Package ‘synthvolforecast’

March 5, 2023

Type Package

Title What the Package Does (Title Case)

Version 0.1.0

Author Who wrote it

Maintainer The package maintainer <yourself@somewhere.net>

Description More about what it does (maybe more than one line)
Use four spaces when indenting paragraphs within the Description.

License What license is it under?

Encoding UTF-8

LazyData true

R topics documented:

`synthvolfit` *A function that furnishes k-step-ahead volatility predictions*

Description

Like I said, a function that furnishes k-step-ahead volatility predictions for a target series.

Usage

```
synthvolfit(Y, X, shocktimevec, shocktimelengths)
```

Arguments

`x` Code will be explained later

Value

k-vector of predictions

Author(s)

David Lundquist

References

references to come

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(X,
                          Y,
                          T_star,
                          shock_est_vec,
                          shock_lengths,
                          garch_param_fit,
                          arch_param_fit,
                          asymmetry_param_fit,
                          normchoice = c('l1','l2')[2],
                          penalty_normchoice = c('l1','l2')[1],
                          penalty_lambda = 0,
                          plots = FALSE)
{ #begin synth_vol_fit

  ## Doc String

  # synth_vol_fit: function that takes (n+1)*(p+1) time series AND a vector of
  # shock times as input and outputs
  # 1) calculates a single weight vector w,
  # 2) calculates a single fixed effects estimate vector omega*,
  # 3) calculates the adjustment estimator vector \hat omega* for time series of interest
  # 4) calculates the volatility of time series of interest at T*+1,T*+2,...,T*+k (i.e. the prediction)
  # 5) calculates estimate of volatility on T*+1 for each series using each of three families, and
  # 6) calculates the squared-error loss of the prediction

  # Estimation/control options
  # --Allow user to enter series of unequal lengths
  # --Allow user to enter a vector of integers corresponding to the number of days
  # the shock effect lasts for each outcome series
  # --Allow user to pick a uniform model for each series (e.g. GARCH(1,1)) OR a BIC-minimizing
  # model for each series (or mix and match).
  # --Allow user to pick error distribution - see ugarchspec

  ##Input
```

```
# Y, a list of length n+1, with each entry containing a time series
# X, a list of length n+1, with each entry containing a dataframe of dimension y_i x p
# shock_time_vec, a vector of length n+1 containing shock time of each series
# shock_time_lengths, a vector of length n+1 containing shock time length of each series

#First, we get the vectors w for all sensible methods
w <- list() #initialize
matrix_of_specs <- matrix(c(rep(1,6),
                            rep(NA,6),

                            rep(c(0,-1,NA),4),

                            rep(c(1,NA), 6)),
                          byrow = FALSE, nrow = 12)

#We drop the 4th row because it's functionally no different from the first OR have lower bound > upper bound
matrix_of_specs <- matrix_of_specs[-4,]

for (i in 1:nrow(matrix_of_specs)){
  w[[i]] <- dbw(X,
                T_star,
                scale = TRUE,
                sum_to_1 = matrix_of_specs[i,1],
                bounded_below_by = matrix_of_specs[i,2],
                bounded_above_by = matrix_of_specs[i,3],
                normchoice = normchoice,
                penalty_normchoice = penalty_normchoice,
                penalty_lambda = penalty_lambda)
}

# Now we place these linear combinations into a matrix
w_mat <- matrix(unlist(w), nrow = nrow(matrix_of_specs), byrow = TRUE)

# Add a linear combination that is just 1/n times that 1 vector
linear_comb_for_arithmetic_mean <- rep(1/ncol(w_mat), ncol(w_mat))
w_mat <- rbind(w_mat, linear_comb_for_arithmetic_mean)

# Now get a linear combination that is the beta_hat we get from
# regressing the estimation shocks against the X matrix
T_star_cov_df <- data.frame()
# covariates for time series pool
T_star_cov_df <- list()
for (i in 1: (length(shock_est_vec) - 1)) {
 T_star_cov_df[[i]] <- X[[i + 1]][T_star[i + 1] , , drop = FALSE] #get 1 row from each donor
}

T_star_cov_df <- matrix( unlist(T_star_cov_df), ncol = length(shock_est_vec) - 1, byrow = FALSE)
linmod <- lm(shock_est_vec[-1] ~ ., data = as.data.frame(t(T_star_cov_df)))
linear_reg_pred <- as.numeric( predict(linmod, newdata = as.data.frame( X[[ 1]][T_star[1], , drop = FALSE] )))

#Second, we calculate omega_star_hat, which is the dot product of w and the estimated shock effects
omega_star_hat_vec <- as.numeric(w_mat
omega_star_hat_vec <- c(omega_star_hat_vec, linear_reg_pred)
```

```
#Third, we get a prediction to T*_+k
y_up_through_T_star <- Y[[1]][,1][1:T_star[1],1]
sigma2_up_through_T_star <- Y[[1]][,3][1:T_star[1],1]
y_up_through_T_star_plus_k <- Y[[1]][,1][1:(T_star[1] + shock_lengths[1]),1]
sigma2_up_through_T_star_plus_k <- Y[[1]][,3][1:(T_star[1] + shock_lengths[1]),1]
sigma2_shock_period_only <- Y[[1]][,3][(T_star[1] + 1):(T_star[1] + shock_lengths[1]),1]

garch_1_1 <- garchx(y_up_through_T_star,
                    order = c(garch_param_fit, arch_param_fit, asymmetry_param_fit),
                    solve.tol = .0000000001,
                    control = list(eval.max = 10000000, iter.max = 10000000),
                    hessian.control = list(maxit = 10000000) )

pred <- as.numeric(predict(garch_1_1, n.ahead = shock_lengths[1]))

adjusted_pred_list <- list() # the ith entry will be using the ith linear combination
MSE_adjusted <- list()
APE_adjusted <- list()
QL_adjusted <- list()
DM_test <- list()

for (i in 1:length(omega_star_hat_vec)) #tk use lapply?
{
  adjusted_pred <- pred + omega_star_hat_vec[i]
  adjusted_pred_list[[i]] <- pmax(adjusted_pred, 0)
  MSE_adjusted[[i]] <- mean((sigma2_shock_period_only - adjusted_pred_list[[i]])**2)
 APE_adjusted[[i]] <- mean(abs(sigma2_shock_period_only - adjusted_pred_list[[i]]) / sigma2_shock_period_only)
 QL_adjusted[[i]] <- mean( sigma2_shock_period_only / adjusted_pred_list[[i]] - log(sigma2_shock_period_only/ad

  # DM_test[[i]] <- dm.test(sigma2_shock_period_only - adjusted_pred_list[[i]],
  #                         sigma2_shock_period_only - pred,
  #                         alternative = c("two.sided", "less", "greater")[1],
  #                         h = 2, #tk
  #                         power = 2)

  #print(DM_test[[i]]) tk

}

#Last, we calculate unadjusted MSE and APE
MSE_unadjusted <- round(mean((sigma2_shock_period_only - pred)**2), 5)
MAPE_unadjusted <-  round(mean(abs(sigma2_shock_period_only - pred)/sigma2_shock_period_only), 5)
QL_unadjusted <-  round(mean( sigma2_shock_period_only / pred - log(sigma2_shock_period_only/pred ) - 1), 5)

#We now make a vector with the names of each of the sensible linear combinations
linear_comb_names <- c('Convex Hull',
                       '1 -1 NA',
                       'Drop Bounded Below',
                       'Unit Ball: Sum-to-1',
                       'Affine Hull',
                       'Drop Sum-to-1',
                       'Bounded Below by -1',
```

```
                         'Bounded Above by 1',
                         'Conic Hull',
                         'Unit Ball',
                         'Unrestricted',
                         'Arithmetic Mean',
                         'Linear Regression')

labels_for_legend <- c('Actual','GARCH (unadjusted)', linear_comb_names)

if (plots == TRUE){

  #Plot the donor pool weights
 par(mfrow=c(floor(sqrt(length(linear_comb_names))),ceiling(sqrt(length(linear_comb_names)))))
  for (i in 1:nrow(w_mat))
  {
    minn <- min(w_mat[i,])
    maxx <- max(w_mat[i,])
    if (minn == maxx)
    {
      minn <- -1 * abs(minn)
      maxx  <- 1 * abs(maxx)
    }
    barplot(w_mat[i,],
            main = paste('Donor Pool Weights:\n',
                         linear_comb_names[i]),
            names.arg = 2:(length(T_star)),
            ylim = c(minn, maxx))
  }

  #Now let's plot the adjustment
  par(mfrow=c(1,2))

 trimmed_prediction_vec_for_plotting <- Winsorize(unlist(adjusted_pred_list), probs = c(0, 0.72))

  #PLOT ON THE LEFT:
  plot(sigma2_up_through_T_star_plus_k,
       main = 'GARCH Prediction versus \nAdjusted Predictions versus Actual',
       ylab = '',
       xlab = "Time",
       xlim = c(0, length(sigma2_up_through_T_star_plus_k) + 5),
    ylim = c(0,  max(pred, trimmed_prediction_vec_for_plotting, sigma2_up_through_T_star_plus_k) ) )

  title(ylab = expression(sigma^2), line = 2.05, cex.lab = 1.99) # Add y-axis text

  # We also plot, in a different line style, the post-shock period
  lines(y = c(sigma2_up_through_T_star[T_star[1]],  sigma2_shock_period_only),
        x = T_star[1]:(T_star[1]+shock_lengths[1]),  lty=2, lwd=2,
     ylim = c(0,  max(pred, trimmed_prediction_vec_for_plotting, sigma2_up_through_T_star_plus_k) ) )

  # Here is the color scheme we will use
  colors_for_adjusted_pred <- c('black', 'red', "green",
                                 brewer.pal(length(linear_comb_names) - 1 ,'Set3'))
```

```
  # Let's add the ground truth
  points(y = sigma2_shock_period_only,
         x = (T_star[1]+1):(T_star[1]+shock_lengths[1]),
         col = colors_for_adjusted_pred[1],
         cex = 1.3, pch = 16)

  # Let's add the plain old GARCH prediction
  points(y = pred,
         x = (T_star[1]+1):(T_star[1]+shock_lengths[1]),
         col = colors_for_adjusted_pred[2],
         cex = 1.3, pch = 15)

  # Now plot the adjusted predictions
  for (i in 1:(length(omega_star_hat_vec)))
  {
   points(y = adjusted_pred_list[[i]], x = (T_star[1]+1):(T_star[1]+shock_lengths[1]),
          col = colors_for_adjusted_pred[i+2], cex = 1.9, pch = 10)
  }

  legend(x = "topleft",  # Coordinates (x also accepts keywords)
         legend = labels_for_legend,
         1:length(labels_for_legend), # Vector with the name of each group
      colors_for_adjusted_pred,  # Creates boxes in the legend with the specified colors
         title = 'Prediction Method',      # Legend title,
         cex = .9
  )

  #PLOT ON THE RIGHT
  plot.ts(fitted(garch_1_1),
          main = 'Pre-shock GARCH fitted values (green) \nversus Actual (black)',
          ylab = '', col = 'green',
          ylim = c(0, max(fitted(garch_1_1), sigma2_up_through_T_star)) ,
          cex.lab = 3.99)
  lines(sigma2_up_through_T_star, col = 'black')

  title(ylab = expression(sigma^2), line = 2.05, cex.lab = 1.99)

 } #end conditional for plots

 #Now arrange the output
 display_df <- data.frame(linear_comb_names)
 display_df$w_star_hat <- round(unlist(omega_star_hat_vec), 5)
 display_df$MSE_adj <- round(unlist(MSE_adjusted), 5)
 display_df$MAPE_adj <- round(unlist(APE_adjusted), 5)
 display_df$QL_adj <- round(unlist(QL_adjusted), 5)

 #Now add the unadjusted row
unadjusted_row <- c('GARCH (unadjusted)', 0, MSE_unadjusted, MAPE_unadjusted, QL_unadjusted)
 display_df <- rbind(display_df, unadjusted_row)

 # display_df$beat_unadjusted <- as.integer(display_df$QL_adj < QL_unadjusted)
# ORDERED_display_df <- display_df[order(display_df$QL_adj, na.last = TRUE, decreasing = FALSE), ]
```

```
  cat('\n Dataframe Comparing the Distance-based-weighting methods \n')
  cat('------------------------------------------------------------ \n')
  print(display_df)

  return(as.vector(display_df))

} #end of synth_vol_fit
```

Index