PS_3_solutions_2020

May 27, 2020

1 PS3 Solutions 2020

[1]:
```
#install.packages('readxl')
#install.packages('tseries')
#install.packages('urca')
#install.packages('vars')
library(readxl)
library(tseries)
library(urca)
library(vars)
```

```
Warning message:
"package 'readxl' was built under R version 3.6.3"Warning message:
"package 'tseries' was built under R version 3.6.3"Registered S3 method
overwritten by 'xts':
  method     from
  as.zoo.xts zoo
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
Warning message:
"package 'urca' was built under R version 3.6.3"Warning message:
"package 'vars' was built under R version 3.6.3"Loading required package: MASS
Loading required package: strucchange
Warning message:
"package 'strucchange' was built under R version 3.6.3"Loading required package:
zoo

Attaching package: 'zoo'

The following objects are masked from 'package:base':

    as.Date, as.Date.numeric

Loading required package: sandwich
Warning message:
"package 'sandwich' was built under R version 3.6.3"Loading required package:
lmtest
```

```
Warning message:
"package 'lmtest' was built under R version 3.6.3"
```

[2]:
```
data = read_excel("C:/Users/Fabio/Dropbox/Lecture_Macroeconometrics/Data/
↪ps3_data_Q1_2020.xlsx")
```

[3]:
```
head(data)
```

Year	Revenue_Total	G_Health	G_Spending_Total
1970	29.80710	2.798418	34.24047
1971	29.27349	2.970855	34.55139
1972	30.40609	3.094316	34.40657
1973	30.63798	3.181722	33.37959
1974	31.23342	3.422005	34.56156
1975	29.61273	3.688566	36.87401

1.1 1.

1.1.1 a)

[4]:
```
data$G_Other = data$G_Spending_Total -  data$G_Health
data = data[, -c(1,4)] # delete variable 1 = Year and 4 = G_Spending_Total
data = data[,c(2,1,3)]
head(data)
```

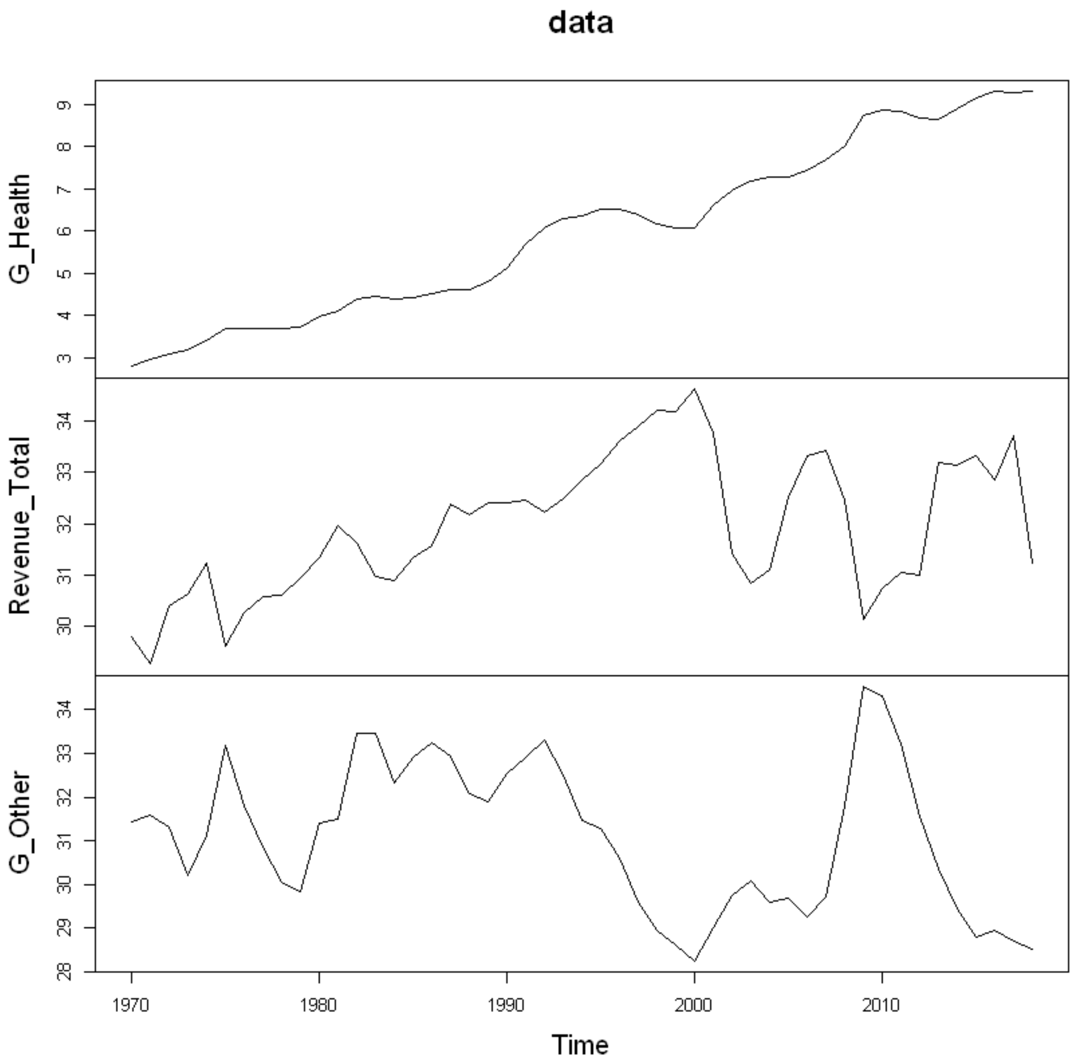
G_Health	Revenue_Total	G_Other
2.798418	29.80710	31.44205
2.970855	29.27349	31.58054
3.094316	30.40609	31.31226
3.181722	30.63798	30.19786
3.422005	31.23342	31.13956
3.688566	29.61273	33.18545

[5]:
```
data = ts(data, start = 1970, end = 2018, frequency = 1)
```

1.1.2 b)

Lets take a look at the time series.

[6]:
```
plot(data, type = "l")
```

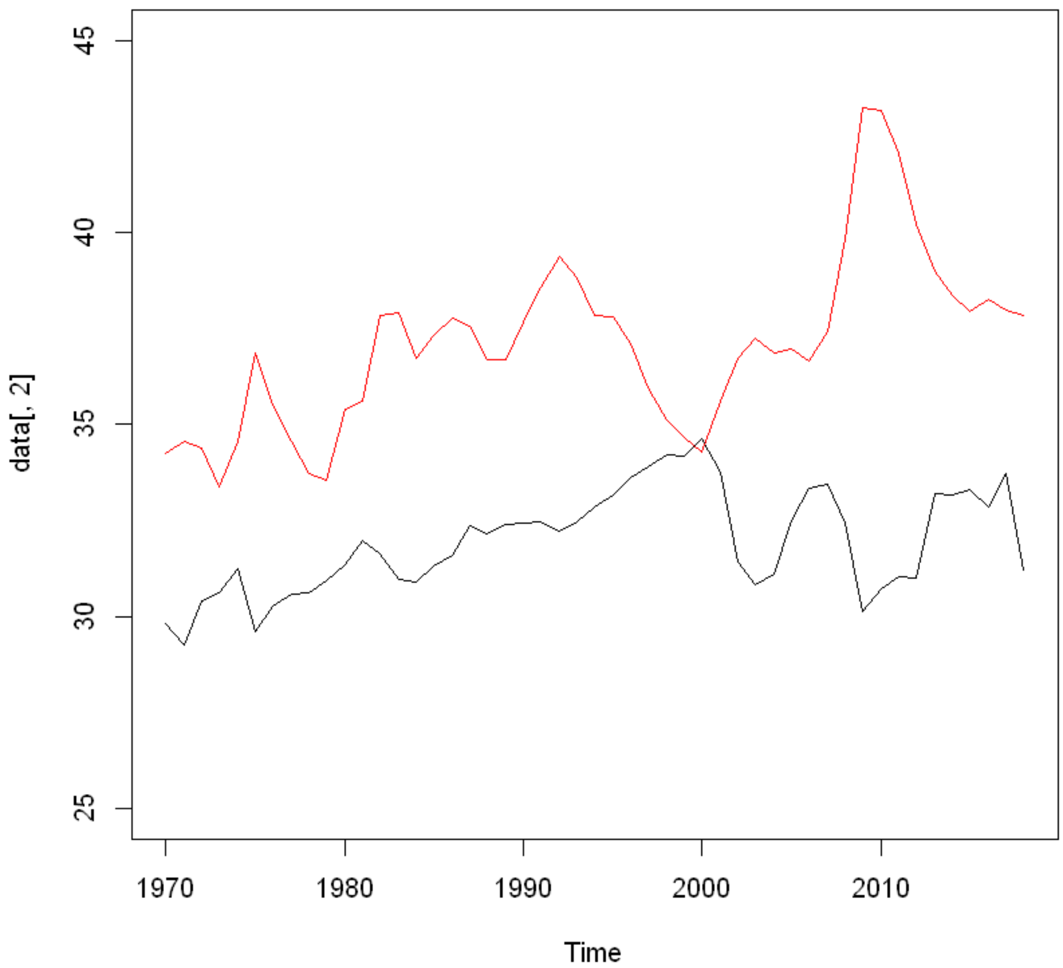


Observations:

- Government Spending on Health and Government Total Revenue exhibit trends
- The Government spending on health seems to be trend stationary while Revenue seems to have a stochastic trend

If we sum health spending with other spending and compare it with revenue they seem to follow the same stochastic trend

[7]:
```
plot(data[,2], type = "l", ylim = c(25,45))
lines(data[,1]+ data[,3], type = "l", col = "red")
```



[8]:
```
for(i in 1:3){
print(adf.test(data[,i]))
    }
```

```
        Augmented Dickey-Fuller Test

data:  data[, i]
Dickey-Fuller = -2.5975, Lag order = 3, p-value = 0.3351
alternative hypothesis: stationary


        Augmented Dickey-Fuller Test
```

```
data:  data[, i]
Dickey-Fuller = -2.2865, Lag order = 3, p-value = 0.4595
alternative hypothesis: stationary


        Augmented Dickey-Fuller Test

data:  data[, i]
Dickey-Fuller = -2.774, Lag order = 3, p-value = 0.2644
alternative hypothesis: stationary
```

1.2 c)

Based on the results from exercise b), we know that the variables in levels are nonstationary. Next, we try to check for the order of integration of the variables.

[9]:
```
for(i in 1:3){
print(adf.test(diff(data[,i])))
    }
```

```
Warning message in adf.test(diff(data[, i])):
"p-value smaller than printed p-value"


        Augmented Dickey-Fuller Test

data:  diff(data[, i])
Dickey-Fuller = -4.2857, Lag order = 3, p-value = 0.01
alternative hypothesis: stationary


Warning message in adf.test(diff(data[, i])):
"p-value smaller than printed p-value"


        Augmented Dickey-Fuller Test

data:  diff(data[, i])
Dickey-Fuller = -4.8024, Lag order = 3, p-value = 0.01
alternative hypothesis: stationary


        Augmented Dickey-Fuller Test

data:  diff(data[, i])
Dickey-Fuller = -4.1612, Lag order = 3, p-value = 0.01069
alternative hypothesis: stationary
```

All variables are stationary in their differences, therefore, we can conclude that the variables have the same order of integration. Consequently, we can proceed with our estimation.

[10]:
```
data = data[, c(1,2)]
VARselect(data, lag.max = 5)
```

$selection **AIC(n)** 2 **HQ(n)** 2 **SC(n)** 2 **FPE(n)** 2

$criteria

	1	2	3	4	5
AIC(n)	-3.70351432	-3.99292917	-3.85611414	-3.78686194	-3.70292327
HQ(n)	-3.61328748	-3.84255109	-3.64558483	-3.51618140	-3.37209151
SC(n)	-3.46021574	-3.58743152	-3.28841744	-3.05696618	-2.81082845
FPE(n)	0.02464724	0.01848201	0.02126565	0.02293351	0.02519091

[11]:
```
trace_test = ca.jo(data, type = "trace",
         ecdet = "const" , K = 2, spec = "transitory")
summary(trace_test)

eigen_test = ca.jo(data, type = "eigen",
         ecdet = "const" , K = 2, spec = "transitory")
summary(eigen_test)
```

```
######################
# Johansen-Procedure #
######################

Test type: trace statistic , without linear trend and constant in cointegration

Eigenvalues (lambda):
[1] 2.669163e-01 1.250291e-01 3.364630e-17

Values of teststatistic and critical values of test:

          test 10pct  5pct  1pct
r <= 1 |  6.28  7.52  9.24 12.97
r = 0  | 20.87 17.85 19.96 24.60

Eigenvectors, normalised to first column:
(These are the cointegration relations)

                 G_Health.l1 Revenue_Total.l1    constant
G_Health.l1         1.000000         1.000000  1.00000000
Revenue_Total.l1   -4.930188        -1.458496  0.02283146
constant          155.651228        37.145650 -5.48496805

Weights W:
(This is the loading matrix)

                 G_Health.l1 Revenue_Total.l1     constant
```

```
G_Health.d      -0.004587278     -0.016447223 3.734854e-17
Revenue_Total.d  0.071273797     -0.004609789 5.068136e-16



######################
# Johansen-Procedure #
######################

Test type: maximal eigenvalue statistic (lambda max) , without linear trend and constant in co

Eigenvalues (lambda):
[1] 2.669163e-01 1.250291e-01 3.364630e-17

Values of teststatistic and critical values of test:

          test 10pct  5pct  1pct
r <= 1 |  6.28  7.52  9.24 12.97
r = 0  | 14.59 13.75 15.67 20.20

Eigenvectors, normalised to first column:
(These are the cointegration relations)

                 G_Health.l1 Revenue_Total.l1    constant
G_Health.l1         1.000000         1.000000  1.00000000
Revenue_Total.l1   -4.930188        -1.458496  0.02283146
constant          155.651228        37.145650 -5.48496805

Weights W:
(This is the loading matrix)

                 G_Health.l1 Revenue_Total.l1     constant
G_Health.d      -0.004587278     -0.016447223 3.734854e-17
Revenue_Total.d  0.071273797     -0.004609789 5.068136e-16
```

The cointegrating vector is

[12]:
```
coint_V = attr(trace_test, "V")
coint_V[,1]
```

G__Health.l1 1 **Revenue__Total.l1** -4.93018833723902 **constant** 155.651227676693

We find that, in the long-run, one out of five dollars of tax revenue we spend invest in the health care sector. The constant shifts this relation up, meaning that the linear relation only holds up to the scalar of 155.

1.2.1 2 d)

The speed of adjustment coefficients vector is

[13]:
```
attr(trace_test, "W")[,1]
```

G__Health.d -0.00458727817757424 **Revenue__Total.d** 0.0712737973422046

So Revenue increases when spending in health is above the equilibrium level (negative residual in the long run equilibirum equation) and it decreases when spending in health is below the revenue in the long run equilibrium. Let's formally test which of the coefficients is different from zero.

[14]:
```
DA <- matrix(c(1,0), c(2,1))
summary(alrtest(trace_test, A=DA, r=1))
```

```
######################
# Johansen-Procedure #
######################

Estimation and testing under linear restrictions on beta

The VECM has been estimated subject to:
beta=H*phi and/or alpha=A*psi

     [,1]
[1,]    1
[2,]    0

Eigenvalues of restricted VAR (lambda):
[1] 0.1254 0.0000 0.0000

The value of the likelihood ratio test statistic:
8.29 distributed as chi square with 1 df.
The p-value of the test statistic is: 0

Eigenvectors, normalised to first column
of the restricted VAR:

                       [,1]
RK.G_Health.l1       1.0000
RK.Revenue_Total.l1 -1.6036
RK.constant         42.0999

Weights W of the restricted VAR:

        [,1]
[1,] -0.0174
[2,]  0.0000
```

We reject this restriction of the second adjustment coefficient being zero. Consequently, the speed of adjustment coefficient for revenue is different from zero.

[15]:
```
DA <- matrix(c(0,1), c(2,1))
summary(alrtest(trace_test, A=DA, r=1))
```

```
######################
# Johansen-Procedure #
######################

Estimation and testing under linear restrictions on beta

The VECM has been estimated subject to:
beta=H*phi and/or alpha=A*psi

     [,1]
[1,]    0
[2,]    1

Eigenvalues of restricted VAR (lambda):
[1] 0.2522 0.0000 0.0000

The value of the likelihood ratio test statistic:
0.93 distributed as chi square with 1 df.
The p-value of the test statistic is: 0.33

Eigenvectors, normalised to first column
of the restricted VAR:

                        [,1]
RK.G_Health.l1        1.0000
RK.Revenue_Total.l1  -7.3774
RK.constant         239.1869

Weights W of the restricted VAR:

       [,1]
[1,] 0.0000
[2,] 0.0382
```

We do not reject this restriction of the first adjusment coefficient being zero. So we conclude that the government revenue adjusts to disequilibria in the long-run. Meaning if expenditure for health is too high, the government is likely to increase their expenditure.

1.3 e)

[16]:
```
# Transform VEC to VAR with r = 1
var = vec2var(trace_test, r =1)
var$A
```

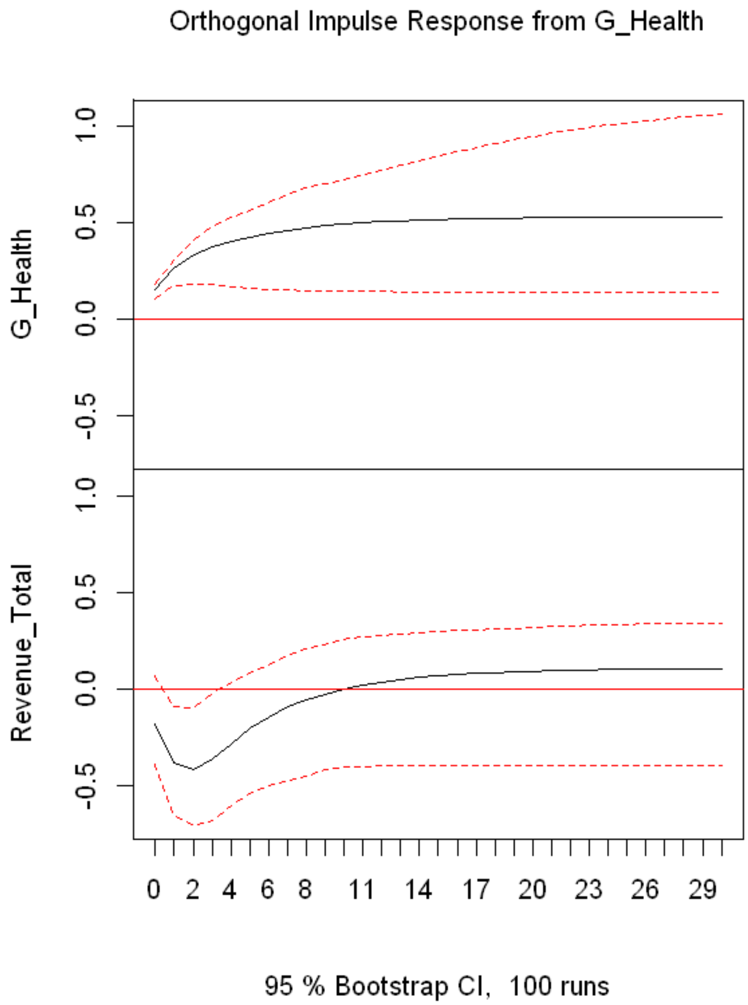
$A1

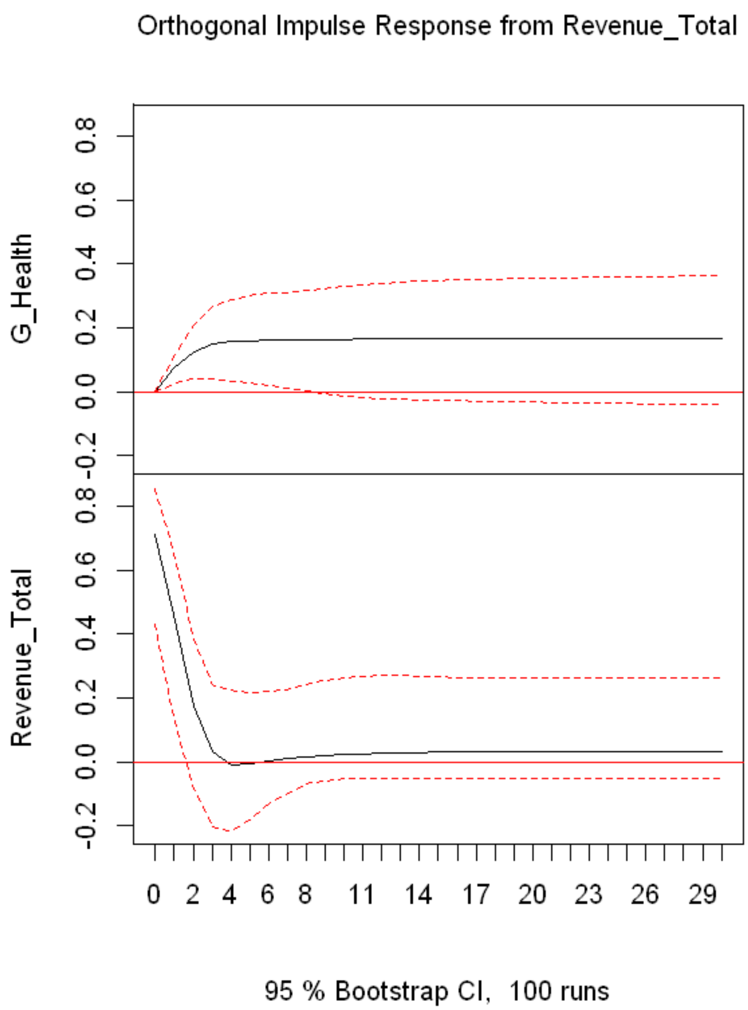
	G_Health.l1	Revenue_Total.l1
G_Health	1.831013	0.1046845
Revenue_Total	-1.665223	0.6489929

$A2

	G_Health.l2	Revenue_Total.l2
G_Health	-0.8355999	-0.0820683275
Revenue_Total	1.7364972	-0.0003861342

[17]:
```
# Obtain IRF
irf1 = irf(var, n.ahead = 30, ortho = TRUE)
plot(irf1)
```


Orthogonal Impulse Response from G_Health
G_Health
Revenue_Total
1.0
0.5
0.0
-0.5
0 2 4 6 8 11 14 17 20 23 26 29
95 % Bootstrap CI, 100 runs



A shock to government expenditure has a long-lasting and persistent effect on expenditure, while only temporarily significantly impacting revenues. If we have an increase in the expenditure in government health, there is a partially crowding out effect that potentially affects consumption and/or investment. Since a large share of the revenues are generated by VAT, this relation is quite reasonable from an economic perspective.

A shock to revenue has a short-run effect on revenue, as well as on health expenditure. The long-run responses are not significant. This kind of makes sense, since, when we have an increase in revenues, this will result in an increase of the money allocated to healthcare.

[18]:
```
# Estimate a VAR
var2 = VAR(data, p = 2)
```

Comparing the two results:

[19]:
```
var$A
summary(var2)
```

$A1

	G_Health.l1	Revenue_Total.l1
G_Health	1.831013	0.1046845
Revenue_Total	-1.665223	0.6489929

$A2

	G_Health.l2	Revenue_Total.l2
G_Health	-0.8355999	-0.0820683275
Revenue_Total	1.7364972	-0.0003861342

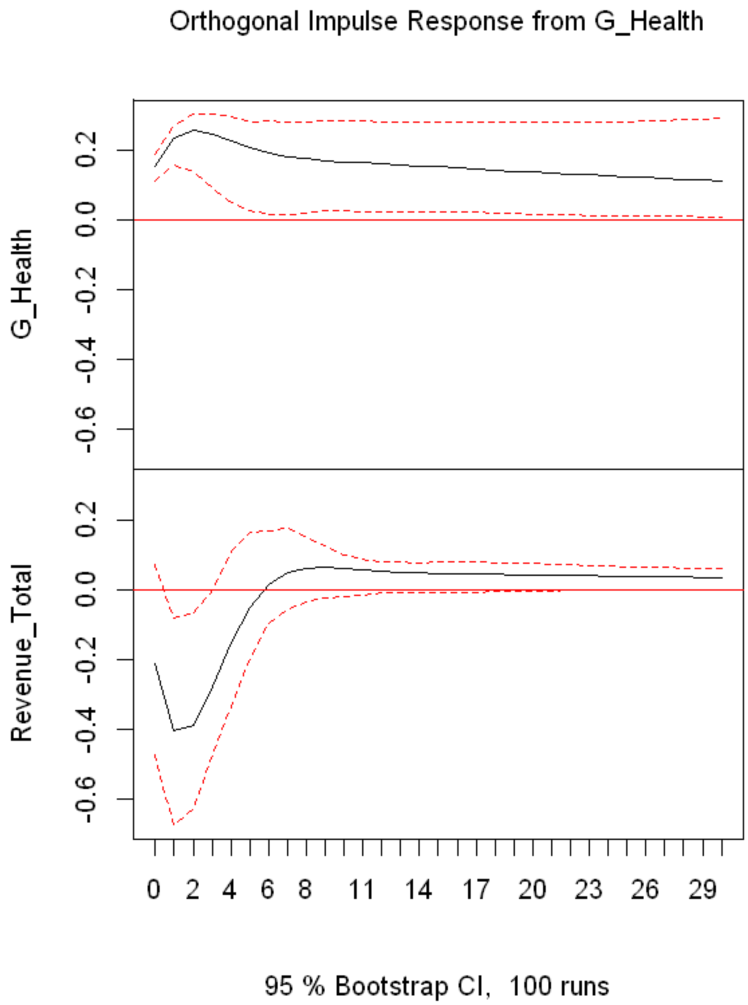
```
VAR Estimation Results:
=========================
Endogenous variables: G_Health, Revenue_Total
Deterministic variables: const
Sample size: 47
Log Likelihood: -26.944
Roots of the characteristic polynomial:
0.9802 0.6081 0.6081 0.1863
Call:
VAR(y = data, p = 2)


Estimation results for equation G_Health:
=========================================
G_Health = G_Health.l1 + Revenue_Total.l1 + G_Health.l2 + Revenue_Total.l2 + const

                 Estimate Std. Error t value Pr(>|t|)
G_Health.l1       1.63468    0.14798  11.046 5.29e-14 ***
Revenue_Total.l1  0.08841    0.03582   2.468   0.0177 *
G_Health.l2      -0.65571    0.14844  -4.417 6.88e-05 ***
Revenue_Total.l2 -0.04180    0.03576  -1.169   0.2491
const            -1.32496    0.62873  -2.107   0.0411 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


Residual standard error: 0.1554 on 42 degrees of freedom
Multiple R-Squared: 0.9943,Adjusted R-squared: 0.9938
F-statistic:  1843 on 4 and 42 DF,  p-value: < 2.2e-16


Estimation results for equation Revenue_Total:
==============================================
Revenue_Total = G_Health.l1 + Revenue_Total.l1 + G_Health.l2 + Revenue_Total.l2 + const

                 Estimate Std. Error t value Pr(>|t|)
G_Health.l1       -1.7203     0.7404  -2.324 0.025064 *
Revenue_Total.l1   0.6444     0.1792   3.596 0.000844 ***
```

```
G_Health.l2        1.7869     0.7426   2.406 0.020599 *
Revenue_Total.l2   0.0109     0.1789   0.061 0.951715
const             10.9226     3.1456   3.472 0.001209 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


Residual standard error: 0.7773 on 42 degrees of freedom
Multiple R-Squared: 0.6549,Adjusted R-squared: 0.622
F-statistic: 19.93 on 4 and 42 DF,  p-value: 2.924e-09



Covariance matrix of residuals:
              G_Health Revenue_Total
G_Health       0.02414      -0.03271
Revenue_Total -0.03271       0.60413

Correlation matrix of residuals:
              G_Health Revenue_Total
G_Health        1.0000       -0.2709
Revenue_Total  -0.2709        1.0000
```

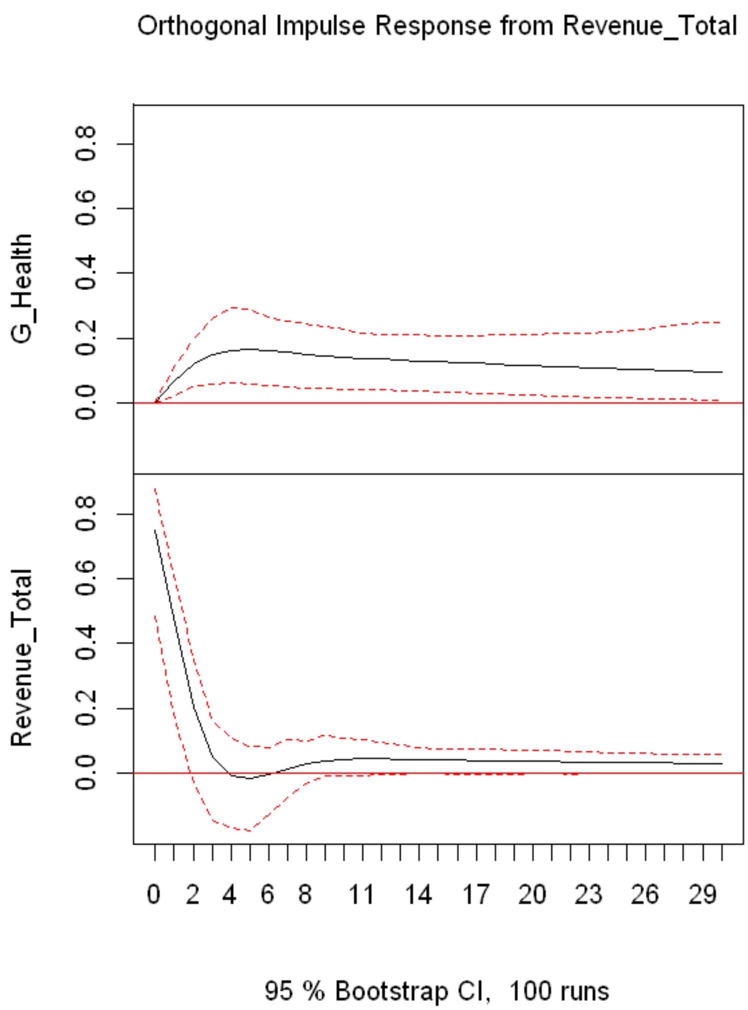
We find that the two methods provide us with slightly different results. This potentially arises since ca.jo and vars use different estimation methods.

[20]:

```
irf2 = irf(var2, n.ahead = 30, ortho = TRUE)
plot(irf2)
```

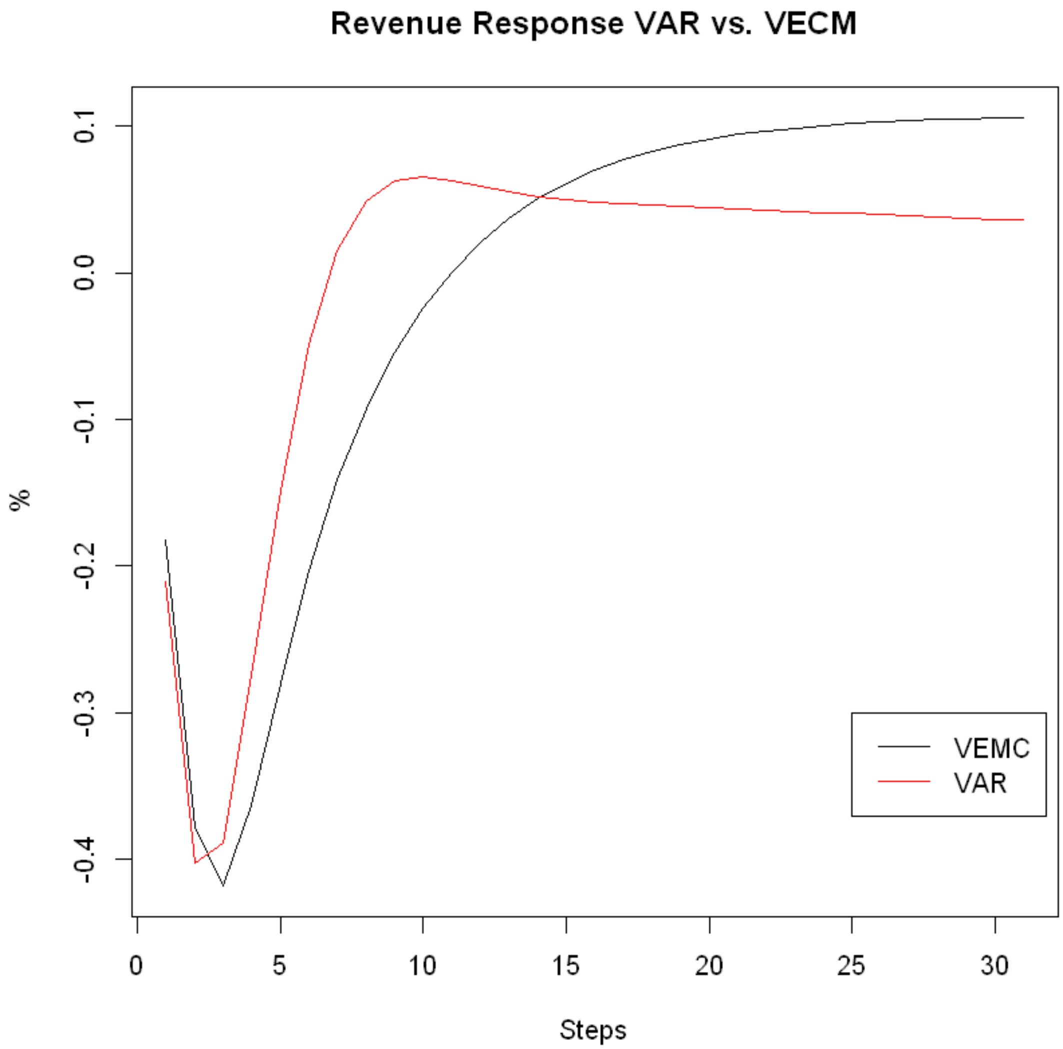


Orthogonal Impulse Response from G_Health
G_Health
Revenue_Total
0.2
0.0
-0.2
-0.4
-0.6
0 2 4 6 8 11 14 17 20 23 26 29
95 % Bootstrap CI, 100 runs



Let's do a comparison of the two.

[21]:
```
plot(irf1$irf$G[,2], type = "l", xlab = "Steps", main = "Revenue Response VAR␣
↪vs. VECM", ylab = "%")
lines(irf2$irf$G[,2], type = "l", col = "red")
legend(25, -0.3, legend = c("VEMC", "VAR"), lty = 1:1, col = c("black", "red"))
```



For both, we would expect in the short run to have a significant decrease in revenues. However, we would expect that future tax revenues will increase, although we do not have statistical evidence for this. In the short-run the pandemic will likely decrease the base of activity on which we can apply taxes to. On the long-run taxes need to be higher to cover the additional expenses.

1.4 2.

[22]:
```
library(quantmod)
```

```
Loading required package: xts
Loading required package: TTR
Version 0.4-0 included new data defaults. See ?getSymbols.
```

[23]:
```r
getSymbols("CPIAUCSL", src = "FRED")
getSymbols("CPILFESL", src = "FRED")
getSymbols("PCEPI", src = "FRED")
getSymbols("PCEPILFE", src = "FRED")
```

```
'getSymbols' currently uses auto.assign=TRUE by default, but will
use auto.assign=FALSE in 0.5-0. You will still be able to use
'loadSymbols' to automatically load data. getOption("getSymbols.env")
and getOption("getSymbols.auto.assign") will still be checked for
alternate defaults.

This message is shown once per session and may be disabled by setting
options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

'CPIAUCSL'

'CPILFESL'

'PCEPI'

'PCEPILFE'

[24]:
```r
CPI = window(CPIAUCSL, start = "1959-01-01", end = "2020-02-01")
CPIX = window(CPILFESL, start = "1959-01-01", end = "2020-02-01")
PCE = window(PCEPI, start = "1959-01-01", end = "2020-02-01")
PCEX = window(PCEPILFE, start = "1959-01-01", end = "2020-02-01")
```

1.5 a)

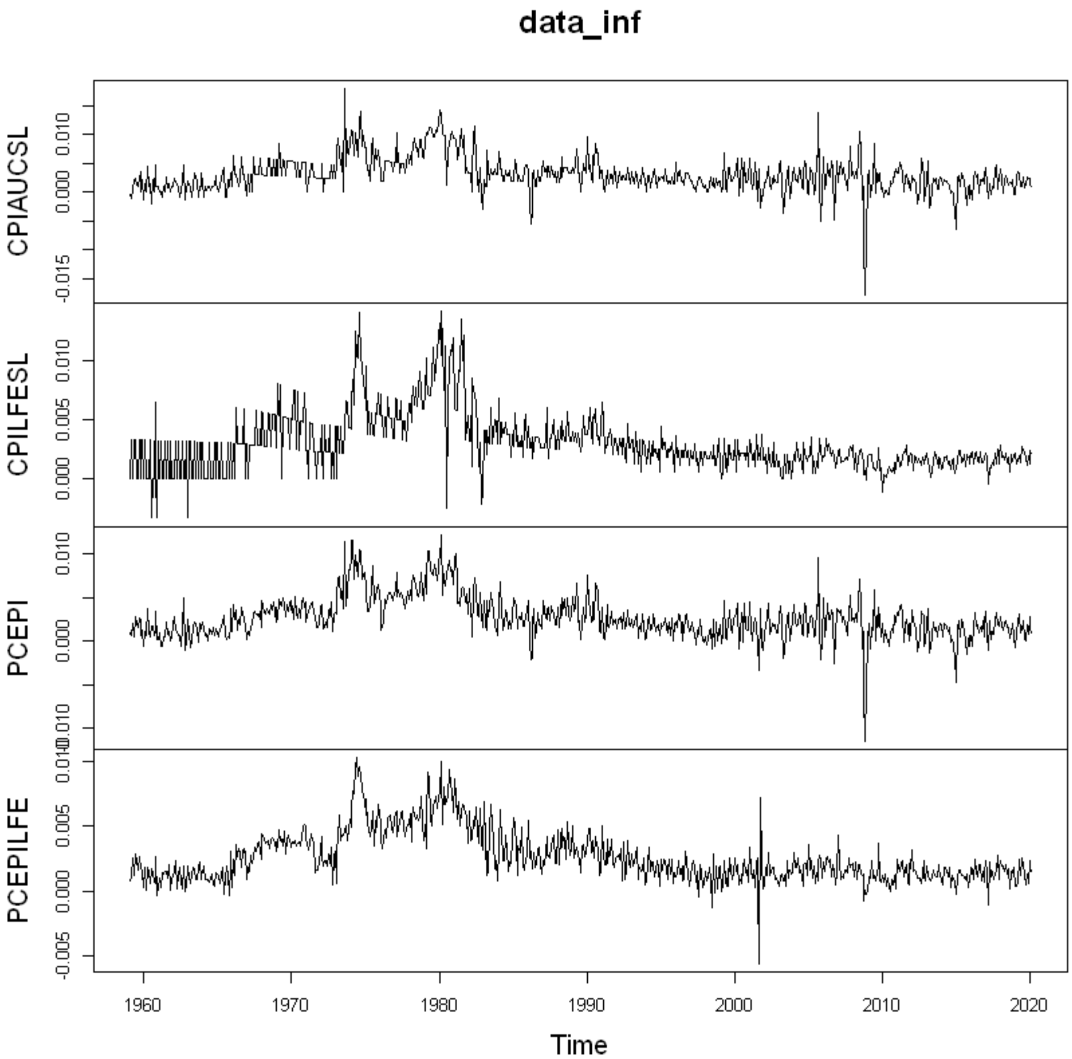
[25]:
```r
data = ts(data.frame(CPI, CPIX, PCE, PCEX), start = 1959, frequency = 12)
```

[26]:
```r
head(data)
```

CPIAUCSL	CPILFESL	PCEPI	PCEPILFE
29.01	29.9	16.074	16.727
29.00	29.9	16.089	16.740
28.97	30.0	16.100	16.759
28.98	30.0	16.132	16.801
29.04	30.1	16.140	16.822
29.11	30.2	16.186	16.871

[27]:
```r
data_inf = diff(log(data))
```

[28]:
```r
plot(data_inf)
```



1.6 b)

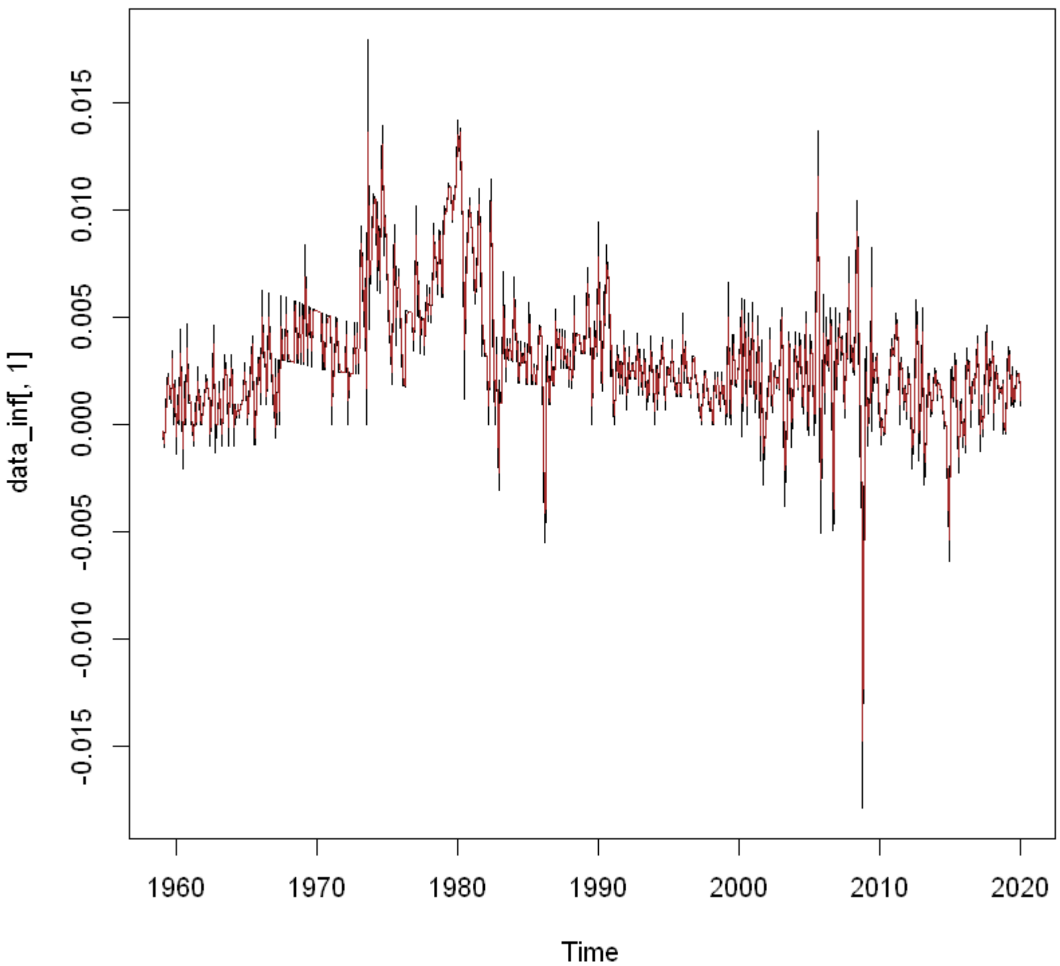
[29]:
```
library(dlm)
```

Warning message:
"package 'dlm' was built under R version 3.6.3"

[30]:
```
# Setting up the necessary estimates
dlm1 = dlm(FF= 1, V = 0.1, GG = 1, W = 0.2, m0 = 0, C0 = 0.5)
```

[31]:
```
kfilter = dlmFilter(data_inf[,1], dlm1)
```

[32]:
```
plot(data_inf[,1], type = 'l')
#$m for the expected value
lines(dropFirst(kfilter$m), type = 'l',pch = 20, col = "brown")
```



The Kalman filter uses a recursive approach to update the posterior estimate of the state. Given a prior believe (choosen prior, or updated prior from former iteration) about our state position, we predict where the state is supposed to be next period (predict state), and the uncertainty associated with it (variance of the prediction).

We use our prediction to estimate where our observable potentially is, given the information we have available up to now (prediction of observable). Associated with the prediction is the prediction error (MSE). We use the prediction to check it against the true observable. Given our law of motion for the states, we then estimate the optional value for the state, which gives rise to the Kalman gain. We then use the Kalman gain and the error of our predicition of the observable to update our priors.

Describing it intuitively: We use a guess on how the states develop to best approximate an observable. After checking for the true value of the observable, we update our state in a manner as to best forecast the observeable. The important part in this answer is to break down the intuition, not to elaborate on the math.

1.7 c)

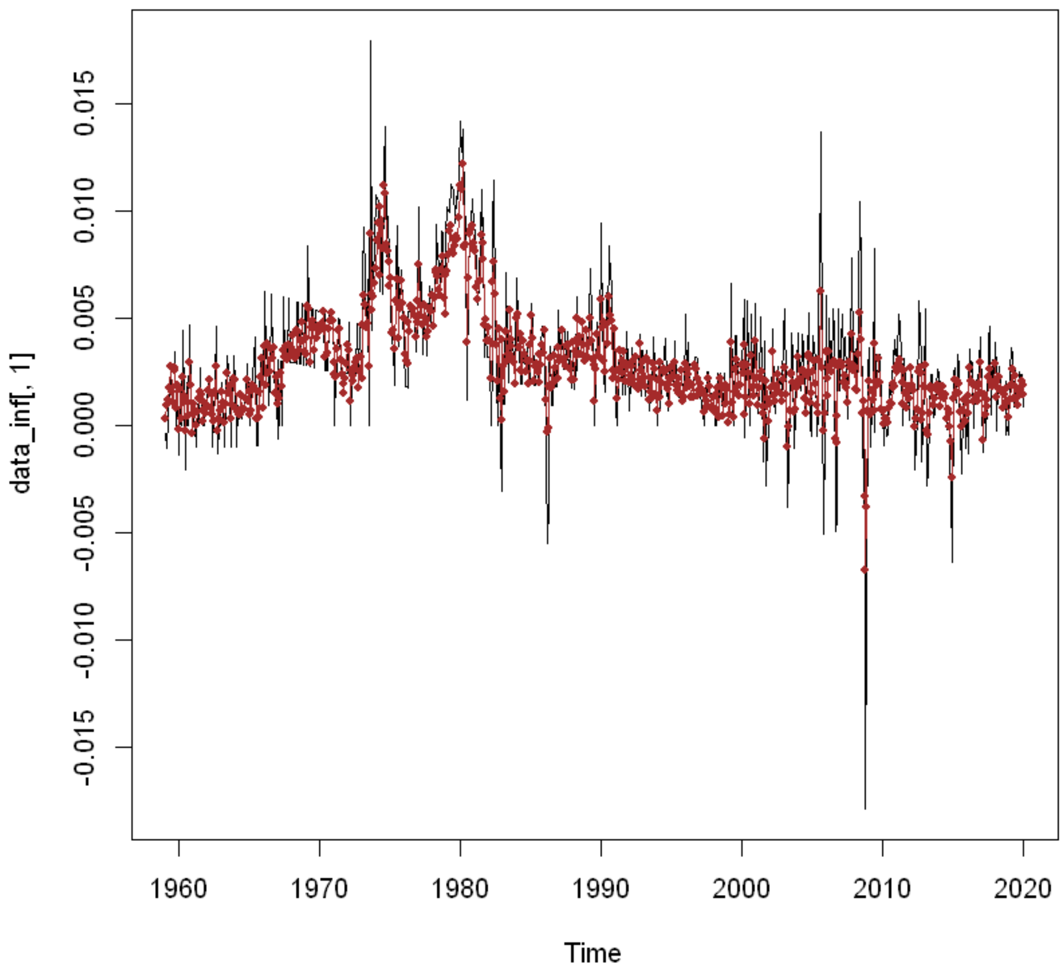
[33]:
```
FF = matrix(c(1,1,1,1), nrow = 4)
```

[34]:
```
V = diag(4)*0.1
V
```

0.1	0.0	0.0	0.0
0.0	0.1	0.0	0.0
0.0	0.0	0.1	0.0
0.0	0.0	0.0	0.1

[35]:
```
# Setting up the necessary estimates
dlm2 = dlm(FF= FF, V = V, GG = 1, W = 0.2, m0 = 0, C0 = 0.5)

kfilter2 = dlmFilter(data_inf, dlm2)
```

[36]:
```
plot(data_inf[,1], type = 'l')
#$m for the expected value
lines(dropFirst(kfilter2$m), type = 'o',pch = 20, col = "brown")
```

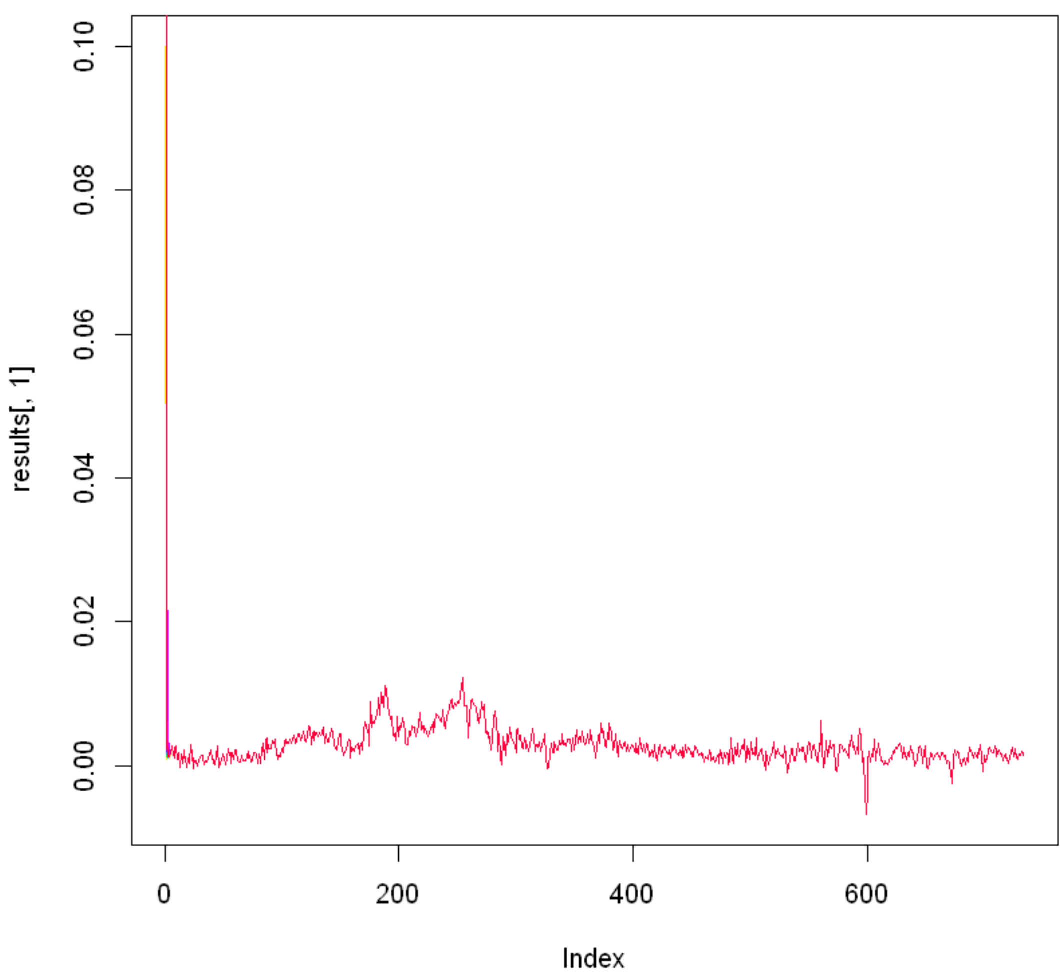


The filtered series using all four series is smoother than the filtered series with only one variable. This makes sense since the filtering with more variables contains more information which can be exploited for the prediction of the state variables.

Next, we want to check the influence of the prior on the result of the Kalman extraction.

[37]:
```
# Setup a loop to change the prior
means = c(0.1,0.2,0.3,0.4,0.5)
var = c(0.1, 1, 2, 3, 4)
results = matrix(nrow=length(kfilter2$m), ncol=25)
count = 1
for (x in means){
    for (y in var){
        FF = matrix(c(1,1,1,1), nrow = 4)
```

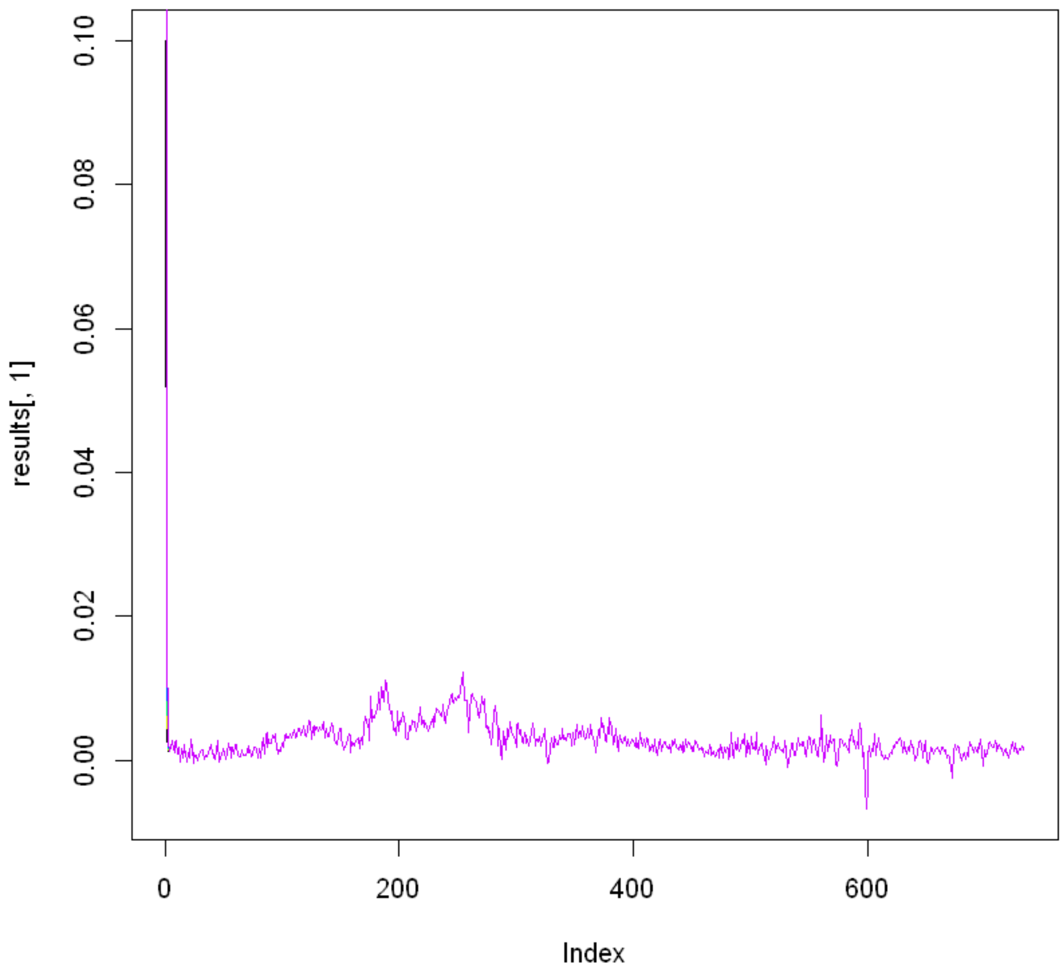
```
        V = diag(4)*0.1
        dlm2 = dlm(FF= FF, V = V, GG = 1, W = 0.2, m0 = x, C0 = y)
        kfilter2 = dlmFilter(data_inf, dlm2)
        results[,count] = kfilter2$m
        count = count + 1
    }
}

plot(results[,1], type = 'l')
color = rainbow(25)
for (x in 2:25){
    lines(results[,x], col = color[x])
}
```



Combining different priors together does not seem to influence the convergence of the Kalman filter. Let's investigate how different changes in the mean and the variance change the results.

[38]:

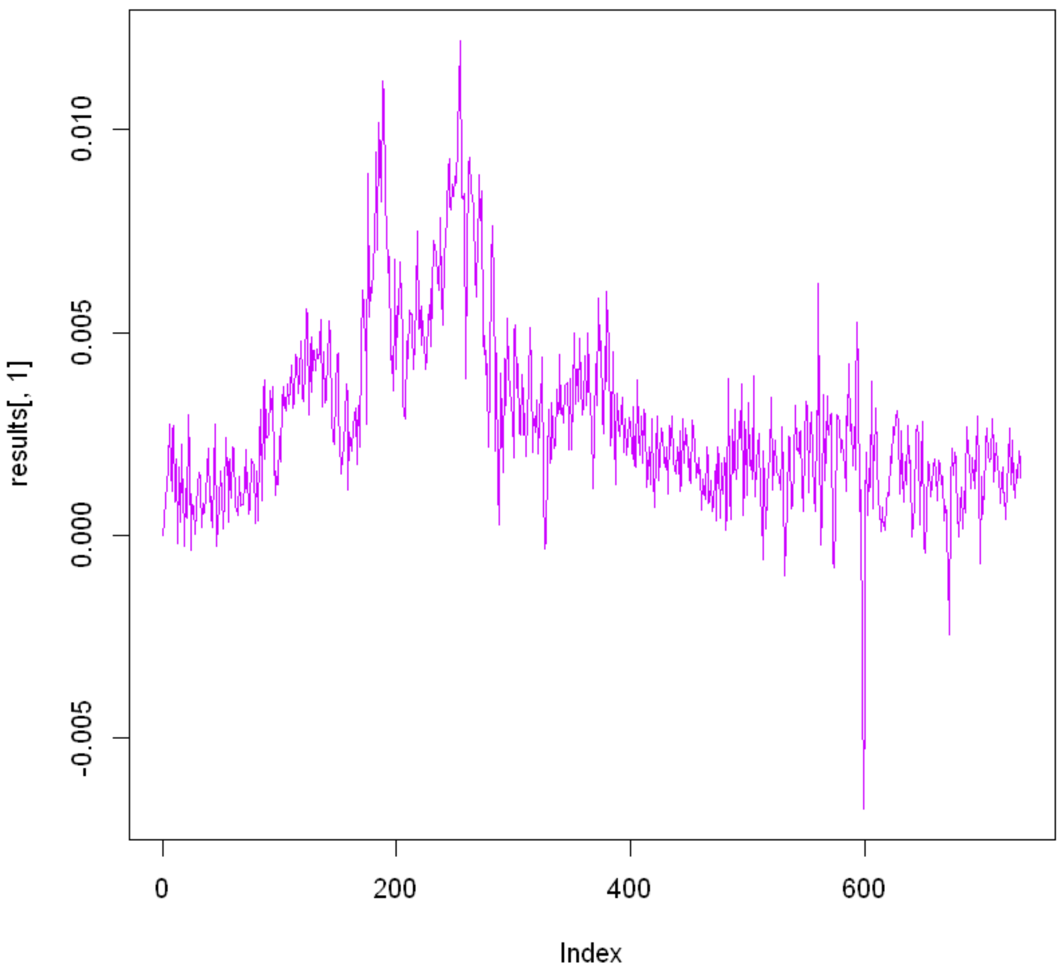
```
# Setup a loop to change the prior
means = c(0.1,0.2,0.3,0.4,0.5)
results = matrix(nrow=length(kfilter2$m), ncol=5)
count = 1
for (x in means){
    FF = matrix(c(1,1,1,1), nrow = 4)
    V = diag(4)*0.1
    dlm2 = dlm(FF= FF, V = V, GG = 1, W = 0.2, m0 = x, C0 = 0.5)
    kfilter2 = dlmFilter(data_inf, dlm2)
    results[,count] = kfilter2$m
    count = count + 1
}

plot(results[,1], type = 'l')
color = rainbow(5)
for (x in 2:5){
    lines(results[,x], col = color[x])
}
```



Different means simply shift up the first guess, but the series converges fast.

[39]:
```
# Setup a loop to change the prior
var = c(0.1, 1, 2, 3, 4)
results = matrix(nrow=length(kfilter2$m), ncol=5)
count = 1
for (y in var){
    FF = matrix(c(1,1,1,1), nrow = 4)
    V = diag(4)*0.1
    dlm2 = dlm(FF= FF, V = V, GG = 1, W = 0.2, m0 = 0, C0 = y)
    kfilter2 = dlmFilter(data_inf, dlm2)
    results[,count] = kfilter2$m
    count = count + 1
```

```
}

plot(results[,1], type = 'l')
color = rainbow(5)
for (x in 2:5){
    lines(results[,x], col = color[x])
}
```



Changing the variance of the prior basically does not generates a difference between the filtered series (there is a difference but it is numerically very small).

1.8 d)

[40]:
```
pca1 <- prcomp(data_inf, center=TRUE, scale.=TRUE)
```

[41]:
```
summary(pca1)
```

```
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.8038 0.6677 0.52647 0.15269
Proportion of Variance 0.8134 0.1115 0.06929 0.00583
Cumulative Proportion  0.8134 0.9249 0.99417 1.00000
```

[42]:
```
pca1 <- prcomp(data_inf, center=TRUE, scale.=TRUE, rank. = 1)
```

[43]:
```
true_inf = pca1$x
loadings = pca1$rotation
```

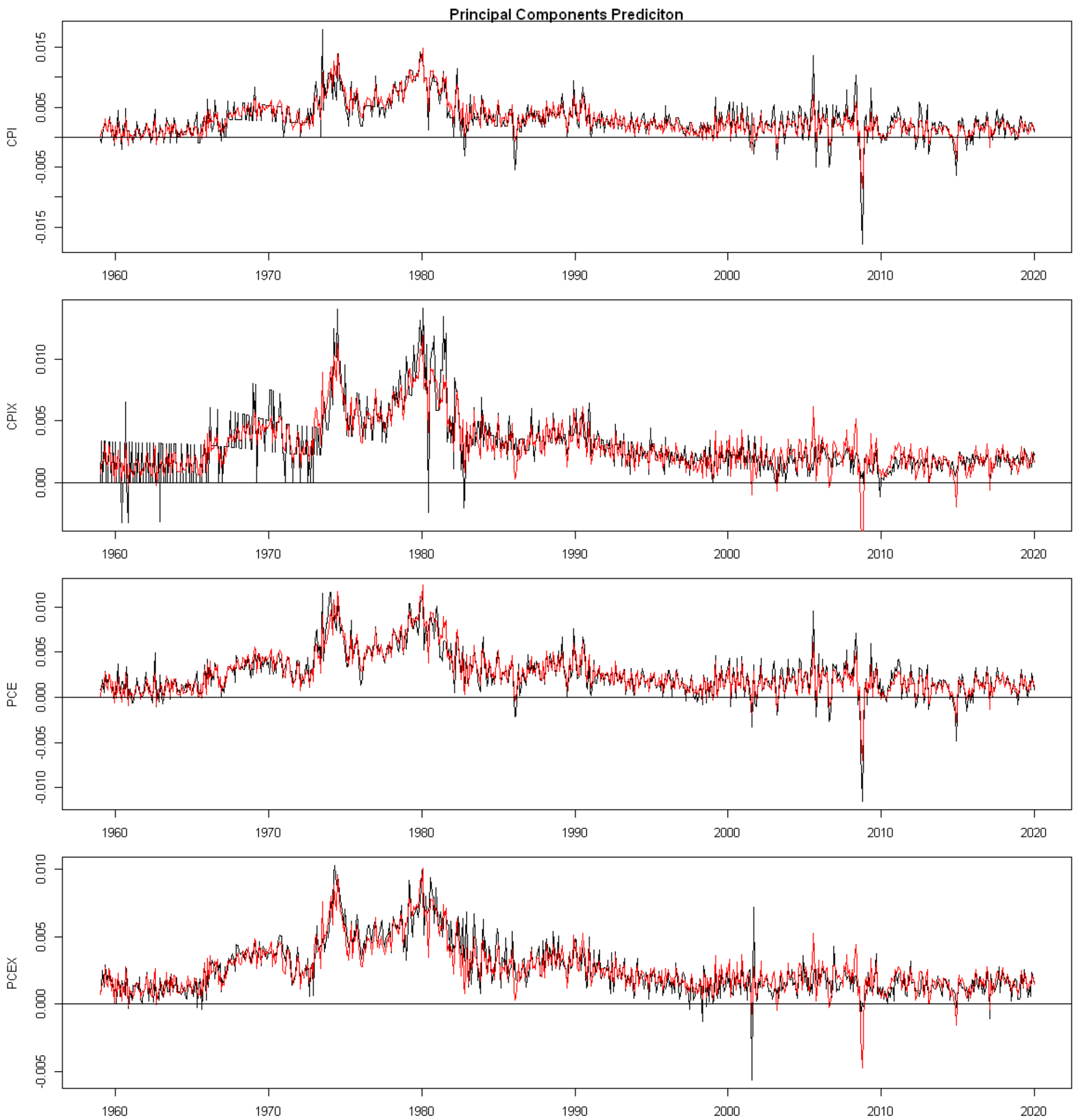
[44]:
```
pca1$scale
```

CPIAUCSL 0.00310166018883366 **CPILFESL** 0.00246921813039313 **PCEPI** 0.00245827958431651 **PCEPILFE** 0.00195976126711119

[45]:
```
# Setting up a matrix we fill below with the PCA
pred = matrix(nrow=length(data_inf), ncol=4)

# Filling the matrix with PCA
for (t in 1:length(data_inf)){
  pred[t,]=  t(pca1$rotation%*%pca1$x[t])
}
```

[46]:
```
library(repr)
options(repr.plot.width=10, repr.plot.height=10)

par(mfrow=c(4,1), mar=c(2,5,1,2)+0.1)
plot(ts(data_inf[,1], start=1959, deltat = 1/12), type='l', main='Principal␣
↪Components Prediciton', ylab='CPI')
lines(ts(pred[,1]*pca1$scale[1]+mean(data_inf[,1]),start=1959, deltat = 1/12),␣
↪type='l', col='red' )
abline(h=0)
plot(ts(data_inf[,2], start=1959, deltat = 1/12), type='l', ylab="CPIX")
lines(ts(pred[,2]*pca1$scale[2]+mean(data_inf[,2]),start=1959, deltat = 1/12),␣
↪type='l', col='red' )
abline(h=0)
plot(ts(data_inf[,3], start=1959, deltat = 1/12), type="l", ylab="PCE")
lines(ts(pred[,3]*pca1$scale[3]+mean(data_inf[,3]),start=1959, deltat = 1/12),␣
↪type='l', col='red' )
abline(h=0)
```

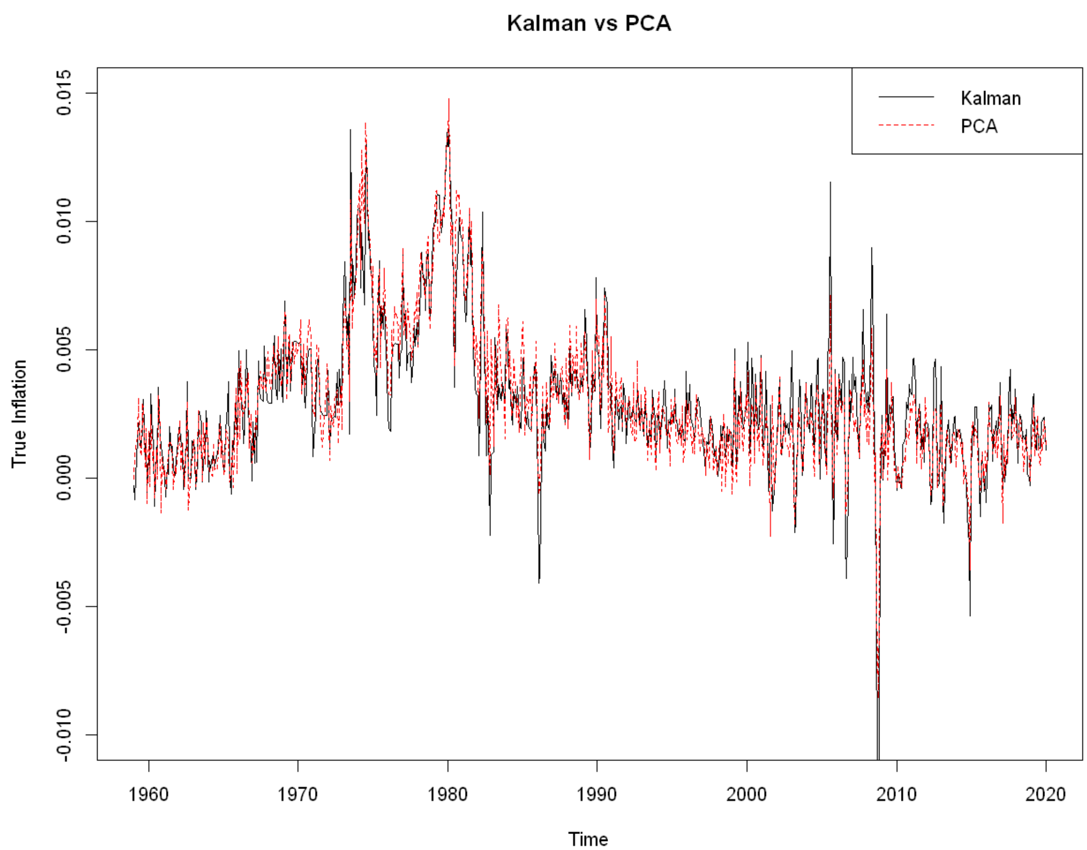
```r
plot(ts(data_inf[,4], start=1959, deltat = 1/12), type="l", ylab="PCEX")
lines(ts(pred[,4]*pca1$scale[4]+mean(data_inf[,4]),start=1959, deltat = 1/12),
 ↪type='l', col='red' )
abline(h=0)
```



Inspecting the series, we conclude that the series are well approximated by the PCA factor. The analysis can be complemented by investing the R^2 of the PCA regression on the single inflation factors.

1.9 e)

[47]:
```
options(repr.plot.width=10, repr.plot.height=8)
plot(ts(dropFirst(kfilter$m), start = 1959, frequency = 12), type = "l", main =␣
↪"Kalman vs PCA", ylab = "True Inflation",
    ylim = c(-0.01, 0.015))
lines(ts(pred[,1]*pca1$scale[1]+mean(data_inf[,1]),start=1959, deltat = 1/12),␣
↪type = "l",lty = 2, col = "red")
legend("topright", legend = c("Kalman", "PCA"), lty = 1:2, col = c("black",␣
↪"red"))
```



The PCA provides a reasonable approximation of the Kalman filtered series. In comparison with the real series, the Kalman filter provides a better approximation of the series.