Package ‘NFCP’

February 25, 2021

Title N-Factor Commodity Pricing Through Term Structure Estimation

Version 0.2.0

Description Commodity pricing models are (systems of) stochastic differential equations that are utilized for the valuation and hedging of commodity contingent claims (i.e. derivative products on the commodity) and other commodity related investments. Commodity pricing models that capture market dynamics are of great importance to commodity market participants in order to exercise sound investment and risk-management strategies. Parameters of commodity pricing models are estimated through maximum likelihood estimation, using available term structure futures data of a commodity. 'NFCP' (n-factor commodity pricing) provides a framework for the modeling, parameter estimation, probabilistic forecasting, option valuation and simulation of commodity prices through state space and Monte Carlo methods, risk-neutral valuation and Kalman filtering. 'NFCP' allows the commodity pricing model to consist of n correlated factors, with both random walk and mean-reverting elements. The n-factor commodity pricing model framework was first presented in the work of Cortazar and Naranjo (2006) <doi:10.1002/fut.20198>. Examples presented in 'NFCP' replicate the two-factor crude oil commodity pricing model presented in the prolific work of Schwartz and Smith (2000) <doi:10.1287/mnsc.46.7.893.12034> with the approximate term structure futures data applied within this study provided in the 'NFCP' package.

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1.9000

RdMacros mathjaxr,
Rdpack

Suggests OptionPricing,
knitr,
rmarkdown

Imports FKF.SP,
LSMRealOptions,
MASS,
numDeriv,
parallel,
rgenoud,
stats,
mathjaxr,
Rdpack,
curl

VignetteBuilder knitr

R topics documented:

`American.Option.Value` *N-Factor Model American Put Option Pricing*

Description

Value American Put Options under the parameters of an N-factor model through the Least-Squares Monte Carlo (LSM) Simulation Method. This function is a wrapper to the 'LSM.AmericanOption' function of the 'LSMRealOptions' package.

Usage

```
American.Option.Value(
  X.0,
  parameters,
  n,
  t,
  dt,
  K,
  r,
  orthogonal = "Power",
  degree = 2,
  verbose = FALSE,
  debugging = FALSE
)
```

Arguments

`X.0`	Initial values of the state vector.
`parameters`	Named vector of parameter values of a specified N-factor model. Function `NFCP.Parameters` is recommended.
`n`	total number of simulated price paths
`t`	Time to expiration of the option (in years)
`dt`	discrete time step of simulation
`K`	Strike price of the American put option
`r`	Risk-free interest rate.
`orthogonal`	The orthogonal polynomial used to approximate the continuation value of the option in the LSM simulation method. Orthogonal polynomial arguments available are: "Power", "Laguerre", "Jacobi", "Legendre", "Chebyshev", "Hermite". See `help(LSM.AmericanOption)`
`degree`	The degree of polynomials used in the least squares fit. See `help(LSM.AmericanOption)`
`verbose`	`logical` Should additional information be output?
`debugging`	`logical` Should additional simulation information be output?

Details

The 'American.Option.Value' function is a wrapper to the 'Spot.Price.Simulate' and 'LSM.AmericanOption' of the 'LSMRealOptions' package that returns the value of American put options under a given N-factor model.

The least-squares Monte Carlo (LSM) simulation method is an option valuation method first presented by Longstaff and Schwartz (2001) that approximates the value of American options.

Methods to solve for the value of options with early exercise opportunities include partial differential equations, lattice-based methods and Monte-Carlo simulation. LSM simulation is the optimal solution method to value American options under an N-factor model due to the multiple factors that can make up the spot price process and influence the option value. Whilst lattice and partial differential equation approaches suffer from the 'curse of dimensionality', LSM simulation may be readily applied under multi-factor settings.

Longstaff and Schwartz (2001) state that as the conditional expectation of the continuation value belongs to a Hilbert space, it can be represented by a combination of orthogonal basis functions. Increasing the number of stochastic state variables therefore increases the number of required basis functions exponentially.

Value

The 'American.Option.Value' function by default returns a `numeric` object corresponding to the calculated value of the American put option.

When `verbose = T`, 6 objects are returned within a `list` class object. The objects returned are:

`Value`	The calculated option value.
`Standard Error`	The standard error of the calculated option value.
`Expected Timing`	The expected time of early exercise..
`Expected Timing SE`	The standard error of the expected time of early exercise.
`Exercise Probability`	The probability of early exercise of the option being exercised.
`Cumulative Exercise Probability`	`vector`. The cumulative probability of option exercise at each discrete observatio

When debugging = T, an additional 2 objects are returned within the list class object. These are the results output by both the 'Spot.Price.Simulate' and 'LSM.AmericanOption' of the 'LSMRealOptions' package respectively. The objects returned are:

State_Variables	A matrix of simulated state variables for each factor is returned when verbose = T. The number of fa
Prices	A matrix of simulated price paths. Each column represents one simulated price path and each row re

References

Longstaff, F.A., and E.S. Schwartz. 2001. Valuing American Options by Simulation: A Simple Least-Squares Approach. The Review of Financial Studies. 14:113-147.

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

Cortazar, G., and L. Naranjo, (2006). An N-factor Gaussian model of oil futures prices. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 26(3), 243-268.

Aspinall, T., A. Gepp, G. Harris, S. Kelly, C. Southam, and B. Vanstone, (2021). LSMRealOptions: Value American and Real Options Through LSM Simulation. R package version 0.1.0.

Examples

```
##Example 1 - An American put option on a stock following 'GBM'
##growing at the risk-free rate:
American.Option.Value(X.0 = log(36),
    parameters = c(mu_star = (0.06 - (1/2) * 0.2^2), sigma_1 = 0.2),
    n = 1e2,
    t = 1,
    dt = 1/50,
    K = 40,
    r = 0.05,
    verbose = FALSE,
    orthogonal = "Laguerre",
    degree = 3)

##Example 2 - An American put option under a two-factor crude oil model:

##Step 1 - Obtain current (i.e. most recent) state vector by filtering the
##two-factor oil model:
Schwartz.Smith.Oil <- NFCP.Kalman.filter(parameter.values = SS.Oil$Two.Factor,
                                        parameters = names(SS.Oil$Two.Factor),
                                        log.futures = log(SS.Oil$Stitched.Futures),
                                        dt = SS.Oil$dt,
                                        TTM = SS.Oil$Stitched.TTM,
                                        verbose = TRUE)

##Step 2 - Calculate 'put' option price:
American.Option.Value(X.0 = Schwartz.Smith.Oil$X.t,
                     parameters = SS.Oil$Two.Factor,
                     n = 1e2,
                     t = 1,
                     dt = 1/12,
                     K = 20,
                     r = 0.05,
                     verbose = FALSE,
                     orthogonal = "Power",
                     degree = 2)
```

`A_T` *Calculate $A(T)$*

Description

Calculate the values of $A(T)$ for a given N-factor model parameters and observations. Primarily purpose is for application within other functions of the `NFCP` package.

Usage

```
A_T(parameters, Tt)
```

Arguments

`parameters`	A named vector of parameters of an N-factor model. Function `NFCP.Parameters` is recommended.
`Tt`	A vector or matrix of the time-to-maturity of observed futures prices

Details

Under the assumption that Factor 1 follows a Brownian Motion, $A(T)$ is given by:

$$A(T) = \mu^* T - \sum_{i=1}^{N} - \frac{1 - e^{-\kappa_i T} \lambda_i}{\kappa_i} + \frac{1}{2}(\sigma_1^2 T + \sum_{i.j \neq 1} \sigma_i \sigma_j \rho_{i,j} \frac{1 - e^{-(\kappa_i + \kappa_j)T}}{\kappa_i + \kappa_j})$$

Value

A matrix of identical dimensions to T providing the values of function $A(T)$ of a given N-factor model and observations.

References

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

Cortazar, G., and L. Naranjo, (2006). An N-factor Gaussian model of oil futures prices. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 26(3), 243-268.

Examples

```
##Calculate time homogeneous values of A(T) for the
##Schwartz and Smith (2000) two-factor model:
SS.Oil.A_T <- A_T(SS.Oil$Two.Factor, SS.Oil$Stitched.TTM)
```

cov_func *N-factor Covariance:*

Description

Calculate the covariance matrix of state variables for a given N-factor model parameters and discrete time step.

Usage

```
cov_func(parameters, dt)
```

Arguments

`parameters`	a named vector of parameters of an N-factor model. Function `NFCP.Parameters` is recommended.
`dt`	a discrete time step

Details

The primary purpose of the `model_covariance` function is to be called within other functions of the `NFCP` package. The covariance of an N-factor model is given by:

$$cov_{1,1}(x_{1,t}, x_{1,t}) = \sigma_1^2 t$$

$$cov_{i,j}(x_{i,t}, x_{j,t}) = \sigma_i \sigma_j \rho_{i,j} \frac{1 - e^{-(\kappa_i + \kappa_j)t}}{\kappa_i + \kappa_j}$$

Value

A `matrix` object with dimensions $N \times N$, where N is the number of factors of the specified N-factor model.

References

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

Cortazar, G., and L. Naranjo, (2006). An N-factor Gaussian model of oil futures prices. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 26(3), 243-268.

Examples

```
#Calculate the covariance matrix of a two-factor model over one discrete (weekly) time step:
SS.Oil.covariance <- cov_func(SS.Oil$Two.Factor, SS.Oil$dt)
```

`European.Option.Value` *N-Factor Model European Option Pricing*

Description

Value European Option Put and Calls under the parameters of an N-factor model.

Usage

```
European.Option.Value(X.0, parameters, t, TTM, K, r, call, verbose = FALSE)
```

Arguments

`X.0`	Initial values of the state vector.
`parameters`	Named vector of parameter values of a specified N-factor model. Function `NFCP.Parameters` is recommended.
`t`	Time to expiration of the option
`TTM`	Time to maturity of the Futures contract.
`K`	Strike price of the European Option
`r`	Risk-free interest rate.
`call`	`logical` is the European option a call or put option?
`verbose`	`logical`. Should additional information be output? see **details**

Details

The `European.Option.Value` function calculates analytic expressions of the value of European call and put options on futures contracts within the N-factor model. Under the assumption that future futures prices are log-normally distributed under the risk-neutral process, there exist analytic expressions of the value of European call and put options on futures contracts. The following analytic expression follows from that presented by Schwartz and Smith (2000) extended to the N-factor framework. The value of a European option on a futures contract is given by calculating its expected future value using the risk-neutral process and subsequently discounting at the risk-free rate.

One can verify that under the risk-neutral process, the expected futures price at time t is:

$$E^*[F_{T,t}] = exp(\sum_{i=1}^{N} e^{-\kappa_i T} x_i(0) + \mu^* t + A(T-t) + \frac{1}{2}(\sigma_1^2 t + \sum_{i.j \neq 1} e^{-(\kappa_i+\kappa_j)(T-t)} \sigma_i \sigma_j \rho_{i,j} . \frac{1 - e^{-(\kappa_i+\kappa_j)t}}{\kappa_i + \kappa_j})) \equiv F_{T,0}$$

This follows from the derivation provided within the vignette of the NFCP package as well as the details of the `Futures.Price.Forecast` package. The equality of expected futures price at time t being equal to the time-t current futures price $F_{T,0}$ is proven by Futures prices being given by expected spot prices under the risk-neutral process ($F_{T,t} = E_t^*[S_T]$) and the law of iterated expectations ($E^*[E_t^*[S_T]] = E^*[S_T]$)

Because future futures prices are log-normally distributed under the risk-neutral process, we can write a closed-form expression for valuing European put and call options on these futures. When $T = 0$ these are European options on the spot price of the commodity itself. The value of a European call option on a futures contract maturing at time T, with strike price K, and with time t until the option expires, is:

$$e^{-rt}E^*\left[\max\left(F_{T,t}-K,0\right)\right]$$

$$=e^{-rt}(F_{T,0}N(d)-KN(d-\sigma_\phi(t,T)))$$

Where:

$$d=\frac{\ln(F/K)}{\sigma_\phi(t,T)}+\frac{1}{2}\sigma_\phi\left(t,T\right)$$

and:

$$\sigma_\phi\left(t,T\right)=\sqrt{(}\sigma_1^2 t+\sum_{i.j\neq 1}e^{-(\kappa_i+\kappa_j)(T-t)}\sigma_i\sigma_j\rho_{i,j}.\frac{1-e^{-(\kappa_i+\kappa_j)t}}{\kappa_i+\kappa_j})$$

Parameter $N(d)$ indicates cumulative probabilities for the standard normal distribution (i.e. $P(Z < d)$).

Similarly, the value of a European put with the same parameters is given by:

$$e^{-rt}E^*[max(K-F_{T,t},0)]$$

$$=e^{-rt}\left(-F_{T,0}N\left(-d\right)+KN\left(\sigma_\phi\left(t,T\right)-d\right)\right)$$

The presented option valuation formulas are analogous to the Black-Scholes formulas for valuing European options on stocks that do not pay dividends

Under this terminology, the stock price corresponds to the present value of the futures commitment $(e^{-rt}F_{T,0})$ and the equivalent annualized volatility would be $\sigma_\phi(t,T)/\sqrt{t}$

When `verbose = T`, the `European.Option.Value` function numerically calculates the sensitivity of option prices to the underlying parameters specified within the N-factor model, as well as some of the most common "Greeks" related to European put and call option pricing. All gradients are calculated numerically by calling the `grad` function from the `numDeriv` package.

Value

The `European.Option.Value` function returns a numeric value corresponding to the present value of an option when `verbose = F`. When `verbose = T`, `European.Option.Value` returns a list with three objects:

`Value`	Present value of the option.
`Annualized.Volatility`	Annualized volatility of the option.
`Parameter.Sensitivity`	Sensitivity of the option value to each parameter of the N-factor model.
`Greeks`	Sensitivity of the option value to different option parameters.

References

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

Cortazar, G., and L. Naranjo, (2006). An N-factor Gaussian model of oil futures prices. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 26(3), 243-268.

Examples

```
##Example 1 - A European 'put' option on a stock following 'GBM'
##growing at the risk-free rate:

### Risk-free rate:
rf <- 0.05
### Stock price:
S0 <- 20
### Stock volatility:
S.sigma <- 0.2
### Option maturity:
Tt <- 1
### Exercise price:
K <- 20
### Calculate 'put' option price:
European.Option.Value(X.0 = log(S0), parameters = c(mu_star = rf, sigma_1 = S.sigma),
t = Tt, TTM = Tt, K = K, r = rf, call = FALSE)

##Example 2 - A European call option under a two-factor crude oil model:

##Step 1 - Obtain current (i.e. most recent) state vector by filtering the
##two-factor oil model:
Schwartz.Smith.Oil <- NFCP.Kalman.filter(parameter.values = SS.Oil$Two.Factor,
                                      parameters = names(SS.Oil$Two.Factor),
                                      log.futures = log(SS.Oil$Stitched.Futures),
                                      dt = SS.Oil$dt,
                                      TTM = SS.Oil$Stitched.TTM,
                                      verbose = TRUE)

##Step 2 - Calculate 'call' option price:
European.Option.Value(X.0 = Schwartz.Smith.Oil$X.t,
                      parameters = SS.Oil$Two.Factor,
                      t = 1,
                      TTM = 1,
                      K = 20,
                      r = 0.05,
                      call = TRUE,
                      verbose = FALSE)
```

`Futures.Price.Forecast`

Forecast N-factor Model Futures Prices

Description

Analytically forecast future expected Futures prices under the risk-neutral version of a specified N-factor model.

Usage

```
Futures.Price.Forecast(X.0, parameters, t = 0, TTM = 1:10, Percentiles = NULL)
```

Arguments

`X.0`	Initial values of the state vector.
`parameters`	A named vector of parameter values of a specified N-factor model. Function `NFCP.Parameters` is recommended.
`t`	a numeric specifying the time point at which to forecast futures prices
`TTM`	a vector specifying the time to maturity of futures contracts to value.
`Percentiles`	Optional. A vector of percentiles to include probabilistic forecasting intervals.

Details

Under the assumption or risk-neutrality, futures prices are equal to the expected future spot price. Additionally, under deterministic interest rates, forward prices are equal to futures prices. Let $F_{T,t}$ denote the market price of a futures contract at time t with time T until maturity. let * denote the risk-neutral expectation and variance of futures prices. The following equations assume that the first factor follows a Brownian Motion.

$$E^*[ln(F_{T,t})] = \sum_{i=1}^{N} e^{-\kappa_i T} x_i(0) + \mu^* t + A(T-t)$$

Where:

$$A(T-t) = \mu^*(T-t) - \sum_{i=1}^{N} - \frac{1-e^{-\kappa_i(T-t)}\lambda_i}{\kappa_i} + \frac{1}{2}(\sigma_1^2(T-t) + \sum_{i.j\neq 1} \sigma_i \sigma_j \rho_{i,j} \frac{1-e^{-(\kappa_i+\kappa_j)(T-t)}}{\kappa_i+\kappa_j})$$

The variance is given by:

$$Var^*[ln(F_{T,t})] = \sigma_1^2 t + \sum_{i.j\neq 1} e^{-(\kappa_i+\kappa_j)(T-t)} \sigma_i \sigma_j \rho_{i,j} \frac{1-e^{-(\kappa_i+\kappa_j)t}}{\kappa_i+\kappa_j}$$

Value

`Futures.Price.Forecast` returns a vector of expected Futures prices under a given N-factor model with specified time to maturities at time t. When `percentiles` are specified, the function returns a matrix with the corresponding confidence bands in each column of the matrix.

References

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

Cortazar, G., and L. Naranjo, (2006). An N-factor Gaussian model of oil futures prices. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 26(3), 243-268.

Examples

```
# Forecast futures prices of the Schwartz and Smith (2000) two-factor oil model:
## Step 1 - Run the Kalman filter for the two-factor oil model:
Schwartz.Smith.Oil = NFCP.Kalman.filter(parameter.values = SS.Oil$Two.Factor,
                                        parameters = names(SS.Oil$Two.Factor),
                                        log.futures = log(SS.Oil$Stitched.Futures),
                                        dt = SS.Oil$dt,
                                        TTM = SS.Oil$Stitched.TTM,
                                          verbose = TRUE)

## Step 2 - Probabilistic forecast of the risk-neutral two-factor
## stochastic differential equation (SDE):
E.Futures = Futures.Price.Forecast(X.0 = Schwartz.Smith.Oil$X.t,
                                   parameters = SS.Oil$Two.Factor,
                                   t = 0,
                                   TTM = seq(0,9,1/12),
                                   Percentiles = c(0.1, 0.9))
```

`Futures.Price.Simulate`

Simulate N-Factor Model Futures Prices

Description

Simulate Futures price data with dynamics that follow the parameters of an N-factor model through Monte Carlo simulation.

Usage

```
Futures.Price.Simulate(X.0, parameters, dt, N.obs, TTM, verbose = TRUE)
```

Arguments

`X.0`	Initial values of the state vector.
`parameters`	A named vector of parameter values of a specified N-factor model. Function `NFCP.Parameters` is recommended.
`dt`	discrete time step of simulation
`N.obs`	The number of observations to simulate
`TTM`	A vector or matrix of the time to maturity of futures contracts to simulate. See **details**
`verbose`	`logical`. Should the simulated state variables and associated prices be output?

Details

The `Futures.Price.Simulate` function simulates futures price data using the Kalman Filter algorithm, drawing from a normal distribution for the shocks in the transition and measurement equations at each discrete time step. At each discrete time point, an observation of the state vector is generated through the transition equation, drawing from a normal distribution with a covariance equal to Q_t. Following this, simulated futures prices are generated through the measurement equation, drawing from a normal distribution with covariance matrix equal to H.

Input `TTM` can be either a matrix specifying the constant time to maturity of futures contracts to simulate, or it can be a matrix where `nrow(TTM) == N.obs` for the time-varying time to maturity of the futures contracts to simulate. This allows for the simulation of both aggregate stitched data and individual futures contracts.

Value

`Futures.Price.Simulate` returns a list with three objects when `verbose = T` and a matrix of simulated futures prices when `verbose = F`. The list objects returned are:

#'

`State.Vector`	A `matrix` of Simulated state variables at each discrete time point. The columns represent each factor of t
`Futures`	A `matrix` of Simulated futures prices, with each column representing a simulated futures contract.
`Spot`	A vector of simulated spot prices

References

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

Cortazar, G., and L. Naranjo, (2006). An N-factor Gaussian model of oil futures prices. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 26(3), 243-268.

Examples

```
##Example 1 - Simulate Crude Oil Stitched futures prices
##under a Two-Factor model, assuming a constant time to maturity:

Simulated.Stitched.Futures <- Futures.Price.Simulate(X.0 = c(log(SS.Oil$Spot[1,1]), 0),
                                                    parameters = SS.Oil$Two.Factor,
                                                    dt = SS.Oil$dt,
                                                   N.obs = nrow(SS.Oil$Stitched.Futures),
                                                    TTM = SS.Oil$Stitched.TTM)

##Example 2 - Simulate Crude Oil Contract Prices under a Two-Factor model

###Assume constant white noise in parameters of 1%:
SS.Oil.Two.Factor <- SS.Oil$Two.Factor
SS.Oil.Two.Factor <- SS.Oil.Two.Factor[!grepl("contract", names(SS.Oil.Two.Factor))]
SS.Oil.Two.Factor["sigma.contracts"] <- 0.01

Simulated.Contracts <- Futures.Price.Simulate(X.0 = c(log(SS.Oil$Spot[1,1]), 0),
                                             parameters = SS.Oil.Two.Factor,
                                             dt = SS.Oil$dt,
                                             N.obs = nrow(SS.Oil$Contracts),
                                             TTM = SS.Oil$Contract.Maturities)
```

NFCP.Domains *N-Factor MLE Search Boundaries*

Description

Generate boundaries for the domain of parameters of the N-factor model for parameter estimation.

Usage

```
NFCP.Domains(
  parameters,
  kappa = NULL,
  lambda = NULL,
  sigma = NULL,
  mu = NULL,
  mu_star = NULL,
  rho = NULL,
  contract = NULL,
  X.0 = NULL,
  E = NULL
)
```

Arguments

parameters	a vector of parameter names of an N-factor model. Function NFCP.Parameters is recommended.
kappa	A vector of length two specifying the lower and upper bounds for the 'kappa' parameter
lambda	A vector of length two specifying the lower and upper bounds for the 'lambda' parameter
sigma	A vector of length two specifying the lower and upper bounds for the 'sigma' parameter
mu	A vector of length two specifying the lower and upper bounds for the 'mu' parameter
mu_star	A vector of length two specifying the lower and upper bounds for the 'mu_star' parameter
rho	A vector of length two specifying the lower and upper bounds for the 'rho' parameter
contract	A vector of length two specifying the lower and upper bounds for the 'contract' parameter
X.0	A vector of length two specifying the lower and upper bounds for the 'X.0' parameter
E	A vector of length two specifying the lower and upper bounds for the 'E' parameter

Details

The `NFCP.Domains` function generates lower and upper bounds for the parameter estimation procedure in the format required of the 'Domains' argument of the 'genoud' function. `NFCP.Domains` allows easy setting of custom boundaries for parameter estimation, whilst also providing default domains of parameters.

Value

A matrix of defaulted domains for the given unknown parameters. The first column corresponds to the lower bound of the allowable search space for the parameter, whilst the second column corresponds to the upper bound. These values were set to allow for the 'realistic' possible values of given parameters as well as restricting some parameters (such as variance and mean-reverting terms) from taking negative values. The format of the returned matrix matches that required by the `Domains` argument of the Genoud function from the package RGenoud.

References

Mebane, W. R., and J. S. Sekhon, (2011). Genetic Optimization Using Derivatives: The rgenoud Package for R. *Journal of Statistical Software*, 42(11), 1-26. URL http://www.jstatsoft.org/v42/i11/.

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

Examples

```
##Specify the Schwartz and Smith (2000) two-factor model
##with constant contract white noise:
SS.parameters <- NFCP.Parameters(N.factors = 2,
                                 GBM = TRUE,
                                 Initial.State = TRUE,
                                 S.Constant = TRUE)

###Generate the default 'domains' argument of 'NFCP.MLE' function:
NFCP.MLE.Bounds <- NFCP.Domains(SS.parameters)
```

NFCP.Kalman.filter *N-Factor Commodity Pricing Kalman Filter*

Description

Given a set of parameters of the N-factor model, filter term structure data using the Kalman filter.

Usage

```
NFCP.Kalman.filter(
  parameter.values,
  parameters,
  log.futures,
  dt,
  TTM,
  verbose = FALSE,
  debugging = FALSE
)
```

Arguments

parameter.values	Vector of parameter values of an N-factor model. The NFCP.Kalman.filter function is designed for application to optim type functions, and thus parameter values and corresponding parameters different inputs within the function.
parameters	Vector of parameter names. Each element of parameters must correspond to its respective value element in object parameter.values.
log.futures	Object of class matrix corresponding to the natural logarithm of observable futures prices. NA's are allowed within the matrix. Every column of the matrix must correspond to a particular futures contract, with each row corresponding to a quoted price on a given date.
dt	Constant, discrete time step of observations
TTM	Object of class 'vector' or 'matrix' that specifies the time to maturity of observed futures contracts. time to maturity can be either constant (ie. class 'vector') or time homogeneous (ie. class 'matrix'). When the time to maturity of observed futures contracts is time homogeneous, the dimensions of TTM must be identical to that of log.futures. Every element of TTM corresponds to the time to maturity, in years, of a futures contract at a given observation date.
verbose	logical. Should additional information be output? see **values**. When verbose = F, the NFCP.Kalman.filter function is significantly faster, see **details**
debugging	logical. Should additional filtering information be output? see **values**

Details

`NFCP.Kalman.filter` applies the Kalman filter algorithm for observable `log.futures` prices against the input parameters of an N-factor model provided through the `parameter.values` and `parameters` input vectors.

The `NFCP.Kalman.filter` function is designed for subsequent input into optimization functions and is called within the N-factor parameter estimation function `NFCP.MLE`. The first input to the `NFCP.Kalman.filter` function is a vector of parameters of an N-factor model, with elements of this vector corresponding to the parameter names within the elements of input vector `parameters`. When `logical` input `verbose = F`, the `NFCP.Kalman.filter` function calls the `fkf_SP` function of the `FKF_SP` package, which itself is a wrapper of a routine of the Kalman filter written in C utilizing Sequential Processing for maximum computational efficiency (see `fkf_SP` for more details). When `verbose = T`, the `NFCP.Kalman.filter` instead applies a Kalman filter algorithm written in base `R` and outputs several other `list objects`, including filtered values and measures for model fit and robustness (see **Returns**)

The N-factor model The N-factor model was first presented in the work of Cortazar and Naranjo (2006, equations 1-3). The N-factor framework describes the spot price process of a commodity as the correlated sum of N state variables x_t.

When `GBM = TRUE`:

$$log(S_t) = \sum_{i=1}^{N} x_{i,t}$$

When `GBM = FALSE`:

$$log(S_t) = E + \sum_{i=1}^{N} x_{i,t}$$

Additional factors within the spot-price process are designed to result in additional flexibility, and possibly fit to the observable term structure, in the spot price process of a commodity. The fit of different N-factor models, represented by the log-likelihood can be directly compared with statistical testing possible through a chi-squared test.

Flexibility in the spot price under the N-factor framework allows the first factor to follow a Brownian Motion or Ornstein-Uhlenbeck process to induce a unit root. In general, an N-factor model where `GBM = T` allows for non-reversible behaviour within the price of a commodity, whilst `GBM = F` assumes that there is a long-run equilibrium that the commodity price will revert to in the long-term.

State variables are thus assumed to follow the following processes:

When `GBM = TRUE`:

$$dx_{1,t} = \mu^* dt + \sigma_1 dw_1 t$$

When `GBM = FALSE`:

$$dx_{1,t} = -(\lambda_1 + \kappa_1 x_{1,t})dt + \sigma_1 dw_1 t$$

And:

$$dx_{i,t} =_{i \neq 1} -(\lambda_i + \kappa_i x_{i,t})dt + \sigma_i dw_i t$$

where:

$$E(w_i)E(w_j) = \rho_{i,j}$$

The following constant parameters are defined as:

`param` μ: long-term growth rate of the Brownian Motion process.

`param` E: Constant equilibrium level.

`param` $\mu^* = \mu - \lambda_1$: Long-term risk-neutral growth rate `param` λ_i: Risk premium of state variable i.

`param` κ_i: Reversion rate of state variable i.

`param` σ_i: Instantaneous volatility of state variable i.

`param` $\rho_{i,j} \in [-1,1]$: Instantaneous correlation between state variables i and j.

Measurement Error:

The Kalman filtering algorithm assumes a given measure of measurement error (ie. matrix H). Measurement errors can be interpreted as error in the model's fit to observed prices, or as errors in the reporting of prices (Schwartz and Smith, 2000) and are assumed independent.

`var` s_i Observation error of contract i.

When `S.Constant = T`, the values of parameter s_i are assumed constant and equal to parameter object 'sigma.contracts'. When `S.Constant = F`, observation errors are assumed unique, where the error of futures contracts s_i is equal to parameter object `'sigma.contract_'` [i] (i.e. `'sigma.contract_1'`, `'sigma.contract_2'`, ...).

Kalman Filtering

The following section describes the Kalman filter equations used to filter the N-factor model.

The Kalman filter iteration is characterised by a transition and measurement equation. The transition equation develops the vector of state variables between discretised time steps (whilst considering a given level of covariance between state variables over time). The measurement equation relates the unobservable state vector to a vector of observable measurements (whilst also considering a given level of measurement error). The typical Kalman filter algorithm is a Gaussian process state space model.

Transition Equation:

$$\hat{x}_{t|t-1} = c_t + G_t \hat{x}_{t-1} + Q_t \eta_t$$

Measurement Equation:

$$\hat{y}_t = d_t + Z_t \hat{x}_{t|t-1} + H_t \epsilon_t$$

$$t = 1, \cdots, n$$

Where η_t and ϵ_t are IID $N(0, I(m))$ and iid $N(0, I(d))$ respectively.

The state vector follows a normal distribution, $x_1 \sim N(a_1, P_1)$, with a_1 and P_1 as the mean vector and variance matrix of the initial state vector x_1, respectively.

When `verbose = F`, the `NFCP.Kalman.filter` function performs Kalman filtering through the `fkf.SP` function of the `FKF.SP` package for maximum filtering efficiency, which is crucial when filtering and estimating parameters of term structure data under the N-factor model, which generally has many observations, contracts and unknown parameters. When `verbose = T`, the `NFCP.Kalman.filter` function instead performs Kalman filtering in R, returning greater details of filtered objects (see **Value**)

The Kalman filter can be used for parameter estimation through the maximization of the Log-Likelihood value. See `NFCP.MLE`.

Filtering the N-factor model

let m represent the number of observations at time t

let n represent the number of factors in the N-factor model

observable futures prices: $y_t = [ln(F(t,T_1)), ln(F(t,T_2)), \cdots, ln(F(t,T_m))]'$

State vector: $x_t = [x_1 t, x_2 t, \cdots, x_n t]'$

Measurement error: $diag(H) = [s_1^2, s_2^2, \cdots, s_n^2]$

Where s_i == "sigma.contract_" [i] when the `logical` of function `NFCP.Parameters` `S.constant = F`

When `S.constant = T`, $s_1 = s_2 = \cdots = s_n =$ "sigma.contracts"

`var` Z is an $m \times n$ matrix, where each element $[i, j]$ is equal to:

$$Z_{i,j} = e^{-\kappa_i T_j}$$

`var` d_t is an $m \times 1$ vector:

$$d_t = [A(T_1), A(T_2), \cdots, A(T_m)]'$$

Under the assumption that Factor 1 follows a Brownian Motion, $A(T)$ is given by:

$$A(T) = \mu^* T - \sum_{i=1}^{N} - \frac{1 - e^{-\kappa_i T}\lambda_i}{\kappa_i} + \frac{1}{2}(\sigma_1^2 T + \sum_{i.j \neq 1} \sigma_i \sigma_j \rho_{i,j} \frac{1 - e^{-(\kappa_i + \kappa_j)T}}{\kappa_i + \kappa_j})$$

`var` v_t is a $n \times 1$ vector of serially uncorrelated Guassian disturbances with $E(V_t) = 0$ and $cov(v_t) = R^2$

Where:

$diag(G_t) = [e^{-\kappa_1 \tau}, e^{-\kappa_2 \tau}, \cdots, e^{-\kappa_n \tau}]$

Where $\tau = T - t$

`var` w_t is an $n \times 1$ vector of serially uncorrelated Guassian disturbances where:

$$E(w_t) = 0$$

and $cov(w_t) = Q_t$

`var` $c_t = [\mu \Delta t, 0, \cdots, 0]'$ is an $N \times 1$ vector of the intercept of the transition equation.

`var` Q_t is equal to the covariance function, given by:

$$Cov_{1,1}(x_{1,t}, x_{1,t}) = \sigma_1^2 t$$

$$Cov_{i,j}(x_{i,t}, x_{j,t}) = \sigma_i \sigma_j \rho_{i,j} \frac{1 - e^{-(\kappa_i + \kappa_j)t}}{\kappa_i + \kappa_j}$$

(see also `cov_func`)

Penalising poorly specified models

The Kalman filter returns non-real log-likelihood scores when the function of the covariance matrix becomes singular or its determinant becomes negative. This occurs when a poorly specified parameter set is input. Non-real log-likelihood scores can break optimization algorithms. To circumvent this, the `NFCP.Kalman.filter` returns a heavily penalized log-likelihood score whilst also returning a warning. Penalized log-likelihood scores are calculated by:

`stats::runif(1,-1.5e6,-1e6)`

Diffuse Kalman filtering

If the initial values of the state vector are not supplied within the `parameters` and `parameter.values` vectors (ie. `Initial.State = F` within the `NFCP.Parameters` function), a 'diffuse' assumption is used within the Kalman filtering algorithm. Factors that follow an Ornstein-Uhlenbeck are assumed to equal zero. When `Estimate.Initial.State = F` and `GBM = T`, the initial value of the first state variable is assumed to equal the first element of `log.futures`. This is an assumption that the initial estimate of the spot price is equal to the closest to maturity observed futures price.

The initial covariance of the state vector for the Kalman filtering algorithm assumed to be equal to matrix Q

Initial states of factors that follow an Ornstein-Uhlenbeck have a transient effect on future observations, however the initial value of a random walk variable persists across observations and therefore influencing model fit more (see Schwartz and Smith (2000) for more details).

Value

`NFCP.Kalman.filter` returns a `numeric` object when `verbose = F`, which corresponds to the log-likelihood of observations. When `verbose = T`, the `NFCP.Kalman.filter` function returns a `list` object of length seven with the following objects:

`LL`	Log-Likelihood of observations
`X.t`	`vector`. The final observation of the state vector
`X`	`matrix`. All observations of the state vector, after the updating equation has been applied
`Y`	`matrix`. Estimated futures prices at each observation
`V`	`matrix`. Estimation error of each futures contracts at each observation
`Filtered.Error`	`matrix`. positive mean error (high bias), negative mean error (low bias), mean error (bias) and root
`TSFit.Error`	`matrix`. The Mean Error (Bias), Mean Absolute Error, Standard Deviation of Error and Root Mean
`TSFit.Volatility`	`matrix`. The theoretical and empirical volatility of futures returns for each observed contract as ret

When `debugging = T`, 9 objects are returned in addition to those returned when `verbose = T`:

`P_t`	`array`. The covariance matrix at each observation point, with the third dimension indexing across time
`F_t`	`vector`. The function of the Kalman filter covariance matrix at each observation point, with the third dimension inc
`K_t`	`matrix`. The Kalman Gain at each observation point, with the third dimension indexing across time
`d`	`matrix`. d_t (see **details**)
`Z`	`matrix`. Z_t (see **details**)
`G_t`	`matrix`. G_t (see **details**)
`c_t`	`vector`. C_t (see **details**)
`Q_t`	`matrix`. Q_t (see **details**)
`H`	`matrix`. H (see **details**)

References

Anderson, B. D. O. and J. B. Moore, (1979). *Optimal filtering* Englewood Cliffs: Prentice-Hall.

Fahrmeir, L. and G. tutz,(1994) *Multivariate Statistical Modelling Based on Generalized Linear Models.* Berlin: Springer.

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

Cortazar, G., and L. Naranjo, (2006). An N-factor Gaussian model of oil futures prices. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 26(3), 243-268.

Durbin, J., and S. J. Koopman, (2012). *Time series analysis by state space methods.* Oxford university press.

Examples

```
##Example 1 - complete, stitched data.
##Replicating the Schwartz and Smith (2000)
##Two-Factor commodity pricing model applied to crude oil:

Schwartz.Smith.Oil.Stitched <- NFCP.Kalman.filter(
 parameter.values = SS.Oil$Two.Factor,
 parameters = names(SS.Oil$Two.Factor),
 log.futures = log(SS.Oil$Stitched.Futures),
 TTM = SS.Oil$Stitched.TTM,
 dt = SS.Oil$dt,
 verbose = FALSE)


##Example 2 - incomplete, contract data.
##Replicating the Schwartz and Smith (2000)
##Two-Factor commodity pricing model applied to all available
##crude oil contracts:

SS.Oil.2F <- SS.Oil$Two.Factor
##omit stitched contract white noise
SS.Oil.2F <- SS.Oil.2F[!grepl("sigma.contract",
                              names(SS.Oil.2F))]
##Assume constant white noise in observable contracts:
SS.Oil.2F["sigma.contracts"] <- 0.01

#Kalman filter
Schwartz.Smith.Oil.Contracts <- NFCP.Kalman.filter(
 parameter.values = SS.Oil.2F,
 parameters = names(SS.Oil.2F),
 ## All available contracts are considered
 log.futures = log(SS.Oil$Contracts),
 ## Respective 'TTM' of these contracts are input
 TTM = SS.Oil$Contract.Maturities,
 dt = SS.Oil$dt,
 verbose = FALSE)
```

`NFCP.MLE` *N-Factor Model Parameter Estimation through the Kalman Filter and Maximum Likelihood Estimation*

Description

the `NFCP.MLE` function performs parameter estimation of an n-factor model given observable term structure futures data through maximum likelihood estimation. `NFCP.MLE` allows for missing observations as well as constant or variable time to maturity of observed futures contracts.

Usage

```
NFCP.MLE(
  log.futures,
  dt,
  TTM,
  N.factors,
  GBM = TRUE,
  S.Constant = TRUE,
  Estimate.Initial.State = FALSE,
  Richardsons.Extrapolation = TRUE,
  cluster = FALSE,
  Domains = NULL,
  ...
)
```

Arguments

`log.futures`	Object of class `matrix` corresponding to the natural logarithm of observable futures prices. NA's are allowed within the `matrix`. Every column of the matrix must correspond to a particular futures contract, with each row corresponding to a quoted price on a given date.
`dt`	Constant, discrete time step of observations
`TTM`	Object of class `vector` or `matrix` that specifies the time to maturity of observed futures contracts. time to maturity can be either constant (i.e. class `vector`) or time dependent (i.e. class `matrix`). When the time to maturity of observed futures contracts is time dependent, the dimensions of `TTM` must be identical to that of `log.futures`. Every element of `TTM` corresponds to the time to maturity, in years, of a futures contract at a given observation date.
`N.factors`	`numeric`. Number of state variables in the spot price process.
`GBM`	`logical`. When `TRUE`, factor 1 of the model is assumed to follow a Brownian Motion, inducing a unit-root in the spot price process.
`S.Constant`	`logical`. When `TRUE`, the white noise of observable contracts are assumed identical and independent.
`Estimate.Initial.State`	`logical`. When `TRUE`, the initial state vector is specified as unknown parameters of the commodity pricing model. When `FALSE`, a "diffuse" assumption is taken instead (see **details**)
`Richardsons.Extrapolation`	`logical`. When `TRUE`, the `grad` function from the `numDeriv` package is called to approximate the gradient within the `genoud` optimization algorithm.
`cluster`	an optional object of the 'cluster' class returned by one of the makeCluster commands in the `parallel` package to allow for parameter estimation to be performed across multiple cluster nodes.
`Domains`	an optional `matrix` of the lower and upper bounds for the parameter estimation process. The `NFCP.Domains` function is highly recommended. When `Domains` is not specified, the standard bounds specified within the `NFCP.Domains` function are used.
`...`	additional arguments to be passed into the genoud function. See `help(genoud)`

Details

`NFCP.MLE` is a wrapper function that uses the genetic algorithm optimization function `genoud` from the `rgenoud` package to optimize the log-likelihood score returned from the `NFCP.Kalman.filter` function. When `Richardsons.Extrapolation = TRUE`, gradients are approximated numerically within the optimization algorithm through the `grad` function from the `numDeriv` package. `NFCP.MLE` is designed to perform parameter estimation as efficiently as possible, ensuring a global optimum is reached even with a large number of unknown parameters and state variables. Arguments passed to the genoud function can greatly influence estimated parameters and must be considered when performing parameter estimation. Recommended arguments to pass into the genoud function are included within the vignette of NFCP. All arguments of the genoud function may be passed through the NFCP.MLE function (except for gradient.check, which is hard set to false).

NFCP.MLE performs boundary constrained optimization of log-likelihood scores and does not allow does not allow for out-of-bounds evaluations within the genoud optimization process, preventing candidates from straying beyond the bounds provided by Domains. When Domains is not specified, the default bounds specified by the NFCP.Domains function are used.

The N-factor model The N-factor model was first presented in the work of Cortazar and Naranjo (2006, equations 1-3). The N-factor framework describes the spot price process of a commodity as the correlated sum of N state variables x_t.

When GBM = TRUE:

$$log(S_t) = \sum_{i=1}^{N} x_{i,t}$$

When GBM = FALSE:

$$log(S_t) = E + \sum_{i=1}^{N} x_{i,t}$$

Additional factors within the spot-price process are designed to result in additional flexibility, and possibly fit to the observable term structure, in the spot price process of a commodity. The fit of different N-factor models, represented by the log-likelihood can be directly compared with statistical testing possible through a chi-squared test.

Flexibility in the spot price under the N-factor framework allows the first factor to follow a Brownian Motion or Ornstein-Uhlenbeck process to induce a unit root. In general, an N-factor model where GBM = T allows for non-reversible behaviour within the price of a commodity, whilst GBM = F assumes that there is a long-run equilibrium that the commodity price will revert to.

State variables are thus assumed to follow the following processes:

When GBM = TRUE:

$$dx_{1,t} = \mu^* dt + \sigma_1 dw_1 t$$

When GBM = FALSE:

$$dx_{1,t} = -(\lambda_1 + \kappa_1 x_{1,t})dt + \sigma_1 dw_1 t$$

And:

$$dx_{i,t} =_{i \neq 1} -(\lambda_i + \kappa_i x_{i,t})dt + \sigma_i dw_i t$$

where:

$$E(w_i)E(w_j) = \rho_{i,j}$$

The following constant parameters are defined as:

var μ: long-term growth rate of the brownian motion process.

var E: Constant equilibrium level.

var $\mu^* = \mu - \lambda_1$: Long-term risk-neutral growth rate

var λ_i: Risk premium of state variable i.

var κ_i: Reversion rate of state variable i.

var σ_i: Instantaneous volatility of state variable i.

var $\rho_{i,j} \in [-1, 1]$: Instantaneous correlation between state variables i and j.

Measurement Error:

The Kalman filtering algorithm assumes a given measure of measurement error (ie. matrix H). Measurement errors can be interpreted as error in the model's fit to observed prices, or as errors in the reporting of prices (Schwartz and Smith, 2000) and are assumed independent.

var s_i Observation error of contract i.

When `S.Constant = T`, the values of parameter s_i are assumed constant and equal to parameter object 'sigma.contracts'. When `S.Constant = F`, observation errors are assumed unique, where the error of futures contracts s_i is equal to parameter object `'sigma.contract_'` [i] (i.e. `'sigma.contract_1'`, `'sigma.contract_2'`, ...).

Diffuse Kalman Filtering

If `Estimate.Initial.State = F`, a 'diffuse' assumption is used within the Kalman filtering algorithm. Factors that follow an Ornstein-Uhlenbeck are assumed to equal zero. When `Estimate.Initial.State = F` and `GBM = T`, the initial value of the first state variable is assumed to equal the first element of `log.futures`. This is an assumption that the initial estimate of the spot price is equal to the closest to maturity observed futures price.

The initial covariance of the state vector for the Kalman Filtering algorithm assumed to be equal to matrix Q

Initial states of factors that follow an Ornstein-Uhlenbeck process are generally not estimated with a high level of precision, due to the transient effect of the initial state vector on future observations, however the initial value of a random walk variable persists across observations (see Schwartz and Smith (2000) for more details).

Value

`NFCP.MLE` returns a `list` with 10 objects. 9 objects are returned when the user has specified not to calculate the hessian matrix at solution.

`MLE`	`numeric` The Maximum-Likelihood-Estimate of the solution
`Estimated.Parameters`	`vector`. The estimated parameters
`Standard.Errors`	`vector`. Standard error of the estimated parameters. Returned only when `hessian = T` is speci
`X.t`	`vector`. The final observation of the state vector
`X`	`matrix`. All observations of the state vector, after the updating equation has been applied
`Y`	`matrix`. Estimated futures prices at each observation
`V`	`matrix`. Estimation error of each futures contracts at each observation
`TSFit.Error`	`matrix`. The Mean Error (Bias), Mean Absolute Error, Standard Deviation of Error and Root
`TSFit.Volatility`	`matrix`. The theoretical and empirical volatility of futures returns for each observed contract a
`proc_time`	`list`. The real and CPU time (in seconds) the `NFCP.MLE` function has taken.
`genoud.value`	`list`. The output of the called genoud function.

References

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

Cortazar, G., and L. Naranjo, (2006). An N-factor Gaussian model of oil futures prices. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 26(3), 243-268.

Mebane, W. R., and J. S. Sekhon, (2011). Genetic Optimization Using Derivatives: The rgenoud Package for R. *Journal of Statistical Software*, 42(11), 1-26. URL http://www.jstatsoft.org/v42/i11/.

Examples

```
##Example 1 - Perform One Generation of Maximum Likelihood Estimation on the
##first 20 weekly observations of the Schwartz and Smith (2000) Crude Oil Data:
##Example 1 - Complete, Stitched Data:
Schwartz.Smith.Two.Factor.Stitched <- NFCP.MLE(
 ####Arguments
 log.futures = log(SS.Oil$Stitched.Futures)[1:5,1],
 dt = SS.Oil$dt,
 TTM= SS.Oil$Stitched.TTM[1],
 S.Constant = FALSE, N.factors = 1, GBM = TRUE,
 ####Genoud arguments:
 hessian = TRUE,
 Richardsons.Extrapolation = FALSE,
 pop.size = 4, optim.method = "L-BFGS-B", print.level = 0,
 max.generations = 0, solution.tolerance = 10)

##Example 2 - Incomplete, Contract Data:
Schwartz.Smith.Two.Factor.Contract <- NFCP.MLE(
 ####Arguments
 log.futures = log(SS.Oil$Contracts)[1:20,1:5],
 dt = SS.Oil$dt,
 TTM= SS.Oil$Contract.Maturities[1:20,1:5],
 S.Constant = TRUE,
 N.factors = 1, GBM = TRUE,
 ####Genoud arguments:
 hessian = TRUE,
 Richardsons.Extrapolation = FALSE,
 pop.size = 4, optim.method = "L-BFGS-B", print.level = 0,
 max.generations = 0, solution.tolerance = 10)
```

`NFCP.Parameters` *Specify parameters of N-factor model*

Description

the `NFCP.Parameters` function specifies the parameters of a commodity pricing model under the N-factor framework first described by Cortazar and Naranjo (2006). This function is a recommended starting position for the application of N-factor models within the `NFCP` package.

Usage

```
NFCP.Parameters(
  N.factors,
  GBM,
  Initial.State,
  S.Constant,
  N.contracts = NULL,
  verbose = TRUE
)
```

Arguments

`N.factors`	`numeric`. Number of state variables in the spot price process.
`GBM`	`logical`. If `GBM = T`, factor 1 of the model is assumed to follow a Brownian Motion, inducing a unit-root in the spot price process.
`Initial.State`	`logical`. If `Initial.State = T`, the initial state vector is specified as unknown parameters of the commodity pricing model.
`S.Constant`	`logical`. If `S.Constant = T`, the white noise of observable contracts are assumed and identical (and independent).
`N.contracts`	`numeric`. The number of individual observation white noise terms when `S.Constant = F`. Optional when `S.Constant = T`. When `N.contracts = 0`, the value of `NFCP.Parameters` returns a vector without any "sigma.contract" or "sigma.contracts" elements.
`verbose`	`logical`. If `verbose = T`, the specified N-factor model is printed when the function is called.

Details

The N-factor model The N-factor model was first presented in the work of Cortazar and Naranjo (2006, equations 1-3). The N-factor framework describes the spot price process of a commodity as the correlated sum of N state variables x_t.

When `GBM = TRUE`:

$$log(S_t) = \sum_{i=1}^{N} x_{i,t}$$

When `GBM = FALSE`:

$$log(S_t) = E + \sum_{i=1}^{N} x_{i,t}$$

Additional factors within the spot-price process are designed to result in additional flexibility, and possibly fit to the observable term structure, in the spot price process of a commodity. The fit of different N-factor models, represented by the log-likelihood can be directly compared with statistical testing possible through a chi-squared test.

Flexibility in the spot price under the N-factor framework allows the first factor to follow a Brownian Motion or Ornstein-Uhlenbeck process to induce a unit root. In general, an N-factor model where `GBM = T` allows for non-reversible behaviour within the price of a commodity, whilst `GBM = F` assumes that there is a long-run equilibrium that the commodity price will revert to in the long-term.

State variables are thus assumed to follow the following processes:

When `GBM = TRUE`:

$$dx_{1,t} = \mu^* dt + \sigma_1 dw_1 t$$

When `GBM = FALSE`:

$$dx_{1,t} = -(\lambda_1 + \kappa_1 x_{1,t})dt + \sigma_1 dw_1 t$$

And:

$$dx_{i,t} =_{i\neq 1} -(\lambda_i + \kappa_i x_{i,t})dt + \sigma_i dw_i t$$

where:

$$E(w_i)E(w_j) = \rho_{i,j}$$

The following constant parameters are defined as:

`var` μ: long-term growth rate of the Brownian Motion process.

`var` E: Constant equilibrium level.

`var` $\mu^* = \mu - \lambda_1$: Long-term risk-neutral growth rate

`var` λ_i: Risk premium of state variable i.

`var` κ_i: Reversion rate of state variable i.

`var` σ_i: Instantaneous volatility of state variable i.

`var` $\rho_{i,j} \in [-1, 1]$: Instantaneous correlation between state variables i and j.

Measurement Error:

The Kalman filtering algorithm assumes a given measure of measurement error (ie. matrix H). Measurement errors can be interpreted as error in the model's fit to observed prices, or as errors in the reporting of prices (Schwartz and Smith, 2000) and are assumed independent.

`var` s_i Observation error of contract i.

When `S.Constant = T`, the values of parameter s_i are assumed constant and equal to parameter object 'sigma.contracts'. When `S.Constant = F`, observation errors are assumed unique, where the error of futures contracts s_i is equal to parameter object `'sigma.contract_'` [i] (i.e. `'sigma.contract_1'`, `'sigma.contract_2'`, ...).

When `N.contracts = 0`, "sigma.contract" parameters are not returned within the parameter vector.

Diffuse Assumption: If `Initial.State = F`, a 'diffuse' assumption is made within Kalman filtering and parameter estimation functions (See `NFCP.MLE` or `NFCP.Kalman.filter` for more information)

Value

A vector of parameter names for a specified N-factor spot price process. This vector is ideal for application within many other functions within the `NFCP` package

References

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

Cortazar, G., and L. Naranjo, (2006). An N-factor Gaussian model of oil futures prices. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 26(3), 243-268.

Examples

```
##Generate parameter of a Two-factor model Crude Oil model
##as first presented by Schwartz and Smith (2000):
SS.Oil.Two.Factor.parameters <- NFCP.Parameters(N.factors = 2,
                                                GBM = TRUE,
                                                Initial.State = FALSE,
                                                S.Constant = FALSE,
                                                N.contracts = 5)
print(SS.Oil.Two.Factor.parameters)
```

`Spot.Price.Forecast` *Forecast N-Factor Model Spot Prices*

Description

Analytically forecast expected spot prices following the "true" process of a given n-factor stochastic model

Usage

```
Spot.Price.Forecast(X.0, parameters, t, Percentiles = NULL)
```

Arguments

`X.0`	Initial values of the state vector.
`parameters`	A named vector of parameter values of a specified N-factor model. Function `NFCP.Parameters` is recommended.
`t`	a vector of discrete time points to forecast
`Percentiles`	Optional. A vector of percentiles to include probabilistic forecasting intervals.

Details

Future expected spot prices under the N-factor model can be forecasted through the analytic expression of expected future prices under the "true" N-factor process.

Given that the log of the spot price is equal to the sum of the state variables (equation 1), the spot price is log-normally distributed with the expected prices given by:

$$E[S_t] = exp(E[ln(S_t)] + \frac{1}{2}Var[ln(S_t)])$$

Where:

$$E[ln(S_t)] = \sum_{i=1}^{N} e^{-(\kappa_i t)} x_i(0) + \mu t$$

Where $\kappa_i = 0$ when `GBM=T` and $\mu = 0$ when `GBM = F`

$$Var[ln(S_t)] = \sigma_1^2 t + \sum_{i.j \neq 1} \sigma_i \sigma_j \rho_{i,j} \frac{1 - e^{-(\kappa_i + \kappa_j)t}}{\kappa_i + \kappa_j}$$

and thus:

$$E[S_t] = exp(\sum_{i=1}^{N} e^{-\kappa_i t} x_i(0) + (\mu + \frac{1}{2}\sigma_1^2)t + \frac{1}{2}\sum_{i.j \neq 1} \sigma_i \sigma_j \rho_{i,j} \frac{1 - e^{-(\kappa_i + \kappa_j)t}}{\kappa_i + \kappa_j})$$

Under the assumption that the first factor follows a Brownian Motion, in the long-run expected spot prices grow over time at a constant rate of $\mu + \frac{1}{2}\sigma_1^2$ as the $e^{-\kappa_i t}$ and $e^{-(\kappa_i + \kappa_j)t}$ terms approach zero.

An important consideration when forecasting spot prices using parameters estimated through maximum likelihood estimation is that the parameter estimation process takes the assumption of risk-neutrality and thus the true process growth rate μ is not estimated with a high level of precision. This can be shown from the higher standard error for μ than other estimated parameters, such as the risk-neutral growth rate μ^*. See Schwartz and Smith (2000) for more details.

Value

`Spot.Price.Forecast` returns a vector of expected future spot prices under a given N-factor model at specified discrete future time points. When `percentiles` are specified, the function returns a matrix with the corresponding confidence bands in each column of the matrix.

References

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

Cortazar, G., and L. Naranjo, (2006). An N-factor Gaussian model of oil futures prices. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 26(3), 243-268.

Examples

```
# Forecast the Schwartz and Smith (2000) two-factor oil model:

##Step 1 - Run the Kalman filter for the two-factor oil model:
Schwartz.Smith.Oil <- NFCP.Kalman.filter(SS.Oil$Two.Factor,
                                      names(SS.Oil$Two.Factor),
                                      log(SS.Oil$Stitched.Futures),
                                      SS.Oil$dt,
                                      SS.Oil$Stitched.TTM,
                                      verbose = TRUE)

##Step 2 - Probabilistic forecast of n-factor stochastic differential equation (SDE):
E.Spot <- Spot.Price.Forecast(X.0 = Schwartz.Smith.Oil$X.t,
                                 parameters = SS.Oil$Two.Factor,
                                 t = seq(0,9,1/12),
                                 Percentiles = c(0.1, 0.9))
```

`Spot.Price.Simulate` *Simulate N-Factor Model Spot Prices*

Description

Simulate risk-neutral price paths of an an N-factor commodity pricing model through Monte Carlo Simulation.

Usage

```
Spot.Price.Simulate(
  X.0,
  parameters,
  t = 1,
  dt = 1,
  n = 2,
  antithetic = TRUE,
  verbose = FALSE
)
```

Arguments

`X.0`	Initial values of the state vector.
`parameters`	A named vector of parameter values of a specified N-factor model. Function `NFCP.Parameters` is recommended.
`t`	the number of years to simulate
`dt`	discrete time step of simulation
`n`	total number of simulations
`antithetic`	`logical`. Should antithetic price paths be simulated?
`verbose`	`logical`. Should simulated state variables be output? see **returns**

Details

The `Spot.Price.Simulate` function is able to quickly simulate a large number of risk-neutral price paths of a commodity following the N-factor model. Simulating risk-neutral price paths of a commodity under an N-factor model through Monte Carlo simulations allows for the valuation of commodity related investments and derivatives, such as American Options and Real Options through dynamic programming methods. The `Spot.Price.Simulate` function quickly and efficiently simulates an N-factor model over a specified number of years, simulating antithetic price paths as a simple variance reduction technique. The `Spot.Price.Simulate` function uses the `mvrnorm` function from the `MASS` package to draw from a multivariate normal distribution for the simulation shocks.

The N-factor model stochastic differential equation is given by:

Brownian Motion processes (ie. factor one when `GBM = T`) are simulated using the following solution:

$$x_{1,t+1} = x_{1,t} + \mu^* \Delta t + \sigma_1 \Delta t Z_{t+1}$$

Where Δt is the discrete time step, μ^* is the risk-neutral growth rate and σ_1 is the instantaneous volatility. Z_t represents the independent standard normal at time t.

Ornstein-Uhlenbeck Processes are simulated using the following solution:

$$x_{i,t} = x_{i,0}e^{-\kappa_i t} - \frac{\lambda_i}{\kappa_i}(1 - e^{-\kappa_i t}) + \int_0^t \sigma_i e^{\kappa_i s} dW_s$$

Where a numerical solution is obtained by numerically discretising and approximating the integral term using the Euler-Maruyama integration scheme:

$$\int_0^t \sigma_i e^{\kappa_i s} dW_s = \sum_{j=0}^{t} \sigma_i e^{\kappa_i j} dW_s$$

Value

`Spot.Price.Simulate` returns a list when `verbose = T` and a matrix of simulated price paths when `verbose = F`. The returned objects in the list are:

`State_Variables`	A matrix of simulated state variables for each factor is returned when `verbose = T`. The number of fa
`Prices`	A matrix of simulated price paths. Each column represents one simulated price path and each row re

References

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

Cortazar, G., and L. Naranjo, (2006). An N-factor Gaussian model of oil futures prices. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 26(3), 243-268.

Examples

```
# Example 1
###Simulate a Geometric Brownian Motion (GBM) process:
## Starting price of 20, with a growth of 5% p.a. and
## volatility of 20% p.a.
Simulated.Spot.Prices <- Spot.Price.Simulate(
 X.0 = log(20),
 parameters = c(mu_star = (0.05 - (1/2) * 0.2^2), sigma_1 = 0.2),
 t = 1,
 dt = 1/12,
 n = 1e3)

# Example 2
###Simulate future spot price paths under Risk-Neutrality and under the
###Schwartz - Smith two factor model:

##Step 1 - Run the Kalman Filter for the Two-Factor Oil Model:
Schwartz.Smith.Oil <- NFCP.Kalman.filter(parameter.values = SS.Oil$Two.Factor,
                                       parameters = names(SS.Oil$Two.Factor),
                                       log.futures = log(SS.Oil$Stitched.Futures),
                                       dt = SS.Oil$dt,
                                       TTM = SS.Oil$Stitched.TTM,
                                       verbose = TRUE)

#Step 2 - Simulate spot prices:
##100 antithetic simulations of one year of monthly observations
Simulated.Spot.Prices <- Spot.Price.Simulate(
 X.0 = Schwartz.Smith.Oil$X.t,
 parameters = SS.Oil$Two.Factor,
 t = 1,
 dt = 1/12,
 n = 1e3,
 antithetic = TRUE,
 verbose = TRUE)
```

`SS.Oil` *Crude Oil Term Structure Futures Data (1990 - 1995)*

Description

The `SS.Oil` `list` object features the approximate weekly observations of Crude Oil (WTI) futures contracts used to develop a two-factor commodity pricing model within the prominent work of Schwartz and Smith (2000) titled: "Short-Term Variations and long-Term Dynamics in Commodity Prices". The two-factor commodity pricing model presented within this study is also included. The `SS.Oil` list object is used extensively within the `NFCP` package to provide working examples and showcase the features of the package.

Usage

```
data(SS.Oil)
```

Format

A `list` Containing eight objects:

Contracts A data frame with 268 rows and 82 columns. Each column represents a Crude Oil futures contract, and each row represents a closing weekly price for that futures contract. Observation dates of the contract object are weekly in frequency from `1990-02-06` to `1995-02-14`. Contracts without observations on a particular date are represented as `NA`.

Stitched.Futures Schwartz and Smith (2000) applied stitched contract observation data to estimate commodity pricing models, which are approximated within this object. The `Stitched.Futures` object was developed using the `Stitch.Contracts` function (see `Stitch.Contracts` examples for more details). Contracts were stitched according to the contract numbers specified within the object `Stitched.TTM`. `Stitched.Futures` is identical to the futures data made available within the MATLAB program "SchwartzSmithModel" developed by Goodwin (2013).

Spot A `data.frame` of spot prices of Crude Oil. weekly in frequency from `1990-02-06` to `1995-02-14`.

Final.Trading.Days Named vector listing the final trading days of each observed futures contract within the `Contracts` object. Each element of `Final.Trading.Days` corresponds to a column of the `Contracts` object. The final trading day of a futures contract is used to calculate the number of business days from a given observation to the maturity of the contract (ie. a contract time to maturity).

Contract.Maturities A data frame with identical dimensions to the `Contracts` data frame. This data frame lists the time to maturity of a given futures contract in years at each observation point. This is identical to the number of business days (in years) between the observed date and the final trading day of a particular futures contract. The maturity matrix assumes 262 trading days a year. If the contract is not yet available or has expired, the `Contract.Maturities` element is `NA`.

Stitched.TTM A vector corresponding to the constant time to maturities that was assumed within the original study of Schwartz and Smith (2000).

dt The discrete time step used to estimate parameters with this data. The time step is 5/262, which represents a weekly frequency of observations where each weekday is a business day (ie. there are no business days on weekends).

Two.Factor The crude oil two-factor commodity pricing model parameters presented within the work of Schwartz and Smith (2000). These parameter estimates are prolific, benchmarked within several subsequent publications.

References

Dominice Goodwin (2013). Schwartz-Smith 2-factor model - Parameter estimation (https://www.mathworks.com/matlab schwartz-smith-2-factor-model-parameter-estimation), MATLAB Central File Exchange. Retrieved November 21, 2020.

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

`Stitch.Contracts` *Stitch Futures Contracts*

Description

Aggregate futures contract price data by stitching according to either approximate maturities and rollover frequency or contract number from closest maturity.

Usage

```
Stitch.Contracts(
  Futures,
  TTM = NULL,
  maturity.matrix = NULL,
  rollover.frequency = NULL,
  Contract.Numbers = NULL,
  verbose = FALSE
)
```

Arguments

`Futures`	Contract futures price data. Each row of `Futures` should represent one observation of futures prices and each column should represent one quoted futures contract. NA's in `Futures` are allowed, representing missing observations.
`TTM`	A `vector` of contract maturities to stitch
`maturity.matrix`	The time-to-maturity (in years) for each contract at each given observation point. The dimensions of `maturity.matrix` should match those of `Futures`
`rollover.frequency`	the frequency (in years) at which contracts should be rolled over
`Contract.Numbers`	A `vector` of contract numbers offset from the closest-to-maturity contract at which to stitch contracts.
`verbose`	`logical`. Should additional information be output? see **details**

Details

This function aggregates a set of futures contract data by stitching contract data over an observation period, resulting in a set of futures observations that is 'complete' (ie. Does not feature missing observations). Aggregated futures data benefit from several computational efficiencies compared to raw contract data, but results in the loss of futures price information.

There are two methods of the `Stitch.Contracts` function that can be utilized the stitch contracts:

Method 1

`Stitch.Contracts(Futures,Contract.Numbers,verbose = T)` Futures data may be aggregated by stitching prices according to maturity matching. This method requires the inputs `TTM`, `maturity.matrix` and `rollover.frequency`. This method stitched contracts by matching the observation prices according to which contract has the closest time-to-maturity of the desired maturity specified in `TTM`. Contracts are rolled over at the frequency specified in `rollover.frequency`.

Method 2

`Stitch.Contracts(Futures,TTM,maturity.matrix,rollover.frequency,verbose = T)` Futures data may be stitched according to the contract numbers offset from the closest-to-maturity contract. This method requires only the input `Contract.Numbers` specifying which contracts should be included. This method is most appropriate when the maturity of available contracts are consistent (ie. contracts expire every month or three months).

Value

`Stitch.Contracts` returns a matrix of stitched futures prices if `verbose = T` and a list with two or three objects otherwise (see below).

`Prices`	A data frame of Stitched futures prices. Each row represents an observation of the specified contracts.
`Maturities`	A data frame of the time-to-maturity of observed futures prices. Each row represents an observation of the
`Tickers`	A data frame of the named columns of observed futures prices (e.g. contract tickers). Returned only when

References

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

Cortazar, G., and L. Naranjo, (2006). An N-factor Gaussian model of oil futures prices. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 26(3), 243-268.

Examples

```
##These examples approximately replicate the Crude Oil data utilized within the
##prominent work of Schwartz and Smith (2000):

###Method 1 - Stitch crude oil contracts according to maturity matching:
SSOilStitched.M1 <- Stitch.Contracts(Futures = SS.Oil$Contracts,
TTM = c(1, 5, 9, 13, 17)/12, maturity.matrix = SS.Oil$Contract.Maturities,
rollover.frequency = 1/12, verbose = TRUE)

###Method 2 - Stitch crude oil contracts according to nearest contract numbers:
SSOilStitched.M2 <- Stitch.Contracts(Futures = SS.Oil$Contracts,
Contract.Numbers = c(1, 5, 9, 13, 17), verbose = TRUE)
```

`TSFit.Volatility` *Volatility Term Structure of futures returns*

Description

Estimate the Theoretical and Empirical Volatility Term Structure of futures returns

Usage

```
TSFit.Volatility(parameters, Futures, TTM, dt)
```

Arguments

`parameters`	A named vector of parameters of an N-factor model. Function `NFCP.Parameters` is recommended.
`Futures`	A Matrix of futures price data. Each column corresponds to a given futures contract, and each row is an observation of the futures contracts.
`TTM`	A vector listing the Time to Maturities of each listed Futures contract from the current observation point.
`dt`	Numeric. The length of the discrete time step (years).

Details

The fit of the models theoretical volatility term structure of futures returns to those obtained directly from observed futures prices can be used as an additional measure of robustness for the models ability to explain the behavior of a commodities term structure. A commodity pricing model should capture all dynamics of a commodities term structure,

The theoretical model volatility term structure of futures returns is given by the following equation:

$$\sigma_F(\tau) = \sum_{i=1}^{N}\sum_{j=1}^{N} \sigma_i \sigma_j \rho_{i,j} e^{-(\kappa_i+\kappa_j)\tau}$$

Under the case that $\kappa_1 = 0$, the model volatility term structure converges to σ_1^2 as τ grows large.

The empirical volatility term structure of futures returns is given by:

$$\hat{\sigma}_F^2(\tau) = \frac{1}{\Delta t}\sum_{i=1}^{N}(log(F(t_i,\tau)/F(t_i-\Delta t,\tau)) - \bar{\mu})^2$$

According to Cortazar and Naranjo (2006): "A larger number of factors gives more flexibility to adjust first and second moments simultaneously, hence explaining why (a) four-factor (may) outperform (a) three-factor one in fitting the volatility term structure."

Value

`TSFit.Volatility` returns a matrix with the theoretical and empirical volatility term structure of futures returns, with the number of columns of this matrix coinciding with the number of input futures contracts.

References

Schwartz, E. S., and J. E. Smith, (2000). Short-Term Variations and Long-Term Dynamics in Commodity Prices. *Manage. Sci.*, 46, 893-911.

Cortazar, G., and L. Naranjo, (2006). An N-factor Gaussian model of oil futures prices. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 26(3), 243-268.

Examples

```
### Test the volatility term structure fit of the Schwartz-Smith two-factor model on crude oil:
V_TSFit <- TSFit.Volatility(
 parameters = SS.Oil$Two.Factor,
 Futures = SS.Oil$Stitched.Futures,
 TTM = SS.Oil$Stitched.TTM,
 dt = SS.Oil$dt)
```

Index