

Package ‘SVDNF’

September 5, 2024

Type Package

Title Discrete Nonlinear Filtering for Stochastic Volatility Models

Version 0.1.9

Author Louis Arsenault-Mahjoubi [aut, cre],
Jean-François Bégin [aut],
Mathieu Boudreault [aut]

Maintainer Louis Arsenault-Mahjoubi <larsenau@sfu.ca>

Description Implements the discrete nonlinear filter (DNF) of Kitagawa (1987) <doi:10.1080/01621459.1987.10478534> to a wide class of stochastic volatility (SV) models with return and volatility jumps following the work of Bégin and Boudreault (2021) <doi:10.1080/10618600.2020.1840995>. Offers several built-in SV models and a flexible framework for users to create customized models by specifying drift and diffusion functions along with a jump arrival distribution for the return and volatility dynamics. Allows for the estimation of factor models with stochastic volatility (e.g., heteroskedastic volatility CAPM) by incorporating expected return predictors. Includes functions to compute likelihood evaluations, filtering and prediction distribution estimates, maximum likelihood parameter estimates, to simulate data from built-in and custom SV models with jumps, and to forecast future returns and volatility values using Monte Carlo simulation from a given SV model.

License GPL-3

Encoding UTF-8

Imports Rcpp (>= 1.0.9), methods, zoo, xts

LinkingTo Rcpp

RoxygenNote 7.2.3

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-09-04 22:30:14 UTC

Contents

| | |
|---------------------------|---|
| DNF.dynamicsSVM | 2 |
|---------------------------|---|

| | |
|---------------------------------|----|
| DNFOptim.dynamicsSVM | 4 |
| dynamicsSVM | 7 |
| extractVolPerc.SVDNF | 10 |
| logLik.SVDNF | 11 |
| modelSim.dynamicsSVM | 12 |
| pars.dynamicsSVM | 13 |
| plot.predict.DNFOptim | 14 |
| plot.SVDNF | 16 |
| predict.DNFOptim | 17 |
| summary.DNFOptim | 19 |

Index 20

| | |
|-----------------|--|
| DNF.dynamicsSVM | <i>Discrete Nonlinear Filtering Algorithm for Stochastic Volatility Models</i> |
|-----------------|--|

Description

The DNF function applies the discrete nonlinear filter (DNF) of Kitagawa (1987) as per the implementation of Bégin & Boudreault (2020) to obtain likelihood evaluations and filtering distribution estimates for a wide class of stochastic volatility models.

Usage

```
## S3 method for class 'dynamicsSVM'
DNF(dynamics, data, factors = NULL, N = 50, K = 20, R = 1, grids, ...)
```

Arguments

| | |
|----------|---|
| dynamics | A dynamicsSVM object representing the model dynamics to be used by the DNF. |
| data | A series of asset returns for which we want to run the DNF. This should be a vector or an xts object. |
| factors | Series of values taken by d explanatory variables. This should be a matrix or an xts object with d rows and T columns. |
| N | Number of nodes in the variance grid. |
| K | Number of nodes in the jump size grid. |
| R | Maximum number of jumps used in the numerical integration at each timestep. |
| grids | Grids to be used for numerical integration by the DNF function. The DNF function creates grids for built-in models. However, this arguments must be provided for custom models. It should contain a list of three sequences: var_mid_points (variance mid-point sequence), j_nums (sequence for the number of jumps), and jump_mid_points (jump mid-point sequence). If there are no variance jumps in the model, set jump_mid_points equal to zero. If there are no jumps in the model, both j_nums and jump_mid_points should be set to zero. |
| ... | Further arguments passed to or from other methods. |

Value

| | |
|----------------|---|
| log_likelihood | Log-likelihood evaluation based on the DNF. |
| filter_grid | Grid of dimensions N by T+1 that stores each time-step's filtering distributions (we assume the filtering distribution is uniform at $t = 0$). |
| likelihoods | Likelihood contribution at each time-step throughout the series. |
| grids | List of grids used for numerical integration by the DNF. |
| dynamics | The model dynamics used by the DNF. |
| data | The series of asset returns to which the DNF was applied. |

References

- Bégin, J.F., Boudreault, M. (2021) Likelihood evaluation of jump-diffusion models using deterministic nonlinear filters. *Journal of Computational and Graphical Statistics*, 30(2), 452–466.
- Kitagawa, G. (1987) Non-Gaussian state-space modeling of nonstationary time series. *Journal of the American Statistical Association*, 82(400), 1032–1041.

Examples

```

set.seed(1)
# Generate 200 returns from the DuffiePanSingleton model
DuffiePanSingleton_mod <- dynamicsSVM(model = "DuffiePanSingleton")
DuffiePanSingleton_sim <- modelSim(t = 200, dynamics = DuffiePanSingleton_mod)

# Run DNF on the data
dnf_filter <- DNF(data = DuffiePanSingleton_sim$returns,
  dynamics = DuffiePanSingleton_mod)

# Print log-likelihood evaluation.
logLik(dnf_filter)

# Using a custom model.
# Here, we define the DuffiePanSingleton model as a custom model
# to get the same log-likelihood found using the built-in option

# Daily observations
h <- 1/252

# Parameter values
mu <- 0.038; kappa <- 3.689; theta <- 0.032
sigma <- 0.446; rho <- -0.745; omega <- 5.125
delta <- 0.03; alpha <- -0.014; rho_z <- -1.809; nu <- 0.004

# Jump compensator
alpha_bar <- exp(alpha + 0.5 * delta^2) / (1 - rho_z * nu) - 1

# Returns drift and diffusion
mu_y <- function(x, mu, alpha_bar, omega, h) {
  return(h * (mu - x / 2 - alpha_bar * omega))
}

```

```

mu_y_params <- list(mu, alpha_bar, omega, h)
sigma_y <- function(x, h) {
  return(sqrt(h * pmax(x, 0)))
}
sigma_y_params <- list(h)

# Volatility factor drift and diffusion
mu_x <- function(x, kappa, theta, h) {
  return(x + h * kappa * (theta - pmax(0, x)))
}
mu_x_params <- list(kappa, theta, h)

sigma_x <- function(x, sigma, h) {
  return(sigma * sqrt(h * pmax(x, 0)))
}
sigma_x_params <- list(sigma, h)

# Jump distribution for the DuffiePanSingleton Model
jump_density <- dpois
jump_dist <- rpois
jump_params <- c(h * omega)

# Create the custom model
custom_mod <- dynamicsSVM(model = 'Custom',
  mu_x = mu_x, mu_y = mu_y, sigma_x = sigma_x, sigma_y = sigma_y,
  mu_x_params = mu_x_params, mu_y_params = mu_y_params,
  sigma_x_params = sigma_x_params, sigma_y_params = sigma_y_params,
  jump_params = jump_params, jump_dist = jump_dist, jump_density = jump_density,
  nu = nu, rho_z = rho_z, rho = rho)
# Define the grid for DNF
N <- 50; R <- 1; K <- 20
var_mid_points <- seq(from = sqrt(0.0000001),
  to = sqrt(theta + (3 + log(N)) * sqrt(0.5 * theta * sigma^2 / kappa)), length = N)^2

j_nums <- seq(from = 0, to = R, by = 1)

jump_mid_points <- seq(from = 0.000001, to = (3 + log(K)) * sqrt(R) * nu, length = K)

grids <- list(var_mid_points = var_mid_points,
  j_nums = j_nums, jump_mid_points = jump_mid_points)

# Run the DNF function with the custom model
dnf_custom <- DNF(data = DuffiePanSingleton_sim$returns, grids = grids,
  dynamics = custom_mod)

# Check if we get the same log-likelihoods
dnf_custom$log_likelihood; dnf_filter$log_likelihood

```

Description

The DNFOptim function finds maximum likelihood estimates for stochastic volatility models parameters using the DNF function.

Usage

```
## S3 method for class 'dynamicsSVM'
DNFOptim(dynamics, data, par, factors, tol, N = 50, K = 20, R = 1,
  grids = 'Default',
  rho = 0, delta = 0, alpha = 0, rho_z = 0, nu = 0, jump_params_list = "dummy",
  ...)
```

Arguments

| | |
|------------------------------|---|
| dynamics | A dynamicsSVM object representing the model dynamics to be used by the optimizer to find maximum likelihood parameter estimates. This should be a vector or an xts object. |
| data | A series of asset returns for which we want to find maximum likelihood estimates. |
| par | Initial values for the parameters to be optimized over. Information about how to pass the initial parameters is given in the 'Note' section. |
| factors | Series of values taken by d explanatory variables. This should be a matrix or an xts object with d rows and T columns. |
| tol | Tolerance hyperparameter for the optimization. The optim function will be re-run until the difference between the maximum values for the likelihood function is less than tol. After the first optim call, which uses the initial parameters, the next optimizers are run using the previous optimizer's maximum likelihood parameter estimates as initial values. |
| N | Number of nodes in the variance grid. |
| K | Number of nodes in the jump size grid. |
| grids | Grids to be used for numerical integration by the DNF function. The DNF function creates grids for built-in models. However, this arguments must be provided for custom models. It should contain a list of three sequences: var_mid_points (variance mid-point sequence), j_nums (sequence for the number of jumps), and jump_mid_points (jump mid-point sequence). If there are no variance jumps in the model, set jump_mid_points equal to zero. If there are no jumps in the model, both j_nums and jump_mid_points should be set to zero. |
| R | Maximum number of jumps used in the numerical integration at each timestep. |
| rho, delta, alpha, rho_z, nu | See help(dynamicsSVM) for a description of each of these arguments individually. These arguments should be used only for custom models and can be fixed to a certain value (e.g., rho = -0.75). If they are estimated, they should be set to 'var' (e.g., to estimate rho set rho = 'var') and include it in the vector par to be passed to the optim function. See Note for more details on the order in which custom models should receive parameters. |

`jump_params_list` List of the names of the arguments in the jump parameter distribution in the order that they are used by the `jump_dist` function. This is used by `DNFOptim` to check for parameters that occur both in the `jump_dist` function and as arguments in drift or diffusion functions.

`...` Further arguments to be passed to the `optim` function. See Note.

Value

`optim` Returns a list obtained from R's `optim` function. See `help(optim)` for details about the output.

`SVDNF` Returns a `SVDNF` object obtained from running the `DNF` function at the MLE parameter values. See `help(DNF)` for details about the output

`rho, delta, alpha, rho_z, nu` See `help(dynamicsSVM)` for a description of each of these arguments individually. If they are estimated, they are set to 'var'. If the parameters were fixed during the estimation, this will return the value at which they were fixed.

Note

When passing the initial parameter vector `par` to the `optim` function (via `...`), the parameters should follow a specific order.

For the `PittMalikDoucet` model, the parameters should be in the following order: `phi`, `theta`, `sigma`, `rho`, `delta`, `alpha`, and `p`.

For the `DuffiePanSingleton` model, the parameters should be in the following order: `mu`, `alpha`, `delta`, `rho_z`, `nu`, `omega`, `kappa`, `theta`, `sigma`, and `rho`.

All other built-in models can be seen as being nested within these two models (i.e., `Heston` and `Bates` models are nested in the `DuffiePanSingleton` model, while `Taylor` and `TaylorWithLeverage` are nested in the `PittMalikDoucet` model). Their parameters should be passed in the same order as those in the more general models, minus the parameters not found in these nested models.

For example, the `Taylor` model contains neither jumps nor correlation between volatility and returns innovations. Thus, its three parameters are passed in the order: `phi`, `theta`, and `sigma`.

When `models = "Custom"`, parameters should be passed in the following order: `mu_y_params`, `sigma_y_params`, `mu_x_params`, `sigma_x_params`, `rho`, `delta`, `alpha`, `rho_z`, `nu`, and `jump_params`. If an argument is repeated (e.g., both `mu_y_params` and `sigma_y_params` use the same parameter), write it only when it first appears in the custom model order.

References

R Core Team (2019). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.

Examples

```
set.seed(1)
```

```

# Generating return data
Taylor_mod <- dynamicsSVM(model = "Taylor", phi = 0.9,
  theta = -7.36, sigma = 0.363)
Taylor_sim <- modelSim(t = 30, dynamics = Taylor_mod, init_vol = -7.36)
plot(Taylor_sim$volatility_factor, type = 'l')
plot(Taylor_sim$returns, type = 'l')

# Initial values and optimization bounds
init_par <- c(0.7, -5, 0.3)
lower <- c(0.01, -20, 0.1); upper <- c(0.99, 0, 1)

# Running DNFOptim to get MLEs
optim_test <- DNFOptim(data = Taylor_sim$returns,
  dynamics = Taylor_mod,
  par = init_par, lower = lower, upper = upper, method = "L-BFGS-B")

# Parameter estimates
summary(optim_test)

# Plot prediction and filtering distributions
plot(optim_test)

```

dynamicsSVM

Stochastic Volatility Models Dynamics

Description

dynamicsSVM creates stochastic volatility model dynamics by either choosing from a set of built-in model dynamics or using custom drift and diffusion functions, as well as custom jump distributions. See Note for information about how to define custom functions.

Usage

```

dynamicsSVM(mu = 0.038, kappa = 3.689, theta = 0.032, sigma = 0.446,
  rho = -0.745, omega = 5.125, delta = 0.03, alpha = -0.014,
  rho_z = -1.809, nu = 0.004, p = 0.01, phi = 0.965, h = 1/252, coefs = NULL,
  model = "Heston", mu_x, mu_y, sigma_x, sigma_y,
  jump_dist = rpois, jump_density = dpois, jump_params = 0,
  mu_x_params, mu_y_params, sigma_x_params, sigma_y_params)

```

Arguments

| | |
|-------|--|
| mu | Annual expected rate of return. |
| kappa | Variance rate of mean reversion. |
| theta | Unconditional mean variance. |
| sigma | Volatility of the variance. |
| rho | Correlation between the return and the variance noise terms. |

| | |
|--------------------|--|
| omega | Jump arrival intensity for models with Poisson jumps. |
| delta | Standard deviation of return jumps. |
| alpha | Mean of return jumps. |
| rho_z | Pseudo-correlation parameter between return and variance jumps. |
| nu | Mean for variance jumps. |
| p | Jump probability for models with Bernoulli jumps. |
| phi | Volatility persistence parameter. |
| h | Time interval between observations (e.g., $h = 1/252$ for daily data). |
| coefs | Vector of regression coefficients for factor stochastic volatility models. This vector should be of the same length as the number of explanatory variables provided. |
| model | Model used by the discrete nonlinear filter. The options are "Heston", "Bates", "DuffiePanSingleton", "Taylor", "TaylorWithLeverage", "PittMalikDoucet", "CAPM_SV" and "Custom". If model = "Custom", users should pass the drift functions (i.e., μ_x and μ_y), the diffusion functions (i.e., σ_x and σ_y), and the jump distribution, (i.e., jump_dist) as well as their parameters to the dynamicsSVM function. See Examples. |
| μ_x | Function for variance drift (to be used with a custom model). |
| μ_y | Function for returns drift (to be used with a custom model). |
| σ_x | Function for variance diffusion (to be used with a custom model). |
| σ_y | Function for returns diffusion (to be used with a custom model). |
| jump_dist | Distribution used to generate return or volatility jumps at each timestep (if both types of jumps are in the model, they are assumed to occur simultaneously). |
| jump_density | Probability mass function used to compute the probability of return or volatility jumps at each timestep (if both types of jumps are in the model, they are assumed to occur simultaneously). |
| jump_params | List of parameters to be used as arguments in the jump_dist and jump_density function (parameters should be listed in the order that jump_dist uses them). |
| μ_x _params | List of parameters to be used as arguments in the μ_x function (parameters should be listed in the order that μ_x uses them). |
| μ_y _params | List of parameters to be used as arguments in the μ_y function (parameters should be listed in the order that μ_y uses them). |
| σ_x _params | List of parameters to be used as arguments in the σ_x function (parameters should be listed in the order that σ_x uses them). |
| σ_y _params | List of parameters to be used as arguments in the σ_y function (parameters should be listed in the order that σ_y uses them). |

Value

Returns an object of type dynamicsSVM.

Note

Custom functions should have x (the volatility factor) as well as the function's other parameters as arguments.

If the custom function does not use any parameters, one should include an argument called `dummy` and its parameters as a `list(0)`. For example, for the Taylor model,

```
sigma_y_taylor <- function(x, dummy) { return(exp(x / 2)) }
sigma_y_params <- list(0)
```

It should also be noted that the custom function is a vector for x . This means that users should use vectorized version of functions. For example, `pmax(0, x)` instead of `max(0, x)` as code seen in the Example section below.

Examples

```
# Create a dynamicsSVM object with model DuffiePanSingleton and default parameters
DuffiePanSingleton_mod <- dynamicsSVM(model = "DuffiePanSingleton")
```

```
# Here, we define the same DuffiePanSingleton model
# using the custom model option.
```

```
# Daily observations
h <- 1/252
```

```
# Parameter values
mu <- 0.038; kappa <- 3.689; theta <- 0.032
sigma <- 0.446; rho <- -0.745; omega <- 5.125
delta <- 0.03; alpha <- -0.014; rho_z <- -1.809; nu <- 0.004
```

```
# Jump compensator
alpha_bar <- exp(alpha + 0.5 * delta^2) / (1 - rho_z * nu) - 1
```

```
# Returns drift and diffusion
mu_y <- function(x, mu, alpha_bar, omega, h) {
  return(h * (mu - x / 2 - alpha_bar * omega))
}
mu_y_params <- list(mu, alpha_bar, omega, h)
sigma_y <- function(x, h) {
  return(sqrt(h * pmax(x, 0)))
}
sigma_y_params <- list(h)
```

```
# Volatility factor drift and diffusion
mu_x <- function(x, kappa, theta, h) {
  return(x + h * kappa * (theta - pmax(0, x)))
}
mu_x_params <- list(kappa, theta, h)
```

```
sigma_x <- function(x, sigma, h) {
  return(sigma * sqrt(h * pmax(x, 0)))
}
sigma_x_params <- list(sigma, h)
```

```

# Jump distribution for the DuffiePanSingleton Model
jump_density <- dpois
jump_dist <- rpois
jump_params <- c(h * omega)

# Create the custom model
custom_DPS <- dynamicsSVM(model = 'Custom',
  mu_x = mu_x, mu_y = mu_y, sigma_x = sigma_x, sigma_y = sigma_y,
  mu_x_params = mu_x_params, mu_y_params = mu_y_params,
  sigma_x_params = sigma_x_params, sigma_y_params = sigma_y_params,
  jump_params = jump_params, jump_dist = jump_dist, jump_density = jump_density,
  nu = nu, rho_z = rho_z)

```

extractVolPerc.SVDNF *Extract Filtering and Prediction Distribution Percentiles*

Description

Function to extract filtering and prediction distribution percentiles from SVDNF and DNFOptim objects.

Usage

```

## S3 method for class 'SVDNF'
extractVolPerc(x, p = 0.5, pred = F, ...)

```

Arguments

| | |
|------|--|
| x | An SVDNF or DNFOptim object. |
| p | Distribution percentile to return. |
| pred | Return the prediction distribution percentile if pred = TRUE, otherwise, return the filtering distribution percentile. |
| ... | Other parameters to be passed through to function. |

Value

Return a vector of the pth percentile filtering or prediction distribution volatility factor values depending on whether pred = TRUE or not.

Examples

```

set.seed(1)

# Define the built-in model using the dynamicsSVM function
Heston_mod <- dynamicsSVM(model = "Heston")

# Generate the data from the Duffie, Pan, and Singleton model

```

```
Heston_sim <- modelSim(t = 10, dynamics = Heston_mod)
Heston_dnf <- DNF(dynamics = Heston_mod, data = Heston_sim$returns)

extractVolPerc(Heston_dnf, p = 0.75)
```

logLik.SVDNF

Extract Log-Likelihood for SVDNF and DNFOptim Objects

Description

Returns the log-likelihood value of the stochastic volatility model represented by object evaluated at the parameters given in the DNF function.

Usage

```
## S3 method for class 'SVDNF'
logLik(object, ...)
```

Arguments

| | |
|--------|--|
| object | an object of class SVDNF or DNFOptim. |
| ... | further arguments passed to or from other methods. |

Value

The log-likelihood of the stochastic volatility model given by object evaluated at the parameters given to the DNF function. For DNFOptim objects, this returns the log-likelihood at the MLE parameter values and the number of free parameters in the model.

Note

It will always be the case `df = NULL` for SVDNF objects as they are evaluations of the DNF algorithm for a fixed set of parameters. However, for DNFOptim objects, `df` will be the number of free parameters in the optimization.

Examples

```
set.seed(1)
# Generate 200 returns from the DuffiePanSingleton model
DuffiePanSingleton_mod <- dynamicsSVM(model = "DuffiePanSingleton")
DuffiePanSingleton_sim <- modelSim(t = 200, dynamics = DuffiePanSingleton_mod)

# Run DNF on the data
dnf_filter <- DNF(data = DuffiePanSingleton_sim$returns,
  dynamics = DuffiePanSingleton_mod)

# Print log-likelihood evaluation.
logLik(dnf_filter)
```

modelSim.dynamicsSVM *Simulation from Stochastic Volatility Models with Jumps*

Description

The modelSim function generates returns and variances for a wide class of stochastic volatility models.

Usage

```
## S3 method for class 'dynamicsSVM'
modelSim(dynamics, t, init_vol = 0.032, ...)
```

Arguments

| | |
|----------|--|
| dynamics | A dynamicsSVM object representing the model dynamics to be used for simulating data. |
| t | Number of observations to be simulated. |
| init_vol | Initial value of the volatility factor (e.i., value of x_0). |
| ... | Further arguments passed to or from other methods. |

Value

| | |
|-------------------|--|
| volatility_factor | Vector of the instantaneous volatility factor values generated by the modelSim function. |
| returns | Vector of the returns generated by the modelSim function. |

Examples

```
set.seed(1)
# Generate 250 returns from the DuffiePanSingleton model
DuffiePanSingleton_mod <- dynamicsSVM(model = "DuffiePanSingleton")
DuffiePanSingleton_sim <- modelSim(t = 200, dynamics = DuffiePanSingleton_mod)

# Plot the volatility factor and returns that were generated
plot(DuffiePanSingleton_sim$volatility_factor, type = 'l',
     main = 'DuffiePanSingleton Model Simulated Volatility Factor', ylab = 'Volatility Factor')

plot(DuffiePanSingleton_sim$returns, type = 'l',
     main = 'DuffiePanSingleton Model Simulated Returns', ylab = 'Returns')

# Generate 250 steps from a custom model
# Set parameters
kappa <- 100; theta <- 0.05; sigma <- 2.3; h <- 1/252 ; mu <- 0.04
rho <- -0.8; omega <- 5; alpha <- -0.025; nu <- 0.01; rho_z <- -1; delta <- 0.025
# Jump compensator
alpha_bar <- exp(alpha + 0.5 * delta^2) / (1 - rho_z * nu) - 1
```

```

# Define returns drift and diffusion functions
# Returns drift and diffusion
mu_y <- function(x, mu, alpha_bar, omega, h){
  return(h * (mu - x / 2 - alpha_bar * omega))
}
mu_y_params <- list(mu, alpha_bar, omega, h)
sigma_y <- function(x, h, sigma){
  return(sigma * sqrt(h) * pmax(x,0))
}
sigma_y_params <- list(h, sigma)

# Volatility factor drift and diffusion functions
mu_x <- function(x, h, kappa, theta){
  return(x + h * kappa * pmax(0,x) * (theta - pmax(0,x)))
}
mu_x_params <- list(h, kappa, theta)

sigma_x <- function(x, sigma, h){
  return(sigma * sqrt(h) * pmax(0,x))
}
sigma_x_params <- list(sigma, h)

# Include simultaneous return and volatility factor jumps
# based on the Poisson distribution for jump times
jump_dist <- rpois
jump_params <- list(omega * h)
custom_mod <- dynamicsSVM(model = "Custom", mu_x = mu_x, mu_y = mu_y,
  sigma_x = sigma_x, sigma_y = sigma_y,
  mu_x_params = mu_x_params, mu_y_params = mu_y_params,
  sigma_x_params = sigma_x_params, sigma_y_params = sigma_y_params,
  jump_dist = jump_dist, jump_params = jump_params,
  nu = nu, rho_z = rho_z, omega = omega, alpha = alpha, delta = delta)
custom <- modelSim(t = 250, dynamics = custom_mod)

plot(custom$volatility_factor, type = 'l',
  main = 'Custom Model Simulated Volatility Factor', ylab = 'Volatility Factor')
plot(custom$returns, type = 'l',
  main = 'Custom Model Simulated Returns', ylab = 'Returns')

```

pars.dynamicsSVM

Parameters Names and Order for Stochastic Volatility Models with Jumps

Description

The `pars` function returns the names of the parameters in a `dynamicsSVM` object in the order that they should be passed to the `DNFOptim` function.

Usage

```
## S3 method for class 'dynamicsSVM'
pars(dynamics, rho = NULL,
     delta = NULL, alpha = NULL, rho_z = NULL, nu = NULL,
     jump_params_list = "dummy", ...)
```

Arguments

dynamics A dynamicsSVM object representing the model dynamics to be used for which we want to get parameter names.

rho, delta, alpha, rho_z, nu See help(dynamicsSVM) for a description of each of these arguments individually. These arguments should be used only for custom models and can be fixed to a certain value (e.g., $\rho = -0.75$). If they are to be estimated, they should be set to 'var' (e.g., to estimate rho set $\rho = \text{'var'}$) and include it in the vector par to be passed to the DNFOptim function. See Note for more details on the order in which custom models should receive parameters.

jump_params_list List of the names of the arguments in the jump parameter distribution in the order that they are used by the jump_dist function.

... Other parameters to be passed through to function.

Value

Returns a vector of strings with the names of the parameters in the given dynamicsSVM object. The parameters names are returned in the order the parameters should be passed to the DNFOptim function

Examples

```
mod <- dynamicsSVM(model = "Taylor")
pars(mod)
```

plot.predict.DNFOptim *Plot Predictions from DNFOptim or SVDNF Objects*

Description

Plot predictions from a DNFOptim or SVDNF object, including volatility and return mean predictions with confidence intervals.

Usage

```
## S3 method for class 'predict.DNFOptim'  
plot(x, ...)  
  
## S3 method for class 'predict.SVDNF'  
plot(x, ...)
```

Arguments

x an object of class predict.DNFOptim or predict.SVDNF.
... further arguments passed to or from other methods.

Details

This function plots the volatility and return predictions with confidence intervals obtained from a DNFOptim object.

For the volatility plot, it displays the DNF's filtering distribution median volatility for all time points in the series and, after the last observation, plots the predicted mean volatility with its confidence interval.

For the returns plot, it displays the observed returns for all time points in the series and, after the last observation, plots the predicted mean return with its confidence interval.

Value

No return value; this function generates two plots.

The first has the median volatility from the filtering distribution as well as the mean predicted volatility from Monte Carlo simulated paths with its confidence interval.

The second has the observed asset returns as well as the mean predicted returns from Monte Carlo simulated paths with its confidence interval.

Examples

```
# Generating return data  
Taylor_mod <- dynamicsSVM(model = "Taylor", phi = 0.9,  
                          theta = -7.36, sigma = 0.363)  
Taylor_sim <- modelSim(t = 100, dynamics = Taylor_mod, init_vol = -7.36)  
  
# Run the DNF  
DNF_Taylor <- DNF(dynamics = Taylor_mod, data = Taylor_sim$returns)  
  
# Predict the next 10 time steps  
predict_Taylor <- predict(DNF_Taylor, n_ahead = 10)  
  
# Plot the predictions  
plot(predict_Taylor)
```

plot.SVDNF

DNF Filtering Distribution Plot Function

Description

This function plots the median of the filtering and prediction distributions estimated from the DNF along with user-selected upper and lower percentiles.

Usage

```
## S3 method for class 'SVDNF'
plot(x, lower_p = 0.05, upper_p = 0.95, tlim = 'default', type = "l",
     location = 'topright', ...)
```

Arguments

| | |
|----------|---|
| x | An SVDNF object. The plot plots the median and selected percentiles from the filtering distribution. |
| lower_p | Lower percentile of the filtering distribution to plot. |
| upper_p | Upper percentile of the filtering distribution to plot. |
| tlim | The tlim argument gives the range over which to plot the filtering and prediction distributions are displayed. For example to plot the first 500 steps, set tlim = c(1, 500). By default, filtering and prediction distribution estimates for every step in the time-series are generated. If tlim is set to a single number (e.g., tlim = c(5)), plot graphs the estimated probability density functions of the filtering (in magenta) and prediction (in blue) distributions at that timestep. If the data are passed in an xts object, standard date-based subsetting can be used for subsetting (e.g., setting tlim = "2005" to get the filtering and prediction distributions for that year). |
| location | Location keyword passed to the legend function to determine the location of the legend. The keyword should be selected from the list "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", and "center". |
| type | Line type argument passed to the plot function. See help(plot.default) for more details. |
| ... | Other parameters to be passed through to function. |

Value

No return value; this function generates two plots.

The first has the median of the volatility factor obtained from the prediction distribution as well as its upper and lower percentiles from lower_p and upper_p.

The second has the median of the volatility factor obtained from the filtering distribution as well as its upper and lower percentiles from lower_p and upper_p.

Examples

```

set.seed(1)
# Generate 500 returns from the Bates model.
Bates_mod <- dynamicsSVM(model = "Bates")
Bates_sim <- modelSim(t = 500, dynamics = Bates_mod)

# Runs DNF on the data.
dnf_filter <- DNF(data = Bates_sim$returns, dynamics = Bates_mod)

# Plot whole interval (default)
plot(dnf_filter, ylim = c(0, 0.15),
     ylab = "Volatility Factor", xlab = 'Time')

# Plot specific interval
tlim <- c(100,350)
plot(dnf_filter, ylim = c(0, 0.15),
     ylab = "Volatility Factor", xlab = 'Time', tlim = tlim)

# Plot specific point
tlim <- c(100)
plot(dnf_filter, ylim = c(0, 0.15), type = 'l',
     ylab = "Volatility Factor", xlab = 'Time', tlim = tlim)

```

predict.DNFOptim

Predict Method for DNFOptim and SVDNF Objects

Description

This function generates Monte Carlo predictions for DNFOptim objects. The function does this by sampling volatilities from the discrete nonlinear filter's last filtering distribution. Then, using these volatilities as initial values for the modelSim function, the predict method generates n_sim path and estimates the means and confidence intervals for future volatility factor and return values.

Usage

```

## S3 method for class 'DNFOptim'
predict(object, n_ahead = 15, new_data = NULL, n_sim = 1000, confidence = 0.95, ...)

```

Arguments

| | |
|------------|---|
| object | An object of class "DNFOptim". |
| n_ahead | Number of periods ahead to predict. |
| new_data | An optional data frame of new predictor values for models with factors (e.g., the CAPM-SV model) for which we want to forecast returns and volatility values. |
| n_sim | Number of simulated paths used to estimate the future volatility factor and return means and confidence intervals. |
| confidence | Confidence level for prediction intervals. Should be between 0 and 1. |
| ... | Other parameters to be passed through to function. |

Details

This function uses Monte Carlo paths simulated from the MLE dynamics obtained via a `DNFOptim` object to generate predictions for a specified number of periods ahead. It returns predicted mean volatility and return values based on simulations with confidence intervals.

Value

A list containing the following components:

| | |
|------------------------------|--|
| <code>volatility_pred</code> | A list with mean volatility values and confidence intervals. Contains the following components: <code>UB_vol</code> : the upper bound of the confidence interval for volatility, <code>mean_vol_pred</code> : the mean prediction for volatility, and <code>LB_vol</code> : the lower bound of the confidence interval for volatility. |
| <code>ret_pred</code> | A list with mean return values and confidence intervals. Contains the following components: (1) <code>UB_ret</code> : the upper bound of the confidence interval for mean returns, (2) <code>mean_ret_pred</code> : the mean prediction for mean returns, and (3) <code>LB_ret</code> : the lower bound of the confidence interval for mean returns. |
| <code>object</code> | The <code>DNFOptim</code> object input to the <code>predict</code> function. |
| <code>confidence</code> | The specified confidence level. |

See Also

[DNFOptim](#),

Examples

```
set.seed(1)

# Generating return data
Taylor_mod <- dynamicsSVM(model = "Taylor", phi = 0.9,
  theta = -7.36, sigma = 0.363)
Taylor_sim <- modelSim(t = 30, dynamics = Taylor_mod, init_vol = -7.36)

# Initial values and optimization bounds
init_par <- c(0.7, -5, 0.3)
lower <- c(0.01, -20, 0.1); upper <- c(0.99, 0, 1)

# Running DNFOptim to get MLEs
optim_test <- DNFOptim(data = Taylor_sim$returns,
  dynamics = Taylor_mod,
  par = init_par, lower = lower, upper = upper, method = "L-BFGS-B")

# Parameter estimates
summary(optim_test)

# Predict 5 steps ahead
preds <- predict(optim_test, n_ahead = 5)

# Plot predictions with 95 percent confidence interval
plot(preds)
```

| | |
|------------------|---|
| summary.DNFOptim | <i>Summarizing Stochastic Volatility Model Fits from the Discrete Non-linear Filter</i> |
|------------------|---|

Description

Summary method for DNFOptim objects.

Usage

```
## S3 method for class 'DNFOptim'
summary(object, confidence, ...)

## S3 method for class 'summary.DNFOptim'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

| | |
|------------|--|
| object | an object of class DNFOptim that you want to summarize the parameter estimates. |
| x | an object of class summary.DNFOptim. |
| confidence | Confidence level for computing confidence intervals. Should be between 0 and 1. Default is 0.95. |
| digits | The number of digits to be printed in the print method for summary.DNFOptim objects. |
| ... | further arguments passed to or from other methods. |

Details

Returns the summary of the output of a DNFOptim object.

Value

Returns a list with the model used and its estimated parameters.

| | |
|--------------|---|
| model | The model that was estimated with the DNFOptim object. |
| coefficients | Table with the maximum likelihood parameters estimates. If hessian = TRUE was set in the DNFOptim function, standard errors and 95% confidence intervals are given. Then, the table has the following columns: (1) Estimate: the parameter estimate, (2) Std Error: the standard error of the estimate, (3) Lower Bound: the lower bound of the confidence interval, and (4) Upper Bound: The upper bound of the confidence interval. |
| logLik | Log-likelihood value at the parameter maximum likelihood estimates and the model's degrees of freedom |

Examples

```
## For examples see example(DNFOptim)
```

Index

DNF (DNF.dynamicsSVM), 2
DNF.dynamicsSVM, 2
DNFOptim, 18
DNFOptim (DNFOptim.dynamicsSVM), 4
DNFOptim.dynamicsSVM, 4
dynamicsSVM, 7

extractVolPerc (extractVolPerc.SVDNF),
10
extractVolPerc.SVDNF, 10

logLik.DNFOptim (logLik.SVDNF), 11
logLik.SVDNF, 11

modelSim (modelSim.dynamicsSVM), 12
modelSim.dynamicsSVM, 12

pars (pars.dynamicsSVM), 13
pars.dynamicsSVM, 13
plot.DNFOptim (plot.SVDNF), 16
plot.predict.DNFOptim, 14
plot.predict.SVDNF
(plot.predict.DNFOptim), 14
plot.SVDNF, 16
predict.DNFOptim, 17
predict.SVDNF (predict.DNFOptim), 17
print.summary.DNFOptim
(summary.DNFOptim), 19

summary.DNFOptim, 19