ON HIGH-DIMENSIONAL MACROECONOMIC FORECASTING

CHRONAS VASILEIOS

CONTENTS

Abstract

This Thesis will present many different methods and techniques that aim to forecast Time-Series using an abundance of Data. Such techniques include Factor models and Penalized regressions. The Thesis starts with a presentation of some basics from many different relevant fields of statistics and then moves on to introduce factor models, penalized regressions and nowcasting. We then conduct Mote-Carlo simulations that compare those said techniques under different circumstances. Finally, we apply those techniques on McCracken's large Macroeconomic dataset. The Monte-Carlo simulations indicate that Principal Components do well, when the errors are not serially correlated. However, when the errors do suffer from serial correlation, Principal Component regression is outperformed by other methods, such as the Three Pass Regression Filter (3PRF) and Partial Least Squares. We also found that, when applied on the McCracken dataset, End-of-Sample Cross-Validated Ridge Regression outperforms all other methods.

Keywords: Forecasting, Time Series, Big Data, Factor Models, Penalized Regressions, Nowcasting

Introduction

Forecasting has long been one of the main Goals of Economics, since being able to accurately predict the future values of economic variables is of enormous importance to governments, firms and institutions. While traditional forecasting methods only used a handful of variables, mainly due to the limited availability of data and computing power, recent econometric techniques are able to handle large and complex datasets, taking advantage of increased computing power as well as easy access to huge datasets.

In the late 50's and 60's , structural macroeconomic forecasting, based on Keynesian economic theory, was prevalent. The main feature of structural macroeconomic models, is that they rely on economic theory to interpret the available data. Structured models are particularly useful when the behavior of the policymakers is taken into account.

Structural macroeconomic models begun falling out of favor in the early 1970's, due to the rising discontent with Keynesian theory. This fall from grace of structural forecasting cleared the way to non-structural macroeconomic forecasting, which begun to appear in this period. Non-structural forecasting uses very little, or even none at all, economic theory. Instead of building on theory to produce forecasts, non-structural forecasting is based on simple statistical extrapolations to produce forecasts. The emergence of non-structural forecasting was helped by a series of published articles in the early seventies, which showed that ''theory-free'' statistics-based forecasts were as good as forecasts made from large-scale Keynesian models. Non-structural forecasting models do well, if current policy remains unchanged.

The foundations of modern non-structural econometrics were laid in the 1920's. Slutsky and Yule argued that autoregressive processes constitute a potent explanatory framework for modelling economic and financial time series. Autoregressive processes are basically linear difference equations that are able to convert a random input into a serially-correlated output, an effect known as the Slutsky-Yule effect. While Slutsky and Yule developed the theoretical foundations of autoregressive processes, Kolmogorov and Wold further contributed to the literature. One of those contributions, the Wold representation of an Autoregressive process, was instrumental into making the Yule-Slutsky framework the cornerstone of time-series analysis. The next cornerstone in the field was when a very flexible and computationally cheap forecasting technique was created by Kalman. The now widely-known Kalman filter was originally used for the Apollo program, but did prove applicable to the field of economics and finance as well. Most of the aforementioned literature was made before non-structural forecasting really appeared in the 1970's. The collective knowledge that had built-up the years before was put together in Box and Jenkins groundbreaking book ''Time series analysis, forecasting and control'' published in 1970. The book introduces the idea of ARMA models, as well as estimation techniques, diagnostic checking and forecasting using ARMA models. Macroeconomists capitalized on the advancements of time-series analysis and an explosion of work followed. This work led to the use of VAR models, which is a very efficient model even to this day.

With the advent of enormous computational power, an abundance of data, a flourishing of new exciting literature, non-structural macroeconomic forecasting inevitably walks down the path of high-dimensional forecasting. As the Big Data revolution affects every aspect of the modern world, macroeconomics will not remain uninterrupted.

There are different classification of big data. The classification proposed by Doornik and Hendry (2015) is the most suitable for out macroeconomic framework. Doornik and Hendry's classification identifies three categories of big data.

1. 'Tall' data. 'Tall' data contains a very large number of observations, but not too many variables ($T \gg N$). Financial transactions data usually belongs to this category.

2. 'Fat' data. 'Fat' data is when the number of variables exceeds the sample size ($N \gg T$). This type of data is very common in macroeconomics. Especially when dealing with 'Traditional' macroeconomic variables, such as GDP, consumption etc. McCracken's dataset belongs to this category.

3. 'Huge' data. This type of data consists of both a huge sample size and a large set of variables.

This Thesis will predominantly deal with the second type of data. The main issue with 'fat' data is the 'The curse of dimensionality' phenomenon.

The curse of dimensionality was introduced by mathematician Richard Belmann. Belmann described the curse as when 'the number of samples needed to estimate an arbitrary function with a given level of accuracy grows exponentially with respect to the number of input variables (i.e., dimensionality) of the function'. In simpler terms, if we increase our model's number of input variables by two, then the data needed to estimate the model will not increase by two, but by a lot more. The "heart and soul" of High-Dimensional Econometrics is overcoming this curse.

.

Chapter 1

PREREQUISITE STATISTICAL THEORY

This section consists of different fields of statistical theory, that are essential to the techniques and methods presented in the thesis. This chapter starts with a thorough presentation of state-space models and the Kalman filter and goes on to discuss Markovian chains and some basic principles of Bayesian Statistics. All of those fields are used on different parts of High-Dimensional Time-Series Analysis and are hence a valuable addition to the Thesis.

1.1 STATE SPACE MODELS

In order to better understand a lot of estimation techniques used in Factor models, especially in the field of Nowcasting, it is important to become familiar with the state-space representation of time-series models and the statistical techniques associated

with this representation, such as the Kalman filter. The state-space representation is able to handle a plethora of different problems in time series. The basic idea is that the time-series model in question consists of an unobserved variable or vector of variables and a series of observations. The dynamic behavior of the variables through time is governed by an equation, which is called the state equation. The observed and the unobserved components of the model are related through the state-space equation.

Kalman filter

The Kalman filter was developed by Rudolf Kalman, and it was used in the Apollo mission. The goal of the Kalman filter is to optimally combine the information from two different source , the observations and the set of equations that govern the dynamic behavior of our variables, in order to infer about the properties of those variables. One of the most important advantages of the Kalman filter is that it is not computationally demanding. In order to better understand the Kalman filter, we need to first understand the idea behind the Kalman gain.

The following is a very simple intuitive explanation of the Kalman gain. The Kalman gain is used to determine how much of the new measurements we can use to update the new estimate. The Kalman gain is equal to

$$KG = E_{EST}/(E_{EST} + E_{MEA})$$

Where E_{ST} is the error in estimate (or error in the state equation) and E_{MEA} is the error in the measurments. The Kalman gain takes values between 0 and 1. A value of zero means that measurements are completely inaccurate and that we should only use our state equation to update our estimate for the variable's path. A Kalman Gain of 1 means that the noise in the state equation is very large and that we should only take into account the measurements to update our estimate. In general, the larger the Kalman gain, the more trustworthy, relative to the state equation, are our measurements.

The new estimate update is done according to this equation

$$EST_t = EST_{t-1} + KG(MEA - EST_{t-1})$$

Where EST_t is the estimate and MEA the measurement for the variable under study at time t. The error in the estimate is also re-calculated at each point in time. The corresponding equation is

$$E_{EST\,t} = \frac{E_{MEA}E_{EST\,t-1}}{E_{MEA} + E_{EST\,t-1}} =) E_{EST\,t} = (1 - KG)E_{EST\,t-1}$$

If the Kalman gain is large, that means the error in the measurement is very small, which means that new data put in can very quickly get us to the true value and therefore the error in the estimate will diminish very quickly. If the Kalman gain is very small, the error in the measurement is large and we thus slowly converge to the true value, as data comes in.

A more mathematically robust explanation of the Kalman filter, which is based on Durbin, Koopman (2012) follows.

Derivation of the Kalman gain in a local level model.

In order to approach the concept of the Kalman gain and the Kalman filter in a more compact mathematical way, we consider the following model. The unobserved variable, whose state we are interested in, follows a random walk.

$$x_{t+1} = x_t + \eta_\tau \tag{1.1.1}$$

Our set of observations is given by

$$y_t = x_t + e_\tau\,, \quad t = 1,2, \dots \dots \dots T \tag{1.1.2}$$

We assume that $\eta_t \sim N(0, \sigma_\eta^2)$ and $e_t \sim N(0, \sigma_e^2)$. The model is not stationary. Our goal is to first derive the Kalman gain and then use it to infer the properties of the state x_t.

Each time a new observation y_t becomes available, we incorporate into the system in order to produce optimal predictions on the current and future state of x_t. We set $Y_{t-1} = (y_1, y_2, y_3 \cdots \cdots y_{t-1})'$. $t = 2,3 \dots$..and assume that the conditional distribution of x_t given Y_{t-1} is $N(\chi_t, P_t)$, where χ_t and P_t are known. When y_t becomes available, the set of observation that belong to the information set is $Y_t = (y_1, y_2, y_3 \cdots \cdots y_t)'$. Then the distribution of x_{t+1} given Y_t is $N(\chi_{t+1}, P_{t+1})$. χ_{t+1} and P_{t+1} are the one-step ahead predictions of the state and the process variance respectively, given Y_t. We also need to define the distribution of x_t given Y_t, which basically tells us how our knowledge of the present has been updated, now that the new observation has come in. We assume $x_t/Y_t \sim N(\chi_{t/t}, P_{t/t})$.

We define the one step ahead prediction error as $v_t = y_t - \chi_t$, (1.1.3). Before the observation y_t, we know that $E(x_t/Y_{t-1}) = \chi_t$. Hence we can see that the one step ahead prediction error represents how far away is the observation from the expected current state.

$$\begin{aligned} E(v_t|Y_{t-1}) &= E(y_t - \chi_t|Y_{t-1}) = E(x_t + e_t - \chi_t|Y_{t-1}) \\ &= E(x_t|Y_{t-1}) + E(e_t|Y_{t-1}) - E(\chi_t|Y_{t-1}). \end{aligned} \tag{1.1.4}$$

χ_t is known , thus $E(\chi_t|Y_{t-1}) = \chi_t$. $E(x_t|Y_{t-1}) = \chi_t$ by definition. $E(e_t|Y_{t-1}) = 0$

Thus $E(v_t|Y_{t-1}) = \chi_t - \chi_t = 0$

$Var(v_t|Y_{t-1}) = Var(x_t + e_t - \chi_t|Y_{t-1})$

$Var(\chi_t) = 0$ since χ_t is known. Thus

$Var(u_t|Y_{t-1}) = Var(x_t + e_t|Y_{t-1}) = Var(x_t|Y_{t-1}) + Var(e_t|Y_{t-1}) = P_t + \sigma_e^2$

Since $Var(x_t|Y_{t-1}) = P_t$ by definition.

$$Var(u_t|Y_{t-1}) = P_t + \sigma_e^2 \tag{1.1.5}$$

We see that the are two sources of uncertainty about the one step ahead prediction error, the process variance P_t and the error in the measurements σ_e^2.

Based on a regression lemma for the bivariate normal distribution, we get the following result

$$p(x_t|Y_t) = N\left(\chi_t + \frac{P_t}{P_t + \sigma_e^2} v_t, \frac{P_t \sigma_e^2}{P_t + \sigma_e^2}\right) \quad (1.1.6)$$

Which is the distribution of x_t when a new observation comes in (that is why are information set includes all observations up until time t).

The Kalman gain is equal to $K_t = \frac{P_t}{P_t + \sigma_e^2}$.

Thus, the prediction for the current state, given that the observation at time t has been made, is

$$x_{t|t} = E(x_t|Y_t) = \chi_t + K_t v_t \quad (1.1.7)$$

The predicted current process variance is

$$P_{t|t} = K_t \sigma_e^2 \quad (1.1.8)$$

The prediction for the next period state and process variance is

$$x_{t+1|t} = E(x_{t+1}|Y_t) = E(x_t + \eta_t|Y_t) = E(x_t|Y_t) = \chi_t + K_t v_t \quad (1.1.9)$$

Which is the same as the current predicted state. This result stems from the fact that our information set is the same and that the variable does not exhibit any autoregressive behavior.

$$P_{t|t+1} = Var(x_{t+1}|Y_t) = Var(x_t + \eta_t|Y_t) = P_{t|t} + \sigma_\eta^2 =)$$

$$P_{t|t+1} = K_t \sigma_e^2 + \sigma_\eta^2 \quad (1.1.10)$$

The uncertainty that stems from the measurement is added into our process variance prediction for $t + 1$, since the measurement for $t + 1$, y_{t+1} has not yet arrived and is not part of the observation set Y_t.

Equations (1.1.3), (1.1.7), (1.1.8), (1.1.9), (1.1.10) constitute the Kalman filter for the simple local model.

State smoothing in the local level model

While we have developed the theory under which we predict the state variable at time t and $t + 1$, using information only available at time t, Kalman state Smoothing is used to estimate the state variable for $t = 1,2,3 \ldots \ldots T$ by taking into account the whole sample Y_T. We have that

$$x_t|Y_T \sim N(\hat{x}_t, V_t)$$

Where $\hat{x}_t = E(x_t|Y_T)$ and $V_t = Var(x_t|Y_T)$

The whole sample Y_T is known , which is equivalent with Y_{t-1} and all forecast errors, from time t up to the end of the sample, being known. Thus $\hat{x}_t = E(x_t|Y_T)$ is equivalent to $\hat{x}_t = E(x_t|Y_{t-1}, v_t, v_{t+1}, \ldots v_T)$. By the regression Lemma

$$\hat{x}_t = \chi_t + \sum_{j=t}^{T} cov(x_t, v_j) Var\left(v_j|Y_{t-1}\right)^{-1} v_j \quad (1.1.11)$$

We set $F_t = Var(v_t|Y_{t-1})$ and the state estimation error $h_t = x_t - \chi_t$. Equation (1.1.11) becomes $\hat{x}_t = \chi_t + \sum_{j=t}^{T} cov(x_t, v_j) F_j^{-1} v_j$

The variance of the state estimation error is $Var(h_t) = Var(x_t + \chi_t) = Var(x_t) = P_t =) Var(h_t) = P_t$

The covariance of the state variable with the forecast errors is equal to $cov(x_t, v_j) = cov(h_t, v_j)$, for $j = t, \dots \dots \dots . T$.

Thus $cov(x_t, v_t) = cov(h_t, v_t) = E(h_t(h_t + e_t)) = Var(h_t) = P_t$

And $cov(h_t, v_{t+1}) = E(h_t(h_{t+1} + e_{t+1})) = E(h_t(B_t h_t + \eta_t - K_t e_t)) = P_t B_t$

Where $B_t = 1 - K_t$

Consequently

$cov(h_t, v_{t+T}) = P_t B_t B_{t+1} \dots . . B_{T-1}$

Having obtained the above result, we now can proceed to finalize our result for (1.1.11)

$$\hat{x}_t = \chi_t + \frac{P_t v_t}{F_t} + \frac{P_t B_t v_{t+1}}{F_{t+1}} + \frac{P_t B_t B_{t+1} v_{t+2}}{F_{t+2}} + \cdots P_t B_t B_{t+1} \dots \frac{B_{T-1} v_T}{F_T}$$

We define $d_{t-1} = \frac{v_t}{F_t} + \frac{B_t v_{t+1}}{F_{t+1}} + \frac{B_t B_{t+1} v_{t+2}}{F_{t+2}} + \cdots (B_t B_{t+1} B_{t+2} \dots B_{T-1} V_T)/F_T$ and we get

$$\hat{x}_t = \chi_t + P_t d_{t-1} \qquad (1.1.12)$$

We can easily see that

$$d_{t-1} = \frac{v_t}{F_t} + B_t d_t \qquad (1.1.13)$$

$d_T = 0$ since there is no sample after time T. Hence, through backwards recursion and the use of (1.13), we are able to compute the values of d_{t-1}.

The equations (1.12) and (1.13) together are used to obtain the smoothed values of the state variable.

The smoothed state variance is equal to

$V_t = Var(x_t|Y_T) = Var(x_t|Y_{t-1}, v_t, v_{t+1}, \dots v_T) =)$

$V_t = P_t - \sum_{j=t}^{T} [cov(x_t, v_j)]^2 F_j^{-1}$

By substituting the expression for the covariance

$$V_t = P_t - \frac{P_t^2}{F_t} - \frac{P_t^2 B_t^2}{F_{t+1}} - \cdots - (P_t^2 B_t^2 B_{t+1}^2 \dots B_{T-1}^2)/F_T = P_t - P_t^2 D_{t-1} \qquad (1.1.14)$$

Where $D_{t-1} = \frac{1}{F_t} + \frac{B_t^2}{F_{t+1}} + \frac{B_t^2 B_{t+1}^2}{F_{t+2}} + \cdots . . \frac{B_t^2 B_{t+1}^2 B_{t+2}^2 \dots B_{T-1}^2}{F_T}$. We also observe that

$$D_{t-1} = \frac{1}{F_t} + B_t^2 D_t \qquad (1.1.15)$$

$D_T = 0$ because there is no observation after time t. Consequently we can use relations (1.1.13) and (1.1.14) to obtain the smoothed state variance.

Linear State Space models

We extend the results of the simple local level to the multivariate case. We have the linear Gaussian state space model. The observed variable is y_t which is of dimension $p \times 1$ and the state variable is the m −dimensional vector x_t.

$$y_t = Z_t x_t + \varepsilon_t, \qquad \varepsilon_t \sim N(0, H_t) \tag{1.1.16}$$

$$x_{t+1} = T_t x_t + R_t \eta_t, \quad \eta_t \sim N(0, Q_t) \tag{1.1.17}$$

$$x_1 \sim N(\chi_1, P_1) \tag{1.1.18}$$

Just like in the local level model, we want to derive the conditional distribution of x_t and x_{t+1} given the information set $Y_{t-1} = (y_1, y_2, \ldots . y_{t-1})'$. We set the one-step ahead forecast of y_t given Y_{t-1}, $v_t = y_t - E(y_t|Y_{t-1}) = y_t - E(Z_t x_t + \varepsilon_t|Y_{t-1}) = y_t - Z_t \chi_t$.

Given that the current observation y_t has not yet been made available, the value of the one-step ahead forecast error is 0.

$$\begin{aligned} E(v_t|Y_{t-1}) &= E(y_t - Z_t\chi_t|Y_{t-1}) = E(Z_t x_t + \varepsilon_t - Z_t\chi_t|Y_{t-1}) \\ &= E(\varepsilon_t|Y_{t-1}) = 0 \end{aligned} \tag{1.1.19}$$

If the current observation becomes available, then the observation set is Y_t. This is equivalent to knowing Y_{t-1} and the one-step ahead forecast error v_t. Thus $x_{t|t} = E(x_t|Y_t) = E(x_t|v_t, Y_{t-1})$.

From the regression lemma.

$$x_{t|t} = E(x_t|Y_{t-1}) + Cov(x_t, v_t)[Var(v_t)]^{-1}(v_t - E(v_t|Y_{t-1}))$$

Since $E(v_t|Y_{t-1}) = 0$,

$x_{t|t} = E(x_t|Y_{t-1}) + Cov(x_t, v_t)[Var(v_t)]^{-1} v_t$

$Cov(x_t, v_t) = E(x_t v_t'|Y_{t-1}) = E(x_t(y_t - Z_t\chi_t)'|Y_{t-1}) =$
$E(x_t(Z_t x_t + \varepsilon_t - Z_t\chi_t)'|Y_{t-1}) = E(x_t(x_t - \chi_t)'Z_t'|Y_{t-1}) = P_t Z_t'$

We define $F_t = Var(v_t|Y_{t-1}) = Var(Z_t x_t + \varepsilon_t - Z_t\chi_t|Y_{t-1}) = Z_t P_t Z_t' + H_t$

Then

$$x_{t|t} = \chi_t + \left[\frac{P_t Z_t'}{Z_t P_t Z_t' + H_t}\right] v_t =) x_{t|t} = \chi_t + P_t Z_t' F_t^{-1} v_t \tag{1.1.20}$$

By the Lemma we have

$$P_{t|t} = Var(x_t|Y_t) = Var(x_t|Y_{t-1}) - Cov(x_t, v_t)[Var(v_t)]^{-1}Cov(x_t, v_t)' =)$$

$$P_{t|t} = P_t - P_t Z_t' F_t^{-1} Z_t P_t \quad (1.1.21)$$

Where $x_{t|t}$ and $P_{t|t}$ are the current predicted state and current predicted variance, given that the observation for time t is known. Just like in the simple local model, there are two sources of uncertainty, the error in the measurements (represented by the matrix H_t) and the process variance of the state variable x_t. The predictions are:

$x_{t+1|t} = E(T_t x_t + R_t \eta_\tau | Y_t) = T_t E(x_t|Y_t) + R_t E(\eta_t|Y_t)$, since $E(\eta_t|Y_t) = 0$ we have $x_{t+1|t} = T_t E(x_t|Y_t) =)$

$$x_{t+1|t} = T_t x_{t|t} \quad (1.1.22)$$

From (1.19) we have $x_{t+1|t} = T_t(\chi_t + P_t Z_t' F_t^{-1} v_t) =)$

$$x_{t+1|t} = T_t \chi_t + T_t P_t Z_t' F_t^{-1} v_t \quad (1.1.23)$$

The Kalman gain K_t is equal to $T_t P_t Z_t' F_t^{-1} v_t$, thus (1.1.22) becomes

$$x_{t+1|t} = T_t \chi_t + K_t v_t$$

The predicted process variance for period $t + 1$ is

$P_{t+1|t} = Var(T_t x_t + R_t \eta_\tau | Y_t) = T_t Var(x_t|Y_t) T_t' + R_t Q_t R_t'$. Since $Var(x_t|Y_t) = P_{t|t}$, from (1.1.20) we get

$$P_{t+1|t} = T_t(P_t - P_t Z_t' F_t^{-1} Z_t P_t) T_t' + R_t Q_t R_t' =)$$

$$P_{t+1|t} = T_t P_t (T_t - K_t Z_t)' + R_t Q_t R_t' \quad (1.1.24)$$

The recursions (1.1.19), (1.1.21), (1.1.20) and (1.1.23) constitute the Kalman filter for the linear state space model.

Smoothing in multivariate linear state space models

The process is similar to the one in the simple local level model, as is the intuition behind the process.

We define h_t as the state estimation error, equal to $h_t = x_t - \chi_t$ and $B_t = T_t - K_t Z_t$

The smoothed state vector is equal to

$$\widehat{x_t} = E(x_t|Y_T) = \chi_t + \sum_{j=t}^{T} cov(x_t, v_j) F_j^{-1} v_j \quad (1.1.25)$$

Where $cov(x_t, v_j) = E(x_t v_j' | Y_{t-1}) = E\left(x_t (Z_j h_j + e_j)' \middle| Y_{t-1}\right) = E(x_t h_j' | Y_{t-1}) Z_j'$, for $j = t, \dots \dots T$.

It can be proven that $E(x_t h_j' | Y_{t-1}) = P_t B_t' B_{t+1}' B_{t+2}' \dots . B_{T-1}'$. Hence, $\widehat{x_t} = \chi_t + P_t Z_t' F_t^{-1} v_t + P_t B_t' Z_{t+1}' F_{t+1}' v_{t+1} + \dots + (P_t B_t' \dots \dots B_{T-1}' Z_T' F_T' v_T)$

We set $d_{t-1} = Z_t' F_t^{-1} v_t + B_t' Z_{t+1}' F_{t+1}' v_{t+1} + \dots + (B_t' \dots \dots B_{T-1}' Z_T' F_T' v_T)$ and (1.25) becomes

$$\widehat{x_t} = \chi_t + P_t d_{t-1}$$

(1.1.25) along with $d_{t-1} = Z_t' F_t v_t + B_t' d_t$, (1.1.24) allows us to solve the system recursively for $t = T \ldots .1$. This is the state smoother for the multivariate linear state space model.

By following the same procedure, we obtain the smoothed state variance matrix. We define

$D_{t-1} = Z_t' F_t^{-1} Z_t + B_t' Z_{t+1}' F_{t+1}^{-1} Z_{t+1} B_t +$
$\cdots (B_t' \ldots B_{T-1}' Z_T' F_T^{-1} Z_T B_{T-1} \ldots B_t')$ (1.1.26).

By skipping some algebra, the smoothed state variance $V_t = Var(x_t|Y_T)$ is proven to be equal to

$V_t = P_t - P_t D_{t-1} P_t$ (1.1.27)

We also have $D_{t-1} = Z_t' F_t^{-1} Z_t + B_t' D_t B_t$ (1.1.28)

We use (1.1.27), (1.1.26) and the fact that $D_T = 0$ to recursively obtain the smoothed state variance.

Dynamic factor models in a state-space form

We consider the usual factor model, where f_t is the factors, Λ is the factor loadings, e is the noise and x_t is the demeaned dataset.

$$x_t = \Lambda f_t + e_t$$

We assume that the noise is normally distributed and are mutually independent.

$e_t \sim N(0, \Sigma_e)$

The factors follow a vector autoregressive process with p lags.

$f_t = \varphi_1 f_{t-1} + \varphi_2 f_{t-2} + \cdots \varphi_p f_{t-p}$

The VAR process of the factors is the equivalent of the state equation, while the factor representation is the measurement equation. In some instances, lags of the factors can also enter the measurement equation or the errors can themselves exhibit autoregressive behavior. The state-space representation is able to deal with these instances.

Regression Lemma

We have two jointly distributed random vectors, x and y. Their joint distribution is assumed normal. The derivation of the kalman filter is based, just like in the simple local level, on the lemma below.

Multivariate case.

$$E\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}, \qquad Var\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{xy}', & \Sigma_{yy} \end{pmatrix}$$

The distribution of x given y is $P(x|y) = N(\mu_x + \Sigma_{xy}\Sigma_{yy}^{-1}(y - \mu_y), \Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{xy}')$

Univariate case.

$$E(x|y) = E(x) + cov(x,y)[var(y)]^{-1}(y - E(y))$$

1.2 Markov-Switching Models

A very common phenomenon in macroeconomic time-series analysis is temporal ins tability. When the dataset spans several decades, it is normal to expect that the underlying relationship in the model might have changed. There are several ways to capture this shift in the model's coefficients, but we will focus on markov-switching. In Markov-Switching time-series models, the variables or the variable's coefficients are assumed to be driven by Markov chains.

Markov chains where developed by Russian Mathematician Andrey Markov. Markov chains are processes where the outcome of the current experiment is affected by the previous experiment's outcome. Markov proved that those processes have a steady state in the long run, in an effort to humiliate his arch enemy, mathematician and theologian Pavel Nekrasov.

In order to get a thorough understanding of a Markov-Switching model. We consider the following univariate autoregressive process.

$$x_t = a_0 + a_1 s_t + \beta_1 x_{t-1} + \beta_2 x_{t-2} + \cdots . \beta_p x_{t-p} + e_t \qquad (1.2.1)$$

The model is an $AR(p)$ with a normally distributed error of zero mean and a variance equal to σ_e^2. The state variable s_t takes the values of either 1 or 0 ($i = 0,1$) and follows a first-order Markovian Chain. The Markovian transition matrix is

$$P = \begin{bmatrix} P(s_t = 0|s_{t-1} = 0) & P(s_t = 1|s_{t-1} = 0) \\ P(s_t = 0|s_{t-1} = 1) & P(s_t = 1|s_{t-1} = 1) \end{bmatrix} = \begin{bmatrix} p_{00} & p_{10} \\ p_{01} & p_{11} \end{bmatrix} \qquad (1.2.2)$$

Where $P(s_t = 0|s_{t-1} = 0)$ is the probability that s_t is zero given that it was equal to zero as well in the last period etc. Since $p_{00} + p_{01} = 1$ and $p_{10} + p_{11} = 1$ we will need to estimate two parameters and not four.

The variable s_t determines the regime switching in the (1.1) model. In this particular model it shifts the long run mean. The transition probabilities determine the persistence of each regime.

Model estimation

We will illustrate how to estimate the model using Quasi-Maximum Likelihood. We denote the information set available at time t as $\Omega_t = \{x_1, x_2, x_3, \ldots . x_t\}$ and the parameter vector $\theta = (\alpha_o, \alpha_1, \beta_1, \ldots \beta_p, \sigma_e^2, p_{00}, p_{11})$. There are three kinds of forecasts of the state variable s_t, according to the information available. The first kind is the prediction probabilities $P(s_t = i|\Omega_{t-1}; \theta)$, which are the probability that the state variable will have a certain value, given information available at the last period and the parameter vector. We also define in a similar way the filtering probabilities $P(s_t = i|\Omega_t; \theta)$ and the smoothing probabilities $P(s_t = i|\Omega_T; \theta)$ (where the full sample is considered known).

As usual, we make the normality assumption. The conditional density is

$$f(x_t|\Omega_{t-1}, s_t = i; \theta) = \frac{1}{(2\pi\sigma_e^2)^{\frac{1}{2}}} e^{\left\{-\frac{(x_t-\alpha_o-\alpha_1 i-\beta_1 x_1-\cdots\beta_p x_{t-p})^2}{2\sigma_e^2}\right\}} \quad (1.2.3)$$

If the value of the state variable at time t is not known, but the prediction probability is known, then we have

$$\begin{aligned} f(x_t|\Omega_{t-1};\theta) = {} & P(s_t = 1|\Omega_{t-1};\theta) f(x_t|\Omega_{t-1}, s_t = 1;\theta) \\ & + P(s_t = 0|\Omega_{t-1};\theta) f(x_t|\Omega_{t-1}, s_t = 0;\theta) \end{aligned} \quad (1.2.4)$$

The conditional density given that s_t takes a certain value is weighted by the probability that s_t has taken this certain value. As the observation at time t becomes known and our information set is updated to Ω_t from Ω_{t-1}. The probability of s_t being equal to i is updated using Bayes theorem.

$$P(s_t = i|\Omega_t, \theta) = \frac{P(s_t = i|\Omega_{t-1}, \theta) f(x_t|s_t = i, \Omega_{t-1};\theta)}{f(x_t|\Omega_{t-1};\theta)} \quad (1.2.5)$$

Intuitively, we can think that if x_t that is observed at time t has a very high value, then it is more likely that $s_t = 1$. The state variable is driven by a Markov chain, hence, we have

$$P(s_{t+1} = i|\Omega_t;\theta) = p_{0i} P(s_t = 0|\Omega_t;\theta) + p_{1i} P(s_t = 1|\Omega_t;\theta) \quad (1.2.6)$$

Equations (1.2.3), (1.2.4), (1.2.5) ad (1.2.6) are used recursively (along with an initial value for the prediction probability). Through those equations we obtain the conditional densities and the filtering probabilities for $t = p, p+1, \ldots . T$. The Quasi-Likelihood function is

$$L(\theta) = \frac{1}{T}\sum_{t=1}^{T} \ln(f(x_t|\Omega_{t-1};\theta)) \quad (1.2.7)$$

The $\hat{\theta}_{QMLE}$ can be obtained by a maximization algorithm. We also need to estimate the smoothing probabilities, which are useful for figuring out which regime occurs at different periods of time. We have that

$$P(s_t = i|s_{t+1} = j, \Omega_t;\theta) = \frac{p_{ij} P(s_t = i|\Omega_t;\theta)}{P(s_{t+1} = j|\Omega_t;\theta)} \quad (1.2.8)$$

The intuition behind (1.2.8) is more clear if we think that we know at which regime we are going to be at time $t+1$, which is regime j, and want to figure out how likely is that we were at state i at time t. If the transition probabilities p_{01} and p_{10} are high and transition probabilities p_{00} and p_{11} are low, then it is more likely that we were at different state last period than the one we are now. If the opposite is true, then it is more likely that we were at the same regime last period. We can calculate the smoothing probability, using (1.2.8) and the following expression.

$$\begin{aligned} P(s_t = i|\Omega_T;\theta) = {} & P(s_{t+1} = 0|\Omega_T;\theta) P(s_t = i|s_{t+1} = 0, \Omega_T;\theta) \\ & + P(s_{t+1} = 1|\Omega_T;\theta) P(s_t = i|s_{t+1} = 1, \Omega_T;\theta) = \end{aligned} \quad (1.2.9)$$

$$P(s_t = i|\Omega_t;\theta)\left[\frac{p_{i0}P(s_{t+1}=0|\Omega_T;\theta)}{P(s_{t+1}=0|\Omega_t;\theta)}+\frac{p_{i1}P(s_{t+1}=1|\Omega_T;\theta)}{P(s_{t+1}=1|\Omega_t;\theta)}\right]$$

Using relations (1.2.5), (1.2.6) and (1.2.9) we can solve the system backwards and get the smoothing probabilities.

We will revisit Markov-Switching later.

1.3 Bayesian approach to statistics

While the frequentist approach to statistics remains dominant in the literature, the Bayesian approach has been gaining a lot of momentum when it comes to high-dimensional techniques.

The frequentist approach treats the parameters of interest, as fixed constants, one whose real value we are trying to discover. On the other hand, the Bayesian approach treats the parameter as a variable with a distribution. This is due to the fact that in the Bayesian world, the focus is on subjective uncertainty. Hence, even though the parameter we are looking for might have a real fixed value, this value is hidden from us, consequently we can assign a distribution at this hidden parameter in order to represent our uncertainty. This even allows us to insert our own prior beliefs about this parameter into the model. As new information comes in the form of a sample, we update our beliefs. This update is done according to the Bayes Theorem.

Bayes Theorem

Bayes theorem is one of the most celebrated formulas in statistics. Its most generic form is this:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

In order to get familiar with Bayesian terminology, we consider the following interpretation of Bayes theorem:

$$P(\theta|data) = \frac{P(data|\theta)P(\Theta)}{P(data)}$$

We have the parameter that we seek to estimate and our data. $P(\theta)$, which is the aforementioned distribution of the parameter, is called the prior distribution. $P(\theta|data)$ is called the posterior distribution. $P(data|\theta)$ is the very familiar likelihood and $P(data)$ can be thought of as a normalizing constant. It is often convenient to express the posterior distribution as:

$$P(\theta|data) \propto P(data|\theta)P(\Theta)$$

The symbol $\propto$ stands for proportionality. The posterior has the same shape with the prior multiplied by the likelihood, with the difference being that the latter does not integrate to one.

We usually choose priors that have a convenient form, so that the posterior is manageable. Those forms that are considered convenient and yield a posterior distribution belonging to the same family of distributions as the prior, are called

conjugate priors. The posterior distribution is a combination of the information that we get from the sample and the prior and it is basically the new updated distribution of our parameter.

While in the frequentist approach, we derive a point estimate for our parameter, in the Bayesian approach we derive a distribution instead of a point estimate. This distribution is simply the posterior. There are several ways to obtain a point estimate out of the posterior, we can consider the mean of the distribution as the point estimate or the mode (the highest point of the distribution). Those two points are not equivalent, when the distribution is skewed. We can also construct credible intervals, which is the Bayesian equivalent of confidence intervals. In order to uniquely identify the credible interval, we also introduce the notion of the highest density interval HDI , which is the interval that contains (for example) 95% of the distribution's density and no point outside the interval has a highest probability than the points inside the interval. The notion of HDI is used in cases where the posterior is skewed. Of course, when the posterior is a normal distribution, due to the fact that normal distributions are symmetrical, the credible interval can be identified just like the confidence interval.

Bayesian Linear Regression

In order to get a better sense of the Bayesian approach, we will take a look at how Bayesians treat the classic linear regression problem.

The explained variable y is drawn from a normal distribution.

$$y \sim N(\beta' X, \sigma^2 I)$$

We assume that the variance is homoscedastic and it is known. The vector β is $k-$dimensional. The aim is to derive the posterior distribution. Bayes formula in linear regression looks like this:

$$P(\beta|y,X) = \frac{P(y,X|\beta)P(\beta)}{P(y,X)}$$

Where $P(\beta|y,X)$ is the posterior, $P(\beta)$ is the prior and $P(y,X|\beta)$ is the likelihood. $P(y,X)$ is of course the normalizing constant that makes sure the posterior distribution's integral is equal to one. The normalizing term is equal to

$$P(y,X) = \int_{-\infty}^{+\infty} P(y,X|\beta)P(\beta)d\beta$$

It is often the case that calculating this integral is difficult, hence, a technique called Monte Chain Monte Carlo is exploited.

Our prior $P(\beta)$ can incorporate previous belief or knowledge about the parameter β into the model. As the sample size increases though, the likelihood tends to overcome the prior and the Bayesian model converges to classic OLS.

Markov Chain Monte Carlo

We will briefly discuss the intuition behind MCMC methods and how they are used in Bayesian analysis. The algorithm we will focus on is called Gibbs sampling. As we have already seen, in a Markovian system, each state only depends on the previous state, which makes the MCMC algorithm less cumbersome computationally. The idea behind Gibbs sampling is to sample only from the conditional distributions in order to

approximate the joint multivariate posterior distribution, hence bypassing the computational impasse. The Gibbs sample algorithm is the following:

1. Assign random values to $\beta_0(0), \beta_1(0), \dots \beta_k(0)$.
2. Sample one value from $f\left(\beta_0 \middle| y, \beta_1(0), .. \beta_k(0)\right)$ obtain $\beta_0(1)$.
3. Sample one value from $f\left(\left(\beta_1 \middle| y, \beta_0(1), .. \beta_k(0)\right)\right.$, obtain $\beta_1(1)$.
4. So forth for all $\beta_j, j = 1 \dots k$.
5. Sample one value from $f\left(\beta_0 \middle| y, \beta_1(1), .. \beta_k(1)\right)$, obtain $\beta_0(2)$
6. So forth for all $\beta_j, j = 1 \dots k$.
7. Repeat the process for l steps.

We may have to discard the first values of the algorithm, since they are way too dependent on the randomly drawn initial values. The resulting samples will be distributed according to the posterior. The Gibbs sampling algorithm can only be used when the conditional distributions are known.

Chapter 2

FACTOR MODELS

One of the most important models in High-Dimensional Forecasting is the Statistical Factor Model. The factors may or may not exhibit dynamic behavior. The model is based on the classical factor models that were first established by psychologists and became widely used in other fields as well. The method was introduced in the Economics literature by Geweke (1977). The basic idea behind the Factor models is that a few latent variables account for much of the variability in the dataset. Thus, by estimating those factors, we can overcome the curse of dimensionality without sacrificing predictive power. Sargent and Sims (1977), as well as Stock and Watson (1989) found out that major Macroeconomic variables are influenced by a few unobserved factors, thus, Factor models have been proven to be particularly useful in Macroeconomics, especially since there is an abundance of available indicators but sample sizes are usually small.

Assume X_t is a multivariate time series. We assume that X_t admits a factor model representation.

$$X_t = \Lambda F_t + e_t \tag{2.1}$$

F_t is an r-dimensional vector of the factors, e_t is the $(N \times 1)$ vector of the idiosyncratic disturbances and Λ is the $(N \times r)$ matrix of factor loadings. We can see

that, there is a source of common variation for every variable in X_t (the factors). The idiosyncratic components on the other hand, represent the variable specific variation. We assume that the factors are orthogonal to each other. Based on our assumptions about the errorse_t , factor models can be distinguished into exact factor and approximate factor models. If, on the other hand, the errors are correlated, then we have an approximate factor model. If the idiosyncratic components display serial correlation, then lags of the factors enter the equation and we get a dynamic factor model.

$$X_t = \Lambda(L)F_t + e_t$$

2.1 Estimation of Factor models

There are plenty of estimation techniques that are used based on the type of factor model that we have to deal with. For static exact factor models, where f_t and e_t are independent Gaussian processes, Maximum Likelihood is used. Since the orthogonal factor representation is not unique, we impose the restriction that the variance of the factors is the identity matrix of dimensions $r \times r$ (r being the number of factors). That is $Var(f_t) = I_r$. Using Maximum Likelihood we get estimates for Λ and Σ_e (where Σ_e is the covariance matrix of the idiosyncratic components). Having obtained the estimate for Λ, $\hat{\Lambda}$ and the estimate for $\widehat{\Sigma_e}$, one of the most common formulas to estimate the factors (among others) is

$$\widehat{f_t} = \hat{\Lambda}'\left(\hat{\Lambda}\hat{\Lambda}' + \hat{\Sigma}_e\right)^{-1} x_t \tag{2.1.1}$$

If the model on our hands is a dynamic factor model, we can utilize the state-space representation in order to proceed with the estimation. We consider the following model, where the factors follow a $VAR(p)$ process and the predictors matrix is driven by m lags of the factors.

$$X_t = \Lambda_0 f_t + \Lambda_1 f_{t-1} + \cdots \Lambda_m f_{t-m} + e_t \tag{2.1.2}$$

$$f_t = \Phi_1 f_{t-1} + \cdots . \Phi_p f_{t-p} + \varepsilon_t \tag{2.1.3}$$

The state-space representation is able to capture a variety of cases. The measurement equation is the $X_t = \Lambda F_t + e_t$ with $\Lambda = (\Lambda_0, \Lambda_1 \ldots \Lambda_m)$ and $F_t = (f_t', f_{t-1}', \ldots f_{t-m}')$'

1) If $m \geq p - 1$ then the state-equation is

$$\begin{bmatrix} f_t \\ \vdots \\ f_{t-p+1} \\ \vdots \\ f_{t-m} \end{bmatrix} = \begin{bmatrix} \Phi_1 & \Phi_2 & \cdots & \Phi_p & 0 & \ldots & 0 \\ I & 0 & \ldots & \ldots & \ldots & \ldots & 0 \\ 0 & I & 0 & \ldots & \ldots & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ldots & \ldots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ & & & \ddots & \ddots & \ddots & \\ 0 & \ldots & \cdots & \ldots & 0 & I & 0 \end{bmatrix} \begin{bmatrix} f_{t-1} \\ f_{t-2} \\ \vdots \\ f_{t-p} \\ \vdots \\ f_{t-m-1} \end{bmatrix} + \begin{bmatrix} I \\ 0 \\ \vdots \\ 0 \end{bmatrix} \varepsilon_t \tag{2.1.4}$$

2) If $m < p - 1$ then the state equation is the following

$$\begin{bmatrix} f_t \\ f_{t-1} \\ \vdots \\ f_{t-p+1} \end{bmatrix} = \begin{bmatrix} \Phi_1 & \Phi_2 & \cdots & \Phi_p \\ I & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & I & 0 \end{bmatrix} \begin{bmatrix} f_{t-1} \\ f_{t-2} \\ \vdots \\ f_{t-p} \end{bmatrix} + \begin{bmatrix} I \\ 0 \\ \vdots \\ 0 \end{bmatrix} \varepsilon_t \tag{2.1.5}$$

3) The state-space representation can also capture the case where the idiosyncratic components are serially-correlated. Assume that $m = 1, p = 1$ and that e_t follows an $AR(1)$ process ($e_{it} = de_{it-1} + \varepsilon_t$). Then the measurement equation is

$$X_t = \Lambda_0 f_t + \Lambda_1 f_{t-1} + e_t$$

The state equation is

$$\begin{bmatrix} f_t \\ f_{t-1} \\ e_t \end{bmatrix} = \begin{bmatrix} \Phi_1 & \Phi_2 & 0 & 0 \\ I & 0 & 0 & 0 \\ 0 & 0 & D & 0 \end{bmatrix} \begin{bmatrix} f_{t-1} \\ f_{t-2} \\ e_{t-1} \end{bmatrix} + \begin{bmatrix} I & 0 \\ 0 & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} u_t \\ \varepsilon_\tau \end{bmatrix} \tag{2.1.6}$$

With $D = diag(d_1, \dots . d_n)$.

The state-space representation allows us to use the Kalman filter for computing the Likelihood function for any value of the parameters. The Likelihood function can be maximized using the EM algorithm.

2.2 Principal Components

Principal component Analysis is one of the most widely used methods of estimating Factor models, especially when the number of dimensions is large and Maximum Likelihood estimation is hence computationally expensive. The method was first used by Chamberlain and Rothschild (1983) on an approximate static factor model.

Before applying PCA, the dataset X_t (of dimensions $n \times k$) is centered. We assume that there are m common components driving the variation in X_t, with $m \ll k$ Our model is the following:

$$X_t = \Lambda F_t + e_t$$

We add the restriction of $\frac{\hat{F}_t' \hat{F}_t}{T} = I_r$, since as stated before, the orthogonal factor representation is not unique. The restriction of $\hat{\Lambda}\hat{\Lambda}'/N = I_r$ can also be used instead.

PCA works by applying spectral decomposition to the variance-covariance matrix of X_t. Spectral decomposition is the process by which a symmetric matrix is decomposed into its eigenvalues and eigenvectors. More specifically, if we have a $n \times n$ matrix A which is positive-definite and all of its eigenvalues are positive and real, then the following expression applies

$$A = EBE'$$

Where B is a diagonal matrix that contains all the eigenvalues of A while E is a $n \times n$ matrix of the corresponding eigenvectors. If $VAR(X_t) = \Sigma_\chi$, then we apply PCA by decomposing $\hat{\Sigma}_\chi$, where $\hat{\Sigma}_\chi$ is the estimated variance-covariance matrix of Σ_χ.

$$\hat{\Sigma}_\chi = \hat{E}\hat{B}\hat{E}'$$

$\hat{E}$ is the matrix containing the eigenvectors of $\hat{\Sigma}_\chi$ and $\hat{B}$ contains the eigenvalues of $\hat{\Sigma}_\chi$. Let the eigenvalue-eigenvector pair be $(\widehat{b_1}\ \widehat{e_1}), (\widehat{b_2}\ \widehat{e_2}) \ldots\ldots (\widehat{b_k}\ \widehat{e_k})$, with $\widehat{b_1} \geq \widehat{b_2} \geq \cdots \widehat{b_k} \geq 0$. The factor loadings are then equal to

$$\hat{B} = \left[\sqrt{\widehat{b_1}}\widehat{e_1}, \sqrt{\widehat{b_2}}\widehat{e_2}, \ldots\ldots \sqrt{\widehat{b_k}}\widehat{e_k}\right]$$

The factors are then obtained by running a regression. The exact formula is

$$\widehat{F_t} = \frac{X\hat{\Lambda}'}{n}$$

We then proceed to estimate the idiosyncratic components and the variance-covariance matrix of the idiosyncratic components.

$$\widehat{e_t} = x_t - \hat{\Lambda}\hat{f}_t$$

$\widehat{\Sigma_e} = \widehat{\Sigma_\chi} - \hat{\Lambda}\hat{\Lambda}'$ (which is obtained after we take into account the restriction of $\frac{\hat{F}_t{}'\hat{F}_t}{T} = I_r$). Stock and Watson (2002) proved that those estimators are consistent.

It is worth noting, that the procedure described above is equivalent to sloving the following least squares problem

$$\min_{\Lambda,F} \frac{1}{nT}\sum_{t=1}^{T}(x_t - \Lambda f_t)'(x_t - \Lambda f_t) \qquad (2.2.1)$$

2-Step estimator

The two-step estimator was conceived by Doz et al. (2011) and it is used when the factors exhibit dynamic behavior. The idiosyncratic terms are also allowed to serially-correlated. To be more specific, we consider a factor representation as in (2.1) and we assume that the factors follow a $VAR(m)$, as in 2.1.3. At the first step, the loadings and the factors are estimated by PCA, the idiosyncratic components are obtained from $\hat{e}_{it} = x_{it} - \widehat{\lambda}'_i\hat{f}_t$ and the sample variance of the idiosyncratic components is also computed and denoted by $\hat{\psi}_{ii}$. The estimated factors are cast into a VAR and the estimates $\widehat{\Phi}_j, j = 1 \ldots m$ are obtained.

In the second step, the model is cast into a state-space form as in (2.1.4). The noise in the VAR is assumed to have an identity matrix variance and the variance of the idiosyncratic components is assumed to be a diagonal matrix, with the diagonal elements being the ones estimated from the first step, $\widehat{Var(e_t)} = diag\{\widehat{\psi_{11}}, \widehat{\psi_{22}}, \ldots . \widehat{\psi_{nn}}\}$. In other words, the idiosyncratic components are taken as non-correlated (even if this might not be the case). We construct the vector $\hat{\theta}$, which contains the first step estimates and then run a Kalman smoother on the data (as described in section **1.1**). The factors are computed as $\hat{f}_{t|T} = E(f_t|x_1, x_2, \ldots . x_T, \hat{\theta})$.

Doz et al (2011) prove under their assumptions, that the 2-step factor estimates are consistent.

Estimating the number of factors.

The accurate estimation of the number of factors is crucial to the correct application of factor models. Many researchers, when doing empirical analysis, often choose the number of factors based on the needs of analytical convenience, rather than the number determined by the data. The first major breakthrough in the literature was by Bai and Ng (2002) who proposed a criterion determining the number of static factors. Their proposed information criterion is of the form

$$IC(r) = \ln(V_r) + rg(N,T) \tag{2.2.2}$$

V_r is the (2.2.1) and $g(N,T)$ is a penalty function that satisfies $g(N,T) \to 0$ and $\min[\sqrt{N},\sqrt{T}]\, g(n,T) \to +\infty$ as $N,T \to +\infty$. Bai and Ng considers the following forms of the penalty function $g(N,T)$.

$$g(n,T) = \frac{r\hat{\sigma}^2(N+T)}{NT}\ln(\frac{NT}{N+T})$$

$$g(n,T) = \frac{r\hat{\sigma}^2(N+T)}{NT}\ln(C_{NT}^2)$$

$$g(n,T) = \frac{r\hat{\sigma}^2\ln(C_{NT}^2)}{C_{NT}^2}$$

Where $C_{NT} = \min(\sqrt{N},\sqrt{T})$ and $\hat{\sigma}^2$ is a consistent estimator of $\frac{1}{(NT)^{-1}}\sum_{i=1}^{N}\sum_{t=1}^{T}E(e_{it}^2)$.

Estimating the number of primitive shocks.

Bai and Ng, besides developing a test for determining the number of static factors, also developed a model for determining the number of primitive shocks in a dynamic factor model (Bai and Ng, 2007). Primitive shocks, also known as the 'stochastic dimension of the economy' are the fundamental shocks that drive all other economic fluctuations.

Assume we have a VAR(p) process, such that $A(L)F_t = u_t$. This VAR process is driven by a q number of primitive shocks, if there exists a $r \times q$ matrix R, such that $u_t = Re_t$, where the matrix R is q-ranked and the e_t is a $q-$dimensional vector of uncorrelated shocks, with $q < r$. Those q fundamental shocks, through the dynamics of the VAR (that represent factors in our case), expand into r dynamic processes. We denote the variance-covariance matrix of u_t as Σ_u, Σ_u is then equal to $\Sigma_u = R\Sigma_e R'$, where $\Sigma_e = E(e_t e_t{}')$. Since we can see that Σ_u is of rank q, a sensible is to apply spectral decomposition to the sample equivalent of the Σ_u matrix, $\hat{\Sigma}_u$, in order to determine q. The problem is thought that neither F_t, nor r are observable. For this reason Bai and Ng suggest applying Principal Components on the dataset and obtain the r static factors and then computing the eigenvalues of the sample

covariance matrix. Bai and Ng go on to propose tests that check whether those eigenvalues ''satisfy an asymptotically shrinking bound that reflects sampling error''. We will not present their tests in detail though.

Forecasting with Factor models

Factor models have been widely used for forecasting. One of the most important contributions in the forecasting with Factor models literature was made by Stock and Watson (2002). Stock and Watson considered the following model:

$$X_t = \Lambda F_t + e_t \tag{2.2.3}$$

Where X_t is the vector of all time series that might contain useful information for forecasting Y_{t+h} (which is the goal). We set $\chi_\tau = \Lambda F_t$. The variation in those N time series is driven by the r common factors F_t

$$Y_{t+h} = \beta'_w w_t + \beta'_f F_t + \varepsilon_{t+h} \tag{2.2.4}$$

Y_t is a scalar time series which we want to forecast, W_t is an m-dimensional vector that contains observed variables (lags of Y_t etc) that are useful for forecasting Y_t (like F_t), while ε_{t+h} is the forecasting error. The variance-covariance matrix of the idiosyncratic components is written as Ω, (that is $VAR(e_t) = \Omega$).

Stock and Watson allow the idiosyncratic disturbances to be serially-correlated, as well as (weakly) cross-sectionally correlated. If the idiosyncratic components are serially-correlated, then we get a dynamic factor model where the lags of the factors enter our two equations. Stock and Watson were able to prove that the principal components are consistent estimates of the latent variables and that the feasible forecast $\hat{y}_{t+h}$ converges to the optimal infeasible optimal forecast under three assumptions:

a. The errors are stationary.

b. The factors have non-trivial loadings.

c. The idiosyncratic components are at most weakly correlated (both serially and cross-sectionally).This means that Ω is allowed to have non-zero elements off the diagonal. As $N \to \infty$ those elements should tend to zero and the eigenvalues of Ω should be bounded.

In order to take a closer look at the assumptions and reveal the underlying intuition. We have that $\Sigma_X = \Lambda\Lambda' + \Omega$, where $VAR(X_t) = \Sigma_x$.

The eigenvalues of $\Lambda\Lambda'$ should be $O(N)$. Since the eigenvalues of Ω are bounded, the eigenvalues of Σ_X are also bounded. As $N \to \infty$, Principal Components are hence able to distinguish between the idiosyncratic variation and the common variation.

We need both $N \to \infty$ and $T \to \infty$ to obtain consistency. Bain and Ng (2003) showed that the unobservability of F_t is insignificant as $\sqrt{T}/N \to 0$.

The estimated factors are given by

$$\hat{F} = X'\hat{\Lambda}'\frac{1}{N}$$

Where $\hat{\Lambda}$ is equal to the eigenvectors of $X'X$ that correspond to the r largest eigenvalues.

Time-Varying loadings

Stock and Watson consider the case where the factor loadings are not constant over time, which is usually when dealing with datasets that span several decades, especially in Macroeconomics. Temporal instability can be captured in several ways. Here Stock and Watson assume that the Factor loadings follow an AR(1) process.

$$\lambda_{it} = \lambda_{it-1} + g_{iT}\zeta_{it}$$

Where g_{iT} is a scalar random variable, that is assumed to be small, $g_{iT} \sim O_p(T^{-1})$. ζ_{it} is an r −dimensional vector of random variables. In order to rule out extensive shifts in a large number of predictors, Stock and Watson assume that ζ_{it} has weak cross-sectional dependence. Stock and Watson prove that under these assumptions, the Principal Components estimator of the factors is still consistent.

2.3 Weaknesses of Principal Component regression

The relationship between data and estimates on factor models.

Jean Boivin and Serena Ng (2005) investigated the relationship between the available data at hand and the corresponding factor estimates. In particular, they wanted to determine whether more data always lead to improvements on the factor estimates and the subsequent forecasting. They discovered that more data not only do not necessarily lead to better forecasting estimates, but they might as well lead to worse ones. Stock and Watson (2002) proved that Principal Component Analysis provides asymptotically consistent factor estimates, even when allowing small cross-correlation in the errors. Boivin and Ng argue that cross-correlation can lead to such problems, as well as when the factor more useful for forecasting, is not among the main drivers of the variation in the dataset.

The paper considers a model similar to that of Stock and Watson (2002)

$$\hat{y}_{t+1|y_1,y_2..y_t,F_t^0} = \hat{\beta}_0 + \hat{\beta}'_1 F_t^0 + \sum_{j=1}^{p} \hat{\gamma}_j \, y_{t-j+1} \qquad (2.3.1)$$

Both lags of y_t and factors are utilized for forecasting y_{t+1}. The factors are not observed, and thus need to be estimated. The N observed series admit a factor representation

$$X_{it} = \lambda_i^{0'} F_t^0 + e_{it}, \text{i} = 1,2 \dots \dots \text{N}, \text{t} = 1,2,3, \dots \text{T} \quad (2.3.2)$$

Boivin and Ng set $x_{it} = \lambda_i^{0'} F_t^0$. After estimating the factors, the forecast is

$$\hat{y}_{t+1|y_1,y_2..y_t,\hat{F}_t^0} = \hat{\beta}_0 + \hat{\beta}'_1 \hat{F}_t^0 \sum_{j=1}^{p} \hat{\gamma}_j \, y_{t-j+1} \quad (2.3.3)$$

It is clear that the forecast is influenced by the estimated factors. Assume that $VAR(X_t) = \Sigma_X$, $VAR(\chi_t) = \Sigma_X$. We have that

$$VAR(X_t) = VAR(\chi_t + e_t) =) \, \Sigma_X = \Sigma_\chi + \Omega \quad (2.3.4)$$

Since it is assumed that all factors are commons to all variables, Σ_X has r non-zero eigenvalues. These eigenvalues represent the amount of variation that each factor is responsible for in X_t. These eigenvalues increase with N. The intuition behind this is that as we add more time series in the dataset, and since all factors are common to all series, the variation that the factors are responsible in X_t for increases. The factors are estimated by obtaining the eigenvalues of the sample covariance matrix $\hat{\Sigma}_X$ and the corresponding eigenvectors. If we denote the eigenvectors corresponding to the jth largest eigenvalue as $v_j = (v_{1j}, v_{2j} \dots \dots v_{Nj})$, then the factor estimates and loading estimates are

$$\hat{F}_{t,N}^j = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} X_{it} \, v_{ij} \quad (2.3.5)$$

In order to illustrate how more data can lead to poorer estimates, when there is cross-correlation in the idiosyncratic components, Boivin and Ng consider the very simple case, where there is only one factor $r = 1$, identical loadings ($\lambda^i = \lambda, i = 1, \dots . N.$) and Σ_X is known. The estimated factor would be $\widehat{F}_t = F_t + \frac{1}{N}\sum_{i=1}^{N} e_{it}$ and (if the idiosyncratic components are iid) $VAR(\widehat{F}_t) = \frac{\sigma^2}{N}$. We can easily see that the variance of the estimation falls as N increases. If though the researcher accidentally adds a second identical set of N_1 series. Now we have $N = 2N_1$, the variance is $VAR(\widehat{F}_t) = \frac{\sigma^2}{N_1}$. There is no gain from the second set of series, since no new information has been added. The same intuition applies when newly added series that actually have a detrimental effect on the estimates.

Boivin and Ng continue with a Monte Carlo simulation in order to prove their findings. The data generation process behind the series to be forecasted is

$$y_{t+1} = \sum_{j+1}^{r} \beta_j \, F_{jt}^0 + \varepsilon_{t+1} \text{ with } \varepsilon_t \sim N(0, \sigma_\varepsilon^2) \quad (2.3.6)$$

The X_{it} are linearly dependent on the factors, with the loadings following a $N(1,1)$ distribution.

$$X_{it} = \sum_{m=1}^{r} \lambda_{im} \, F_{mt} + e_{it} \quad (2.3.7)$$

At first, Boivin and Ng consider a model with three set of series. The sample size for each set of series is N_1, N_2 and N_3. The total sample size is $N = N_1 + N_2 + N_3$. The three set of series display the following characteristics regarding their variance

$$N_1: e_{it} = \sigma_1 u_{it}$$
$$N_2: e_{it} = \sigma_2 u_{it}$$
$$N_3: e_{it} = \sigma_3 (u_{it} + \sum_{j=1}^{C} \rho_{ij} u_{jt})$$

u_{it} follows a normal distribution with a mean of zero and a variance of 1.

The first N_1 set of series and the next N_2 set have mutually uncorrelated series, but with the N_2 set of series having higher variance ($\sigma_2^2 > \sigma_1^2$). Each series belonging to the N_3 is correlated to some C series belonging to the N_1 set with ρ_{ij} correlation coefficients. The assumption of $\sigma_1 = \sigma_3 < \sigma_2$ is also made in order to make the distinction between the cases of cross-correlation and large variance identifiable. All the above assumptions produce an Ω equal to

$$\Omega = \begin{bmatrix} \sigma_1^2 I_{N1} & O_{N1\times N_2} & \Omega_{1\times 3} \\ O_{N_2\times N_1} & \sigma_2^2 I_{N1} & O_{N_2\times N_3} \\ \Omega'_{1\times 3} & O_{N3\times N_2} & \Omega_{3\times 3} \end{bmatrix}$$

Boivin and Ng conclude that 'clean' series like the series that belong to the N_1 set of series are beneficial to the researcher. On the other hand, increasing data that belong to the N_2 or N_3 set of series is actually detrimental to the efficiency of the factor estimates and the forecasting. This is because those sets display cross-correlation and vastly unequal variances.

In their second Monte Carlo simulation, Boivin and Ng focus on oversampling. The data is assumed to be driven by two factors

$$X_{it} = \lambda_{i1} F_{1t} + \lambda_{i2} F_{2t} + e_{it}$$

The two factors follow an $AR(1)$ process.

$$F_{mt} = 0.5 F_{mt\_1} + u_{mt}, u_{mt} \sim N(0,1)$$

The series to be forecasted are

$$y_{t+1}^A = \beta^A F_{1t} + \varepsilon_{t+1}^A$$
$$y_{t+1}^B = \beta^B F_{2t} + \varepsilon_{\tau+1}^B$$

There are 5 set of series in this Monte Carlo exercise

$$N_1: X_{it} = 0.8 F_{1t} + e_{it},\ e_{it} \sim N(0, 1 - 0.8^2)$$
$$N_2: X_{it} = 0.6 F_{2t} + e_{it},\ e_{it} \sim N(0, 1 - 0.6^2)$$
$$N_3: X_{it} = 0.4 F_{1t} + 0.1 F_{2t} + e_{it}, e_{it} \sim N(0, 1 - 0.4^2 - 0.1^2)$$
$$N_4: X_{it} = 0.1 F_{1t} + 0.4 F_{2t} + e_{it}, e_{it} \sim N(0, 1 - 0.1^2 - 0.4^2)$$
$$N_5: X_{it} = e_{it},\ e_{it} \sim N(0,1)$$

We can see that some series are influenced by the first factor, some by the second factor and some by both, albeit with different loadings. It is clear that the estimated factor space will depend on which set of series the researcher will use. If, for example, the researcher use information from the N_1 set, then the principal component estimates will estimate the space spanned by the first factor. This will cause no problem

if the goal is to forecast y_{t+1}^{A}. If, on the other hand, the goal is to forecast y_{t+1}^{B}, then using data where the variation is driven by the first factor would be a disaster. The same applies to the case where the dataset is driven by both factors, but one dominates the other. For example, if the researcher uses the N_3 dataset and estimates only one factor, then forecasting y_{t+1}^{B} would prove to be problematic. In order to obtain an accurate forecast for y_{t+1}^{B} , we would need to estimate two factors. Boivin and Ng point out that when the data is more informative about some factors than other factors, and the factors in the forecasting equation are not dominant in their datasets, then underestimating the number of factors in the forecasting equation can be disastrous. Hence, in cases where the factors needed for forecasting, are dominated in their respective datasets , then the inclusion of a greater number of factors (than the true number in the forecasting equation) is needed to obtain accurate forecasts. This, though comes at the cost of variability, since more factors also induce more sampling variability. One can see that, a smaller dataset can actually be preferable to a larger one. For example, forecasting y^{B} using the N_2 dataset produces a smaller forecasting error than using the $N_2 + N_3$ dataset.

Data generation procees of the target variable

There are more instances where Principal Component Regression can fail. As Groen and Kapetanios (2015) explain, most econometric work, such as Stock and Watson (2002) and Kelly and Pruitt (2015), assumes that the target variable is only influenced by the factors. In other words, the function of the target variables involves the factors, rather than the individual predictors. Groen and Kapetanios suggest that, if the function of the target variable involves the individual predictors, then Principal Component Regression will not be able to asymptotically achieve the best infeasible forecast.

Assume that

$$y_t = a' x_t + \varepsilon_t, \quad t = 1 \dots T \tag{2.3.8}$$

Where y_t is the target variable of dimensions $T \times 1$, x_t is the $N \times 1$ vector of the individual predictors (or time series) and a is the predictive coefficient on x_t. x_t admits a factor representation. The number of factors is r.

$$x_t = \Lambda' F_t + e_t \tag{2.3.9}$$

If we combine (2.3.8) and (2.3.9), we get $y_t = a'(\Lambda' F_t + e_t) + \varepsilon_t =) \; y_t = a'\Lambda' F_t + a' e_t + \varepsilon_t$. Set $\zeta' = a'\Lambda'$. We end up with

$$y_t = \zeta' F_t + (a' e_t + \varepsilon_t) \tag{2.3.10}$$

Kapetanios and Groen consider the case where $y_t = x_{it}$, for some i and denote as x_{-it} the set of all the predictors except x_{it}. Since Ω is not necessarily diagonal, x_{-it} contains information useful in forecasting x_{it}, that is not contained in F_t. Thus, including x_{-it} in the predictive equation for y_t can improve forecasting performance.

If we apply Principal Component on the data, we get

$$y_t = \hat{\zeta}' \widehat{F_t} + \left(\left(\zeta' F_t - \hat{\zeta}' \widehat{F_t} \right) + a' e_t + \varepsilon_t \right) \tag{2.3.11}$$

$\widehat{F_t}$ is the PC estimate for the factors and $\hat{\zeta}'$ is the OLS slope coefficient of $\widehat{F_t}$ on y_t. Kapetanios and Groen explain that the asymptotic properties of PCR is based on the asymptotic behavior of a, since $\hat{\zeta}' \widehat{F_t}$ converges to $\zeta' F_t$ for $N \to \infty$. If $\|a' e_t\|$is $O_p(1)$ then PCR does not achieve the infeasible best forecast. Kapetanios and Marcelinio (2010) assumed that the factor loadings are a function of N.

$$\Lambda_N = N^{-p}\bar{\Lambda} \tag{2.3.12}$$

$\bar{\Lambda}$ is a matrix of fixed loadings and dimension of $r \times N$. Kapetanios and Marcelinio (2010) prove that in this case, the eigenvalues of ${\Lambda_N}'\Lambda_N$ converge at a rate of N^{1-p} for $p \in [0,1)$. In that case, the eigenvalues of $VAR(x_t)$ in (2.3.9) are bounded for $p \in [0.5\ , 1)$. Hence, as $N \to \infty$, Principal Component regression is not able to distinguish between common and idiosyncratic variation.

In conclusion, there are many instances where PC regression fails to be the most useful method for macroeconomic forecasting.

2.4 Partial Least Squares

Partial least squares is a method that bears a lot of resemblance to Principal Component Analysis. It is widely used in the field of Chemometrics, but not so much in Econometrics, especially when compared to Principal Components. The main difference between the two methods is that, while PCA applies spectral decomposition to the variance-covariance matrix of the predictors X, PLS does this on the variance-covariance matrix of X with the target variable y. In mathematical terms, PLS decomposes the $COV(X, y) = X'yy'X$ matrix, instead of the $VAR(X) = X'X$ matrix. The factors constructed from PLS are those linear combinations of x_t, that give the maximum covariance between y and the themselves, while having zero covariance with each other.

One algorithm, among many, to construct k PLS factors is the following:

1. Set $j = 1$. Demean y_t and normalize x_t, so that each series has zero mean and unit time series variance.

2. Compute individual covariances $r_{ij} = COV(x_{it}, y_t)$ for $i = 1,2, \dots N$. This step gives us the Covariance of the target variable with each individual predictor. We set $r_j = (r_{1j}, \ r_{2j} \ \dots, \ r_{Nj})'$ and construct the j_{th} factor by taking ${r'}_j x_t$. That is $f_{j,t} = r_{1j}x_{1t} + r_{2j}x_{2t} + \cdots r_{Nj}x_{Nt}$

3. Regress $x_{it}, i = 1,2 \dots N$ and y_t on $f_{j,t}$. We denote the residuals of those two regressions v_t and u_t respectively.

4. If $j = k$, stop, else, we set $j = j + 1$ and repeat the process from step 2 with the residuals v_t and u_t instead of x_{it} and y_t. That is we compute r_j by taking $r_{ij} = COV(v_{it}, u_t)$.

This algorithm is similar to the automatic proxy selection algorithm of the 3PRF method, which will be presented later.

Once the k factors have been constructed by the algorithm, we regress y_t on the factors $f_{j,t}$. The closed form formula for y will be

$$\hat{y} = XX'y(y'XX'XX'y)^{-1}y'XX'y$$

2.5 Three Pass regression filter

Kelly and Pruitt (2015) introduced a new estimator called the Three Pass Regression Filter (3PRF). The idea behind the 3PRF is that while PCA is able to identify the factors that drive the variation in the dataset X, those factors might be uncorrelated with the series to be forecasted (or forecast target) y. In other words , those extracted factors are

irrelevant for y and hence unable to be used for forecasting y efficiently. The goal is therefore, to be able to identify the factors that are relevant for the forecast target y, while in the same time get rid of the factors that are irrelevant for y, even if they are dominant in the dataset X. The Three Pass Regression Filter is able to do all of the above.

The 3PRF relies on the use of proxies in order to extract and utilize the factors that are relevant to the forecast target y. Proxies are variables that are driven exclusively by the target relevant factors and have zero loadings on the irrelevant factors. In the most simple form of the 3PRF, proxies are automatically created by an algorithm (called the automatic proxy selection algorithm) which uses the available data from the predictor matrix X and target y. Alternatively, the researcher can provide the method with proxies based on economic theory.

In order to present all the different formulations that the 3PRF can take, we need to mathematically define the framework wherein the 3PRF is going to be used. The dataset, or matrix of predictors is denoted by X. The time series in X are standardized so that they have unit time series variance. The dimensions of X are $T \times N$, where N is the number of variables and T is the number of time observations. X need not be a balanced panel, since 3PRF can handle unbalanced panels, as well as missing data. Kelly and Pruitt use x_t to indicate an N-dimensional cross-section of predictors, observed at time t, and x_i to indicate a T-dimensional time series of the ith variable (or predictor).

An intuitive illustration of the denotations x_t and x_i can be seen in the table below.

$$\text{Time periods} \quad \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_t \\ \vdots \\ x_T \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \dots & x_{2N} \\ \vdots & \vdots & \dots & \vdots \\ x_{t1} & x_{t2} & \dots & x_{tN} \\ \vdots & \vdots & \dots & \dots \\ x_{T1} & x_{T2} & \cdots & x_{TN} \end{bmatrix}$$

We see that $x_t = [x_{t1} \quad x_{t2} \quad \dots \quad x_{tN}]'$ is the N-dimensional cross-section of predictors, observed at time t.

$$\text{Variables}$$
$$[x_1 \quad x_2 \quad \dots \quad x_i \quad \dots \quad x_N]$$
$$||$$
$$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1i} & \dots & x_{1N} \\ x_{21} & x_{22} & \dots & x_{2i} & \dots & x_{2N} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{T1} & x_{T2} & \dots & x_{Ti} & \dots & x_{TN} \end{bmatrix}$$

Just like x_t is the vector of observations for the N variables at time t, x_i is the vector of T observations for the ith predictor (or just the whole time series of the ith predictor).

$$x_i = [x_{1i} \quad x_{2i} \quad \dots \quad x_{Ti}]'$$

The forecast target y is a vector of $T \times 1$ dimensions. The proxies Z are of dimensions $\times L$, where $L \ll \min(N, T)$.

The underlying model is assumed to be this

$$x_t = \Phi_0 + \Phi F_t + \varepsilon_t$$
$$y_{t+1} = \beta_0 + \beta' F_t + \eta_{t+1}$$
$$z_t = \lambda_0 + \Lambda F_t + \omega_t$$

The 3PRF can be expressed in close form. In order though to provide an intuitive explanation of the 3PRF, Kelly and Pruitt first describe how the 3PRF is equivalent to a sequence of 3 OLS regressions, which gives the estimator its name. Those three regressions are named 'passes' by the authors.

First Pass

The first pass consists of N time series regressions, one for each predictor. More specifically, each predictor is regressed against the proxies

$$x_i = \varphi_{0,i} + z'\varphi_i + \varepsilon_{it} \quad for\ i = 1, \dots N \tag{2.5.1}$$

We retain the slope coefficient of each regression. In other words, each one of the N time series corresponds to an estimated slope coefficient $\widehat{\varphi_i}$.

$$[x_1 \quad x_2 \quad \dots \quad x_i \quad \dots \quad x_N]$$
$$\downarrow$$
$$\hat{\varphi} = [\widehat{\varphi_1} \quad \widehat{\varphi_2} \quad \cdots \quad \hat{\varphi}_i \quad \dots \quad \widehat{\varphi_N}]$$

Second Pass

We run T cross-section regressions, where the regressors are the estimated slope coefficients retained from the first step and the regressed variable are the x_t.

$$x_t = \varphi_{0,t} + \hat{\varphi}' F_t + e_{it} \tag{2.5.2}$$

The second pass produces the estimated factors at each point in time. For each x_t we obtain the value of the estimated factors.

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_t \\ \vdots \\ x_T \end{bmatrix} \text{Regress each } x_t \text{ on } \hat{\varphi}' \rightarrow \begin{bmatrix} \widehat{F_1} \\ \widehat{F_2} \\ \vdots \\ \widehat{F_t} \\ \vdots \\ \widehat{F_T} \end{bmatrix}$$

Third Pass

The third pass is a simple time series regressions. The regressors are the factors that were estimated from the second step and the regressed variable is our forecast target.

$$y_{t+1} = \beta_0 + \widehat{F_\tau}' \beta + \eta_{t+1} \tag{2.5.3}$$

The fitted value $\widehat{\beta_0} + \widehat{F_\tau}' \hat{\beta}$ is the forecast.

The 3PRF forecast can be expressed in a close form. If we set $J_T = I_T - \frac{1}{T} i_T i_T'$, where is the $T-$dimensional identity matrix, and $i_T = [1,1,1 \dots \quad 1]'$, which is of length T. J_N is the equivalent for N and $\bar{y} = i_T' y / T$. The J matrices enter because each regression is run with a constant. The close form expression for the forecast is

$$\hat{y} = i_T\bar{y} + J_T X J_N X' J_T Z\big((J_N X' J_T Z)' X' J_T X (J_N X' J_T Z)\big)^{-1}(J_N X' J_T Z)' X' J_T y \qquad (2.5.4)$$

If we set $Q_{XZ} = J_N X' J_T Z$, $H_{XX} = X' J_T X$ and $H_{Xy} = X' J_T y, H_{zz} = Z' J_T Z$, $H_{XZ} = X' J_T Z$,then the above unreadable expression is condensed into

$$\hat{y} = i_T\bar{y} + J_T X Q_{XZ}(Q'_{XZ} H_{XX} Q_{XZ})^{-1}\, Q_{XZ}' H_{Xy} \qquad (2.5.5)$$

The forecast can also be written as

$$\hat{y} = i_T\bar{y} + \hat{F}\hat{\beta} \qquad (2.5.6)$$

Where

$$\hat{F}' = H_{ZZ}(Q'_{XZ} H_{XZ})^{-1} Q'_{XZ} X' \qquad (2.5.7)$$

$$\hat{\beta} = H_{ZZ} Q_{XZ} H_{XZ}(Q'_{XZ} H_{XX} Q_{XZ})^{-1} Q'_{XZ} H_{XY} \qquad (2.5.8)$$

$\widehat{\beta}$ here is the predictive coefficient on the factors. 3PRF is also able to produce the predictive coefficients on the individual variables x_i.

$$\hat{y} = i_T\bar{y} + J_T X \hat{a} \qquad 2.5.9)$$

Where

$$\hat{a} = Q_{XZ}(Q'_{XZ} H_{XX} Q'_{XZ})^{-1} Q_{XZ} H_{XY} \qquad (2.5.10)$$

Selecting Proxies

As Kelly and Pruitt explain, proxies are the main tool the 3PRF uses to distinguish between irrelevant and relevant factors, even if the irrelevant factors dominate the variation in X. Proxies can be provided by the researcher based on economic theory. If though such theory-based proxies are not available, the 3PRF can automatically construct proxies, using the automatic proxy selection algorithm. If the relevant factor is just one, then the forecast target itself can be used as a proxy (this is the target-proxy 3PRF) . The forecast target y can be used as a proxy since it is by definition correlated with the relevant factors and uncorrelated with the irrelevant factors. The Target-proxy 3PRF functions without any problems. If they are more than one relevant factors, then the Target-proxy 3PRF is not able to asymptotically converge to the optimal infeasible forecast. This is due to the fact that the 3PRF will be consistent and asymptotically normal only when the number of proxies is equal to the number of relevant factors in the dataset. This is where the L-automatic-proxy 3PRF enters the fray. The first proxy that the algorithm uses is the forecast target . The second proxy is the residuals from the target-proxy-3PRF forecast. The residuals form the target-proxy 3PRF forecast have zero loadings on the irrelevant factors and non-zero loadings on the relevant factors (due to omitted variable bias). Hence, the selection algorithm uses the residuals from the target-proxy 3PRF as proxies and runs the sequence of regressions, then use the residuals from the two-proxies 3PRF and so on until the number of proxies is equal to the number of relevant factors.

Relationship between the 3PRF and Partial Least Squares (PLS)

The 3PRF is identical to the PLS regression, when i) the predictors x_i are normalized (demeaned and variance-standardized), ii) the first two passes do not include the

constant, iii) The proxies are provided by the automatic proxy selection algorithm. Since the first two passes are ran without a constant, the *J* matrices are removed from the forecast expression

$$\hat{y} = XX'y(y'XX'XX'y)^{-1}y'XX'y$$

2.6 Markov-Switching Three Point Regression Filter

As previously mentioned, when dealing with macroeconomic data, it is plausible to assume that the relationship between the variables does not remain constant over time. The Markov-Switching Three Point Regression Filter (MS-3PRF) was introduced into the literature by Guerin, Leiva-Leon and Marcellino (2018) and differs from the original 3PRF in that the parameters of the three OLS regressions are now time-varying and governed by Markov chains. The data generation process is assumed to be the following

$$y_t = \beta_0(S_{yt}) + \beta(S_{yt})f_{t-1} + \eta_t, \qquad t = 1 \dots T \tag{2.6.1}$$

$$z_{j,t} = \lambda_{0,j}\left(S_{z_j.t}\right) + \lambda_{j.t}\left(S_{z_j.t}\right)f_t + \omega_{jt}, \qquad j = 1 \dots k_f \tag{2.6.2}$$

$$x_{it} = \Phi_{0,i}(S_{x_i,t}) + \Phi_{f,i}(S_{x_i,t})f_t + \Phi_{g,i}(S_{x_i,t})g_t + \varepsilon_{it}, \quad i = 1, \dots . N \tag{2.6.3}$$

The variables are the same as in Kelly and Pruitt (2015), although the vector of non-relevant factors g_t is part of the f_t in Kelly and Pruitt. The difference in the Data generation process here with the one assumed by Kelly and Pruitt, is the existence of the M-state Markov chains: $S_{yt}, S_{z_j.t}, S_{x_i,t}$. The Markovian chains cause time-variation in the parameters. Each of the Markov chains has *M* states and therefore an $M \times M$ transition matrix.

The factors are assumed to be orthogonal to each other and with an identity variance-covariance matrix.

The algorithm

While the algorithm still consists of the three filters, there are significant changes due to the introduction of the Markovian variables. Specifically, the first and third filter are no longer simple OLS regressions but Markov-Switching regressions. The procedure behind the estimation of a Markov-Switching regressions has been explained at chapter **1.2**.

- First step

We regress the x_{it} against the proxies.

$$x_{it} = \Phi_{0,i}(S_{x_i,t}) + \Phi_i(S_{x_i,t})z_t + e_{it} \tag{2.6.4}$$

Since Markov-Switching regressions are estimated via QMLE, we need the normality assumption for the error e_{it}. We assume $e_{it} \sim N(0, \sigma^2_{x_i}(S_{x_i,t}))$. The error variance is also driven by the same Markov chain. For each predictor *i* we obtain an estimated $1 \times k_f$ vector of coefficients, that is *N* vectors, just like in the classic 3PRF. The difference is that the vector of coefficients is also influenced by which Markov state applies at any particular period of time. For each $\widehat{\Phi_{it}}$, there are *M* different possible values based

on the Markov state that prevails at time t. For each Markov state we calculate the smoothing probability $P(S_{x_i,t} = j|\Omega_T)$. Ω_T is the full sample information set. Then, at each period of time we get the weighted average of each state multiplied by the corresponding smoothing probability.

$$\widehat{\Phi}_{A,it} = \sum_{j=1}^{M} \widehat{\varphi}_{\iota} P(S_{x_i,t} = j|\Omega_T). \tag{2.6.5}$$

An alternative way to 'collapse' the values produced by the M regimes into one is to just select the value given by the regime most likely to apply at each point in time.

$$\widehat{\Phi}_{B,it} = \sum_{j=1}^{M} \widehat{\varphi}_{\iota} I(P(S_{x_i,t} = j|\Omega_T)) \tag{2.6.6}$$

$I(.)$ is a function that chooses the regime with the highest smoothing probability. As a result we only choose one state, rather than summarizing the information contained in all states as in (2.6.5). We can illustrate the first step as we did in Kelly and Pruitt

$$[x_1 \quad x_2 \quad \dots \quad x_i \quad \dots \quad x_N]$$
$$\downarrow$$

$$\widehat{\varphi_q} = \begin{bmatrix} \widehat{\varphi_{11}} & \widehat{\varphi_{21}} & \cdots & \widehat{\varphi_{\iota 1}} & \cdots & \widehat{\varphi_{N1}} \\ \widehat{\varphi_{12}} & \widehat{\varphi_{22}} & \cdots & \widehat{\varphi_{\iota 2}} & \cdots & \widehat{\varphi_{N1}} \\ \vdots & \vdots & \ddots & \vdots & \cdots & \vdots \\ \widehat{\varphi_{1t}} & \widehat{\varphi_{2t}} & \cdots & \widehat{\varphi_{\iota t}} & \cdots & \widehat{\varphi_{Nt}} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \widehat{\varphi_{1T}} & \widehat{\varphi_{2T}} & \cdots & \widehat{\varphi_{\iota T}} & \cdots & \widehat{\varphi_{NT}} \end{bmatrix}$$

Where $q = A\ or\ B$. We see that in contrast with Kelly and Pruitt, the estimated coefficients have a time dimension, instead of just a cross-sectional dimension.

- Second step

The second step is almost similar to that of Kelly and Pruitt. There are T cross-sectional OLS regressions.

$$x_{it} = a_{0,t} + \hat{\varphi}_{q,it} f_t + u_{it}, \quad t = 1 \dots T \tag{2.6.7}$$

Where $u_{it} \sim N(0, \sigma_{ui}^2)$, q is either A or B. While in the classic 3PRF the regressors where the same for each time period, and only the regressed variable were different across the time dimension, in the MS-3PRF both the regressed variable and the regressors change for each t. We keep the OLS estimates $\hat{f}_t$.

- Third step

We run one time-series Markov-Switching regression of the forecast target y_t on the extracted factors from the second step.

$$y_t = \beta_0(S_{yt}) + \beta(S_{yt})\hat{f}_{t-1} + \eta_t, \quad t = 1 \dots . T \tag{2.6.8}$$

Where $\eta_t \sim N(0, \sigma_\eta^2(S_{yt}))$. We get the QMLE estimates for β_0 and β. Now we can proceed to forecast the target one period ahead.

$$y_{T+1|T} = \sum_{j=1}^{M} [P(S_{yT+1} = j|\Omega_T)\widehat{\beta_0}(S_{yT+1} = j) + P(S_{yT+1} = j|\Omega_T)\widehat{\beta_1}(S_{yT+1} = j)\widehat{f_T}]$$

The intercept and slope coefficient is weighted by the prediction probability of each regime at time T. This is the same technique used in the first step to obtain $\widehat{\Phi}_{A,it}$. The Markov switching allows us to take into account any time variation in the intercept, as well as time variation in the predictive power of the estimated factors for the target variable. However, it is also possible to run the third step as a regular OLS regression if we believe that there are no regime switches between the predicted variable and the predictors and if we also seek to ease the computational burden.

The number of states can be determined using quasi-likelihood tests (Cho and White (2007)) or goodness-of-fit measures (Smith et al. (2006)). To avoid computational complexity a choice of two or three states is usually enough.

The Monte Carlo simulations showed that the MS-3PRF dominated all other considered techniques (PCA .PC-LARS, linear 3PRF) when there was an underlying instability in the DGP.

Guevin, Leiva-Leon and Marcellino also applied the MS-3PRF to the McCracken dataset. They discovered that the MS-3PRF is able to identify Business cycles (which are associated with different regimes). The smoothed probability of being in the first regime (which represents recessions) shows significant time variation. This corroborates the hypothesis of the existence of regimes. The MS-3PRF is also shown to produce superior forecasts in the majority of cases considered.

Just like the regular 3PRF, MS-3PRF can automatically construct proxies or use the target itself as a proxy if the number of factors is one.

Chapter 3

PENALIZED REGRESSION

Penalized regressions concern all models that include a penalty for overfitting a dataset. Those penalties are usually associated with a particular tuning parameter that constrains the parameters and shrinks them towards zero. If the penalty is too large then some of those penalized regression can function as variable selectors.

Those regressions can also be interpreted in the Bayesian tradition. Bayesian shrinkage regressions incorporate a-priori information into the model by the form of informative priors. Thus, the relative importance of the dataset as the only source of information supplied in the model is reduced and overfitting is therefore avoided.

3.1 Bayesian Shrinkage

Bayesian methods are particularly useful when it comes to large panels of data. This is because the priors can function as shrinkage operators and overcome the curse of dimensionality. The choice of the prior distribution hands us a great deal of flexibility. De Mol, Giannone and Reichlin (2006) compared the forecasting performance of Bayesian regression with Principal Components regression in a large dataset. The authors consider two priors distributions, a Gaussian prior and a double exponential prior. The first leads to the ridge regression estimator, while the later to the LASSO operator.

Instead of focusing on the factor representation of the dataset X_t and the relationship of the target y_t with the factors, we directly examine the relationship between y_t and X_t. We introduce shrinkage to overcome the curse of dimensionality. The errors are assumed normally distributed, $u_t \sim N(0, \sigma_u^2)$.

Gaussian Priors

We assume a Gaussian prior for the $m-$dimensional vector of parameters , $\beta \sim N(\beta_0, \Phi_0)$. The mode of the posterior distribution of β is

$$\hat{\beta}_{bayes} = \left(X'X + \frac{\sigma_u^2}{\Phi_0} \right)^{-1} X'y$$

If the parameters are identically and independently distributed, then $\Phi_0 = \sigma_\beta^2 I$. Then we get:

$$\hat{\beta}_{bayes} = \left(X'X + \frac{\sigma_u^2}{\sigma_\beta^2} I \right)^{-1} X'y$$

This estimator is equivalent to the one produced by the following penalized regression, which is called ridge regression, when $\lambda = \frac{\sigma_u^2}{\sigma_\beta^2}$.

$$\min_\beta \sum_{i=1}^{n} (y_i - x_i'\beta)^2 + \lambda \sum_{j=1}^{m} \beta_j^2$$

The resulting equivalent estimator is $\hat{\beta}_{ridge} = (X'X + \lambda I)^{-1} X'y$. We will briefly abandon the Bayesian world and enter the Frequentist realm once again in order to explore some of the properties of the ridge regressor. The idea behind ridge regression is to introduce artificial bias into the model, in exchange for lower variance. We can calculate the bias of an estimator b as:

$$Bias(b) = E(b) - \beta_{true}$$

Where β_{true} is the true value of the parameter. While it is known that OLS estimator has no bias, ridge estimator does. Specifically.

$$Bias\left(\hat{\beta}_{ridge}\right) = -\lambda (X'X + \lambda I)^{-1} \beta_{true}$$

But at the expense of variance.

$$Var(\hat{\beta}_{ridge}) = \sigma_u^2 (X'X + \lambda I)^{-1} X'X (X'X + \lambda I)^{-1}$$

The ridge estimator (or Bayesian for $\lambda = \frac{\sigma_u^2}{\sigma_\beta^2}$) shrinks all the slope coefficients β_j but does not set any equal to zero. If the dataset includes numerous variables that have zero explanatory power on the target y, the Bayesian estimator does poorly. That is why the authors also explore a second prior distribution.

Double exponential Prior

By assuming that β follows a double exponential prior and that the coefficients are iid (just like for ridge), we obtain the LASSO operator. This is equivalent to the solution of the following problem.

$$\min_{\beta} \sum_{i=1}^{n} (y_i - x_i'\beta)^2 + \lambda \sum_{j=1}^{m} |\beta_j|$$

The double exponential prior is denser around zero and in the tails. The resulting posterior produces estimates (modes of the parameters) that are either equal to zero or large. That is why LASSO functions as a variable selector, besides as just a shrinkage operator.

Comparison

Ridge regression does not discard any variables. This might cause problems if a large number variables have slope coefficients equal to zero (are useless for forecasting y). The ridge estimator will not get rid of those variables, only reduce their slope coefficient. This will burden our model with increased variance. Another instance where the Ridge estimator (as well as PCR) fails is when variables with significant explanatory power on y_t are not dominant in the dataset X_t. The ridge estimator weakens the influence of such variables and inadvertently loses considerable explanatory power. LASSO on the other hand, is able to overcome such a problem, as it will discard such kinds of variables.

As we have mentioned, LASSO is not without its shortcomings. If there is a group of highly correlated variables in the dataset, LASSO tends to aggressively throw away all variables in said group and just keep one. Those discarded variables might had valuable information on y_t which is discarded by LASSO.

The authors discovered, the forecasts produced by the Bayesian methods are highly similar to those produce by PCR, for priors that ensure good forecasting performance of the Bayesian methods. This particularly the case when the dataset displays a strong factor structure.

3.2 Targeted Predictors

Another methodology that builds on top of the PCR tradition of Stock and Watson (2002) and penalized regressions is the targeted predictors methodology developed by Jushan Bai, Serena Ng (2008). The researchers build a model that takes into account which components are relevant to the forecast target, so that the problem described by Boivin and Ng (2005) of the excessive use of noisy and uninformative series can be overcome. This is done through thresholding rules that rely on the LARS algorithm. In addition to that, Jushan Bai and Serena Ng also allow the factors to have a non-linear relationship to the forecast target. The model is more complex than classical Principal

Component Regression of Stock and Watson (2002) but is also shown to produce superior empirical results.

As usually, we have a forecast target y_t and a dataset X_t which admits a factor representation.

$$y_{t+h} = a'W_t + \Gamma' X_t + \varepsilon_t \tag{3.2.1}$$

$$X_t = \Lambda F_t + e_t \tag{3.2.2}$$

W_t contains lags of y_t or a constant. Since the factors are linear combinations of the variables in X_t , we can replace the factors with the term $\Gamma' X_t$. Stock and Watson (2002) construct factors that contain information from all N predictors, that means that Γ has no elements that equal zero . Bain and Ng on the other hand introduce a methodology that selects the predictors that have relevant information for y_t and discards the others.

$$y_{t+h} = a'W_t + \bar{\Gamma}' X_t + \varepsilon_t \tag{3.2.3}$$

$\bar{\Gamma}'$ has elements equal to zero where it needs to in order to discard non-relevant predictors. There are various thresholding processes that define $\bar{\Gamma}'$. We start by explaining the LARS algorithm (Efron et al 2004) and is able to select the variables that will be included in the final model. LARS has many advantages, it avoids including strongly correlated variables in the model, it has a low computational cost and it is not as 'impulsive' as some of the other selection algorithms. The LARS variable selection algorithm is an improvement on Forward stagewise selection regression. Forward stagewise selection regression works by iteratively adding those predictors in the model that are the most correlated with the residual vector from the previous iteration. At the first step, the algorithm selects and adds to the model the predictor that displays the maximum covariance with the prediction target y. Then the algorithm runs the regression of y on this predictor and constructs the prediction vector and the residuals vector. The algorithm selects it's next predictor to be the one that has the maximum correlation with the residuals vector. In other words, the predictor that contains the most information not yet included in the model is chosen. This not yet accounted for information is contained in the residuals, and thus the predictor having the greatest correlation with the residuals (sharing the most information with the residuals) is chosen. Hence, the algorithm can proceed with adding new information at each iteration. An algebraic representation of the algorithm is the following

The algorithm starts by setting $\hat{\mu} = 0$.

$$\hat{c} = c(\hat{\mu}) = X'(y - \hat{\mu})$$

Where $\hat{\mu}$ is the prediction vector and equal to $\hat{\mu} = \sum_{j=1}^{m} x_j \hat{\beta}_j$ and $y - \hat{\mu}$ it's the residuals vector. $c(\hat{\mu})$ is the vector of current correlation.

The algorithm chooses j (a predictor) that satisfies: $\hat{j} = argmax|\hat{c_j}|$. Then $\hat{\mu}$ is updated according to $\hat{\mu} \to \hat{\mu} + \varepsilon \cdot sign(\hat{c}_j) x_j$. If $\varepsilon = |\hat{c}_j|$ then the algorithm (called forward selection regression) moves way too aggressively to the direction of the predictor x_j. Variables that are correlated with x_j can be discarded by the algorithm even if they are useful in predicting y. This is because predictors that are strongly correlated

with the variable that just go into the "in" set will have low correlation with the residual vector (they will contain a low amount of new information, since identical information has just been included in the model). While avoiding the inclusion of strongly correlated predictors is a good thing, forward selection regression is way too 'greedy' . The Forward Stagewise regression sets ε equal to a small constant, so that the algorithm is more cautious at each iteration.

LARS differs from the Forward Stagewise regression in that it does not move to the direction of the *j* predictor. Instead, it moves equiangularly in the direction between variables in the "most correlated set". That means that the residual vector, instead of being made to be maximally correlated with the chosen *j* predictor, it will instead be equally correlated with all the predictors in the set. LARS is computationally less demanding than the Stagewise algorithm. The exact algebraic procedure of the LARS algorithm is the following:

At first set $\mu = 0$. Just like before, $\hat{c} = c(\hat{\mu}) = X'(y - \hat{\mu})$.

We define A as the set of set of indices corresponding to the predictors with the greatest absolute current correlations.

$\hat{C} = \max\{|\hat{c}_j|\}, A = \{j: |\hat{c}_j| = \hat{C}\}$

Let $s_j = sign(\hat{c}_j)$ for $j \in A$ .We compute the following matrices:

$X_A = s_j x_j, j \in A$, which is the active set A matrix.

$G_A = X_A{}' X_A$

$Q_A = (1_A{}' G_A{}^{-1} 1_A)^{-\frac{1}{2}}$

The unit equiangular vector with columns of the active set matrix X_A can be defined as

$$u_A = X_A Q_A G_A^{-1} 1_A$$

We also define $q_A = X' u_A$. Each of the algorithms updates $\hat{\mu}$ according to $\hat{\mu} \to \hat{\mu} + \varepsilon \cdot sign(\hat{c}_j) x_j$. Stagewise regression chooses a small value for ε, while forward selection regression chooses $\varepsilon = |\hat{c}_j|$. LARS chooses updates $\hat{\mu}$ according to $\hat{\mu}_{new} = \hat{\mu} + \hat{\gamma} u_A$ where $\hat{\gamma} = \overset{+}{\min}\{\frac{\hat{C} - \hat{c}_j}{Q_A - q_j}, \frac{\hat{C} + \hat{c}_j}{Q_A + q_j}\}$ j ∈ A. The algorithm chooses the minimum over the positive elements of the set. The size of the active set can be determined by an information criterion.

Non-Linearity

In addition to variable selection, the researchers introduce the possibility of non-linearity in the model. One way to include non-linearity is to introduce it in the forecast equation. This method is called squared factors. The forecasting equation is then

$$y_{t+h}^h = a' W_t + \beta_1{}' \hat{F}_t + \beta_2{}' \hat{F}_t^2 + \varepsilon_{t+h} \quad (3.2.4)$$

In this model, the volatility of the factors affects the forecast target. Another way to add non-linearity is in the factor representation. Conceptually, this means that the factors are non-linear equations of the predictors X_{it}.

$$g(X_{it}) = \Phi_i{}' J_t + e_{it} \quad (3.2.5)$$

$g(.)$ is a non-linear function, Φ_i is the factor loadings and J_t are the common factors. If we define $X_t^* = g(X_t)$, then we have $X_t^* = \Phi J_t + e_t$. X_t^* can include terms such as the predictors squared or the cross products, for example $X_t^* = \{X_{it}, X_{it}^2, X_{it}X_{jt}\}, j \neq i$. The dimension of the predictor matrix is adjusted accordingly. Including a lot of terms though can be computationally expensive, as a result the researchers only consider the case of $X_t^* = \{X_{it}, X_{it}^2\}$. The dimension in this case is $N^* = 2N$ and the procedure is called squared principal components. The common factors are then estimated with PCA as usually.

LASSO AND ELASTIC NET

Additional techniques that can be used to perform variable selection are the LASSO and the Elastic Net. We will start with the LASSO. LASSO, which stands for least absolute shrinkage and selection operator, solves the following minimization problem

$$\min_{\beta,\alpha} RSS \;\; subject\ to \;\; \sum_{j=1}^{N} |\beta_j| \leq c$$

The LASSO estimator minimizes the RSS (squared sum of residuals) like the OLS, the difference between the two methods is that while OLS performs uninterrupted optimization, LASSO is restricted by the c parameter. Thus an amount of ''artificial'' shrinkage is introduced in the model. LASSO is able to discard variables that have no effect on the target (that are equal to zero) and hence operates as a variable selector. This is in contrast with some other techniques of the same spirit, such as Ridge regression. LASSO does really well, when many $\beta_j's$ in the model are equal to zero.

Despite its strengths, LASSO is not without its flaws. If there is a set of predictors highly correlated with each other, then LASSO tends to just keep one of those predictors and throw away the rest. In order to overcome this flaw, Zou and Hastie (2015) suggested the use of a new estimator, Elastic Net.

The Elastic Net combines the LASSO penalty with the Ridge penalty. Elastic Net solves the following problem.

$$\min_{\beta} RSS + \lambda_1 \sum_{J=1}^{N} |\beta_j| + \lambda_2 \sum_{j=1}^{N} \beta_j^2$$

Efron et al (2004) proved that LASSO is a special case of LARS.

Chapter 4

NOWCASTING

4.1 Monitoring Macroeconomic data in real time

A very common problem that statistical authorities, central banks, governments and other institutions face, is that the present state of the economy is not entirely known. A lot of variables, such as GDP, are published with a lag and are usually revised subsequently. As a result, it is important for institutions to be able to estimate the present value of important economic variables. For this purpose, various statistical techniques have been developed that utilize factor models in order to monitor the economy in real time.

There are many challenges that nowcasting presents to the econometrician, such as mixed frequency data and unbalanced panels. Mixed frequency data is due to the fact that GDP is released quarterly, while other variables that act as predictors of GDP are released monthly. Different publication lags for each variable means that our data is unbalanced at the end of the sample.

A statistical model that addresses all of the aforementioned problems was developed by Giannone et al. (2008). The model is able to condense the big amount of information available by summarizing into a few factors. The researchers also deal with the unbalanced panel by 'chopping' the data so that it is balanced, with the last observation being the one that contains no missing values. In other words, all the observations in the end of the sample for which the value of some series have not yet been released, is removed.

The framework

Our goal is to estimate the current value of GDP, y_q where q is the current quarter. In order to do that, we will use our available information set Ω_v^n, which consists of a set of time-series of monthly frequency. The subscript v denotes the month, up to which our information is available, while n is the number of time-series that belong to the information set. We distinguish the set of time-series into two groups:n_1 and n_2 with $n_1 + n_2 = n$. Variables that belong in the n_1 group are released on time. In other words, the September value for those variables are released in September. Variables that belong in the n_2 group, have their values released with an 1-month lag. Hence, if the month is September, those variables are only known up until month August, because in September their 'August' value becomes known, rather than their 'September' value. In summary, variables of the n_1 group have their v-month values released in month v and variables of the n_2 group have their v-1 month values released in month v. The series that make up the information set are denoted by

$$X_{it|v_j},\ t = 1 \ldots\ldots T_{v_j}, i = 1 \ldots\ldots n$$

Where T_{v_j} is equal to the last month, for which information is available. For variables of the n_2 group, T_{v_j} is equal to $v - 1$, while for variables in the n_1 group, T_{v_j} is equal to v. The notation j represents the latest data release and v_j the corresponding month of that release. It is pretty clear that the information set is expanding with each new data release: $\Omega_{v_j} \supseteq \Omega_{v_{j-1}}$.

We need to introduce some additional timing conventions. We assume that the quarter q is equal to the quarter's last month. Thus, if we are in the first quarter, q is

equal to 3 i.e. March. Hence, if we set $k = 1,2,3,4$ then $q = 3k$. In each, quarter, there are three data releases and hence three different information sets corresponding to those releases. Each new information set is bigger than the last. Those three information sets are denoted by Ω_{v_j}, where $v = 3k - 2, 3k - 1$ and $3k$. For example, if we are at the first quarter, then $k = 1$, and the monthly data releases are $v = 3 - 2 = 1$, $v = 3 - 1 = 2$ and $v = 3$, that is one data release in January, one in February and one in March.

Estimating the model

Central banks and other institutions possess a very big amount of information. Consequently, factor models are exploited in order to utilize all available information. The model at hand is this

$$X_{t|v_j} = \mu + \Lambda F_t + e_{t|v_j} \quad (4.1.1)$$

Where $X_{t|v_j} = (X_{1t|v_j}, X_{2t|v_j}, \dots . X_{nt|v_j})'$ each $X_{it|v_j}$is a monthly time-series. $F_t = (f_{1t}, f_{2t}, \dots . f_{rt})'$ which is the r-factors that drive the variation of the time-series and $e_{t|v_j} = (e_{1t|v_j}, e_{2t|v_j}, \dots e_{nt|v_j})$ which is the variable-specific noise. The variance-covariance matrix of the idiosyncratic components $e_{t|v_j}$ is considered to be diagonal and (possibly) heteroskedastic.

$$Var\left(e_{t|v_j}\right) = diag\left\{\tilde{\psi}_{1t|v_j}, \tilde{\psi}_{2t|v_j}, \dots \tilde{\psi}_{nt|v_j}.\right\}$$

Where $\tilde{\psi}_{1t|v_j} = f(x) = \begin{cases} \psi_{1t|v_j} & , x_{it|v_j} is\ available \\ \infty & , x_{it|v_j}\ is\ not\ available \end{cases}$

When an observation is not available, then the uncertainty is 'infinite'.

The factors follow a $VAR(1)$. A is an $r \times r$ matrix and B a $r \times q$ matrix.

$$F_t = AF_{t-1} + Bu_t \quad (4.1.2)$$

With $u_t \sim N(0, I_q)$ and $E\left(e_{t|v_j} u'_{t-s|v_j}\right) = 0,\ s > 0$, for all v, j

The above model is estimated using the 2-step estimator from chapter **2.1**. A brief revisit (based on the Giannone et al, 2004) of the 2-step estimator follows. We demean the data. At first the factor and loadings are estimated by solving:

$$\min_{F_t, \Lambda} \sum_{t=1}^{T} \sum_{i=1}^{n} \left(x_{it|v_j} - \Lambda_i F_t\right)^2$$

The sample variance-covariance matrix of the idiosyncratic components is computed as

$$\widehat{\Psi} = \frac{1}{T} \sum_{t=1}^{T} X_{t|v_j} X'_{t|v_j} - \hat{\Lambda}' \hat{\Lambda}$$

We use the estimates of the factors $\hat{F}_t$ to estimate the (4.1.2).

$$\hat{A} = \sum_{t=2}^{T} \hat{F}_t \, \hat{F}'_{t-1} \left(\sum_{t=2}^{T} \hat{F}_{t-1} \, \hat{F}'_{t-1} \right)^{-1}$$

We also calculate the sample covariance matrix of the residuals of the VAR(1), $\hat{\Sigma}$ and apply spectral decomposition. We also obtain $\hat{B} = M\sqrt{P}$, where P is the diagonal matrix consisting of the q largest eigenvalues of $\hat{\Sigma}$. M is the $r \times q$ matrix of the corresponding eigenvector. The following model is cast into a state-space form.

$$X_{t|v_j} = \hat{\Lambda} F_t + e_{t|v_j}$$

Which is the measurement equation.

$$F_t = \hat{A} F_{t-1} + \hat{B} u_t$$

Which is the state equation.

$$\widehat{Var(e_{t|v_j})} = diag\hat{\Psi}$$

$$\widehat{u_t} = P^{-1/2} M' (\hat{F}_t - A\hat{F}_{t-1})$$

The factors are re-estimated from the Kalman smoother.

$$\hat{F}_{t|v_j} = E\left(F_t \middle| x_1, \dots x_T; \hat{\theta}\right).$$

Where $\hat{\theta}$ is the vector of the first step's paramater estimates.

The GDP nowcast is computed from an OLS regression of the estimated factors on GDP. We only take into account the sample were GDP is observed. That means $k = 1,2, \dots T_{y|v_j,}$ with $T_{y|v_j,}$ being the last month for which GDP is available.

$$\hat{y}_{3k|v_j} = \gamma_0 + \hat{\gamma}' \hat{F}_{3\kappa|v_j} \qquad (4.1.3)$$

We forecast h quarters ahead, which corresponds to the months $v = 3(k-h) - 2, 3(k-h) - 1, 3(k-h)$. The nowcast is the case of $h = 0$. The residuals are computed, as well as the sample variance-covariance matrix $\widehat{Var(\hat{e}_{3k|v_j})}$.

The forecast error is given by

$$Vy_{3k|v_j} = \hat{\gamma}' \hat{V}_{0|v_j} \hat{\gamma} + \widehat{Var(\hat{e}_{3k|v_j})}. \qquad (4.1.4)$$

Chapter 5

MONTE CARLO AND EMPIRICS

This chapter of the Thesis consists of the implementation, using R, of most of the methods described in the previous chapters. At first we conduct a Monte Carlo simulation, where the methods are compared under a different set of dataset characteristics. Then we proceed to apply those methods on McCracken's large Macroeconomic dataset.

5.1 MONTE CARLO

We have the target variable y and the dataset X which assumes a factor representation. In particular, the number of the factors is $r = 3$. Each factor has a different degree of dominance in the dataset. The least dominant factor is the only relevant, in that it is the only one that affects the target variable y. The factors follow an $AR(1)$.

$f_{it} = g_{f_i} * f_{it-1} + \eta_t,\ \eta_t \sim N(0,1)$

The first and second factors have variances 2.75 and 2.25 times bigger than the relevant factor.

The target equation is generated as

$y = f_{3t} + \sigma_y u_t$

Where $u_t \sim N(0,1)$ and σ_y is chosen in such an way so that the R-squared of the target's regression on the factor is around 0.5-0.6.

Each predictor has the following representation.

$x_{it} = \lambda_{i1} f_{1t} + \lambda_{i2} f_{2t} + \lambda_{i3} f_{3t} + e_t$

Where $e_t = g_e * e_{t-1} + \sigma_e \varepsilon_t$, $\varepsilon_t \sim N(0,1)$ and σ_e is chosen in such an way so that the median R-squared of the factors regression on each predictor is either 0,1 or 0,2 to 0,3. This is done in order to examine different cases of factor strength.

The number of predictors is $N = 100$ and the number of time-series observations is $T + 100$. The added 100 time-series observations are used to obtain recursive one-step ahead forecasts. The following techniques are compared: PCR (with r factors),PLSR(r) (with r factors), 3PRF (with 3 proxies), PLSR(A) and Ridge regression(with the number of factors and shrinkage parameter Lambda respectively chosen by end-of-sample cross-validation algorithms, similar to that described in Groen-Kapetanios-2015). Their mean square error is calculated and compared with a naïve forecast MSE based on the mean of the first T observations.

The following are the result of 100 Monte-Carlo Simulations for each case.

First Case: Normal Factors (median R square of 0.3).

g_e	$g_{f_i}, i = 1,2$	g_{f_3}	RMSE-PCR	RMSE-PLSR(r=3)	RMS-PLSR(A)	RMSE-3PRF	RMSE-RIDGE
0	0	0	0.569	0.663	0.626	0.594	0.622
0	0.3	0.3	0.535	0.635	0.576	0.557	0.596
0.3	0.3	0.3	0.993	0.507	0.535	0.507	0.535
0	0.9	0.3	0.692	0.729	0.746	0.714	0.738
0	0.3	0.9	0.567	0.721	0.633	0.653	0.601

0.9	0.9	0.3	1.025	0.622	0.626	0.623	0.640
0.9	0.3	0.9	0.493	0.457	0.477	0.458	0.457

Second Case: Weak Factors (median R square of 0.1)

g_e	$g_{f_i}, i = 1,2$	g_{f_3}	RMSE-PCR	RMSE-PLSR	RMS-PLSR(A)	RMSE-3PRF	RMSE-RIDGE
0	0	0	0.813	1.03	0.840	0.900	0.823
0	0.3	0.3	0.711	0.919	0.682	0.778	0.762
0.3	0.3	0.3	0.989	0.531	0.583	0.532	0.596
0	0.9	0.3	0.983	1.204	1.111	1.059	0.951
0	0.3	0.9	0.534	0.754	0.522	0.661	0.597
0.9	0.9	0.3	1.018	0.711	0.771	0.711	0.813
0.9	0.3	0.9	0.445	0.417	0.450	0.423	0.440

Third Case: Moderately Weak, Non-pervasive Factors (Median R squared of 0.2, half of the predictors have zero loadings on the relevant factor).

g_e	$g_{f_i}, i = 1,2$	g_{f_3}	RMSE-PCR	RMSE-PLSR	RMS-PLSR(A)	RMSE-3PRF	RMSE-RIDGE
0	0	0	0.622	0.729	0.689	0.642	0.6803
0	0.3	0.3	0.55	0.62009	0.604	0.571	0.695
0.3	0.3	0.3	1.008	0.593	0.629	0.596	0.653
0	0.9	0.3	0.945	0.993	1.031	0.960	0.901
0	0.3	0.9	0.438	0.565	0.461	0.487	0.497
0.9	0.9	0.3	1.012	0.706	0.704	0.716	0.696
0.9	0.3	0.9	0.744	0.485	0.431	0.483	0.534

We can see that traditional Principal Components do especially well when the errors are not serially correlated. When there is though serial-correlation on the errors PCR does really poorly.

3PRF and Partial Least Squares perform well under a variety of circumstances, particularly when the relevant factor's dynamics are dominant in the dataset.

Ridge regression has a middle-of-the-road performance, usually being not the worse but also not the best among all the methods.

5.2 EMPIRICAL EXERCISE

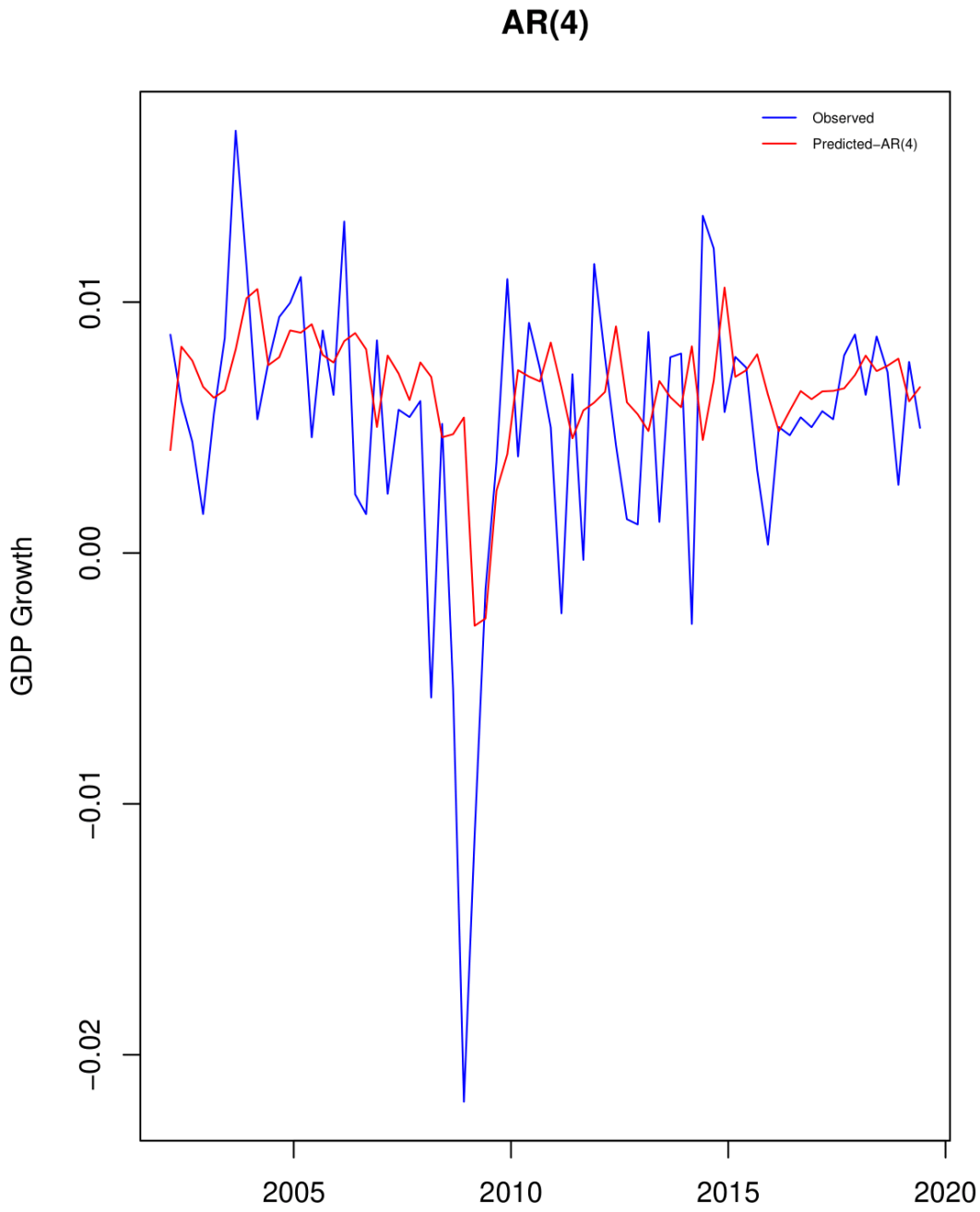
On this section, we apply all of the various High-Dimensional Techniques on the McCracken dataset.

The McCracken dataset consists of 243 quarterly observations of 205 variables (after having removed those that contain missing values). The observations start at the third quarter of 1959 and end at the second quarter of 2019. After we have applied all of the necessary stationarity transformations, we can proceed with the one-step ahead forecasting exercises. The various models will be compared with an AR(4).

The first forecasting exercise concerns GDP and it starts at 2001-12-01. In other words , the data available up to 2001-12-01 is used to obtain the GDP forecast for the

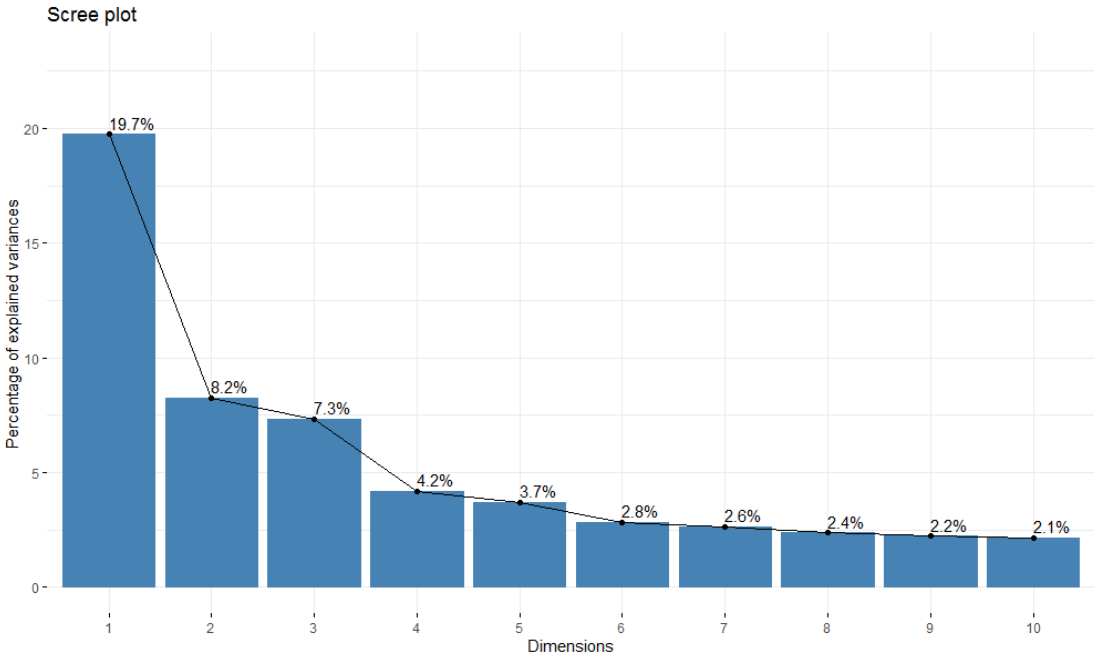
next period (the first quarter of 2002), then, the predicted GDP value is then compared with the actual GDP value and so forth until the end of the dataset. The Mean Squared error is subsequently calculated.

The first model, which will be used as a benchmark, is a simple AR(4) of GDP. All of the resulting MSE's will be compared with the MSE of the AR(4). The AR(4) predicted values against the actual values can be seen in the diagram below.



Principal Components

At first, we apply Principal Components on the dataset. The corresponding Scree plot is this:

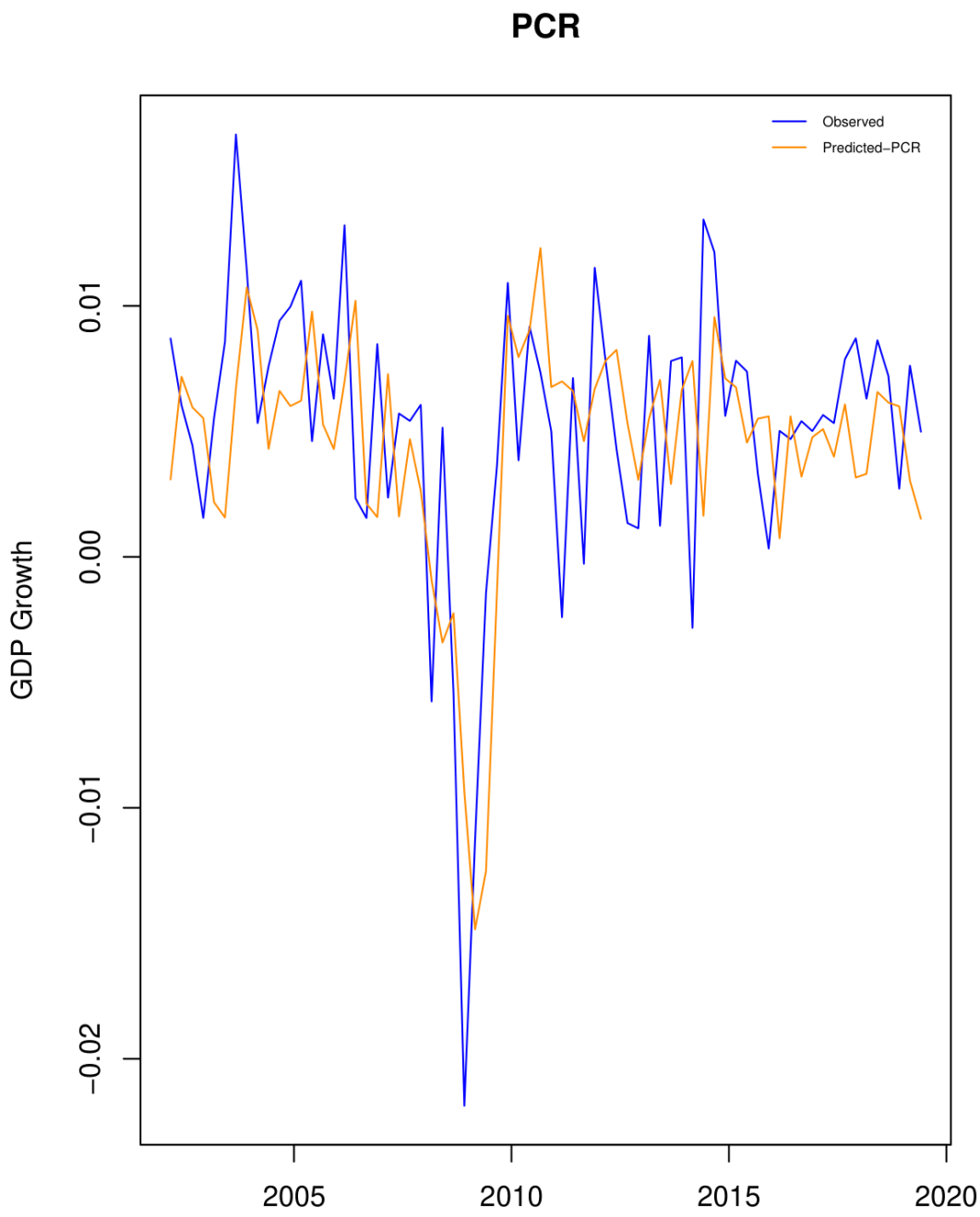


As we can see, the first Principal Component explains 19.7% of the variation in the dataset, the second 8.2% etc. By applying Bai and Ng's criterion, we deduce that the number of factors in the dataset is 6. Those 6 factors explain 45.98707% of the variation in the dataset. By regressing each of the variables in the dataset against the Principal components we get a separate R square value for each variable. The median R square is 0.3441191, which suggests that the dataset exhibits a normal factor structure.

Models considered

Principal Component Regression

We apply recursive one-step ahead forecasting with Principal Component regression (PCR) and obtain the resulting RMSE (relative to the AR(4) MSE). The exercise is done using a different number of components around the number established by the criterion and choosing the number of factors that produces the minimum RMSE, which is 5. The resulting RMSE of PCR(5) is 0.7746326. Below is the plot of the actual values vs the fitted values.



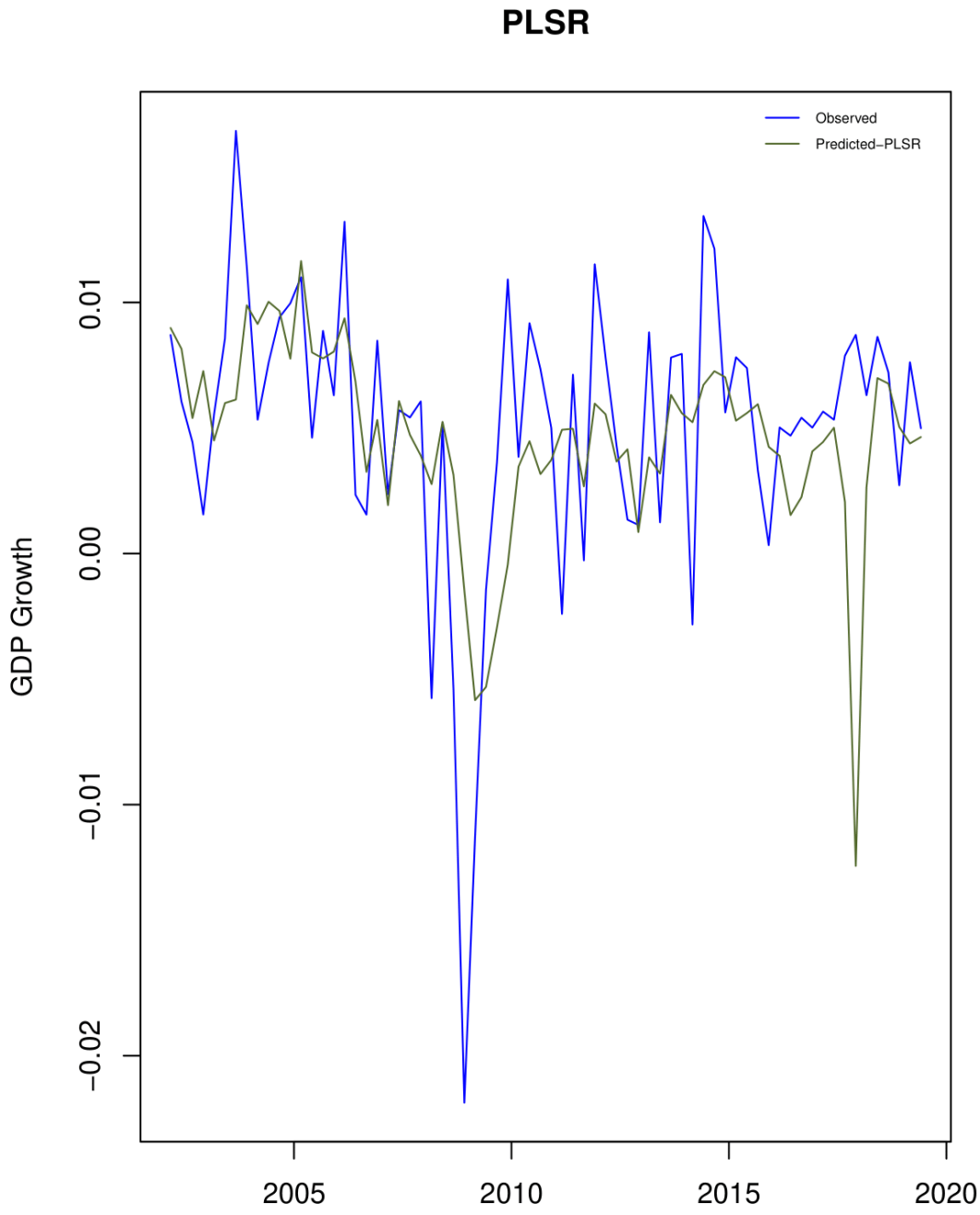
Squared Principal Components

SPC is suitable for capturing possible non-linearities in the dataset. This method is used in Bain, Ng (2017). SPC is done by adding extra columns in the X matrix with the squares of each element and then extracting the Principal Components as usual. In more mathematical terms, we have $X_t^* = \{X_{it}, X_{it}^2\}$. Just like with PCR, we try a different number of principal components and choose the number that results in the least RMSE.

Using 5 factors from Squared Principal Components results in the least RMSE (which is 0.7207909). There is a slight improvement relative to PCR.

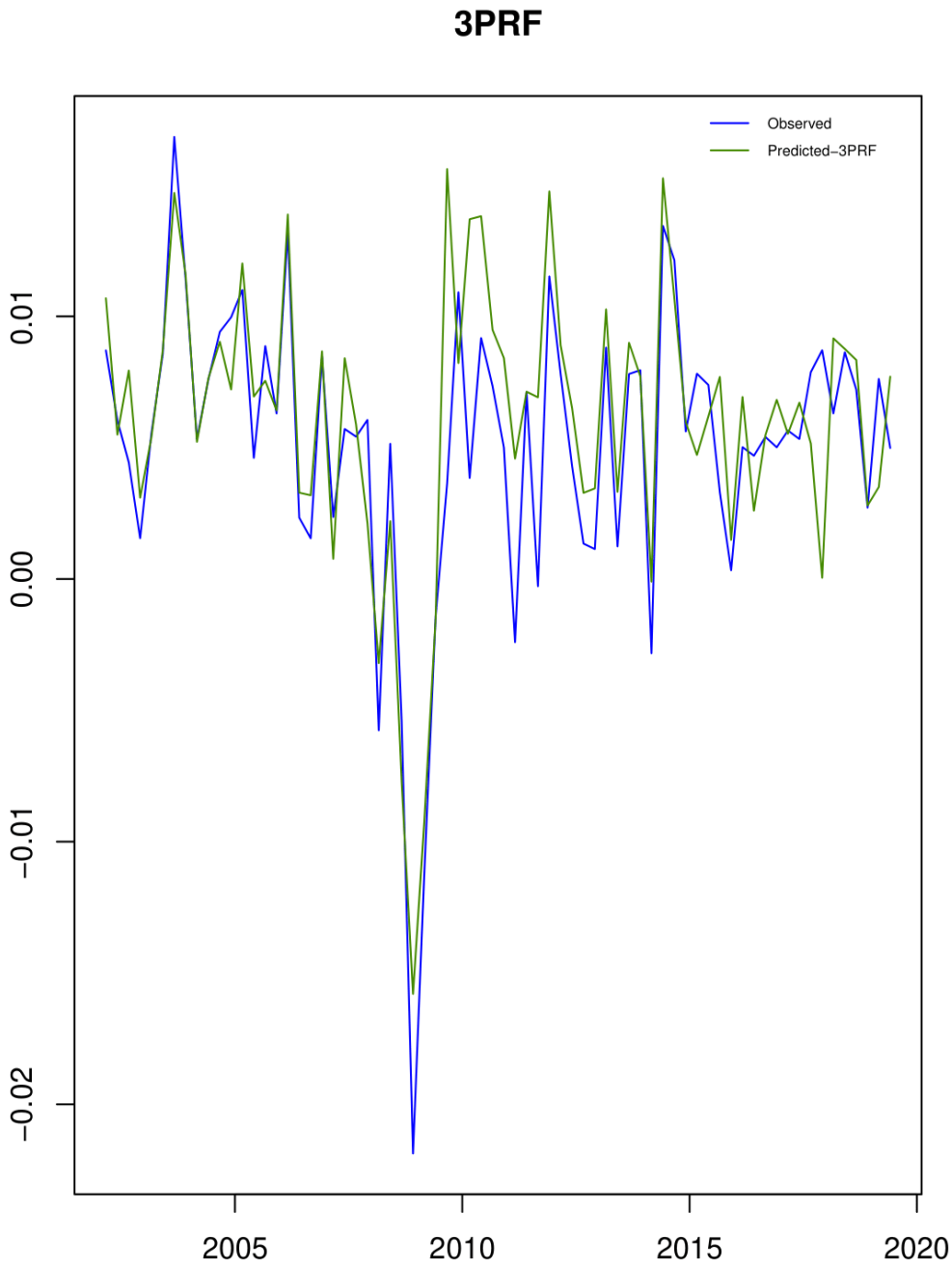
Partial Least Squares

PLSR produces a RMSE of 0.8985426. The corresponding plot shows that PLSR basically collapses between the years 2015 and 2020.



Three Pass Regression Filter

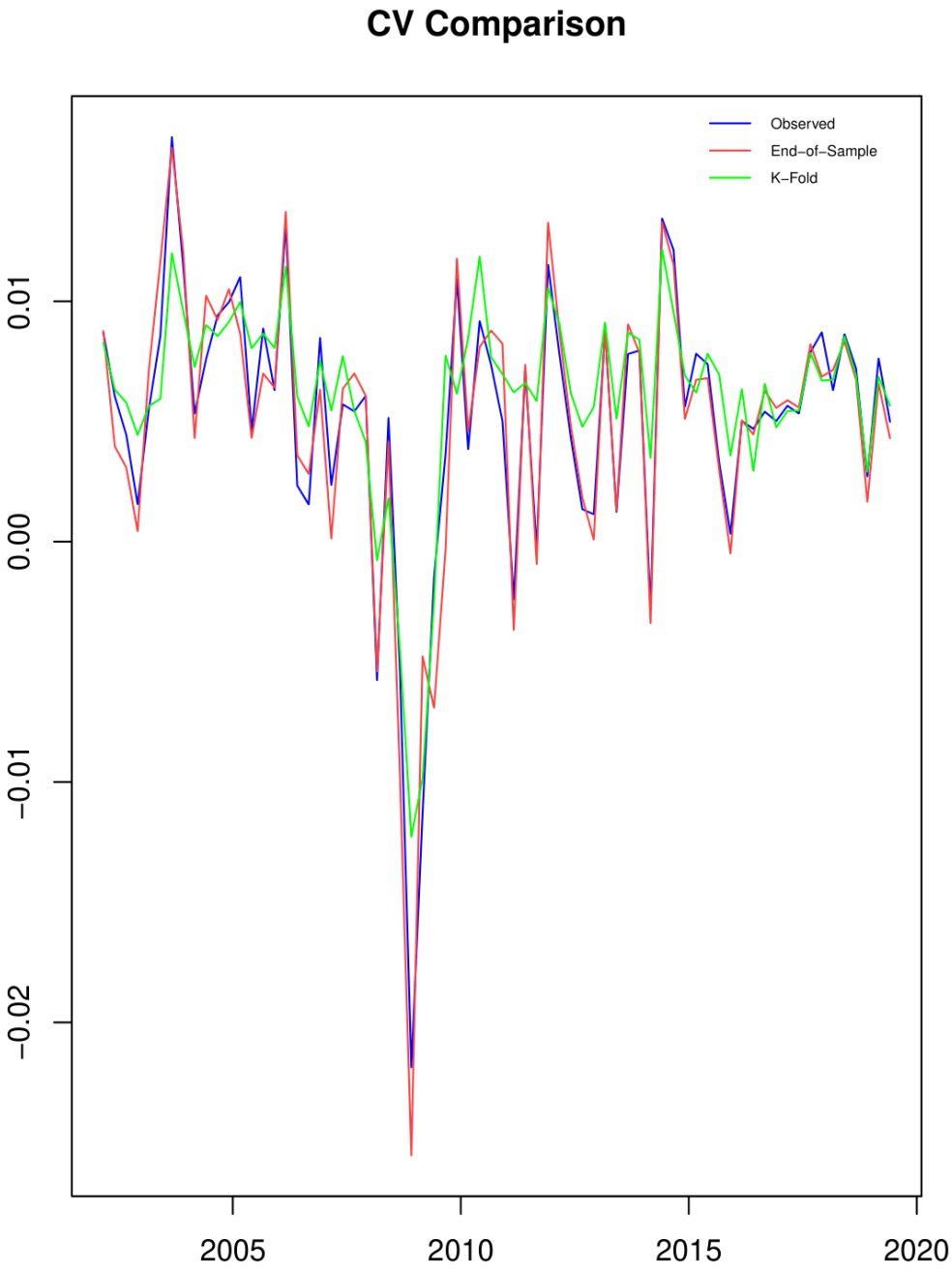
The Three-Pass regression filter, which we have already seen, is particularly efficient when the dominant factors in the dataset are not relevant for the forecast target. 3PRF gives us a RMSE of 0.3370435, thus achieving a far superior result relative to PCR and SPCR. The resulting plot of the predicted GDP growth values is:



Ridge Regression

Before applying Ridge regression we need to choose a value for the shrinkage parameter. The first step towards choosing a shrinkage parameter is creating a vector of values for the shrinkage parameter which is done as in Friedman, Hastie, Tibshirani (2010). Then, the value of the shrinkage parameter that will be used for the forecasting exercise is chosen according to the algorithm described in Groen, Kapetanios (2015). This algorithm does one-step ahead forecasting for the last 20 values of the sample for each value of the shrinkage parameter and chooses the value that results in the least MSE. After the algorithm selects an optimal value for the shrinkage parameter we proceed with the one-step ahead forecasting exercise for the next 70 values, the emerging RMSE is 0.09840685, vastly outperforming PCR, SPCR and 3PRF.

We also apply Ridge regression using R's glmnet package. The tuning parameter is chosen with K-fold cross-validation instead of end-of-sample cross-validation. K-fold cross-validated ridge regression gives us a RMSE of 0.2642736, which is lower than end-of-sample cross-validated Ridge regression. This is to be expected, since K-fold cross-validation is not suitable for Time-Series. The two methods are compared in the diagram.



LARS selection

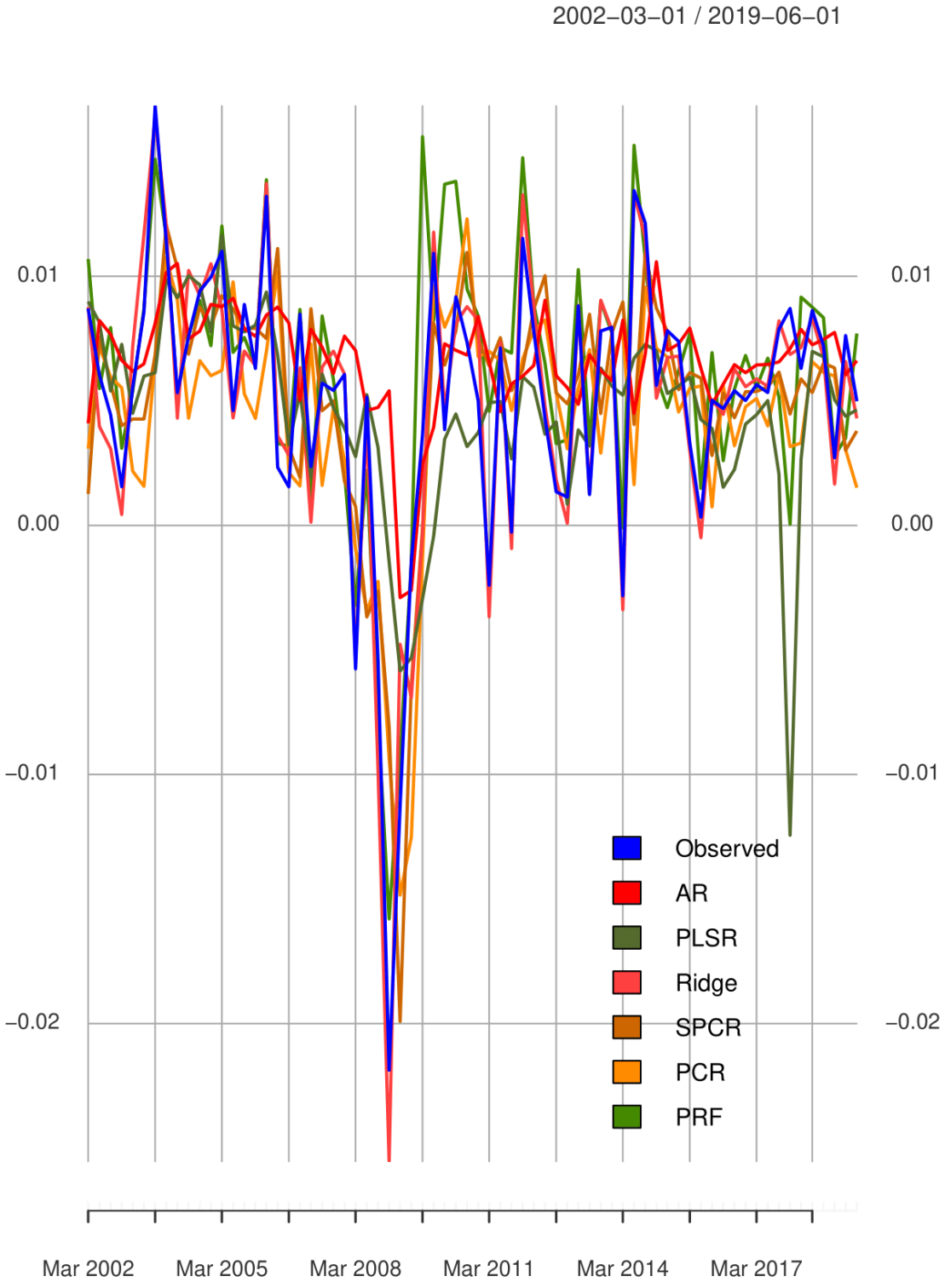
By applying LARS to the dataset, we are able to obtain the predictors most relevant for the target, as in Bai,Ng (2008). Then we can extract factors from the smaller dataset or regress the relevant predictors directly. At first we select the first 5 most relevant predictors and regress them directly against GDP. We denote this method as LARS(5). we get an RMSE of 1.285376. Then we select the 50 most relevant predictors and apply PCR on this subset of the data. The procedure is denoted as PCR-LARS(50).The resulting

RMSE using 3 extracted factors is 0.9063368. The inferiority of those methods suggests that the variables discarded do carry important information for the target.

Summary

PCR(5)	SPCR(5)	PLSR	3PRF	Ridge	LARS(5)	PCR-LARS(50)
0.7746326	0.7207909	0.8985426	0.3370435	0.09840685	1.285376	0.9063368

End-of-sample cross-validated ridge, PLSR, PCR, SPCR, 3PRF and the AR(4) are plotted against each other on the diagram.


2002–03–01 / 2019–06–01
0.01
0.00
–0.01
–0.02
Observed
AR
PLSR
Ridge
SPCR
PCR
PRF
Mar 2002
Mar 2005
Mar 2008
Mar 2011
Mar 2014
Mar 2017

APPENDIX (R-CODE)

R code

Below is the R code that was used in the thesis. The code is essential for replicating the analysis done in the empirical section and the simulations of the thesis.

Forecast functions

This piece of code contains several one-step ahead functions. Those functions are used in the analysis of the McCracken dataset.

The common arguments among all functions are the t.start and t.end arguments, which indicate the start of the one-step ahead forecast exercise (t.start) and the end of the forecast exercise (t.start+t.end). The function's output includes the predicted values and the MSE.

The roll.forecast.mse.2 relies on the pls package. The extra argument is the number of factors. The 3PRF based function relies on IshmaelBengazi's code on github. roll.forecast.arima uses the forecast package. The extra arguments are p,q and k that denote the order of the ARIMA fit.

```
library(forecast)
library(glmnet)
library(pls)



#Recursive Forecasting



#ols recursive forecast


roll.forecast.ols=function(X.pred,y.target,df,t.start,t.end){
  rolling=c()
  for (t in 0:(t.end-1)){
    lm1=lm(y.target[1:(t.start+t)]~X.pred[1:(t.start+t),],data = df)
    nea.data=df$X.pred[(t.start+t):(t.start+t+1),]
    pred=predict(lm1,newdata=nea.data)
    rolling[t+1]=pred[length(pred)]
  }
  mse=mean((rolling-y.target[(t.start+1):(t.end+t.start)])^2)
  output=list('mse'=mse,'predicted.y'=rolling)
}




roll.forecast.mse.2=function(X.pred,y.target,t.start,t.end,n.factors){
  rolling=c()
  for (t in 0:t.end-1){
    plsr1=plsr(y.target[1:(t.start+t)]~X.pred[1:(t.start+t),],ncomp=n.factors)

pred=predict(plsr1,newdata=X.pred[(t.start+t):(t.start+t+1),],type='response',ncomp=n
    rolling[t+1]=pred[length(pred)]
  }
  mse=mean((rolling-y.target[(t.start+1):(t.end+t.start)])^2)
  output=list('mse'=mse,'predicted.y'=rolling)

}









#Three Pass Regression Filter


roll.forecast.mse.prf=function(X.pred,y.target,t.start,t.end,n.proxies=2,Z.proxies=NU
  rolling=c()
  for (t in 0:t.end-1){
    threeprf=TPRF(X.pred[1:(t.start+t),],y.target[1:(t.start+t)],L=n.proxies,Z=Z.prox
    pred=predict(threeprf,newdata=X.pred[(t.start+t):(t.start+t+1),])
    rolling[t+1]=pred[length(pred)]
  }
  mse=mean((rolling-y.target[(t.start+1):(t.end+t.start)])^2)
  output=list('mse'=mse,'predicted.y'=rolling)

}
```

```
#PCR

pcr.rolling=function(X.pred,y.target,n.pcs,t.start,t.end,pcr.scale=TRUE){
  rolling=c()
  for (t in 0:t.end-1){
    pc.obj=prcomp(X.pred[1:(t.start+t),],scale = pcr.scale)
    pc.comp=pc.obj$x[,1:n.pcs]
    lm.obj=lm(y.target[1:(t.start+t)]~pc.comp)
    pc.obj2=prcomp(X.pred[1:(t.start+t+1),],scale = pcr.scale)
    pc.comp2=pc.obj$x[,1:n.pcs]
    pred=predict(lm.obj,newdata = data.frame(pc.comp2))
    rolling[t+1]=pred[length(pred)]
  }
  mse=mean((rolling-y.target[(t.start+1):(t.end+t.start)])^2)
  output=list('mse'=mse,'predicted.y'=rolling)
}




#Arima

roll.forecast.arima=function(series,t.start,t.end,horizon=1,p,q,k){
  rolling=c()
  for (t in 0:t.end-1){
    model=Arima(series[1:(t.start+t)],order=c(p,q,k))
    pred=forecast(model,h=horizon)
    rolling[t+1]=pred$mean[1]
  }
  mse=mean((rolling-series[(t.start+1):(t.end+t.start)])^2)
  output=list('mse'=mse,'predicted.y'=rolling)

}
```

Squared Principal Components

This is the code for Squared Principal Components. It has three functions. The first function (SPC) does Squared Principal Components on a dataset using the prcomp function of R base. The function creates a new matrix that has twice the columns of the dataset matrix. The new columns contain the elements of the original matrix squared. The function returns all the arguments that prcomp does, as well as the dimensions of the new matrix. The scale argument (as in prcomp) indicates whether the variables should be scaled to have unit variance.

The second function is called SPCR (Squared Principal Components Regression) and it is an extension of the first function. It uses SPC to extract squared principal components from a dataset and then uses those squared principal components as regressors. The arguments are the target variable (y.target), the predictors' matrix or dataset and the number of components.

The third function does recursive, one step ahead forecast using Squared Principal Component regression. The arguments are the target, the predictors' matrix, the period at which the forecasting exercise starts (t.start) and the number of periods after t.start at which it ends (t.end). The user also chooses the number of squared principal components to be used. The predicted target values, as well as the mean square error of the forecast is returned.
For more information on Squared Principal Components, See Bai,Ng (2008), 'Forecasting economic time series using targeted predictors'

```
#Squared Principal Components



#Squared Principal Components is done by creating a new X matrix and then
#doing usual PCA to the new matrix.
#The new matrix has thise elements
#X_it={X_it^2,X_it}




SPC=function(X,f.scale){

T=dim(X)[1]
N=dim(X)[2]
X_2=matrix(nrow=T,ncol=N)

for (n in 1:N){
  for (t in 1:T){
    X_2[t,n]=(X[t,n])^2
  }
}


X_new=cbind(X_2,X)
dimensions=dim(X_new)
prc=prcomp(X_new,scale. = f.scale)

output=list('sdev'=prc$sdev,'rotation'=prc$rotation,'x'=prc$x,'center'=prc$center,
'new.dimensions'=dimensions)
return(output)
}




#Squared Principal Components Regression (SPCR)




SPCR=function(X.pred,y.target,n.spcs,spcr.scale=TRUE){
  spc.obj=SPC(X.pred,f.scale = spcr.scale)
  spc.comp=spc.obj$x[,1:n.spcs]
  spcr.obj=lm(y.target~spc.comp)
  return(spcr.obj)
}
```

```
#SPCR recursive one-step ahead forecast function
#X.pred is the predictors matrix and y.target the target variable
#n.spcs is the number of principal components to be used in the regression
#t.start is the point at which the forecasting procedure begins, t.end denotes for
# how many periods after t.start
#should the forecasting procedure be applied.
#In other words, the forecasting exercise begins at t.start and ends at t.start+t.end

spcr.rolling=function(X.pred,y.target,n.spcs,t.start,t.end,spcr.scale=TRUE){
  rolling=c()
  for (t in 0:t.end-1){
  spc.obj=SPC(X.pred[1:(t.start+t),],f.scale = spcr.scale)
  spc.comp=spc.obj$x[,1:n.spcs]
  lm.obj=lm(y.target[1:(t.start+t)]~spc.comp)
  spc.obj2=SPC(X.pred[1:(t.start+t+1),],f.scale = spcr.scale)
  spc.comp2=spc.obj$x[,1:n.spcs]
  pred=predict(lm.obj,newdata = data.frame(spc.comp2))
  rolling[t+1]=pred[length(pred)]
  }
  mse=mean((rolling-y.target[(t.start+1):(t.end+t.start)])^2)
  output=list('mse'=mse,'predicted.y'=rolling)
}



#the output is the vector of predicted y values, as well as the mean square
#error of the forecast
```

Functions for Lars/lasso based on the lars package.

There is a function for determining the optimal value for the shrinkage parameter through end-of-sample cross-validation. This function can be combined with the lars/lasso variable selection functions.

There are also two one-step ahead forecast functions. One that chooses a different value for lambda at each step and one that uses one value for lambda throughout the forecasting exercise.

```
library(lars)


#End of sample CV for LARS/LASSO
#This function does end of sample cross-validation for either LARS or LASSO
#and it is based on the lars package.
#In particular, the function calculates the MSE of an one-step ahead forecast for
#the last eval.obs observations of the dataset X on the target y. Then the lambda tha
results
#in the least MSE is chosen.
#The function takes as arguments the dataset X, the target y, and the
#the number of observations in the end of the sample that will be used
#in the evaluation. The set of lambdas that will be evaluated is
#given by the user
#The type argument is as in the lars function, lasso or lars can be chosen.

end_of_sample_cv_lars=function(X,y,eval.obs,lambdas,type){

T=dim(X)[1]

mres=c()
for (h in 1:length(lambdas)){

  lmbd=lambdas[h]
   rolling=c()
   j=0
for (t in (T-(eval.obs+1)):(T-1)){
  j=j+1

  lars.fit=lars(X[1:t,],y[1:t],type=type)

  lars.pred=predict(lars.fit,type='fit',mode='lambda',s=lmbd,newx=X[t:(t+1),])
  rolling[j]=lars.pred$fit[length(lars.pred$fit)]
}

mres[h]=mean((rolling-y[(T-eval.obs):T])^2)


}
list_output=list('optimal.lambda'=lambdas[which.min(mres)],'MSEs'=mres)

return(list_output)


}




#Lars recursive forecast function with end of sample CV at each period
#The arguments are the predictors matrix, the target, the time period where the
#the forecasting exercise begins (t.start) and the number of periods after the start
that will be
#included in the exercise (t.end). The user also provides the function with a
#set of lambdas (with a value for lambda chosen at each period) and the number of
observations
#that the end of sample cv alrogithm uses to choose a value for lambda
```

```
lars.forecast=function(X.pred,y.target,t.start,t.end,lambdas,type_1='lasso',eval.obs)
  rolling=c()
  for (t in 0:(t.end-1)){
    optimal.lambda=end_of_sample_cv_lars(X.pred[1:(t.start+t),],
y.target[1:(t.start+t)],lambdas=lambdas,type=type_1,eval.obs=eval.obs)
    fit=lars(X.pred[1:(t.start+t),],y.target[1:(t.start+t)],type = type_1)
    pred=predict(fit,newx=X.pred[(t.start+t):(t.start+t+1),],
s=optimal.lambda,mode='lambda')
    rolling[t+1]=pred$fit[length(pred$fit)]
  }
  mse=mean((rolling-y.target[(t.start+1):(t.end+t.start)])^2)
  output=list('mse'=mse,'predicted.y'=rolling)

}
```

Lars/Lasso selection

This function does variable selection out of a matrix of predictors. The function is based on the lars package. The arguments are the matrix of predictors, the target variable y, the number of variables to be extracted and the value of the shrinkage parameter lambda to be used. There are end-of-sample-CV functions that come up with an optimal value for lambda. The output is a new matrix consisting of the chosen subset of variables.

```
library(lars)


#lars/lasso variable selection
#The function chooses the most important predictors out of a predictors
#matrix using lars or lasso. The chosen variables are sorted based on their
importance
#lambda is the value of the shrinkage parameter and it is entered
#by the user.
#n.vars is how many variables should be extracted.
#The output of the function is the matix of the chosen variables.



lars.selection.b=function(X.pred,y.target,lambda,n.vars,type1='lar'){

  lar.fit=lars(X.pred,y.target,type=type1)
```

```
lar.pred=predict(lar.fit,s=lambda,mode='lambda',type='coefficients',newx=X.pred)
  lar.coef=lar.pred$coefficients[lar.pred$coefficients!=0]
  epil.names=names(sort(lar.coef,decreasing = TRUE)[1:n.vars])
  return(X.pred[,epil.names])

}

######
lars.selection.c=function(X.pred,y.target,lambda,n.vars,type1='lar'){
  lar.fit=lars(X.pred,y.target,type=type1)

lar.pred=predict(lar.fit,s=lambda,mode='lambda',type='coefficients',newx=X.pred)
  lar.coef=lar.pred$coefficients[lar.pred$coefficients!=0]
  epil.names=names(sort(lar.coef,decreasing = TRUE)[1:n.vars])
  return(epil.names)

}
```

Ridge regression

The 'own ridge' code is a set of functions for applying ridge regression on a dataset. Besides the functions for ridge regression and prediction , there is a function for generating a set of values for the shrinkage parameter (lambda or v) and a function for end of sample cross-validation. End of sample CV is preferable to K-fold when dealing with Time-series. There are also two function for recursive one-step ahead forecast, one that searches for the optimal lambda (or v) at each iteration and another one that uses one value of lambda for all iterations.

```
#Ridge regression


ridge=function(y,X,v){
N=dim(X)[2]
a_ridge=(solve(t(X)%*%X+v*diag(N),tol = 1e-28))%*%(t(X)%*%y)
y_hat=X%*%a_ridge
output_list=list('y_hat'=y_hat,'coef'=a_ridge)
return(output_list)
}
```

```
#Ridge prediction function


ridge.predict=function(model,newx){
  y_new=newx%*%model$coef
  return(y_new)
}


#end of sample cv of ridge regression
#The values of the shrinkage parameter that are considered are
# entered by the user through the lambdapath function (hastie,  tibshirani)
#Last last.eval observations are used to for out of sample one step
#ahead forecasting, the shrinkage parameter that results in the
#least mse is chosen


#Function for creating lambda path
lambdapath=function(X,y,K,epsilon,mult.max.lambda=1){

#Compute maximum lambda

sx <- as.matrix(scale(X))
sy <- as.vector(scale(y))
lambda_max=mult.max.lambda*max(abs(colSums(sx*sy)))/100

#create sequence of lambdas

lambdapath <- round(exp(seq(log(lambda_max), log(lambda_max*epsilon),
                            length.out = K)), digits = 10)
return(lambdapath)
}

end_of_sample_cv_ridge=function(X,y,eval.obs,v_param){
T=dim(X)[1]

mres=c()
for (i in 1:length(v_param)){
v1=v_param[i]




j=0
rolling=c()
for (t in (T-(eval.obs+1)):(T-1)){
  j=j+1
  fit=ridge(y[1:t],X[1:t,],v=v1)
  pred=ridge.predict(fit,newx =X[(t+1),] )
  rolling[j]=pred
}

mres[i]=mean((rolling-y[(T-eval.obs):(T)])^2)

}

return(v_param[which.min(mres)])
```

```
}




#Function that combines end of sample cv and ridge so that it picks the best
#value of the shrinkage parameter and then proceeds with the regression

cv.ridge=function(y,X,v_param,eval.obs){
  optimal.v=end_of_sample_cv_ridge(y=y,X=X,v_param = v_param,eval.obs =
eval.obs)
  fit=ridge(y=y,X=X,v=optimal.v)
  return(list('fit'=fit,'optimal.v'=optimal.v))
}



#Recursive one step ahead ridge forecast with end of sample cv



ridge.forecast=function(X.pred,y.target,t.start,t.end,eval.obs,v_param ){
  rolling=c()
  for (t in 0:(t.end-1)){

ridge.reg=cv.ridge(X=X.pred[1:(t.start+t),],y=y.target[1:(t.start+t)],v_param
=v_param ,eval.obs = eval.obs)
    pred=ridge.predict(ridge.reg$fit,newx=X.pred[(t.start+t+1),])
    rolling[t+1]=pred
  }
  mse=mean((rolling-y.target[(t.start+1):(t.end+t.start)])^2)
  output=list('mse'=mse,'predicted.y'=rolling)

}


#Recursive one step ahead ridge forecast with one lambda

ridge.forecast.b=function(X.pred,y.target,t.start,t.end,v ){
  rolling=c()
  for (t in 0:(t.end-1)){
    ridge.reg=ridge(X=X.pred[1:(t.start+t),],y=y.target[1:(t.start+t)],v =v)
    pred=ridge.predict(ridge.reg,newx=X.pred[(t.start+t+1),])
    rolling[t+1]=pred
  }
  mse=mean((rolling-y.target[(t.start+1):(t.end+t.start)])^2)
  output=list('mse'=mse,'predicted.y'=rolling)

}



#same function as above but returns the set of mses

end_of_sample_cv_ridge.b=function(X,y,eval.obs,v_param){
  T=dim(X)[1]
```

```
  mres=c()
  for (i in 1:length(v_param)){
    v1=v_param[i]




    j=0
    rolling=c()
    for (t in (T-(eval.obs+1)):(T-1)){
      j=j+1
      fit=ridge(y[1:t],X[1:t,],v=v1)
      pred=ridge.predict(fit,newx =X[(t+1),] )
      rolling[j]=pred
    }

    mres[i]=mean((rolling-y[(T-eval.obs):(T)])^2)

  }

  return(mres)


}
```

Entering the data and transforming the time series.

The 'transformation of time series' piece of code applies all the appropriate transformations in the series in order to achieve stationarity. Those transformations are as in McCracken's website. The excel file has the transformations code as a last time series observation, which is then removed.

```
library(dplyr)
library(tidyr)
library(forecast)
library(lattice)
library(lubridate)
library(ggplot2)
library(zoo)
library(xts)
library(factoextra)
library(pls)
library(nowcasting)
library(readxl)
library(glmnet)
library(lars)

library(remotes)



setwd("D:/Desktop/programming/r/R/mcracken")


current_copy_2 <- read_excel("current_copy_2.xlsx")


current_copy_2=current_copy_2%>%
  mutate(sasdate=ymd(sasdate))

variables_names=colnames(current_copy_2)
new_vars=c()


for (i in variables_names){
  j=paste('ts',sep="_",i)
  print(j)
  new_vars=c(new_vars,j)
}



for (j in 2:dim(current_copy_2)[1]){
  assign(new_vars[j],xts(current_copy_2[,j],order.by =
current_copy_2$sasdate))

}

rm(ts_NONBORRES)
ts_AAA[1:length(ts_AAA)-1]



trans_function=function(ts){
  num=0
  if (is.xts(ts)){
    num=as.numeric(ts[length(ts)])
    ts=ts[1:length(ts)-1]

    if (num==2){
      ts_tel<-diff(ts)
```

```
    } else if (num==3){
      ts_tel<-diff(ts,differences = 2)

    } else if (num==4){
      ts_tel=log(ts)
    } else if (num==5){
      ts_tel=diff(log(ts))
    } else if (num==6){
      ts_tel=diff(log(ts),differences = 2)
    } else if (num==1){
      ts_tel=ts
    }
    return(ts_tel)}
  else {
    print('not a time series object')
  }

}



results = lapply(setNames(ls(), ls()), function(i) {
  x = get(i)
  if(is.xts(x)) {
    trans_function(x)
  }
})

results = results[!sapply(results, is.null)]

df=data.frame(results)

df=df[3:dim(df)[1],]

df=df%>%
  select_if(~!any(is.na(.)))

rownames(df)=ymd(rownames(df))

#Delete everything starting with ts_
rm(list = ls()[grep("^ts", ls())])
```

On the Empirics1 file, I apply High-Dimensional Forecasting techniques on the McCracken dataset.The McCracken dataset consists of 243 quarterly observations of 205 variables (after having removed those that contain missing values). The observations start at the third quarter of 1959 and end at the first quarter of 2020. After I have applied all of the necessary stationarity transformations, I proceed with the one-step ahead forecasting exercises. The various models will be compared with an AR(4). The first forecasting exercise

concerns GDP and it starts at 2001-12-01. In other words , the data available up to 2001-12-01 is used to obtain the GDP forecast for the next period (the first quarter of 2002), then, the predicted GDP value is then compared with the actual GDP value and so forth until the end of the dataset. The Mean Squared error is subsequently calculated. The first model, which will be used as a benchmark, is a simple AR(4) of GDP.

```
library(pls)
library(nowcasting)
library(factoextra)
library(zoo)
library(xts)


#PCA in the data


vector_y=matrix(df$GDPC1,ncol=1,nrow=243)
matrix_X=as.matrix(df[,-grep('GDPC1',colnames(df))])

pc1=prcomp(matrix_X,scale. = TRUE)



fviz_eig(pc1, addlabels=TRUE)+ylim(c(0,23))


#Estimated number of factors is 7
ICfactors(matrix_X,type = 2)


#Percentage of explained variance of those 7 factors.
get_eig(pc1)[1:6,]

#Get the Principal Components
p.components=pc1$x[,1:6]

#Strenth of factor structure
rsquares=c()
for (i in 1:N){
  rsquares[i]=summary(lm(matrix_X[,i]~p.components))$r.squared
}

print(median(rsquares))

#AR(4)
#Create an AR for GDP



gdpc=as.xts(df$GDPC1,order.by = current_copy_2$sasdate[3:245])


auto.arima(gdpc)

#Forecasting exercise

ar_4=roll.forecast.arima(gdpc,t.start = 170,t.end=70,p=4,q=0,k=0,horizon=1)
```

```
ar_4$mse


date=ymd(rownames(matrix_X)[171:240])

zoo1=zoo(vector_y[171:240],order.by = date)
zoo2=zoo(ar_4$predicted.y,order.by = date)

plot.zoo(cbind(zoo1, zoo2),
         plot.type = "single",
         col = c("blue", "red"),main='AR(4)',ylab='GDP Growth',xlab='')

legend("topright", inset=c(0,0), y.intersp = 1, legend =
c("Observed","Predicted-AR(4)"),  lty = 1, bty = "n", col = c("blue", "red"),
cex = .5)







#PLSR
plsr=roll.forecast.mse.2(X.pred=matrix_X,y.target = vector_y,t.start =
170,t.end=70,n.factors  =7)


plsr$mse/ar_4$mse


zoo2=zoo(plsr$predicted.y,order.by = date)

plot.zoo(cbind(zoo1, zoo2),
         plot.type = "single",
         col = c("blue", "darkolivegreen"),main='PLSR',ylab='GDP
Growth',xlab='')

legend("topright", inset=c(0,0), y.intersp = 1, legend =
c("Observed","Predicted-PLSR"),  lty = 1, bty = "n", col = c("blue",
"darkolivegreen"), cex = .5)



#3PRF

prf1=roll.forecast.mse.prf(X.pred=matrix_X,y.target=vector_y,t.start =
170,t.end=70,n.proxies=2)

prf1$mse/ar_4$mse



zoo2=zoo(prf1$predicted.y,order.by = date)

plot.zoo(cbind(zoo1, zoo2),
         plot.type = "single",
         col = c("blue", "chartreuse4"),main='3PRF',ylab='GDP
Growth',xlab='')

legend("topright", inset=c(0,0), y.intersp = 1, legend =
```

```
c("Observed","Predicted-3PRF"),  lty = 1, bty = "n", col = c("blue",
"chartreuse4"), cex = .5)



spcr=spcr.rolling(X.pred=data.frame(matrix_X),y.target=vector_y,t.start =
170,t.end=70,n.spcs=5)

spcr$mse/ar_4$mse



zoo2=zoo(spcr$predicted.y,order.by = date)

plot.zoo(cbind(zoo1, zoo2),
         plot.type = "single",
         col = c("blue", "darkorange3"),main='SPCR',ylab='GDP
Growth',xlab='')

legend("topright", inset=c(0,0), y.intersp = 1, legend =
c("Observed","Predicted-SPCR"),  lty = 1, bty = "n", col = c("blue",
"darkorange3"), cex = .5)




prcr=pcr.rolling(X.pred=data.frame(matrix_X),y.target=vector_y,t.start =
170,t.end=70,n.pcs=5)

prcr$mse/ar_4$mse



zoo2=zoo(prcr$predicted.y,order.by = date)

plot.zoo(cbind(zoo1, zoo2),
         plot.type = "single",
         col = c("blue", "darkorange"),main='PCR',ylab='GDP Growth',xlab='')

legend("topright", inset=c(0,0), y.intersp = 1, legend =
c("Observed","Predicted-PCR"),  lty = 1, bty = "n", col = c("blue",
"darkorange"), cex = .5)



#Ridge regression K-fold
bayes2=mse.bayesian.forecast(X.pred=matrix_X,y.target=vector_y,
t.start=170,t.end=70,Alpha=0)

print(bayes2$mse/ar_4$mse)



zoo2=zoo(bayes2$predicted.y,order.by = date)

plot.zoo(cbind(zoo1, zoo2),
         plot.type = "single",
         col = c("blue", "brown4"),main='Ridge',ylab='GDP Growth',xlab='')
```

```
legend("topright", inset=c(0,0), y.intersp = 1,
legend = c("Observed","Predicted-Ridge"),  lty = 1, bty = "n",
col = c("blue", "brown4"), cex = .5)


#Ridge regression own ridge

lambdas=seq((from=(2.970148e-06))*10,to=(2.970148e-06)*1000,length.out=200)

mres=end_of_sample_cv_ridge.b(X=matrix_X[1:170,],
y=vector_y[1:170],eval.obs=30,v_param=lambdas)

plot(mres)

print(which.min(mres))
print(mres[which.min(mres)])



bayes1=ridge.forecast.b(X.pred=matrix_X,y.target=vector_y,
t.start=170,t.end=70,v=2.970148e-06  )

print(bayes1$mse/ar_4$mse)



zoo2=zoo(bayes1$predicted.y,order.by = date)

plot.zoo(cbind(zoo1, zoo2),
         plot.type = "single",
         col = c("blue", "brown1"),main='Ridge',ylab='GDP Growth',xlab='')

legend("topright", inset=c(0,0), y.intersp = 1,
 legend = c("Observed","Predicted_Ridge"),  lty = 1, bty = "n",
col = c("blue", "brown1"), cex = .5)




#ridge vs ridge


bayes1z=zoo(bayes1$predicted.y,order.by = date)
bayes2z=zoo(bayes2$predicted.y,order.by = date)
plot.zoo(cbind(zoo1, bayes1z,bayes2z),
         plot.type = "single",
         col = c("blue", "brown1",'green'),main='CV Comparison'
,ylab='GDP Growth',xlab='')

legend("topright", inset=c(0,0), y.intersp = 1,
legend = c("Observed","End-Of-Sample CV","K-Fold CV"),  lty = 1, bty = "n",
col = c("blue", "brown1",'green'), cex = .5)




#with lars selection
#ols with top 5

matrix_x5=lars.selection.b(matrix_X,vector_y,lambda = 0.002405036,n.vars = 5,
type1='lar')
```

```
df1=data.frame(vector_y,matrix_x5)


ols1=roll.forecast.ols(X.pred=matrix_x5,y.target=vector_y,df=df1,
t.start=170,t.end=70)

print(ols1$mse/ar_4$mse)




zoo2=ols1$predicted.y

plot.zoo(cbind(zoo1, zoo2),
         plot.type = "single",
         col = c("blue", "azure4"),main='Ridge',ylab='GDP Growth',xlab='')

legend("topright", inset=c(0,0), y.intersp = 1,
legend = c("Observed","Predicted-LASSO(5)"),  lty = 1, bty = "n",
col = c("blue", "azure4"), cex = .5)









#lars pcr 50

matrix_x50=lars.selection.b(matrix_X,vector_y,lambda = 0.0002093577,
n.vars = 50,type1='lar')


prcr50=pcr.rolling(X.pred=data.frame(matrix_x50),y.target=vector_y,t.start =
170,t.end=70,n.pcs=3)

print(prcr50$mse/ar_4$mse)




#Plot with all methods
Observed=xts(vector_y[171:240],order.by = date)
AR=xts(ar_4$predicted.y,order.by = date)
PLSR=xts(plsr$predicted.y,order.by = date)
Ridge=xts(bayes1$predicted.y,order.by = date)

SPCR=xts(spcr$predicted.y,order.by = date)
PCR=xts(prcr$predicted.y,order.by = date)
PRF=xts(prf1$predicted.y,order.by = date)

plot.xts(cbind(Observed, AR,PLSR,Ridge,SPCR,PCR,PRF),
         plot.type = "single",
         col = c("blue",
"red","darkolivegreen","brown1","darkorange3","darkorange","chartreuse4"),
main='',ylab='GDP Growth',xlab='',legend.loc
='bottomleft')
```

```
legend("topright", inset=c(0,0), y.intersp = 1, legend =
c("Observed","AR(4),PLSR,Ridge,SPCR,PCR,3PRF"),  lty = 1, bty = "n",
col = c("blue", "red","darkolivegreen","brown1","darkorange3","darkorange",
"chartreuse4"), cex = .5)
```

The following R code was used to run the Monte Carlo Simulations (based on Kelly-Pruit).

This code generates the normal factor structure dataset with serial correlation in the errors. Entering another value for errors_noise can result in the weak factor dataset.

```
library(MASS)


library(pls)

#Monte Carlo


#PCR,PLS,3PRF




#Generate Factors and predictors


T=100
N=100
r=3

T2=T+100

rmse_pcr=c()
rmse_plsr=c()
rmse_prf=c()
rmse_ridge=c()
rmse_plsr_cv=c()

median_rsquared_vector=c()
```

```
y_rsquare_vector=c()


for (mc in 1:100){
  #Factor Matrix

  F=matrix(ncol=r,nrow=T2)



  #Choose the serial correlation in the factors



  ar_coef=c(0.9,0.9,0.3)



  #Set the first observation of the factors

  F[1,]=matrix(runif(n=r),nrow=1,ncol=r)



  #Generate the factors, each follows an AR(1)
  #Each factor has a different variance
  fvar=c(2.75,2.25,1)

  for (j in 1:r){
    for (t in 2:T2){
      F[t,j]=F[(t-1),j]*ar_coef[j]+rnorm(n=1,0,4*sqrt(fvar[j]))
    }
  }


  #Generate the Loadings
  #The factors will not be equal in their importance.




  loadings=matrix(nrow=N,ncol=r)



  for (c in 1:r){
    loadings[,c]=rnorm(n=N,0,1)
  }

  #Create the X matrix
  t_X=matrix(nrow=N,ncol=T2)



  #Create Serially Correlated errors for the X=LF_t+e_t
  #Assume e_t=r_e*e_t-1+u_t
  #u_t is N(0,1)

  #Choose the ar coefficient of the errors
  error_coef=0.9
```

```
  errors_X=matrix(nrow=T2,ncol=N)
  #Choose strength of the errors in order to select strong or weak factor
model

  errors_noise=14

  #Initialize the first row of errors
  errors_X[1,]=matrix(rnorm(n=N,0,1),ncol=N,nrow=1)


  for (t in 2:T2){
    errors_X[t,]=errors_X[t-
1]*error_coef+matrix(rnorm(n=N,0,1),ncol=N,nrow=1)*errors_noise
  }

  #Transpose the errors matrix
  t_errors_X=t(errors_X)


  for (i in 1:N){
    t_X[i,]=loadings[i,]%*%t(F)+t_errors_X[i,]
  }


  #We transpose t_X  in order to have the observations as rows and
  #the columns as variables

  X=t(t_X)


  #Find if we have a strong or a weak factor model based on the R squared
  #This is done by regressing each x_i on the factors obtaining the R squared
  #and the choosing the median R squared

  r.squares=c()
  for (i in 1:N){
    lm.fit=lm(X[,i]~F)
    r.squares[i]=summary(lm.fit)$r.squared
  }

  median_rsquared_vector[mc]=median(r.squares)


  print(median(r.squares))



  #Create the target
  #The target will be driven by the lesser factors

  y=matrix(ncol=1,nrow=T2)


  #Choose how much noise will there be in the target
  noise=4.2

  for (t in 1:T2){
    y[t,]=F[t,3]+rnorm(n=1,0,1)*noise
  }

  #Determine the R squared (signal to noise ratio)
```

```
  y_rsquare_vector[mc]=summary(lm(y~F[,c(3)]))$r.squared

  print(summary(lm(y~F[,c(3)]))$r.squared)



  #We apply one-step ahead forecast for 100 periods and obtain the MSE


  mse_plsr=roll.forecast.mse(X.pred=X, y.target=y , fun=plsr,n.factors=r)


  mse_pcr=roll.forecast.mse.pcr(X.pred=X, y.target=y ,n.factors=r)



    #Plsr end of sample cross-validation. The number of factors to be used is
chosen between one up to ten.


  opt=end_of_sample_cv_plsr(X=X[1:T,],y=y[1:T],eval.obs = 20,factor_n =
c(1:10))



  mse_plsr_cv=roll.forecast.mse(X.pred=X,
y.target=y ,fun=plsr,n.factors=which.min(opt))




  #3PRF

  mse.prf=roll.forecast.mse.prf(X.pred=X, y.target=y ,n.proxies=2)


  naive.mse=mean((y[(T+1):T2]-mean(y[1:T]))^2)



  vpms=lambdapath(X=X[1:T,],y=y[1:T],K=800,epsilon=10^(-
9),mult.max.lambda=10^(7))

  lopt=end_of_sample_cv_ridge(X[1:T,],y[1:T],eval.obs=20,v_param=vpms)

  mse.ridge=ridge.forecast.b(X.pred=X,y.target=y,t.start=T,t.end=100,v=lopt)

  rmse_pcr[mc]=mse_pcr/naive.mse

  rmse_plsr[mc]=mse_plsr/naive.mse
  rmse_prf[mc]=mse.prf/naive.mse
  rmse_ridge[mc]=mse.ridge$mse/naive.mse
  rmse_plsr_cv[mc]=mse_plsr_cv/naive.mse

  print(mc)
```

```
}


mean(rmse_prf)
mean(rmse_pcr)
mean(rmse_plsr)
mean(rmse_plsr_cv)

mean(rmse_ridge)
mean(y_rsquare_vector)
mean(median_rsquared_vector)
```

Non-pervasive factors.
The following code creates the non-pervasive factor dataset.

```
library(MASS)


library(pls)



#Monte Carlo


#PCR,PLS,3PRF



#Generate Factors and predictors


T=100
N=100
r=3

T2=T+100

rmse_pcr=c()
rmse_plsr=c()
rmse_prf=c()
rmse_plsr_cv=c()

rmse_ridge=c()

median_rsquared_vector=c()
y_rsquare_vector=c()


for (mc in 1:100){
  #Factor Matrix

  F=matrix(ncol=r,nrow=T2)
```

```
  #Choose the serial correlation in the factors


  ar_coef=c(0.9,0.9,0.3)



  #Set the first observation of the factors

  F[1,]=matrix(runif(n=r),nrow=1,ncol=r)



  #Generate the factors, each follows an AR(1)
  #Each factor has a different variance
  fvar=c(2.75,2.25,1)

  for (j in 1:r){
    for (t in 2:T2){
      F[t,j]=F[(t-1),j]*ar_coef[j]+rnorm(n=1,0,4*sqrt(fvar[j]))
    }
  }


  #Generate the Loadings

  loadings=matrix(nrow=N,ncol=r)



  for (c in 1:r){
    loadings[,c]=rnorm(n=N,0,1)
  }

  #Non pervasive factor structure, that is half of the
  #predictors have a loading of zero on the relevant factor

  #Choose the predictors that will have zero loading on the relevant factor
  zero.load=c(sample(c(1:N),size=N/2,prob=c(rep(1/N,N)),replace = FALSE))

  loadings[zero.load,3]=0


  #Create the X matrix
  t_X=matrix(nrow=N,ncol=T2)



  #Create Serially Correlated errors for the X=LF_t+e_t
  #Assume e_t=r_e*e_t-1+u_t
  #u_t is N(0,1)

  #Choose the ar coefficient of the errors
  error_coef=0.9

  errors_X=matrix(nrow=T2,ncol=N)

  #Choose strength of the errors in order to select strong or weak factor
model
```

```
  errors_noise=17

  #Initialize the first row of errors
  errors_X[1,]=matrix(rnorm(n=N,0,1),ncol=N,nrow=1)


  for (t in 2:T2){
    errors_X[t,]=errors_X[t-
1]*error_coef+matrix(rnorm(n=N,0,1),ncol=N,nrow=1)*errors_noise
  }

  #Transpose the errors matrix
  t_errors_X=t(errors_X)


  for (i in 1:N){
    t_X[i,]=loadings[i,]%*%t(F)+t_errors_X[i,]
  }


  #We transpose t_X  in order to have the observations as rows and
  #the columns as variables

  X=t(t_X)


  #Find if we have a strong or a weak factor model based on the R squared
  #This is done by regressing each x_i on the factors obtaining the R squared
  #and the choosing the median R squared
  r.squares=c()
  for (i in 1:N){
    lm.fit=lm(X[,i]~F)
    r.squares[i]=summary(lm.fit)$r.squared
  }

  median_rsquared_vector[mc]=median(r.squares)



  #Create the target
  #The target will be driven by the lesser factors

  y=matrix(ncol=1,nrow=T2)


  #Choose how much noise will there be in the target

  noise=3.5

  for (t in 1:T2){
    y[t,]=F[t,3]+rnorm(n=1,0,1)*noise
  }


  #Determine the R squared (signal to noise ratio)

  y_rsquare_vector[mc]=summary(lm(y~F[,c(3)]))$r.squared
```

```
  #We apply one-step ahead forecast for 100 periods and obtain the MSE

  mse_plsr=roll.forecast.mse(X.pred=X, y.target=y , fun=plsr,n.factors=r)

  mse_pcr=roll.forecast.mse.pcr(X.pred=X, y.target=y ,n.factors=r)

    #Plsr end of sample cross-validation. The number of factors to be used is
chosen between one up to ten.

  opt=end_of_sample_cv_plsr(X=X[1:T,],y=y[1:T],eval.obs = 20,factor_n =
c(1:10))



  mse_plsr_cv=roll.forecast.mse(X.pred=X,
y.target=y ,fun=plsr,n.factors=which.min(opt))



  #3PRF

  mse.prf=roll.forecast.mse.prf(X.pred=X, y.target=y ,n.proxies=2)

  naive.mse=mean((y[(T+1):T2]-mean(y[1:T]))^2)



  vpms=lambdapath(X=X[1:T,],y=y[1:T],K=300,epsilon=10^(-
8),mult.max.lambda=10^8)

  mres=end_of_sample_cv_ridge.b(X=X[1:T,],y=y[1:T],eval.obs=20,v_param=vpms)
  lopt=vpms[which.min(mres)]

  plot(mres)

  ridge_mse=ridge.forecast.b(X=X,y=y,t.start=T,t.end=100,v=lopt)



  print(ridge_mse$mse/naive.mse)
  print(mse.prf/naive.mse)
  print(mse_pcr/naive.mse)
  print(mse_plsr/naive.mse)

  rmse_plsr_cv[mc]=mse_plsr_cv/naive.mse

  rmse_pcr[mc]=mse_pcr/naive.mse

  rmse_plsr[mc]=mse_plsr/naive.mse
  rmse_prf[mc]=mse.prf/naive.mse
```

```
  rmse_ridge[mc]=ridge_mse$mse/naive.mse

  print(mc)
}



mean(rmse_prf)
mean(rmse_pcr)
mean(rmse_plsr)
mean(rmse_plsr_cv)

mean(rmse_ridge)
mean(y_rsquare_vector)
mean(median_rsquared_vector)
```

The end-of-sample cross-validation algorithm for Partial Least Squares.

```
#End-of-sample cross validation algorithm for partial least squares
#regression
#X is the argument for the matrix of predictors, y the regressed variable
#eval.obs is the number of observations at the end of the
#sample that shall be taken into account when computing the MSE's
#algorithm
#factor_n takes a vector of integers. The algorithm produces the
#MSE of one-step ahead forecasts for PLS regression over each of the
#number of factors included in the factor_n vector. The
#The optimal number of factors can be chosen with which.min(opt)!!!


end_of_sample_cv_plsr=function(X,y,eval.obs,factor_n){
  T=dim(X)[1]

  mres=c()
  for (i in 1:length(factor_n)){
    v1=factor_n[i]




    j=0
    rolling=c()
    for (t in (T-(eval.obs+1)):(T-1)){
      j=j+1
      pr1=plsr(y[1:t]~X[1:t,],ncomp=v1)
      pred=predict(pr1,newdata=X[t:(t+1),],ncomp=v1)
      rolling[j]=pred[length(pred)]
    }

    mres[i]=mean((rolling-y[(T-eval.obs):(T)])^2)

  }

  return(mres)
```

```
}
```

Recursive Forecast Functions used in the Monte-Carlo Simulations

```
#Functions for Monte Carlo

#Recursive forecast

roll.forecast.mse=function(X.pred,y.target,fun,n.factors){
  rolling=c()
  for (t in 0:99){
    pr1=fun(y.target[1:(T+t)]~X.pred[1:(T+t),],ncomp=n.factors)
    pred=predict(pr1,newdata=X.pred[(T+t):(T+t+1),],ncomp=n.factors)
    rolling[t+1]=pred[length(pred)]
  }
  return(mean((rolling-y.target[(T+1):T2])^2))
}



roll.forecast.mse.pcr=function(X.pred,y.target,n.factors){
  rolling=c()
  for (t in 0:99){
    pr1=pcr(y.target[1:(T+t)]~X.pred[1:(T+t),],ncomp=n.factors)
    pred=predict(pr1,newdata=X.pred[(T+t):(T+t+1),],ncomp=n.factors)
    rolling[t+1]=pred[length(pred)]
  }
  return(mean((rolling-y.target[(T+1):T2])^2))
}

roll.forecast.mse.prf=function(X.pred,y.target,n.proxies=2){
  rolling=c()
  for (t in 0:99){
    threeprf=TPRF(X.pred[1:(T+t),],y.target[1:(T+t)],L=n.proxies)
    pred=predict.t3prf(threeprf,newdata=X.pred[(T+t):(T+t+1),])
    rolling[t+1]=pred[length(pred)]
  }

  return(mean((rolling-y.target[(T+1):T2])^2))
}
```

R packages

Hadley Wickham, Romain François, Lionel Henry and Kirill Müller (2020). dplyr: A Grammar of Data Manipulation. R package version 1.0.2. https://CRAN.R-project.org/package=dplyr

Hyndman R, Athanasopoulos G, Bergmeir C, Caceres G, Chhay L, O'Hara-Wild M, Petropoulos F, Razbash S, Wang E, Yasmeen F (2020). _forecast: Forecasting functions for time series and linear models_. R package version 8.13, <URL: https://pkg.robjhyndman.com/forecast/>.

Jeffrey A. Ryan and Joshua M. Ulrich (2020). xts: eXtensible Time Series. R package version 0.12.1. https://CRAN.R-project.org/package=xts

Garrett Grolemund, Hadley Wickham (2011). Dates and Times Made Easy with lubridate. Journal of Statistical Software, 40(3), 1-25. URL https://www.jstatsoft.org/v40/i03/.

Achim Zeileis and Gabor Grothendieck (2005). zoo: S3 Infrastructure for Regular and Irregular Time Series. Journal of Statistical Software, 14(6), 1-27. doi:10.18637/jss.v014.i06

Bjørn-Helge Mevik, Ron Wehrens and Kristian Hovde Liland (2020). pls: Partial Least Squares and Principal Component Regression. R package version 2.7-3. https://CRAN.R-project.org/package=pls

Hadley Wickham and Jennifer Bryan (2019). readxl: Read Excel Files. R package version 1.3.1. https://CRAN.R-project.org/package=readxl

Daiane Marcolino de Mattos, Pedro Costa Ferreira, Serge de Valk and Guilherme Branco Gomes (2019). nowcasting: Predicting Economic Variables using Dynamic Factor Models. R package version 1.1.4. https://CRAN.R-project.org/package=nowcasting

Trevor Hastie and Brad Efron (2013). lars: Least Angle Regression, Lasso and Forward Stagewise. R package version 1.2. https://CRAN.R-project.org/package=lars

Alboukadel Kassambara and Fabian Mundt (2020). factoextra: Extract and Visualize the Results of Multivariate Data Analyses. R package version 1.0.7. https://CRAN.R-project.org/package=factoextra

Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0

Bibliography

James H. Stock and Mark W. Watson. Forecasting Using Principal Components from a Large Number of Predictors. *Journal of the American Statistical Association , Dec., 2002, Vol. 97, No. 460 (Dec., 2002), pp. 1167-1179*.

Jean Boivin, Serena Ng. *Are more data always better for factor analysis?* Journal of Econometrics 132 (2006) 169–194.

Jushan Bai and Serena Ng. Determining the Number of Factors in Approximate Factor Models. *Econometrica, Vol. 70, No. 1 (Jan., 2002), pp. 191-221*.

Jushan Bai and Serena Ng. Determining the Number of Primitive Shocks in Factor Models. Journal of Econometrics 146 (2008) 304-317.

Jushan Bai and Serena Ng. Forecasting economic time series using targeted predictors. Journal of Business & Economic Statistics , Jan. 2007, Vol. 25, No. 1

Bryan Kelly, Seth Pruitt The three-pass regression filter: A new approach to forecasting using many predictors. *Journal of Econometrics 186 (2015) 294–316*.

Pierre Guerin, Danilo Leiva and Leon Massimiliano Marcellino Markov-Switching Three-Pass Regression Filter. Journal of Business and Economic Statistics · July 2018

Jan J.J. Groen, George Kapetanios*, Revisiting useful approaches to data-rich macroeconomic forecasting*. Computational Statistics and Data Analysis 100 (2016) 221–239.

Christine De Mol, Domenico Giannone, Lucrezia Reichlin . *Forecasting using a large number of predictors, is Bayesian regression a valid alternative to principal components?* ECB, Working Paper Series, No 700, December 2006.

Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani. *LEAST ANGLE REGRESSION.* The Annals of Statistics 2004, Vol. 32, No. 2, 407–499

J.Durbin and S.J Koopman. Time Series Analysis by State Space Methods, Second Edition, 2012.

Catherine Doz, Domenico Giannone, Lucrezia Reichlin . A two-step estimator for large approximate dynamic factor models based on Kalman filtering. Journal of Econometrics 164 (2011) 188–205

Domenico Giannone, Lucrezia Reichlin and Luca Sala. Monetary Policy in Real Time. NBER Macroeconomics Annual , 2004, Vol. 19 (2004), pp. 161-200