Youssef Mahmoud 905854027

In [1]:
```
import yfinance as yf
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import datetime
import io
import datetime
import matplotlib.lines as mlines
from fredapi import Fred
import statsmodels.formula.api as smf
import datetime

```

In [2]:
```
df = pd.read_csv("desktop/hw2.csv", parse_dates = True, index_col = 0)
df
```

Out[2]:

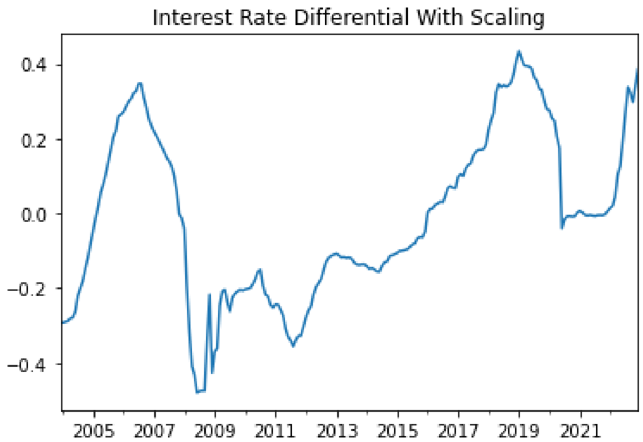
	I_US	I_EU	Inf_US	Inf_EU	Euro
2003-12-01	1.11	2.1590	0.019284	0.021933	1.196501
2004-01-01	1.10	2.1463	0.020352	0.020207	1.258194
2004-02-01	1.06	2.0895	0.020263	0.018343	1.246805
2004-03-01	1.05	2.0706	0.016885	0.016654	1.244803
2004-04-01	1.05	2.0288	0.017401	0.017175	1.236507
...	...	...	...	...	...
2022-08-01	2.50	0.0366	0.084821	0.088662	1.020825
2022-09-01	2.76	0.3947	0.082492	0.091406	1.003905
2022-10-01	3.21	1.0109	0.082224	0.099272	0.982956
2022-11-01	3.85	1.4277	0.077631	0.106206	0.988631
2022-12-01	4.46	1.8252	0.071179	0.100546	1.042535

229 rows × 5 columns

In [3]:
```
def scale(x):
    return (x-x.min())/(x.max()-x.min())

# Apply Scaling
df["ir_diff"] = scale(df["I_US"]) - scale(df["I_EU"])
df.dropna(inplace= True)
```

In [4]:
```
plt.title("Interest Rate Differential With Scaling")
df["ir_diff"].plot()
```

Out[4]: <AxesSubplot:title={'center':'Interest Rate Differential With Scaling'}>



In [5]:
```
k = 0.05
w = 5
z = 0

df["Filter"] = df.ir_diff.ewm(alpha = k, adjust = False).mean()
```

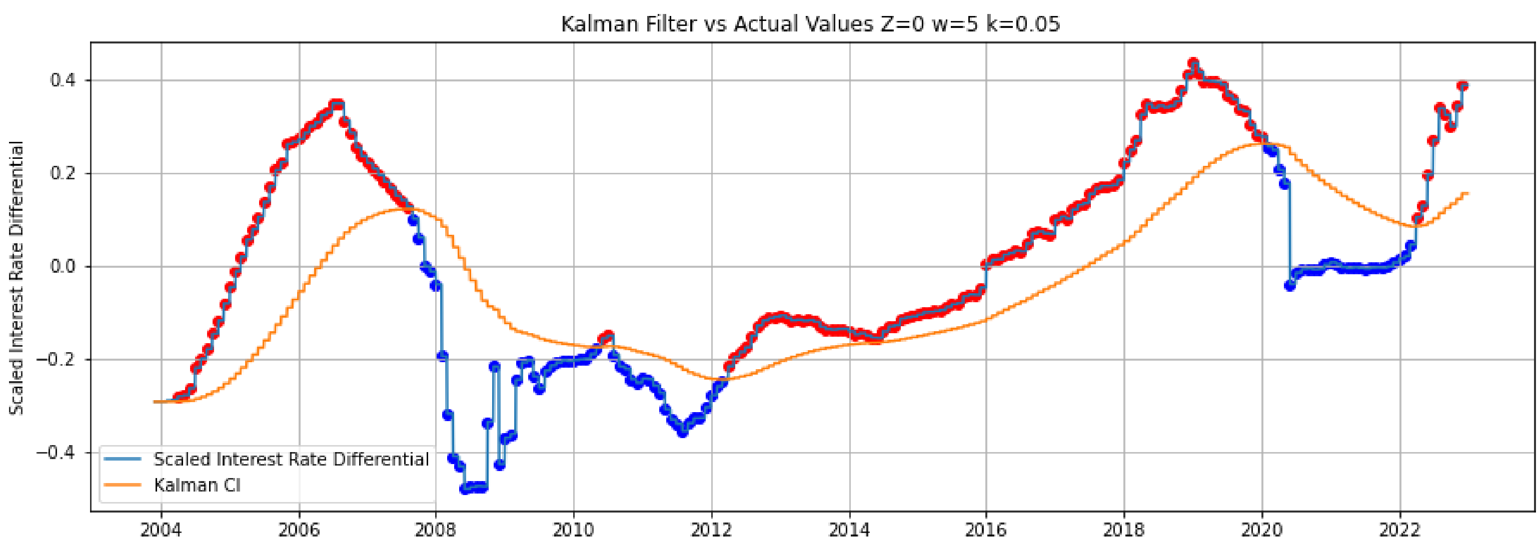
In [6]:
```
df["Filter Error"] = df.ir_diff - df["Filter"]

# compute the rolling standard deviation
df["std"] = df["Filter Error"].rolling(w).std()

# create our confidence intervals or "boundaries of inaction"
# these are scaled by teh number of standard deviations "z"
df["Upper"] = df["Filter"] + z*df["std"]
df["Lower"] = df["Filter"] - z*df["std"]

# Create signal that evaluates whether we are outside the threshold
# then multiply by the direction of the mistake
# (we use economic theory to decide which direction is long or short)
df["test"] = np.where(df["Filter Error"].abs()>z*df["std"], -1, 0)*np.sign(df["Filter Error"])
```

In [ ]:
```

```

In [8]:
```
i = 41

# create a new column that we will populate with our daily position
daily.loc[:, str(i)+"_signal"] = 0

# loop through each day in the dataset
for j in daily.index:
    # If our monthly signal is not 0
    if daily.loc[j, "test"] != 0:
        # Make the next i days equal to the monthly signal
        daily.loc[j:j+datetime.timedelta(i), str(i)+"_signal"] = daily.loc[j, "test"]

# Below is the holding period I use for the CI strategy
i = 171
daily.loc[:, str(i)+"_signal"] = 0
for j in daily.index:
    if daily.loc[j, "test"] != 0:
        daily.loc[j:j+datetime.timedelta(i), str(i)+"_signal"] = daily.loc[j, "test"]

```

In [9]:

```
data2 = daily.dropna()
fig, ax = plt.subplots(figsize = (15, 5))
ax.set_title("Kalman Filter vs Actual Values " + "Z="+str(z) + " w=" + str(w)+ " k=" + str(k))

ax.set_ylabel("Scaled Interest Rate Differential")

# Plot the actual series and the filter
ax.plot(data2["ir_diff"])
ax.plot(data2["Filter"])

# This code block is used to add confidence intervals when z > 0
#ax.fill_between(data2.index, data2.Lower, data2.Upper, color='b', alpha=.2)

# add scatterplots using boolean indexing
# We change the colors and shapes based on the conditions
ax.scatter(data2[data2.test == 1].index, data2[data2.test == 1]["ir_diff"], color = "blue")
ax.scatter(data2[data2.test == -1].index, data2[data2.test == -1]["ir_diff"], color = "red")
ax.legend(["Scaled Interest Rate Differential", "Kalman CI"])

# this code can let us zoom in on certain time periods
#plt.xlim([datetime.date(2022, 1, 1), datetime.date(2023, 1, 1)])
ax.grid()
```



I would short if the price today is low but expected to increse in the future. I would long if price is low today but expected to rise in the future.

In [11]:
```
daily["Euro"]=df['Euro']
```

In [13]:
```
daily["Euro"] = daily["Euro"].ffill()
```

In [14]:
```
start = daily[["Euro", "41_signal"]].index[0]
```

In [15]:
```
df1 = daily[daily.test != 0][["41_signal", "Euro"]].copy()
df1['D'] = df1["41_signal"]
```

In [16]:
```
df1 = df1[:-1].copy()
```

In [17]:
```
df1['s_current'] = daily[daily.index.isin(df1.index)]["Euro"].values
df1['s_future'] = daily[daily.index.isin(df1.index+datetime.timedelta(41))]["Euro"].values

# Get the realized exchange rate
df1['R'] = np.where(df1['s_future'] >= df1['s_current'], 1, -1)
```

In [19]:
```
df1['W'] = (df1['D']-np.mean(df1['D']))*(df1['R']-np.mean(df1['R']))
T_B = np.mean(df1['W'])
T_B
```

Out[19]:
```
-0.0015943877551020410
```

Yes, it did pass the binomial test.

In [20]:
```
dy = df1['W'] - np.mean(df1['W'])
gamma_0 = sum((dy)**2)/len(df1)
gamma_1 = np.mean((dy*dy.shift(-1))[:len(df1)-1])
LRV = gamma_0 + 2*(1-1/2)*gamma_1
```

In [21]:
```
from scipy.stats import norm

statistic = T_B/np.sqrt(LRV/df1.shape[0])
print('Test statistic : ', statistic, ', 5 % critical value : ', round(norm.ppf(0.95),2))
```

```
Test statistic :  -0.02468504071733396 , 5 % critical value :  1.64
```

Based on the t-stat, the Newey-West LRV estimator Null cannot be rejected.

In [22]:
```
df1['W_2'] = df1['D']*(df1['s_future']-df1['s_current'])
T_WB = np.mean(df1['W_2'])
```

In [23]:
```
T_WB
```

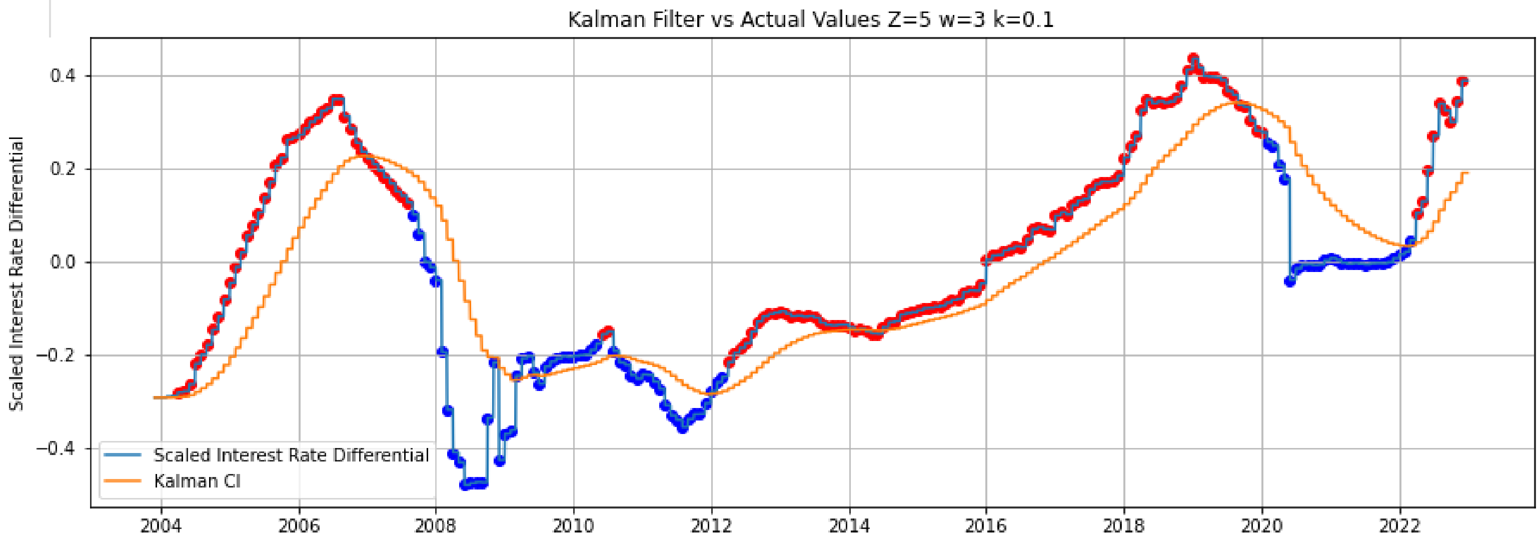
Out[23]:
```
-0.000108124954359872
```

It passed the weighted directional test which allows to reject the null hypothesis that the expected value of our weighted forecasts is 0 .

In [37]:

```
k = 0.1
w = 3
z = 5

df["Filter"] = df.ir_diff.ewm(alpha = k, adjust = False).mean()
# Create a dataframe at a daily frequency wiyh start and end
# dates that cover the observation period
drange = pd.date_range(start =df.index[0], end = "01/01/2023")
daily  = pd.DataFrame(index = drange)

# Integrate the monthly dta into the daily data
daily["test"] = df["test"]

daily["Upper"] = df["Upper"]
daily["Lower"] = df["Lower"]
daily["Filter"] = df["Filter"]
daily["ir_diff"] = df["ir_diff"]

# Fill NA values with the last available value
daily["Upper"] = daily["Upper"].ffill()
daily["Lower"] = daily["Lower"].ffill()
daily["Filter"] = daily["Filter"].ffill()
daily["ir_diff"] = daily["ir_diff"].ffill()

# fill the remaining NA values with 0's
# also populates the test column
daily = daily.fillna(0)

# We let the holding period (i) be 41 days
i = 41

# create a new column that we will populate with our daily position
daily.loc[:, str(i)+"_signal"] = 0

# loop through each day in the dataset
for j in daily.index:
    # If our monthly signal is not 0
    if daily.loc[j, "test"] != 0:
        # Make the next i days equal to the monthly signal
        daily.loc[j:j+datetime.timedelta(i), str(i)+"_signal"] = daily.loc[j, "test"]

# Below is the holding period I use for the CI strategy
i = 171
daily.loc[:, str(i)+"_signal"] = 0
for j in daily.index:
    if daily.loc[j, "test"] != 0:
        daily.loc[j:j+datetime.timedelta(i), str(i)+"_signal"] = daily.loc[j, "test"]

data2 = daily.dropna()
fig, ax = plt.subplots(figsize = (15, 5))
```

```python
ax.set_title("Kalman Filter vs Actual Values " + "Z="+str(z) + " w=" + str(w)+ " k=" + str(k))

ax.set_ylabel("Scaled Interest Rate Differential")

# Plot the actual series and the filter
ax.plot(data2["ir_diff"])
ax.plot(data2["Filter"])

# This code block is used to add confidence intervals when z > 0
#ax.fill_between(data2.index, data2.Lower, data2.Upper, color='b', alpha=.2)

# add scatterplots using boolean indexing
# We change the colors and shapes based on the conditions
ax.scatter(data2[data2.test == 1].index, data2[data2.test == 1]["ir_diff"], color = "blue")
ax.scatter(data2[data2.test == -1].index, data2[data2.test == -1]["ir_diff"], color = "red")
ax.legend(["Scaled Interest Rate Differential", "Kalman CI"])

# this code can let us zoom in on certain time periods
#plt.xlim([datetime.date(2022, 1, 1), datetime.date(2023, 1, 1)])
ax.grid()
```



In [29]:

```
daily["Eruo"]=df['Euro']
daily["Eruo"] = daily["Eruo"].ffill()
start = daily[["Eruo", "41_signal"]].index[0]
df1 = daily[daily.test != 0][["41_signal", "Eruo"]].copy()
df1['D'] = df1["41_signal"]
df1 = df1[:-1].copy()
df1['s_current'] = daily[daily.index.isin(df1.index)]["Eruo"].values
df1['s_future'] = daily[daily.index.isin(df1.index+datetime.timedelta(41))]["Eruo"].values

# Get the realized exchange rate
df1['R'] = np.where(df1['s_future'] >= df1['s_current'], 1, -1)
## Sample Covariance
df1['W'] = (df1['D']-np.mean(df1['D']))*(df1['R']-np.mean(df1['R']))
T_B = np.mean(df1['W'])
## Newey-West LRV estimator
dy = df1['W'] - np.mean(df1['W'])
gamma_0 = sum((dy)**2)/len(df1)
gamma_1 = np.mean((dy*dy.shift(-1))[:len(df1)-1])
LRV = gamma_0 + 2*(1-1/2)*gamma_1
## Test-statistic
from scipy.stats import norm

statistic = T_B/np.sqrt(LRV/df1.shape[0])
print('Test statistic : ', statistic, ', 5 % critical value : ', round(norm.ppf(0.95),2))
```

```
Test statistic :  -0.02468504071733396 , 5 % critical value :  1.64
```

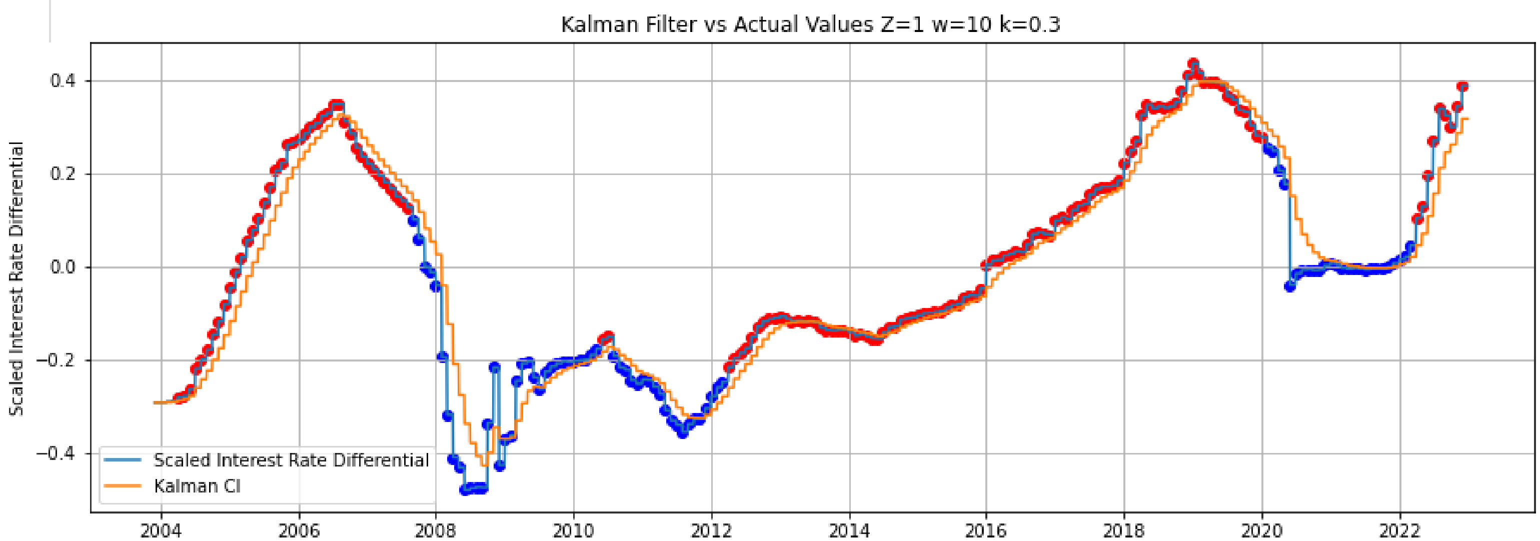
In [30]:

```
# Weighted Mean
df1['W_2'] = df1['D']*(df1['s_future']-df1['s_current'])
T_WB = np.mean(df1['W_2'])
## Newey-West LRV estimator
dy_2 = df1['W_2'] - np.mean(df1['W_2'])
gamma_0 = sum((dy_2)**2)/len(df1)
gamma_1 = np.mean((dy_2*dy_2.shift(-1))[:len(df1)-1])
LRV_2 = gamma_0 + 2*(1-1/2)*gamma_1
## Test-statistic
statistic_2 = T_WB/np.sqrt(LRV_2/len(df1))
print('Test statistic : ', statistic_2, ', 5 % critical value : ', round(norm.ppf(0.95),2))
```

```
Test statistic :  -0.04557798345955551 , 5 % critical value :  1.64
```

In [31]:
```
k = 0.3
w = 10
z = 1

df["Filter"] = df.ir_diff.ewm(alpha = k, adjust = False).mean()
# Create a dataframe at a daily frequency wiyh start and end
# dates that cover the observation period
drange = pd.date_range(start =df.index[0], end = "01/01/2023")
daily  = pd.DataFrame(index = drange)

# Integrate the monthly dta into the daily data
daily["test"] = df["test"]

daily["Upper"] = df["Upper"]
daily["Lower"] = df["Lower"]
daily["Filter"] = df["Filter"]
daily["ir_diff"] = df["ir_diff"]

# Fill NA values with the last available value
daily["Upper"] = daily["Upper"].ffill()
daily["Lower"] = daily["Lower"].ffill()
daily["Filter"] = daily["Filter"].ffill()
daily["ir_diff"] = daily["ir_diff"].ffill()

# fill the remaining NA values with 0's
# also populates the test column
daily = daily.fillna(0)

# We let the holding period (i) be 41 days
i = 41

# create a new column that we will populate with our daily position
daily.loc[:, str(i)+"_signal"] = 0

# loop through each day in the dataset
for j in daily.index:
    # If our monthly signal is not 0
    if daily.loc[j, "test"] != 0:
        # Make the next i days equal to the monthly signal
        daily.loc[j:j+datetime.timedelta(i), str(i)+"_signal"] = daily.loc[j, "test"]

# Below is the holding period I use for the CI strategy
i = 171
daily.loc[:, str(i)+"_signal"] = 0
for j in daily.index:
    if daily.loc[j, "test"] != 0:
        daily.loc[j:j+datetime.timedelta(i), str(i)+"_signal"] = daily.loc[j, "test"]

data2 = daily.dropna()
fig, ax = plt.subplots(figsize = (15, 5))
```

```python
ax.set_title("Kalman Filter vs Actual Values " + "Z="+str(z) + " w=" + str(w)+ " k=" + str(k))

ax.set_ylabel("Scaled Interest Rate Differential")

# Plot the actual series and the filter
ax.plot(data2["ir_diff"])
ax.plot(data2["Filter"])

# This code block is used to add confidence intervals when z > 0
#ax.fill_between(data2.index, data2.Lower, data2.Upper, color='b', alpha=.2)

# add scatterplots using boolean indexing
# We change the colors and shapes based on the conditions
ax.scatter(data2[data2.test == 1].index, data2[data2.test == 1]["ir_diff"], color = "blue")
ax.scatter(data2[data2.test == -1].index, data2[data2.test == -1]["ir_diff"], color = "red")
ax.legend(["Scaled Interest Rate Differential", "Kalman CI"])

# this code can let us zoom in on certain time periods
#plt.xlim([datetime.date(2022, 1, 1), datetime.date(2023, 1, 1)])
ax.grid()
```



In [32]:

```
daily["Euro"]=df['Euro']
daily["Euro"] = daily["Euro"].ffill()
start = daily[["Euro", "41_signal"]].index[0]
df1 = daily[daily.test != 0][["41_signal", "Euro"]].copy()
df1['D'] = df1["41_signal"]
df1 = df1[:-1].copy()
df1['s_current'] = daily[daily.index.isin(df1.index)]["Euro"].values
df1['s_future'] = daily[daily.index.isin(df1.index+datetime.timedelta(41))]["Euro"].values

# Get the realized exchange rate
df1['R'] = np.where(df1['s_future'] >= df1['s_current'], 1, -1)
## Sample Covariance
df1['W'] = (df1['D']-np.mean(df1['D']))*(df1['R']-np.mean(df1['R']))
T_B = np.mean(df1['W'])
## Newey-West LRV estimator
dy = df1['W'] - np.mean(df1['W'])
gamma_0 = sum((dy)**2)/len(df1)
gamma_1 = np.mean((dy*dy.shift(-1))[:len(df1)-1])
LRV = gamma_0 + 2*(1-1/2)*gamma_1
## Test-statistic
from scipy.stats import norm

statistic = T_B/np.sqrt(LRV/df1.shape[0])
print('Test statistic : ', statistic, ', 5 % critical value : ', round(norm.ppf(0.95),2))
```

```
Test statistic :  -0.024685040717333396 , 5 % critical value :  1.64
```

In [33]:

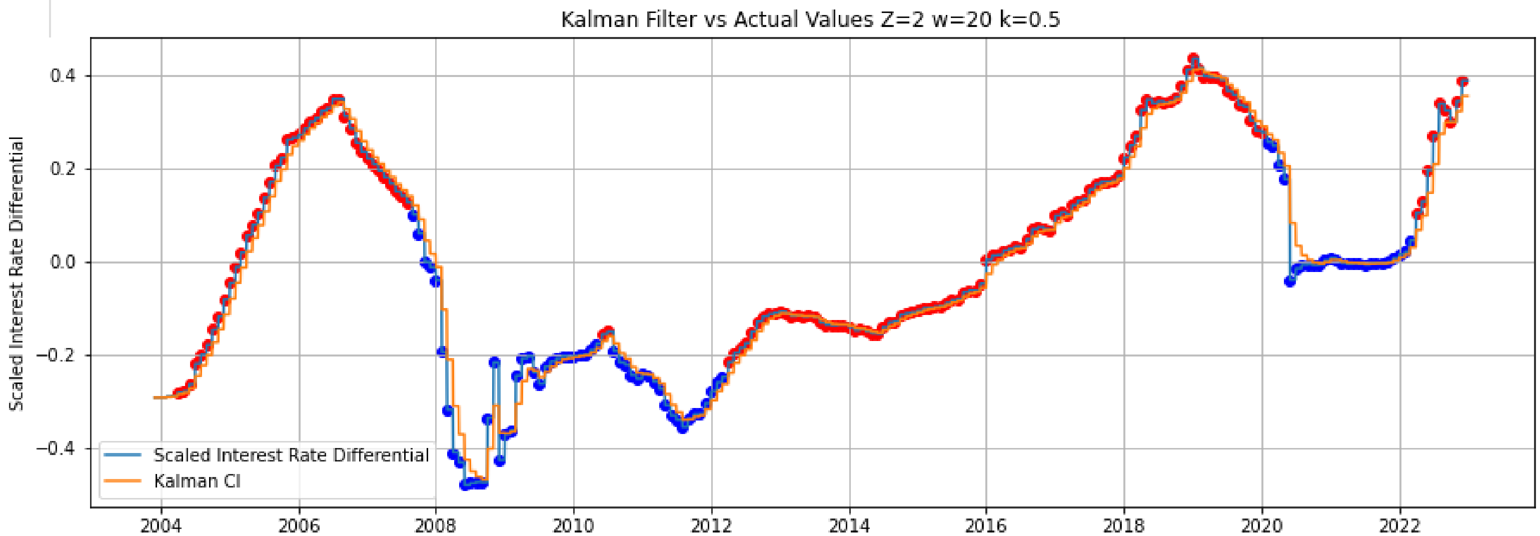
```
# Weighted Mean
df1['W_2'] = df1['D']*(df1['s_future']-df1['s_current'])
T_WB = np.mean(df1['W_2'])
## Newey-West LRV estimator
dy_2 = df1['W_2'] - np.mean(df1['W_2'])
gamma_0 = sum((dy_2)**2)/len(df1)
gamma_1 = np.mean((dy_2*dy_2.shift(-1))[:len(df1)-1])
LRV_2 = gamma_0 + 2*(1-1/2)*gamma_1
## Test-statistic
statistic_2 = T_WB/np.sqrt(LRV_2/len(df1))
print('Test statistic : ', statistic_2, ', 5 % critical value : ', round(norm.ppf(0.95),2))
```

```
Test statistic :  -0.04557798345955551 , 5 % critical value :  1.64
```

In [34]:

```
k = 0.5
w = 20
z = 2

df["Filter"] = df.ir_diff.ewm(alpha = k, adjust = False).mean()
# Create a dataframe at a daily frequency wiyh start and end
# dates that cover the observation period
drange = pd.date_range(start =df.index[0], end = "01/01/2023")
daily  = pd.DataFrame(index = drange)

# Integrate the monthly dta into the daily data
daily["test"] = df["test"]

daily["Upper"] = df["Upper"]
daily["Lower"] = df["Lower"]
daily["Filter"] = df["Filter"]
daily["ir_diff"] = df["ir_diff"]

# Fill NA values with the last available value
daily["Upper"] = daily["Upper"].ffill()
daily["Lower"] = daily["Lower"].ffill()
daily["Filter"] = daily["Filter"].ffill()
daily["ir_diff"] = daily["ir_diff"].ffill()

# fill the remaining NA values with 0's
# also populates the test column
daily = daily.fillna(0)

# We let the holding period (i) be 41 days
i = 100

# create a new column that we will populate with our daily position
daily.loc[:, str(i)+"_signal"] = 0

# loop through each day in the dataset
for j in daily.index:
    # If our monthly signal is not 0
    if daily.loc[j, "test"] != 0:
        # Make the next i days equal to the monthly signal
        daily.loc[j:j+datetime.timedelta(i), str(i)+"_signal"] = daily.loc[j, "test"]

# Below is the holding period I use for the CI strategy
i = 100
daily.loc[:, str(i)+"_signal"] = 0
for j in daily.index:
    if daily.loc[j, "test"] != 0:
        daily.loc[j:j+datetime.timedelta(i), str(i)+"_signal"] = daily.loc[j, "test"]

data2 = daily.dropna()
fig, ax = plt.subplots(figsize = (15, 5))
```

```
ax.set_title("Kalman Filter vs Actual Values " + "Z="+str(z) + " w=" + str(w)+ " k=" + str(k))

ax.set_ylabel("Scaled Interest Rate Differential")

# Plot the actual series and the filter
ax.plot(data2["ir_diff"])
ax.plot(data2["Filter"])

# This code block is used to add confidence intervals when z > 0
#ax.fill_between(data2.index, data2.Lower, data2.Upper, color='b', alpha=.2)

# add scatterplots using boolean indexing
# We change the colors and shapes based on the conditions
ax.scatter(data2[data2.test == 1].index, data2[data2.test == 1]["ir_diff"], color = "blue")
ax.scatter(data2[data2.test == -1].index, data2[data2.test == -1]["ir_diff"], color = "red")
ax.legend(["Scaled Interest Rate Differential", "Kalman CI"])

# this code can let us zoom in on certain time periods
#plt.xlim([datetime.date(2022, 1, 1), datetime.date(2023, 1, 1)])
ax.grid()
```



In [35]:

```
daily["Euro"]=df['Euro']
daily["Euro"] = daily["Euro"].ffill()
start = daily[["Euro", "100_signal"]].index[0]
df1 = daily[daily.test != 0][["100_signal", "Euro"]].copy()
df1['D'] = df1["100_signal"]
df1 = df1[:-1].copy()
df1['s_current'] = daily[daily.index.isin(df1.index)]["Euro"].values
df1['s_future'] = daily[daily.index.isin(df1.index+datetime.timedelta(41))]["Euro"].values

# Get the realized exchange rate
df1['R'] = np.where(df1['s_future'] >= df1['s_current'], 1, -1)
## Sample Covariance
df1['W'] = (df1['D']-np.mean(df1['D']))*(df1['R']-np.mean(df1['R']))
T_B = np.mean(df1['W'])
## Newey-West LRV estimator
dy = df1['W'] - np.mean(df1['W'])
gamma_0 = sum((dy)**2)/len(df1)
gamma_1 = np.mean((dy*dy.shift(-1))[:len(df1)-1])
LRV = gamma_0 + 2*(1-1/2)*gamma_1
## Test-statistic
from scipy.stats import norm

statistic = T_B/np.sqrt(LRV/df1.shape[0])
print('Test statistic : ', statistic, ', 5 % critical value : ', round(norm.ppf(0.95),2))
```

```
Test statistic :  -0.02468504071733396 , 5 % critical value :  1.64
```

In [36]:

```
# Weighted Mean
df1['W_2'] = df1['D']*(df1['s_future']-df1['s_current'])
T_WB = np.mean(df1['W_2'])
## Newey-West LRV estimator
dy_2 = df1['W_2'] - np.mean(df1['W_2'])
gamma_0 = sum((dy_2)**2)/len(df1)
gamma_1 = np.mean((dy_2*dy_2.shift(-1))[:len(df1)-1])
LRV_2 = gamma_0 + 2*(1-1/2)*gamma_1
## Test-statistic
statistic_2 = T_WB/np.sqrt(LRV_2/len(df1))
print('Test statistic : ', statistic_2, ', 5 % critical value : ', round(norm.ppf(0.95),2))
```

```
Test statistic :  -0.04557798345955551 , 5 % critical value :  1.64
```

The behavior of the strategy did not really change. After trying different parameters, I got very similar results and all strategies with different parameters passed all tests. I noticed that as I increase the values of k, z, w; the Kalman confidence interval and the Scaled Interest Rate Differential fit the data slightly better.

3

In [83]:
```
import math
```

In [93]:
```
face_value = 1000
coupon = 50
market_price = 984.96
t = 10

YTM = ((50 + (face_value-market_price)/2))/face_value+market_price/2


print("YTM = {:.5f}".format(ytm))
```

```
YTM = 0.05000
```

In [94]:
```
probability = 0.2
expected_cash_flow = (1 - probability) * face_value
present_value = expected_cash_flow / (1 + ytm)
present_value
```

Out[94]:
```
761.9047619047619
```

4

A

In [103]:
```
def yield_to_maturity(face_value, coupon_rate, market_price, iterations=100):
    ytm = coupon_rate
    for i in range(iterations):
        modified_duration = face_value / (coupon_rate + ytm)
        ytm = coupon_rate + (market_price - face_value) / modified_duration
    return ytm

face_value = 1000
coupon_rate = 50 / face_value
market_price = 1000

ytm = yield_to_maturity(face_value, coupon_rate, market_price)
print("Yield to maturity:", ytm)
```

```
Yield to maturity: 0.05
```

In [104]:
```
probability = 0.2
expected_cash_flow = (1 - probability) * face_value
present_value = expected_cash_flow / (1 + ytm)
present_value
```

Out[104]:
```
761.9047619047619
```

B

In [96]:

```
def cashFlw(F, c, n):
    csh = F*c
    cf = list()
    for i in range(n-1):
        cf.append(csh)
    cf.append(F + csh)
    return cf

def discount(r, n):
    d = list()
    for i in range(1, n+1):
        d.append(1/((1+r)**i))
    return d

def numerator(F, c, r, n):
    cf = cashFlw(F, c, n)
    d = discount(r, n)
    total = 0
    for i in range(1, n+1):
        total += (i*cf[i-1]*d[i-1])
    return total

def denominator(F, c, r, n):
    csh = F*c
    den = 0
    for i in range(1, n):
        den = den + (csh/((1+r)**i))
    den += ((csh+F)/((1+r)**n))
    return den

def duration(F, c, r, n):
    num = numerator(F, c, r, n)
    den = denominator(F, c, r, n)
    return num/den
```

In [100]:

```
# 10-Year Coupon Bond

F = 1000
n = 10
i = 0.1
c = 0.15

duration1 = duration(F, c, i, n)
duration1
```

Out[100]: 6.281090250274969

In [102]:

```
# 7-Year Coupon Bond


F2 = 1000
n2 = 7
i2 = 0.03
c2 = 0.03

duration2 = duration(F2, c2, i2, n2)
duration2
```

Out[102]: 6.417191443878188

Despite the magnitude of change in interest rate, the 7-year coupon bond would experience a greater change since it has a slightly higher duration, implying that is is more sensitive to changes in interest rates.

C

Portfolio duration = (0.3 x 6.417) + (0.7 x 6.237) = 6.291

5

A

The ECP announced that they would raise the interest rate by 0.5% in Feb 2023 preceding an increase of 0.5% in Dec 2022.In Dec 2022, the ECP was reassesing the rate of holdings of securities but in Feb 2023 they decided to reduce their holdings of securities. In both statements, The FOMC decided to raise the interest rate by 0.25% in Feb 2023 and Dec 2022. Both FOMC statements were not different to me. They increased interest rates to reduce uneployment & inflation and they decided to reduce their treasury holdings.

B

Despite the ECP's clear intentions of incresing interest rates consistently, I was surprised that the Euro depriciated because I expected foreign investments to sufficiently increase money to flow though they EU countries and reduce inflation. Nothing surprised me in the the FOMC February statement. What is most surprising is that they both deicded to raise interest rates within a few months.

C

Their might have not been enough foreign investments in Europe. The ECP has the capabality of redcung they treasury holdings even more to attemp to reduce inflation.

6

The Federal reserve can sell more government bonds to decrease money supply, which could slow down economic growth and therefore lower inflation.