Title

code - courseName

Christian Oppegård Moen

dd-mm-yyyy

Contents


```
library(viridis)
```

GARCH(p,q)

Definitions for notational purposes:

- $\mathcal{R}_1 = (r_1, ..., r_{max(p,q)})$
- $\mathcal{R}_2 = (r_t, ..., r_{t+max(p,q)})$
- $\boldsymbol{\alpha} = (\alpha_0, ..., \alpha_p)$
- $\boldsymbol{\beta} = (\beta_1, ..., \beta_q)$
- $\boldsymbol{r}_{tp} = (1, r_{t-1}, ..., r_{t-p})^\top$
- $\boldsymbol{\sigma}^2_{tq} = (\sigma^2_{t-1}, ..., \sigma^p_{t-q})^\top$

GARCH(p, q) is given by

$$r_t = \sigma_t \epsilon_t, \tag{1}$$

$$\sigma_t^2 = \alpha_0 + \sum_{j=1}^{p} \alpha_j r_{t-j}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2. \tag{2}$$

$$\sigma_t^2 = \boldsymbol{\alpha} \boldsymbol{r}_{tp}^2 + \boldsymbol{\beta} \boldsymbol{\sigma}_{tq}^2. \tag{3}$$

We know that $r_t | \mathcal{R}_2 \sim \mathcal{N}(0, \sigma_t^2)$

The log likelihood $l(\boldsymbol{\alpha}, \boldsymbol{\beta} | \mathcal{R}_1) \propto - \ln L(\boldsymbol{\alpha}, \boldsymbol{\beta} | \mathcal{R}_1)$ is given by

$$l(\boldsymbol{\alpha}, \boldsymbol{\beta} | \mathcal{R}_1) = \sum_{t=m+1}^{n} \ln(\sigma_t^2)) + \frac{r_t^2}{\sigma_t^2} \tag{4}$$

In the following chunk we implement the so called critical function.

```
garchLL = function(params, r, p = 1, q = 0) {

    n = length(r)
    m = max(p, q)

    alpha = exp(params[1:(p + 1)])
    if (q == 0) {
        beta = 0
    } else {
        beta = exp(params[-(1:(p + 1))])
    }

    # initialize variance and set first m values
    sigma_sq = numeric(n)
    sigma_sq[1:m] = t(alpha) %*% c(1, r[p:1]^2)  # should they be zero?
    # sigma_sq[1:m] = 0 # says so above eq. (5.52)

    # Iteratively compute each variance
    for (t in (m + 1):n) {
        sigma_sq[t] = sum(alpha * c(1, r[(t - 1):(t - p)]^2)) + sum(beta * sigma_sq[(t -
            1):(t - q)])
```

```
    }
    ll = sum(log(sigma_sq) + r^2/sigma_sq)
    return(ll)
}
```

The variance σ_t^2 can be estimated by *one-step-ahead* forecasting given by

$$\hat{\sigma}_t^2 = \hat{\boldsymbol{\alpha}} \boldsymbol{r}_{tp}^2 + \hat{\boldsymbol{\beta}} \hat{\boldsymbol{\sigma}}_{tq}^2 \tag{5}$$

```
sigmaForecast = function(mod) {
    n = mod$n
    m = mod$m
    p = mod$p
    r = mod$r

    alpha = mod$estim[1:(p + 1)]
    if (mod$q == 0)
        {
            beta = 0
            q = 1
        }  # If ARCH(p) model
 else {
        beta = mod$estim[-(1:(p + 1))]
        q = mod$q
    }

    sf = numeric(n)  # sigma squared forecasts
    # sf[1:p] = sum(alpha*c(1,r[p:1]^2)) sf[1:q] = 0 sf[1:m] =
    # alpha[1]/(1-sum(c(alpha, beta))) # GPT
    sf[1:m] = sum(alpha * c(1, r[1:p]^2))  # used in sim test

    for (t in (m + 1):n) {
        sf[t] = sum(alpha * c(1, r[(t - 1):(t - p)]^2)) + sum(beta * sf[(t - 1):(t -
            q)])
    }
    return(sf)
}

# TODO: make garch function
Garch = function(r, p = 1, q = 0, init = c(0.1, 0.1)) {
    n = length(r)

    alpha = init[1:(p + 1)]
    if (q == 0)
        {
            beta = 0
            q = 1
        }  # If ARCH(p) model
 else {
        beta = init[-(1:(p + 1))]
        q = q
    }
```

```
    estim = exp(optim(par = log(init), fn = garchLL, r = r, p = p, q = q, method = "BFGS")$par)

    mod = list(p = p, q = q, n = n, m = max(p, q), init = init, estim = estim, r = r)

    mod$sigmaForecast = sigmaForecast(mod)
    return(mod)

}

modResults = function(mod, main = "", plot = T) {
    # print(rbind(estim = mod$estim, init = mod$init))
    if (plot) {
        plot(mod$r, type = "l", main = main)
        lines(mod$sigmaForecast, col = "cyan")
    }
}
```

Tests

```
testModel = function(mod) {
    n = mod$n
    m = mod$m
    p = mod$p

    alpha = mod$paramSim[1:(p + 1)]
    if (mod$q == 0) {
        beta = 0
        q = 1
    } else {
        beta = mod$paramSim[-(1:(p + 1))]
        q = mod$q
    }

    r = numeric(n)
    r[1:m] = rnorm(m)
    sigma_sq = numeric(n)
    sigma_sq[1:m] = sum(alpha * c(1, r[1:p]^2))
    # sigma_sq[1:m] = alpha[1]/(1-sum(c(alpha, beta))) # GPT

    # browser()
    for (t in (m + 1):n) {
        sigma_sq[t] = sum(alpha * c(1, r[(t - 1):(t - p)]^2)) + sum(beta * sigma_sq[(t -
            1):(t - q)])
        r[t] = rnorm(1, sd = sqrt(sigma_sq[t]))
    }
    # browser() estimation
    estim = exp(optim(par = log(mod$init), fn = garchLL, r = r, p = p, q = q, method = "BFGS")$par)
    comparison = (rbind(real = mod$paramSim, estim = estim, init = mod$init))
    return(list(r = r, estim = estim, comparison = comparison))
}
```

```r
set.seed(420)
arch1 = list(p = 1, q = 0, m = 1, n = 10000, init = rep(0.1, 2), paramSim = c(0.01,
    0.2))


testResult = testModel(arch1)
testResult$comparison
```

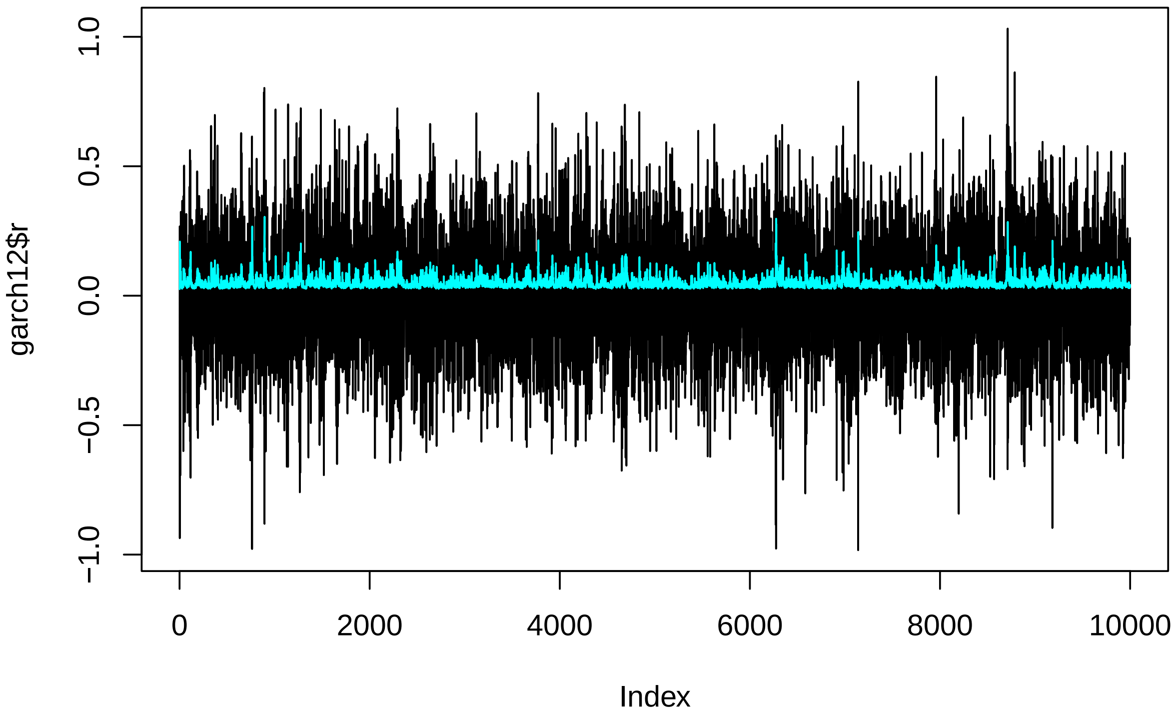
```
##              [,1]      [,2]
## real  0.010000000 0.2000000
## estim 0.009775681 0.1997736
## init  0.100000000 0.1000000
```

```r
# qqnorm(testResult$r) plot(testResult$r)
```

```r
set.seed(420)
garch12 = list(p = 1, q = 2, m = 2, n = 10000, init = rep(0.1, 4), paramSim = c(0.01,
    0.2, 0.1, 0.5))


testResult = testModel(garch12)
garch12$estim = testResult$estim
round(testResult$comparison, 3)
```

```
##       [,1] [,2]  [,3]  [,4]
## real  0.01 0.20 0.100 0.500
## estim 0.01 0.21 0.112 0.464
## init  0.10 0.10 0.100 0.100
```

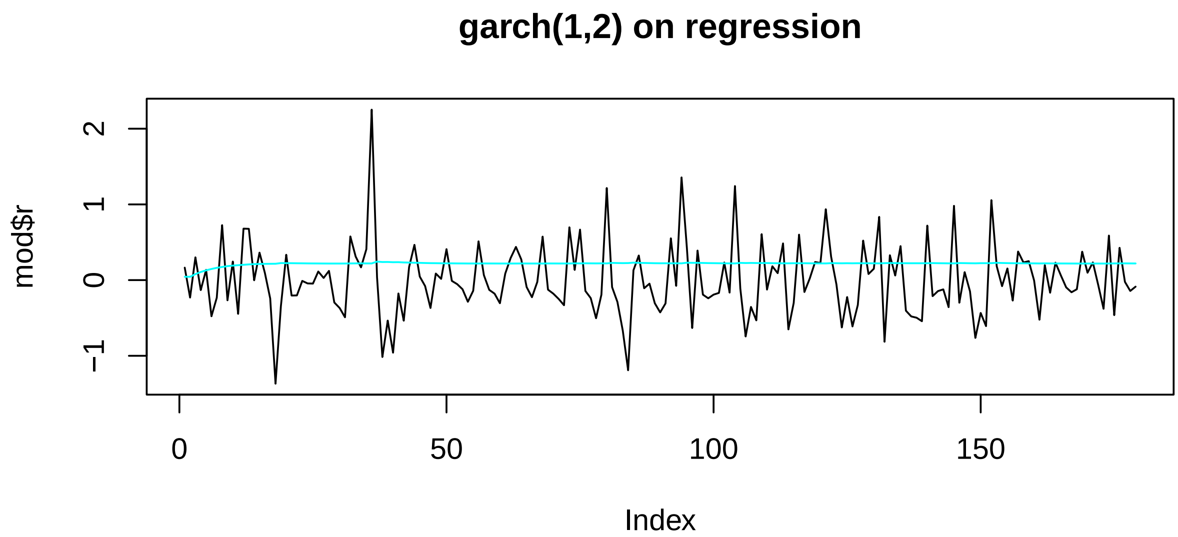
```r
garch12$r = testResult$r
garch12$sigmaForecast = sigmaForecast(garch12)
plot(garch12$r, type = "l")
lines(garch12$sigmaForecast, col = "cyan")
```

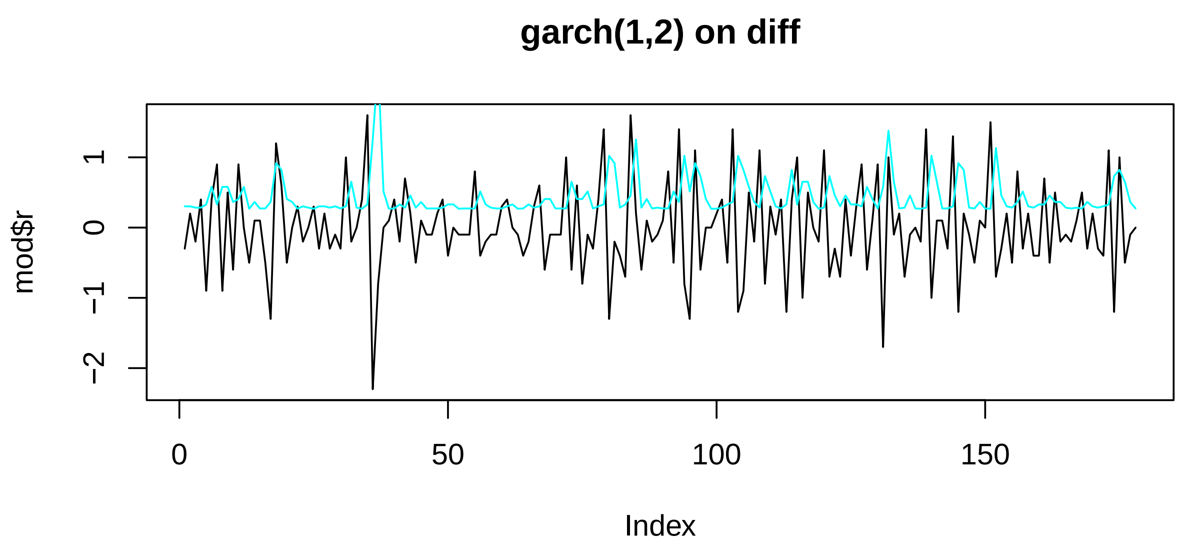


Real data (HOW YOU FIT IT AND VIEW IT)

```
df <- read.csv("projectdata.csv", header = T, sep = ";", dec = ",", stringsAsFactors = FALSE)


# qqnorm(r) plot(r, type = 'l')
```

```
# make data to fit
fit = lm(Inflation ~ Unemployed + Consumption + InterestRate, data = df)
r = df$Inflation - fit$fitted.values
d = diff(df$Inflation)
par(mfrow = c(2, 1))
# inflation - regression
garch12regr = Garch(r, p = 1, q = 2, init = rep(0.1, 1 + 2 + 1))
modResults(garch12regr, main = "garch(1,2) on regression")

# diff inflation
garch12 = Garch(d, p = 1, q = 2, init = rep(0.1, 1 + 2 + 1))
modResults(garch12, "garch(1,2) on diff")
```

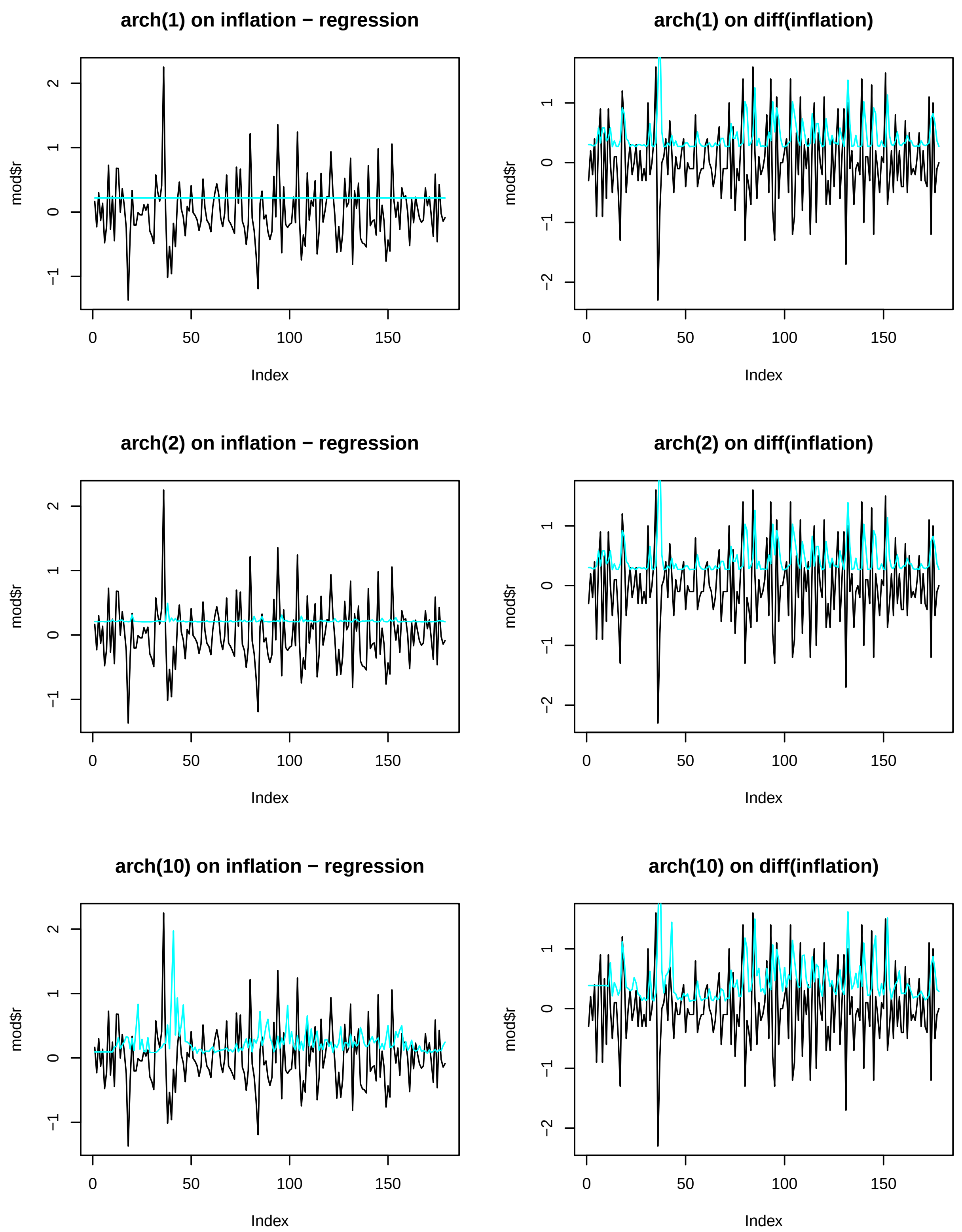





```r
par(mfrow = c(1, 1))
```

```r
par(mfrow = c(3, 2))
arch1regr = Garch(r, p = 1, q = 0, init = rep(0.1, 2))
modResults(arch1regr, main = "arch(1) on inflation - regression")

arch1d = Garch(d, p = 1, q = 0, init = rep(0.1, 2))
modResults(arch1d, main = "arch(1) on diff(inflation)")

arch2regr = Garch(r, p = 2, q = 0, init = rep(0.1, 3))
modResults(arch2regr, main = "arch(2) on inflation - regression")

arch2d = Garch(d, p = 2, q = 0, init = rep(0.1, 3))
```

```r
modResults(arch2d, "arch(2) on diff(inflation)")

arch10regr = Garch(r, p = 10, q = 0, init = rep(0.1, 11))
modResults(arch10regr, main = "arch(10) on inflation - regression")

arch10d = Garch(d, p = 10, q = 0, init = rep(0.1, 11))
modResults(arch10d, "arch(10) on diff(inflation)")
```




```
par(mfrow = c(1, 1))
```

```r
list(arch1regr = arch1regr$estim, arch1d = arch1d$estim, arch2regr = arch2regr$estim,
    arch2d = arch2d$estim, arch10regr = arch10regr$estim, arch10d = arch10d$estim)
```

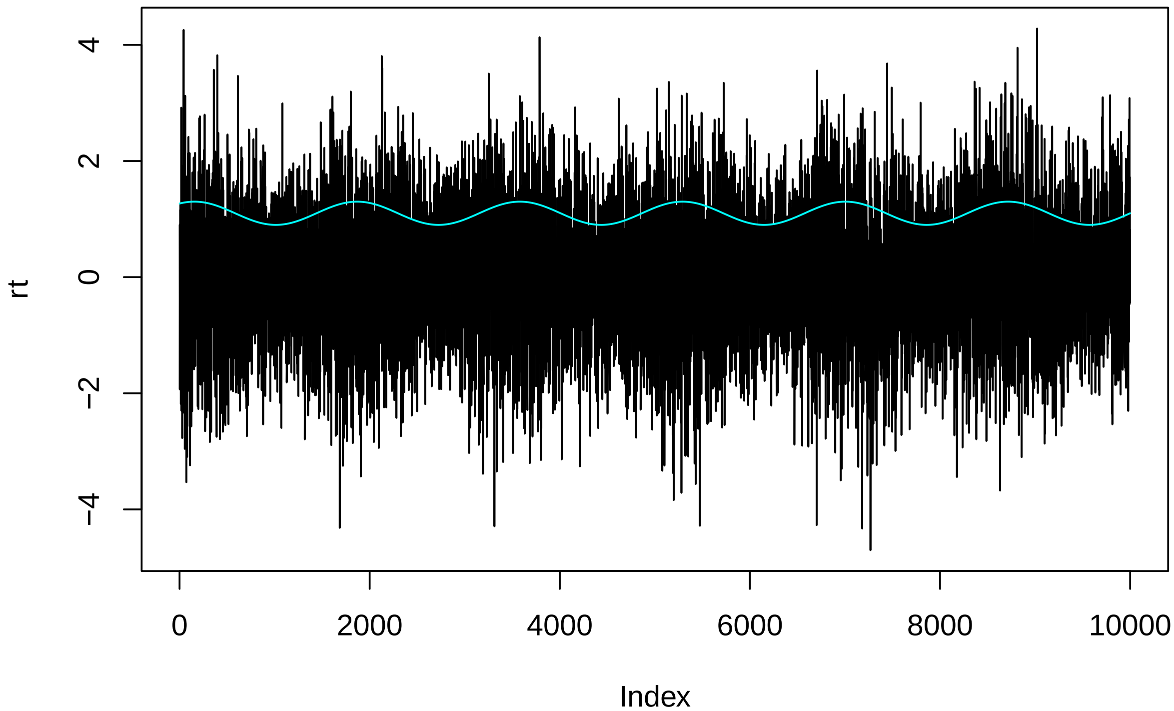
```
## $arch1regr
## [1] 2.172051e-01 2.631814e-05
##
## $arch1d
## [1] 0.2681048 0.3845943
##
## $arch2regr
## [1] 2.040528e-01 2.842043e-05 5.669071e-02
##
## $arch2d
## [1] 2.673872e-01 3.877460e-01 9.556793e-05
##
## $arch10regr
##  [1] 6.607910e-02 1.507028e-06 7.553767e-02 4.280618e-05 1.562249e-01
##  [6] 3.652726e-01 2.839493e-07 8.483807e-02 2.002727e-06 6.598773e-07
## [11] 1.301979e-01
##
## $arch10d
##  [1] 1.143832e-01 4.845580e-01 5.315562e-08 2.358752e-08 5.056129e-02
##  [6] 6.419528e-02 5.766002e-08 2.064997e-01 3.922694e-09 4.332273e-13
## [11] 4.898945e-08
```

```r
models = list(arch1regr = arch1regr, arch1d = arch1d, arch2regr = arch2regr, arch2d = arch2d,
    arch10regr = arch10regr, arch10d = arch10d, garch12regr = garch12regr, garch12 = garch12)
```

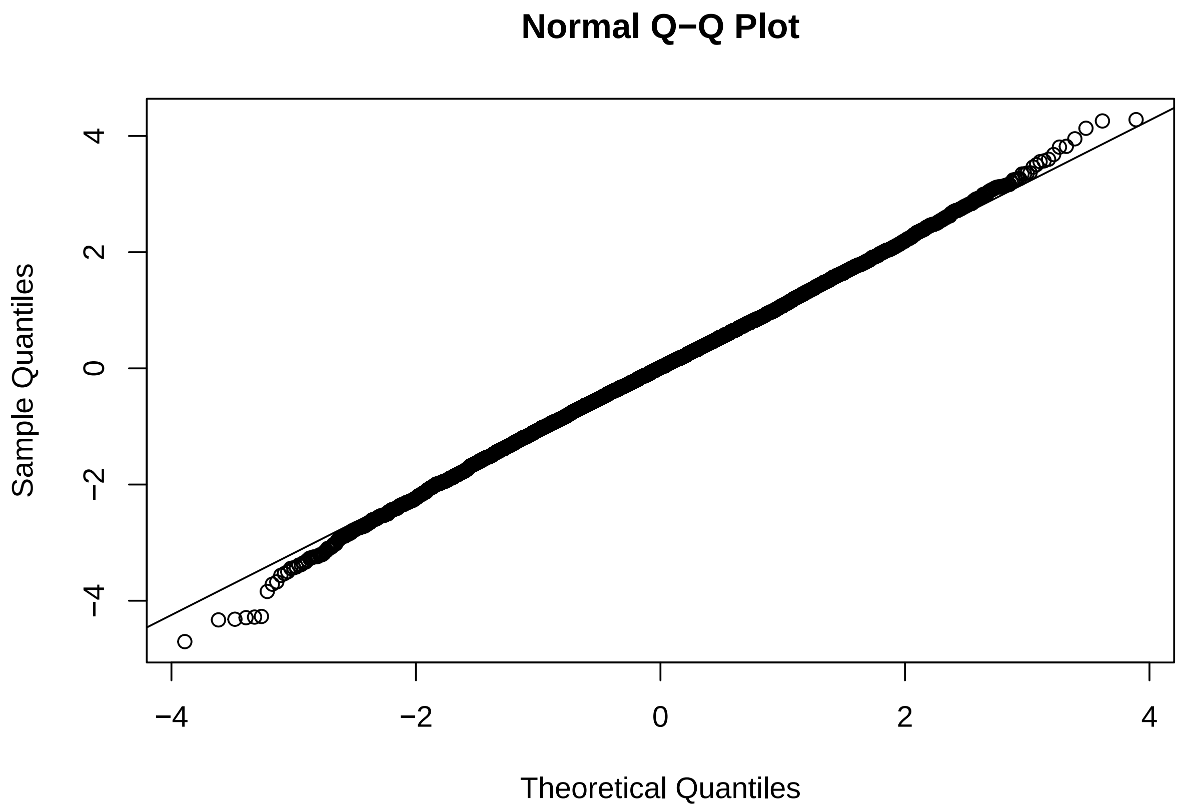
Make more models

A Small test to see what type of data we want. Can delete

```r
nt = 10000
xt = 1.1 + sin(seq(1, 6 * 2 * pi, length.out = nt))/5
rt = rnorm(nt, sd = xt)
par(mfrow = c(1, 1))
plot(rt, type = "l")
lines(xt, col = "cyan")
```




```
qqnorm(rt)
qqline(rt)
```



Model selection

```
logLike = function(mod, m = 1) {
    i = (m + 1):(mod$n)  # can burn m first values
    ll = sum(-0.5 * (log(mod$sigmaForecast[i]) + mod$r[i]^2/mod$sigmaForecast[i]))
    return(ll)
}


aic = function(mod, m = 1) {
    return(2 * (mod$p + mod$q) - 2 * logLike(mod, m))
}

loglikes = numeric(length(models))
aics = numeric(length(models))
loglikesM = numeric(length(models))
aicsM = numeric(length(models))

for (i in 1:length(models)) {
    loglikes[i] = logLike(models[[i]])
    aics[i] = aic(models[[i]])
    loglikesM[i] = logLike(models[[i]], models[[i]]$m)
```

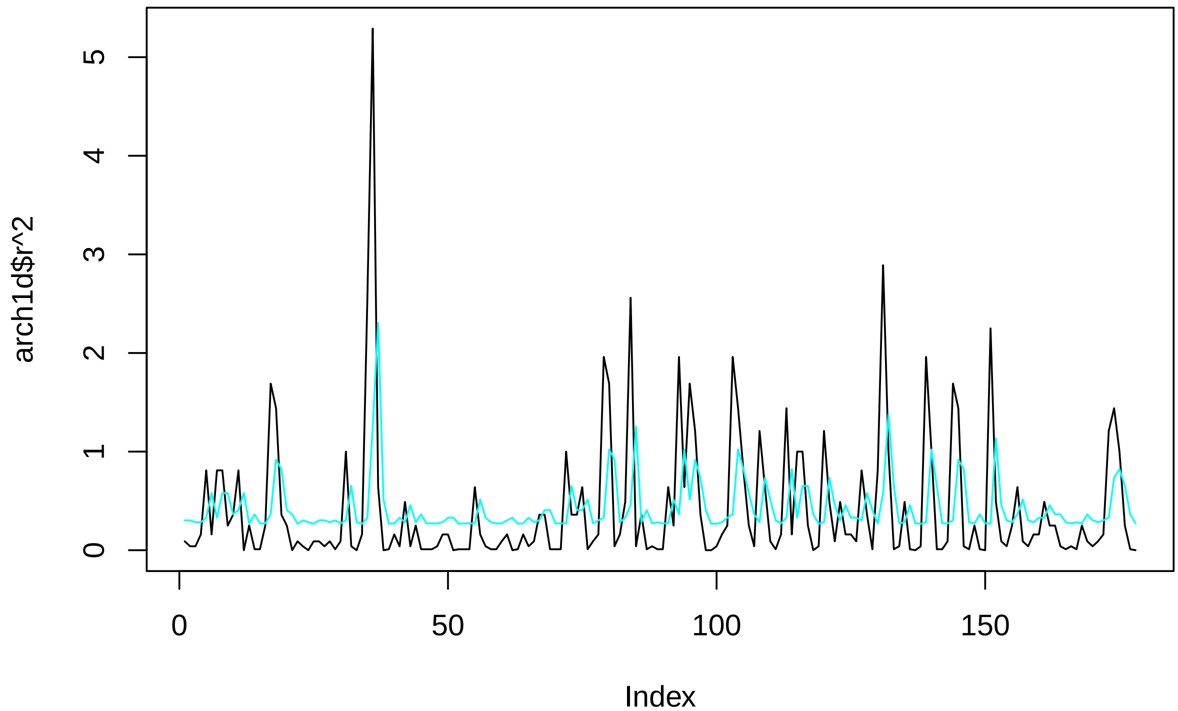
```r
    aicsM[i] = aic(models[[i]], models[[i]]$m)
    print(sum(models[[i]]$r^2/models[[i]]$sigmaForecast))
}
```

```
## [1] 178.9952
## [1] 178
## [1] 178.9979
## [1] 177.9857
## [1] 184.6648
## [1] 177.6991
## [1] 178.8455
## [1] 177.9982
```

```r
modSel = cbind(ll = loglikes, llM = loglikesM, aic = aics, aicM = aicsM)
rownames(modSel) = names(models)
modSel
```

```
##                    ll        llM       aic      aicM
## arch1regr   46.4575718 46.4575718 -88.91514 -88.91514
## arch1d      -4.4729068 -4.4729068  12.94581  12.94581
## arch2regr   47.2728846 46.6138710 -88.54577 -87.22774
## arch2d      -4.4763785 -5.0084038  14.95276  16.01681
## arch10regr  53.9399839 49.2894127 -85.87997 -76.57883
## arch10d     -0.6363949 -0.4641352  23.27279  22.92827
## garch12regr 47.3758710 46.4187290 -88.75174 -86.83746
## garch12     -4.4729649 -5.0043698  14.94593  16.00874
```

```r
plot(arch1d$r^2, type = "l")
lines(arch1d$sigmaForecast, col = "cyan")
```




```
sum(arch1d$r^2/arch1d$sigmaForecast)
```

```
## [1] 178
```

```
length(arch1d$sigmaForecast)
```

```
## [1] 178
```

```
fgarch = fGarch::garchFit(~garch(1, 0), data = d, trace = F)
fit = fGarch::volatility(fgarch)
fit - arch1d$sigmaForecast
```

```
##   [1]  0.35440169  0.24922689  0.24183423  0.24856248  0.23595246  0.19512649
##   [7]  0.23595246  0.17418245  0.19512649  0.23050035  0.23906245  0.17418245
##  [13]  0.24540461  0.24526913  0.24367574  0.24367574  0.24526913  0.05947873
##  [19]  0.07930232  0.22231636  0.24526913  0.24540461  0.23947362  0.24856248
##  [25]  0.24540461  0.23947362  0.24922689  0.24183423  0.24922689  0.24711361
##  [31]  0.24922689  0.14832394  0.24856248  0.24540461  0.23595246 -0.13538916
##  [37] -0.75405448  0.21458181  0.24540461  0.24367574  0.23595246  0.24856248
##  [43]  0.21063996  0.24183423  0.24526913  0.24367574  0.24711361  0.24711361
##  [49]  0.24183423  0.23595246  0.24839407  0.24540461  0.24711361  0.24711361
##  [55]  0.24711361  0.19479056  0.24839407  0.24856248  0.24711361  0.24711361
##  [61]  0.23947362  0.23595246  0.24540461  0.24711361  0.24839407  0.24856248
```

```
##  [67]  0.23947362  0.22231636  0.23906245  0.24711361  0.24711361  0.24711361
##  [73]  0.14832394  0.23906245  0.22231636  0.21458181  0.24711361  0.24922689
##  [79]  0.23595246 -0.01472241  0.05947873  0.24856248  0.24839407  0.22904623
##  [85] -0.13538916  0.24183423  0.23906245  0.24367574  0.24856248  0.24711361
##  [91]  0.24367574  0.19479056  0.24526913 -0.01472241  0.21458181  0.05947873
##  [97]  0.11680841  0.23906245  0.24540461  0.24540461  0.24183423  0.23595246
## [103]  0.24526913 -0.01472241  0.10268116  0.19512649  0.23050035  0.24856248
## [109]  0.11680841  0.21458181  0.23947362  0.24711361  0.23595246  0.10268116
## [115]  0.23595246  0.14832394  0.17022817  0.23050035  0.24540461  0.24856248
## [121]  0.11680841  0.22904623  0.24922689  0.22904623  0.23595246  0.24839407
## [127]  0.23947362  0.17418245  0.23906245  0.24367574  0.17418245 -0.18058596
## [133]  0.14832394  0.24711361  0.24183423  0.22904623  0.24711361  0.24540461
## [139]  0.24856248 -0.01472241  0.17022817  0.24367574  0.24367574  0.24922689
## [145]  0.03553300  0.10268116  0.24183423  0.24711361  0.24526913  0.24367574
## [151]  0.24540461 -0.07164645  0.22904623  0.24922689  0.24183423  0.24526913
## [157]  0.19479056  0.24922689  0.24183423  0.24839407  0.24839407  0.21063996
## [163]  0.24526913  0.23050035  0.24856248  0.24711361  0.24856248  0.24367574
## [169]  0.23050035  0.24922689  0.24183423  0.24922689  0.24839407  0.11680841
## [175]  0.10268116  0.14832394  0.24526913  0.24711361
## attr(,"type")
## [1] "sigma"
```

```r
sum(d^2/fit^2)
```

```
## [1] 178.8808
```

```r
fGarch::summary(fGarch::garchFit(~garch(1, 0), data = d, trace = F))$show[18]
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  fGarch::garchFit(formula = ~garch(1, 0), data = d, trace = F)
##
## Mean and Variance Equation:
##  data ~ garch(1, 0)
## <environment: 0x000002715f53dfc0>
##  [data = d]
##
## Conditional Distribution:
##  norm
##
## Coefficient(s):
##       mu     omega    alpha1
## 0.022472  0.263492  0.395732
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu       0.02247     0.03790    0.593   0.5533
```

```
## omega    0.26349     0.04388    6.005 1.91e-09 ***
## alpha1   0.39573     0.15060    2.628   0.0086 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  -167.3975    normalized:  -0.9404351
##
## Description:
##  Fri Nov 17 16:47:39 2023 by user: kikka
##
##
## Standardised Residuals Tests:
##                                 Statistic      p-Value
##  Jarque-Bera Test   R    Chi^2   11.2181424 3.664471e-03
##  Shapiro-Wilk Test  R    W         0.9680396 4.171312e-04
##  Ljung-Box Test     R    Q(10)   60.2226812 3.289101e-09
##  Ljung-Box Test     R    Q(15)  118.9905101 0.000000e+00
##  Ljung-Box Test     R    Q(20)  148.4393392 0.000000e+00
##  Ljung-Box Test     R^2  Q(10)   15.5022697 1.147947e-01
##  Ljung-Box Test     R^2  Q(15)   28.0330311 2.136318e-02
##  Ljung-Box Test     R^2  Q(20)   31.2118334 5.245529e-02
##  LM Arch Test       R    TR^2    23.7586298 2.193391e-02
##
## Information Criterion Statistics:
##      AIC      BIC      SIC     HQIC
## 1.914578 1.968204 1.914022 1.936325
```

```
## [1] "0.022472  0.263492  0.395732  "
```

```
arch1d$estim
```

```
## [1] 0.2681048 0.3845943
```

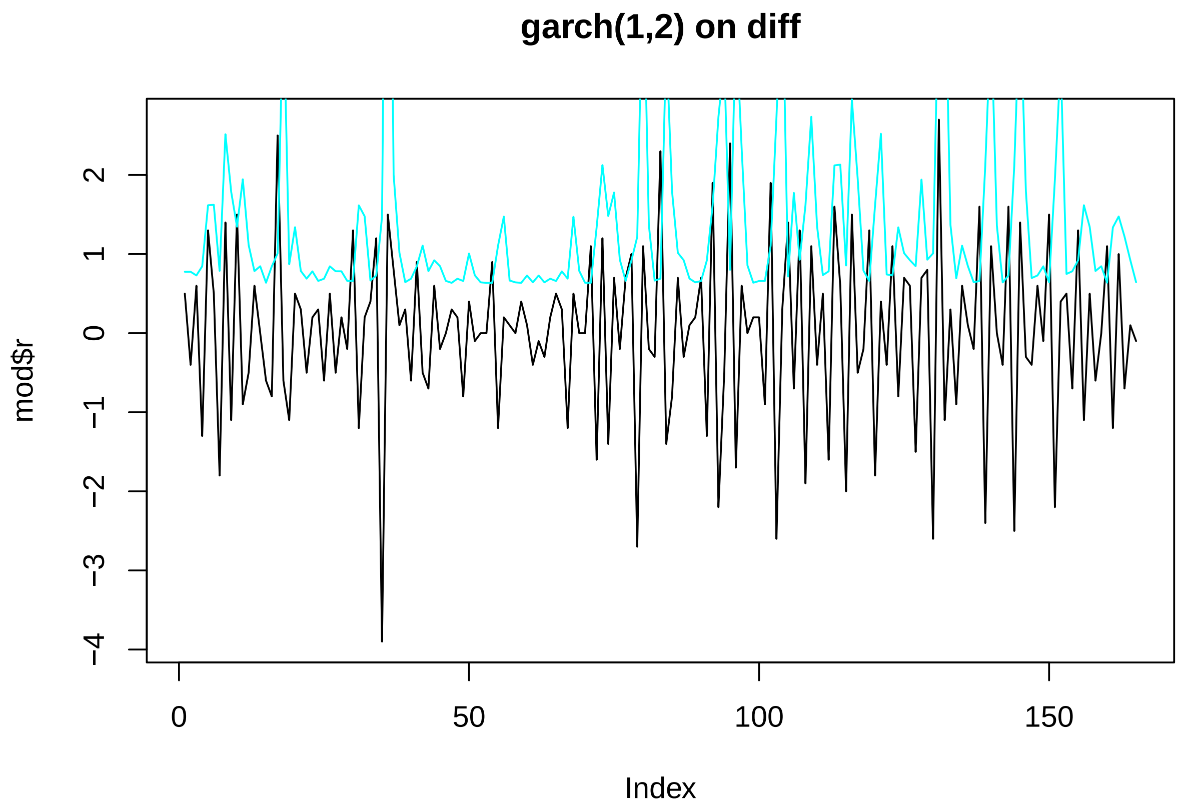
Forecasting

```
forecast = function(mod, n) {
    m = mod$m
    p = mod$p
    r = mod$r
    nTot = mod$n + n

    alpha = mod$estim[1:(p + 1)]
    if (mod$q == 0)
        {
            beta = 0
            q = 1
        }  # If ARCH(p) model
 else {
        beta = mod$estim[-(1:(p + 1))]
```

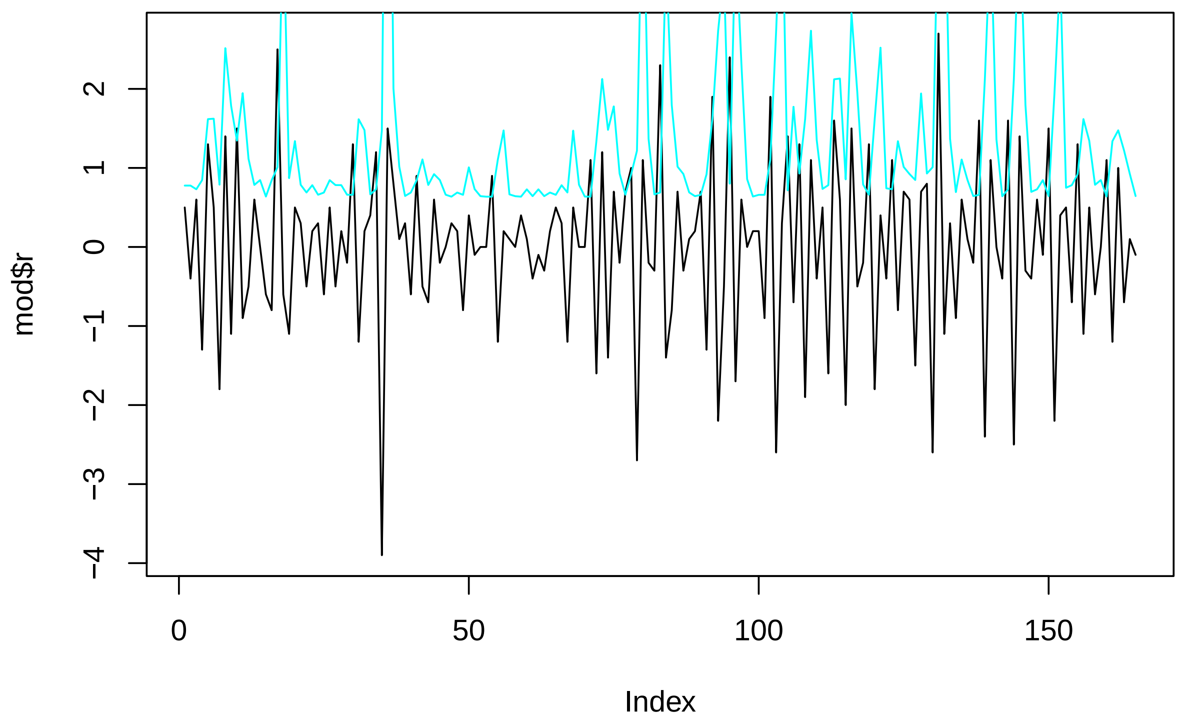
```r
        q = mod$q
    }

    rf = r
    length(rf) = nTot
    sf = mod$sigmaForecast
    length(sf) = nTot
    for (t in mod$n:(nTot)) {
        sf[t + 1] = sum(alpha * c(1, rf[(t + 1 - 1):(t + 1 - p)]^2)) + sum(beta *
            sf[(t + 1 - 1):(t + 1 - q)])
        rf[t + 1] = rnorm(1, sd = sqrt(sf[t + 1]))
    }
    return(rf)
}
```

Fit models without one year

```r
# make data to fit
without = 12
dTot = diff(df$Inflation)
nTot = length(dTot)
d = diff(dTot[1:(nTot - without)])

garch12 = Garch(d, p = 1, q = 2, init = rep(0.1, 1 + 2 + 1))
modResults(garch12, "garch(1,2) on diff")
```




```r
garch12$w = without
garch12$forecast = forecast(garch12, garch12$w)
modResults(garch12)
```




```
# plot((nTot-50):nTot,dTot[(nTot-25):nTot], type = 'l')
# lines((nTot-without):nTot, garch12$forecast[(nTot - without):nTot],
# col='cyan')


plotForecast = function(models) {
    c = viridis_pal(option = "D")(length(models))

    for (i in 1:length(models)) {
        nTot = length(models[[i]]$forecast)
        w = models[[i]]$w
        lines((nTot - without):nTot, models[[i]]$forecast[(nTot - without):nTot],
            col = c[i], lty = 2)
    }
}
```

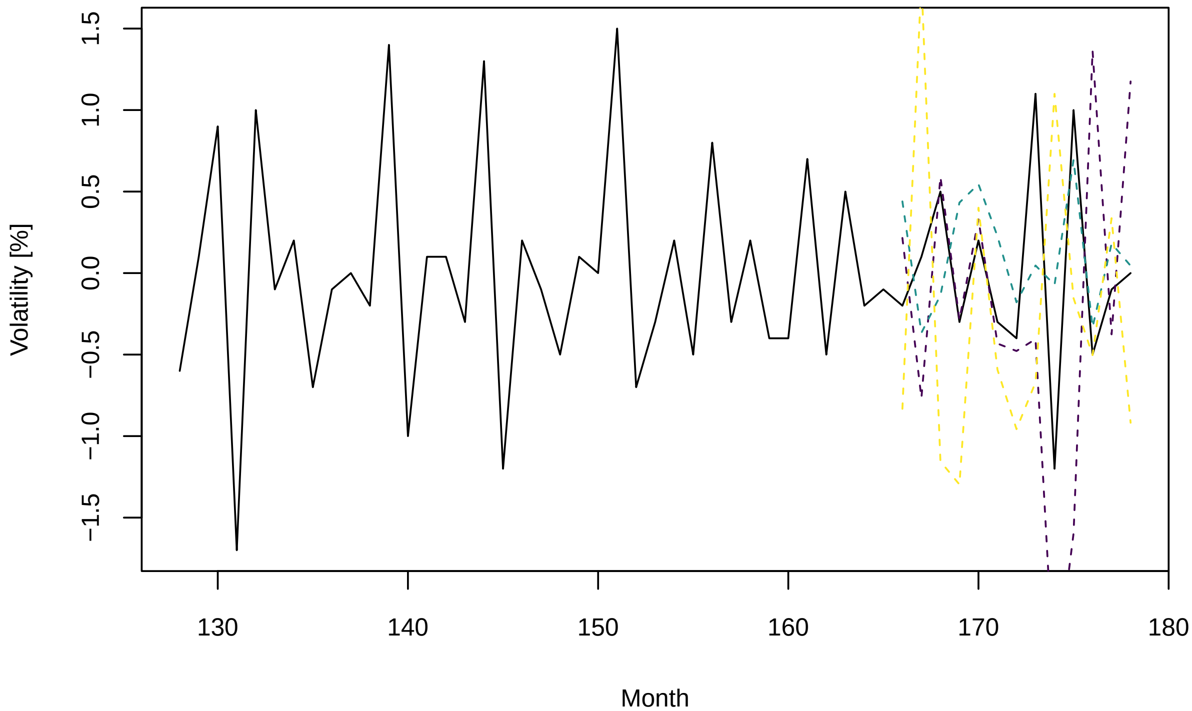
Run all with forecast

```
set.seed(420)
without = 12
dTot = diff(df$Inflation)
nTot = length(dTot)
```

```r
d = diff(dTot[1:(nTot - without)])


arch1df = Garch(d, p = 1, q = 0, init = rep(0.1, 2))

arch2df = Garch(d, p = 2, q = 0, init = rep(0.1, 3))

arch10df = Garch(d, p = 10, q = 0, init = rep(0.1, 11))

modelsf = list(arch1df = arch1df, arch2df = arch2df, arch10df = arch10df)
modelsf$arch1df$w = 12
modelsf$arch1df$w
```

```
## [1] 12
```

```r

# for (mod in modelsf){
for (i in 1:length(modelsf)) {
    # browser()
    w = 12
    modelsf[[i]]$w = w
    # mod$w = w
    modelsf[[i]]$forecast = forecast(modelsf[[i]], w)
}

back = 50
plot((nTot - back):nTot, dTot[(nTot - back):nTot], type = "l", xlab = "Month", ylab = "Volatility [%]",
    cex.main = 0.8, cex.axis = 0.8, cex.lab = 0.8)
plotForecast(modelsf)
```




```
pdf(file = paste0(figPath, "garchForecasts.pdf"), height = 4)
plot((nTot - back):nTot, dTot[(nTot - back):nTot], type = "l", xlab = "Month", ylab = "Volatility [%]",
    cex.main = 0.8, cex.axis = 0.8, cex.lab = 0.8)
plotForecast(modelsf)
dev.off()
```

```
## pdf
##   2
```