Computational Tools for Macroeconometrics

Assignment 1

Introduction

This assignment introduces students to practical and theoretical aspects of macroeconometrics, focusing on forecasting using the FRED-MD dataset. Students will learn to handle macroeconomic data, perform necessary transformations, apply univariate models to predict key economic indicators and to evaluate these forecasts.

The FRED-MD dataset

The FRED-MD dataset is a comprehensive monthly database for macroeconomic research compiled by the Federal Reserve Bank of St. Louis. It features a wide array of economic indicators. The list of economic indicators can be obtained from the paper accompanying the data pdf.

The data can be downloaded here. The page contains all the different vintages of the data.

Let us start to download the `current.csv` file:

```
import pandas as pd

# Load the dataset
df = pd.read_csv('~/Downloads/current.csv')

# Clean the DataFrame by removing the row with transformation codes
df_cleaned = df.drop(index=0)
df_cleaned.reset_index(drop=True, inplace=True)
df_cleaned
```

	sasdate	RPI	W875RX1	DPCERA3M086SBEA	CMRMTSPLx	RETAILx	INDPRO
0	1/1/1959	2583.560	2426.0	15.188	2.766768e+05	18235.77392	21.9665
1	2/1/1959	2593.596	2434.8	15.346	2.787140e+05	18369.56308	22.3966
2	3/1/1959	2610.396	2452.7	15.491	2.777753e+05	18523.05762	22.7193
3	4/1/1959	2627.446	2470.0	15.435	2.833627e+05	18534.46600	23.2032
4	5/1/1959	2642.720	2486.4	15.622	2.853072e+05	18679.66354	23.5528
...	...	...	...	...	...	...	...
776	9/1/2023	19111.748	15741.9	116.594	1.507530e+06	705304.00000	103.2096
777	10/1/2023	19145.402	15784.6	116.663	1.505477e+06	703528.00000	102.3722
778	11/1/2023	19213.108	15859.9	117.127	1.514733e+06	703336.00000	102.6710
779	12/1/2023	19251.946	15899.0	117.773	1.530296e+06	706180.00000	102.6715
780	1/1/2024	19377.558	15948.8	117.639	NaN	700291.00000	102.5739

```
# Extract transformation codes
transformation_codes = df.iloc[0, 1:].to_frame().reset_index()
transformation_codes.columns = ['Series', 'Transformation_Code']
```

The transformation codes map variables to transformations we must apply to the data to render them (approximately) stationary. In the first column, the data frame `transformation_codes` has the variable's name and its transformation (second column). There are six possible type of transformation:

- `transformation_code=1`: no trasformation
- `transformation_code=2`: Δx_t
- `transformation_code=3`: $\Delta^2 x_t$
- `transformation_code=4`: $log(x_t)$
- `transformation_code=5`: $\Delta log(x_t)$
- `transformation_code=6`: $\Delta^2 log(x_t)$
- `transformation_code=7`: $\Delta(x_t/x_{t-1} - 1)$

We can apply these trasformations using the following code:

```
import numpy as np

# Function to apply transformations based on the 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")

# Applying the transformations to each column in df_cleaned based on transformation_codes
for series_name, code in transformation_codes.values:
    df_cleaned[series_name] = apply_transformation(df_cleaned[series_name].astype(float), fl

df_cleaned.head()
```

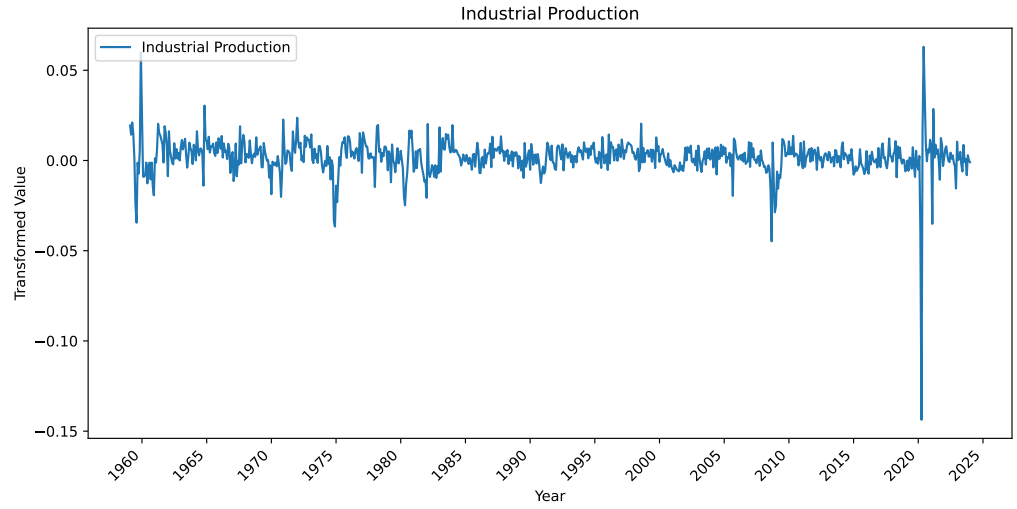
	sasdate	RPI	W875RX1	DPCERA3M086SBEA	CMRMTSPLx	RETAILx	INDPRO	IPFP
0	1/1/1959	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	2/1/1959	0.003877	0.003621	0.010349	0.007336	0.007310	0.019391	0.013
2	3/1/1959	0.006457	0.007325	0.009404	-0.003374	0.008321	0.014306	0.006
3	4/1/1959	0.006510	0.007029	-0.003622	0.019915	0.000616	0.021075	0.014
4	5/1/1959	0.005796	0.006618	0.012043	0.006839	0.007803	0.014955	0.008

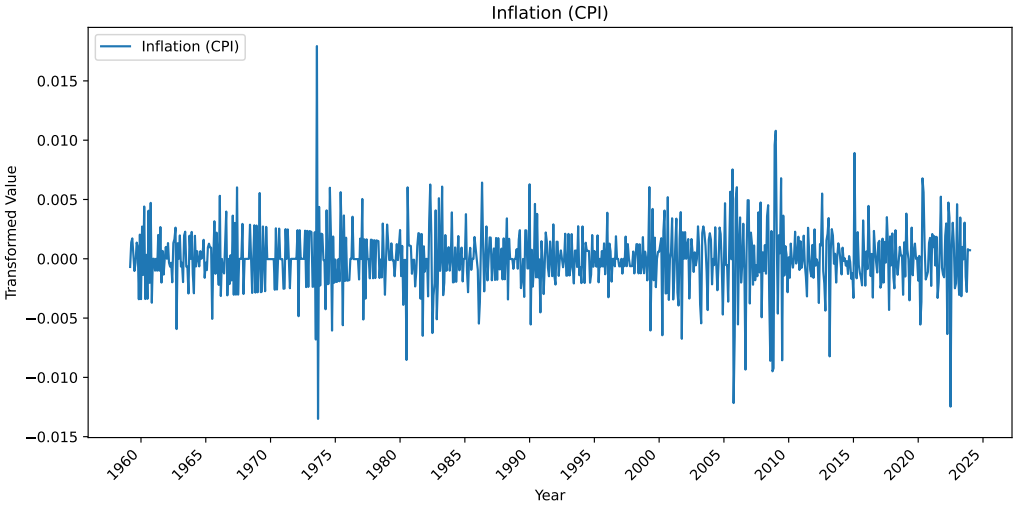
```
series_to_plot = ['INDPRO', 'CPIAUCSL', 'TB3MS']
series_names = ['Industrial Production', 'Inflation (CPI)', 'Federal Funds Rate']
# 'INDPRO'   for Industrial Production,
# 'CPIAUCSL' for Inflation (Consumer Price Index),
# 'TB3MS'    3-month treasury bill.
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
```

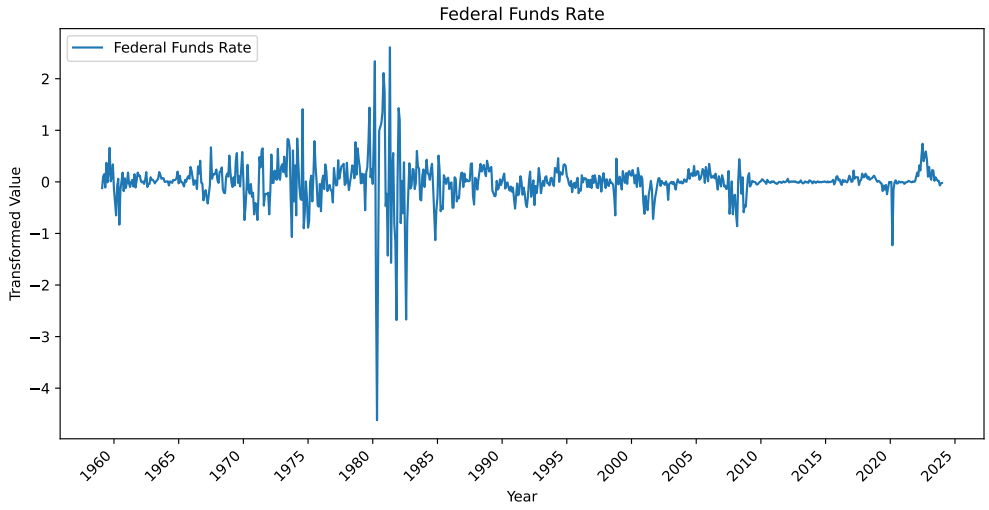
```
# Create a figure and a grid of subplots
fig, axs = plt.subplots(len(series_to_plot), 1, figsize=(10, 15))

# Iterate over the selected series and plot each one
for ax, series_name, plot_title in zip(axs, series_to_plot, series_names):
    if series_name in df_cleaned.columns:
        # Convert 'sasdate' to datetime format for plotting
        dates = pd.to_datetime(df_cleaned['sasdate'], format='%m/%d/%Y')
        ax.plot(dates, df_cleaned[series_name], label=plot_title)
        # Formatting the x-axis to show only every five years
        ax.xaxis.set_major_locator(mdates.YearLocator(base=5))
        ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
        ax.set_title(plot_title)
        ax.set_xlabel('Year')
        ax.set_ylabel('Transformed Value')
        ax.legend(loc='upper left')
        # Improve layout of date labels
        plt.setp(ax.xaxis.get_majorticklabels(), rotation=45, ha='right')
    else:
        ax.set_visible(False)  # Hide plots for which the data is not available

plt.tight_layout()
plt.show()
```


Industrial Production
Industrial Production
Transformed Value
0.05
0.00
−0.05
−0.10
−0.15
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020
2025
Year


Inflation (CPI)
Inflation (CPI)
Transformed Value
0.015
0.010
0.005
0.000
−0.005
−0.010
−0.015
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020
2025
Year


Federal Funds Rate
Federal Funds Rate
Transformed Value
2
1
0
−1
−2
−3
−4
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020
2025
Year

Forecasting in Time Series

Forecasting in time series analysis involves using historical data to predict future values. The objective is to model the conditional expectation of a time series based on past observations.

Direct Forecasts

Direct forecasting involves modeling the target variable directly at the desired forecast horizon. Unlike iterative approaches, which forecast one step ahead and then use those forecasts as inputs for subsequent steps, direct forecasting directly models the relationship between past observations and future value.

ARX Models

Autoregressive Moving Average with Exogenous Inputs (ARMAX) models are a class of univariate time series models that extend ARMA models by incorporating exogenous (independent) variables. These models are formulated as follows:

$$Y_{t+h} = \alpha + \sum_{i=0}^{p} \phi_i Y_{t-i} + \sum_{k=1}^{r} \beta_k X_{tk} + \epsilon_{t+h}$$

- Y_{t+h}: The target variable at time (t).
- X_{tk}: Exogenous variables that influence (Y_t).
- p and r: lags.
- ϕ, θ, and β: Parameters of the model.
- ϵ_{t+h}: Error term.

The following code produce Y_{T+1}, the forecast for industrial production for the first period

```
Y = df_cleaned['INDPRO'].dropna()
X = df_cleaned[['CPIAUCSL', 'FEDFUNDS']].dropna()

h = 1 ## One-step ahead
p = 4
r = 4

Y_target = Y.shift(-h).dropna()
Y_lagged = pd.concat([Y.shift(i) for i in range(p+1)], axis=1).dropna()
X_lagged = pd.concat([X.shift(i) for i in range(r+1)], axis=1).dropna()
common_index = Y_lagged.index.intersection(Y_target.index)
common_index = common_index.intersection(X_lagged.index)
```

```
## This is the last row needed to create the forecast
X_T = np.concatenate([[1], Y_lagged.iloc[-1], X_lagged.iloc[-1]])

## Align the data
Y_target = Y_target.loc[common_index]
Y_lagged = Y_lagged.loc[common_index]
X_lagged = X_lagged.loc[common_index]

X_reg = pd.concat([X_lagged, Y_lagged], axis = 1)

from numpy.linalg import solve

X_reg = pd.concat([X_lagged, Y_lagged], axis=1)
X_reg_np = np.concatenate([np.ones((X_reg.shape[0], 1)), X_reg.values], axis=1)
Y_target_np = Y_target.values

# Solving for the OLS estimator beta: (X'X)^{-1} X'Y
beta_ols = solve(X_reg_np.T @ X_reg_np, X_reg_np.T @ Y_target_np)

## Produce the One step ahead forecast
## % change month-to-month INDPRO
forecast = X_T@beta_ols*100
```

Forecasting Exercise

Apply ARX models to forecast:

- Industrial production
- Inflation
- Interest rates

using the FRED-MD dataset.

Use expanding windows to evaluate the forecasts using a MSFE loss function. The first window ends on January 8.