task2_durei

January 20, 2025

1 Task2 by Durei

1.1 Yield curve modelling

[1]:
```
import datetime as dt

import numpy as np
import pandas as pd

from plotly import express as px
from plotly import graph_objects as go
from plotly.subplots import make_subplots
```

2.a: Pick government securities from a country. The country selected should be one of the countries from your group so that you can fit a Nelson-Siegel mode Load gilts table with clean price and yield of a day.

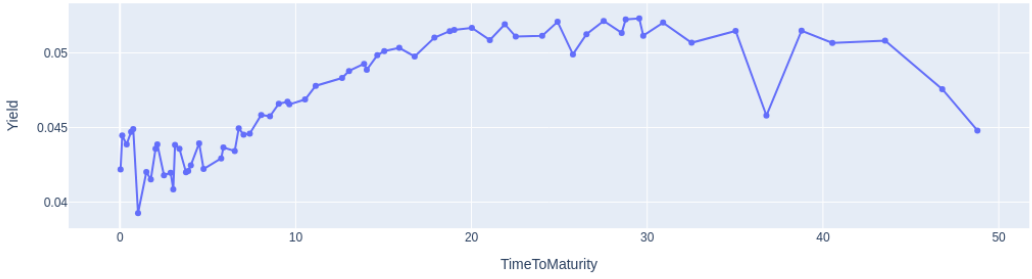
Data Source: dividenddata cross-checked with Trading Economics and Matket Watch's Gilt section.

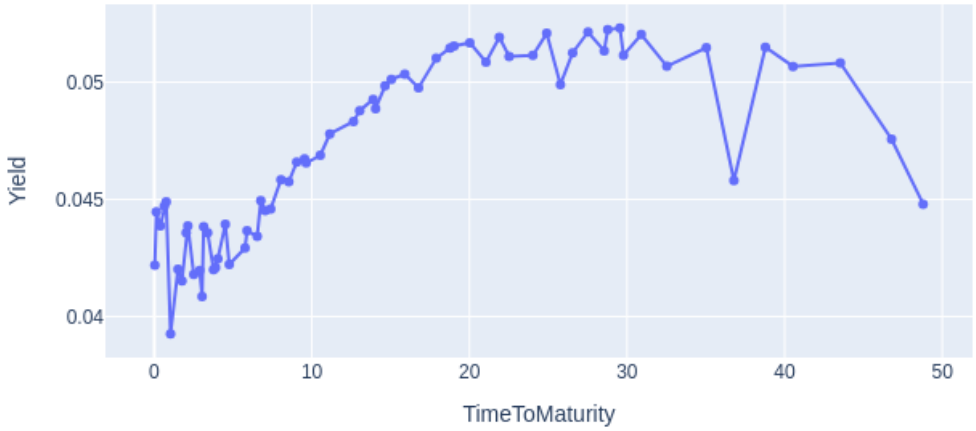
[2]:
```
gilts_df = pd.read_csv('task2_gilt_20250116.csv')

gilts_df['Maturity Date'] = pd.to_datetime(gilts_df['Maturity Date'])
gilts_df['Time To Maturity'] = (gilts_df['Maturity Date'] - pd.Timestamp(dt.
 ↪date.today())).dt.days / 365.
gilts_df['Yield (%)'] /= 100.0

ttm = 'TimeToMaturity'
prx = 'Price'
yld = 'Yield'
gilts_df = gilts_df.rename(
    columns={
        'Maturity Date': 'Maturity',
        'Time To Maturity': ttm,
        'Price (£)': prx,
        'Yield (%)': yld
    }
)
# gilts_df.iloc[10:15]
```

[3]:
```
px.line(data_frame=gilts_df, x=ttm, y='Yield', markers=True)
```



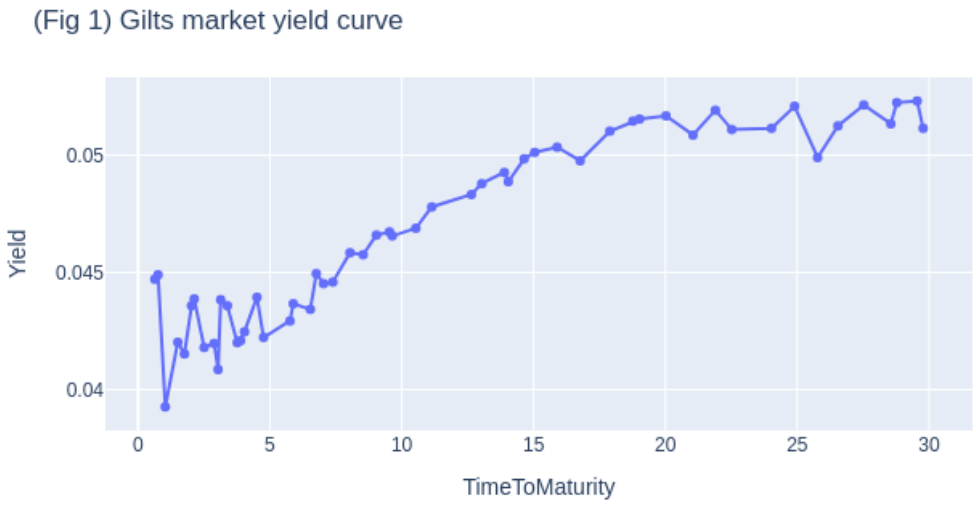


1.1.1 2.b: Be sure to pick maturities ranging from short-term to long-term (e.g. 6 month maturity to 20 or 30 year maturities).

We shorten up the list upto 30 years of time to maturity for convience, and the trades rarely occur beyond that hence the true price is less trustful anyway. On the front end, we also cut down gilts less then a half year to mature - under true market circumstance, near expiry bonds are subject to repo financing and base rates dynamics rather than curve hence are rather noises in our curve modeling purposes.

[4]:
```
t_min = 0.5
t_max = 30
gilts_df_trunc = gilts_df[(gilts_df[ttm] <= t_max) & (gilts_df[ttm] >= t_min)].
↪reset_index(drop=True)
# gilts_df_trunc.iloc[[0,1,2,-3,-2,-1]]
```

[5]:
```
px.line(data_frame=gilts_df_trunc, x=ttm, y=yld, markers=True).
↪update_layout(title='(Fig 1) Gilts market yield curve')
```



1.2 2.c Fit the Nelson-Siegel model curve

$$r_t \sim f(t) = b_0 + b_1 \cdot \frac{1 - \exp(-t/\lambda)}{t/\lambda} + b_2 \cdot \left(\frac{1 - \exp(-t/\lambda)}{t/\lambda} - \exp(-t/\lambda) \right)$$

[6]:
```
from typing import Dict, List
import scipy as sp
```

[7]:
```
gilts_input_df_NS = gilts_df_trunc.set_index(ttm)
# gilts_input_df.iloc[[0,1,2,-3,-2,-1]]
```

[8]:
```
def get_nelson_seigel_rate(tenor: float, parameters: Dict[str, float]) -> float:
    """ Calculate rate of the tenor via Nelson-Siegel model with given␣
↪parameters.

    r(tenor) = b0 + b1 * (1 - exp(-tenor/lambda))/(tenor/lambda) + b2 * ((1 -␣
↪exp(-tenor/lambda))/(tenor/lambda) - exp(-tenor/lambda))
    """
    b0 = parameters['b0']
    b1 = parameters['b1']
    b2 = parameters['b2']
    lamda = parameters['lamda']

    x = tenor/lamda
    decay: float = np.exp(-x)
    concaved: float = (1. - decay)/x
```

```
    r = b0 + b1 * concaved + b2 * (concaved - decay)
    return r.item()


def get_nelson_siegel_model_rates_over_tenors(tenors: List, parameters_arr: np.
 ↪ndarray):
    rates = []
    for tn in tenors:
        r_tn = get_nelson_seigel_rate(
            tenor = tn,
            parameters = dict(b0=parameters_arr[0], b1=parameters_arr[1],␣
 ↪b2=parameters_arr[2], lamda=parameters_arr[3])
        )
        rates.append(r_tn)

    return rates


def minimise_mse_gilts_vs_model(parameters_arr: np.ndarray) -> float:
    rates = get_nelson_siegel_model_rates_over_tenors(gilts_input_df_NS.index.
 ↪to_list(), parameters_arr)

    return np.sqrt(np.mean(np.square(gilts_input_df_NS[yld].values - np.
 ↪array(rates)).mean()))


example_parameters = dict(b0=0.04, b1= 0.015, b2= 0.01, lamda=3.)
print(f'Nelson-Siegel model yield at 10yr tenor with parameters:␣
 ↪{get_nelson_seigel_rate(tenor=10., parameters = example_parameters)}')
print(f'L2 SME difference between Market vs Model yields:␣
 ↪{minimise_mse_gilts_vs_model(np.array(list(example_parameters.values())))}')
```

```
Nelson-Siegel model yield at 10yr tenor with parameters: 0.04687570511642308
L2 SME difference between Market vs Model yields: 0.007611501574915662
```

[9]:
```
# Given that gilts_df[[ttm, yld]] is the table of time_to_maturity and yield,
# We can find the least-square fitted parameters of the nelson_siegel curve␣
 ↪model via scipy optimiser in sp.optimize.minimize().

opt_res = sp.optimize.minimize(
    fun = minimise_mse_gilts_vs_model,
    x0 = np.array([0.04, 0.015, 0.01, 3.0])
)
opt_res
```

[9]:
```
  message: Optimization terminated successfully.
  success: True
   status: 0
      fun: 0.0009179555967679618
        x: [ 5.510e-02 -9.459e-03 -2.679e-02  2.375e+00]
      nit: 26
      jac: [-3.099e-06 -1.083e-06 -5.160e-07  3.165e-09]
  hess_inv: [[ 1.712e-02 -2.835e-02  8.907e-03  6.076e+00]
             [-2.835e-02  1.061e-01 -1.590e-01 -1.742e+01]
             [ 8.907e-03 -1.590e-01  4.074e-01  2.544e+01]
             [ 6.076e+00 -1.742e+01  2.544e+01  3.736e+03]]
     nfev: 175
     njev: 35
```

[10]:
```
parameters_opt_arr = opt_res.x
parameters_opt_arr
```

[10]:
```
array([ 0.05509689, -0.00945863, -0.02679413,  2.37490955])
```

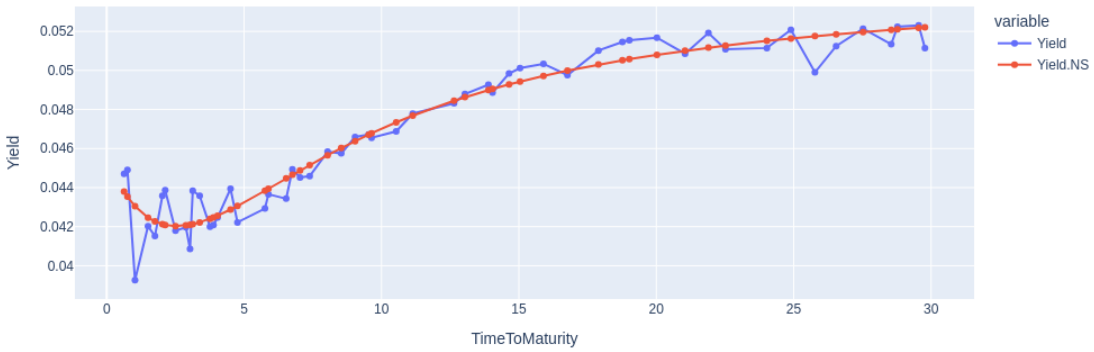
[11]:
```
yld_NS: List[float] =␣
 ↪get_nelson_siegel_model_rates_over_tenors(gilts_input_df_NS.index.to_list(),␣
 ↪parameters_opt_arr)
yld_Market_vs_NS = pd.DataFrame({yld: gilts_input_df_NS[yld].to_list(), f'{yld}.
 ↪NS': yld_NS}, index=gilts_input_df_NS.index)
yld_Market_vs_NS.iloc[[0,1,2,-3,-2,-1]]
```

[11]:

TimeToMaturity	Yield	Yield.NS
0.632877	0.04471	0.043799
0.756164	0.04490	0.043537
1.030137	0.03926	0.043048
28.775342	0.05224	0.052105
29.547945	0.05231	0.052183
29.775342	0.05115	0.052205

[12]:
```
px.line(data_frame=yld_Market_vs_NS, markers=True, width=1000, height=400).
 ↪update_layout(
    yaxis_title=yld,
    title='(Fig 2) Yield of Gilts: Market vs Nelson-Siegel'
)
```

(Fig 2) Yield of Gilts: Market vs Nelson-Siegel



1.3 2.d Fit the Cubic-Spline model curve

Cubic spline curve is a piece-wise linear regression model with polynomial terms of tenors upto third order.

Concretely, the curve is an ensemble of the **linear regressors** each of which estimates neighbouring data points as a group of dataset with **boundary conditions** of continuity of zero-th, first, and second order derivatives equality between neightbouring regression models.

Following the deifinition of the cubic spline model given in M1-L3 note of FE600 course of our programme, the general formula of the curve with N pieces will be stated as below:

$r_t \sim f_i(t) = w_{3,i} \cdot t^3 + w_{2,i} \cdot t^2 + w_{1,i} \cdot t + b_i \quad$ where $t \in \{T_i, \cdots, T_{i+1}\}$ and $i \in [1, N] \in \mathbb{N}^1$

requires 4N equations for 4N parameters defined as:

$f_i(T_i) = r_{T_i},\ f_i(T_{i+1}) = r_{T_{i+1}}$

to anchor the regressors match with end points of pieces: gives 2N equations

$$\frac{df_i(t = T_{i+1})}{dt} = \frac{df_{i+1}(t = T_{i+1})}{dt}$$

due to the 1st order derviative continuity condition: gives N-1 equations

$$\frac{d^2 f_i(t = T_{i+1})}{dt^2} = \frac{d^2 f_{i+1}(t = T_{i+1})}{dt^2}$$

due to the 2nd order derviative continuity condition: gives N-1 equations

$$\frac{d^2 f_1(t = T_1)}{dt^2} = 0, \frac{d^2 f_N(t = T_{N+1})}{dt^2} = 0$$

from the 0 second order derivatives at the left and right ends of the entire dataset: gives 2 equations

In matrix form, it becomes a multiplication form :

$$\underline{\underline{X}} \cdot \underline{\theta}^T = \underline{r}^T$$

, conceretly

$$\underline{\theta_i} = [w_{3,i},\ w_{2,i},\ w_{1,i},\ b_i], \tag{1}$$

$$\underline{x_i} = [T_i^3, T_i^2, T_i, 1] \tag{2}$$

$$\underline{x_i^{(1)}} = [3T_i^2, 2T_i, 1, 0] \tag{3}$$

$$\underline{x_i^{(2)}} = [6T_i, 2, 0, 0] \tag{4}$$

$$\left[\begin{array}{ccccc}
\underline{x_1} & \underline{0} & \cdots & \underline{0} & \\
\underline{x_2} & \underline{0} & \cdots & \underline{0} & \\
\vdots & \vdots & \ddots & \vdots & \\
\underline{0} & \cdots & \underline{0} & \underline{x_N} & \\
\underline{0} & \cdots & \underline{0} & \underline{x_{N+1}} & \\
\hline
\underline{x_1^{(1)}} & \underline{-x_2^{(1)}} & \underline{0} & \cdots & \underline{0} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\underline{0} & \cdots & \underline{0} & \underline{x_{N-1}^{(1)}} & \underline{-x_N^{(1)}} \\
\hline
\underline{x_1^{(2)}} & \underline{-x_2^{(2)}} & \underline{0} & \cdots & \underline{0} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\underline{0} & \cdots & \underline{0} & \underline{x_{N-1}^{(2)}} & \underline{-x_N^{(2)}} \\
\hline
\underline{x_1^{(2)}} & \underline{0} & \cdots & \underline{0} & \\
\underline{0} & \cdots & \underline{0} & \underline{x_{N+1}^{(2)}} &
\end{array}\right]
\bullet
\begin{bmatrix} \underline{\theta_1}^T \\ \vdots \\ \underline{\theta_N}^T \end{bmatrix}
=
\left[\begin{array}{c}
r_{T_1} \\ r_{T_2} \\ \vdots \\ r_{T_N} \\ r_{T_{N+1}} \\
\hline
0 \\ \vdots \\ 0 \\
\hline
0 \\ \vdots \\ 0 \\ \vdots \\
\hline
0 \\ 0
\end{array}\right] \tag{5}$$

where zero vectors here are row vectors with some zeros to make the matrix rectangular.

[13]:
```
gilts_input_df_CS = gilts_df_trunc.copy()
```

[14]:
```
interval_boundaries = [0.5, 2, 5, 10, 30]

interval_left_ends = [gilts_df_trunc[gilts_df_trunc[ttm]>=t_left][ttm].min()␣
 ↪for t_left in interval_boundaries[:-1]]
interval_right_ends = [gilts_df_trunc[gilts_df_trunc[ttm]<=t_right][ttm].max()␣
 ↪for t_right in interval_boundaries[1:]]

# Create a list of pieces left and right end pair as list - right end except␣
 ↪the last piece copies from the next left end value
t_intervals = [[t_l, t_r] for t_l, t_r in zip(interval_left_ends,␣
 ↪interval_right_ends)] # knots
for intv_ind in range(len(t_intervals)-1):
    t_intervals[intv_ind][1] = t_intervals[intv_ind+1][0]
N = len(t_intervals)
t_intervals
```

[14]:
```
[[np.float64(0.6328767123287671), np.float64(2.0273972602739727)],
 [np.float64(2.0273972602739727), np.float64(5.758904109589041)],
 [np.float64(5.758904109589041), np.float64(10.534246575342467)],
```

```
[np.float64(10.534246575342467), np.float64(29.775342465753425)]]
```

[15]:
```
# gilts_df
```

[16]:
```
import itertools

def get_r_vector(tenors_intervals, df_TtmYld):
    return list(itertools.chain(*[[
            df_TtmYld[df_TtmYld[ttm]==t_l][yld].values[0],
            df_TtmYld[df_TtmYld[ttm]==t_r][yld].values[0]
    ] for [t_l, t_r] in tenors_intervals]))

r_list = get_r_vector(t_intervals, gilts_input_df_CS[[ttm,yld]])
r_arr = np.array(r_list + [0 for _ in range((N-1)+(N-1)+2)])
r_arr
```

[16]:
```
array([0.04471, 0.04358, 0.04358, 0.04293, 0.04293, 0.04688, 0.04688,
       0.05115, 0.     , 0.     , 0.     , 0.     , 0.     , 0.     ,
       0.     , 0.     ])
```

[17]:
```
def get_x_matrix(tenors_intervals):
    rows_list = []

    n_intervals = len(tenors_intervals)
    n_columns = n_intervals * 4 # as number of parameters per regression is 4␣
 ↪for linear regressor with cubic features

    # left and right end regressor values to match with ylds
    for i in range(n_intervals):
        t_l, t_r = tenors_intervals[i]

        zeros_i_left = [0 for _ in range(4 * i)]
        zeros_i_right = [0 for _ in range(n_columns - len(zeros_i_left) - 4)]

        x_i = [t_l ** 3, t_l ** 2, t_l, 1]
        row_i = zeros_i_left + x_i + zeros_i_right
        rows_list.append(row_i)

        x_ipp = [t_r ** 3, t_r ** 2, t_r, 1]

        row_ipp = zeros_i_left + x_ipp + zeros_i_right
        rows_list.append(row_ipp)

    # first order derivatives continuity between regressors
    for j in range(len(tenors_intervals)-1):
        _, t_r = tenors_intervals[j] # T_j, T_{j+1]
```

```
        zeros_j_left = [0 for _ in range(4 * j)]
        zeros_j_right = [0 for _ in range(n_columns - len(zeros_j_left) - 4*2)]

        x_j_r_1st = [3 * t_r **2, 2 * t_r, 1, 0]
        minus_x_j_r_1st = [-v for v in x_j_r_1st]

        row_j = zeros_j_left + x_j_r_1st + minus_x_j_r_1st + zeros_j_right
        rows_list.append(row_j)

    # second order derivatives continuity between regressors
    for k in range(len(tenors_intervals)-1):
        t_l, t_r = tenors_intervals[k]

        zeros_k_left = [0 for _ in range(4 * k)]
        zeros_k_right = [0 for _ in range(n_columns - len(zeros_k_left) - 4*2)]

        x_k_r_2nd = [6 * t_r, 2, 0, 0]
        minus_x_k_r_2nd = [-v for v in x_k_r_2nd]

        row_k = zeros_k_left + x_k_r_2nd + minus_x_k_r_2nd + zeros_k_right
        rows_list.append(row_k)

    # zero second order derivatives assumed at both end of dataset
    zeros_end = [0 for _ in range(n_columns - (4*1))]

    t_l_left, _ = tenors_intervals[0]
    x_2nd_left_end = [6 * t_l_left, 2, 0, 0]
    rows_list.append(x_2nd_left_end + zeros_end)

    _, t_r_right = tenors_intervals[-1]
    x_2nd_right_end = [6 * t_r_right, 2, 0, 0]
    rows_list.append(zeros_end + x_2nd_right_end)

    return rows_list

x_arr = np.array(get_x_matrix(t_intervals))
# x_mat
```

[18]:
```
# get parameters as array of piece (row) by parameters (col)
theta_arr = (np.linalg.inv(x_arr) @ r_arr).reshape(N,4)
theta_arr
```

[18]:
```
array([[ 3.02240745e-05, -5.73843388e-05, -8.32773512e-04,
         4.52523658e-02],
       [ 3.35686873e-06,  1.06027160e-04, -1.16407354e-03,
         4.54762581e-02],
       [-1.52262440e-05,  4.27082252e-04, -3.01299903e-03,
```

```
        4.90255196e-02],
       [ 9.37382042e-07, -8.37326140e-05,  2.36805072e-03,
        3.01304180e-02]])
```

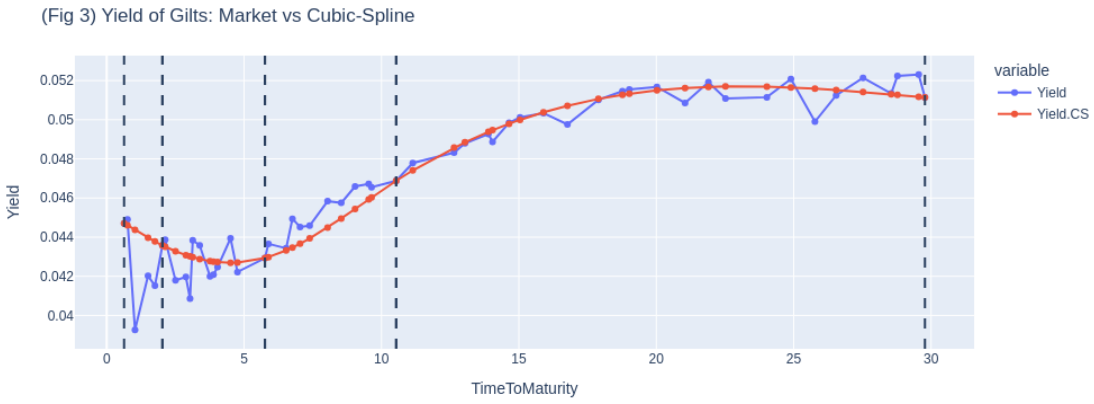
[19]:
```
def get_piece_index(tenor, tenors_intervals):
    for i, (t_l, t_r) in enumerate(tenors_intervals):
        if (tenor - t_l) * (tenor - t_r) <= 0:
            return i

gilts_input_df_CS['Block'] = gilts_input_df_CS[ttm].apply(lambda tenor:␣
 ↪get_piece_index(tenor, t_intervals))
```

[20]:
```
def get_cubic_spline_yield(tenor, parameters_arr):
    return np.array([tenor **3, tenor ** 2, tenor, 1]) @ parameters_arr

gilts_input_df_CS[f'{yld}.CS'] = gilts_input_df_CS.apply(lambda row:␣
 ↪get_cubic_spline_yield(row[ttm], theta_arr[row['Block']]), axis=1)
```

[21]:
```
yld_Market_vs_CS = gilts_input_df_CS[[ttm, yld, f'{yld}.CS']].set_index(ttm)
fig = px.line(data_frame=yld_Market_vs_CS, markers=True, width=1000,␣
 ↪height=400).update_layout(
    yaxis_title=yld,
    title='(Fig 3) Yield of Gilts: Market vs Cubic-Spline'
)
for tn in list(itertools.chain(*t_intervals)):
    fig.add_vline(x=tn, line_dash='dash')
fig.show()
```



1.4 2.e Compare the models in terms of fit and interpretation

Comparing the yield curves of Nelson-Siegel model and the cubic spline model shown in Fig 2 and 3 leaves a few observation points:

- Distribution of model vs market yield differences: the Nelson-Siegel model curve penetrates

middle through the market yield points over the time to maturity hence the market versus model yield divergence centers around zero, on the other hand the Cubic Spline curve does *not* penetrates through mid for some slices of time-to-maturity. This is due to the difference in model fittings applied to the models where the Nelson-Sigel model is fitted to minimise Mean Square Error s.t the model vs data yield gaps tends to centres around. On the other hand, the Cubic Spline model only cares about the boundary conditions that do not concern about difference between the model vs market gap for the entire datapoints except knots.

[22]:
```
fig = px.line(data_frame=yld_Market_vs_NS[yld_Market_vs_NS.index<=3.0],␣
 ↪markers=True, width=1000, height=400).update_layout(
    yaxis_title=yld,
    title='Yield of Gilts: Market vs Nelson-Sigel for short end of tenors'
)
fig.show()
```

[23]:
```
fig = px.line(data_frame=yld_Market_vs_CS[yld_Market_vs_CS.index<=3.0],␣
 ↪markers=True, width=1000, height=400).update_layout(
    yaxis_title=yld,
    title='Yield of Gilts: Market vs Cubic-Spline for short end tenors'
)
fig.show()
```

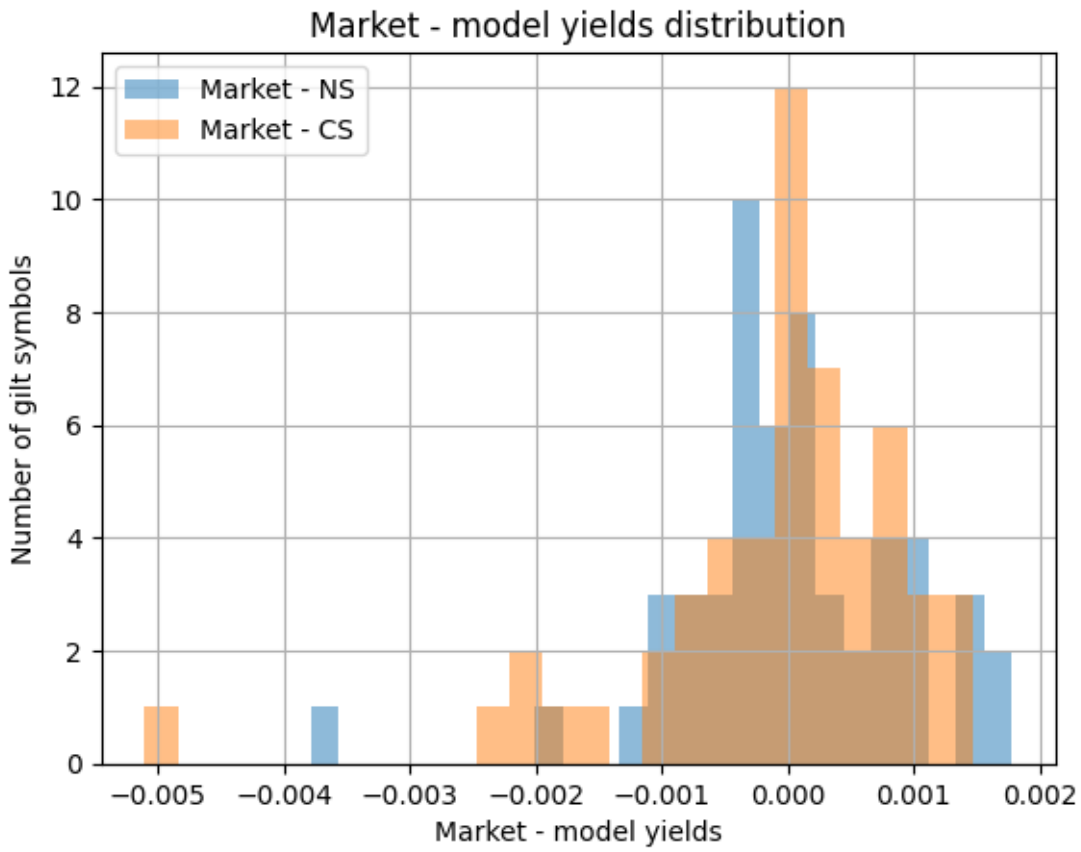
- Note, the Cubic-Spline model still provides reasonable fit with the differences around zero on the entire domain (of data points) as the formula captures continuities at knot points for upto 2nd order differentials.

[24]:
```
ax_hist = (yld_Market_vs_NS[yld] - yld_Market_vs_NS[f'{yld}.NS']).hist(bins=25,␣
 ↪alpha=0.5)
(yld_Market_vs_CS[yld] - yld_Market_vs_CS[f'{yld}.CS']).hist(bins=25, ax =␣
 ↪ax_hist, alpha=0.5)

ax_hist.set_ylabel('Number of gilt symbols')
ax_hist.set_xlabel('Market - model yields')
ax_hist.set_title('Market - model yields distribution')

ax_hist.legend(['Market - NS', 'Market - CS'])
```

[24]:
```
<matplotlib.legend.Legend at 0x7ecaf5d030d0>
```



- Equality of Market and model yield at *knots* or any points: the Nielson-Sigel is model with single function with single set of parameters that rules the entire domain with minimal MSE, hence the model does not usually provide the exact match of its estimate versus data points in any datapoints. In contrast, we define the fitting rule for the Cubic-Spline which includes the *exact* match between the observed yield and the model's estimate at knots - please refer to Fig 3 for this behavior at knots on the vertical lines.
- Yet both models offer one important property that the curves are smoothly continuous as well as guaranteed to be differentiable upto second order (infinite order for the Nelson-Sigel). Fig 2 and 3 show the visual aspect of this property. This is a useful point to calculate theoretical rate at tenor with no observable points, as well as delta risk.

1.5 2.f Model parameters specification

Below are the display of parameters for each model - Nelson-Siegl and Cubic-Spline. Please refer to section 2.c and 2.d for the mathematical formula of the models and corresponding parameters.

[25]:
```
parameters_NS = pd.DataFrame({'value': parameters_opt_arr},␣
↪index=['b0','b1','b2','lambda']).T
parameters_NS
```

[25]:
```
              b0        b1        b2   lambda
value   0.055097 -0.009459 -0.026794  2.37491
```

[26]:
```
parameters_CS = pd.DataFrame(theta_arr.tolist())
parameters_CS.index = pd.Index([(t_l, t_r) for t_l, t_r in t_intervals],␣
 ↪name=('tenor_left', 'tenor_right'))
parameters_CS.columns = ['w3', 'w2', 'w1', 'b']
parameters_CS
```

[26]:

tenor_left	tenor_right	w3	w2	w1	b
0.632877	2.027397	3.022407e-05	-0.000057	-0.000833	0.045252
2.027397	5.758904	3.356869e-06	0.000106	-0.001164	0.045476
5.758904	10.534247	-1.522624e-05	0.000427	-0.003013	0.049026
10.534247	29.775342	9.373820e-07	-0.000084	0.002368	0.030130

1.6 2.g About the ethics for smoothing data for the Nelson-Siegel curve model and practice

First of all, the ethics of engineering practice shall be assessed in aspect of its purpose of use and groundness of approach in perspective of each body of interest.

The use of the Nelson-Siegel model for yield curve smoothing in this practice was to calibrate smooth and differentiable yield curve with one analytical function with a parameters set throughout tenors for Gilts.

In this regard, take is not inherently unethical, provided it is used appropriately and transparently. Nelson-Siegel model is a widely recognized and accepted as a well grounded approach with a stack of research papers and industry tech reports due to its usefulness on understanding the term structure of interest rates where the model structure navigates through the good mid value among neighbouring market data points without any localised biases in tenor.

However, ethical concerns arise if the results of the smoothing are misrepresented or used inappropriately. - For example, presenting the smoothed curve as raw market data could mislead stakeholders about the actual volatility or risks. - Another example of concern is that the model ignores is the probabilistic statement of uncertainty of model against the real market data points as it does not provide confidence bounds that mark on the model vs market deviation based uncertainty. These aspect may be categorised under the “Smoothing Data” section in the lecture note. However, these do not align with the concerns highlighted in the “Smoothing Data” section, where the example issues are with biased summary of output datas such as underreporting volatility which is identified as unethical in purpose it artificially improves performance metrics like Sharpe ratios to report to the summary consumers such as clients and business stakeholders. However, this practice clarified that the yield data points are fresh and prepared as clean yield convention with data quality cross checked with multiple public resources.

To ensure ethical use and delivery for the information consumer, transparency is critical. Model assumptions, limitations, and objectives must be clearly communicated. Stakeholders should understand that the smoothed curve is a model-based representation and not a direct reflection of market conditions. Furthermore, reporting both smoothed and raw data, where feasible, enhances

transparency and prevents misrepresentation. The Nelson-Siegel curve model based service on yield term structure complies all aspects of reliability and methodological transparency and disclosure of purpose as these are all on public and examined over decades.

[ ]: