Assignment CompMacro

2024-04-03

Estimation & Forecasts

We first define a function to calculate the betas of the estimation and the forecast. We include the option to choose a different end date but for now we will set it equal to the latest date in the dataset.

```python
import pandas as pd
import numpy as np
from numpy.linalg import solve, inv

# importing dataframe
df = pd.read_csv('~/Downloads/current.csv')
df_cleaned = df.drop(index=0)
df_cleaned.reset_index(drop=True, inplace=True)
transformation_codes = df.iloc[0, 1:].to_frame().reset_index()
transformation_codes.columns = ['Series', 'Transformation_Code']

def apply_transformation(series, code):
    if code == 1:
        # No transformation
        return series
    elif code == 2:
        # First difference
        return series.diff()
    elif code == 3:
        # Second difference
        return series.diff().diff()
    elif code == 4:
        # Log
        return np.log(series)
    elif code == 5:
        # First difference of log
        return np.log(series).diff()
    elif code == 6:
        # Second difference of log
        return np.log(series).diff().diff()
    elif code == 7:
        # Delta (x_t/x_{t-1} - 1)
        return series.pct_change()
    else:
        raise ValueError("Invalid transformation code")

for series_name, code in transformation_codes.values:
    df_cleaned[series_name] = apply_transformation(df_cleaned[series_name].astype(float), float(code))
```

```python
df_cleaned['sasdate'] = pd.to_datetime(df_cleaned['sasdate']) # convert sasdate to datetime

#define function to obtain betas
def calculate_betas(h=1, p=4,
                    target = 'INDPRO',
                    CoVars = ['CPIAUCSL', 'TB3MS'], end_date = '1/1/2024'):

    new_data = df_cleaned[df_cleaned['sasdate'] <= pd.Timestamp(end_date)] # subset data
    Reg_data = pd.DataFrame() # initiate empty dataset to store all variables for estimation
    Reg_data['Y_lead'] = new_data[target].shift(-h) #lead the target variable
    #lagging Target Variable:
    for i in range(0, p+1):
        Reg_data[f'{target}_lag{i}'] = new_data[target].shift(i)

    #lagging Covariates:
    for col in CoVars:
        for i in range(0, p+1):
            Reg_data[f'{col}_lag{i}'] = new_data[col].shift(i)

    #Drop NA values:
    Reg_data = Reg_data.dropna()
    Y_reg = Reg_data['Y_lead'].values # target vector
    X_reg = Reg_data.iloc[:,1:] # Covariate matrix
    X_reg.insert(0, 'constant', 1) # insert constant
    X_t = X_reg.iloc[-1]
    X_reg = X_reg.values

    beta_OLS = inv(X_reg.T @ X_reg) @ X_reg.T @ Y_reg #OLS formula
    forecast = X_t @ beta_OLS
    return [beta_OLS, forecast]
```

Industrial Production

```python
#Estimating betas for INDPRO and forecasting:
print('Betas from regression of y = INDPRO and x = CPIAUCSL, TB3MS:', calculate_betas()[0])
```

```
## Betas from regression of y = INDPRO and x = CPIAUCSL, TB3MS: [ 1.40634199e-03  2.69641786e-01 -7.9728
##   3.46340754e-02  3.67477784e-03  4.39766060e-01  7.12251108e-02
##   2.19555791e-01  1.30306127e-01  1.71913835e-01  3.70673014e-03
##  -2.26245763e-04 -2.36966543e-04 -1.03719702e-03 -4.98357633e-05]
```

```python
print('Forecast from regression of y = INDPRO and x = CPIAUCSL, TB3MS:', calculate_betas()[1]*100)
```

```
## Forecast from regression of y = INDPRO and x = CPIAUCSL, TB3MS: 0.08233253321125814
```

Inflation

```python
# Estimating betas for CPIAUCSL and forecasting:
cpi_reg =  calculate_betas(target='CPIAUCSL', CoVars=['INDPRO', 'TB3MS'])
print('Betas from regression of y = CPIAUCSL and x = INDPRO, TB3MS:', cpi_reg[0])
```

```
## Betas from regression of y = CPIAUCSL and x = INDPRO, TB3MS: [ 1.01631345e-05 -4.93685519e-01 -4.616
##  -2.31593172e-01 -1.53817794e-01  3.36970552e-03  9.59528066e-04
##   6.26667047e-03 -4.88242549e-03 -1.00006194e-02  5.14384638e-04
##   6.12957118e-04 -1.45600590e-04  3.60300939e-04 -1.78894846e-04]
```

```
print('Forecast from regression of y = CPIAUCSL and x = INDPRO, TB3MS:', cpi_reg[1]*100)
```

```
## Forecast from regression of y = CPIAUCSL and x = INDPRO, TB3MS: 0.015249769043965529
```

Interest Rates

```
# Estimating betas for TB3ms and forecasting:
tb_reg =  calculate_betas(target='TB3MS', CoVars=['INDPRO', 'CPIAUCSL'])
print('Betas from regression of y = TB3MS and x = INDPRO, CPIAUCSL:', tb_reg[0])
```

```
## Betas from regression of y = TB3MS and x = INDPRO, CPIAUCSL: [-1.12284984e-02  4.02373954e-01 -2.422
##  -2.91692041e-02  6.00444182e-02  3.72626222e+00  2.23156864e+00
##   8.68592913e-01 -5.62720396e-01  7.74591697e-01  1.02336079e+01
##  -9.88078877e+00 -2.27822818e+00 -2.11266077e+01 -1.36406747e+00]
```

```
print('Forecast from regression of y = TB3MS and x = INDPRO, CPIAUCSL:', tb_reg[1]*100)
```

```
## Forecast from regression of y = TB3MS and x = INDPRO, CPIAUCSL: 2.903456930846914
```

Estimation Evaluation

We define a function to calculate the Error and a second function to plot it.

```
## Estimation evaluation:
end_date = '12/1/1999'
H = [1,4,8]
p = 4
def MSFE(H, p, target, CoVars, end_date):
    ## Get the actual values of target at different steps ahead
    Y_actual = []
    Y_hat = []
    for h in H:
        os = pd.Timestamp(end_date) + pd.DateOffset(months=h)
        Y_actual.append(df_cleaned[df_cleaned['sasdate'] == os][target]*100)
        Y_hat = calculate_betas(h, p, target, CoVars, end_date)[1]*100
    return np.array(Y_actual) - np.array(Y_hat)


print(MSFE(H, 4, 'INDPRO', ['CPIAUCSL', 'TB3MS'], end_date))
```

```
## [[-0.57499422]
##  [ 0.13796011]
##  [-0.76930542]]
```

```
t0 = pd.Timestamp('12/1/1999')
e = []
T = []
for j in range(0, 10):
    t0 = t0 + pd.DateOffset(months=1)
    print(f'Using data up to {t0}')
    ehat = MSFE(p = 4, H = [1,4,8], end_date = t0, target='INDPRO', CoVars=['CPIAUCSL', 'TB3MS'])
    e.append(ehat.flatten())
    T.append(t0)

## Create a pandas DataFrame from the list
```

```
## Using data up to 2000-01-01 00:00:00
## Using data up to 2000-02-01 00:00:00
## Using data up to 2000-03-01 00:00:00
## Using data up to 2000-04-01 00:00:00
## Using data up to 2000-05-01 00:00:00
## Using data up to 2000-06-01 00:00:00
## Using data up to 2000-07-01 00:00:00
## Using data up to 2000-08-01 00:00:00
## Using data up to 2000-09-01 00:00:00
## Using data up to 2000-10-01 00:00:00
```

```
edf = pd.DataFrame(e)
## Calculate the RMSFE, that is, the square root of the MSFE
print(np.sqrt(edf.apply(np.square).mean()))
```

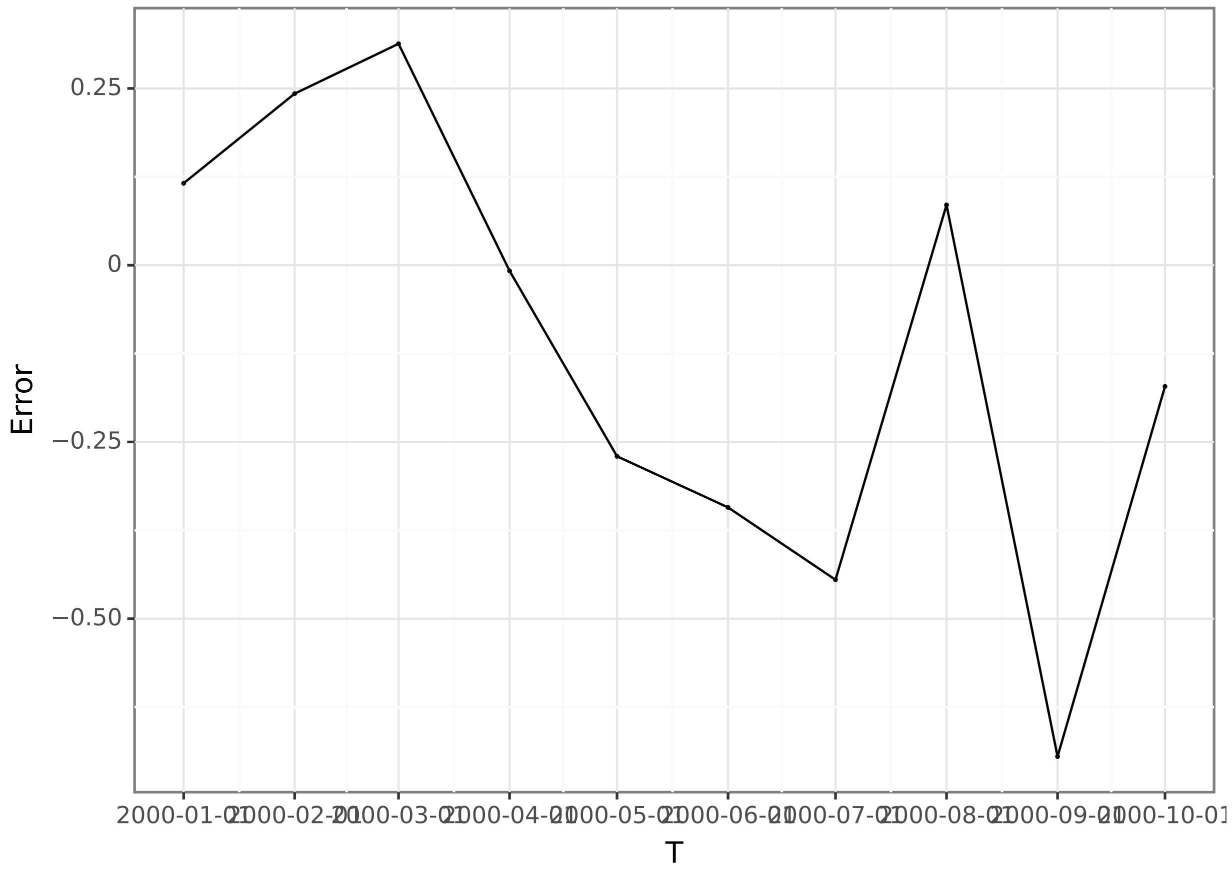
```
## 0    0.328426
## 1    0.520060
## 2    0.638129
## dtype: float64
```

```
from plotnine import *


def plotFE(H, p, target, CoVars, t0, range_end):
    t0 = pd.Timestamp(t0)
    e = []
    T = []
    for j in range(0, range_end):
        t0 = t0 + pd.DateOffset(months=1)
        ehat = MSFE(p=p, H=H,target=target, CoVars=CoVars, end_date=t0,)
        e.append(ehat.flatten())
        T.append(t0)

    edf = pd.DataFrame(e)
    print('The RMSE are', np.sqrt(edf.apply(np.square).mean()))

    plotlist = []
    for i in range(0, len(H)):
        plotdata = {'T' : T, 'Error' : edf.iloc[:,i]}
        plotdata = pd.DataFrame(plotdata)
        plotMSE = ggplot(plotdata) + aes(x = 'T', y = 'Error') + geom_line() + geom_point(size = 0.01) 
        plotlist.append(plotMSE)
    return plotlist
```

For example, we can now plot the ten errors starting from 12/1/1999 and using $h = 1$:

```
print(plotFE(H, p, 'INDPRO', ['CPIAUCSL', 'TB3MS'], t0 = '12/1/1999', range_end=10)[0])
```

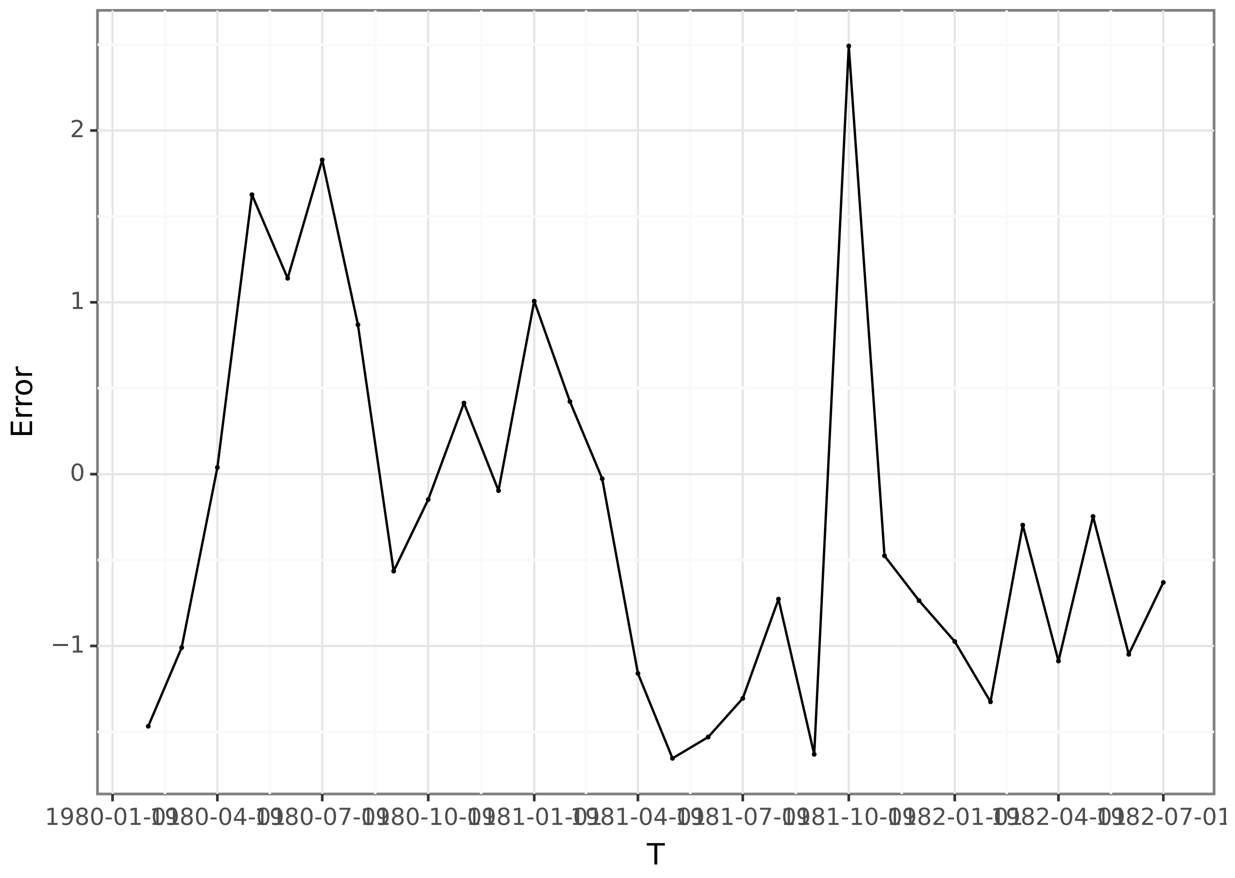
```
## The RMSE are 0    0.328426
## 1    0.520060
## 2    0.638129
## dtype: float64
```



Or we can plot 30 errors starting frmo 1/1/1980 and using $h = 4$

```
print(plotFE(H, p, 'INDPRO', ['CPIAUCSL', 'TB3MS'], t0 = '1/1/1980', range_end=30)[1])
```

```
## The RMSE are 0    1.176550
## 1    1.107000
## 2    1.120199
## dtype: float64
```



Error
2
1
0
−1
1980-01-01
1980-04-01
1980-07-01
1980-10-01
1981-01-01
1981-04-01
1981-07-01
1981-10-01
1982-01-01
1982-04-01
1982-07-01
T