Homework 3: Neo Lee

Introduction to Time Series, Fall 2023

Due Thursday October 5 at 5pm

The total number of points possible for this homework is 38. The number of points for each question is written below, and questions marked as "bonus" are optional. Submit the **knitted html file** from this Rmd to Gradescope.

If you collaborated with anybody for this homework, put their names here:

Regression troubles

1. (5 pts) Suppose that $y \in \mathbb{R}^n$ is a response vector and $X \in \mathbb{R}^{n \times p}$ is a predictor matrix, with $p > n$. Prove that there is at least one $\eta \neq 0$ (not equal to the zero vector) that is in null(X), the null space of X. Prove that if $\tilde{\beta}$ is a least squares solution in the regression of y on X, then any vector of the form

$$\hat{\beta} = \tilde{\beta} + \eta, \quad \text{for } \eta \in \text{null}(X)$$

is also a solution.

SOLUTION GOES HERE

Proof of null(X) contains at least one non-zero vector η:

Since $p > n$, the column vectors are linear dependent. Denote $(v_1, \cdots, v_p)$ as the column vectors of X. Then, there are non trivial coefficients $(c_1, \cdots, c_p)$ such that

$$\sum_{i=1}^{p} c_i v_i = 0.$$

Hence, $\eta = (c_1, \cdots, c_p)$ is a non-zero vector in null(X).

Proof of $\hat{\beta} = \tilde{\beta} + \eta$ is also a least squares solution for $\eta \in \text{null}(X)$:

Denote the prediction from $\tilde{\beta}$ as $\tilde{y} = X\tilde{\beta}$ with MSE $= y - \tilde{y}$. Then the prediction from $\hat{\beta}$

$$\begin{aligned} \hat{y} &= X\hat{\beta} && (1)\\ &= X(\tilde{\beta} + \eta) && (2)\\ &= X\tilde{\beta} + X\eta && (3)\\ &= \tilde{y} + X\eta && (4)\\ &= \tilde{y}. && (5) \end{aligned}$$

Therefore, they have the same MSE. Since $\hat{\beta}$ is a least squares solution, $\tilde{\beta} + \eta$ is also a least squares solution.

2. (6 pts) With X, y as in Q1, suppose that $\tilde{\beta}$ is a least squares solution with $\tilde{\beta}_j > 0$, and suppose that null$(X) \not\perp e_j$, where e_j is the j^{th} standard basis vector (i.e., e_j is a vector with all 0s except for a 1 in the j^{th} component), and recall we write $S \perp v$ for a set S and vector v provided $u^T v = 0$ for all $u \in S$. Prove that there exists another least squares solution $\hat{\beta}$ such that $\hat{\beta}_j < 0$.

SOLUTION GOES HERE

Since $\text{null}(X) \not\perp e_j$, there exists some $v \in \text{null}(X)$ that has non-zero j-th coorindate. Denote the j-th coordinate of v as a real number c. If $c > 0$, we can construct $\hat{\beta} = \tilde{\beta} - \left(\frac{\tilde{\beta}_j}{c}\right) v - v$, which has the j-th coordinate less than 0. If $c < 0$, we can construct $\hat{\beta} = \tilde{\beta} + \left(\frac{\tilde{\beta}_j}{c}\right) v - v$, which also has the j-th coordinate less than 0.

For either case, the prediction

$$\begin{aligned}
\hat{y} &= X\tilde{\beta} \\
&= X\left[\tilde{\beta} \pm \left(\frac{\tilde{\beta}_j}{c}\right) v - v\right] \\
&= X\tilde{\beta} \pm X\left(\frac{\tilde{\beta}_j}{c}\right) v - Xv \\
&= X\tilde{\beta} \pm \left(\frac{\tilde{\beta}_j}{c}\right) Xv - Xv \\
&= X\tilde{\beta} \qquad (\because v \in \text{null}(X)) \\
&= \tilde{y}.
\end{aligned}$$

Hence, $\tilde{\beta}$ and $\hat{\beta}$ will have the same prediction under X, so as the MSE.

Ridge and lasso

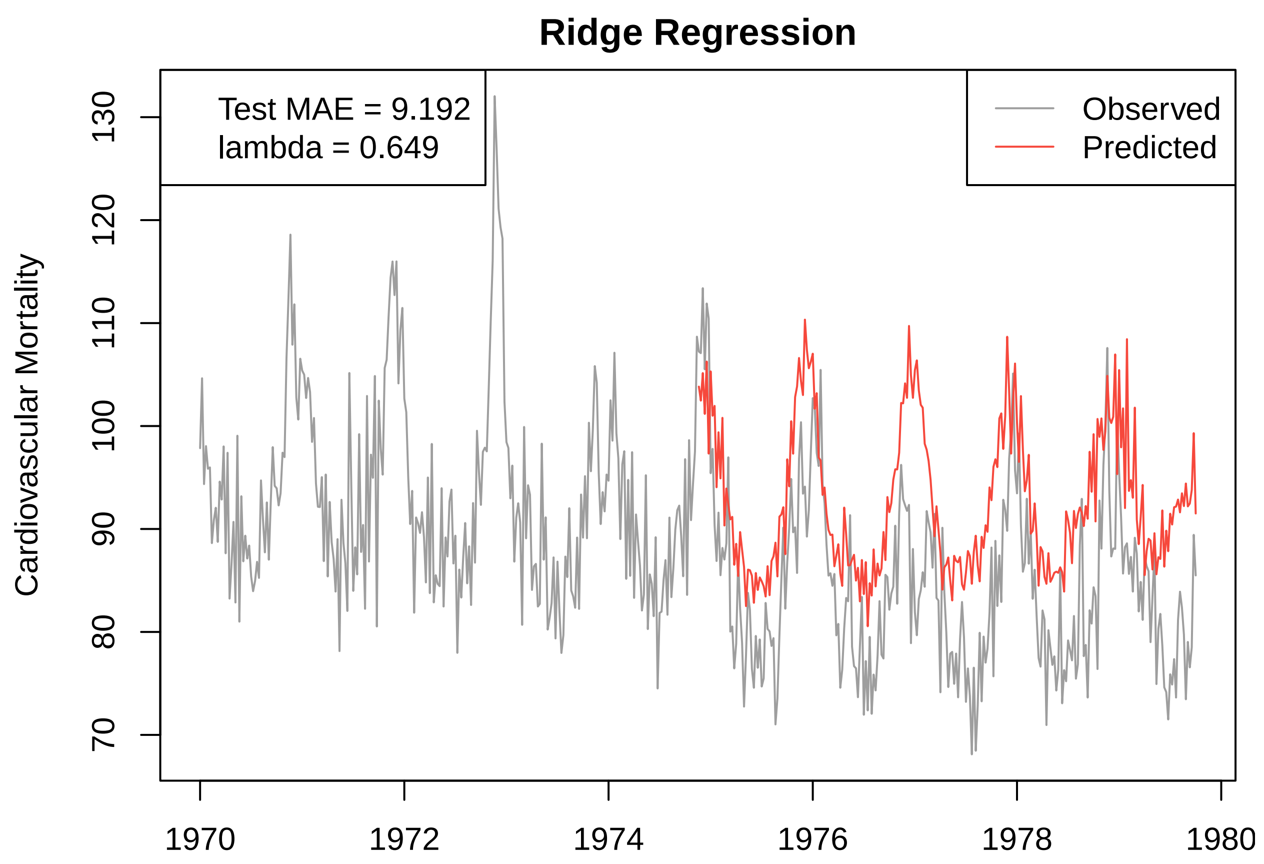
3. (8 pts) On the cardiovascular mortality regression data, form lagged features from the particulate matter and temperature variables, using lags 4, 8, . . . , 40 from each. Using the `glmnet` package, fit a ridge regression and lasso regression (two separate models), each over a grid of tuning parameter values λ chosen by the `glmnet()` function, with cardiovascular mortality as the response and all the lagged features as predictors (you should have 20 in total: 10 from particulate matter, and 10 from temperature). However, make sure you do this in a split-sample setup for validation, as follows, for each of ridge and lasso:

- fit the `glmnet` object on the *first half of the time series*;
- make predictions on the *second half of the time series*, for each λ;
- record the MAE of the predictions on the second half, for each λ;
- choose and report the value of λ with the lowest MAE;
- plot the cardiovascular mortality time series, along with the predictions on the second half, and print the MAE and the selected value of λ on the plot.

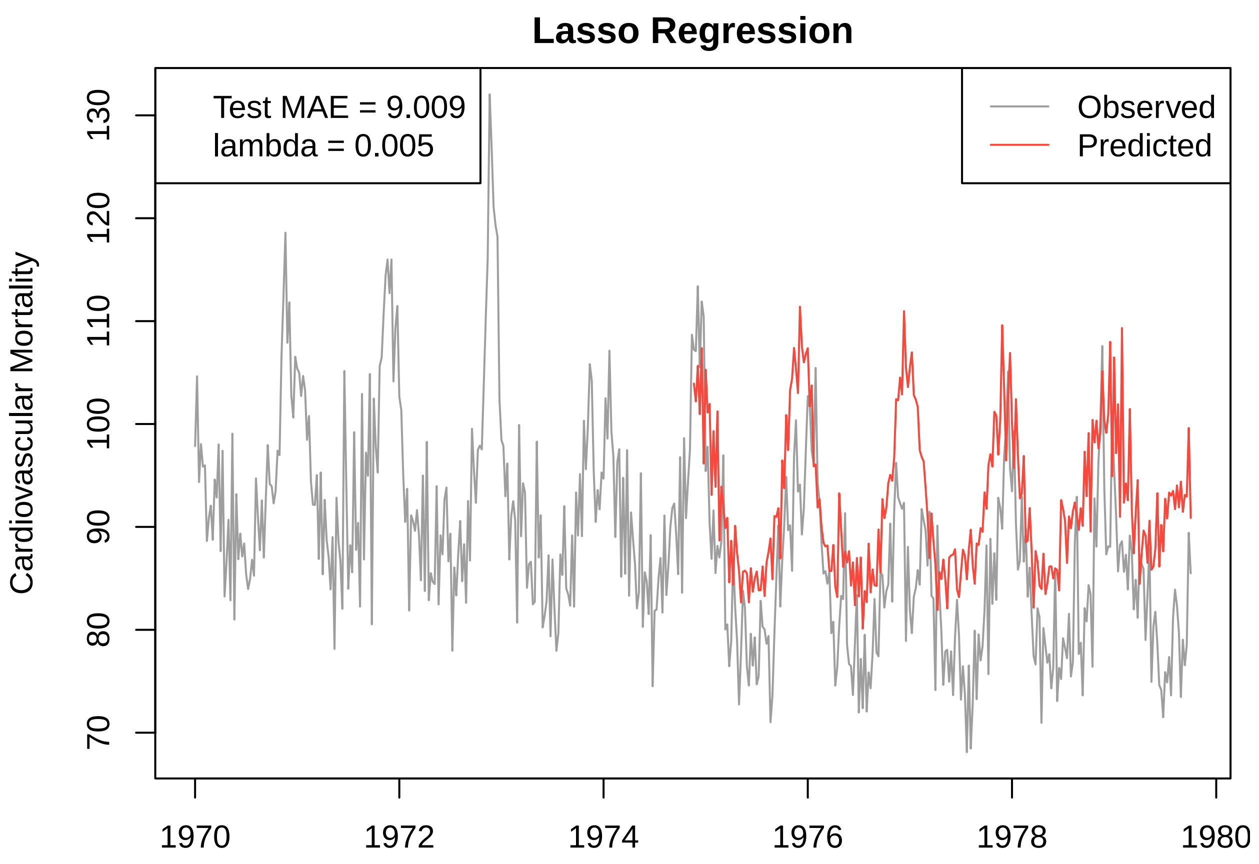
You can build off the code given in the regularization lecture (weeks 5-6: "Regularization and smoothing") for fitting the ridge and lasso models, and the regression lecture (weeks 3-4: "Regression and prediction") for the split-sample validation. *Note carefully that the lecture code includes lag 0, and here we do not, so that we can make ex-ante 4-week ahead forecasts!*

```
# CODE GOES HERE
library(astsa)
library(glmnet)

lags = seq(4, 40, 4)
y = as.numeric(cmort)
y_train = y[1:floor(length(y) / 2)]
y_test = y[(floor(length(y) / 2) + 1):length(y)]
```

```r
tempr_vec = as.numeric(tempr)
part_vec = as.numeric(part)
x = dplyr::lag(part_vec, lags[1])
x = cbind(x, dplyr::lag(tempr_vec, lags[1]))

for (i in lags[-1]) {
  x = cbind(x, dplyr::lag(part_vec, i))
  x = cbind(x, dplyr::lag(tempr_vec, i))
}
x_train = x[1:floor(length(y) / 2), ]
x_test = x[(floor(length(y) / 2) + 1):length(y), ]

na_obs = 1:max(lags)
x_train = x_train[-na_obs, ]
y_train = y_train[-na_obs]

####################### above is data preparation #############################

# ridge regression on first half of the time series
ridge = glmnet(x_train, y_train, alpha = 0)
beta_ridge = coef(ridge)
lambda_ridge = ridge$lambda

# ridge prediction on second half of the time series
y_predict_ridge = predict(ridge, newx = x_test, s = lambda_ridge)
ridge_mae = apply(abs(y_predict_ridge - y_test), 2, mean)
least_mae_ridge_lambda = lambda_ridge[which.min(ridge_mae)]

par(mar = c(2, 4, 2, 0.5))
plot(time(cmort), cmort,
    type = "l", col = 8,
    xlim = range(time(cmort)),
    ylim = range(cmort),
    xlab = "Year",
    ylab = "Cardiovascular Mortality",
    main = "Ridge Regression"
)
lines(time(cmort)[(floor(length(y) / 2) + 1):length(y)],
    y_predict_ridge[, which.min(ridge_mae)],
    col = 2
)
legend("topright",
    legend = c("Observed", "Predicted"),
    col = c(8, 2),
    lty = 1
)
legend("topleft",
    legend = c(paste("Test MAE =", round(ridge_mae[which.min(ridge_mae)], 3)),
        paste("lambda =", round(least_mae_ridge_lambda, 3))
    )
)
```



```r
# lasso regression on first half of the time series
lasso = glmnet(x_train, y_train, alpha = 1)
beta_lasso = coef(lasso)
lambda_lasso = lasso$lambda

# lasso prediction on second half of the time series
y_predict_lasso = predict(lasso, newx = x_test, s = lambda_lasso)
lasso_mae = apply(abs(y_predict_lasso - y_test), 2, mean)
least_mae_lasso_lambda = lambda_lasso[which.min(lasso_mae)]

par(mar = c(2, 4, 2, 0.5))
plot(time(cmort), cmort,
    type = "l", col = 8,
    xlim = range(time(cmort)),
    ylim = range(cmort),
    xlab = "Year",
    ylab = "Cardiovascular Mortality",
    main = "Lasso Regression"
)
lines(time(cmort)[(floor(length(y) / 2) + 1):length(y)],
    y_predict_lasso[, which.min(lasso_mae)],
    col = 2
)
legend("topright",
    legend = c("Observed", "Predicted"),
    col = c(8, 2),
    lty = 1
)
legend("topleft",
    legend = c(paste("Test MAE =", round(lasso_mae[which.min(lasso_mae)], 3)),
        paste("lambda =", round(least_mae_lasso_lambda, 3))
    )
)
```



4. (1 pts) Which lagged features were present in the MAE-optimal lasso model, selected by split-sample validation, in Q3?

```
# CODE GOES HERE
optimal_lasso_beta = beta_lasso[-1, which.min(lasso_mae)]
optimal_lasso_beta != 0
```

```
##    x
## TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##
## TRUE TRUE TRUE TRUE
```

All features are used in the MAE-optimal lasso model.

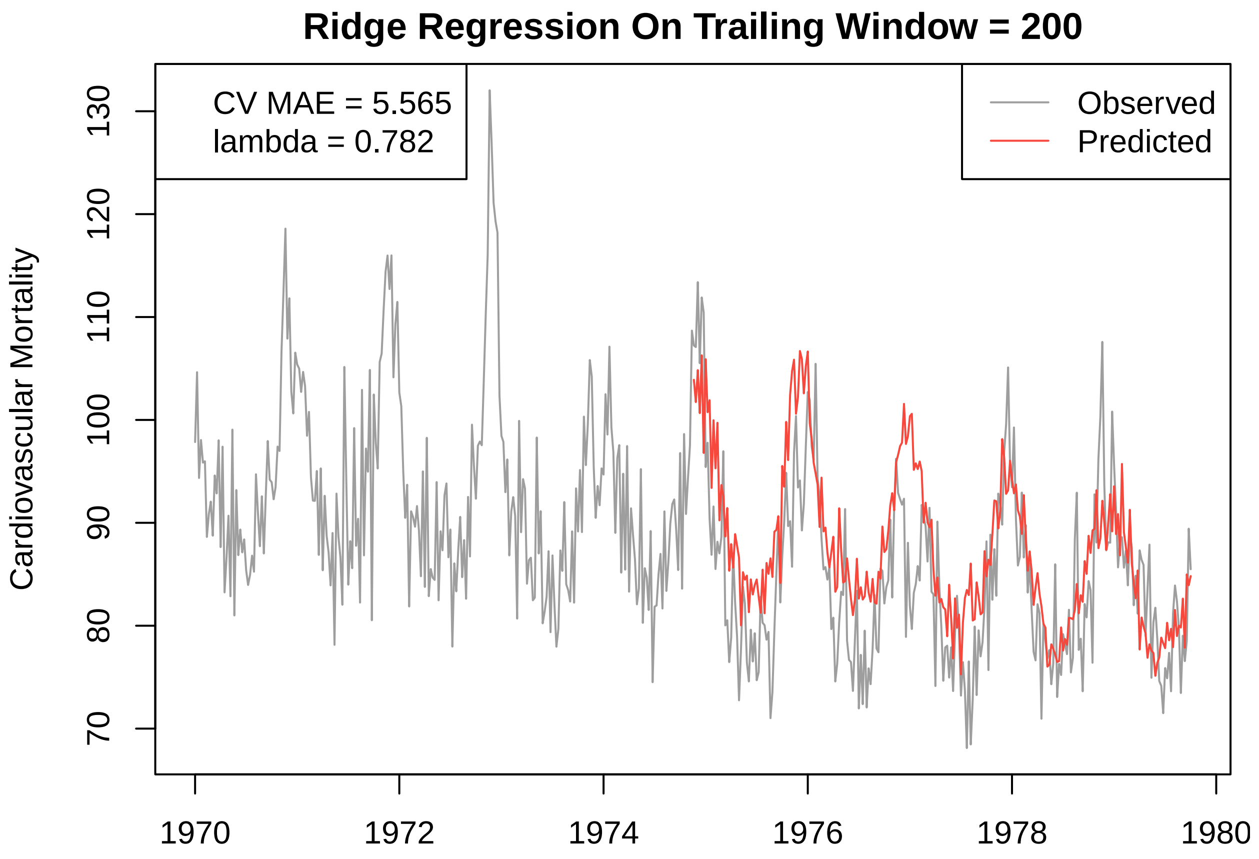
5. (8 pts) Repeat Q3, except implement time series cross-validation instead of split-sample validation. You should begin time series cross-validation on the second half of the time series, treating the first half as a burn-in set. Also, be sure to fit each ridge or lasso model using a trailing window of 200 time points (not all past).

   Warning: doing time series cross-validation properly will require us to pay attention to the following. The `glmnet()` function chooses a sequence of tuning parameter values λ based on the passed feature matrix `x` and response vector `y` (its first two arguments). However, in time series cross-validation, this will change at each time point. So if you just call `glmnet()` naively, then you will not have a consistent λ sequence over which to calculate MAE and perform tuning.

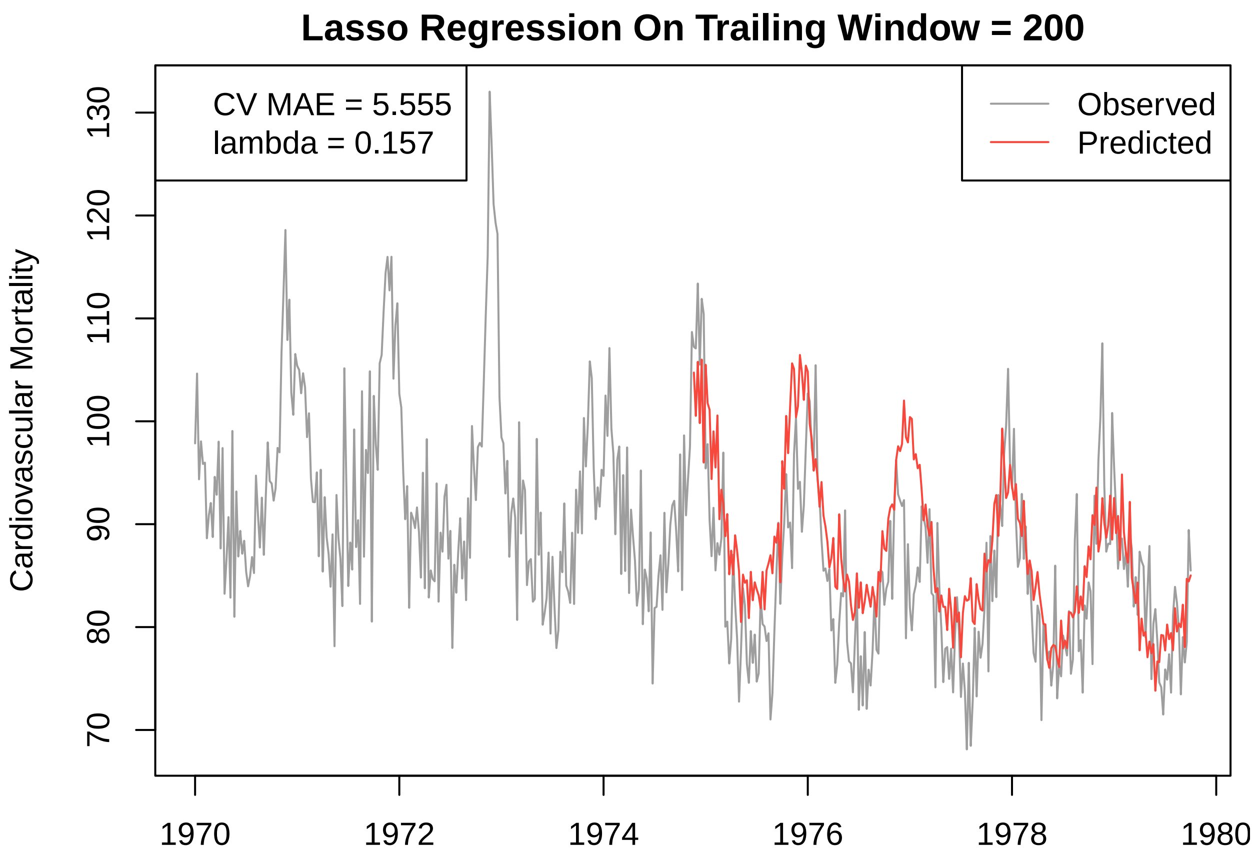
You can circumvent this issue by defining your own λ sequence ahead of time, and forcing `glmnet()` to use it by passing it through its `lambda` argument. Indeed, the best thing to do here is just to use the `lambda` sequence that `glmnet()` itself derived for the ridge and lasso models fit to the first half of the time series, which you already have from Q3. Do this, and then just as in Q3, produce a plot of the cardiovascular mortality time series, along with the predictions from time series CV on the second half, and print the MAE and the selected value of λ on the plot.

You can build off the code given in the regression lecture for time series cross-validation (or the code you wrote in Homework 2 to implement time series cross-validation).

```r
# CODE GOES HERE
window_size = 200
lags = seq(4, 40, 4)
y = as.numeric(cmort)
tempr_vec = as.numeric(tempr)
part_vec = as.numeric(part)
x = dplyr::lag(part_vec, lags[1])
x = cbind(x, dplyr::lag(tempr_vec, lags[1]))
for (i in lags[-1]) {
  x = cbind(x, dplyr::lag(part_vec, i))
  x = cbind(x, dplyr::lag(tempr_vec, i))
}
x_train = x[1:floor(length(y) / 2), ]
x_test = x[(floor(length(y) / 2) + 1):length(y), ]

na_obs = 1:max(lags)
x_train = x_train[-na_obs, ]
y_train = y_train[-na_obs]

t0 = floor(length(y) / 2) + 1
####################### above is data preparation #############################

# ridge regression on trailing window of 200 time points
y_predict_ridge = matrix(NA, length(y) - t0 + 1, length(lambda_ridge))
for (t in t0:length(y)) {
    x_train = x[(t - window_size - min(lags) + 1):(t - min(lags)), ]
    y_train = y[(t - window_size - min(lags) + 1):(t - min(lags))]
    ridge = glmnet(x_train, y_train, alpha = 0, lambda = lambda_ridge)
    y_predict_ridge[t - t0 + 1, ] = predict(
        ridge,
        newx = x[t, ], s = lambda_ridge
    )
}

cv_mae_ridge = apply(abs(y_predict_ridge - y[t0:length(y)]), 2, mean)
optimal_lambda_ridge = lambda_ridge[which.min(cv_mae_ridge)]

par(mar = c(2, 4, 2, 0.5))
plot(time(cmort), cmort,
    type = "l", col = 8,
    xlim = range(time(cmort)),
    ylim = range(cmort),
    xlab = "Year",
    ylab = "Cardiovascular Mortality",
    main = "Ridge Regression On Trailing Window = 200"
)
lines(time(cmort)[t0:length(y)],
    y_predict_ridge[, which.min(cv_mae_ridge)],
    col = 2
)
legend("topright",
    legend = c("Observed", "Predicted"),
    col = c(8, 2),
    lty = 1
)
legend("topleft",
    legend = c(paste("CV MAE =",
        round(cv_mae_ridge[which.min(cv_mae_ridge)], 3)),
        paste("lambda =", round(optimal_lambda_ridge, 3))
    )
)
```



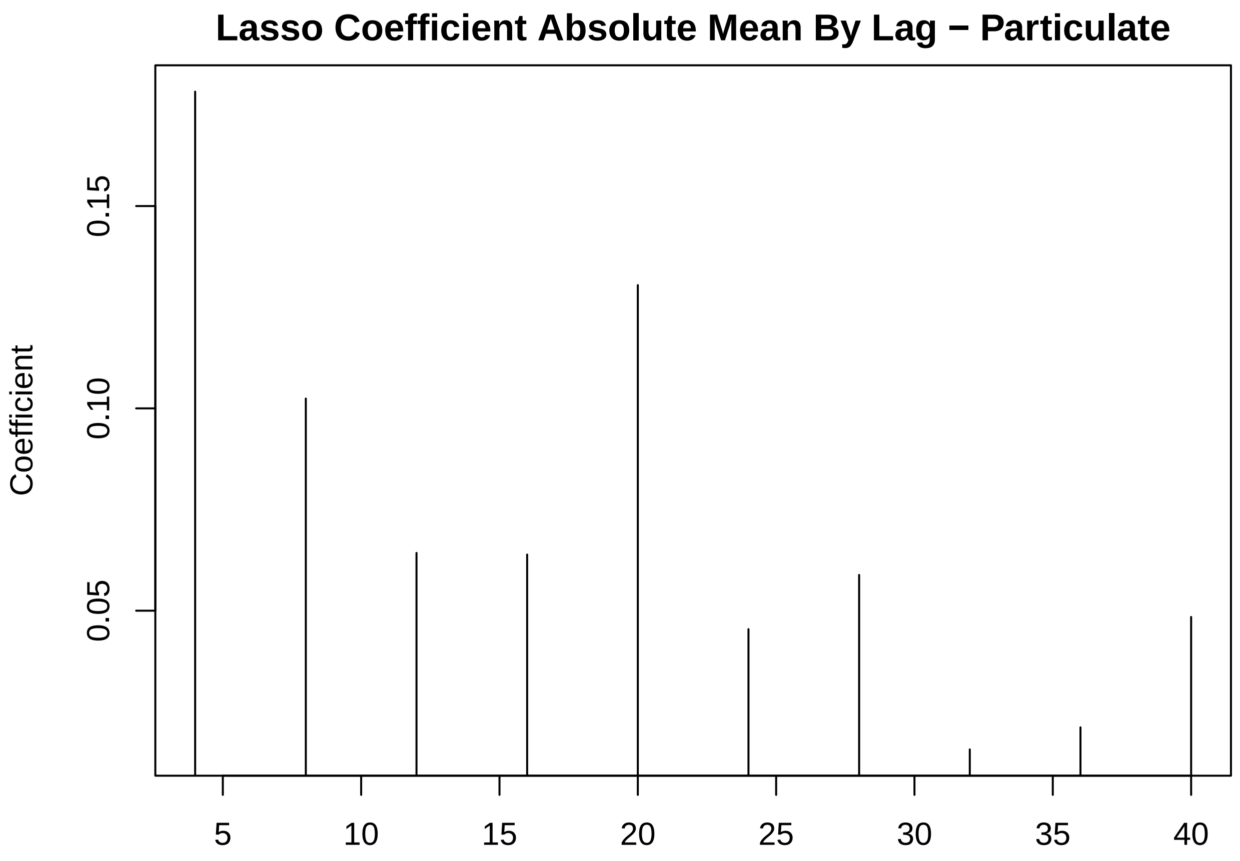
```r
# lasso regression on trailing window of 200 time points
y_predict_lasso = matrix(NA, length(y) - t0 + 1, length(lambda_lasso))
for (t in t0:length(y)) {
    x_train = x[(t - window_size - min(lags) + 1):(t - min(lags)), ]
    y_train = y[(t - window_size - min(lags) + 1):(t - min(lags))]
    lasso = glmnet(x_train, y_train, alpha = 1, lambda = lambda_lasso)
    y_predict_lasso[t - t0 + 1, ] = predict(
        lasso,
        newx = x[t, ], s = lambda_lasso
    )
}

cv_mae_lasso = apply(abs(y_predict_lasso - y[t0:length(y)]), 2, mean)
optimal_lambda_lasso = lambda_lasso[which.min(cv_mae_lasso)]

par(mar = c(2, 4, 2, 0.5))
plot(time(cmort), cmort,
    type = "l", col = 8,
    xlim = range(time(cmort)),
    ylim = range(cmort),
    xlab = "Year",
    ylab = "Cardiovascular Mortality",
    main = "Lasso Regression On Trailing Window = 200"
)
lines(time(cmort)[t0:length(y)],
    y_predict_lasso[, which.min(cv_mae_lasso)],
    col = 2
)
legend("topright",
    legend = c("Observed", "Predicted"),
    col = c(8, 2),
    lty = 1
)
legend("topleft",
    legend = c(paste("CV MAE =",
        round(cv_mae_lasso[which.min(cv_mae_lasso)], 3)),
        paste("lambda =", round(optimal_lambda_lasso, 3))
    )
)
```



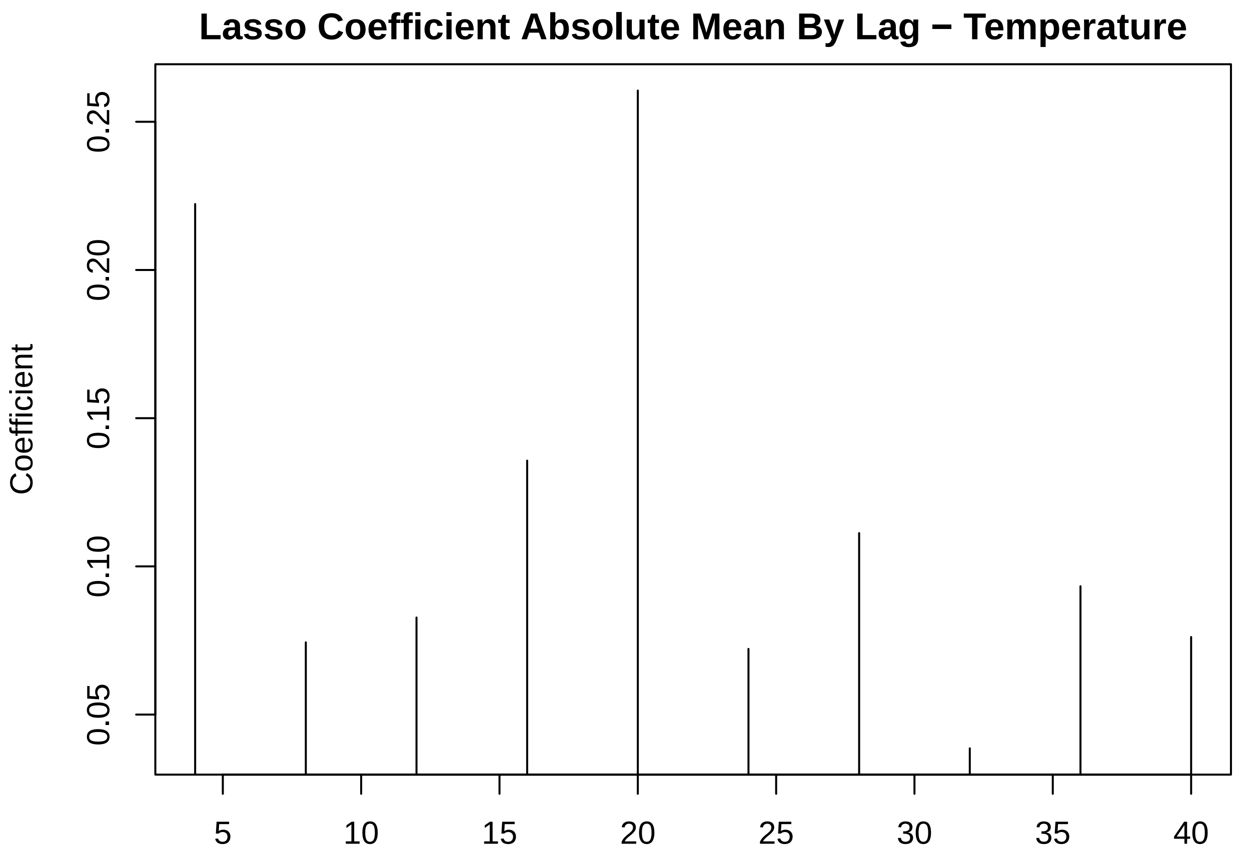
6. (Bonus) Report which lagged features were most frequently selected by the lasso. Because the lasso models will be refit at each time point (in the second half of the data set), you will have to additionally store the lasso solutions along your time series CV pass. Then, look back at the solutions that correspond to the MAE-optimal λ value, and choose some way of summarizing which of its components were consistently large in magnitude over time.

```
# CODE GOES HERE
window_size = 200
lags = seq(4, 40, 4)
y = as.numeric(cmort)
tempr_vec = as.numeric(tempr)
part_vec = as.numeric(part)
x = dplyr::lag(part_vec, lags[1])
x = cbind(x, dplyr::lag(tempr_vec, lags[1]))
for (i in lags[-1]) {
  x = cbind(x, dplyr::lag(part_vec, i))
  x = cbind(x, dplyr::lag(tempr_vec, i))
}
x_train = x[1:floor(length(y) / 2), ]
x_test = x[(floor(length(y) / 2) + 1):length(y), ]

na_obs = 1:max(lags)
x_train = x_train[-na_obs, ]
y_train = y_train[-na_obs]

t0 = floor(length(y) / 2) + 1

####################### above is data preparation #############################
```

```r
# lasso regression on trailing window of 200 time points
y_predict_lasso = matrix(NA, length(y) - t0 + 1, length(lambda_lasso))
lasso_coeff = matrix(NA, length(y) - t0 + 1, 20)
for (t in t0:length(y)) {
    x_train = x[(t - window_size - min(lags) + 1):(t - min(lags)), ]
    y_train = y[(t - window_size - min(lags) + 1):(t - min(lags))]
    lasso = glmnet(x_train, y_train, alpha = 1, lambda = least_mae_lasso_lambda)
    y_predict_lasso[t - t0 + 1, ] = predict(
        lasso,
        newx = x[t, ], s = lambda_lasso
    )
    lasso_coeff[t - t0 + 1, ] = coef(lasso)[-1]

}

part_beta_mean = apply(abs(lasso_coeff[, seq(1, 20, 2)]), 2, mean)
tempr_beta_mean = apply(abs(lasso_coeff[, seq(2, 20, 2)]), 2, mean)

par(mar = c(2, 4, 2, 0.5))
plot(seq(4, 40, 4),
    part_beta_mean,
    type = "h", xlab = "Lag", ylab = "Coefficient",
    main = "Lasso Coefficient Absolute Mean By Lag - Particulate"
)
```



```r
plot(seq(4, 40, 4),
    tempr_beta_mean,
    type = "h", xlab = "Lag", ylab = "Coefficient",
    main = "Lasso Coefficient Absolute Mean By Lag - Temperature"
)
```



From question 5, we have already found the optimal lambda for lasso is around 0.005. Now, we re-fit the lasso model with the optimal lambda on a trailing window of 200 time points and record its beta coefficients. Then, we plot the absolute mean of all the fitted beta coefficients by lag. From the plot, we can see lag 4, 8, and 20 of Particulate is significant, while lag 4, 20, and 16 of Temperature is significant.

HP filter

7. (5 pts) Recall in lecture we saw the HP filter could be written explicitly as

$$\hat{\theta} = \underbrace{(I + \lambda D^T D)^{-1}}_{K} y,$$

where $D \in \mathbb{R}^{(n-2)\times n}$ is the second difference matrix on n points. In other words, defining $K \in \mathbb{R}^{n\times n}$ as above,

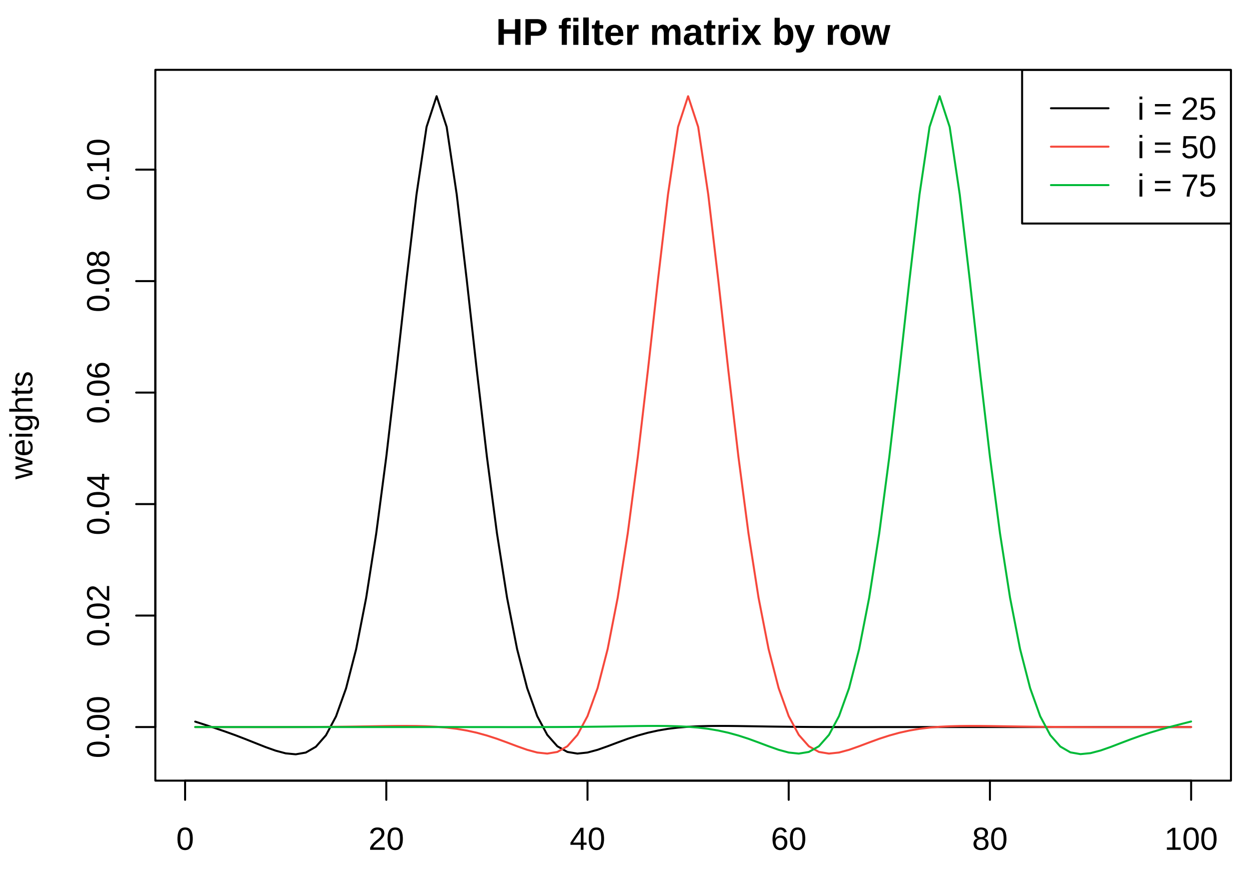
$$\hat{\theta}_i = \sum_{j=1}^{n} K_{ij} y_j, \quad i = 1, \dots, n.$$

Compute the matrix K empirically for a problem of size $n = 100$, and setting the tuning parameter to be $\lambda = 100$; inspect three of its rows, at indices $i = 25, 50, 75$. For each i, plot the i^{th} row as a curve over the underlying position $1, \dots, n$; that is, plot the x-y pairs

$$(x_j, y_j) = (j, K_{ij}), \quad j = 1, \dots, n$$

as a curve. Overlay the curves for all three rows on the same plot, each in a different color. What do these curves look like to you? Use the plot to argue that the HP filter acts like a kernel smoother.

```r
# CODE GOES HERE
n = 100
lambda = 100

D = matrix(0, nrow = n - 2, ncol = n)
for (i in 1:(n - 2)) {
    D[i, i] = 1
    D[i, i + 1] = -2
    D[i, i + 2] = 1
}

K = solve(diag(n) + lambda * t(D) %*% D)

par(mar = c(2, 4, 2, 0.5))
plot(1:n, K[25, ],
    col = 1,
    main = "HP filter matrix by row",
    type = "l",
    xlab = "n",
    ylab = "weights"
)
lines(1:n, K[50, ], col = 2)
lines(1:n, K[75, ], col = 3)
legend("topright",
    legend = c("i = 25", "i = 50", "i = 75"),
    col = c(1, 2, 3),
    lty = 1
)
```



The curves look like Gaussian kernel smoother to me, except with negative weights at the tail of thr curve. Indeed, HP filter acts like a kernal smoother as we can see from the plot: looking at $i = 50$, it looks like a smoother with kernel function that produces a bell shaped weights for bandwidth $= 50$ from 25 to 75.

8. (Bonus) Do a literature search to find theory on the *asymptotically equivalent kernel* for the HP filter. This should have a closed-form. Plot this and comment on whether or not your empirical results adhere to what is known asymptotically.

```
# CODE GOES HERE
```

Trend filter

9. (Bonus) Implement 5-fold cross-validation in order to tune λ in trend filtering applied to the Boston marathon men's data set. Recall the description of how to set folds in a special "structured" way, for tuning smoothers, given near the end of the lecture notes (weeks 5-6: "Regularization and smoothing"). The code below shows how to run trend filtering and derive estimates at the held-out points for one fold. You can build off this code for your solution. You will need to install the `glmgen` package from GitHub, which you can do using the code that has been commented out.

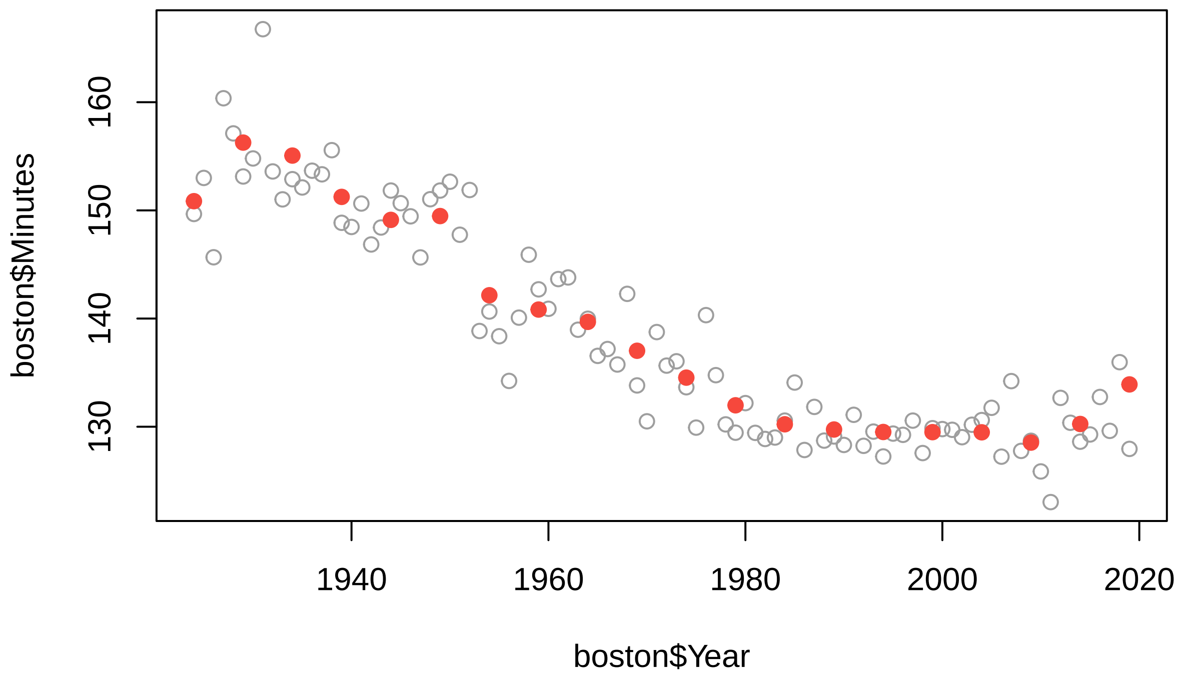
   Important note: just like `glmnet()`, the `trendfilter()` function (in the `glmnet` package) computes its own `lambda` sequence. So you will need to define an initial `lambda` sequence to pass to each subsequent call to `trendfilter()`, so that you can have a consistent grid of tuning parameter values over which to perform cross-validation. We do this below by using the `lambda` sequence that `trendfilter()` itself derived when the trend filtering model is fit on the full data set.

   After implementing cross-validation, compute and report the λ value with the smallest cross-validated MAE. Then, lastly, plot the solution at this value of λ when the model is fit to the full data set (this is already available in the `tf` object below.)

```r
# devtools::install_github("glmgen/glmgen", subdir = "R_pkg/glmgen")
library(glmgen)
library(fpp3)

boston = boston_marathon |>
  filter(Year >= 1924) |>
  filter(Event == "Men's open division") |>
  mutate(Minutes = as.numeric(Time)/60) |>
  select(Year, Minutes)

# Fit trend filtering on the entire data in order to grab the lambda sequence
tf = trendfilter(x = boston$Year, y = boston$Minutes, k = 1)
lambda = tf$lambda

n = nrow(boston)       # Number of points
k = 5                  # Number of folds
inds = rep_len(1:k, n) # Folds indices

# Fit trend filtering on all points but those in first fold. We are forcing it
# to use the lambda sequence that we saved above
tf_subset = trendfilter(x = boston$Year[inds != 1],
                        y = boston$Minutes[inds != 1],
                        k = 1, lambda = lambda)

# Compute the predictions on the points in the first fold. Plot the predictions
# (as a sanity check) at a particular value of lambda in the middle of the grid
yhat = predict(tf_subset, x.new = boston$Year[inds == 1])
```

```
## Warning: In predict:
##     Predict called at new x values out of the original range.
```

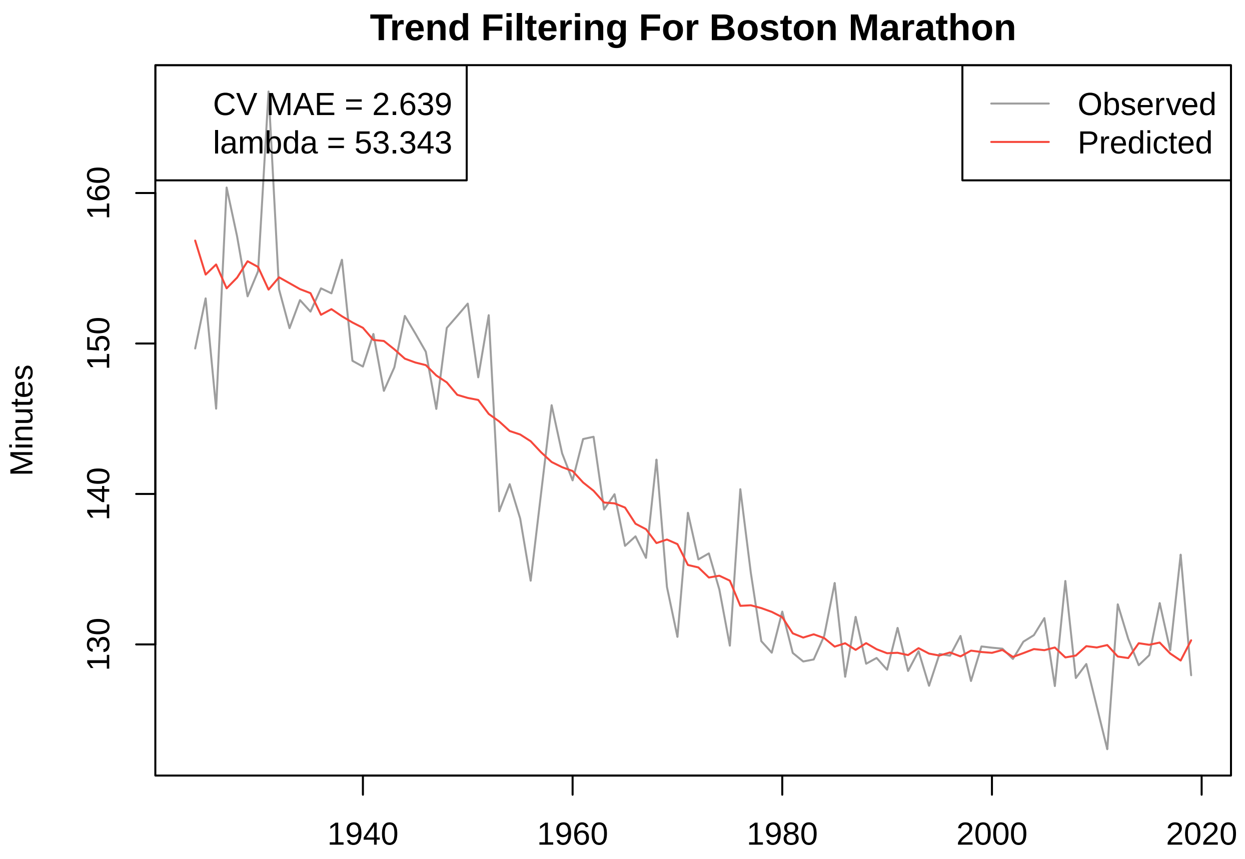
```r
plot(boston$Year, boston$Minutes, col = 8)
points(boston$Year[inds == 1], yhat[, 25], col = 2, pch = 19)
```



```r
y_predict = matrix(NA, nrow = n, ncol = length(lambda))
for (i in 1:5) {
    tf_fit = trendfilter(
        x = boston$Year[inds != i],
        y = boston$Minutes[inds != i],
        k = 1, lambda = lambda
    )
    y_predict[inds == i, ] = predict(tf_fit, x.new = boston$Year[inds == i])
}
```

```
## Warning: In predict:
##     Predict called at new x values out of the original range.
```

```r
cv_mae = apply(abs(y_predict - boston$Minutes), 2, mean)
optimal_lambda_index = which.min(cv_mae)
optimal_lambda = lambda[optimal_lambda_index]

par(mar = c(2, 4, 2, 0.5))
plot(boston$Year,
    boston$Minutes,
    col = 8,
    type = "l",
    xlab = "Year",
    ylab = "Minutes",
    main = "Trend Filtering For Boston Marathon"
)
lines(boston$Year, y_predict[, optimal_lambda_index], col = 2)
legend("topright",
    legend = c("Observed", "Predicted"),
    col = c(8, 2),
    lty = 1
)
legend("topleft",
    legend = c(
        paste("CV MAE =", round(cv_mae[optimal_lambda_index], 3)),
        paste("lambda =", round(optimal_lambda, 3))
    )
)
```



Spectral analysis

10. (3 pts) Let ω_j, $j = 1, \dots, p$ be fixed and arbitrary frequencies and let U_{j1}, U_{j2}, $j = 1, \dots, p$ be uncorrelated random variables with mean zero, where U_{j1}, U_{j2} have variance σ_j^2. Define

$$x_t = \sum_{j=1}^{p} \Big(U_{j1} \cos(2\pi\omega_j t) + U_{j2} \sin(2\pi\omega_j t) \Big)$$

for $t = 1, 2, 3, \dots$. Prove that this process is stationary, and show that its auto-covariance function is of the form given in lecture (weeks 7-8, "Spectral analysis and filtering").

SOLUTION GOES HERE

$$\begin{aligned}
\mu_t &= \mathbb{E}(x_t) \\
&= \mathbb{E}\left[\sum_{j=1}^{p}\Big(U_{j1}\cos(2\pi\omega_j t) + U_{j2}\sin(2\pi\omega_j t)\Big)\right] \\
&= \mathbb{E}\left[\sum_{j=1}^{p}(U_{j1} + Uj2)\right] \\
&= 0.
\end{aligned}$$

Notation: Use S to represent x_s and T to represent x_t, so $S_{k1} = U_{k1}\cos(2\pi\omega_i s)$, $S_{k2} = U_{k2}\sin(2\pi\omega_i s)$, while $T_{k1} = U_{k1}\cos(2\pi\omega_i t)$, $T_{k2} = U_{k2}\sin(2\pi\omega_i t)$. Notice the S and T are uncorrelated if they have different subscripts. Then the auto-covariance is

$$\begin{aligned}
\gamma(s,t) &= \mathrm{Cov}\left(x_s, x_t\right) \\
&= \mathrm{Cov}\left(\sum_{j=1}^{p}\Big(U_{j1}\cos(2\pi\omega_j s) + U_{j2}\sin(2\pi\omega_j s)\Big), \sum_{j=1}^{p}\Big(U_{j1}\cos(2\pi\omega_j t) + U_{j2}\sin(2\pi\omega_j t)\Big)\right) \\
&= \sum_{j=1}^{p}\sum_{i=1}^{p}\mathrm{Cov}\left(S_{j1}, T_{i1}\right) + \sum_{j=1}^{p}\sum_{i=1}^{p}\mathrm{Cov}\left(S_{j1}, T_{i2}\right) + \sum_{j=1}^{p}\sum_{i=1}^{p}\mathrm{Cov}\left(S_{j2}, T_{i1}\right) + \sum_{j=1}^{p}\sum_{i=1}^{p}\mathrm{Cov}\left(S_{j2}, T_{i2}\right) \\
&= \sum_{i=1}^{p}\mathrm{Cov}\left(S_{i1}, T_{i1}\right) + \sum_{i=1}^{p}\mathrm{Cov}\left(S_{i2}, T_{i2}\right) \\
&= \sum_{i=1}^{p}\sigma^2\cos(2\pi\omega_i s)\cos(2\pi\omega_i t) + \sum_{i=1}^{p}\sigma^2\sin(2\pi\omega_i s)\sin(2\pi\omega_i t) \\
&= \sigma^2\sum_{i=1}^{p}\left[\cos(2\pi\omega_i s)\cos(2\pi\omega_i t) + \sin(2\pi\omega_i s)\sin(2\pi\omega_i t)\right] \\
&= \sigma^2\sum_{i=1}^{p}\cos(2\pi\omega_i s - 2\pi\omega_i t) \\
&= \sigma^2\sum_{i=1}^{p}\cos(2\pi\omega_i(s-t)),
\end{aligned}$$

which after reparametrizing to the lag h is the same as

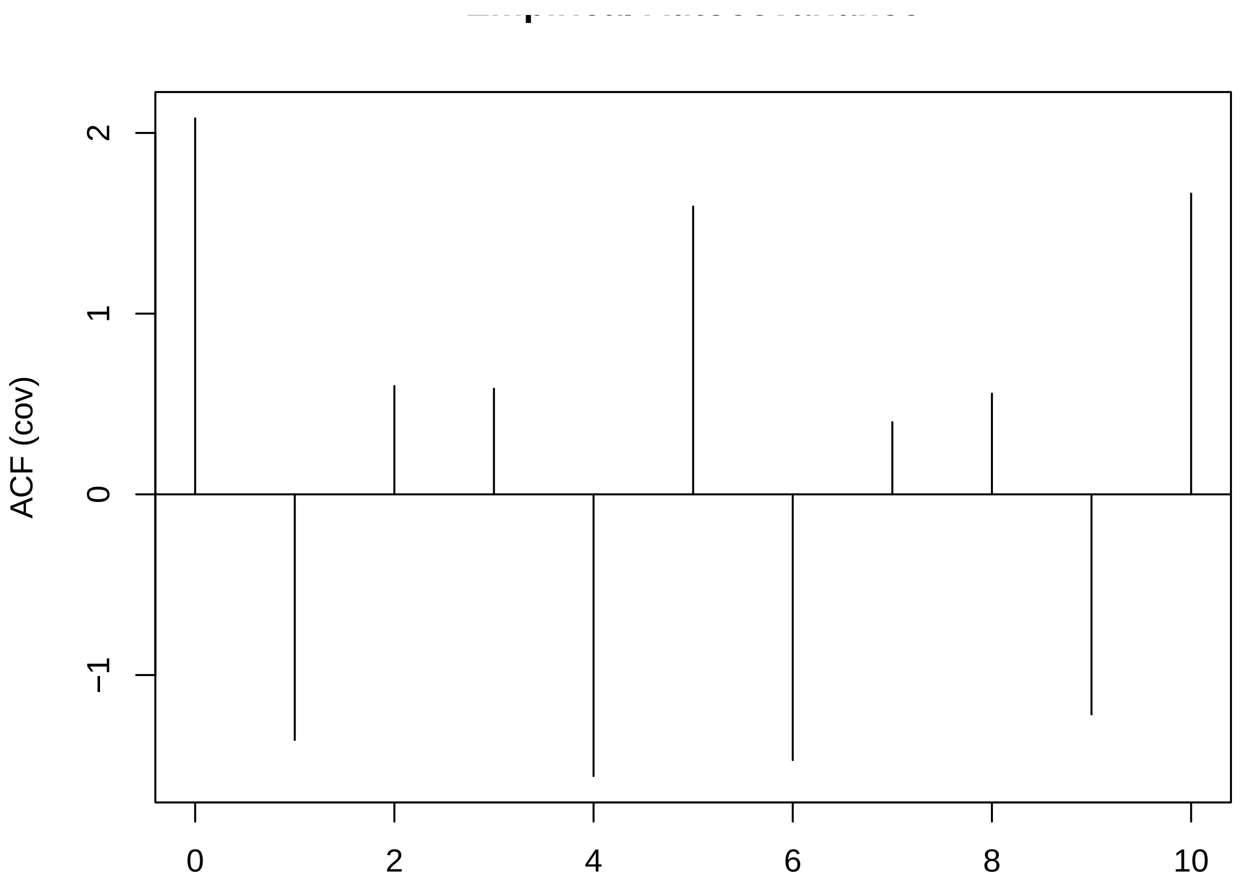
$$\gamma(h) = \sigma^2\sum_{i=1}^{p}\cos(2\pi\omega_i h).$$

Since the auto-covariance is dependent on the lag only and mean is a constant 0, the process is weakly stationary.

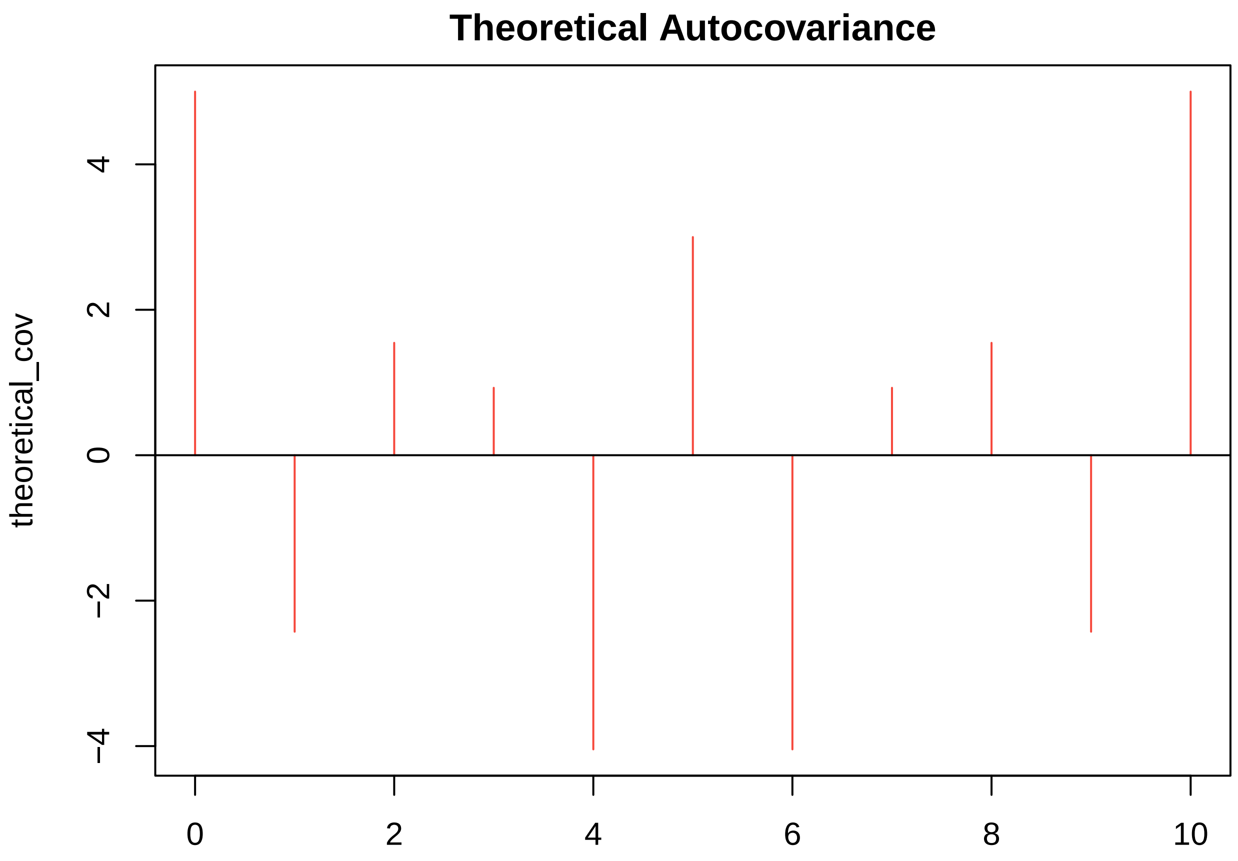
11. (2 pts) Construct a small empirical example to verify the auto-covariance formula you derived in Q10. That is, generate a process with at least $p = 2$ components. compute its auto-correlation function with `acf()`, and compare to the analytic formula you derived.

```
# CODE GOES HERE
n = 50
sd1 = 1
sd2 = 2
U_11 = rnorm(1, mean = 0, sd = sd1)
```

```r
U_12 = rnorm(1, mean = 0, sd = sd1)
U_21 = rnorm(1, mean = 0, sd = sd2)
U_22 = rnorm(1, mean = 0, sd = sd2)

w_1 = 0.1
w_2 = 0.4

x = U_11 * cos(2 * pi * w_1 * 1:n) + U_12 * sin(2 * pi * w_1 * 1:n) +
    U_21 * cos(2 * pi * w_2 * 1:n) + U_22 * sin(2 * pi * w_2 * 1:n)

theoretical_acf = function(h) {
    sd1^2 * cos(2 * pi * w_1 * h) + sd2^2 * cos(2 * pi * w_2 * h)
}
theoretical_cov = rep(0, 11)
for (i in 1:11) {
    theoretical_cov[i] = theoretical_acf(i - 1)
}

par(mar = c(2, 4, 2, 0.5))
acf(x, type = "covariance", lag.max = 10, main="Empirical Autocovariance")
```



```r
plot(0:10, theoretical_cov,
    col = 2, type = "h",
    main = "Theoretical Autocovariance"
)
abline(h = 0, col = 1)
```


Theoretical Autocovariance
theoretical_cov
4
2
0
−2
−4
0
2
4
6
8
10