data
  )
}
```

パラメータは以下のように fitSSM 関数を用いることで推定する．パラメータの初期値は任意である．推定の際にエラーが出る場合などは，適宜初期値を修正する．

```
fit_ssm <- fitSSM(
  build_ssm,
  inits = c(-17,-30, 0.5, 0, -1, -3), # パラメータの初期値(任意)
  update_func,
  method = "Nelder-Mead",
  control = list(maxit = 5000, reltol = 1e-16)
)
```

パラメータの初期値の設定に比較的敏感であるため，Helske(2017)では 1 回目に推定されたパラメータを，2 回目の推定の際の初期値に利用している．

パラメータの推定結果に対して，以下のように KFS 関数を適用することで，フィルタ化推定量と平滑化推定量を得ることができる．

```
result_ssm <- KFS(
  fit_ssm$model,
```

```
    filtering = c("state", "mean"),
    smoothing = c("state", "mean", "disturbance")
  )
```

連続する上記の処理を 1 つの関数にまとめる．また，Helske(2017)にならい，1 回目に推定されたパラメータを，2 回目の推定の際の初期値に利用するように修正したコードを以下に示す．

```
# モデルを作る関数(黒潮流路のデータ利用)
make_ssm_kuroshio <- function(ts_data) {
  # モデルの構造を決める
  build_ssm <- SSModel(
    H = NA,
    temp ~
      SSMtrend(degree = 2,                # 平滑化トレンドモデル
               Q = c(list(0), list(NA))) +
      SSMseasonal(
        sea.type = "dummy", # ダミー変数を利用した季節成分
        period = 12,        # 周期は 12 とする
        Q = NA
      ) +
      SSMarima(
        ar = c(0, 0),       # 2 次の AR 成分
        d = 0,
        Q = 0
      ) + distance + kuroshio_a + winter_a, # 外生変数
    data = ts_data
  )

  # optim に渡す前にパラメータを exp したり artransform したり，変換する
  # ほぼ build_ssm と同じだが，パラメータだけ変更されている
  update_func <- function(pars, model) {
    model <- SSModel(
      H = exp(pars[6]),
      temp ~
        SSMtrend(degree = 2,
                 Q = c(list(0), list(exp(pars[1])))) +
```

```
      SSMseasonal(
        sea.type = "dummy",
        period = 12,
        Q = exp(pars[2])
      ) +
      SSMarima(
        ar = artransform(pars[3:4]),
        d = 0,
        Q = exp(pars[5])
      ) + distance + kuroshio_a + winter_a,
    data = ts_data
  )
}

# 最適化その 1. まずは Nelder-Mead 法を用いて暫定的なパラメータを推定
fit_ssm_bef <- fitSSM(
  build_ssm,
  inits = c(-17,-30, 0.5, 0, -1, -3), # パラメータの初期値(任意)
  update_func,
  method = "Nelder-Mead",
  control = list(maxit = 5000, reltol = 1e-16)
)

# 最適化その 2. 先ほどの結果を初期値に使ってもう一度最適化する
fit_ssm <- fitSSM(
  build_ssm,
  inits = fit_ssm_bef$optim.out$par,
  update_func,
  method = "BFGS",
  control = list(maxit = 5000, reltol = 1e-16)
)

# フィルタリングとスムージング
result_ssm <- KFS(
  fit_ssm$model,
  filtering = c("state", "mean"),
```

```
    smoothing = c("state", "mean", "disturbance")
  )

  # 結果の出力
  return(list(fit_ssm, result_ssm))

}
```

上記のコードから+ distance + kuroshio_a + winter_a を取り除いたもの（外生変数を除いたもの）を make_ssm_without という関数として用意しておく．

## モデル化

訓練データに対してモデルを当てはめる．fit_xxx はパラメータの推定結果であり，result_xxx にフィルタ化推定量と平滑化推定量が格納されている．

```
# 黒潮流路のデータ利用
list_kuroshio <- make_ssm_kuroshio(train)
fit_kuroshio    <- list_kuroshio[[1]]
result_kuroshio <- list_kuroshio[[2]]

# 黒潮流路未使用
list_without <- make_ssm_without(train)
fit_without    <- list_without[[1]]
result_without <- list_without[[2]]
```

## 推定結果

平滑化推定量は alphahat に格納されている．最初の 3 つの状態は外生変数のパラメータとなっている．

```
> # 係数
> result_kuroshio$alphahat[1, 1:3]
    distance   kuroshio_a     winter_a
-0.006811364  0.310578063  0.749976947
```

パラメータの 95%信頼区間を得る場合は，confint 関数を使う．

```
> # 係数の 95%信頼区間
> res <- confint(result_kuroshio, level = 0.95)
```

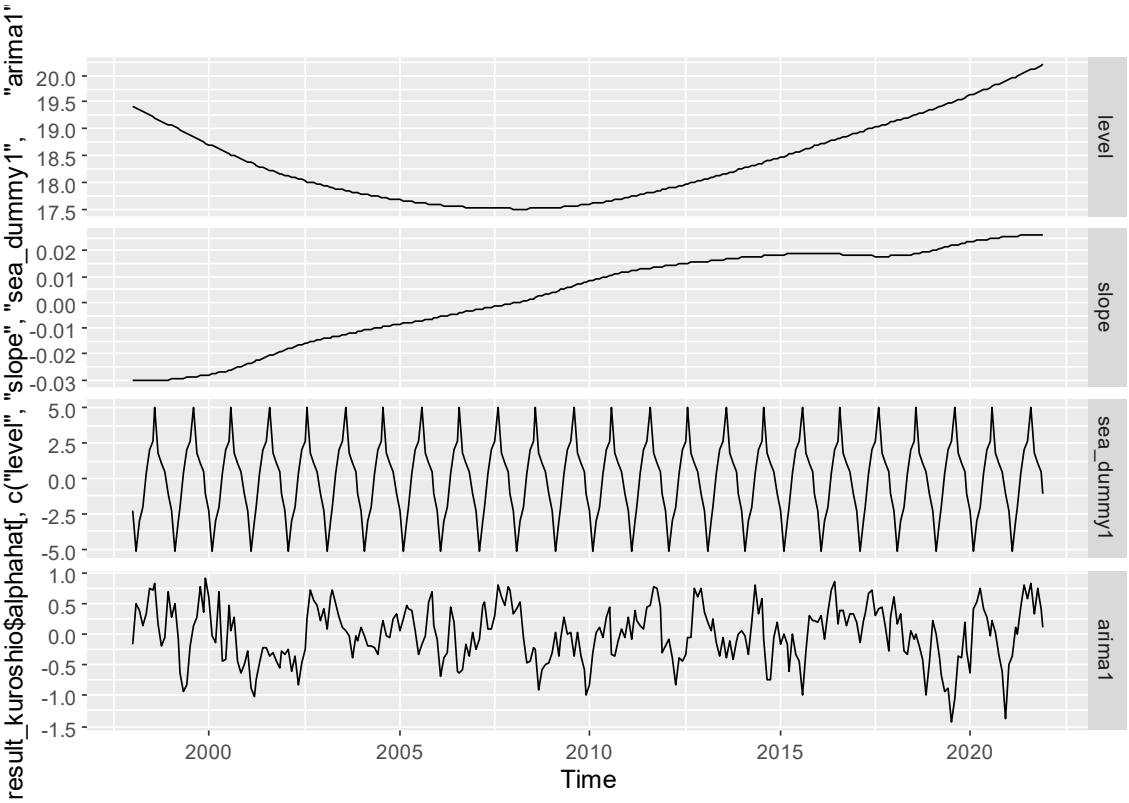
```
> res[c("distance", "kuroshio_a", "winter_a")] %>% lapply(head, n = 1)
$distance
                 lwr          upr
Jan 1998 -0.008200424 -0.005422304


$kuroshio_a
                lwr       upr
Jan 1998 -0.03786653 0.6590227


$winter_a
              lwr      upr
Jan 1998 0.4585907 1.041363
```

平滑化推定量は以下のようになっている. なお level が水準成分, slope がドリフト成分, sea_dummy1 が季節成分, arima1 が定常 AR 成分の平滑化推定量である.

```
# 状態の可視化
autoplot(
  result_kuroshio$alphahat[, c("level", "slope", "sea_dummy1", "arima1")],
  facets = TRUE
)
```




## 予測

予測の際には predict 関数を利用する.

外生変数を用いないモデル result_without の予測を行う場合は, 以下のように引数 n.ahead を指定するのが簡単である.

```
> # 簡単な方法
> predict(result_without$model, n.ahead = 14)
         Jan      Feb      Mar      Apr      May      Jun      Jul      Aug      Sep
2022 18.83750 15.78001 18.00774 18.76232 21.14003 22.84697 23.34855 25.80887 22.52110
2023 18.95388 15.95764
         Oct      Nov      Dec
```

```
2022 21.85236 21.53284 19.99911
2023
```

n.ahead の代わりに引数 newdata を指定することもできる．

引数 newdata を指定する場合，モデルの構造を指定する際とほぼ同様に SSModel 関数を用いてモデルの構造を記述する．ただし，パラメータには推定された結果を格納する．また，予測を行うため，将来の水温については NA を予測したい期間だけ指定しておく．

```
# 推定されたパラメータ
pars <- fit_without$optim.out$par


# newdata を用いた予測
pred_without <- predict(
  result_without$model,
  newdata = SSModel(
    H = exp(pars[6]),
    rep(NA, nrow(validate)) ~
      SSMtrend(degree = 2,
               Q = c(list(0), list(exp(pars[1])))) +
      SSMseasonal(sea.type = "dummy",
                  period = 12,
                  Q = exp(pars[2])) +
      SSMarima(ar = artransform(pars[3:4]),
               d = 0,
               Q = exp(pars[5])),
    data = validate
  )
)
```

今回は検証データである validate を newdata の作成の際に利用した．精度の検証のために将来の海水温がわかっているという想定で newdata を作ったのであるが，本来的な将来予測を行う場合は，検証データが存在しないだろう．

予測の際に validate の temp 列は利用されていない．そのため，以下のように作成された target データを利用して new_data を作成しても同じ結果が得られる．

```
target <- validate
target[, "temp"] <- NA
```

なお引数 n.ahead を利用しても，引数 newdata を利用しても，予測結果はまったく変わらない．

```
> all(pred_without == predict(result_without$model, n.ahead = 14))
[1] TRUE
```

外生変数があるモデル result_kuroshio を対象として予測を行う場合は，引数 newdata を利用することが必須である．仮に引数 n.ahead を指定すると Model contains time varying system matrices, cannot use argument 'n.ahead'. Use 'newdata' instead.というエラーが出る．

引数 newdata を指定して予測を行う．SSModel 関数の中はほぼモデルの推定コードと等しい．

```
# newdata の利用
pars <- fit_kuroshio$optim.out$par


pred_kuroshio <- predict(
  result_kuroshio$model,
  newdata = SSModel(
    H = exp(pars[6]),
    rep(NA, nrow(validate)) ~
      SSMtrend(degree = 2,
               Q = c(list(0), list(exp(pars[1])))) +
      SSMseasonal(sea.type = "dummy",
                  period = 12,
                  Q = exp(pars[2])) +
      SSMarima(ar = artransform(pars[3:4]),
               d = 0,
               Q = exp(pars[5])
      ) + distance + kuroshio_a + winter_a,
    data = validate
  )
)
```

predict 関数の詳細は?predict.SSModel としてヘルプなども参照されたい．

予測値は以下の通りである．

```
> # 予測値
> pred_kuroshio
```

```
         Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
2022 18.99910 15.96967 17.89792 18.07057 20.66269 22.64790 23.04583 25.61873
2023 18.99250 16.21390
         Sep      Oct      Nov      Dec
2022 22.04893 21.52715 21.78005 20.22480
2023
> pred_without
         Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
2022 18.83750 15.78001 18.00774 18.76232 21.14003 22.84697 23.34855 25.80887
2023 18.95388 15.95764
         Sep      Oct      Nov      Dec
2022 22.52110 21.85236 21.53284 19.99911
2023
>
```

予測精度は forecast パッケージの accuarcy 関数を利用して計算するのが容易である.

```
> # 予測精度
> accuracy(pred_kuroshio, validate[, "temp"])
                ME      RMSE       MAE       MPE     MAPE       ACF1 Theil's U
Test set -0.2656954 0.3635745 0.2804646 -1.296928 1.383106 -0.1269102 0.1778875
> accuracy(pred_without,  validate[, "temp"])
                ME      RMSE       MAE       MPE     MAPE      ACF1 Theil's U
Test set -0.3834954 0.5720084 0.4796726 -1.825898 2.375367 0.3095375 0.3010908
```

なお, MASE を計算する場合は, 上記で計算された MAE を訓練データにおけるナイーブ予測の MAE で除すことによって求める.

## 参考文献


馬場真哉（2018）「時系列分析と状態空間モデルの基礎：R と Stan で学ぶ理論と実装」. プレアデス出版, 長野, 349pp.

Helske, J. (2017) KFAS: Exponential Family State Space Models in R. Journal of Statistical Software, **78**, 1-39.

野村俊一（2016）「カルマンフィルタ ：R を使った時系列予測と状態空間モデル」. 共立出版, 東京, 154pp.