Title

code - courseName

Christian Oppegård Moen

dd-mm-yyyy

Contents

```r
source("chrisFunctions.r")
library(ggplot2)
```

GARCH

GARCH(p, q) is given by

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

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

It can be shown that a GARCH(p, q) admits a non-Gaussian ARMA(p, q) for the squared process

$$r_t^2 \tag{3}$$

acf and pacf

For fun we consider the simple acf and pacf of the difference

$$\nabla x_t = x_t - x_{t-1}$$

```r
d = diff(df$Inflation)
par(mfrow = c(2, 2))
dacf = sampleAcf(d, 22)
dacf2 = acf(d)

dpacf = samplePacf(d, 22)
dpacf2 = pacf(d)
```

Now let's look at the return

$$\nabla r_t = \frac{x_t - x_{t-1}}{x_{t-1}}. \tag{4}$$

Problem: Some values are zero. Let's make sure that they are not. (Should not matter since we are looking at variance, right?)

```r
n = dim(df)[1]
shift = min(df$Inflation - 0.1)
inflationShifted = df$Inflation - shift
r = diff(inflationShifted)/inflationShifted[1:(n - 1)]
par(mfrow = c(1, 2))
dacf = sampleAcf(r, 22)

dpacf = samplePacf(r, 22)
```

Finally, let's consider the squared return, which is supposed to be ARMA(p,q)

```r
par(mfrow = c(1, 2))
dacf = sampleAcf(r^2, 22)

dpacf = samplePacf(r^2, 22)  # Verified with pacf fnc.
```

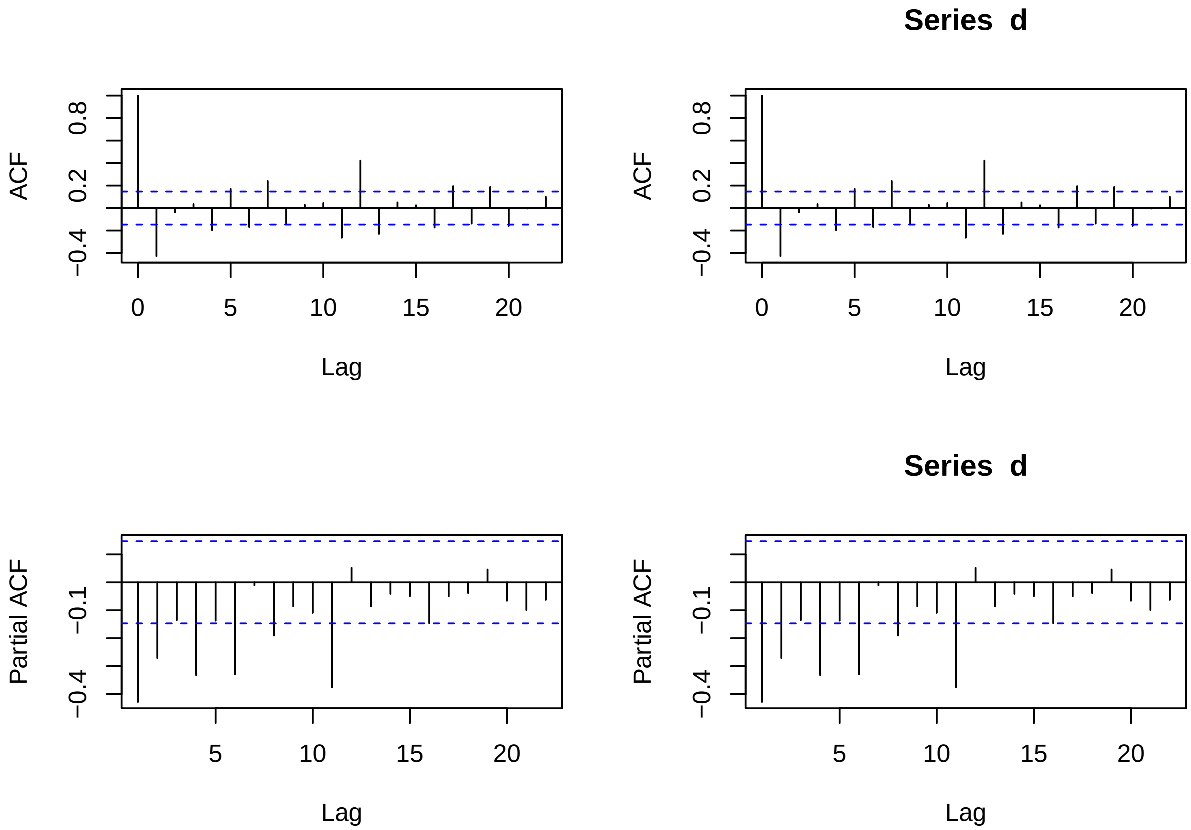


Figure 1: Manual to the left and built-in to the right

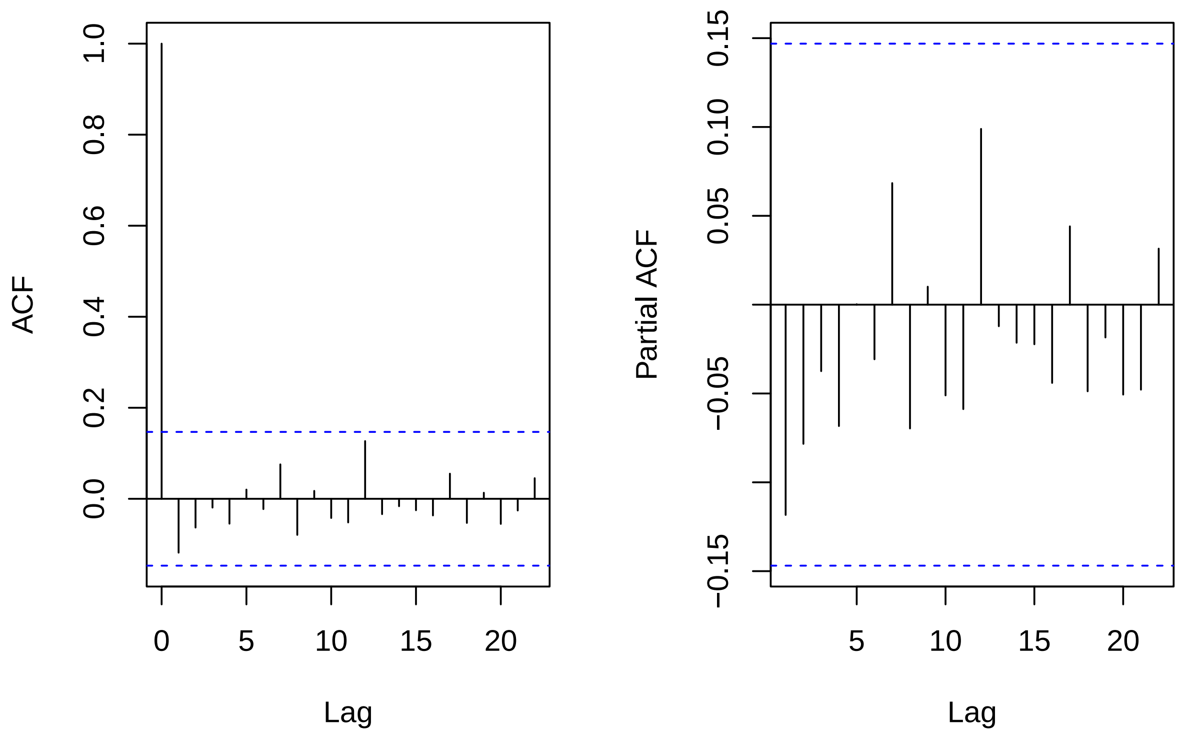


Figure 2: Return

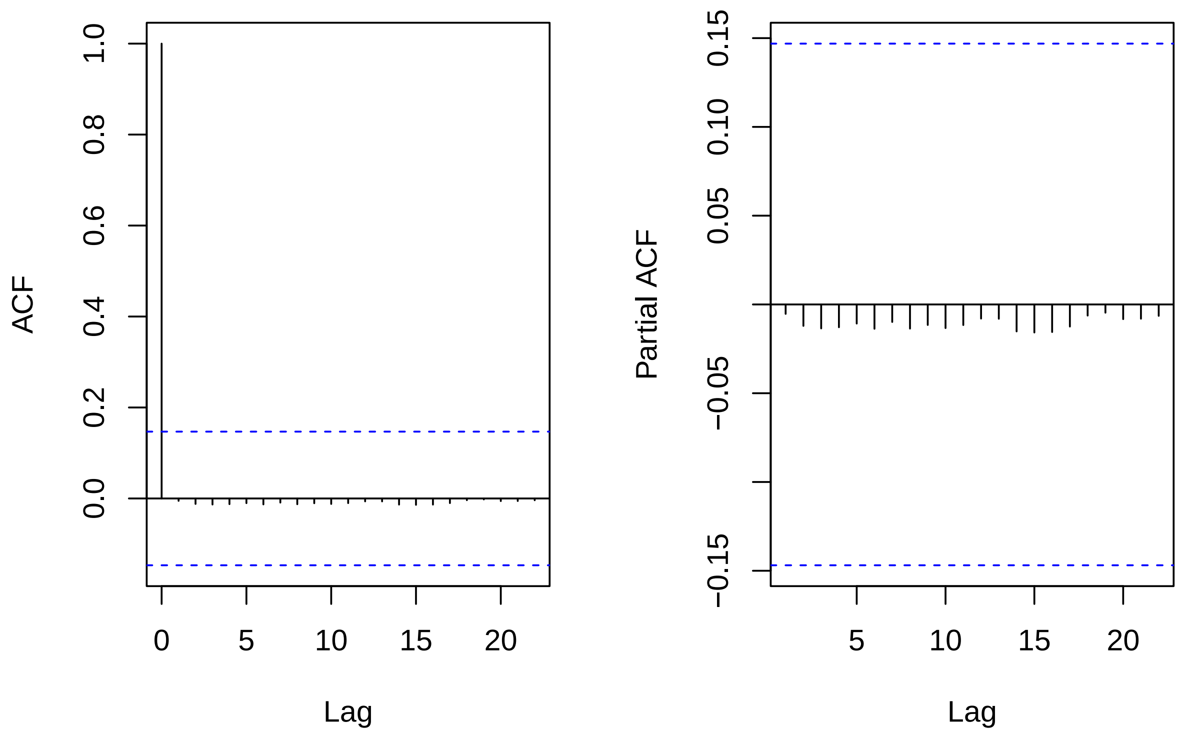


Figure 3: Squared return

Table 3.1. Behavior of the ACF and PACF for ARMA models

	AR(p)	MA(q)	ARMA(p,q)
ACF	Tails off	Cuts off after lag q	Tails off
PACF	Cuts off after lag p	Tails off	Tails off

(5)

GARCH(p,q) Implementation

Definitions for notational purposes:

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

It can be shown that $r_t|\mathcal{R}_2 \sim \mathcal{N}(0, \sigma_t^2)$, $\mathcal{R}_2 = (r_t, ..., r_{t+max(p,q)})$. Because of the normality of the conditional return we can find MLE of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ parameters. That is, the likelihood to be maximized is given by

$$\begin{aligned} L(\boldsymbol{\alpha}, \boldsymbol{\beta}|\mathcal{R}_1) &= \prod_{t=\max(p,q)+1}^{n} f_{\boldsymbol{\alpha},\boldsymbol{\beta}}(r_t|\mathcal{R}_2) \\ &= \prod_{t=\max(p,q)+1}^{n} (2\pi\sigma_t^2)^{-1/2} \exp(-\frac{1}{2}\frac{r_t^2}{\sigma_t^2}), \end{aligned} \tag{6}$$

where $\mathcal{R}_1 = (r_1, ..., r_{max(p,q)})$ and σ_t^2 is given by equation (2). We define the criterion function l to be minimized as proportional to $-\ln L$, such that

$$\begin{aligned} l(\boldsymbol{\alpha}, \boldsymbol{\beta}|\mathcal{R}_1) &= \sum_{t=\max(p,q)+1}^{n} \ln(\sigma_t^2) + \frac{r_t^2}{\sigma_t^2} \\ &= \sum_{t=\max(p,q)+1}^{n} \ln(\alpha_0 + \sum_{j=1}^{p} \alpha_j r_{t-j}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2) + \frac{r_t^2}{\alpha_0 + \sum_{j=1}^{p} \alpha_j r_{t-j}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2} \\ &= \sum_{t=\max(p,q)+1}^{n} \ln(\boldsymbol{\alpha}\boldsymbol{r}_{tp} + \boldsymbol{\beta}\boldsymbol{\sigma}_{tq}^2) + \frac{r_t^2}{\boldsymbol{\alpha}\boldsymbol{r}_{tp} + \boldsymbol{\beta}\boldsymbol{\sigma}_{tq}^2}, \end{aligned} \tag{7}$$

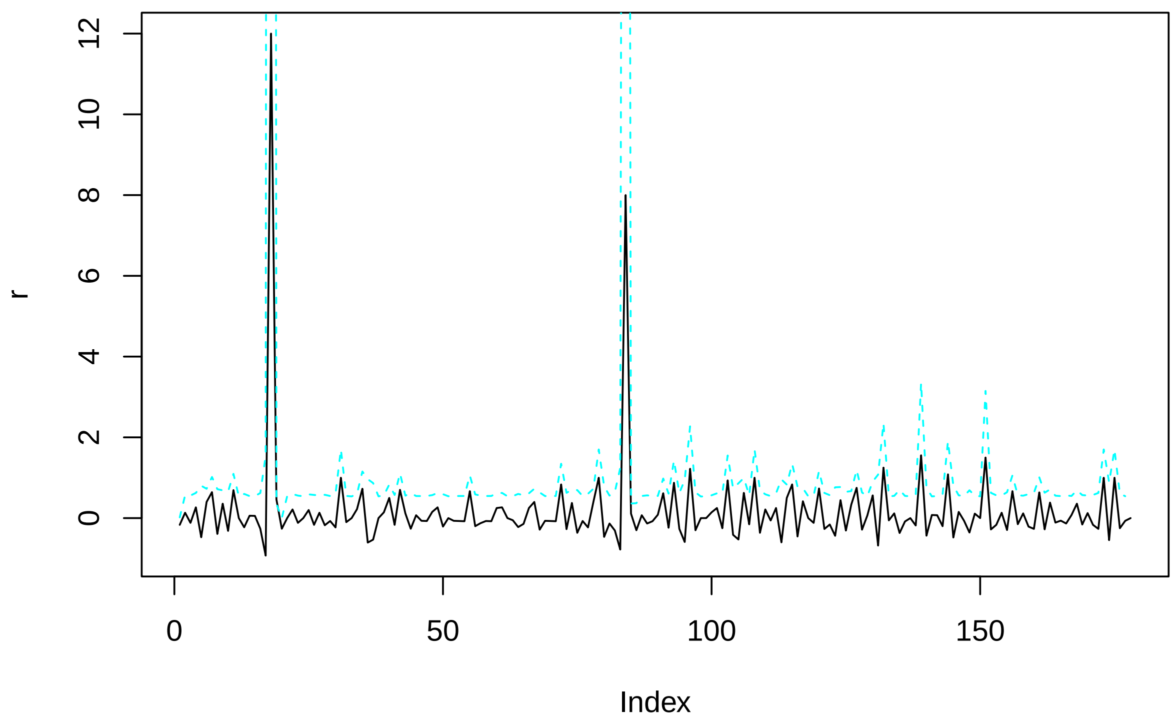
where $\boldsymbol{r}_{tp} = (1, r_{t-1}, ..., r_{t-p})^\top$ and $\sigma^2_{tq} = (\sigma^2_{t-1}, ..., \sigma^p_{t-q})^\top$

```r
critFunc = function(params, r, p = 1, q = 1) {
    # LogLike of alpha and beta given the first r_t values t=(1,...,
    # max(p,q))(Eq. 5.46) Defaults to GARCH(1,1) and uses only the first three
    # params (alpha0, alpha1, beta1) params: vector of parameters. First p+1
    # are alphas, rest are betas r: vector of returns (Eq. 5.34) p,q:
    # GARCH(p,q) model parameters

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

    # If else statements in case it is not 'generalized' (ARCH(p))
    if (p == 0) {
        # Prolly not needed, never gonna remove the AR part?
        alpha = c(0)
        p = 1
    } else {
        alpha = params[1:(p + 1)]
    }
    if (q == 0) {
        beta = c(0)
        q = 1
    } else {
        beta = params[(p + 2):(1 + p + q)]
    }

    # Initialize first m conditional variances.  TODO: This is not correct
    # initialization of the first sigma[1:m], but it works for now.
    sigma = numeric(n)
    sigma[1:m] = (r[1:m]^2)

    ll = 0  # log likelihood (objective is to minimize this)
    for (t in (m + 1):n) {
        sigma[t] = t(alpha) %*% c(1, r[(t - 1):(t - p)]^2) + t(beta) %*% sigma[(t -
            1):(t - q)]
        ll = ll + log(sigma[t]) + r[t]^2/(sigma[t])
    }
    return(ll)
}

# r shifted ---- Shifting inflation since r (eq. 5.34) cannot have zeroes in
# the denominator.
set.seed(420)  # Why? don't ask me
shift = min(df$Inflation - 0.1)  # inflation seems to be discretized per 0.1
inflationShifted = df$Inflation - shift
r = diff(inflationShifted)/inflationShifted[1:(n - 1)]

# Test garch(1,2) ----
alpha_init = c(0.1, 0.1)
beta_init = c(0.1, 0.1)
p = length(alpha_init) - 1
q = length(beta_init)
loglike = critFunc(c(alpha_init, beta_init), r, p = p, q = q)
opt = optim(par = c(alpha_init, beta_init), fn = critFunc, r = r, p = p, q = q)
```

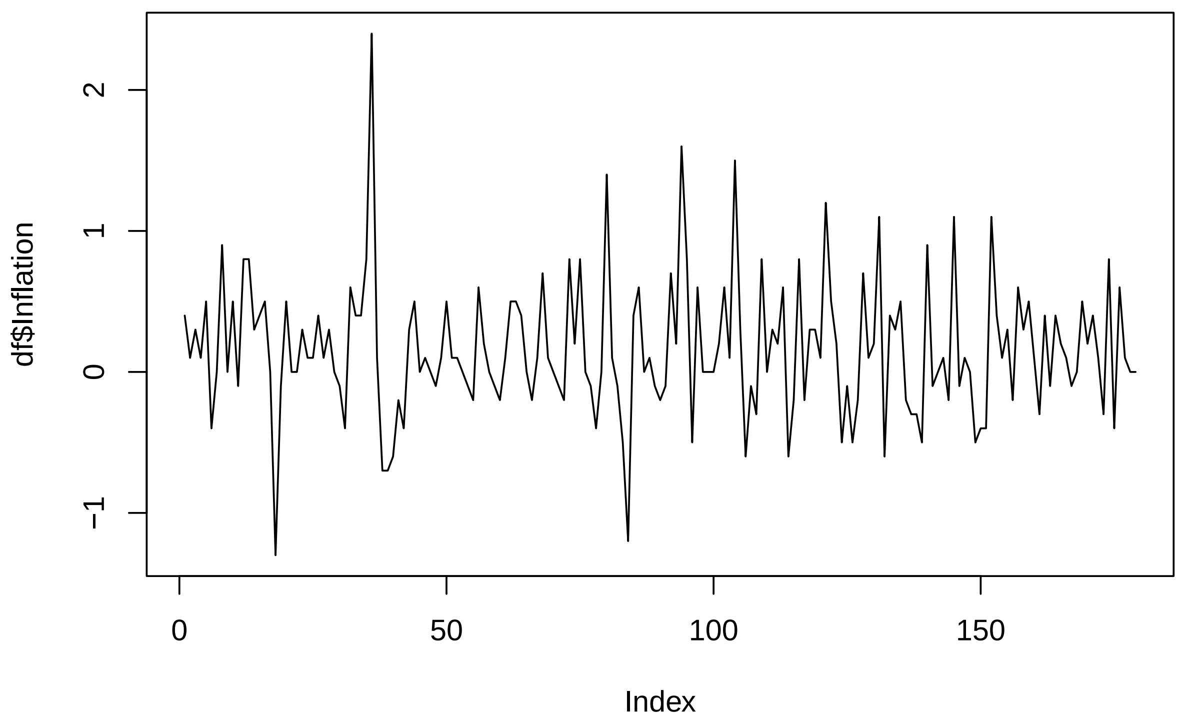
```r
alpha = opt$par[1:(p + 1)]
beta = opt$par[(p + 2):(1 + p + q)]
cbind(alpha, beta)
```

```
##         alpha         beta
## [1,] 0.545009 -0.002592886
## [2,] 1.159176 -0.003728481
```

```r
# Sigma estimation ----
sigmaForecast = function(alpha, beta, r) {
    # One-step-ahead forecasts of the volatility sigma (Eq. 5.52) alpha =
    # (alpha_0,..., alpha_p): Estimated parameters by num. optim.  beta =
    # (beta_1,..., beta_q): Estimated parameters by num. optim.  r: vector of
    # returns.

    p = length(alpha) - 1
    q = length(beta)
    m = max(p, q)

    # Initialize first m conditional variances.  TODO: This is not correct
    # initialization of the first sigma[1:m], but it works for now.
    sigma = numeric(n)
    sigma[1:m] = (r[1:m]^2)
    for (t in (m + 1):n) {
        sigma[t] = t(alpha) %*% c(1, r[(t - 1):(t - p)]^2) + t(beta) %*% sigma[(t -
            1):(t - q)]
    }
    return(sigma)
}

sigma_estim = sigmaForecast(alpha, beta, r)
plot(r, type = "l", )
lines(sigma_estim[2:length(r)], col = "cyan", lty = 2)
```




```r
plot(df$Inflation, type = "l")
```




```
alphas = c(0.3, 0.7)
p = length(alpha) - 1
n = 10000
rs = numeric(n)  # r sample
for (t in (p + 1):n) {
    rs[t] = rnorm(1, sd = sqrt(t(alphas) %*% c(1, rs[(t - 1):(t - p)]^2)))
}
optAlph = optim(par = c(0.1, 0.1), fn = critFunc, r = rs, p = p, q = 0)
```

```
# Set GARCH(2,2) parameters
p <- 2
q <- 2
omega <- 0.01
alpha1 <- 0.2
alpha2 <- 0.1
beta1 <- 0.3
beta2 <- 0.2

# Set the length of the time series
n <- 10000

# Initialize vectors to store simulated data
returns <- numeric(n)
conditional_variances <- numeric(n)
```

```r
# Set initial values
returns[1:2] <- rnorm(2)
conditional_variances[1:2] <- omega/(1 - alpha1 - alpha2 - beta1 - beta2)  # Ensure that it's a valid G

# Simulate GARCH(2,2) process
for (i in 3:n) {
    epsilon <- rnorm(1)
    conditional_variances[i] <- omega + alpha1 * returns[i - 1]^2 + alpha2 * returns[i -
        2]^2 + beta1 * conditional_variances[i - 1] + beta2 * conditional_variances[i -
        2]
    returns[i] <- epsilon * sqrt(conditional_variances[i])
}


initParams = c(0.1, 0.1, 0.1, 0.1, 0.1)
params = optim(par = c(0.1, 0.1, 0.1, 0.1, 0.1), fn = critFunc, r = returns, p = 2,
    q = 2, method = "BFGS")$par
round(rbind(real = c(omega, alpha1, alpha2, beta1, beta2), estim = params, init = initParams),
    2)
```

```
##       [,1] [,2] [,3]  [,4] [,5]
## real  0.01  0.2 0.10  0.30 0.20
## estim 0.01  0.2 0.16 -0.09 0.41
## init  0.10  0.1 0.10  0.10 0.10
```

Regression and GARCH

What if

$$x_t = \boldsymbol{c}^\top \boldsymbol{z}_t + y_t \tag{8}$$

where y_t is modelled as a GARCH, $\boldsymbol{z}_t$ are covariates and $\boldsymbol{c}$ is a vector of coeffs. to be estimated.

```r
fit = lm(Inflation ~ Unemployed + Consumption + InterestRate, data = df)
coeffs = fit$coefficients
coeffs
```

```
##   (Intercept)    Unemployed   Consumption  InterestRate
##  2.605176e-01 -1.815662e-06 -5.993765e-02  1.171267e-02
```

```r
preds = 0
```