Group E

Julius Gruber
Yiyang Fan
Tengxiao Fan
Xiaoliang Liu

Glimpse on Data

	observation_date	DGS3MO	DGS6MO	DGS1	DGS2	DGS3	DGS5	DGS7	DGS10
0	2014-01-02	0.07	0.09	0.13	0.39	0.76	1.72	2.41	3.00
1	2014-01-03	0.07	0.10	0.13	0.41	0.80	1.73	2.42	3.01
2	2014-01-06	0.05	0.08	0.12	0.40	0.78	1.70	2.38	2.98
3	2014-01-07	0.04	0.08	0.13	0.40	0.80	1.69	2.37	2.96
4	2014-01-08	0.05	0.08	0.13	0.43	0.87	1.77	2.44	3.01
...	...	...	...	...	...	...	...	...	...
2517	2024-02-13	5.45	5.32	4.99	4.64	4.44	4.31	4.33	4.31
2518	2024-02-14	5.43	5.31	4.94	4.56	4.38	4.25	4.27	4.27
2519	2024-02-15	5.43	5.30	4.93	4.56	4.36	4.22	4.25	4.24
2520	2024-02-16	5.44	5.31	4.98	4.64	4.43	4.29	4.31	4.30
2521	2024-02-20	5.44	5.32	4.97	4.59	4.38	4.25	4.28	4.27

2522 rows × 9 columns

The model

The interest fitting framework

$$y(s) = \beta_0 + \beta_1 e^{-\lambda s} + \beta_2 \lambda s e^{-\lambda s}$$

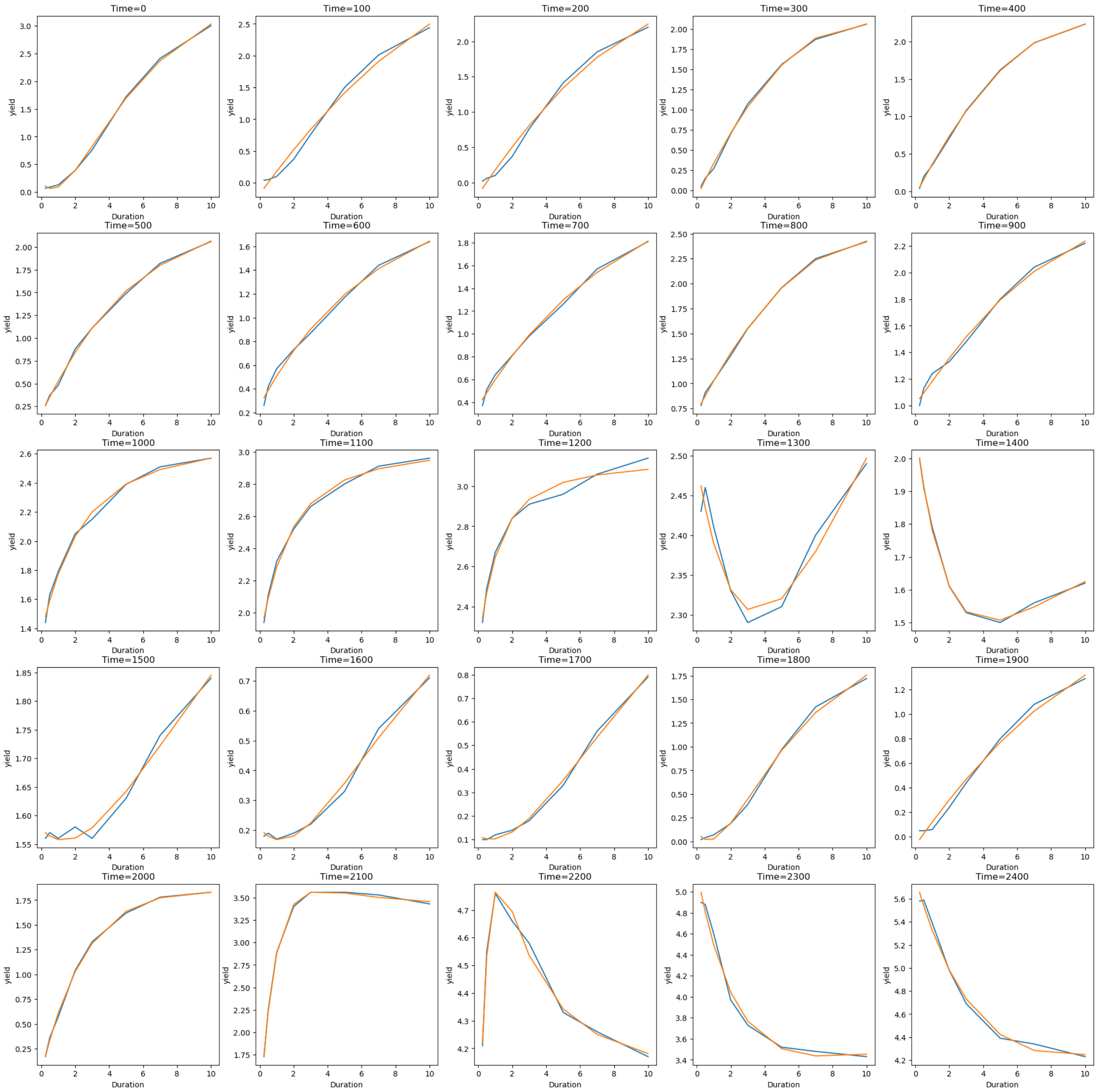
The zero coupon rate $Y(T)$ for maturity T corresponding to the instantaneous forward rate $y(s)$ is given by

$$Y(T) = \frac{1}{T} \int_0^T y(s) ds$$

Take integrals for both sides from 1 to T, we get

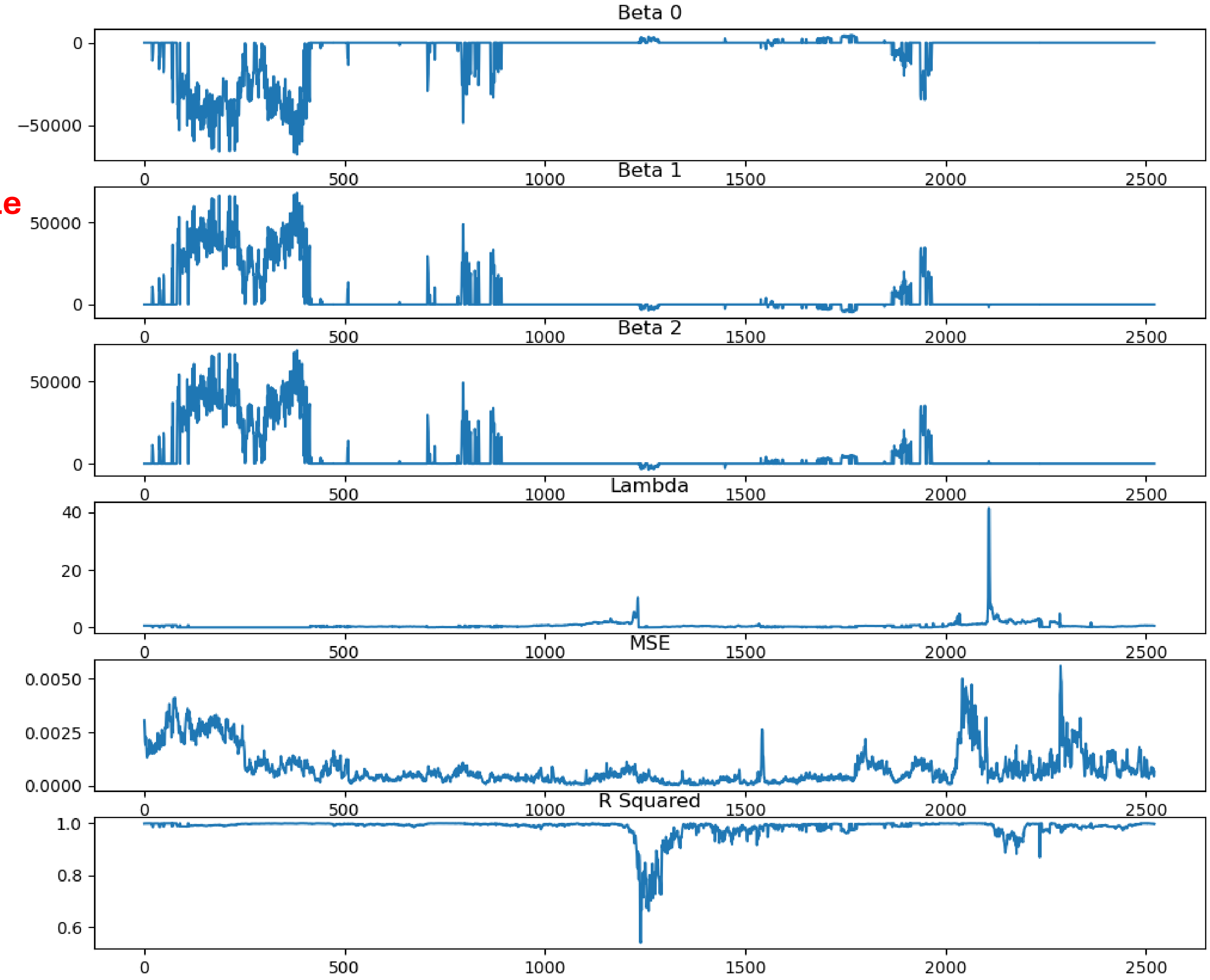
$$Y(T) = \beta_0 + \beta_1 \frac{1 - e^{-\lambda T}}{\lambda T} + \beta_2 \left(\frac{1 - e^{-\lambda T}}{\lambda T} - e^{-\lambda T} \right)$$

Lambda Not Fixed

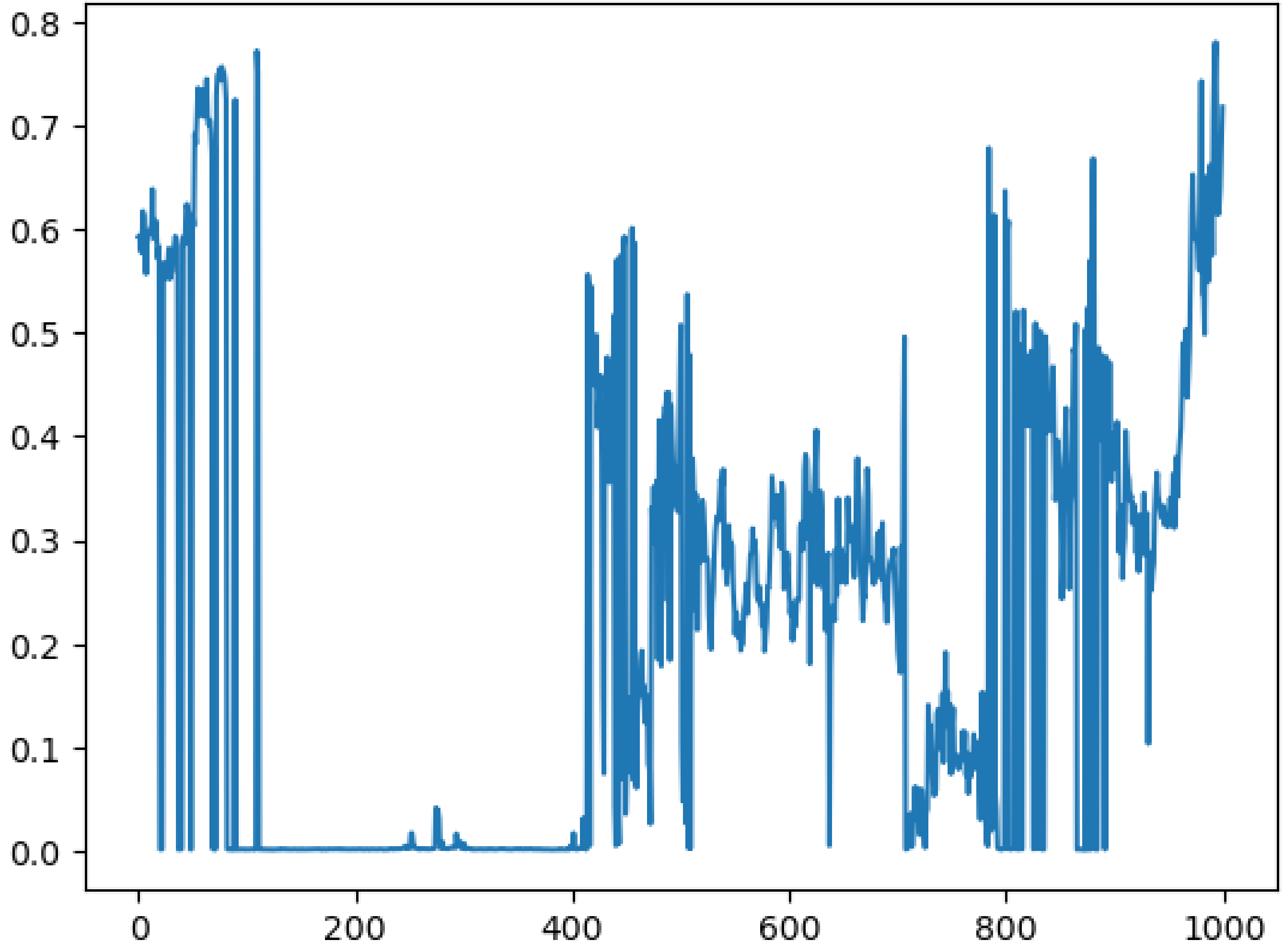


Not Stable
Not Predictable

Lambda Not Fixed

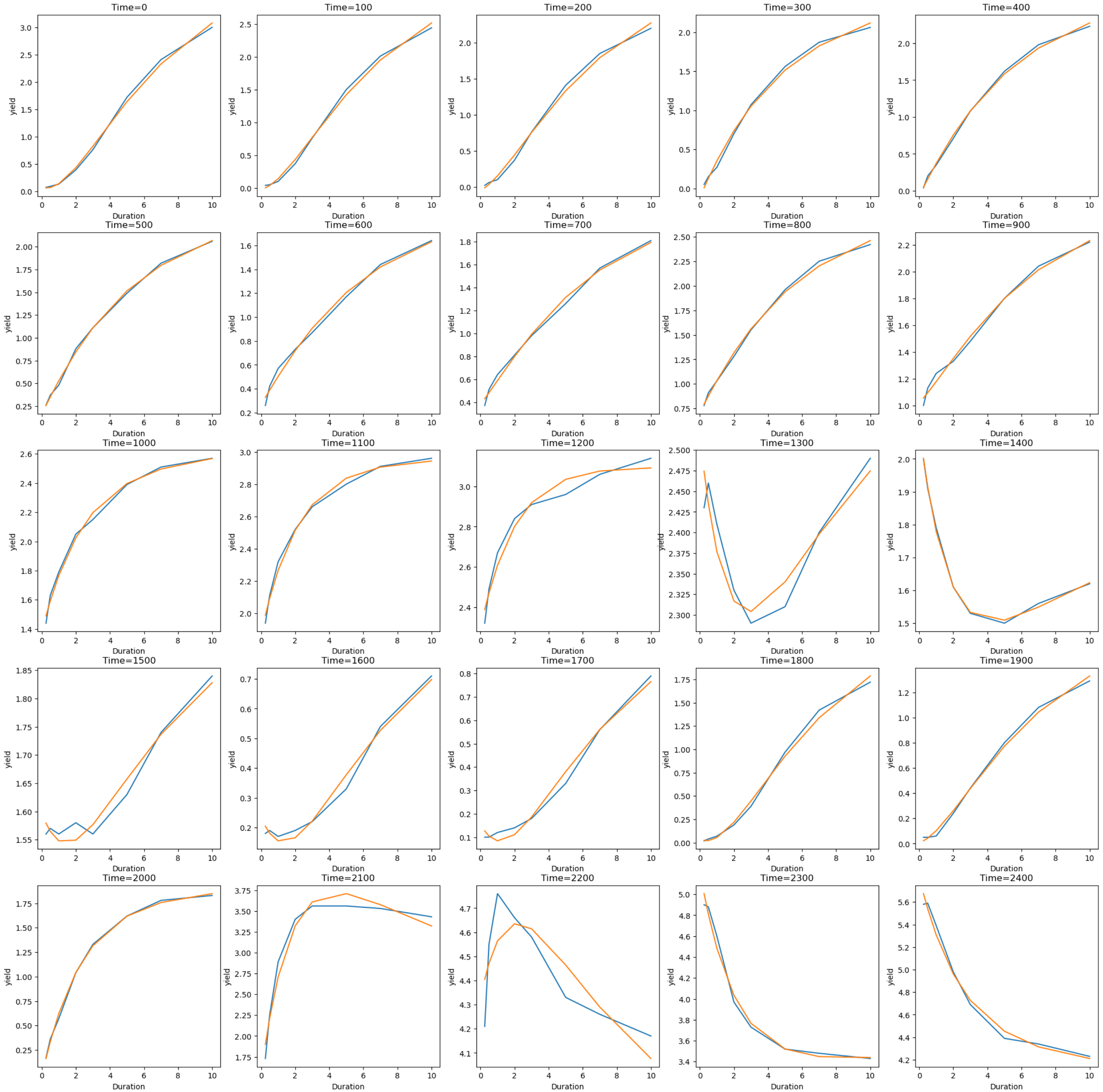


Part of Lambda

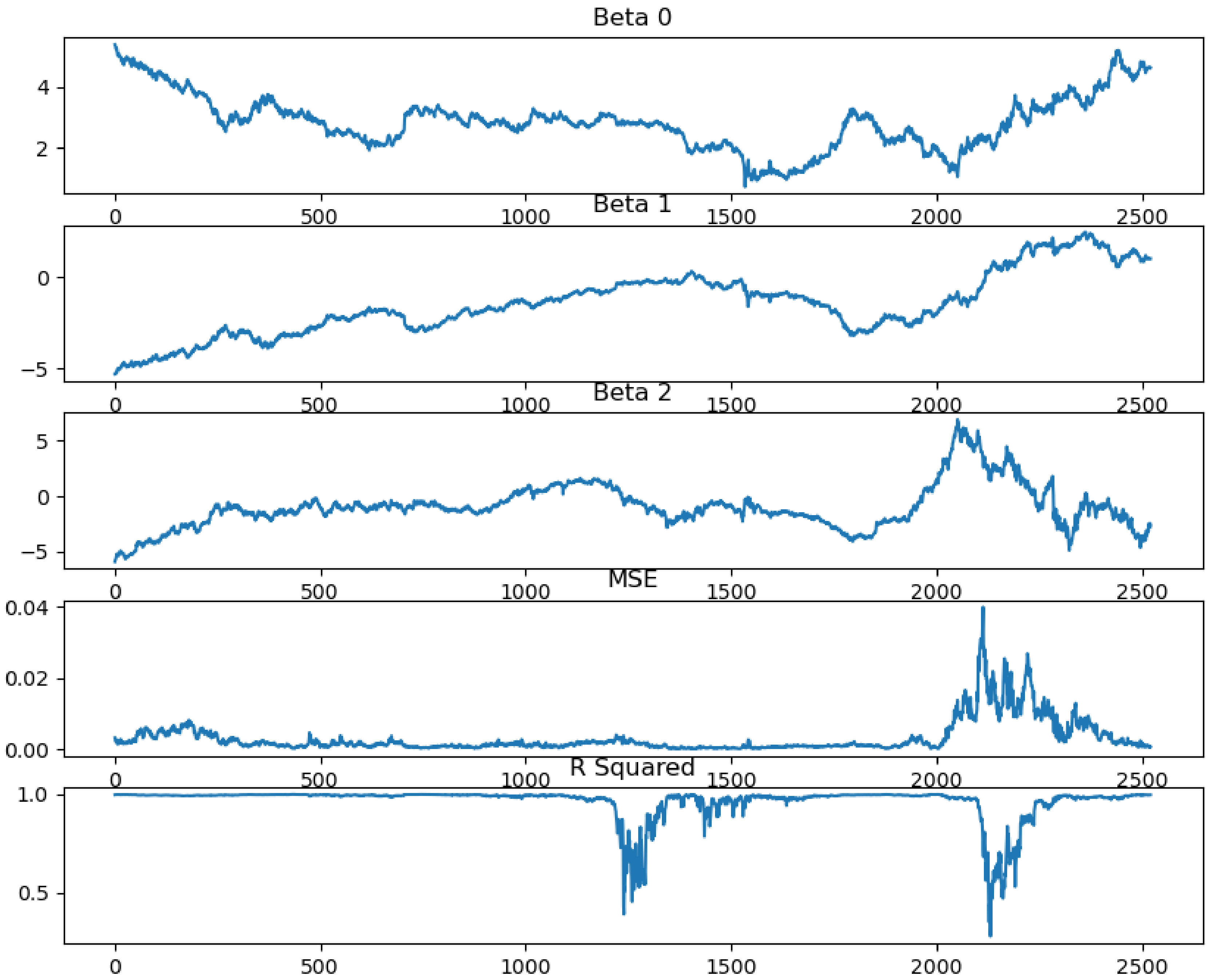


Lambda goes to zero -> Unstability

Fixed Lambda = 0.47

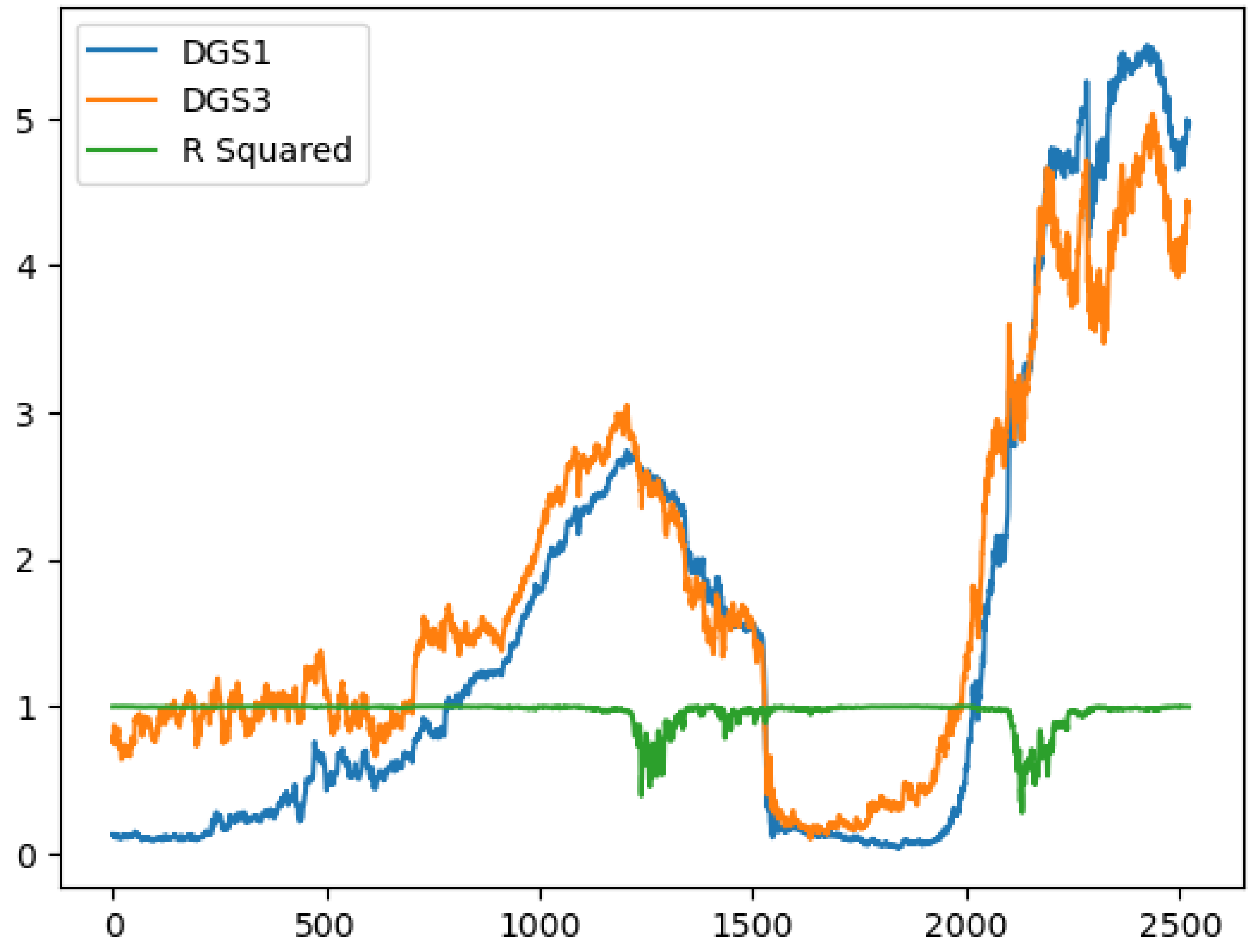


Fixed Lambda = 0.47



Stable
But Worse Fit

Inverted Yield Curve

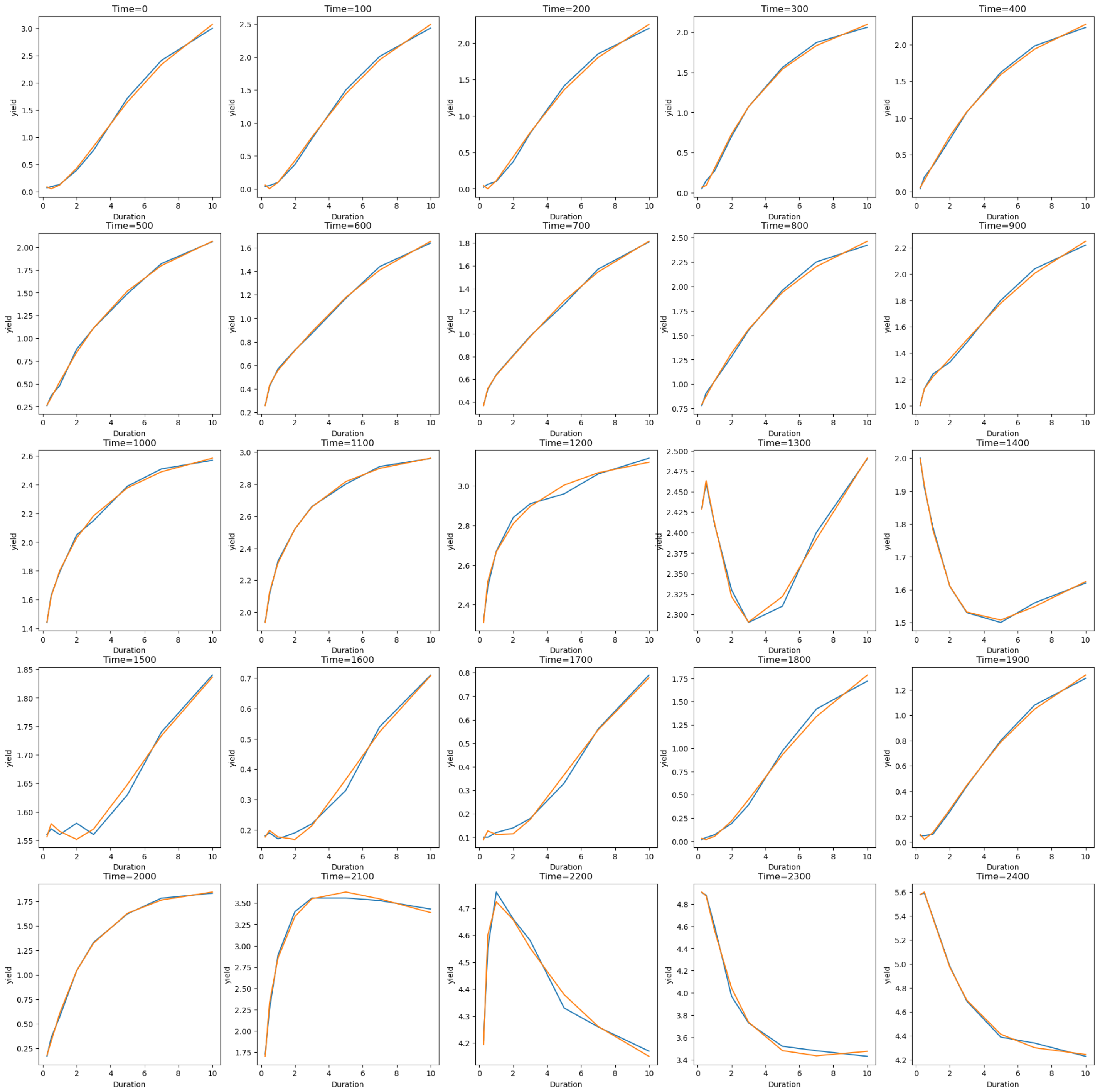


Improvement: Svensson model

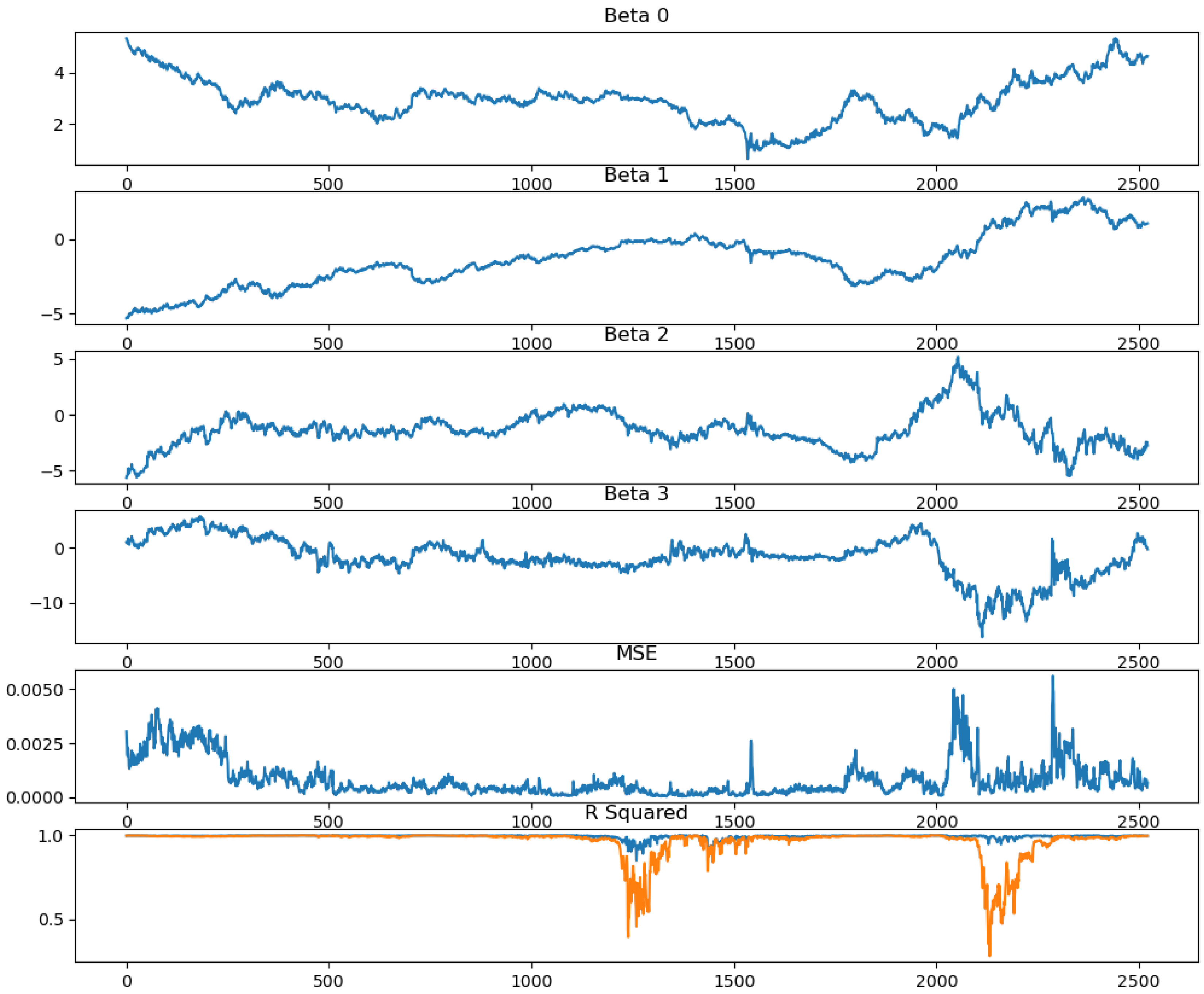
$$Y(T) = \beta_0 + \beta_1 \frac{1 - e^{-\lambda_1 T}}{\lambda_1 T} + \beta_2 \left(\frac{1 - e^{-\lambda_1 T}}{\lambda_1 T} - e^{-\lambda_1 T} \right) + \beta_3 \left(\frac{1 - e^{-\lambda_2 T}}{\lambda_2 T} - e^{-\lambda_2 T} \right)$$

Add one more lambda term to add some degrees of freedom. Typically, λ_2 are chosen to be large to fit the distinct shape of yield curves

Svensson Model



Svensson Model



Stable
And Good FIt

pmdarima 2.0.4

```
pip install pmdarima
```

Latest version

Released: Oct 23, 2023

Python's forecast::auto.arima equivalent

Project description

pmdarima

pypi package 2.0.4 | PASSED | Mac and Windows Builds passing | codecov 98% | python 3.7+ | downloads 54M | downloads/week 498k

Pmdarima (originally `pyramid-arima`, for the anagram of 'py' + 'arima') is a statistical library designed to fill the void in Python's time series analysis capabilities. This includes:

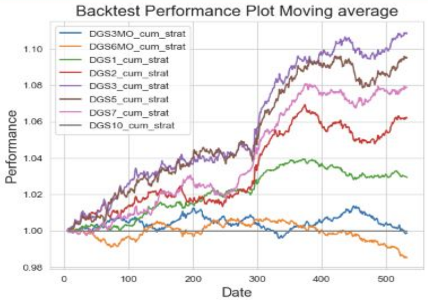
- The equivalent of R's `auto.arima` functionality
- A collection of statistical tests of stationarity and seasonality
- Time series utilities, such as differencing and inverse differencing
- Numerous endogenous and exogenous transformers and featurizers, including Box-Cox and Fourier

```python
arima_model_lambdas_beta_0 = pm.auto_arima(training_data["beta_0"], trace=True,max_p=10,max_d=3,max_q=10, stepwise=False,
information_criterion='aic')
arima_model_lambdas_beta_1 = pm.auto_arima(training_data["beta_1"], trace=True,max_p=10,max_d=3,max_q=10, stepwise=False,
information_criterion='aic')
arima_model_lambdas_beta_2 = pm.auto_arima(training_data["beta_2"], trace=True,max_p=10,max_d=3,max_q=10, stepwise=False,
information_criterion='aic')
```

```
 ARIMA(0,1,0)(0,0,0)[1] intercept   : AIC=-5459.318, Time=0.08 sec
 ARIMA(0,1,1)(0,0,0)[1] intercept   : AIC=-5457.640, Time=0.09 sec
 ARIMA(0,1,2)(0,0,0)[1] intercept   : AIC=-5455.855, Time=0.19 sec
 ARIMA(0,1,3)(0,0,0)[1] intercept   : AIC=-5461.411, Time=0.28 sec
 ARIMA(0,1,4)(0,0,0)[1] intercept   : AIC=-5466.153, Time=0.34 sec
 ARIMA(0,1,5)(0,0,0)[1] intercept   : AIC=-5464.557, Time=0.37 sec
 ARIMA(1,1,0)(0,0,0)[1] intercept   : AIC=-5457.634, Time=0.04 sec
 ARIMA(1,1,1)(0,0,0)[1] intercept   : AIC=-5455.637, Time=0.13 sec
 ARIMA(1,1,2)(0,0,0)[1] intercept   : AIC=-5459.492, Time=0.52 sec
 ARIMA(1,1,3)(0,0,0)[1] intercept   : AIC=-5462.922, Time=0.44 sec
 ARIMA(1,1,4)(0,0,0)[1] intercept   : AIC=-5464.384, Time=0.36 sec
 ARIMA(2,1,0)(0,0,0)[1] intercept   : AIC=-5455.771, Time=0.05 sec
 ARIMA(2,1,1)(0,0,0)[1] intercept   : AIC=-5453.785, Time=0.23 sec
 ARIMA(2,1,2)(0,0,0)[1] intercept   : AIC=-5460.230, Time=0.71 sec
 ARIMA(2,1,3)(0,0,0)[1] intercept   : AIC=-5463.627, Time=0.63 sec
 ARIMA(3,1,0)(0,0,0)[1] intercept   : AIC=-5461.211, Time=0.06 sec
 ARIMA(3,1,1)(0,0,0)[1] intercept   : AIC=-5463.084, Time=0.59 sec
 ARIMA(3,1,2)(0,0,0)[1] intercept   : AIC=-5462.714, Time=0.66 sec
 ARIMA(4,1,0)(0,0,0)[1] intercept   : AIC=-5465.505, Time=0.16 sec
 ARIMA(4,1,1)(0,0,0)[1] intercept   : AIC=-5463.478, Time=0.18 sec
 ARIMA(5,1,0)(0,0,0)[1] intercept   : AIC=-5463.826, Time=0.29 sec

Best model:  ARIMA(0,1,4)(0,0,0)[1] intercept
Total fit time: 6.406 seconds
 ARIMA(0,2,0)(0,0,0)[1]             : AIC=-4010.832, Time=0.01 sec
```

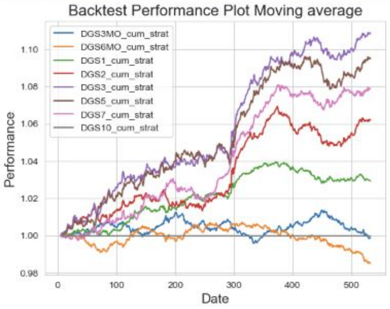
[6]:
```python
def bond_price(yie, time, r):
    # Adjust yield to decimal form
    yie_decimal = yie / 100

    # Initialize sum of cash flows
    sum_cash_flows = 0

    # If time is less than or exactly 1 year, calculate the price directly
    if time <= 1:
        # For a bond maturing in less than or equal to one year, we have only one cash flow
        # This cash flow includes the final coupon payment plus the face value, discounted back to present value
        sum_cash_flows = (r + 100) / (1 + yie_decimal * time)
    else:
        # For bonds with more than one year to maturity, calculate the present value of each coupon payment
        for count in range(1, int(time)):
            sum_cash_flows += r / ((1 + yie_decimal) ** count)
        # Add the present value of the final coupon payment plus face value
        sum_cash_flows += (r + 100) / ((1 + yie_decimal) ** time)

    return sum_cash_flows
```

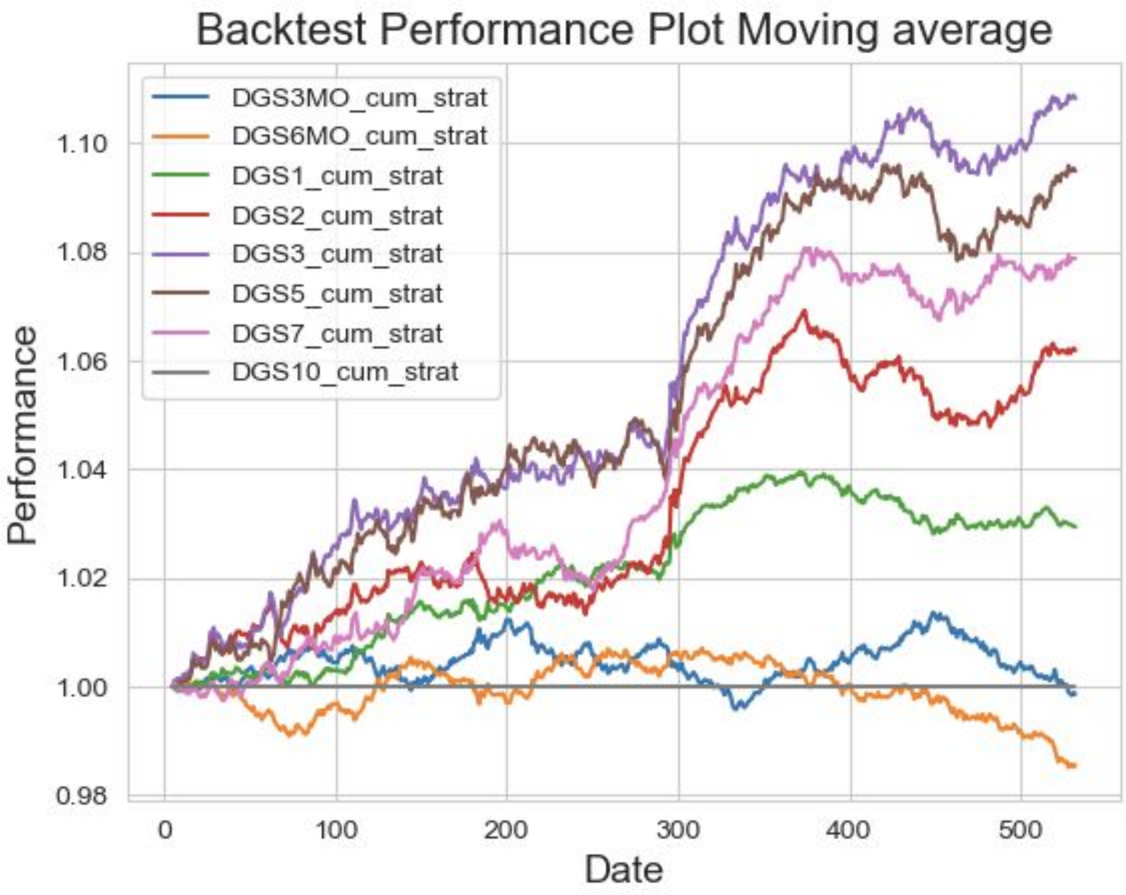
```python
def model(T, beta_0, beta_1, beta_2, lambd):
    term1 = (1 - np.exp(-lambd * T)) / (lambd * T)
    term2 = term1 - np.exp(-lambd * T)
    Y = beta_0 + beta_1 * term1 + beta_2 * term2
    return Y

for row in range(5,len(backtest)):
    y_data = backtest.iloc[row].values[1:]

    x_data = np.array([0.25,0.5,1, 2, 3, 5, 7, 10])
    y_pred = model(x_data, y_data[12] , y_data[13], y_data[14] ,y_data[11])
    #print(y_pred)
    i = 0
    r = backtest["DGS10"][row]
    for index, column in enumerate(columns_to_predict):
        backtest.loc[row,f'{column}_pred'] = y_pred[i]
        i +=1
        yie = backtest[column][row]
        time = x_data[index]
        backtest.loc[row, f'{column}_b_price'] = bond_price(yie, time, r)
        #updated_model = fitted_model.append(new_data, refit=False)
        # Now you can forecast future values from the updated model
        #forecast = updated_model.forecast(steps=n_steps)
        backtest.loc[row, f'{column}_b_price_pred'] = bond_price(y_pred[index], time, r)

        if backtest[f'{column}_b_price'][row - 1]!=0:
            backtest.loc[row,f'{column}_bond_log_return' ] = np.log(backtest[f'{column}_b_price'][row] / backtest[f'{column}_b_price'][row - 1])
            backtest.loc[row,f'{column}_strategy_return'] = backtest[f'{column}_bond_log_return'][row] * backtest[f'{column}_pos'][row]
        else:
            backtest.loc[row,f'{column}_bond_log_return' ] = 0
            backtest.loc[row,f'{column}_strategy_return'] = backtest[f'{column}_bond_log_return'][row] * backtest[f'{column}_pos'][row]

        if backtest.loc[row, f'{column}_b_price_pred'] < backtest.loc[row, f'{column}_b_price']:
            backtest.loc[row,f'{column}_pos'] = 1
        else:
            backtest.loc[row,f'{column}_pos'] = -1
        backtest.loc[row,f'{column}_strategy_return'] = backtest[f'{column}_bond_log_return'][row] * backtest[f'{column}_pos'][row]



plt.xticks(rotation=45)
sns.set_style('whitegrid')
plt.title('Backtest Performance Plot Moving average', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Performance', fontsize=14)

#plt.xticks(time_period)
for column in columns_to_predict:
    backtest[ f'{column}_cum_strat'] = backtest[f'{column}_strategy_return'][5:].cumsum().apply(np.exp)
    backtest[ f'{column}_cum_strat'].plot()
plt.legend()

plt.savefig("047_moving.png")
```



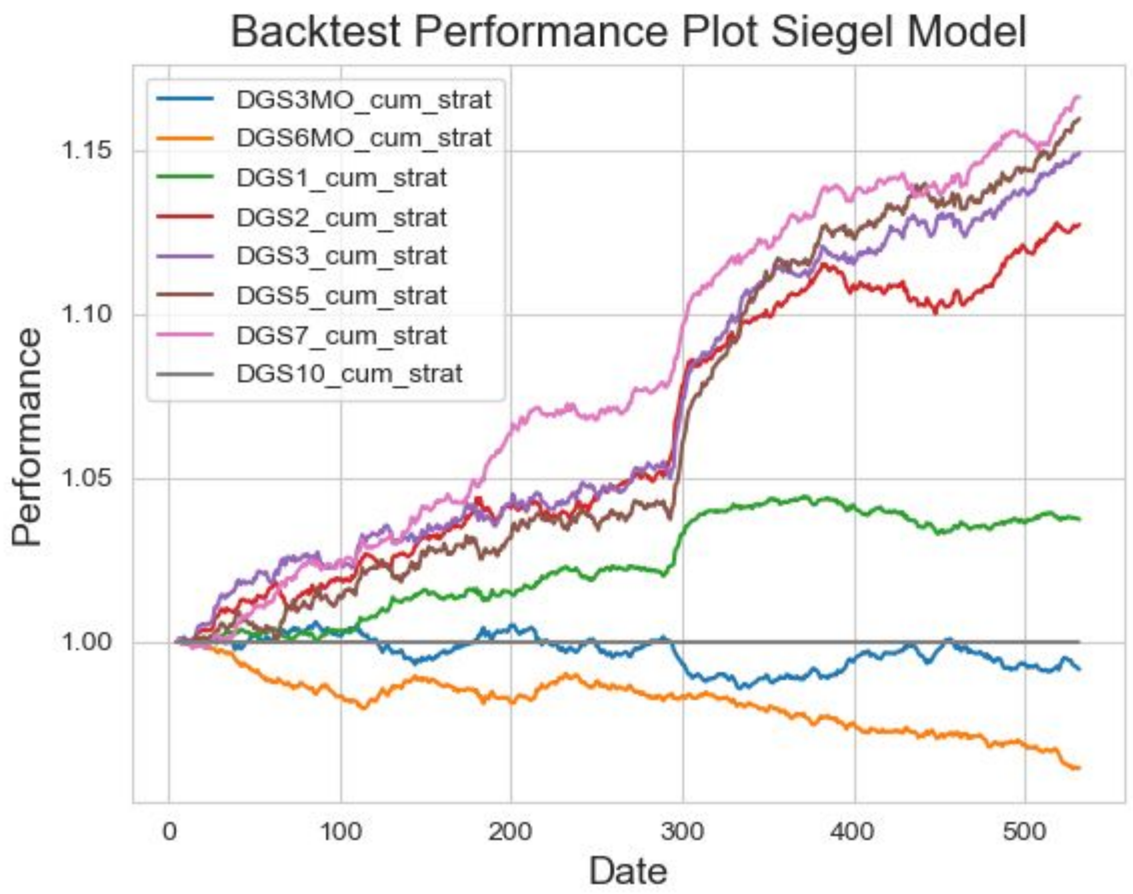
```python
from tqdm import tqdm
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA

# Initialize prediction lists
beta_0_pred_arima = []
beta_1_pred_arima = []
beta_2_pred_arima = []
beta_3_pred_arima = []

# Assuming ar_beta_0, ar_beta_1, and ar_beta_2 are defined and initialized ARIMA models
# And backtest is a DataFrame with 'beta_0', 'beta_1', 'beta_2' columns
# tr_b0, tr_b1, tr_b2 are the training datasets for each beta respectively

for t in tqdm(range(len(backtest)), desc='Forecasting ARIMA'):
    # Forecast beta_0
    forecast_b0 = ar_beta_0.forecast(steps=1)
    beta_0_pred_arima.append(forecast_b0.iloc[0])
    new_obs_beta_0 = backtest["beta_0"][t]
    tr_b0 = pd.concat([tr_b0, pd.Series(new_obs_beta_0)], ignore_index=True)
    model_b0 = ARIMA(tr_b0, order=(0, 1, 4))
    ar_beta_0 = model_b0.fit()

    # Forecast beta_1
    forecast_b1 = ar_beta_1.forecast(steps=1)
    beta_1_pred_arima.append(forecast_b1.iloc[0])
    new_obs_beta_1 = backtest["beta_1"][t]
    tr_b1 = pd.concat([tr_b1, pd.Series(new_obs_beta_1)], ignore_index=True)
    model_b1 = ARIMA(tr_b1, order=(5, 2, 0))
    ar_beta_1 = model_b1.fit()

    # Forecast beta_2
    forecast_b2 = ar_beta_2.forecast(steps=1)
    beta_2_pred_arima.append(forecast_b2.iloc[0])
    new_obs_beta_2 = backtest["beta_2"][t]
    tr_b2 = pd.concat([tr_b2, pd.Series(new_obs_beta_2)], ignore_index=True)
    model_b2 = ARIMA(tr_b2, order=(3, 1, 1))
    ar_beta_2 = model_b2.fit()

    # Forecast beta_3
    forecast_b3 = ar_beta_3.forecast(steps=1)
    beta_3_pred_arima.append(forecast_b3.iloc[0])
    new_obs_beta_3 = backtest["beta_3"][t]
    tr_b3 = pd.concat([tr_b3, pd.Series(new_obs_beta_3)], ignore_index=True)
    #fix the parameter lag
    model_b3 = ARIMA(tr_b3, order=(2, 1, 1))
    ar_beta_3 = model_b3.fit()
```

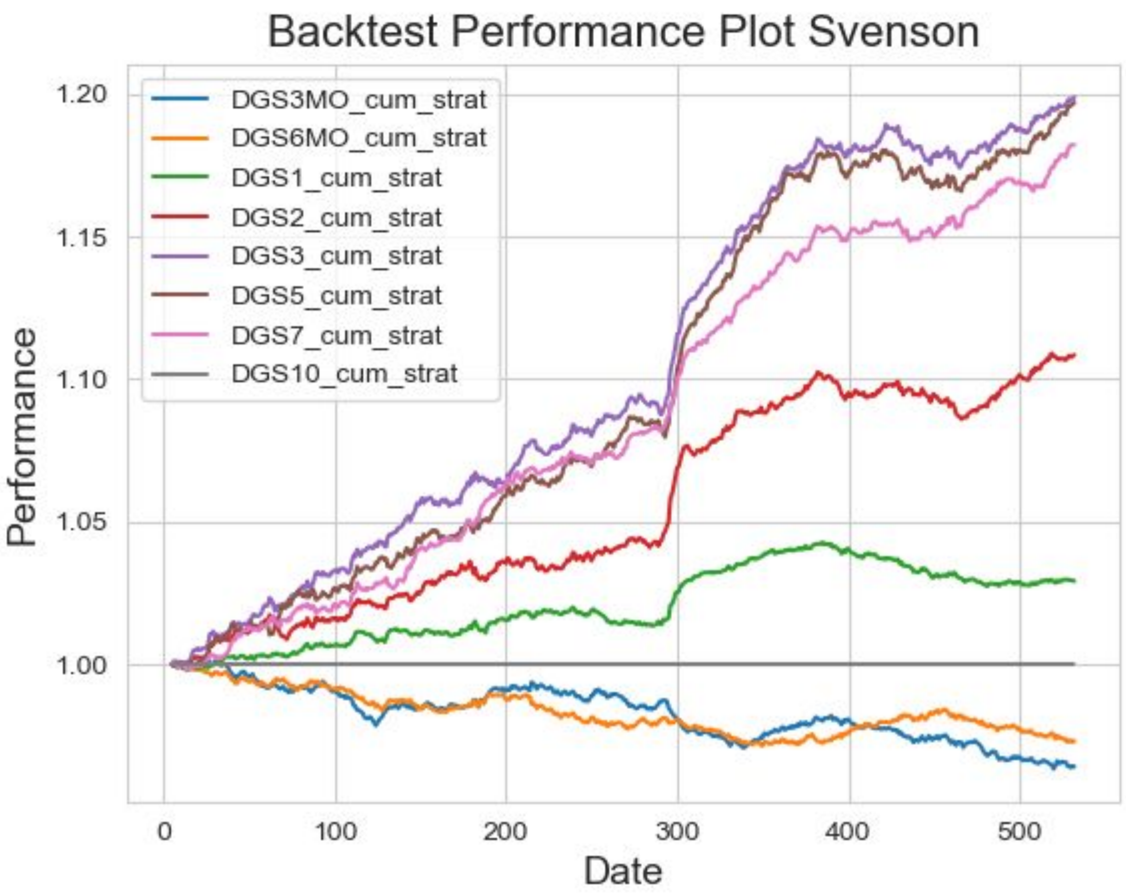
```python
]: def model(T, beta_0, beta_1, beta_2, lambd):
    term1 = (1 - np.exp(-lambd * T)) / (lambd * T)
    term2 = term1 - np.exp(-lambd * T)
    Y = beta_0 + beta_1 * term1 + beta_2 * term2
    return Y

for row in range(5,len(backtest)):
    y_data = backtest.iloc[row].values[1:]

    x_data = np.array([0.25,0.5,1, 2, 3, 5, 7, 10])
    y_pred = model(x_data, y_data[12] , y_data[13], y_data[14] ,y_data[11])
    #print(y_pred)
    i = 0
    r = backtest["DGS10"][row]
    for index, column in enumerate(columns_to_predict):
        backtest.loc[row,f'{column}_pred'] = y_pred[i]
        i +=1
        yie = backtest[column][row]
        time = x_data[index]
        backtest.loc[row, f'{column}_b_price'] = bond_price(yie, time, r)
        #updated_model = fitted_model.append(new_data, refit=False)
        # Now you can forecast future values from the updated model
        #forecast = updated_model.forecast(steps=n_steps)
        backtest.loc[row, f'{column}_b_price_pred'] = bond_price(y_pred[index], time, r)

        if backtest[f'{column}_b_price'][row - 1]!=0:
            backtest.loc[row,f'{column}_bond_log_return' ] = np.log(backtest[f'{column}_b_price'][row] / backtest[f'{column}_b_price'][row - 1])
            backtest.loc[row,f'{column}_strategy_return'] = backtest[f'{column}_bond_log_return'][row] * backtest[f'{column}_pos'][row]
        else:
            backtest.loc[row,f'{column}_bond_log_return' ] = 0
            backtest.loc[row,f'{column}_strategy_return'] = backtest[f'{column}_bond_log_return'][row] * backtest[f'{column}_pos'][row]

        if backtest.loc[row, f'{column}_b_price_pred'] < backtest.loc[row, f'{column}_b_price']:
            backtest.loc[row,f'{column}_pos'] = 1
        else:
            backtest.loc[row,f'{column}_pos'] = -1
        backtest.loc[row,f'{column}_strategy_return'] = backtest[f'{column}_bond_log_return'][row] * backtest[f'{column}_pos'][row]



plt.xticks(rotation=45)
sns.set_style('whitegrid')
plt.title('Backtest Performance Plot Moving average', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Performance', fontsize=14)

#plt.xticks(time_period)
for column in columns_to_predict:
    backtest[ f'{column}_cum_strat'] = backtest[f'{column}_strategy_return'][5:].cumsum().apply(np.exp)
    backtest[ f'{column}_cum_strat'].plot()
plt.legend()

plt.savefig("047_moving.png")
```





Backtest Performance Plot Moving average
DGS3MO_cum_strat
DGS6MO_cum_strat
DGS1_cum_strat
DGS2_cum_strat
DGS3_cum_strat
DGS5_cum_strat
DGS7_cum_strat
DGS10_cum_strat
Performance
1.10
1.08
1.06
1.04
1.02
1.00
0.98
0
100
200
300
400
500
Date


Backtest Performance Plot Siegel Model
DGS3MO_cum_strat
DGS6MO_cum_strat
DGS1_cum_strat
DGS2_cum_strat
DGS3_cum_strat
DGS5_cum_strat
DGS7_cum_strat
DGS10_cum_strat
1.15
1.10
1.05
1.00
Performance
0
100
200
300
400
500
Date



Backtest Performance Plot Svenson
DGS3MO_cum_strat
DGS6MO_cum_strat
DGS1_cum_strat
DGS2_cum_strat
DGS3_cum_strat
DGS5_cum_strat
DGS7_cum_strat
DGS10_cum_strat
1.20
1.15
1.10
1.05
1.00
Performance
0
100
200
300
400
500
Date

Our Approach

We aim to construct features from raw data, including bond rates and β_{0-2} values, to predict future bond rates across different maturities. Our constructed features includes:

1. P - representing raw bond rates and β_{0-2} values.
2. ret - the log-return.
3. ret_{MA} - the log-return with moving average over windows of 5, 30, and 120 days.

Utilizing the auto-ML library, specifically pycaret, we engage machine learning algorithms to aggregate these features for predicting future rates. Our prediction target is:

$$ret_{\text{FUTURE-1day}}$$

Predicting the log-return in rates, which is dimensionless and unaffected by spikes in rates outside the training set's range, is our focus. We also attempt predicting the log-return of β s, aiming to forecast future curve fits and potentially unveil profitable trading strategies.

Result

The summary of our findings is as follows:

Utilizing the naively constructed features (and their subsets) with a variety of models (including tree-based models, neural networks, and linear regression), we could not surpass a random regression outcome for one-day-ahead return prediction. But this does't rule out the possibility of longer horizen trading strategy. A similar outcome is observed for β prediction, with nearly 0 R^2 on the training set, indicating minimal improvement over naive mean prediction. The test set exhibited a negative R^2, suggesting predictions worse than naive mean predictions:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

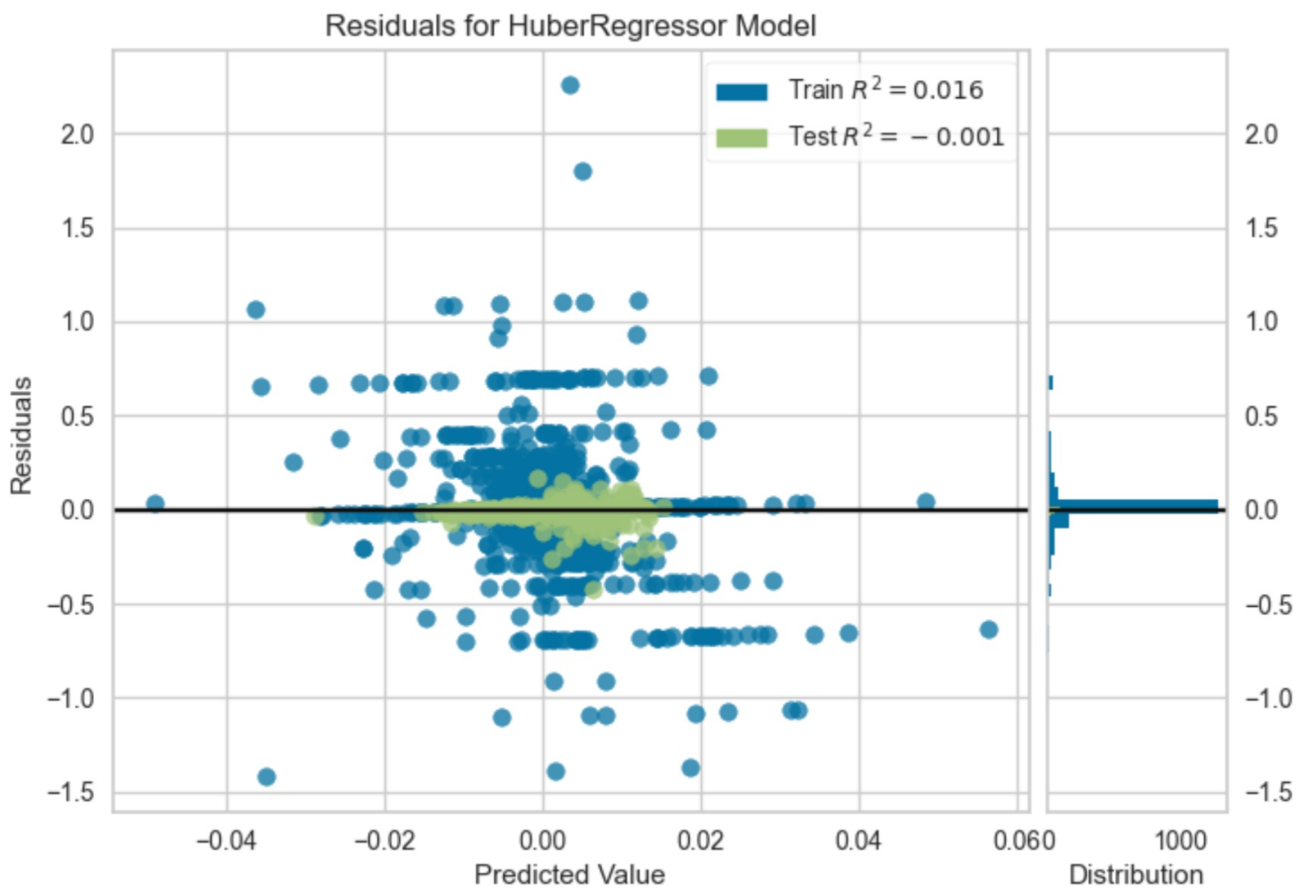
Adjusting the prediction target, including binary classification and raw rate value prediction, was also explored.

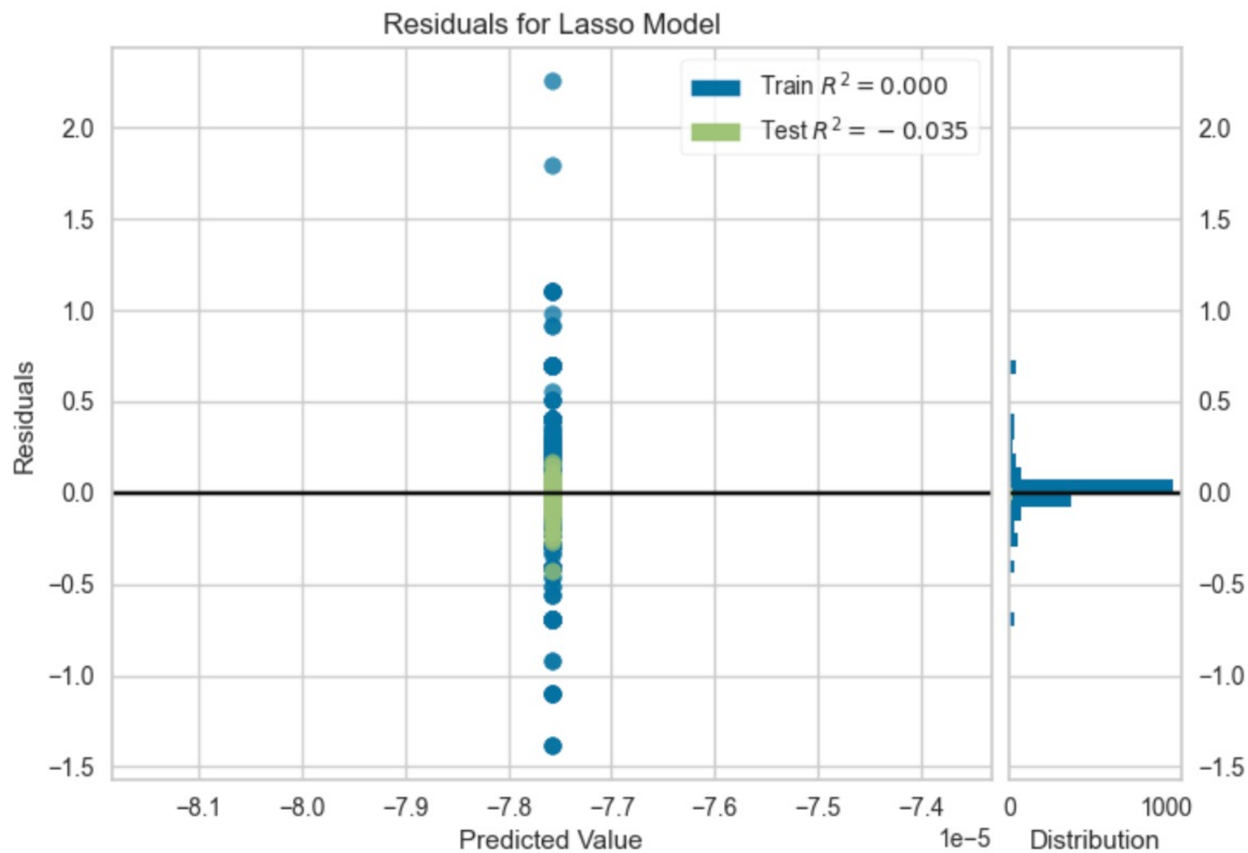
Furthermore, improper cross-validation splitting, e.g., using k-fold cross-validation, falsely inflates R^2. This phenomenon, stemming from look-ahead bias (data leakage), results in poor test set performance. A time-series split should be employed to lower R^2 yet yield more reliable and realistic outcomes for trading scenarios.

```python
from pycaret.regression import *

train_df = df.drop(columns="observation_date")
# construct log returns of each column
for name in names:
    train_df[name+'_ret'] = np.log(train_df[name]) - np.log(train_df[name].shift(1))
for name in ['beta_0', 'beta_1', 'beta_2']:
    train_df[name+'_ret'] = train_df[name] - train_df[name].shift(1)
# drop the row with nan
train_df["target"] = train_df["DGS3MO_ret"].shift(-1)
train_df = train_df.dropna()
train_df_raw = train_df.drop(columns=["beta_0", "beta_1", "beta_2", "beta_0_ret","beta_1_ret","beta_2_ret"])
train_df_ret = train_df.drop(columns=["DGS3MO", "DGS6MO", "DGS1", "DGS2" ,"DGS3", "DGS5", "DGS7","DGS10"])
# for name in [_+"_ret" for _ in ["DGS3MO", "DGS6MO", "DGS1", "DGS2" ,"DGS3", "DGS5", "DGS7","DGS10"]+["beta_0", "beta_1", "beta_2"]]:
#     for window in [5,30]:
#         train_df_ret[name+"_rolling_mean_"+str(window)] = train_df_ret[name].rolling(window).mean()
train_df_ret.dropna(inplace=True)

rg1 = setup(train_df, target = 'target',test_data = test_df,index = False, fold_strategy="timeseries", fold = 2, fold_shuffle=False, data_split_shuffle = False)
# rg1 = setup(train_df_ret, target = 'target',test_data = test_df,index = False)

# Compare all models
best_model = compare_models(turbo = False)

# create_model
model = create_model(best_model)
# model = create_model('rf')

# tune_model
tuned_model = tune_model(model)

plot_model(model)

predictions = predict_model(model)
```

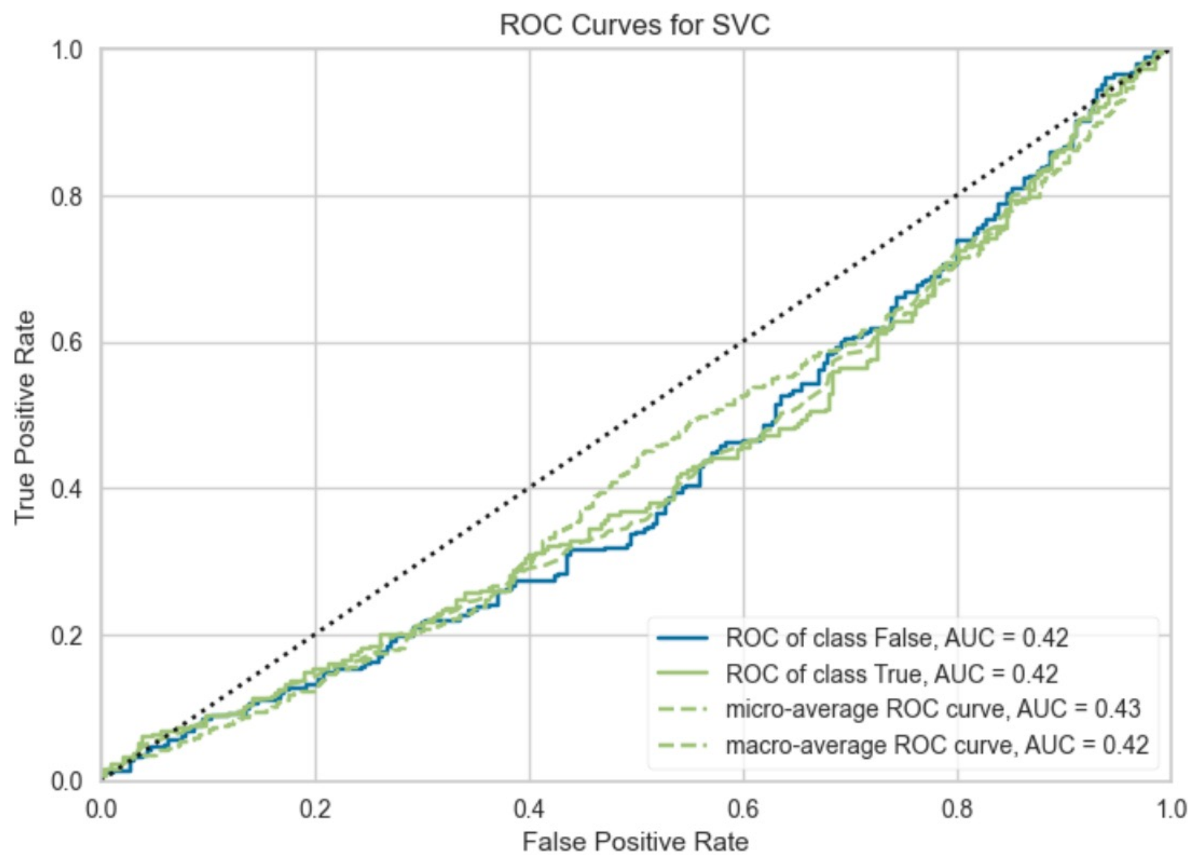
	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
lasso	Lasso Regression	0.0562	0.0241	0.1247	-0.0008	0.0938	1.0097	0.0200
en	Elastic Net	0.0562	0.0241	0.1247	-0.0008	0.0938	1.0097	0.0050
dummy	Dummy Regressor	0.0562	0.0241	0.1247	-0.0008	0.0938	1.0097	0.0150
llar	Lasso Least Angle Regression	0.0562	0.0241	0.1247	-0.0008	0.0938	1.0097	0.0150
omp	Orthogonal Matching Pursuit	0.0592	0.0231	0.1235	-0.0630	0.0827	1.1162	0.0050
svm	Support Vector Regression	0.0609	0.0239	0.1256	-0.1018	0.0910	1.5216	0.0150
ard	Automatic Relevance Determination	0.0636	0.0233	0.1262	-0.2665	0.0825	1.4999	0.0150
knn	K Neighbors Regressor	0.0830	0.0289	0.1476	-1.5191	0.0993	2.9235	0.0200
ada	AdaBoost Regressor	0.0903	0.0310	0.1533	-1.7654	0.0994	3.0131	0.0650
lightgbm	Light Gradient Boosting Machine	0.0962	0.0301	0.1557	-2.5832	0.0995	3.8432	0.2300
et	Extra Trees Regressor	0.1029	0.0298	0.1593	-3.7131	0.1034	4.7308	0.0700
gbr	Gradient Boosting Regressor	0.1108	0.0394	0.1799	-4.1980	0.1169	4.6750	0.2100
rf	Random Forest Regressor	0.1145	0.0341	0.1807	-9.3891	0.1261	7.3775	0.2000
xgboost	Extreme Gradient Boosting	0.1425	0.0458	0.2072	-11.2558	0.1381	8.4099	0.0800
huber	Huber Regressor	0.1464	0.0480	0.2191	-23.3891	0.1713	13.5419	0.0200
br	Bayesian Ridge	0.1925	0.0727	0.2643	-48.3633	0.1982	19.3542	0.0050
mlp	MLP Regressor	0.2132	0.0755	0.2701	-49.2790	0.1984	20.2355	0.0300
par	Passive Aggressive Regressor	0.4081	0.1931	0.4256	-50.4853	0.3002	27.5886	0.0150
dt	Decision Tree Regressor	0.2296	0.1473	0.3832	-64.4757	0.2506	17.3517	0.0250
ridge	Ridge Regression	0.2567	0.1286	0.3362	-102.3695	0.2455	27.9224	0.0050
kr	Kernel Ridge	0.2580	0.1299	0.3377	-103.6070	0.2464	28.0824	0.0200
tr	TheilSen Regressor	0.4049	0.3416	0.5072	-307.6756	0.3389	47.6177	0.3850
lr	Linear Regression	0.4356	0.3853	0.5342	-349.7519	0.3528	51.3597	0.0150
ransac	Random Sample Consensus	0.9306	1.6460	1.1561	-1423.3857	0.6067	103.9402	0.0250
lar	Least Angle Regression	1504065.4340	5955967809092.7285	1725806.6631	-5771051198958624.0000	9.7046	204101941.0188	0.0200

Result of falsely using K-Fold Cross-Validation





Residuals for Lasso Model
Train $R^2 = 0.000$
Test $R^2 = -0.035$
Residuals
Predicted Value
1e−5
Distribution
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−8.1
−8.0
−7.9
−7.8
−7.7
−7.6
−7.5
−7.4
0
1000


ROC Curves for SVC
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC of class False, AUC = 0.42
ROC of class True, AUC = 0.42
micro-average ROC curve, AUC = 0.43
macro-average ROC curve, AUC = 0.42

Potential Improvement

Future enhancements may include:

- Developing additional features, including:
  - Macroeconomic indicators.
  - Other financial instruments.
  - Further time series features.
- Incorporating more sophisticated models, such as:
  - Long Short-Term Memory (LSTM) networks.
  - Deep Neural Networks (DNNs).
- Refining the prediction target, for instance, by forecasting returns over longer time horizons.