robKalman — a package on Robust Kalman Filtering

Peter Ruckdeschel[1] Bernhard Spangl[2]

[1] UNIVERSITÄT BAYREUTH Fakultät für Mathematik und Physik

Peter.Ruckdeschel@uni-bayreuth.de

www.uni-bayreuth.de/departments/math/org/mathe7/RUCKDESCHEL

[2] Universität für Bodenkultur, Wien

Bernhard.Spangl@boku.ac.at

www.rali.boku.ac.at/statedv.html

RsR Workshop at BIRS, 2007/10/30

Classical setup: Linear state space models (SSMs)

- State equation:

$$\boxed{X_t = F_t X_{t-1} + v_t}$$

- Observation equation:

$$\boxed{Y_t = Z_t X_t + \varepsilon_t}$$

- Ideal model assumption:

$$X_0 \sim \mathcal{N}_p(a_0, \Sigma_0), \quad v_t \sim \mathcal{N}_p(0, Q_t), \quad \varepsilon_t \sim \mathcal{N}_q(0, V_t),$$

  all independent
- (preliminary ?) simplification:
  Hyper parameters F_t, Z_t, V_t, Q_t constant in t

Classical setup: Linear state space models (SSMs)

- State equation:

$$\boxed{X_t = F_t X_{t-1} + v_t}$$

- Observation equation:

$$\boxed{Y_t = Z_t X_t + \varepsilon_t}$$

- Ideal model assumption:

$$X_0 \sim \mathcal{N}_p(a_0, \Sigma_0), \quad v_t \sim \mathcal{N}_p(0, Q_t), \quad \varepsilon_t \sim \mathcal{N}_q(0, V_t),$$

  all independent
- (preliminary ?) simplification:
  Hyper parameters F_t, Z_t, V_t, Q_t constant in t

Kalman filter

- goal: reconststruct X_t by means of $Y_s, s \le t$
- practical reason: restriction to **linear** procedures / Gaussian assumptions $\rightsquigarrow$ classical **Kalman Filter**

0. Initialization $(t = 0)$:

$$X_{0|0} = a_0, \quad \Sigma_{0|0} = \Sigma_0$$

1. Prediction $(t \ge 1)$:

$$X_{t|t-1} = FX_{t-1|t-1}, \quad \mathrm{Cov}(X_{t|t-1}) = \Sigma_{t|t-1} = F\Sigma_{t-1|t-1}F' + Q$$

2. Correction $(t \ge 1)$:

$$\begin{aligned} X_{t|t} &= X_{t|t-1} + K_t(Y_t - ZX_{t|t-1}) \\ K_t &= \Sigma_{t|t-1}Z'(Z\Sigma_{t|t-1}Z' + V)^{-}, \qquad \text{(Kalman gain)} \\ \mathrm{Cov}(X_{t|t}) &= \Sigma_{t|t} = \Sigma_{t|t-1} - K_t Z \Sigma_{t|t-1} \end{aligned}$$

Kalman filter

- goal: reconststruct X_t by means of $Y_s, s \le t$
- practical reason: restriction to **linear** procedures / Gaussian assumptions $\rightsquigarrow$ classical **Kalman Filter**

0. Initialization ($t = 0$):

$$X_{0|0} = a_0, \quad \Sigma_{0|0} = \Sigma_0$$

1. Prediction ($t \ge 1$):

$$X_{t|t-1} = FX_{t-1|t-1}, \quad \mathrm{Cov}(X_{t|t-1}) = \Sigma_{t|t-1} = F\Sigma_{t-1|t-1}F' + Q$$

2. Correction ($t \ge 1$):

$$\begin{aligned} X_{t|t} &= X_{t|t-1} + K_t(Y_t - ZX_{t|t-1}) \\ K_t &= \Sigma_{t|t-1}Z'(Z\Sigma_{t|t-1}Z' + V)^-, \qquad \text{(Kalman gain)} \\ \mathrm{Cov}(X_{t|t}) &= \Sigma_{t|t} = \Sigma_{t|t-1} - K_t Z \Sigma_{t|t-1} \end{aligned}$$

Robustification

Types of outliers AO/SOs (exogeneous):

- either error ε_t is affected (AO)
- or observations Y_t are modified (SO)

Robustification considered is to

- retain recursivity (three-step approach / performance!)
- modify correction step $\rightsquigarrow$ bound influence of Y_t
- retain init./pred.step but with modified filter past $X_{t-1|t-1}$

Robustification

Types of outliers AO/SOs (exogeneous):

- either error ε_t is affected (AO)
- or observations Y_t are modified (SO)

Robustification considered is to

- retain recursivity (three-step approach / performance!)
- modify correction step $\rightsquigarrow$ bound influence of Y_t
- retain init./pred.step but with modified filter past $X_{t-1|t-1}$

Contents of package `robKalman`

- Kalman filter: filter, Kalman gain, covariances
- ACM-filter: filter, GM-estimator
- rLS-filter: filter, calibration of clipping height
- extensible to further recursive filters:
  $\rightsquigarrow$ general interface `recursiveFilter`
  with arguments:
  - data
  - state space model (hyper parameters)
  - **functions for the init./pred./corr.step**
  - control parameters

Contents of package `robKalman`

- Kalman filter: filter, Kalman gain, covariances
- ACM-filter: filter, GM-estimator
- rLS-filter: filter, calibration of clipping height
- extensible to further recursive filters:
  $\rightsquigarrow$ general interface `recursiveFilter`
  with arguments:
  - data
  - state space model (hyper parameters)
  - **functions for the init./pred./corr.step**
  - control parameters

Contents of package robKalman

- Kalman filter: filter, Kalman gain, covariances
- ACM-filter: filter, GM-estimator
- rLS-filter: filter, calibration of clipping height
- extensible to further recursive filters:
  $\rightsquigarrow$ general interface `recursiveFilter`
  with arguments:
  - data
  - state space model (hyper parameters)
  - **functions for the init./pred./corr.step**
  - control parameters

Implementation concept

- Programming language
  - completely in `S`
  - perhaps some code in `C` later
- Use existing infrastructure
  - time series classes: `ts`, `its`, `irts`, `zoo`, `zoo.reg`, `tframe`
  - for: graphics, diagnostics, management of date/time
- Split user interface and "Kalman code"
  - internal functions: no S4-objects, no time stamps
  - user interface: S4-objects, time stamps

Implementation concept

- Programming language
  - completely in `S`
  - perhaps some code in `C` later
- Use existing infrastructure
  - time series classes: `ts`, `its`, `irts`, `zoo`, `zoo.reg`, `tframe`
  - for: graphics, diagnostics, management of date/time
- Split user interface and "Kalman code"
  - internal functions: no S4-objects, no time stamps
  - user interface: S4-objects, time stamps

Implementation concept

- Programming language
  - completely in `S`
  - perhaps some code in `C` later
- Use existing infrastructure
  - time series classes: `ts`, `its`, `irts`, `zoo`, `zoo.reg`, `tframe`
  - for: graphics, diagnostics, management of date/time
- Split user interface and "Kalman code"
  - internal functions: no `S4`-objects, no time stamps
  - user interface: `S4`-objects, time stamps

Implementation so far: interfaces

- preliminary, “S4-free” interfaces
  - Kalman filter (in our context) `KalmanFilter`
  - rLS (P.R.): `rLSFilter`
  - ACM (B.S.) `ACMfilt`, `ACMfilter`
  - all realized as wrappers to `recursiveFilter`
- required packages — all available from `CRAN`: `methods`, `graphics`, `startupmsg`, `dse1`, `dse2`, `MASS`, `limma`, `robustbase`
- availability: package `robKalman` version `0.1` (incl. demos) under

```
http://www.uni-bayreuth.de/departments/
        /math/org/mathe7/robKalman/
```

Implementation so far: interfaces

- preliminary, "S4-free" interfaces
  - Kalman filter (in our context) `KalmanFilter`
  - rLS (P.R.): `rLSFilter`
  - ACM (B.S.) `ACMfilt`, `ACMfilter`
  - all realized as wrappers to `recursiveFilter`
- required packages — all available from `CRAN`: `methods`, `graphics`, `startupmsg`, `dse1`, `dse2`, `MASS`, `limma`, `robustbase`
- availability: package `robKalman` version `0.1` (incl. demos) under

```
http://www.uni-bayreuth.de/departments/
      /math/org/mathe7/robKalman/
```

Implementation so far: interfaces

- preliminary, "S4-free" interfaces
  - Kalman filter (in our context) `KalmanFilter`
  - rLS (P.R.): `rLSFilter`
  - ACM (B.S.) `ACMfilt`, `ACMfilter`
  - all realized as wrappers to `recursiveFilter`
- required packages — all available from `CRAN`: `methods`, `graphics`, `startupmsg`, `dse1`, `dse2`, `MASS`, `limma`, `robustbase`
- availability: package `robKalman` version `0.1` (incl. demos) under

```
http://www.uni-bayreuth.de/departments/
       /math/org/mathe7/robKalman/
```

Implementation so far: interfaces

- preliminary, "S4-free" interfaces
  - Kalman filter (in our context) `KalmanFilter`
  - rLS (P.R.): `rLSFilter`
  - ACM (B.S.) `ACMfilt`, `ACMfilter`
  - all realized as wrappers to `recursiveFilter`
- required packages — all available from `CRAN`: `methods`, `graphics`, `startupmsg`, `dse1`, `dse2`, `MASS`, `limma`, `robustbase`
- availability: package `robKalman` version `0.1` (incl. demos) under

```
http://www.uni-bayreuth.de/departments/
       /math/org/mathe7/robKalman/
```

Next steps — to do in Banff :-)

- Time-stamps
  - any preferences in the RsR-audience?
  - casting/conversion functions for various time series classes
- S4 classes
  - for SSM's
  - for output-class
  - for control-class (reuse `robustbase`-code)
- intefacing functions between S4-layer and S4-free layer

Next steps — to do in Banff :-)

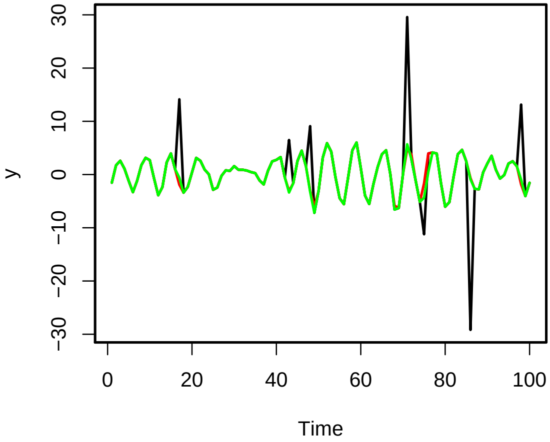
- Time-stamps
  - any preferences in the RsR-audience?
  - casting/conversion functions for various time series classes
- `S4` classes
  - for SSM's
  - for output-class
  - for control-class (reuse `robustbase`-code)
- intefacing functions between `S4`-layer and `S4`-free layer

Next steps — to do in Banff :-)

- Time-stamps
  - any preferences in the RsR-audience?
  - casting/conversion functions for various time series classes
- `S4` classes
  - for SSM's
  - for output-class
  - for control-class (reuse `robustbase`-code)
- intefacing functions between `S4`-layer and `S4`-free layer

Demonstration: ACMfilt

```
## generation of data from AO model:
set.seed(361)
Eps <- as.ts(rnorm(100))
ar2 <- arima.sim(list(ar = c(1, -0.9)),
       100, innov = Eps)
Binom <- rbinom(100, 1, 0.1)
Noise <- rnorm(100,sd = 10)
y <- ar2 + as.ts(Binom*Noise)

## determination of GM-estimates
y.arGM <- arGM(y, 3)
## ACM-filter
y.ACMfilt <- ACMfilt(y, y.arGM)

plot(y)
lines(y.ACMfilt$filt, col=2)
lines(ar2,col="green")
```



green: ideal time series,
black: AO contam. time series,
red: result ACM

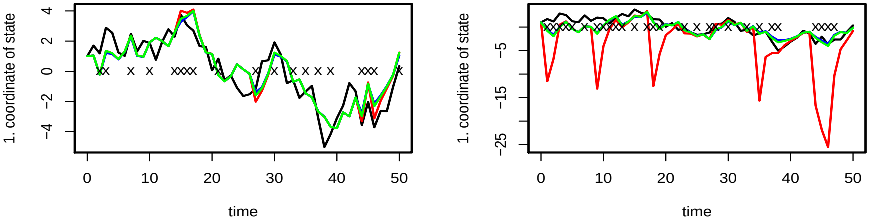
Demonstration: rLSFilter

```
## specification of SSM: (p=2, q=1)
a0  ← c(1, 0); S0  ← matrix(0, 2, 2)
F   ← matrix(c(.7, 0.5, 0.2, 0), 2, 2)
Q   ← matrix(c(2, 0.5, 0.5, 1), 2, 2)
Z   ← matrix(c(1, -0.5), 1, 2)
Vi  ← 1;
## time horizon:
TT ← 50
## specify AO-contamination for simulation
mc  ← -20; Vc  ← 0.1; ract ← 0.1
## for calibration
r1 ← 0.1;  eff1 ← 0.9

#Simulation::
X    ← simulateState(a0, S0, F, Q, TT)
Yid  ← simulateObs(X, Z, Vi, mc, Vc, r=0)
Yre  ← simulateObs(X, Z, Vi, mc, Vc, ract)
```

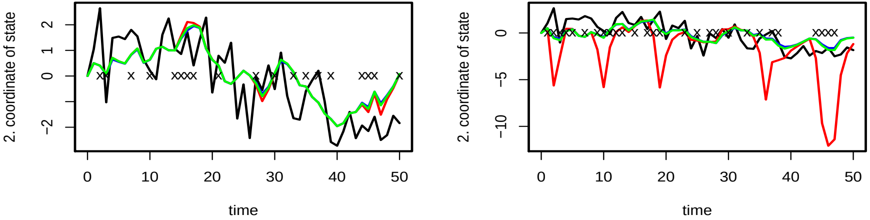
Demonstration: `rLSfilter` II

```
### calibration b
#limiting S_{t|t-1}
SS <- limitS(S0, F, Z, Q, Vi)
####
# tune rLS by efficiency in the ideal model
(B1 <- rLScalibrateB(eff=eff1, S=SS, Z=Z, V=Vi))
# tune rLS by contamination radius
(B2 <- rLScalibrateB(r=r1, S=SS, Z=Z, V=Vi))

### evaluation of rLS
rerg1.id <- rLSFilter(Yid, a0, SS, F, Q, Z, Vi, B1$b)
rerg1.re <- rLSFilter(Yre, a0, SS, F, Q, Z, Vi, B1$b)
rerg2.id <- rLSFilter(Yid, a0, SS, F, Q, Z, Vi, B2$b)
rerg2.re <- rLSFilter(Yre, a0, SS, F, Q, Z, Vi, B2$b)

## for details to obtain the following plot see
##        demo(rLSfilter, package="robKalman")
## CAVEAT: plot() functionality is preliminary and
##          subject to change
```

ideal situation | SO-contaminated situation



black: real state,
red: class. Kalman filter
blue: rLS filter (B1)(mostly covered by:),
green: rLS filter (B2)



x: clipping instances of the robust filter