3.2 Classes

The package *schwartz97* provides the class `schwartz2f`. This class contains all parameters which are needed to define the dynamics of the state variables *spot price* and *convenience yield* under the objective measure $\mathbb{P}$. The class `schwartz2f` has the following slots:

Slot name	Class	Symbol	Description
s0	numeric	s_0	Initial spot price.
delta0	numeric	δ_0	Initial convenience yield.
mu	numeric	μ	Drift parameter of the spot price process.
sigmaS	numeric	σ_S	Diffusion parameter of the spot price process.
kappaE	numeric	κ	Speed of mean-reversion of the convenience yield process.
alpha	numeric	α	Mean-level of the convenience yield process.
sigmaE	numeric	σ_ϵ	Diffusion parameter of the convenience yield process.
rhoSE	numeric	ρ	Correlation between the two Brownian motions.
call	call		The function call.

The above set of parameters contains the symbols appearing in (1) and (2) as well as the initial values s_0 and δ_0. To create an object of class `schwartz2f` the constructor with the same name can be used (see section 4).

The function `fit.schwartz2f`, which estimates parameters of the two-factor model, returns an object of class `schwartz2f.fit`. This class inherits from the class `schwartz2f` and adds the following slots.

Slot name	Class	Symbol	Description
r	numeric	r	Risk-free interest rate.
alphaT	numeric	$\tilde{\alpha}$	Mean-value of the convenience yield process under $\mathbb{Q}$.
lambda	numeric	λ	Market price of convenience yield risk.
deltat	numeric		Time-increment of the transition equation.
n.iter	numeric		Number of iterations.
llh	numeric		Log-likelihood value.
converged	logical		States whether the fit converged or not.
error.code	numeric		An error code or 0.
error.message	character		Contains the error message if any.
fitted.params	logical		States which parameters were fitted.
trace.pars	matrix		Contains the parameter evolution during the estimation.
meas.sd	numeric		Standard deviation of the measurement equation.

These slots together with the ones contained in the class `schwartz2f` fully determine the dynamics of the model under both, the objective measure and the pricing measure. Notice that one of the parameters `lambda` and `alphaT` is redundant according to equation 5.

3.3 Object Orientation

As mentioned earlier most of the functions dealing with the state variables and futures prices are set to generic. The idea is to leave some freedom to the user, who can decide whether he wants to use an object-oriented approach or provide a fairly large set of arguments for each function-call.

Consider the function `dfutures` for example. The function headers for different signatures are:

```
## S4 method for signature 'ANY,ANY,ANY,numeric':
dfutures(x, time = 0.1, ttm = 1, s0 = 50, delta0 = 0,
         mu = 0.1, sigmaS = 0.3, kappa = 1, alpha = 0,
         sigmaE = 0.5, rho = 0.75, r = 0.05, lambda = 0,
         alphaT = NULL, measure = c("P", "Q"), ...)

## S4 method for signature 'ANY,ANY,ANY,schwartz2f':
dfutures(x, time = 0.1, ttm = 1, s0, r = 0.05,
         lambda = 0, alphaT = NULL, measure = c("P", "Q"), ...)

## S4 method for signature 'ANY,ANY,ANY,schwartz2f.fit':
dfutures(x, time = 0.1, ttm = 1, s0, measure = c("P", "Q"), ...)
```

Without object-orientation (first header) the function has 15 arguments. Ten parameters are needed to describe the dynamics under both measures.

If a `schwartz2f.fit` object is provided for `s0` the only additional arguments required are `x` (quantiles), `time` (time where the futures process is evaluated), and `ttm` (time to maturity of the futures contract).

4 Object Initialization

A `schwartz2f` object with reasonable parameters is constructed in the following code chunk.

```
> s0 <- 100
> delta0 <- 0
> mu <- 0.1
> sigmaS <- 0.2
> kappa <- 1
> alpha <- 0.1
> sigmaE <- 0.3
> rho <- 0.4
> obj <- schwartz2f(s0 = s0, delta0 = delta0, alpha = alpha,
+     mu = mu, sigmaS = sigmaS, sigmaE = sigmaE,
+     rho = rho, kappa = kappa)
> obj
```

(α) is 5%. The mean return (μ) of corn is 10% and the volatility is 30%. The speed of mean-reversion of the convenience yield (κ) is 1.5 and its volatility is 40%. Correlation is assumed to be 60%. The risk-free rate is 3% and the market price of convenience yield risk (λ) is zero.

First the object is initialized. Next a trajectory is generated based on a weekly sampling over five years. Then futures prices are calculated with time to maturities ranging from zero (which is the spot) to two years. Finally, a call option which matures in one year written on a futures contract with time to maturity of two years is priced. Fig. 2 plots forward curves.

```
> s0 <- 80
> delta0 <- 0.05
> mu <- 0.1
> sigmaS <- 0.3
> kappa <- 1.5
> alpha <- 0.05
> sigmaE <- 0.4
> rho <- 0.6
> lambda <- 0.04
> r <- 0.03
> set.seed(1)
> obj <- schwartz2f(s0, delta0, mu, sigmaS, kappa,
+     alpha, sigmaE, rho)
> state.traj <- simstate(n = 52 * time, time, obj)
> pricefutures(seq(0, 2, by = 0.4), obj, lambda = lambda,
+     r = r)

[1] 80.00000 79.28309 78.64870 78.17279 77.81715 77.53741

> priceoption(type = "call", time = 1, Time = 2,
+     K = 85, obj, r = r, lambda = lambda)

[1] 4.991482
```

7 Contango, Backwardation, and Hump Shapes

Fig. 2 shows the ability of the Schwartz two-factor model to generate contango and backwardation situations. Mixed shapes (humps and "inverse" humps) are possible too. E.g. an upwards sloping forward curve at the short end which points downwards at the long end.

Looking at the $\mathbb{Q}$-dynamics in equations (3) and (4) it is obvious that, locally, the drift of the spot price is positive when $\delta_t < r$. This corresponds to a (local) contango situation. However, the long-term mean of δ_t is $\tilde{\alpha}$. This means that short and long-term futures can point in different directions.

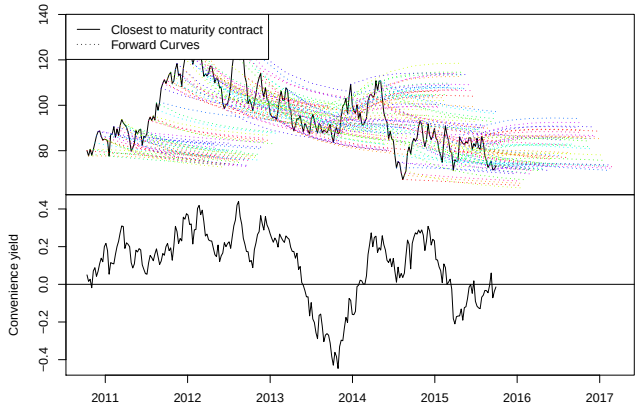

Figure 2: Forward curves with time to maturity up to two years are plotted for the trajectory *state.traj*. The "closest to maturity contract" is in fact the spot price because the time to maturity is zero. The forward curves are steeper (stronger contango) the lower the convenience yield is.

Four different shapes are generated in the following example and plotted in fig. 3:

Pure contango: If $\delta_0 < r$ and $\tilde{\alpha} < r$.

Short end backwardation, long end contango: If $\delta_0 > r$ and $\tilde{\alpha} < r$.

Pure backwardation: If $\delta_0 > r$ and $\tilde{\alpha} > r$.

Short end contango, long end backwardation: If $\delta_0 < r$ and $\tilde{\alpha} > r$.

```
> s0 <- 1
> delta0 <- 0
> sigmaS <- 0.3
> kappa <- 1
> sigmaE <- 0.4
> rho <- 0.5
> r <- 0.03
```

8 Parameter Estimation

As mentioned in section 1, we believe that the package's most valuable piece of code is the function `fit.schwartz2f`. This function estimates the parameters involved in equations (1) - (4) including the initial values of the state variables s_0 and δ_0. Because log-futures prices linearly depend on a bivariate Gaussian random vector (the log-spot price and the convenience yield), it is most straightforward to use a linear state-space model. Therefore, the estimation procedure is based on the Kalman filter as proposed in Schwartz (1997).

The header of the function `fit.schwartz2f` looks like

```
> args(fit.schwartz2f)

function (data, ttm, deltat = 1/260, s0 = data[1, 1], delta0 = 0,
    mu = 0.1, sigmaS = 0.3, kappa = 1, alpha = 0, sigmaE = 0.3,
    rho = 0.7, lambda = 0, meas.sd = rep(0.1, ncol(data)), opt.pars = c(s0 = FALSE,
        delta0 = FALSE, mu = TRUE, sigmaS = TRUE, kappa = TRUE,
        alpha = TRUE, sigmaE = TRUE, rho = TRUE, lambda = FALSE),
    opt.meas.sd = c("scalar", "all", "none"), r = 0.03, silent = FALSE,
    ...)
NULL
```

The data inputs are `data` and `ttm`. `data` must be a regularly spaced time-series matrix of futures prices and `ttm` a matrix giving the time-to-maturity.

The time-to-maturity matrix admits the following interpretation: `data[i,j]` denotes the futures price whose time to maturity was `ttm[i,j]` when it was observed. The unit is defined by `deltat` which is the time between observations `data[i,j]` and `data[i+1,j]`.

The arguments from `s0` to `lambda` are initial values of the parameters.

`meas.sd` gives (initial) values of the measurement error standard deviations. Note that the off-diagonals of the measurement error covariance matrix are assumed to be zero.

`opt.pars` states which parameters shall be estimated. Note that some parameters are held constant by default.

`opt.meas.sd` specifies how measurement uncertainty is treated in the fit: According to the model there should be a one-to-one correspondance between the spot and the futures price. In reality, the term structure does not fully match for any set of parameters. This is reflected in the measurement uncertainty-vector `meas.sd`. All components of `meas.sd` can be fitted. However, it might be sufficient to fit only a scalar where the measurement uncertainty is parametrized by `scalar * meas.sd`. In this case define the vector `meas.sd` and set `opt.meas.sd` to "scalar". `meas.sd` can be set to a vector with each component set to, e.g., 2%, giving each point in the term

structure equal weight. Another reasonable specification takes open interest or volumes into account: The higher the volume, the higher the weight and therefore the lower the corresponding component of `meas.sd`. If all components of `meas.sd` shall be fitted choose "all". If the measurement uncertainty is known set `meas.sd` to "none". Note that the measurement errors are assumed to be independent in this implementation (even though the model and the filter do not require independence).

Finally, the risk-free rate `r` must be given.

8.1 Statistical and Computational Considerations

Estimation of the Schwartz two-factor model parameters is statistically fragile and computationally demanding. Multiple local maxima of the likelihood may exist which can result in absurd parameter estimates as, e.g., a yearly drift of 300% and or a market price of convenience yield risk of -200%. Therefore, a reasonable parameter estimation is most likely an iteration where several initial values are used and different combinations of parameters are held constant during estimation. Also, simulation studies showed that a fairly large sample is required to get adequate estimates (e.g. 20000 daily observations, depending on the number of parameters which shall be estimated). For this reason the default is to hold `s0`, `delta0`, and `lambda` constant.

Several utility functions as `fitted`, `resid`, `plot`, and `coef` may help to investigate the quality of the fit (see example below).

The fitting procedure generally requires a large number of iterations to achieve a reasonable tolerance level. Each optimization iteration involves the filtering of the data set by the Kalman filter. Therefore, an efficient implementation of the Kalman filter is key.

8.2 Example: Estimating Wheat Parameters

This section takes you through a "real-world" example of a Schwartz two-factor parameter estimation. There are daily observations of the five closest to maturity wheat futures prices from Jan. 1995 to April 2010 (approx. 4000 observations).

The default parameters of `fit.schwartz` are used, i.e., all parameters except the initial values of the state variables and the market price of convenience yield risk `lambda` are estimated. The maximum number of iterations is limited to 300 to save (build) time. Then the object is printed and the parameter evolution is plotted.

```
> data(futures)
> wheat.fit <- fit.schwartz2f(futures$wheat$price,
+     futures$wheat$ttm/260, deltat = 1/260, control = list(maxit = 300),
```

```
------------------------------------------------------------------
Fitted Schwartz97 two-factor model:

SDE (P-dynamcis)
d S_t     = S_t   * (mu - delta_t) * dt    + S_t * sigmaS * dW_1
d delta_t = kappa * (alpha - delta_t) * dt + sigmaE * dW_2
E(dW_1 * dW_2) = rho * dt

SDE (Q-dynamcis)
d S_t     = S_t   * (r - delta_t) * dt      + S_t * sigmaS * dW*_1
d delta_t = kappa * (alphaT - delta_t) * dt + sigmaE * dW*_2
alphaT = alpha - lambda/kappa

Parameters
s0    : 395.5
delta0: 0
mu    : 0.0973917026862455
sigmaS: 0.315675278006413
kappa : 1
alpha : 0.0439034736965505
sigmaE: 0.280503762575782
rho   : 0.543979631300489
r     : 0.03
lambda: 0
alphaT: 0.0439034736965505
------------------------------------------------------------------
Optimization information
Converged:          FALSE
Fitted parameters:  mu, sigmaS, alpha, sigmaE, rho, meas.sd1; (Number: 6)
log-Likelihood:     -3927591873
Nbr. of iterations: 301
------------------------------------------------------------------
```

```
> plot(wheat.fit.constr, type = "trace.pars")
```

Parameters are more reasonable now: μ, α, and ρ seem to be fine at 10%, 4.3%, and 54%, respectively. Also, as a quick check, simulated trajectories in fig. 6 look plausible. Real term structures are compared to model generated term structures in fig. 7.

```
> wheat.2007 <- lapply(futures$wheat, function(x) x[as.Date(rownames(x)) >
+     "2007-01-01" & as.Date(rownames(x)) < "2008-07-01",
+     ])
> par(mfrow = c(1, 2))
```


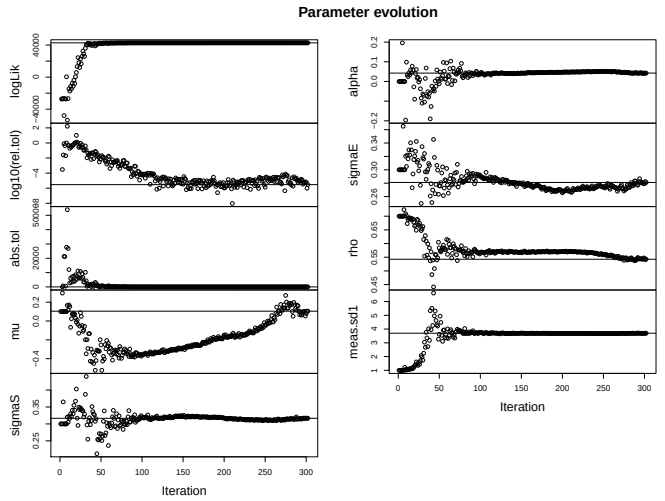

Figure 5: This figure shows the parameter evolution of the first 300 iterations of the constrained parameter estimation of wheat. The relative tolerance gets below 10^{-6} after 150 iterations the parameter values get more and more stationary.

```
> futuresplot(wheat.2007, type = "forward.curve")
> plot(wheat.fit.constr, type = "forward.curve",
+     data = wheat.2007$price, ttm = wheat.2007$ttm/260)
```

8.2.1 Residual Analysis

Hands-on model validation is done here via graphical residual analysis. "Residuals" refer to prediction errors of the Kalman filter's measurement equation. According to the model residuals should be serially independent Gaussian random variables.

Different types of residulas can be obtained by the generic `resid` function and the specific argument `type`, which can be "filter" (raw filter residuals), "filter.std" (standardized filter residuals), and "real" (observed minus fitted prices). Standardized residuals are of interest here, hence the argument is "filter.std". First, serial independence is checked and then normality of residuals. Both assumptions are violated as shown in fig. 8 and fig. 9.

Before rejecting the two-factor model one could try different settings for the measurement error standard deviations (argument `meas.sd`), e.g. "all". Beside that different parameters could be hold constant.

```
> model.resid <- resid(wheat.fit.constr, data = futures$wheat$price,
+     ttm = futures$wheat$ttm/260, type = "filter.std")
> acf(model.resid, na.action = na.pass)
> par(mfrow = c(3, 2))
> invisible(apply(model.resid, 2, function(x) plot(density(na.omit(x)))))
```


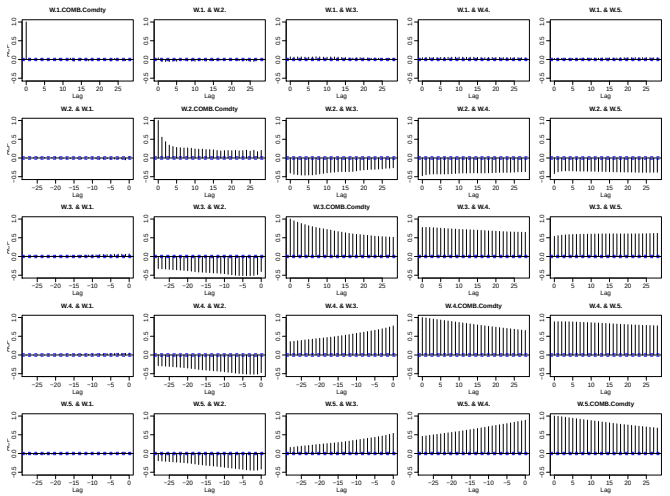

Figure 8: Residual's auto- and crosscorrelation estimates. Recall that off-diagonals of the measurement error covariance matrix are not estimated, hence the crosscorrelation is not relevant. Residuals of the closest to maturity futures show insignificant autocorrelation. However, residuals of all other futures are heavily autocorrelated. As the measurement error standard deviations are set proportional to the average traded volumes of the wheat futures, the two closest to maturity futures clearly get highest weights.

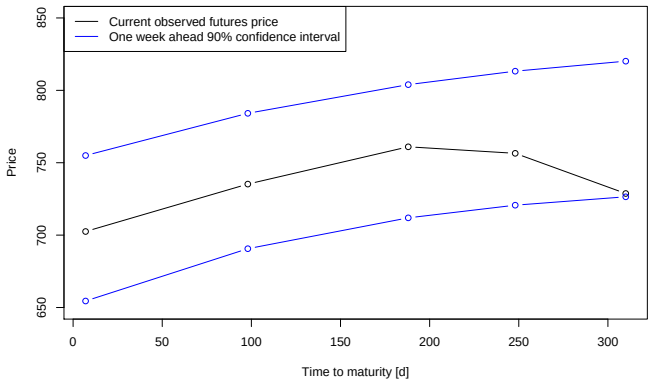

Figure 10: Current observed futures prices and one week ahead confidence intervals for the Sept. 2010, Dec. 2010, Mar. 2011, May 2011, and July 2011 wheat constracts as of Sept. 7, 2010.

References

Rajna Gibson and Eduardo S. Schwartz. Stochastic convenience yield and the pricing of oil contingent claims. *The Journal of Finance*, 45(3):959–976, 1990.

Jimmy E. Hilliard and Jorge Reis. Valuation of commodity futures and options under stochastic convenience yields, interest rates, and jump diffusions in the spot. *Journal of Financial and Quantitative Analysis*, 33 (1):61–86, 1998.

Kristian R. Miltersen and Eduardo S. Schwartz. Pricing of options on commodity futures with stochastic term structures of convenience yields and interest rates. *Journal of Financial and Quantitative Analysis*, 33:33–59, 1998.

Eduardo S. Schwartz. The stochastic behavior of commodity prices: Implications for valuation and hedging. *Journal of Finance*, 52(3):923–973, 1997.