Read training data

```
options(repos = c(CRAN = "https://cloud.r-project.org"))
install.packages("ggplot2")
```

```
##
## The downloaded binary packages are in
##  /var/folders/8c/7zdp5y6j3fbbjxh65ghw770w0000gn/T//Rtmpb8G9Xd/downloaded_packages
```

```
library(ggplot2)
str(D)
```

```
## function (expr, name)
```

```
D <- read.csv("DST_BIL54.csv")
```

See the help

```
?strftime
D$time <- as.POSIXct(paste0(D$time,"-01"), "%Y-%m-%d", tz="UTC")
D$time
```

```
##  [1] "2018-01-01 UTC" "2018-02-01 UTC" "2018-03-01 UTC" "2018-04-01 UTC"
##  [5] "2018-05-01 UTC" "2018-06-01 UTC" "2018-07-01 UTC" "2018-08-01 UTC"
##  [9] "2018-09-01 UTC" "2018-10-01 UTC" "2018-11-01 UTC" "2018-12-01 UTC"
## [13] "2019-01-01 UTC" "2019-02-01 UTC" "2019-03-01 UTC" "2019-04-01 UTC"
## [17] "2019-05-01 UTC" "2019-06-01 UTC" "2019-07-01 UTC" "2019-08-01 UTC"
## [21] "2019-09-01 UTC" "2019-10-01 UTC" "2019-11-01 UTC" "2019-12-01 UTC"
## [25] "2020-01-01 UTC" "2020-02-01 UTC" "2020-03-01 UTC" "2020-04-01 UTC"
## [29] "2020-05-01 UTC" "2020-06-01 UTC" "2020-07-01 UTC" "2020-08-01 UTC"
## [33] "2020-09-01 UTC" "2020-10-01 UTC" "2020-11-01 UTC" "2020-12-01 UTC"
## [37] "2021-01-01 UTC" "2021-02-01 UTC" "2021-03-01 UTC" "2021-04-01 UTC"
## [41] "2021-05-01 UTC" "2021-06-01 UTC" "2021-07-01 UTC" "2021-08-01 UTC"
## [45] "2021-09-01 UTC" "2021-10-01 UTC" "2021-11-01 UTC" "2021-12-01 UTC"
## [49] "2022-01-01 UTC" "2022-02-01 UTC" "2022-03-01 UTC" "2022-04-01 UTC"
## [53] "2022-05-01 UTC" "2022-06-01 UTC" "2022-07-01 UTC" "2022-08-01 UTC"
## [57] "2022-09-01 UTC" "2022-10-01 UTC" "2022-11-01 UTC" "2022-12-01 UTC"
## [61] "2023-01-01 UTC" "2023-02-01 UTC" "2023-03-01 UTC" "2023-04-01 UTC"
## [65] "2023-05-01 UTC" "2023-06-01 UTC" "2023-07-01 UTC" "2023-08-01 UTC"
## [69] "2023-09-01 UTC" "2023-10-01 UTC" "2023-11-01 UTC" "2023-12-01 UTC"
## [73] "2024-01-01 UTC" "2024-02-01 UTC" "2024-03-01 UTC" "2024-04-01 UTC"
## [77] "2024-05-01 UTC" "2024-06-01 UTC" "2024-07-01 UTC" "2024-08-01 UTC"
## [81] "2024-09-01 UTC" "2024-10-01 UTC" "2024-11-01 UTC" "2024-12-01 UTC"
```

```
class(D$time)
```

```
## [1] "POSIXct" "POSIXt"
```

```r
## Year to month for each of them
D$year <- 1900 + as.POSIXlt(D$time)$year + as.POSIXlt(D$time)$mon / 12

## Make the output variable a floating point (i.e.\ decimal number)
D$total <- as.numeric(D$total) / 1E6

## Divide intro train and test set
teststart <- as.POSIXct("2024-01-01", tz="UTC")
Dtrain <- D[D$time < teststart, ]
Dtest <- D[D$time >= teststart, ]
```

4.2 Implement the update equations in a for-loop in a computer. Calculate the ˆ up to time t = 3. Present the values and comment: Do you think it is intuitive to understand the details in the matrix calculations? If yes, give a short explanaition.

```r
N <- 3
R_0 <- matrix(8)
theta_0 <- matrix(1)

R_t_1 <- matrix(1)%*%t(matrix(1))
theta_t_1 <- theta_0 + matrix(R_t_1)^-1%*%matrix(1)%*%(Dtrain$total[1] - t(matrix(1))%*%theta_0)
theta_N <- c(theta_0, theta_t_1)

for (t in 2:N){
  R_t_1 <- R_t_1 + matrix(t^2)
  theta_t_1 <- theta_t_1 + (R_t_1^-1)*t*(Dtrain$total[t] - t * theta_t_1)
  theta_N <- c(theta_N, theta_t_1)
}
theta_N
```

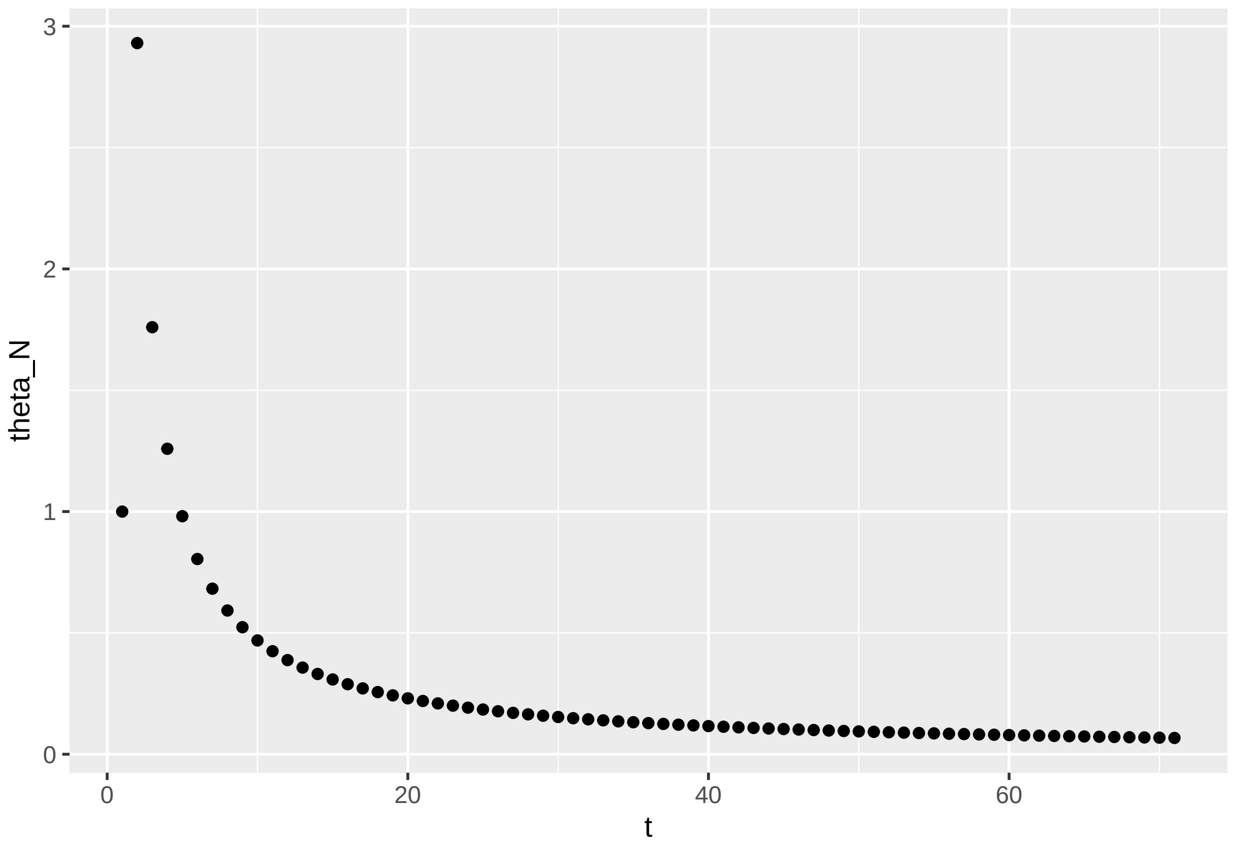
```
## [1] 1.000000 2.930483 1.759714 1.258774
```

Theta_t for t = 3 is 1.258774. I do not find it intuitive to understand the details in the matrix calculations. From what I do understand, R_t is the squared sum of the values for each field until t(in this case one field). As far as theta_t is concerned I don't understand it at all.

#4.3 Calculate the estimates of ˆN and compare them to the OLS estimates of , are they close? Can you find a way to decrease the difference by modifying some initial values and explain why initial values are important to get right

```r
N <- 70
R_0 <- matrix(8)
theta_0 <- matrix(1)

R_t_1 <- matrix(1)%*%t(matrix(1))
theta_t_1 <- theta_0 + matrix(R_t_1)^-1%*%matrix(1)%*%(Dtrain$total[1] - t(matrix(1))%*%theta_0)
theta_N <- c(theta_0, theta_t_1)
```

```r
for (t in 2:N){
  R_t_1 <- R_t_1 + matrix(t^2)
  theta_t_1 <- theta_t_1 + (R_t_1^-1)*t*(Dtrain$total[t] - t * theta_t_1)
  theta_N <- c(theta_N, theta_t_1)
}
theta_N_plot <- data.frame(theta_N=theta_N, t=1:length(theta_N))
ggplot(theta_N_plot) +
  geom_point(aes(x=t, y=theta_N))
```



NOTE: I NEED OLS ESTIMATES IN ORDER TO PROPERLY ANSWER THIS

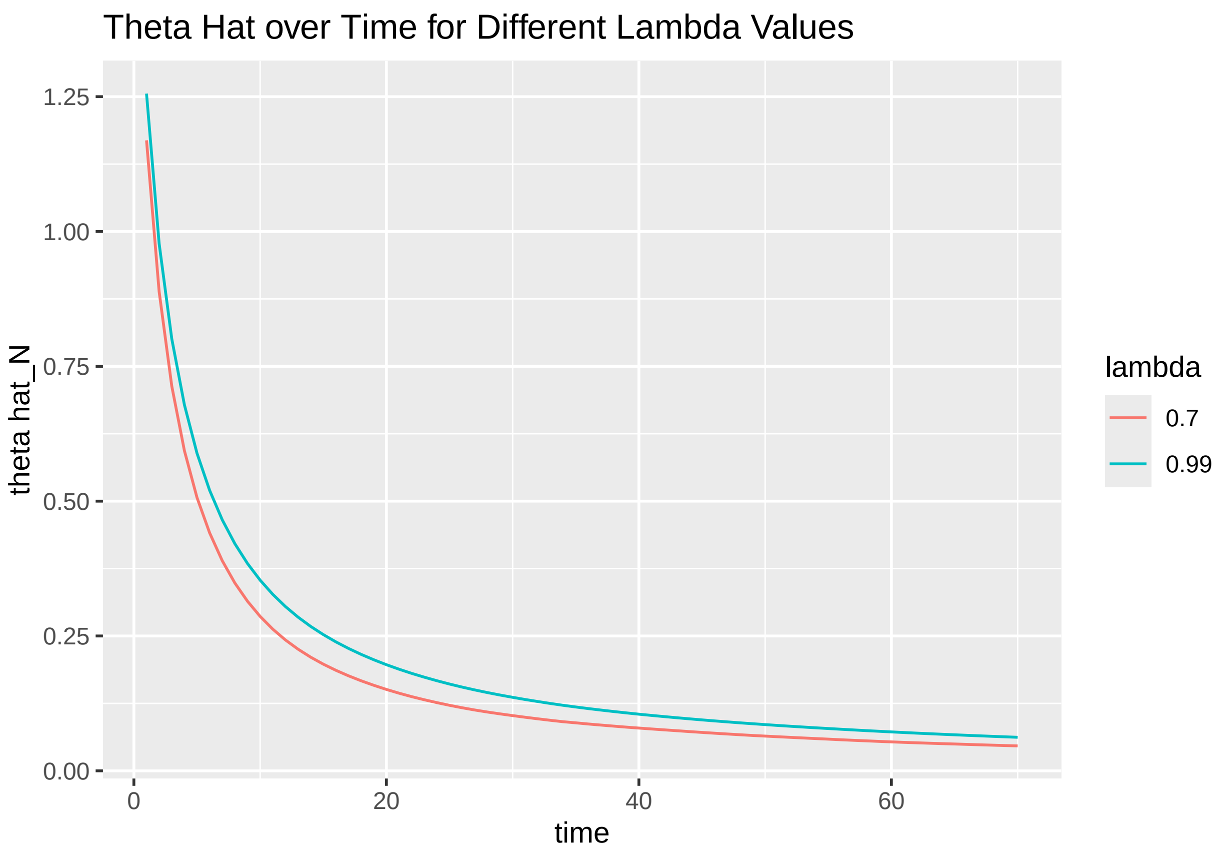
#4.4 Now implement RLS with forgetting (you just have to multiply with at one position in the Rt update).

Calculate the parameter estimates: ˆ1,t and ˆ2,t, for t = 1, . . . , N first with = 0.7 and then with = 0.99. Provide a plot for each parameter. In each plot include the estimates with both values (a line for each). Comment on the plots.

You might want to remove the first few time points, they are what is called a "burn-in" period for a recursive estimation).

Tip: It can be advantageous to put the loop in a function, such that you don't repeat the code too much! (it's generally always a good idea to use functions, as soon as you need to run the same code more than once).

```r
N <- 72
R_0 <- matrix(1)
theta_0 <- matrix(1)

R_t_1 <- matrix(1)%*%t(matrix(1))
theta_t_1 <- theta_0 + matrix(R_t_1)^-1%*%matrix(1)%*%(Dtrain$total[1] - t(matrix(1))%*%theta_0)
theta_N_forget_7 <- c(theta_0, theta_t_1)

for (t in 2:N){
  R_t_1 <- 0.7 * R_t_1 + matrix(t^2)
  theta_t_1 <- theta_t_1 + (R_t_1^-1)*t*(Dtrain$total[t] - t * theta_t_1)
  theta_N_forget_7 <- c(theta_N_forget_7, theta_t_1)
}

##########################################################################
N <- 72
R_0 <- matrix(1)
theta_0 <- matrix(1)

R_t_1 <- matrix(1)%*%t(matrix(1))
theta_t_1 <- theta_0 + matrix(R_t_1)^-1%*%matrix(1)%*%(Dtrain$total[1] - t(matrix(1))%*%theta_0)
theta_N_forget_99 <- c(theta_0, theta_t_1)

for (t in 2:N){
  R_t_1 <- 0.99 * R_t_1 + matrix(t^2)
  theta_t_1 <- theta_t_1 + (R_t_1^-1)*t*(Dtrain$total[t] - t * theta_t_1)
  theta_N_forget_99 <- c(theta_N_forget_99, theta_t_1)
}
theta_N_plot = data.frame(x=rep(1:70, 2),
                          y=c(theta_N_forget_7[4:73], theta_N_forget_99[4:73]),
                          lambda=c(rep("0.7", 70), rep("0.99", 70)))
ggplot(data=theta_N_plot, aes(x=x, y=y, color=lambda)) +
  geom_line() +
  labs(x = "time", y = "theta hat_N", title = "Theta Hat over Time for Different Lambda Values")
```



The plots are largely identical. It appears that at a lower lambda (0.7) theta hat of N is smaller but at the larger lambda (0.99), theta hat of N is larger Both lines follow the same overall trend and shape, however the lower lambda value has a slightly more extreme drop/converges faster to its asymptote.

You might want to compare the estimates for t = N with the WLS estimates for the same values. Are they equal?

I CANNOT DO THIS WITHOUT THE WLS ESTIMATES FOR THE LAMBDA VALUES

4.5. Make one-step predictions

The notation t + 1|t means the variable one-step ahead, i.e. at time t + 1, given information available at time t. So this notation is used to denote predictions. For xt+1|t we do have the values ahead in time for a trend model – in most other situations we must use forecasts of the model inputs.

```r
lambda <- 0.7
X <- cbind(1, c(1:72))
y <- cbind(Dtrain$total)

n <- length(X[,1])
Theta <- matrix(NA, nrow=n, ncol=2)
OneStepPred_7 <- matrix(NA, nrow=n)
```

```r
# 1st step
x1 <- X[1,]

R_1 <- x1%*%t(x1)
h_1 <- x1*y[1]

# 2nd step
x2 <- X[2,]
R_2 <- lambda*R_1 + x2%*%t(x2)
h_2 <- lambda*h_1 + x2*y[2]

solve(R_2)
```

```
##           [,1]      [,2]
## [1,]  6.714286 -3.857143
## [2,] -3.857143  2.428571
```

```r
# estimation time
Theta[2,] <- solve(R_2) %*% h_2
OneStepPred_7[2+1] <- X[2+1,]%*%Theta[2,]

# 3rd step - first time using the update formula
x3 <- X[3,]
R_3 <- lambda*R_2 + x3%*%t(x3)
Theta[3,] <- Theta[2,] + solve(R_3)%*%x3%*%(y[3] - t(x3)%*%Theta[2,])

# predicting one step ahead
OneStepPred_7[3+1] <- X[3+1,]%*%Theta[3,]

# now we loop
R <- R_3
for (i in 4:n){
  x <- X[i,]
  # Update
  R <- lambda*R + x %*% t(x)
  Theta[i,] <- Theta[i-1,] + solve(R) %*% x %*% (y[i] - t(x) %*% Theta[i-1,])
}

# predict
for(i in 4:n - 1){
  OneStepPred_7[i+1] <- X[i+1,]%*%Theta[i,]
}
```

```r
#################################### Now with 0.99 ##############################
lambda <- 0.99
X <- cbind(1, c(1:72))
y <- cbind(Dtrain$total)

n <- length(X[,1])
Theta <- matrix(NA, nrow=n, ncol=2)
OneStepPred_99 <- matrix(NA, nrow=n)
```

```r
# 1st step
x1 <- X[1,]

R_1 <- x1%*%t(x1)
h_1 <- x1*y[1]

# 2nd step
x2 <- X[2,]
R_2 <- lambda*R_1 + x2%*%t(x2)
h_2 <- lambda*h_1 + x2*y[2]

solve(R_2)
```

```
##            [,1]      [,2]
## [1,]  5.040404 -3.020202
## [2,] -3.020202  2.010101
```

```r
# estimation time
Theta[2,] <- solve(R_2) %*% h_2
OneStepPred_99[2+1] <- X[2+1,]%*%Theta[2,]

# 3rd step - first time using the update formula
x3 <- X[3,]
R_3 <- lambda*R_2 + x3%*%t(x3)
Theta[3,] <- Theta[2,] + solve(R_3)%*%x3%*%(y[3] - t(x3)%*%Theta[2,])

# predicting one step ahead
OneStepPred_99[3+1] <- X[3+1,]%*%Theta[3,]

# now we loop
R <- R_3
for (i in 4:n){
  x <- X[i,]
  # Update
  R <- lambda*R + x %*% t(x)
  Theta[i,] <- Theta[i-1,] + solve(R) %*% x %*% (y[i] - t(x) %*% Theta[i-1,])
}

# predict
for(i in 4:n - 1){
  OneStepPred_99[i+1] <- X[i+1,]%*%Theta[i,]
}
```

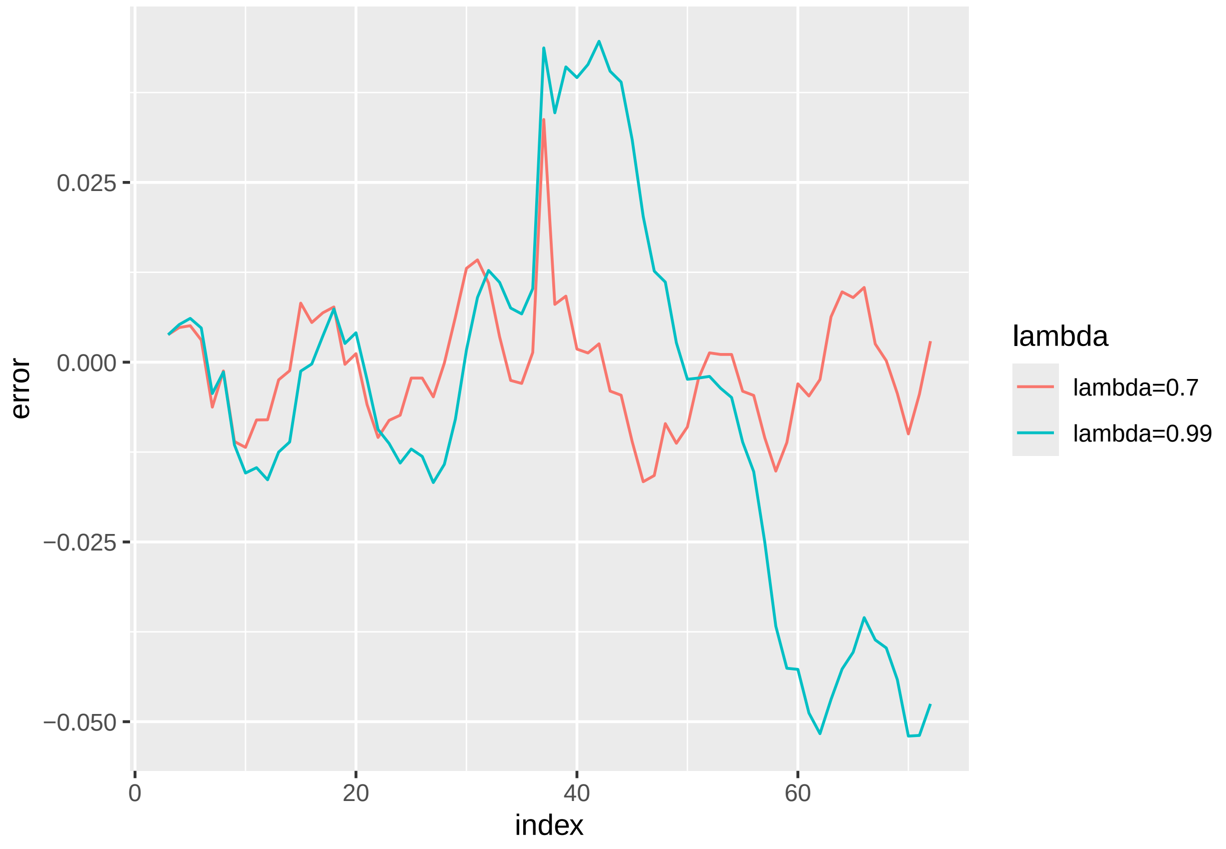
Now calculate the one-step ahead residuals

ˆt|t−1 = yˆt|t−1 − yt−1

note the shift "t + 1|t" to "t|t − 1": both means one-step ahead.

Plot them for t = 5,...,N first with = 0.7 and then = 0.99 (note, we remove a burn-in period (t = 1, . . . , 4), might not be necessary, but usually a good idea when doing recursive estimation – depends on the initialization values).

```
error_7 <- Dtrain$total[3:length(Dtrain$total)] - OneStepPred_7[3:nrow(OneStepPred_7)]
error_99 <- Dtrain$total[3:length(Dtrain$total)] - OneStepPred_99[3:nrow(OneStepPred_99)]
to_plot <- data.frame(error=c(error_7, error_99), index=rep(c(3:length(Dtrain$total)), 2), lambda=rep(c

ggplot(data=to_plot, aes(x=index, y=error, color=lambda)) +
  geom_line()
```



Comment on the residuals, e.g. how do they vary over time?

The risiduals appear to vary more wildly when lambda=0.99. In particular after index 45, the residuals increase in magnitude a lot in comparison to when lambda=0.7, which appears more stable. This is likely due to the sharp change in direction the graph takes around that time. When lambda is lower, the predictions are a lot more influenced by recent points, and thus can more quickly respond to sudden trend changes. On the other end, the large values of lambda tend to be more resistant to change and thus do not respond quickly to sudden trend changes.

4.6. Optimize the forgetting for the horizons k = 1, . . . , 12. First calculate the k-step residuals

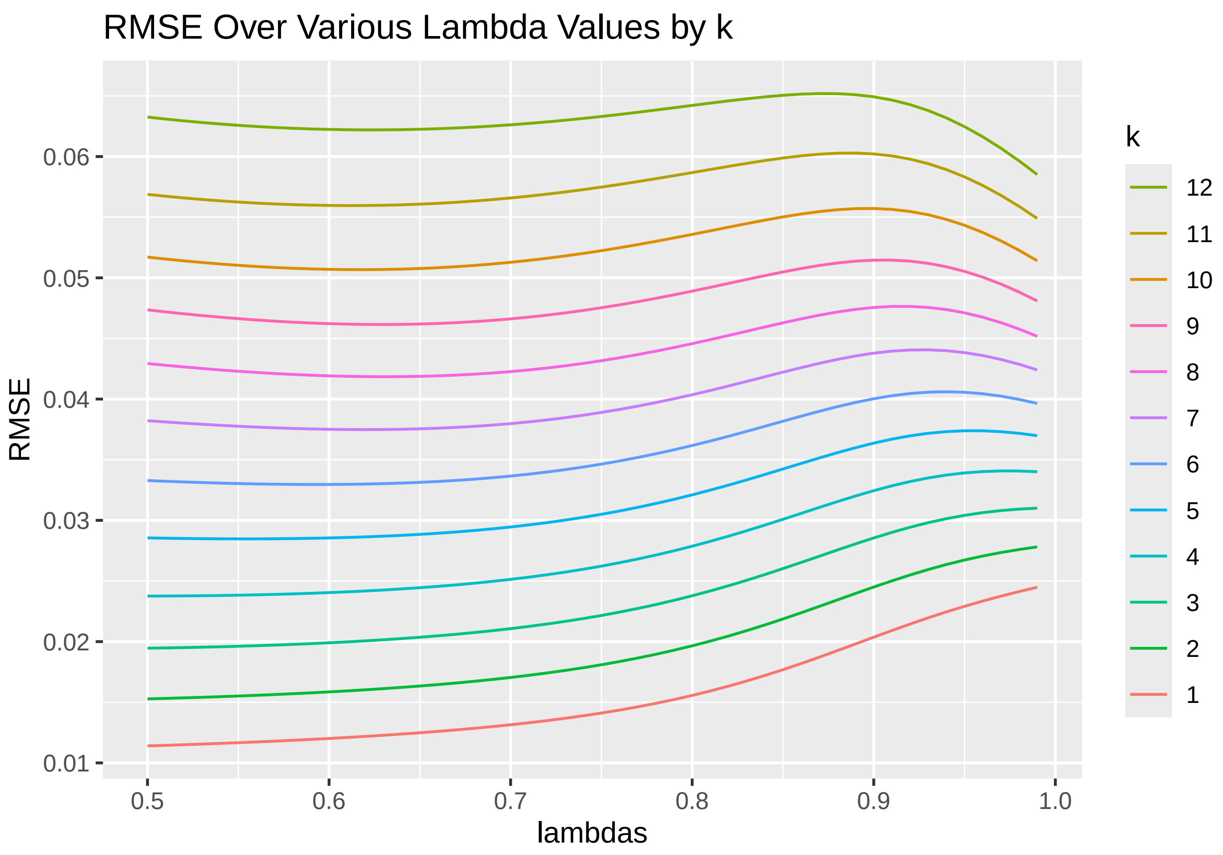
```r
k_step_residuals <- function(Dtrain, lambda){
  X <- cbind(1, c(1:72))
  y <- cbind(Dtrain$total)

  n <- length(X[,1])
  Theta <- matrix(NA, nrow=n, ncol=2)
  # create error matrix where the row corresponds to t and each column corresponds to k
  KStepPred <- matrix(NA, nrow=n - 12, ncol=12)

  # 1st step
  x1 <- X[1,]

  R_1 <- x1%*%t(x1)
  h_1 <- x1*y[1]

  # 2nd step
  x2 <- X[2,]
  R_2 <- lambda*R_1 + x2%*%t(x2)
  h_2 <- lambda*h_1 + x2*y[2]

  solve(R_2)

  # estimation time
  Theta[2,] <- solve(R_2) %*% h_2
  for (k in 1:12){
    KStepPred[3, k] <- X[2+k,]%*%Theta[2,]
  }

  # now we loop
  R <- R_2
  for (i in 3:(length(Dtrain$total) - 13)){
    x <- X[i,]
    # Update
    R <- lambda*R + x %*% t(x)
    Theta[i,] <- Theta[i-1,] + solve(R) %*% x %*% (y[i] - t(x) %*% Theta[i-1,])
    for (k in 1:12){
      KStepPred[i + 1, k] <- X[i+k,]%*%Theta[i,]
    }
  }

  # now we need to get the difference betwen the predicted values and the actual values
  KStepPred <- KStepPred[3:nrow(KStepPred),]
  KStepResiduals <- KStepPred
  for (k in 1:12){
    y_actual <- Dtrain$total[(3+k):(n - 12 + k)]
    KStepResiduals[,k] <- y_actual - KStepPred[,k]
  }
  return(KStepResiduals)
}
```

then calculate the k-step Root Mean Square Error (RMSEk)

```
RMSE_k <- function(KStepResiduals, k){
  frac <- 1/(nrow(KStepResiduals) - k)
  squared_sum <- t(KStepResiduals[,k])%*%KStepResiduals[,k]
  return(sqrt(frac * squared_sum))
}

KStepResiduals <- k_step_residuals(Dtrain, lambda)      # calculating residuals for all t for all k
# calculating RMSE for all k
RMSE <- matrix(NA, nrow=1, ncol=12)
for (k in 1:12){
  RMSE[1,k] <- RMSE_k(KStepResiduals, k)
}
```

Do this for a sequence of values (e.g. 0.5,0.51,. . . ,0.99) and make a plot

```
lambdas <- seq(0.5, 0.99, by=0.01)
to_plot <- data.frame(lambdas=rep(lambdas, each=12), RMSE=rep(NA, (length(lambdas) * 12)), k=as.characte
i <- 1
for (l in lambdas){
  # all the math n
  KStepResiduals <- k_step_residuals(Dtrain, l)
  for (k in 1:12){
    to_plot$RMSE[i] <- RMSE_k(KStepResiduals, k)
    i <- i + 1
  }
}

# plotting
ggplot(to_plot) +
  geom_line(aes(x=lambdas, y=RMSE, color=k)) +
  scale_color_discrete(breaks=rev(c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"))) +
  ggtitle("RMSE Over Various Lambda Values by k")
```



Comment on: Is there a pattern and how would you choose an optimal value of ? Would you let depend on the horizon?

The pattern in the data is difficult to discern, as different values of k rise or fall at different rates for at different lambdas. For example, at $k < 5$, it appears that increasing lambda strictly reduces the accuracy of future predictions insofar as this dataset is concerned. However there are other points at $5 < k < 11$ where setting lambda at ~0.6 yields the lowest RMSE value. Finally, for $k = 11$ & $k = 12$ the lowest RMSE values were held by lambdas at and near 1.0. Therefore we think creating a chart like this is incredibly important, as these are trends that are difficult to see otherwise. It's important when deciding the optimal value of lambda to know how far into the future you are expecting to predict, as lambdas may have drastically different performances depending on how far out we plan on looking as we've demonstrated. We would let lambda depend on the horizon, as the horizon strongly influences the performance of lambda, as demonstrated in our analysis of the graph above.

4.7. Make predictions of the test set using RLS. You can use a single value for all horizons, or choose some way to have different values, and run the RLS, for each horizon.

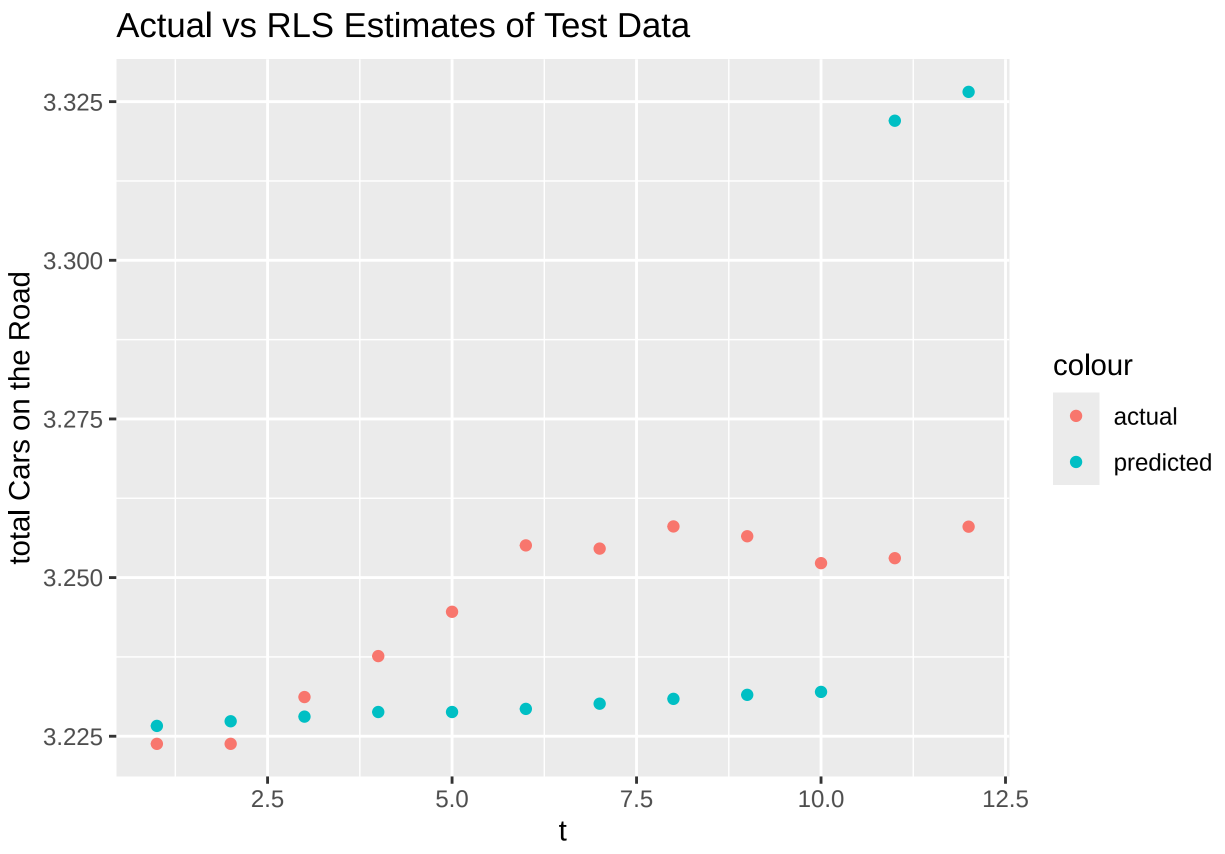
```
findBestLambdas <- function(df){
  best_lambdas <- data.frame(lambdas=rep(NA, 12))
  for (k in 1:12){
    df_k <- df[df$k == k,]
```

```r
    min_RMSE <- min(df_k$RMSE)
    best_lambda <- df_k[df_k$RMSE == min_RMSE, 1]
    best_lambdas$lambdas[k] <- best_lambda
  }
  return(best_lambdas)
}
best_lambdas <- findBestLambdas(to_plot)
```

```r
k_step_preds_with_optimal_lambdas <- function(Dtrain, lambdas){
  X <- cbind(1, c(1:(72 + 12)))
  y <- cbind(Dtrain$total)

  n <- length(X[,1])
  Theta <- matrix(NA, nrow=n, ncol=2)
  # create error matrix where the row corresponds to t and each column corresponds to k
  KStepPred <- matrix(NA, nrow=n, ncol=12)

  # 1st step
  x1 <- X[1,]

  R_1 <- x1%*%t(x1)
  h_1 <- x1*y[1]

  # 2nd step
  x2 <- X[2,]
  R_2 <- list()
  h_2 <- list()
  for (k in 1:12){
    R_2[[k]] <- lambdas[k]*R_1 + x2%*%t(x2)
    h_2[[k]] <- lambdas[k]*h_1 + x2*y[2]

    # estimation time
    Theta[2,] <- solve(R_2[[k]]) %*% h_2[[k]]
    KStepPred[3, k] <- X[2+k,]%*%Theta[2,]
  }

  # now we loop
  R <- R_2
  for (k in 1:12){
    for (i in 3:(length(Dtrain$total))){
      x <- X[i,]
    # Update
      R[[k]] <- lambdas[k]*R[[k]] + x %*% t(x)
      Theta[i,] <- Theta[i-1,] + solve(R[[k]]) %*% x %*% (y[i] - t(x) %*% Theta[i-1,])
      KStepPred[i + 1, k] <- X[i+k,]%*%Theta[i,]
    }
  }
  return(KStepPred)
}


KStepPredOptLambda <- k_step_preds_with_optimal_lambdas(Dtrain, best_lambdas$lambdas)
```

Make a plot and compare to the predictions from the other models (OLS and WLS). You can play around a bit, for example make a plot of the 1 to 12 steps forecasts at each time step to see how they behave.

```r
test_pred <- KStepPredOptLambda[73,]
rls_comparison_plot <- data.frame(t=c(1:12), rls_pred=test_pred, actual=Dtest$total)
ggplot(rls_comparison_plot) +
  geom_point(aes(x=t, y=actual, color="actual")) +
  geom_point(aes(x=t, y=rls_pred, color="predicted")) +
  ggtitle("Actual vs RLS Estimates of Test Data") +
  ylab("total Cars on the Road")
```



4.8. Reflexions on time adaptive models - are there pitfalls!?

In our experience time adaptive models can be prone to underfitting. This is largely due to their univariate nature. With only time to use as a factor it's necessarily going to be difficult to make any meaningful predictions when other factors can be influential. ## Consider overfitting vs. underfitting. In this example, we believe time adaptive models overfit, as shown by the last graph in which we tried to fit it only 12 months (1 year) out. While the estimators were good for the first3-4 months, because of the model's inflexibility to move (outside of the last two points), it had difficulty adjusting to the different trend the number of cars on the road followed in the testing data. ## Are there challenges in creating test sets when data depends on time (in contrast to data not dependend on time)? The challenge is largeley in the univariate nature of the data. While there are many techniques at ones' disposal, there's a reason most real world models don't rely solely on time in deterministic models. There are typically more influential parameters on the dependant variable than time. While time cannot be discounted, it's important to gather a more diverse set of data than just time. ## Can recursive estimation and prediction aliviate challenges with test sets for time dependent data? They can aid, but not aliviate the challenges with test sets for time dependent data. As time continues to pass, some trends and fads will die out and spawn, while others will prove their staying power. Recursive estimation and prediction are methods which aim to take those into account whereas an OLS is indiscriminate and insensitive to the way trends change over time. ## Can you come up with other techniques for time adaptive estimation? Honestly no. ## Additional thoughts and comments?