tspred.R

georges

2025-03-02

```
#
#  Prediction for time series models
#
#  The prediction function is called 'tspred'
#
DO_TESTS <- TRUE

library(latticeExtra)
```

```
## Loading required package: lattice
```

```
Layer <- latticeExtra::layer # to avoid conflict with tidyverse
library(magrittr)
library(cv)
```

```
## Loading required package: doParallel
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

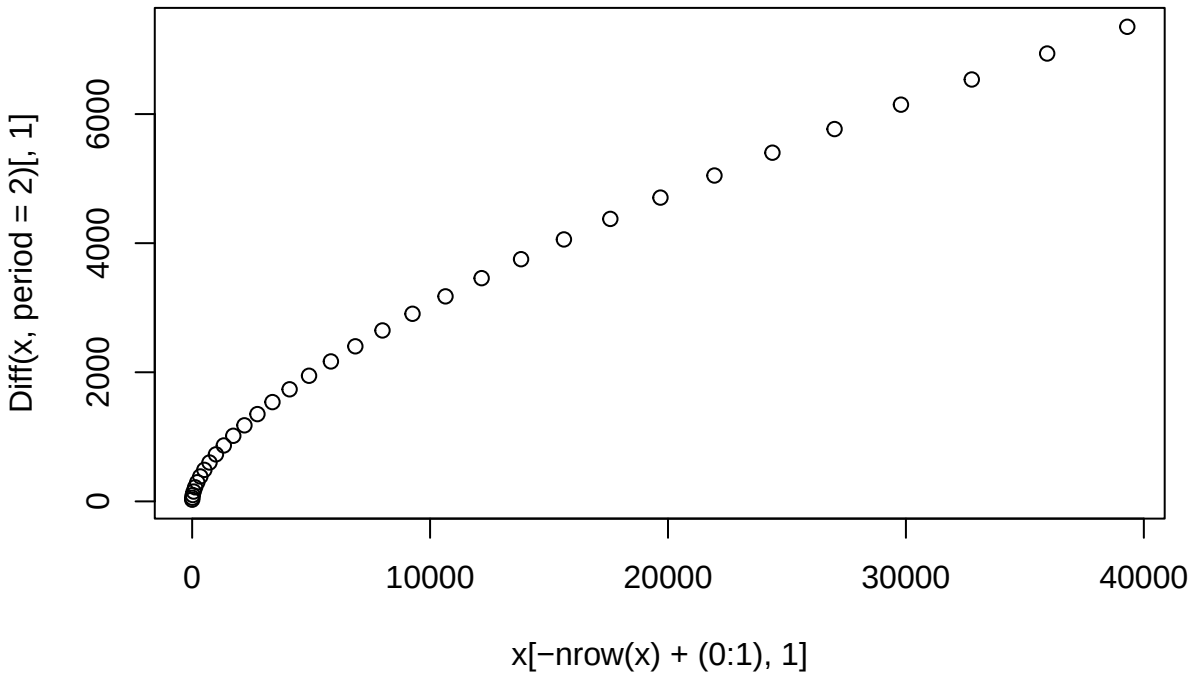
```
## Loading required package: parallel
```

```
Diff <- function(
    x,                            # a vector or matrix
    order = 1 * (seasonal == 0),  # order
    seasonal = 1 * (period != 1), # seasonal order
    period = 1) {                 # period

  # Returns
  #
  # Differenced vector or matrix with attributes:
  #   - x: the input (including its attributes if input was a differenced object)
  #   - period: the period for the last differencing (1, or >1 if seasonal)
  #   - periods: vector with cumulative periods differenced
  #   - offset: sum of periods
  #   - the cummulative periods differenced , recursively,
  #

  if(seasonal > 0) {                  # seasonal order first
    for(i in 1:seasonal){
      x <- diffts(x, order = 1, period = period)
    }
  }
```

```
  if(order > 0) {
    for(i in 1:order){
      x <- diffts(x, order = 1, period = 1)
    }
  }
  x
}

diffts <- function(x, order, period) {
  #
  # works on matrices and vectors
  #
  # - returns a result that is 'order * period' shorter
  #   or, in the case of matrix, has 'order * period' fewer rows
  #   than the input.
  #

  if(order > 1) return(
    diffts(
      diffts(x, order = order - 1, period = period),
      order = 1,
      period = period)
  )
  if(order == 0) return(x)

  #
  # if order == 1:
  #

  atts <- list(
    x = x,
    period = period,
    periods = c(attr(x,'periods'), period),
    offset = if(is.null(attr(x,'offset'))){
      period
    } else {
      period + attr(x,'offset')
    })

  r <- diff(x, lag = period)
  for(i in seq_along(atts)) attr(r, names(atts[i])) <- atts[[i]]
  r
}


rediffts <- function(x, like) {
  #
  # apply differencing to x like differencing performed in 'like'
  #
  periods <- attr(like, 'periods')

  ret <- x
```

```
  for(p in periods){
    ret <- diffts(ret, order = 1, period = p)
  }
  ret
}

if(DO_TESTS){   # Test Diff, diffts

  #
  # if x is a matrix:
  #
  x <- cbind((1:36)^3, 1)
  colnames(x) <- c('one','two')
  rownames(x) <- 1:36

  dim(x)
  Diff(x)
  Diff(x) %>% dim
  Diff(x, order = 2)
  Diff(x, order = 2) %>% dim
  Diff(x, order = 3)
  Diff(x, order = 4)
  Diff(x, order = 4) %>% dim

  Diff(x, period = 12) %>% dim
  plot(x[-nrow(x)+(0:1),1], Diff(x, period = 2)[,1])
  Diff(x, period = 12) %>% class
  Diff(x, period = 12)
  #
  # if x is a vector
  x[-length(x)+(0:14),1] %>% dim
  #
  x <- (1:36)^3
  names(x) <- paste0('x',1:36)
  Diff(x, period = 12)
  Diff(x, period = 12) %>% Diff(period = 3) %>% Diff()
  #
  #
  xx <- Diff(x, order = 2, seasonal = 2, period = 12)
  str(xx)
  Diff(x, seasonal = 3, period = 12) %>% attributes
  # Diff(x, seasonal = 4, period = 12)

  x <- (1:24)^3
  Diff(x) %>% attributes
  diffts(x, order = 1, period = 1) %>% attributes


  Diff(Diff(x))
  Diff(Diff(Diff(x)))
  xx <- Diff(Diff(Diff(x)))
```

```
  xx2 <- Diff(x, order = 3)

  all.equal(xx, xx2)

  # Seasonal

  Diff(x, period = 3)

  x1 <- Diff(x, seasonal = 2, period = 3)
  x2 <- Diff(Diff(x, period = 3), period = 3)

  all.equal(x1, x2)

  #
  # Does 'rediffts' do the right thing?
  #
  xm <- cbind(cube =(1:36)^3, square = (1:36)^2)

  xmres <- Diff(xm, order = 2, seasonal = 1, period = 12)

  xmres2 <- rediffts(xm, like = xmres)
  all.equal(xmres, xmres2)


  #
  # Periods must divide into previous periods
  #

  Diff(x, period = 12)
  Diff(x, period = 12) %>% Diff(period = 3) # monthly and quarterly

}
```



```
##  Named num [1:10] 0 0 0 0 0 0 0 0 0 0
##  - attr(*, "names")= chr [1:10] "x27" "x28" "x29" "x30" ...
##  - attr(*, "x")= Named num [1:11] 864 864 864 864 864 864 864 864 864 864 ...
##   ..- attr(*, "names")= chr [1:11] "x26" "x27" "x28" "x29" ...
##   ..- attr(*, "x")= Named num [1:12] 11232 12096 12960 13824 14688 ...
##   .. ..- attr(*, "names")= chr [1:12] "x25" "x26" "x27" "x28" ...
##   .. ..- attr(*, "x")= Named num [1:24] 2196 2736 3348 4032 4788 ...
##   .. .. ..- attr(*, "names")= chr [1:24] "x13" "x14" "x15" "x16" ...
##   .. .. ..- attr(*, "x")= Named num [1:36] 1 8 27 64 125 216 343 512 729 1000 ...
##   .. .. .. ..- attr(*, "names")= chr [1:36] "x1" "x2" "x3" "x4" ...
##   .. .. ..- attr(*, "period")= num 12
##   .. .. ..- attr(*, "periods")= num 12
##   .. .. ..- attr(*, "offset")= num 12
##   .. ..- attr(*, "period")= num 12
##   .. ..- attr(*, "periods")= num [1:2] 12 12
##   .. ..- attr(*, "offset")= num 24
##   ..- attr(*, "period")= num 1
##   ..- attr(*, "periods")= num [1:3] 12 12 1
##   ..- attr(*, "offset")= num 25
##  - attr(*, "period")= num 1
##  - attr(*, "periods")= num [1:4] 12 12 1 1
##  - attr(*, "offset")= num 26
```

```
## [1] 1836 2052 2268 2484 2700 2916 3132 3348 3564
## attr(,"x")
##  [1]  2196  2736  3348  4032  4788  5616  6516  7488  8532  9648 10836 12096
## attr(,"x")attr(,"x")
##  [1]     1     8    27    64   125   216   343   512   729  1000  1331  1728
## [13]  2197  2744  3375  4096  4913  5832  6859  8000  9261 10648 12167 13824
## attr(,"x")attr(,"period")
## [1] 12
## attr(,"x")attr(,"periods")
## [1] 12
## attr(,"x")attr(,"offset")
## [1] 12
## attr(,"period")
## [1] 3
## attr(,"periods")
## [1] 12  3
## attr(,"offset")
## [1] 15
```

```
#
# Antiderivative
#

Diffinv <- function(x, all = TRUE, at = NA, val = NA, ...){
  # idea:
  # Take a differenced object and
  #
  # - just antidifference 'into' the original object by using
  #   the corresponding values derived from the original object
  #   when antidifferencing.  This has the result of returning
  #   the original object.
  #
```

```
  # - antidifference into a different object as one would
  #   when using a model to predict with a different
  #   trigger series
  #
  if(is.na(at)) {
    if(!all) {
      diffinvts(x)  # antidifference one step
    } else {
      depth <- length(attr(x, 'periods'))
      ret <- x
      for(i in 1:depth) {
        ret <- diffinvts(ret)
      }
    }
  }
  ret
}


diffinvts <- function(x, at = 1, value = NA, period = attr(x, 'period')) {
  #
  # period:  only used if antidifferencing a raw vector
  #          i.e. a vector not created by Diff
  #          For vectors created by Diff, the period attribute of x is used

  if(is.null(period)) period <- 1

  ismat <- is.matrix(x)
  xx <- as.matrix(x)

  if(is.null(xp <- attr(x,'x'))) {
    xp <- matrix(0,ncol = ncol(xx), nrow = 0)
  } else {
    xp <- as.matrix(xp)
  }

  if(nrow(xp) < period){
    r <- rbind(
      matrix(0, ncol = ncol(xx), nrow = period - nrow(xp)),
      xp,
      xx)
  } else {
    r <- rbind(xp[seq_len(period),,drop = FALSE], xx)
  }

  r <- diffinv(r, lag = period)
  r <- r[-seq_len(period),,drop = FALSE]

  if(!ismat) r <- drop(r)

  attr(r,'x') <- attr(attr(x, 'x'), 'x')
  attr(r,'period') <- attr(attr(x, 'x'), 'period')
  # if(is.null(attr(r, 'period'))) attr(r, 'period') <- period
```

```
  attr(r,'periods') <- attr(attr(x,'x'), 'periods')

  r
}

if(DO_TESTS){  # Tests

  x <- (1:12)^3
  xx <- Diff(x, period = 3)
  Diffinv(xx)
  xxx <- Diff(xx, period = 2)

  length(xxx)
  xxxi <- diffinvts(xxx)


  xxxi == xx

  all.equal(xxxi, xx)

  xx2 <- Diff((1:12)^3, order = 2, period = 3)

  xx2 %>% Diffinv
  xxq <- xx2 %>% diffinvts %>% diffinvts
  all.equal(xxq, xx)
  xxq == xx

  xx2 %>% diffinvts %>% diffinvts %>% diffinvts # FIXED: had period and no periods attr
  #
}
```

```
##  [1]    1    8   27   64  125  216  343  512  729 1000 1331 1728
```

```
#
# Modelling and predicting
#
# The goal is to be able to apply an arima model
# to predict with new data
#
# To do this we need to:
#
# - difference new Y and new xreg
# - difference leadup Y and leadup xreg to get ARMA residuals
# - Use model on ARMA residuals to get ARMA predicted residuals
# - Since the residuals are ARMA (in contrast with whitened residuals)
#   we need to apply 'pi' weights for prediction.
# - Add predicted value from the regression model
# - Antidifference to get predicted value
#

arma2psi <- function(ar=0, ma=0, ar.seasonal=0, ma.seasonal=0,
                     period, lag.max=100, trunc.psi=TRUE,
                     trunc.at=0.001, ...){
  #
```

```
  # Copied from scratch/notes_john/arma2psi.R
  #
  lag.max.tot <- if (!(missing(period) || is.na(period))) lag.max*period else lag.max
  psi <- ARMAtoMA(ar = ar, ma = ma, lag.max=lag.max.tot)
  if (!(missing(period) || is.na(period))) {
    psi.seasonal <- ARMAtoMA(ar = ar.seasonal, ma = ma.seasonal, lag.max=lag.max)
    psi <- psi + as.vector(rbind(matrix(0, period - 1, lag.max),
                                 psi.seasonal))
  }
  if (trunc.psi){
    which.psi <- which(abs(psi) >= trunc.at)
    if (length(which.psi) == 0) {
      return(numeric(0))
    }
    if (max(which.psi) == lag.max.tot) {
      warning("all ", lag.max.tot, " psi weights retained")
    } else {
      psi <- psi[1:max(which.psi)]
    }
  }
  psi
}

arma2pi <- function(ar=0, ma=0, ar.seasonal=0, ma.seasonal=0,
                    period, lag.max=100, trunc.pi=TRUE,
                    trunc.at=0.001, ...){
  #
  # Returns 'pi' weights to predict Y(t+h) recursively from Y(t-k),...,Y(t)
  # where Ys are ARMA(ar,ma)
  #
  # Dual to psi:
  #
  # use -ARMAtoMA(ar = -ma, ma = -ar)  with B&D signs for ma parameters
  #
  #

  lag.max.tot <- if (!(missing(period) || is.na(period))) lag.max*period else lag.max
  Pi <- - ARMAtoMA(ar = -ma, ma = -ar, lag.max=lag.max.tot)
  if (!(missing(period) || is.na(period))) {
    Pi.seasonal <- - ARMAtoMA(ar = - ma.seasonal, ma = - ar.seasonal, lag.max=lag.max)
    Pi <- Pi + as.vector(rbind(matrix(0, period - 1, lag.max),
                               Pi.seasonal))
  }
  if (trunc.pi){
    which.pi <- which(abs(Pi) >= trunc.at)
    if (length(which.pi) == 0) {
      return(numeric(0))
    }
    if (max(which.pi) == lag.max.tot) {
      warning("all ", lag.max.tot, " pi weights retained")
    } else {
      Pi <- Pi[1:max(which.pi)]
    }
```

```
  }
  Pi
}

if(DO_TESTS) {  # test inversion
  arma2pi(ma = arma2psi(ar=c(.2,.2)))
  arma2psi(ar = arma2pi(ma=c(-.2,-.2)))

  arma2pi(ma = arma2psi(ar=c(.45,.45)))
  arma2psi(ar = arma2pi(ma=c(-.45,-.45)))

  arma2pi(ma = arma2psi(ar=c(.499,.499)))  # nearly non-stationary
  arma2psi(ar = arma2pi(ma=c(-.499,-.499)))  # nearly non-invertible

}
```

```
## Warning in arma2psi(ar = c(0.499, 0.499)): all 100 psi weights retained
```

```
## Warning in arma2pi(ma = c(-0.499, -0.499)): all 100 pi weights retained
```

```
## [1] -0.499 -0.499
```

```
tspred <- function(model, newdata, refit = FALSE, demean = FALSE) {
  #
  # Rough version function to see if the idea works
  #
  # Prediction with new predictors, Xs, and a sequence of
  # 'lead-up' response values, Ys, preceding the predicted values
  # requires predictor values for the lead-up sequence as
  # well as for the values to be predicted.
  #
  # 'newdata' is a data.frame similar to the data frame
  # used to fit the 'model'.  It consists of
  # n.leadup rows with values for Xs and Y
  # followed by n.ahead rows with values for Xs
  # and NA's for Ys.
  #
  # n.leadup must be long enough to allow ordinary
  # and seasonal differencing to the order in 'model'. (revisit this)
  #
  # The function computes predicted values for the Ys
  # that are missing by:
  #
  # 1. Getting the model matrix and the response
  #    vector corresponding to 'newdata' using
  #    the model.
  # 2. Obtaining the residual by subtracting the
  #    predicted value.
  # 2. Differencing the residual using the 'I' orders
  #    in the ARIMA model
  # 3. Using the ARMA model to predict ARMA residuals.
  # 4. Integrating the ARMA sequence to get predicted
  #    ARIMA redisuals.
  # 5. Adding the values predicted from the predictor model.
  # The function returns the vector of predicted
```

```
# values and the new data frame with leadup and
# predicted values.
#
# Note that, for cross-validation, this approach can use
# a model that omits an internal fold but,
# with differencing, the 'leadup' data needs to
# precede the predicted values.  Without differencing
# the values to be predicted need not be at the
# end of 'newdata'.
#

# Get y and xreg matrix from new Ys and xreg

y_name <- as.character(model$call$formula[[2]])
y_new <- newdata[[y_name]]
to_pred <- is.na(y_new)

#
# FIX THIS TO AVOID REFITTING THE MODEL
# The problem here is getting a model matrix
# that omits the intercept as 'model.matrix'
# applied to an ARIMA object does depending
# on other arguments to 'Arima'.
#
model_new <- update(model, data = newdata)
xreg_new <- model.matrix(model_new)

diff_order <- model$order[2]
seasonal_diff_order <- model$seasonal$order[2]

##FIXME: following should use truncation of leadup data with a warning
##       and extension of prediction xreg followed by truncation

if(seasonal_diff_order > 0) {
  if((sum(!is.na(y_new)) %% seasonal_diff_order) != 0 ){
    stop('Number of lead-up data rows should be a multiple of seasonal differencing order (',
         seasonal_diff_order,')')
  }
  if((sum(is.na(y_new)) %% seasonal_diff_order) != 0 ) {
    stop('Number of predicted data rows should be a multiple of seasonal differencing order (',
         seasonal_diff_order,')')
  }
}

# Parse model coefficients

coefs <- coef(model)

if ("(Intercept)" %in% names(coefs)){   # from cv::Predict.ARIMA
  xreg_new <- if (!is.null(xreg_new)){
    cbind(1, xreg_new)
  } else {
    matrix(1, nrow=length(y_new), ncol=1)
```

```
  }
}
ar <- coefs[grepl('^ar[0-9]+$',names(coefs))]
ma <- coefs[grepl('^ma[0-9]+$',names(coefs))]
sar <- coefs[grepl('^sar[0-9]+$',names(coefs))]
sma <-  coefs[grepl('^sma[0-9]+$',names(coefs))]
beta <- coefs[-seq_len(length(ar) + length(ma) + length(sar) + length(sma))]

# Get regression residuals

y_new_res <- y_new - if(!is.null(beta)) xreg_new %*% beta else 0    # ARIMA residuals

y_new_diff <- y_new_res

if(model$seasonal$order[2] > 0) {
  stopifnot(length(y_new) %% model$seasonal$period == 0)
  stopifnot(sum(is.na(y_new)) %% model$seasonal$period == 0)
  y_new_diff <- Diff(y_new_diff,
                     seasonal = model$seasonal$order[2],
                     period = model$seasonal$period)

}
if(model$order[2] > 0) {
  y_new_diff <- Diff(y_new_diff,
                     order = model$order[2])
}

# Pi weights

Pi <- arma2pi(ar = ar, ma = ma, ar.seasonal = sar, ma.seasonal = sma)

convolve <- function(y, w) {
  yrev <- rev(y)
  len <- min(length(y),length(w))
  sum(yrev[1:len]*w[1:len])
}

to_pred_diff <- which(is.na(y_new_diff))

for(i in to_pred_diff) y_new_diff[i] <- convolve(y_new_diff[1:(i-1)], Pi)

if(diff_order > 0 || seasonal_diff_order > 0){
  y_pred <- Diffinv(y_new_diff)
} else {
  y_pred <- y_new_diff
}

new_pred <- y_pred + if(!is.null(beta)) xreg_new %*% beta else 0

newdata[[y_name]] <- new_pred
newdata$.predicted <- to_pred
attr(newdata,'Pi') <- Pi
attr(newdata,'Mod') <- Mod(polyroot(c(1, -Pi)))
```

```
  newdata
}


if(DO_TESTS){   # test tspred

  #
  # sample model data
  #

  {
    # Generate sample data using (1,1,1) model
    set.seed(321)

    N <- 9999
    eps <- arima.sim(list(ar = .9, ma = .8), n = N)
    x <- rnorm(N)
    dd <- (data.frame(y = cumsum(eps + x), x = x))
    dd$time <- seq_len(nrow(dd))
  }

  system.time({
    models <- within(
      list(),
      {
        `101/111(3)` <- cv::Arima(y ~ 1 +x, data = dd, order = c(1,0,1), seasonal = list(order = c(1,1,1
        `111/111(3)` <- cv::Arima(y ~ x, data = dd, order = c(1,1,1), seasonal = list(order = c(1,1,1),
        `101` <- cv::Arima(y ~ x, data = dd, order = c(1,0,1))
        `111` <- cv::Arima(y ~ x, data = dd, order = c(1,1,1))
      }
    ) %>% rev
  })


  # Predicting last 9 observations with different models fit to all data

  dd_pred <- dd
  dd_pred$y[nrow(dd)-(9:1) +1] <- NA

  system.time(
    # predicting from 4 models
    preds <- lapply(models, function(mod) tspred(mod, dd_pred) )
  )

  tail(dd_pred, 15)
  tail(preds[[3]],15)

  # Plot last 31 observations

  end <- nrow(dd) - (30:0)
  ylim <- range(c(dd[end,1],sapply(preds, \(pred) pred[pred$.predicted,1])))

  cols <- trellis.par.get('superpose.line')$col
```

```
xyplot(y ~ time , dd[end,],
       ylim = ylim,
       type = 'b') %>% print

{
  xyplot(y ~ time , dd[end,], type = 'b', lwd = 4,
         ylim = ylim,
         key = simpleKey(
           cex = 1,
           space = 'right',
           lines = TRUE,
           col = cols,
           text = c('101/111', '111/111','101','111')
         ),
         sub = 'Predicting last 9 from whole series using all data for leadup. Correct model: 111'
  ) +
    xyplot(y ~ time, subset(preds[[1]], .predicted), col = 'red', type = 'b') +
    xyplot(y ~ time, subset(preds[[2]], .predicted), col = 'black', type = 'b') +
    xyplot(y ~ time, subset(preds[[3]], .predicted), col = 'green', type = 'b') +
    xyplot(y ~ time, subset(preds[[4]], .predicted), col = 'magenta', type = 'b')
} %>% print


#
# Show preds as function
#


Show_pred <- function(models, data, pred, last = nrow(data), main = '', sub = ''){
  # FIX:
  # assumes dependent var is named 'y'
  # assumes 4 models

  system.time(
    preds <- lapply(models, function(mod) tspred(mod, pred) )
  )

  # Plot last observations

  toplot <- nrow(data) - last:1 + 1

  ylim <- range(c(data[toplot,'y'],sapply(preds, \(pred) pred[pred$.predicted,'y'])))

  # cols <- trellis.par.get('superpose.line')$col
  cols = c('black','red','green','blue')
  trellis.par.set('superpose.line', list(col = cols))

  {
    xyplot(y ~ time , dd[toplot,], type = 'b', lwd = 4,
           ylim = ylim,
           key = simpleKey(
             cex = 1,
             space = 'right',
```

```
              lines = TRUE,
              col = cols,
              text = names(preds)
            ),
            #par.settings = list(superpose.line=list(col = cols)),
            main = main,
            sub = sub
      ) +  # FIXME for varying number of models
        xyplot(y ~ time, subset(preds[[1]], .predicted), col = cols[1], type = 'b') +
        xyplot(y ~ time, subset(preds[[2]], .predicted), col = cols[2], type = 'b') +
        xyplot(y ~ time, subset(preds[[3]], .predicted), col = cols[3], type = 'b') +
        xyplot(y ~ time, subset(preds[[4]], .predicted), col = cols[4], type = 'b')
    }
  }

  dd_pred <- dd
  dd_pred$y[nrow(dd_pred)-(9:1) +1] <- NA
  main = 'Predicting last 9 from whole series using all data for leadup'
  sub = 'Correct model: 111'

  Show_pred(models, dd, dd_pred, last = nrow(dd), main = main, sub = sub) %>% print
  Show_pred(models, dd, dd_pred, last = 100, main = main, sub = sub) %>% print
  Show_pred(models, dd, dd_pred, last = 30, main = main, sub = sub) %>% print

  dd_pred <- tail(dd, 30)
  dd_pred$y[nrow(dd_pred)-(9:1) +1] <- NA
  main = 'Predicting last 9 from whole series using previous 21 for leadup'
  sub = 'Correct model: 111'

  Show_pred(models, dd, dd_pred, last = nrow(dd), main = main, sub = sub) %>% print
  Show_pred(models, dd, dd_pred, last = 100, main = main, sub = sub) %>% print
  Show_pred(models, dd, dd_pred, last = 50, main = main, sub = sub) %>% print

  dd_pred <- tail(dd, 99)
  dd_pred$y[nrow(dd_pred)-(9:1) +1] <- NA
  main = 'Predicting last 9 from whole series using previous 90 for leadup'
  sub = 'Correct model: 111'

  Show_pred(models, dd, dd_pred, last = nrow(dd), main = main, sub = sub) %>% print
  Show_pred(models, dd, dd_pred, last = 100, main = main, sub = sub) %>% print
  Show_pred(models, dd, dd_pred, last = 50, main = main, sub = sub) %>% print

}
```

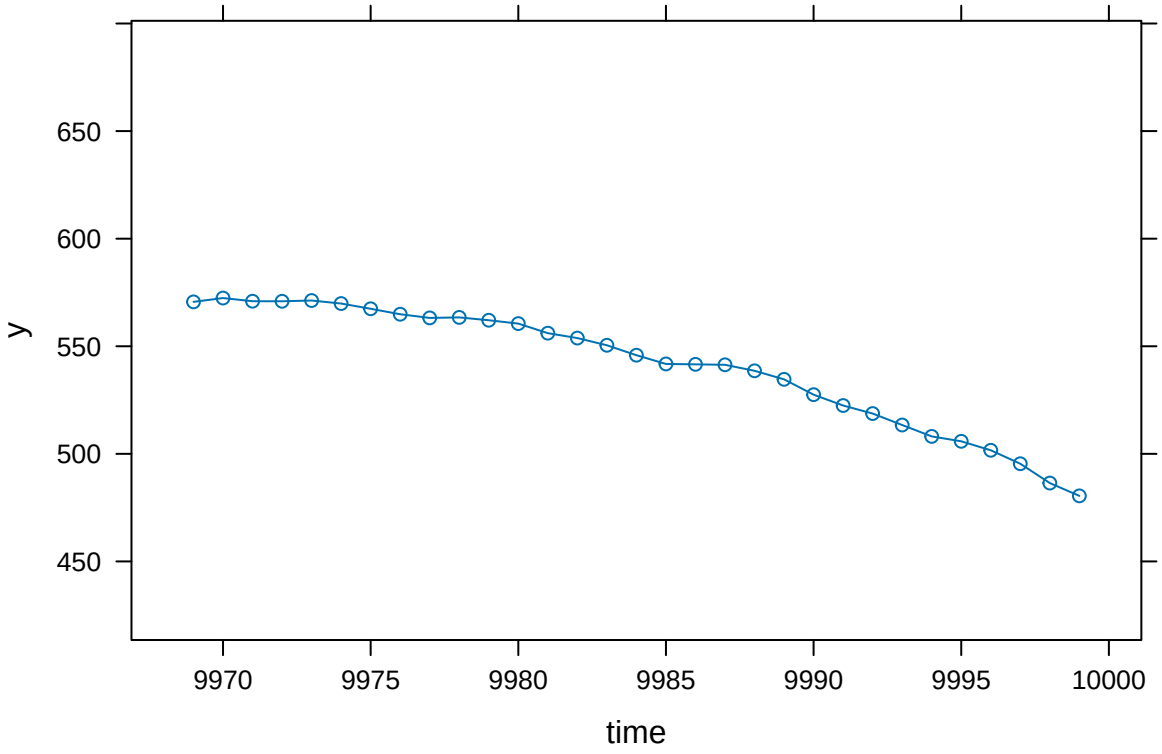
```
## Note: 'data' coerced to 'ts_data_frame'
```

```
## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'
```

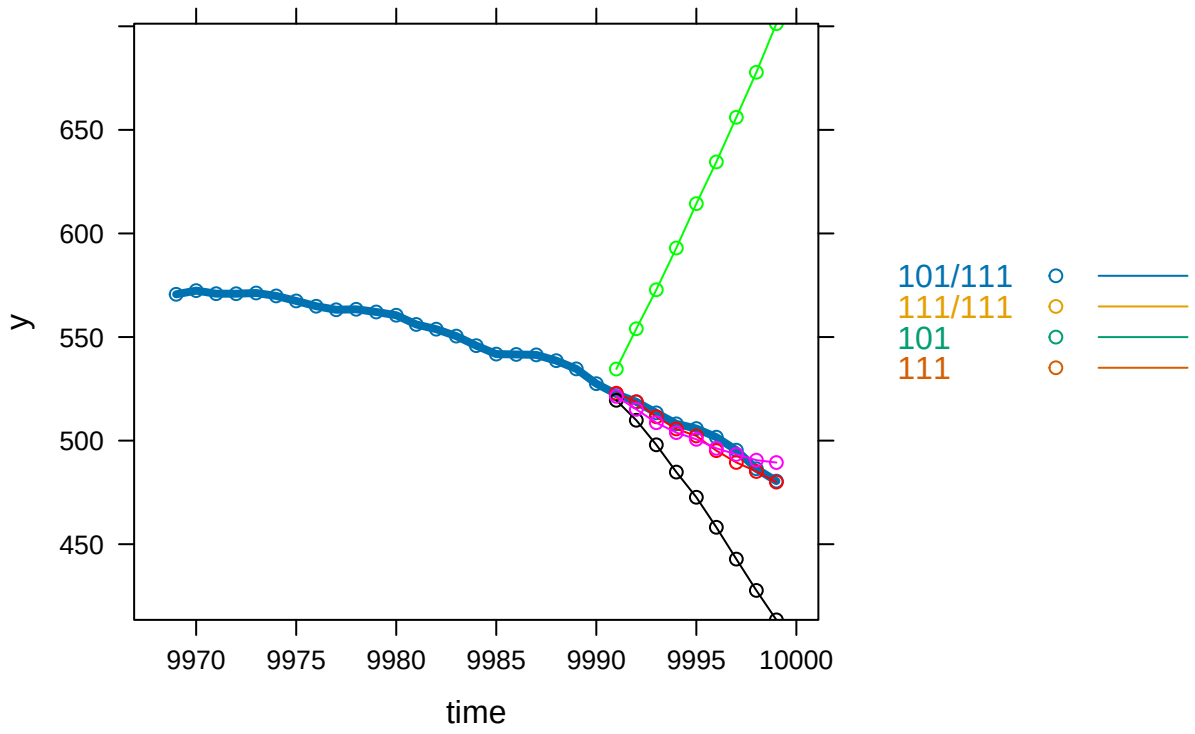
```
## Warning in stats::arima(y, order = order, seasonal = seasonal, xreg = x, :
## possible convergence problem: optim gave code = 1
```

```
## Note: 'data' coerced to 'ts_data_frame'
```

```
## Note: 'data' coerced to 'ts_data_frame'
## Warning in arma2pi(ar = ar, ma = ma, ar.seasonal = sar, ma.seasonal = sma): all
## 100 pi weights retained
## Note: 'data' coerced to 'ts_data_frame'
```

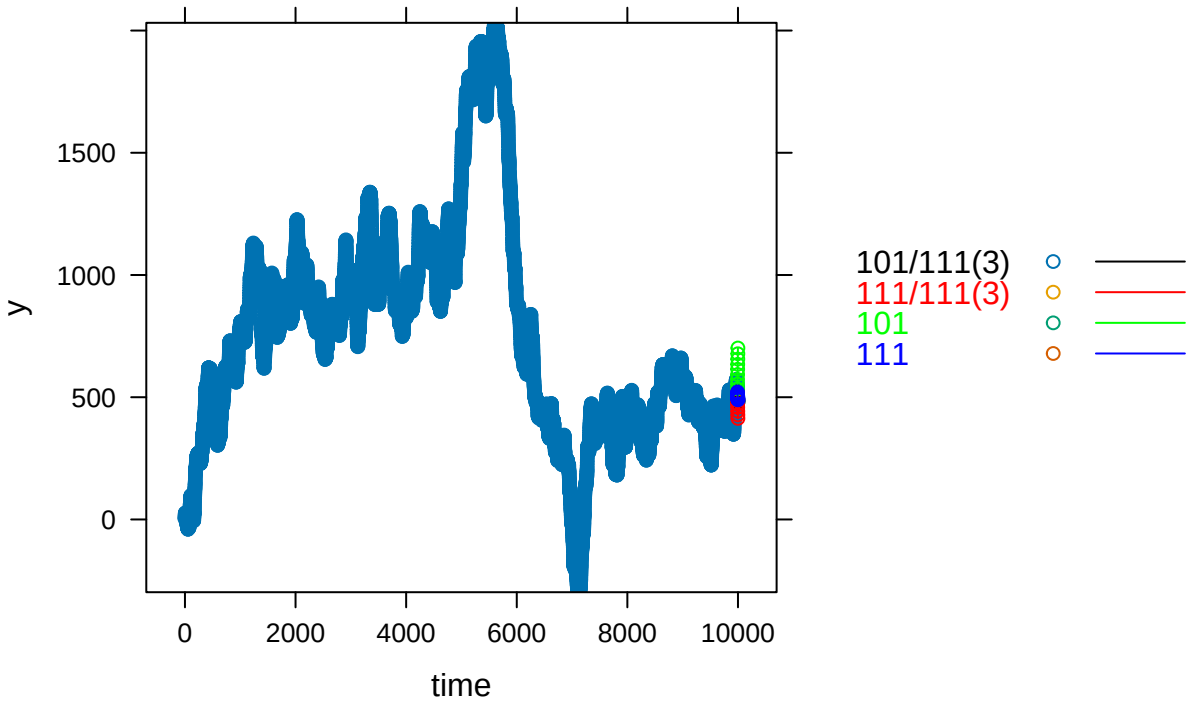



```
## Note: 'data' coerced to 'ts_data_frame'
## Warning in stats::arima(y, order = order, seasonal = seasonal, xreg = x, :
## possible convergence problem: optim gave code = 1
## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'
## Warning in arma2pi(ar = ar, ma = ma, ar.seasonal = sar, ma.seasonal = sma): all
## 100 pi weights retained
## Note: 'data' coerced to 'ts_data_frame'
```



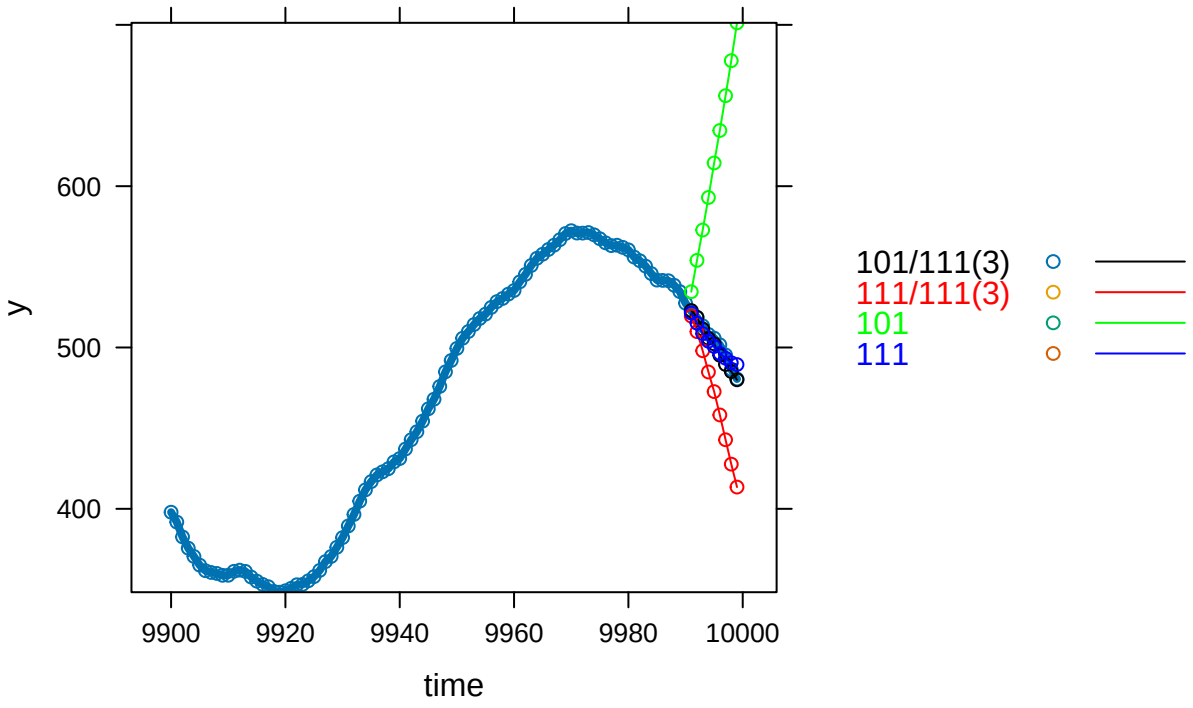
Predicting last 9 from whole series using all data for leadup. Correct model: 111

```
## Note: 'data' coerced to 'ts_data_frame'

## Warning in stats::arima(y, order = order, seasonal = seasonal, xreg = x, :
## possible convergence problem: optim gave code = 1

## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'

## Warning in arma2pi(ar = ar, ma = ma, ar.seasonal = sar, ma.seasonal = sma): all
## 100 pi weights retained

## Note: 'data' coerced to 'ts_data_frame'
```



Correct model: 111

```
## Note: 'data' coerced to 'ts_data_frame'

## Warning in stats::arima(y, order = order, seasonal = seasonal, xreg = x, :
## possible convergence problem: optim gave code = 1

## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'

## Warning in arma2pi(ar = ar, ma = ma, ar.seasonal = sar, ma.seasonal = sma): all
## 100 pi weights retained

## Note: 'data' coerced to 'ts_data_frame'
```

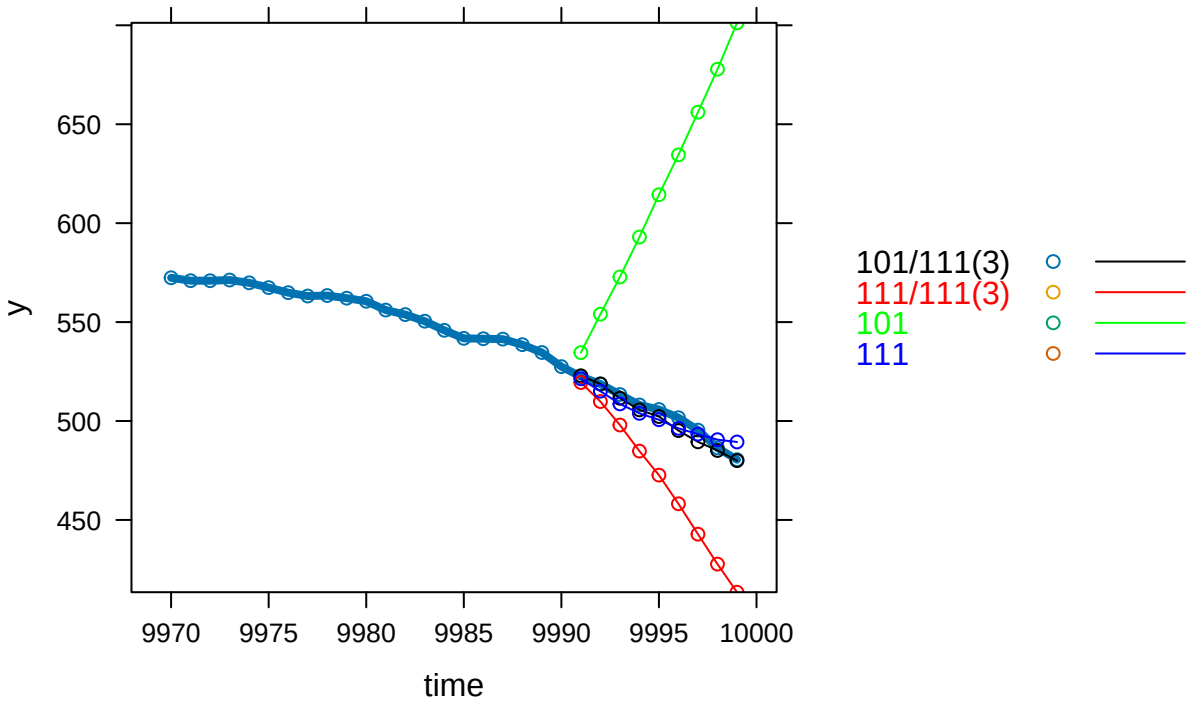


Correct model: 111

```
## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'
```

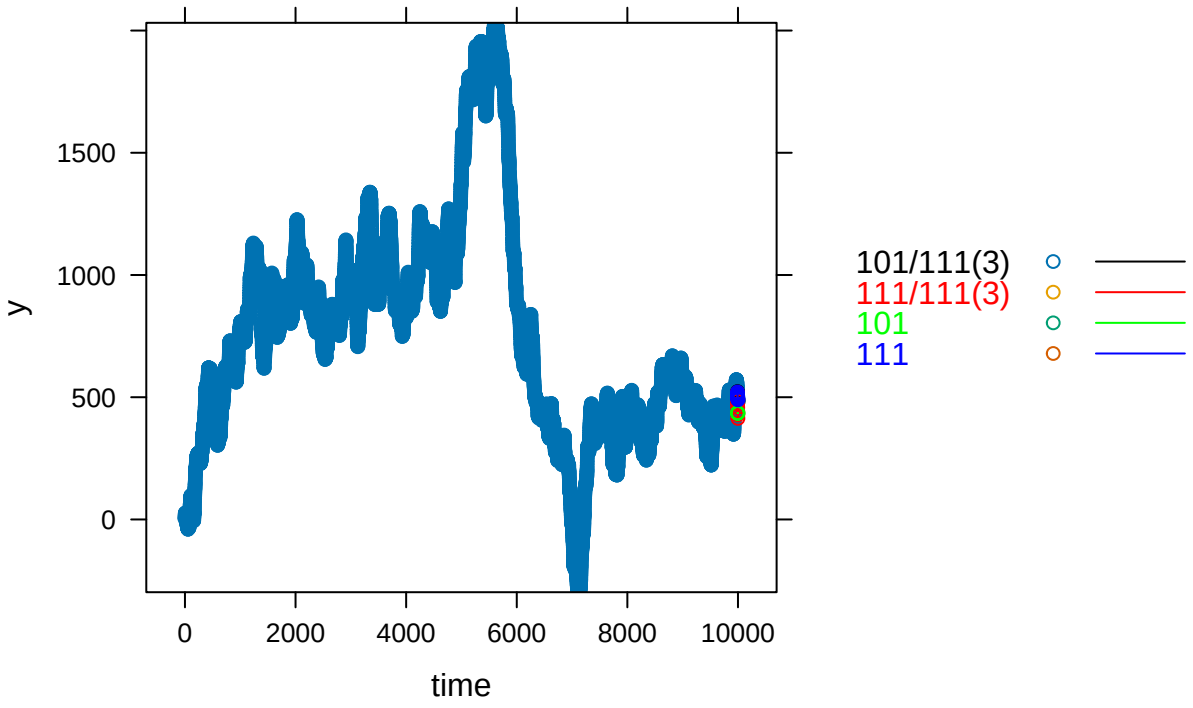
```
## Warning in arma2pi(ar = ar, ma = ma, ar.seasonal = sar, ma.seasonal = sma): all
## 100 pi weights retained
```

```
## Note: 'data' coerced to 'ts_data_frame'
```



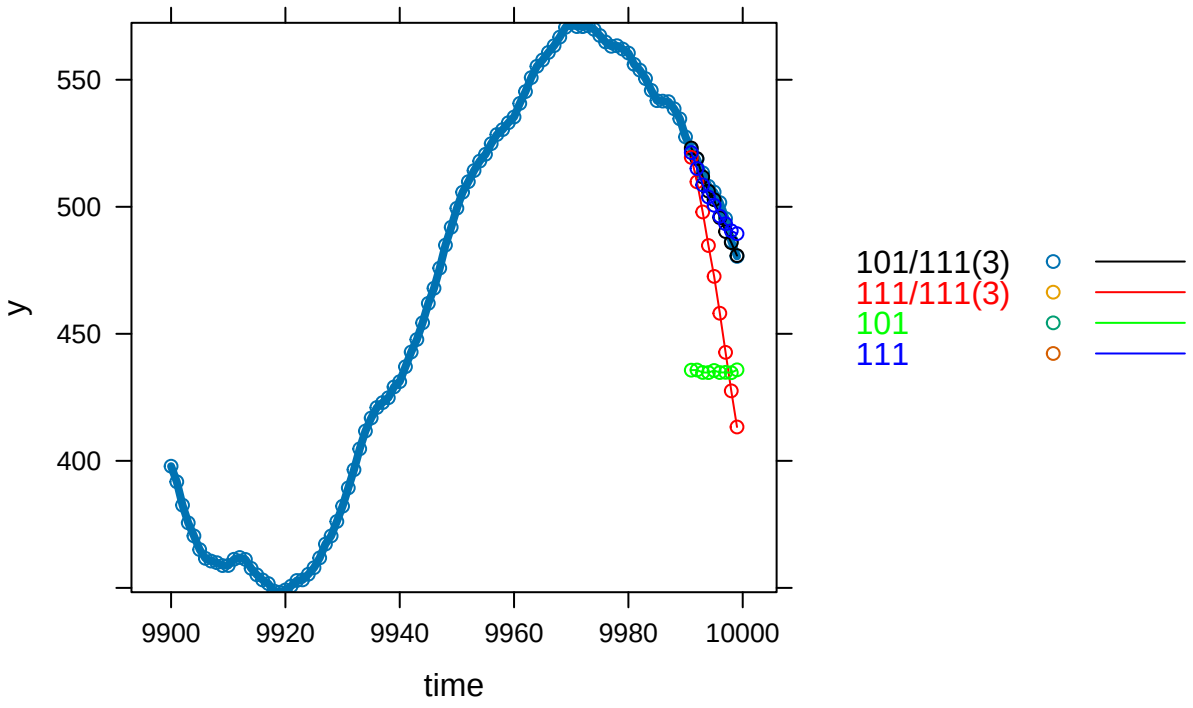
Correct model: 111

```
## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'

## Warning in arma2pi(ar = ar, ma = ma, ar.seasonal = sar, ma.seasonal = sma): all
## 100 pi weights retained

## Note: 'data' coerced to 'ts_data_frame'
```



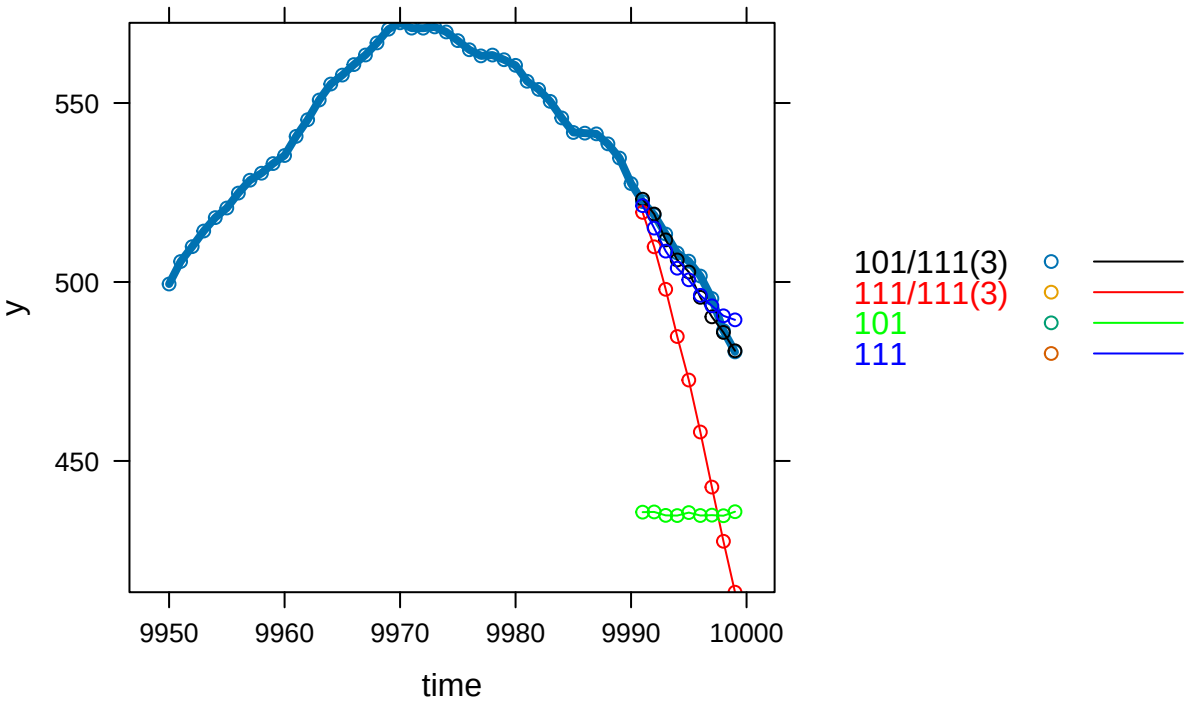
Correct model: 111

```
## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'

## Warning in arma2pi(ar = ar, ma = ma, ar.seasonal = sar, ma.seasonal = sma): all
## 100 pi weights retained

## Note: 'data' coerced to 'ts_data_frame'
```



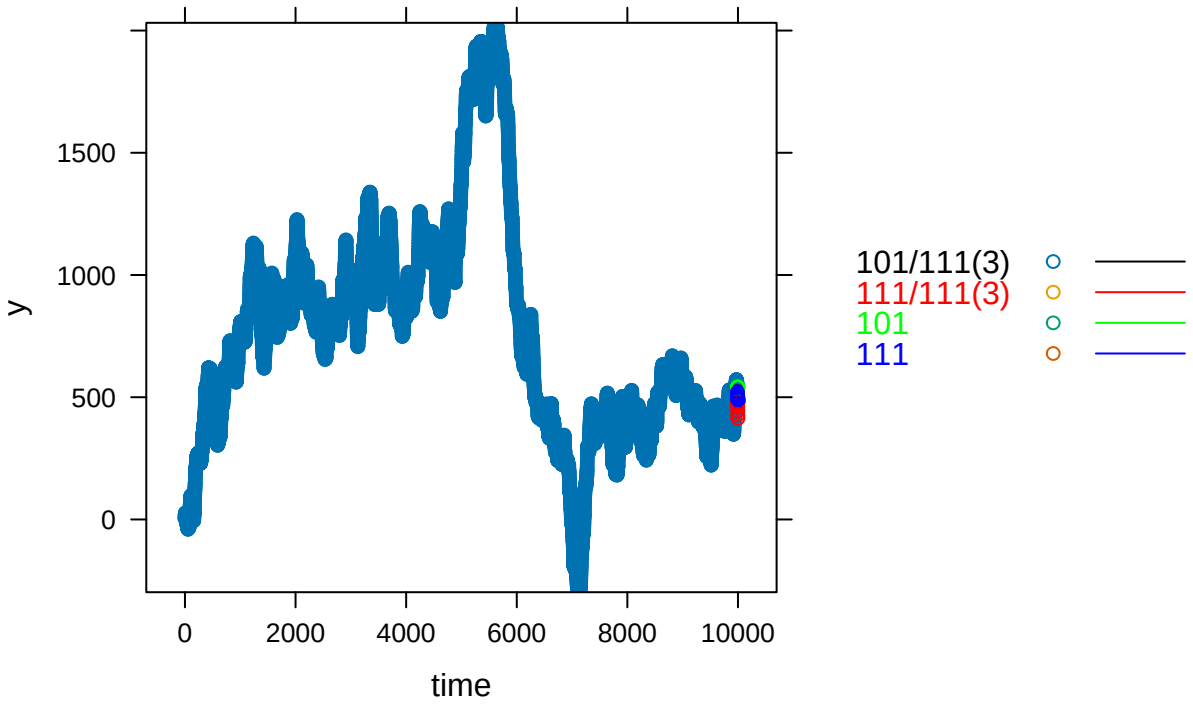
Correct model: 111

```
## Note: 'data' coerced to 'ts_data_frame'

## Warning in log(s2): NaNs produced

## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'

## Warning in arma2pi(ar = ar, ma = ma, ar.seasonal = sar, ma.seasonal = sma): all
## 100 pi weights retained

## Note: 'data' coerced to 'ts_data_frame'
```



Correct model: 111

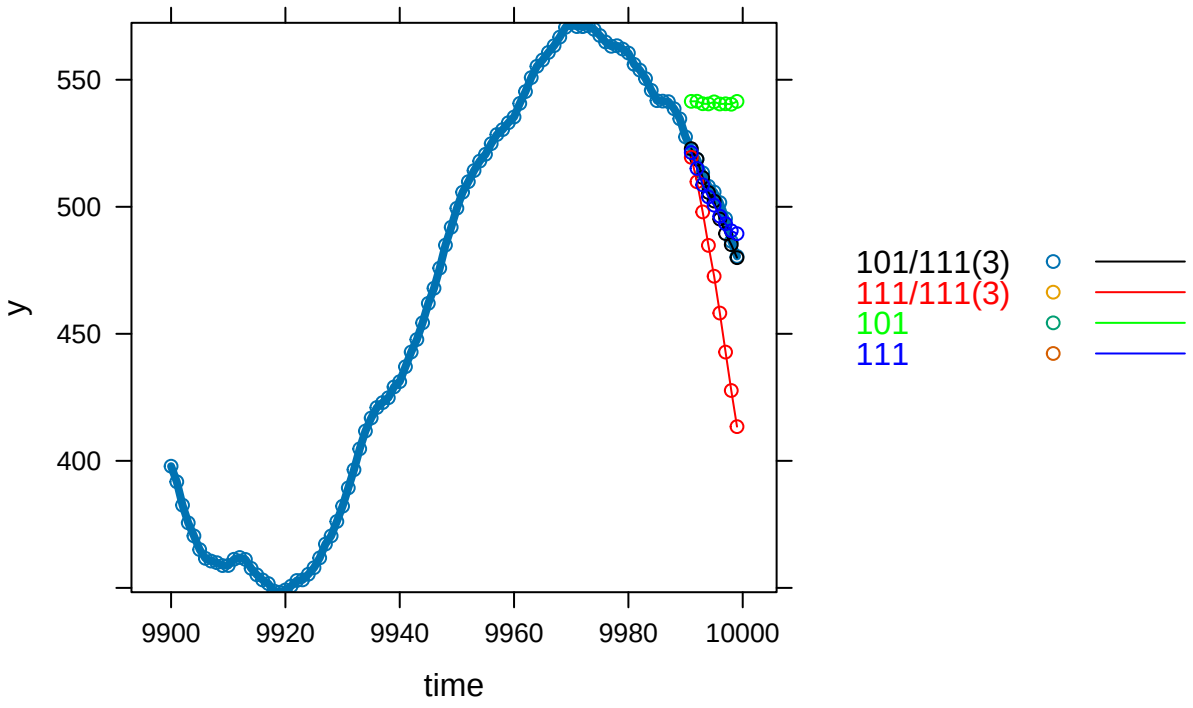
```
## Note: 'data' coerced to 'ts_data_frame'

## Warning in log(s2): NaNs produced

## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'

## Warning in arma2pi(ar = ar, ma = ma, ar.seasonal = sar, ma.seasonal = sma): all
## 100 pi weights retained

## Note: 'data' coerced to 'ts_data_frame'
```



Correct model: 111

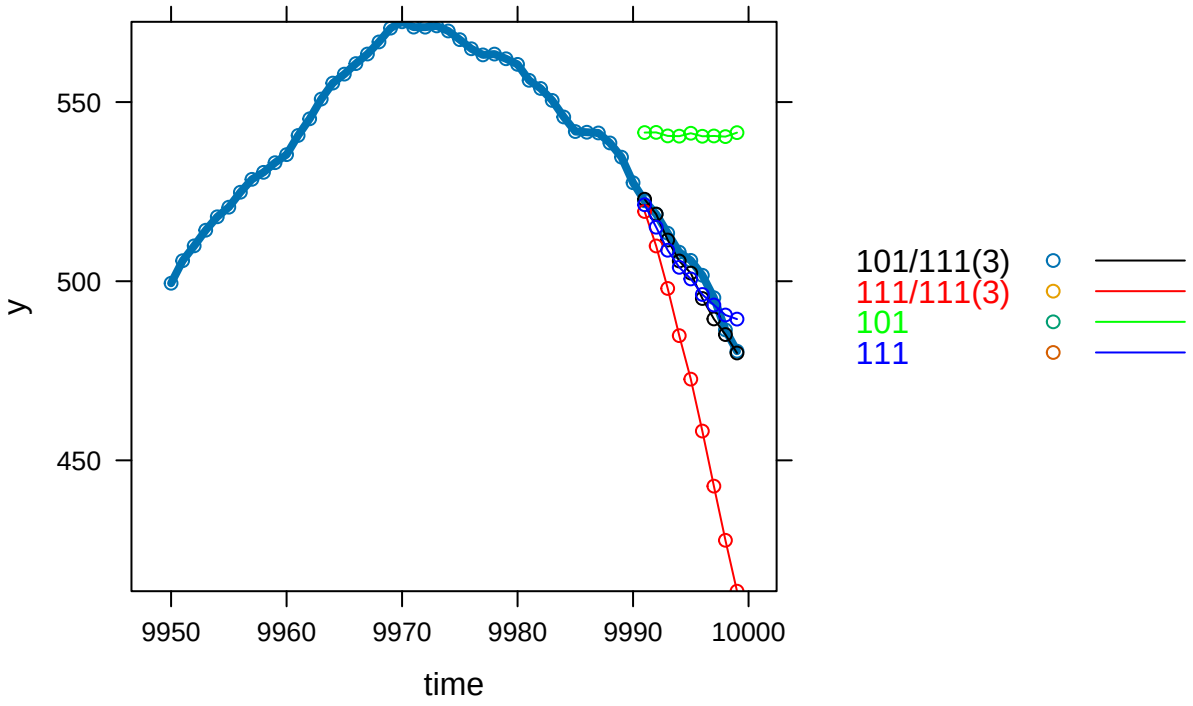
```
## Note: 'data' coerced to 'ts_data_frame'

## Warning in log(s2): NaNs produced

## Note: 'data' coerced to 'ts_data_frame'
## Note: 'data' coerced to 'ts_data_frame'

## Warning in arma2pi(ar = ar, ma = ma, ar.seasonal = sar, ma.seasonal = sma): all
## 100 pi weights retained

## Note: 'data' coerced to 'ts_data_frame'
```

Predicting last 9 from whole series using previous 90 for leadup



Correct model: 111

Predicting last 9 from whole series using previous 90 for leadup



Correct model: 111