Finn 6211 - Final Project

Davis Vaughan

2018-04-18

Contents

Chapter 1

Introduction

This is the final project of the Finn 6211 class, with the intention of getting comfortable with spot rate data, their relation to yield curve factors, and various hedging strategies.

The instructions for the report can be found here.

An R package has been created to accompany the report. It contains a number of helper functions for cleaning data, manipulating the time series, and creating the hedging strategies. The package is named `ratekit` and can be found on Github here.

The book is intended to provide a summary of the methods used in the project. In each section of the book is a link to the script that was actually run to generate the results for the project. Those scripts are more in depth and cover every aspect of the project.

This report was written with bookdown, a book authoring package for R.

Chapter 2

Data

2.1 Getting the data

Script) 01-download.R

The data is retrieved from the Federal Reserve website, under the discussion series: *The U.S. Treasury Yield Curve: 1961 to the Present*. The link for that site is here. The specific data set that was downloaded was the XLS file included on that site.

The data was immediately opened in Excel, and was resaved as an `xlsx` file. The format of the data is not a true `xls` file, instead it is some kind of `xml` file. This does not play nicely with R's packages for importing Excel data, so a resave was necessary and is done manually.

`ratekit` provides the `download_rates_xls()` helper function for this.

2.2 Cleaning

Script) 02-cleaning.R

Data is brought in using the `readxl` package and the `ratekit` helper, `read_rates()`. This function reads the rectangle of rates data only, and sets any `-999.99` values to `NA`. These are often found through the dataset, and I assume they are meant to represent missing values. To visualize the `NA` values in the dataset, I use the `visdat` package.

First, let's look at what is immediately brought in by `read_rates()`. We will need a few packages throughout the chapter, so let's load those now as well.

```
library(visdat)
library(ratekit)
library(dplyr)
library(tibbletime)

raw <- read_rates("data/raw/feds200628.xlsx")

raw
```

```
## # A tibble: 14,163 x 100
##   date       SVENY01 SVENY02 SVENY03 SVENY04 SVENY05 SVENY06 SVENY07
##   <date>       <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 2018-03-29    2.10    2.27    2.40    2.49    2.56    2.62    2.66
```

```
## 2 2018-03-28    2.10    2.28    2.41    2.51    2.59    2.65    2.69
## 3 2018-03-27    2.09    2.26    2.40    2.50    2.58    2.64    2.69
## 4 2018-03-26    2.10    2.30    2.45    2.57    2.65    2.71    2.76
## 5 2018-03-23    2.09    2.27    2.42    2.53    2.61    2.68    2.73
## # ... with 1.416e+04 more rows, and 92 more variables: SVENY08 <dbl>,
## #   SVENY09 <dbl>, SVENY10 <dbl>, SVENY11 <dbl>, SVENY12 <dbl>,
## #   SVENY13 <dbl>, SVENY14 <dbl>, SVENY15 <dbl>, SVENY16 <dbl>,
## #   SVENY17 <dbl>, SVENY18 <dbl>, SVENY19 <dbl>, SVENY20 <dbl>,
## #   SVENY21 <dbl>, SVENY22 <dbl>, SVENY23 <dbl>, SVENY24 <dbl>,
## #   SVENY25 <dbl>, SVENY26 <dbl>, SVENY27 <dbl>, SVENY28 <dbl>,
## #   SVENY29 <dbl>, SVENY30 <dbl>, SVENPY01 <dbl>, SVENPY02 <dbl>,
## #   SVENPY03 <dbl>, SVENPY04 <dbl>, SVENPY05 <dbl>, SVENPY06 <dbl>,
## #   SVENPY07 <dbl>, SVENPY08 <dbl>, SVENPY09 <dbl>, SVENPY10 <dbl>,
## #   SVENPY11 <dbl>, SVENPY12 <dbl>, SVENPY13 <dbl>, SVENPY14 <dbl>,
## #   SVENPY15 <dbl>, SVENPY16 <dbl>, SVENPY17 <dbl>, SVENPY18 <dbl>,
## #   SVENPY19 <dbl>, SVENPY20 <dbl>, SVENPY21 <dbl>, SVENPY22 <dbl>,
## #   SVENPY23 <dbl>, SVENPY24 <dbl>, SVENPY25 <dbl>, SVENPY26 <dbl>,
## #   SVENPY27 <dbl>, SVENPY28 <dbl>, SVENPY29 <dbl>, SVENPY30 <dbl>,
## #   SVENF01 <dbl>, SVENF02 <dbl>, SVENF03 <dbl>, SVENF04 <dbl>,
## #   SVENF05 <dbl>, SVENF06 <dbl>, SVENF07 <dbl>, SVENF08 <dbl>,
## #   SVENF09 <dbl>, SVENF10 <dbl>, SVENF11 <dbl>, SVENF12 <dbl>,
## #   SVENF13 <dbl>, SVENF14 <dbl>, SVENF15 <dbl>, SVENF16 <dbl>,
## #   SVENF17 <dbl>, SVENF18 <dbl>, SVENF19 <dbl>, SVENF20 <dbl>,
## #   SVENF21 <dbl>, SVENF22 <dbl>, SVENF23 <dbl>, SVENF24 <dbl>,
## #   SVENF25 <dbl>, SVENF26 <dbl>, SVENF27 <dbl>, SVENF28 <dbl>,
## #   SVENF29 <dbl>, SVENF30 <dbl>, SVEN1F01 <dbl>, SVEN1F04 <dbl>,
## #   SVEN1F09 <dbl>, BETA0 <dbl>, BETA1 <dbl>, BETA2 <dbl>, BETA3 <dbl>,
## #   TAU1 <dbl>, TAU2 <dbl>
```

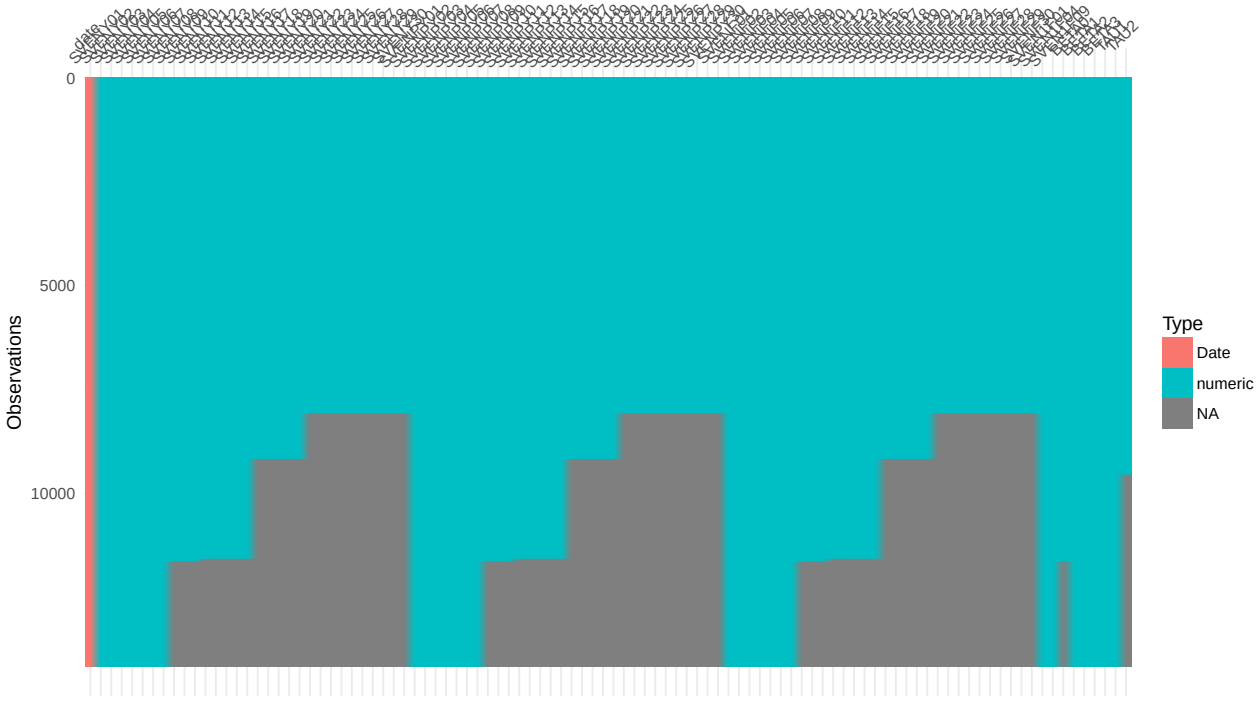
Not a bad start, but I'm worried about missing values. Also, what are those column names?

The column names in the data correspond to different types and lengths of rates used in the paper. The key for understanding the column names is below:

series	compounding_convention	key
Zero-coupon yield	Continuously Compounded	SVENYXX
Par yield	Coupon-Equivalent	SVENPYXX
Instantaneous forward rate	Continuously Compounded	SVENFXX
One-year forward rate	Coupon-Equivalent	SVEN1FXX
Parameters	NA	BETA0 to TAU2

Using `vis_dat()`, we can take a look at our dataset all at once to determine which data points to exclude.

```
vis_dat(raw, warn_large_data = FALSE)
```



A clear pattern is seen in the missing values, with the number of missing values increasing as you go further back in time and look at longer rates (10 year VS 30 year). This might be a bit difficult to see if you look at everything, but becomes clearer if you zoom in on just one set of series.

The parameters are affected by this as well, but not as much, with only `TAU2` being affected.

Since the parameters are all we care about for this project, I decided to throw out any row with an `NA` value for `TAU2`. This threw out every data point before 1980. We can ensure that we don't have any missing values now with `vis_miss()`.

```
# This is the cleaned parameter set, cleaned using 02-cleaning.R
parameters <- readRDS("data/cleaned/parameters/parameters.rds")
vis_miss(parameters)
```

2.3 Monthly and Ascending

Script) 03-to-monthly-and-ascending.R

At this point, our dataset looks like this:

```
parameters <- readRDS("data/cleaned/parameters/parameters.rds")
parameters
```

```
## # A tibble: 9,543 x 7
##   date       BETA0 BETA1      BETA2 BETA3  TAU1  TAU2
##   <date>     <dbl> <dbl>      <dbl> <dbl> <dbl> <dbl>
## 1 2018-03-29  4.13 -2.25  0.000228  -3.07  2.84  11.4
## 2 2018-03-28  4.35 -2.48 -0.000089  -3.53  2.99  12.2
## 3 2018-03-27  4.44 -2.59  0.000314  -3.67  3.18  12.5
## 4 2018-03-26  4.27 -2.44 -0.0000588 -3.12  2.73  11.8
## 5 2018-03-23  4.53 -2.69  0.000245  -3.78  3.17  12.9
## # ... with 9,538 more rows
```

We want monthly data, and we will need to put it in ascending order. We can convert to monthly with `as_period()` from `tibbletime`, and arrange it by ascending date with `arrange()` from `dplyr`.

```
parameters_monthly <- parameters %>%
  as_tbl_time(date) %>%
  arrange(date) %>%
  as_period("monthly", side = "end")

parameters_monthly
```

```
## # A time tibble: 459 x 7
## # Index: date
##   date       BETA0  BETA1 BETA2 BETA3  TAU1  TAU2
##   <date>     <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1980-01-31  11.8  0.979 -622.  617.  2.50  2.50
## 2 1980-02-29  11.8  1.53  -617.  621.  1.15  1.15
## 3 1980-03-31  13.2  2.95  -622.  617.  1.78  1.76
## 4 1980-04-30  11.1  0.607 -621.  617.  1.59  1.59
## 5 1980-05-30  11.3 -3.44  -620.  618.  1.36  1.35
## # ... with 454 more rows
```

This leaves us with 459 rows of data for our project, spanning 1980-01-31 to 2018-03-29.

Chapter 3

Fixed Income Features Calculations

In this chapter, I will construct spot rates, zero coupon bond prices, excess returns, and yield factors. These features will be used later in hedging strategies and for general exploration of the data.

The following packages are required in this chapter.

```
library(readr)
library(ratekit)
library(dplyr)
```

3.1 Spot Rates

Script) 04-spot-rates-and-prices.R

Spot rates series can be constructed from the 6 parameters in the rates dataset. The following formula is used to construct the spot rates. It is integrated form of the Svensson extension of the Nelson and Siegal approach to calculating instantaneous forward rates. Svensson added a second hump term to the model that Nelson and Siegal created. Integrating the instantaneous forward rates gives us the spot rates.

$$y_t(n) = \beta_{0,t} + \beta_{1,t}\frac{1-\exp(-\frac{n}{\tau_{1,t}})}{\frac{n}{\tau_{1,t}}}$$

$$+ \beta_{2,t}\left[\frac{1-\exp(-\frac{n}{\tau_{1,t}})}{\frac{n}{\tau_{1,t}}} - \exp(-\frac{n}{\tau_{1,t}})\right] +$$

To reconstruct this, I used a programming concept known as a *function factory*. This is a specialized function

that returns a function. This extends naturally to this use case because the outer function can accept the time series of the 6 parameters, and the inner function that get's returned is parameterized by `n`, corresponding to the n-year spot rate at time t. The `spot_rate_factory()` function lives in `ratekit`, and looks like this.

```
spot_rate_factory
```

```
## function(beta_0, beta_1, beta_2, beta_3, tau_1, tau_2) {
##
##   # Spot rate function based on Equation 22 of
##   # Gurkaynak, Sack and Wright (2006)
##   spot_rate_n <- function(n) {
##
##     spot_rate_percentage <-
##     beta_0 +
##     beta_1 *  (1 - exp( -n / tau_1)) / (n / tau_1) +
##     beta_2 * ((1 - exp( -n / tau_1)) / (n / tau_1) - exp( -n / tau_1)) +
##     beta_3 * ((1 - exp( -n / tau_2)) / (n / tau_2) - exp( -n / tau_2))
##
##     spot_rate_percentage / 100
##   }
##
##
##   spot_rate_n
## }
## <environment: namespace:ratekit>
```

As you can see, it accepts the 6 parameters, and returns a function parameterized by `n`. Because we want to calculate the spot rate for a number of different years, this parameterized function will be very useful. Below is an example usage of this concept.

```
# The monthly parameters from the Data chapter
parameters_monthly <- read_rds("data/cleaned/parameters/parameters_monthly.rds")

# The generated function. The function signature is generate_spot_rates(n)
generate_spot_rates <- with(
  data = parameters_monthly,
  expr = spot_rate_factory(BETA0, BETA1, BETA2, BETA3, TAU1, TAU2)
)

# Calculate the series of 1/12 year, 11/12 year, and 1 year spot rates
spot_rates <- parameters_monthly %>%
  select(date) %>%
  mutate(
    spot_1_month  = generate_spot_rates(1/12),
    spot_11_month = generate_spot_rates(11/12),
    spot_12_month = generate_spot_rates(1)
  )

spot_rates
```

```
## # A time tibble: 459 x 4
## # Index: date
##   date       spot_1_month spot_11_month spot_12_month
##   <date>            <dbl>         <dbl>         <dbl>
## 1 1980-01-31        0.126         0.118         0.118
## 2 1980-02-29        0.136         0.145         0.145
```

```
## 3 1980-03-31       0.161         0.152         0.151
## 4 1980-04-30       0.116         0.107         0.107
## 5 1980-05-30       0.0795        0.0867        0.0871
## # ... with 454 more rows
```

3.2 Zero Coupon Bond Prices

Script) 04-spot-rates-and-prices.R

n-year zero coupon bond prices can be calculated easily from their corresponding spot rates. Below is the relationship between the two.

$$P_t(n) = \exp[-y_t(n) \times n]$$

Like for the spot rates, a function factory was constructed that accepted the spot rate function, and returned a function that calculates a vector of bond prices parameterized by `n`.

```
zero_bond_price_factory
```

```
## function(spot_rate_fn) {
##
##   # Zero coupon bond for 1 dollar is just discounted spot rate
##   zero_bond_price_fn <- function(n) {
##     exp( - spot_rate_fn(n) * n)
##   }
##
##   zero_bond_price_fn
## }
## <environment: namespace:ratekit>
```

Using this relationship, the zero coupon bond prices were computed as the following:

```
generate_zero_prices <- zero_bond_price_factory(generate_spot_rates)

zero_prices <- spot_rates %>%
  transmute(
    date,
    zero_prices_1_month = generate_zero_prices(1/12),
    zero_prices_11_month = generate_zero_prices(11/12),
    zero_prices_12_month = generate_zero_prices(1)
  )

zero_prices
```

```
## # A time tibble: 459 x 4
## # Index: date
##   date       zero_prices_1_month zero_prices_11_month zero_prices_12_month
##   <date>                   <dbl>                <dbl>                <dbl>
## 1 1980-01-31               0.990                0.897                0.889
## 2 1980-02-29               0.989                0.875                0.865
## 3 1980-03-31               0.987                0.870                0.860
```

```
## 4 1980-04-30               0.990               0.906               0.899
## 5 1980-05-30               0.993               0.924               0.917
## # ... with 454 more rows
```

3.3 One Month Returns

Script) 05-returns.R

The time $t + \Delta$ return on a n-year bond is:

$$\text{RET}_{t+\Delta}(n) = \frac{P_{t+\Delta}(n - \Delta)}{P_t(n)} - 1$$

Using the zero bond prices from before, it is easy to calculate returns. For example, 1 month returns for 1 year zero coupon bonds can be calculated as:

```
returns <- zero_prices %>%
  mutate(
    zero_prices_12_month_lag = lag(zero_prices_12_month),
    one_month_return = zero_prices_11_month / zero_prices_12_month_lag - 1
  ) %>%
  select(-zero_prices_1_month, -zero_prices_12_month_lag)

returns
```

```
## # A time tibble: 459 x 4
## # Index: date
##   date       zero_prices_11_month zero_prices_12_month one_month_return
##   <date>                    <dbl>                <dbl>            <dbl>
## 1 1980-01-31                0.897                0.889         NA
## 2 1980-02-29                0.875                0.865         -0.0153
## 3 1980-03-31                0.870                0.860          0.00574
## 4 1980-04-30                0.906                0.899          0.0543
## 5 1980-05-30                0.924                0.917          0.0275
## # ... with 454 more rows
```

3.4 Excess Returns

Script) 05-returns.R

Excess returns are calculated over the 1 month treasury, specifically:

$$\text{ER}_{t+\Delta}(n) = \text{RET}_{t+\Delta}(n) - \text{RET}_{t+\Delta}(\Delta)$$

For excess returns, the one month on the n-year bond must be calculated, and the return on the benchmark (1 month treasury) must be calculated. We already have 1-year bond returns, but we need to calculate our benchmark returns. That can be done using the same formula as the 1-year returns, but where $P_{t+\Delta}(n-\Delta) = 1$, the maturity value.

```
returns_bench <- zero_prices %>%
  mutate(return_benchmark = 1 / lag(zero_prices_1_month) - 1) %>%
  select(date, zero_prices_1_month, return_benchmark)

returns_bench
```

```
## # A time tibble: 459 x 3
## # Index: date
##   date       zero_prices_1_month return_benchmark
##   <date>                   <dbl>            <dbl>
## 1 1980-01-31               0.990         NA
## 2 1980-02-29               0.989          0.0106
## 3 1980-03-31               0.987          0.0114
## 4 1980-04-30               0.990          0.0135
## 5 1980-05-30               0.993          0.00973
## # ... with 454 more rows
```

With these two sets of returns in hand, we can calculate excess returns for the one year bond.

```
excess_returns <- returns %>%
  left_join(returns_bench, "date") %>%
  transmute(date, excess_returns = one_month_return - return_benchmark)

excess_returns
```

```
## # A time tibble: 459 x 2
## # Index: date
##   date       excess_returns
##   <date>              <dbl>
## 1 1980-01-31       NA
## 2 1980-02-29       -0.0259
## 3 1980-03-31       -0.00568
## 4 1980-04-30        0.0408
## 5 1980-05-30        0.0177
## # ... with 454 more rows
```

3.5 Yield Curve Factors

Script) 06-yield-curve-factors.R

Finally, the yield curve factors, level, slope, and curvature are calculated as:

$$\text{Level}_t = y_t(1/4),$$

$$\text{Slope}_t = y_t(8) - y_t(1/4),$$

$$\text{Curvature}_t = [y_t(8) - y_t(2)] - [y_t(2) -$$

The implementation of these is straightforward from the set of spot rates, so no example is shown here.

Chapter 4

Question 1

Question:

What are the time-series properties of the spot rates $y_t(1)$, $y_t(5)$, and $y_t(10)$? Report their summary statistics, including mean, standard deviation, skewness, kurtosis, and the first four autocorrelation coefficients, and the correlation matrix of the spot rates. Comment on your results. Also plot them and comment on the time series patterns.

In this section, the following packages are used:

```
library(readr)
library(dplyr)
library(tidyr)
library(ratekit)
library(ggplot2)
library(broom)
library(purrr)
```

We will need the data for the $y_t(1)$, $y_t(5)$, and $y_t(10)$ spot rates. These have already been calculated in the script referenced in 3.1, so we can just load them in.

```
rates <- read_rds("data/computed/rates.rds")
n <- c("1", "5", "10")

spot_rates_q1 <- rates %>%
  filter(maturity_nm %in% n)

spot_rates_q1
```

```
## # A tibble: 1,377 x 4
##   maturity maturity_nm date       spot_rate
##      <dbl> <chr>       <date>         <dbl>
## 1        1 1           1980-01-31    0.118
## 2        1 1           1980-02-29    0.145
## 3        1 1           1980-03-31    0.151
## 4        1 1           1980-04-30    0.107
## 5        1 1           1980-05-30    0.0871
## # ... with 1,372 more rows
```

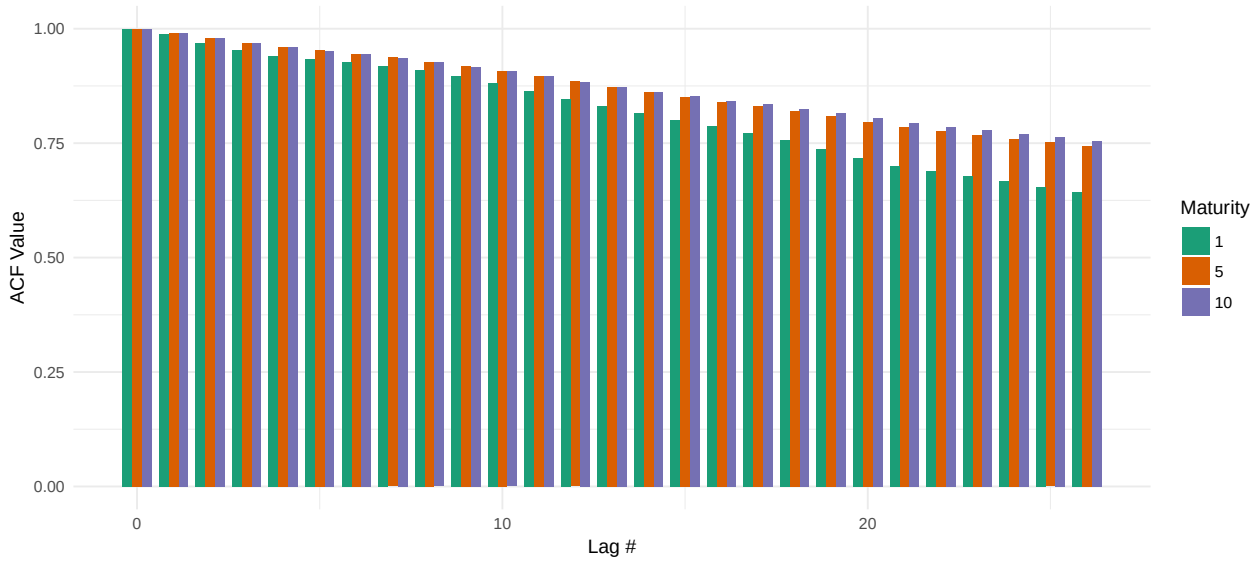


Figure 4.1: ACF for the 1, 5, and 10 year spot rates

4.1 Summary Statistics

Reported below are summary statistics on the three spot rate series.

Maturity	Mean	Standard Deviation	Kurtosis	Skewness
1	0.0480538	0.0375029	2.913858	0.6505396
5	0.0566592	0.0345430	2.567191	0.5659543
10	0.0627654	0.0314939	2.586681	0.5793584

Unsurprisingly, the average spot rate increases with the maturity, but interestingly, the shorter maturity spot rates have higher volatility. The kurtosis of all three are less than that of a normal distribution, but the 1 year maturity is very close. All three are right-skewed, with longer right tails, which makes sense considering extremely high interest rate periods do happen, but are rare.

4.2 Autocorrelations

The first four autocorrelation correlation coefficients of the 3 series are reported below, along with a plot of the ACF for the series. Each of the series are highly autocorrelated.

Maturity	Lag 1	Lag 2	Lag 3	Lag 4
1	0.9878979	0.9697433	0.9527499	0.9411647
5	0.9911143	0.9791933	0.9686252	0.9599084
10	0.9906205	0.9793246	0.9691974	0.9606667

By looking at the entire ACF, we can see that the amount of autocorrelation increases in maturity. At farther out lags, 1 is less autocorrelated than 5 and 5 less than 10.

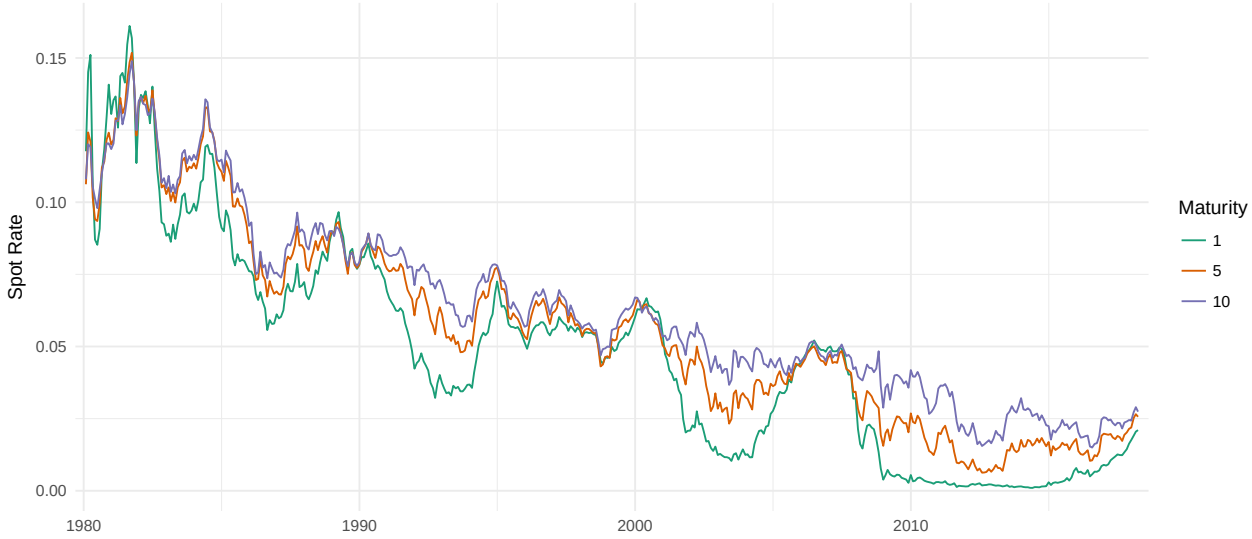


Figure 4.2: A look at the 1, 5, and 10 year spot rates over time

4.3 Correlations

Moving on to correlations, it is clear that the the series are *highly* correlated. This should not be surprising whatsoever. Intuitively, the 1 year is more correlated with the 5 year than with the 10 year.

	1	5	10
1	1.0000000	0.9783620	0.9539364
5	0.9783620	1.0000000	0.9932312
10	0.9539364	0.9932312	1.0000000

4.4 Time Series Visualizations

A look at the time series of the three series confirms the highly autocorrelated and correlated nature of the three series. 1 year spot rates are almost always below the longer maturity rates, as one would expect. Since 2010, 1 year spot rates have been incredibly low, but have started to pick back up in the last few years.

Chapter 5

Question 2

Question:

Can the three yield curve factors explain the time-series variation in spot rates? Regress $y_t(1)$ on a constant and X_t and comment on the regression statistics. Perform the same analysis for $y_t(5)$ and $y_t(10)$.

In this section, the following packages are used:

```
library(readr)
library(dplyr)
library(tidyr)
library(ratekit)
library(ggplot2)
library(broom)
library(purrr)
library(furrr)
library(rsample)
library(tibbletime)
library(forcats)
```

We will need the data for the $y_t(1)$, $y_t(5)$, and $y_t(10)$ spot rates along with the yield curve factors. These have already been calculated in the scripts referenced in 3.1 and 3.5 so we can just load them in.

```
rates    <- read_rds("data/computed/rates.rds")
yield_curve_factors <- read_rds("data/computed/yield_curve_factors.rds")
n <- c("1", "5", "10")

spot_rates_q2 <- rates

spot_rates_q2
```

```
## # A tibble: 6,426 x 4
##   maturity maturity_nm date       spot_rate
##      <dbl> <chr>       <date>         <dbl>
## 1   0.0833 1/12        1980-01-31    0.126
## 2   0.0833 1/12        1980-02-29    0.136
## 3   0.0833 1/12        1980-03-31    0.161
## 4   0.0833 1/12        1980-04-30    0.116
## 5   0.0833 1/12        1980-05-30    0.0795
## # ... with 6,421 more rows
```

5.1 Regression

Using the concept of multiple models from the book, `R 4 Data Science`, implementing these regressions in R is incredibly straightforward. First, we shard the series into separate data frames.

```
nested_spot <- spot_rates_q2 %>%
  # Add on the curve factors
  left_join(yield_curve_factors, by = "date") %>%

  # Group by maturity and nest
  group_by(maturity) %>%
  select(-maturity_nm) %>%
  nest()
```

Then we apply the linear model to each shard. The far right column, `model` contains the results of the linear models. I went ahead and calculated models for every maturity because I am going to use them in some exploration later.

```
nested_models <- nested_spot %>%
  mutate(model = map(data, ~ lm(spot_rate ~ level + slope + curvature, data = .x)))

nested_models
```

```
## # A tibble: 14 x 3
##    maturity data               model
##       <dbl> <list>             <list>
##  1   0.0833 <tibble [459 x 5]> <S3: lm>
##  2   0.25   <tibble [459 x 5]> <S3: lm>
##  3   0.917  <tibble [459 x 5]> <S3: lm>
##  4   1      <tibble [459 x 5]> <S3: lm>
##  5   2      <tibble [459 x 5]> <S3: lm>
##  6   2.92   <tibble [459 x 5]> <S3: lm>
##  7   3      <tibble [459 x 5]> <S3: lm>
##  8   4.92   <tibble [459 x 5]> <S3: lm>
##  9   5      <tibble [459 x 5]> <S3: lm>
## 10   6.92   <tibble [459 x 5]> <S3: lm>
## 11   7      <tibble [459 x 5]> <S3: lm>
## 12   8      <tibble [459 x 5]> <S3: lm>
## 13   9.92   <tibble [459 x 5]> <S3: lm>
## 14  10      <tibble [459 x 5]> <S3: lm>
```

5.2 One Year Spot Rate

As you can see below, all estimates for the 1 year spot rate model are highly significant, and the Adjusted R^2 is nearing 100%, suggesting that the model can explain essentially all of the variation in the spot rate.

Term	Estimate	Standard Error	Statistic	P-Value
(Intercept)	0.001014021	0.0001280165	7.921022	1.8e-14
level	0.989941953	0.0015226534	650.142657	0.0e+00
slope	0.264774017	0.0034323179	77.141460	0.0e+00
curvature	-0.428350107	0.0057618892	-74.341954	0.0e+00

R Squared	R Squared Adj	Residual Std Error
0.9995224	0.9995193	0.0008223

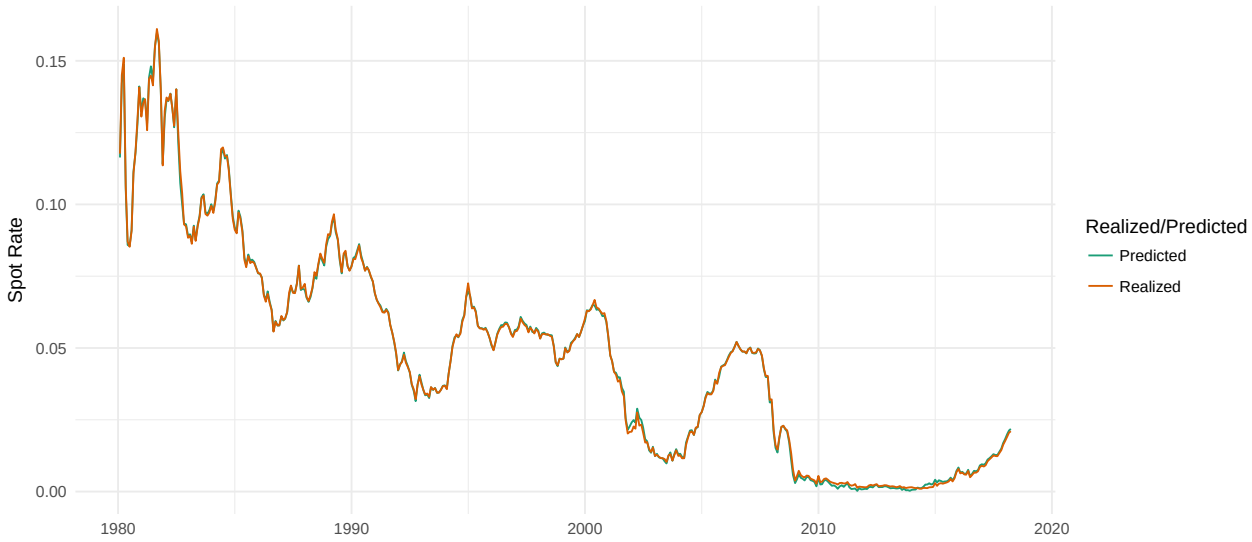


Figure 5.1: One year spot rate: In sample predictions VS realized

A chart of the realized VS predicted time series confirms how well the variation is explained. It is important to remember that this model is not predicting future rates, and is simply used to gather intuition about past rates.

```
plot_rates_vs_predictions <- function(.rates, .model, .maturity) {
  .rates %>%
    filter(maturity == .maturity) %>%
    rename(Realized = spot_rate) %>%
    mutate(Predicted = predict(.model)) %>%
    gather("Realized/Predicted", "Spot Rate", Realized, Predicted) %>%
    ggplot(aes(x = date, y = `Spot Rate`, color = `Realized/Predicted`)) +
    geom_line() +
    theme_minimal() +
    scale_color_brewer(palette = "Dark2") +
    labs(x = "")
}
```

5.3 Five Year Spot Rate

The model for the 5 year rate is similar to the 1 year rate in terms of explanatory power.

Term	Estimate	Standard Error	Statistic	P-Value
(Intercept)	-0.001326488	0.0001178948	-11.25146	0
level	1.010756975	0.0014022642	720.80351	0
slope	0.862622600	0.0031609403	272.90063	0
curvature	-0.237885069	0.0053063231	-44.83049	0

R Squared	R Squared Adj	Residual Std Error
0.9995226	0.9995194	0.0007572

Figure 5.2: Five year spot rate: In sample predictions VS realized

5.4 Ten Year Spot Rate

And again, the 10 year model performs well too.

Term	Estimate	Standard Error	Statistic	P-Value
(Intercept)	0.001285415	9.632731e-05	13.34424	0
level	0.990119282	1.145736e-03	864.17722	0
slope	1.041806029	2.582683e-03	403.38126	0
curvature	0.101703126	4.335593e-03	23.45772	0

R Squared	R Squared Adj	Residual Std Error
0.9996166	0.9996141	0.0006187

5.5 Decomposing the Spot Curve

Although not specifically asked for, I thought it might be interesting to decompose and plot the spot curve at a few particular points in time. We will need a few functions to do so. The functions essentially allow us to abstract away all the work so that we can just check out the spot rate graph at any date.

```
extract_rates <- function(.rates, .date) {
  .rates %>%
    as_tbl_time(date) %>%
    group_by(maturity) %>%
    filter_time(~.date)
}
```

```
tidy_models <- function(.nested_models, .date) {
  coefs <- .nested_models %>%
    mutate(coef = map(model, tidy)) %>%
    unnest(coef) %>%
    select(maturity, term, estimate) %>%
    spread(term, estimate) %>%
    select(-`(Intercept)`)
```

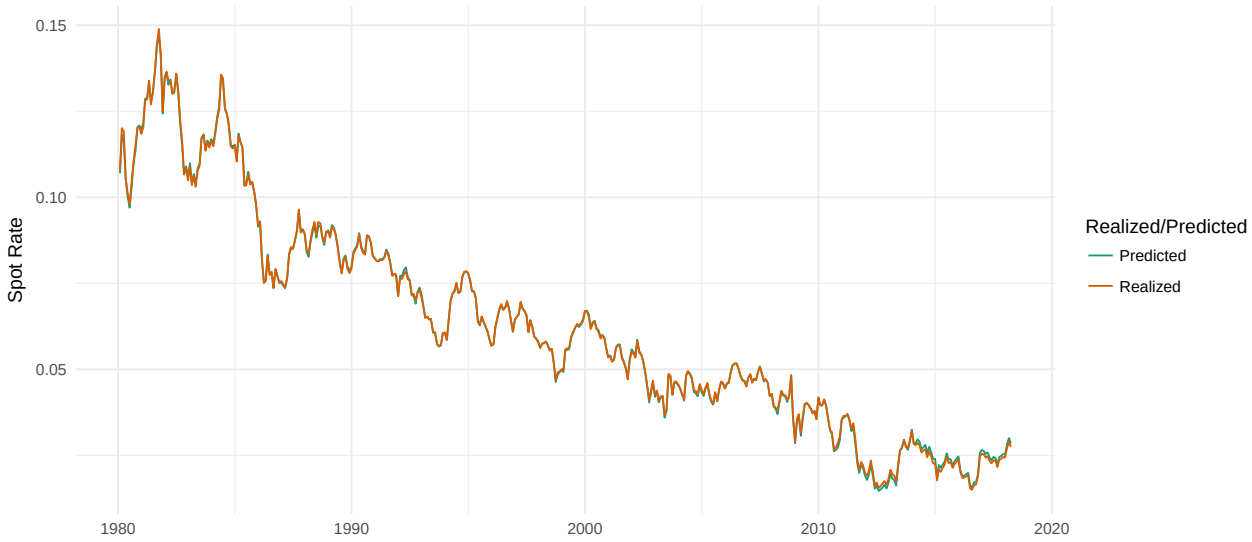


Figure 5.3: Ten year spot rate: In sample predictions VS realized

```
  yield_fct <- yield_curve_factors %>%
    as_tbl_time(date) %>%
    filter_time(~.date)

  # Multiply the date's yield curve factors by the coefficients to get the
  # decomp of the term
  decomp <- coefs %>%
    mutate(level = level * yield_fct$level,
           slope = slope * yield_fct$slope,
           curvature = curvature * yield_fct$curvature)

  decomp
}
```

```
tidy_decomposed_spot_rate <- function(.rates, .nested_models, .date) {
  .rates %>%
    extract_rates(.date) %>%
    left_join(tidy_models(.nested_models, .date), "maturity") %>%
    gather("line", "value", -(maturity:date)) %>%
    select(maturity, line, value)
}
```

What would these three functions work together to give? Well, for example, we can try looking at January of 2012. For that date, we get the spot curve values along with the level, slope and curvature components from each maturity's model.

```
rates %>%
  tidy_decomposed_spot_rate(nested_models, "2012-01") %>%
  spread(line, value)
```

```
## # A tibble: 14 x 5
##    maturity curvature   level     slope spot_rate
##       <dbl>     <dbl>   <dbl>     <dbl>     <dbl>
##  1   0.0833  2.89e- 3 0.00225 -1.04e- 3   0.00254
```

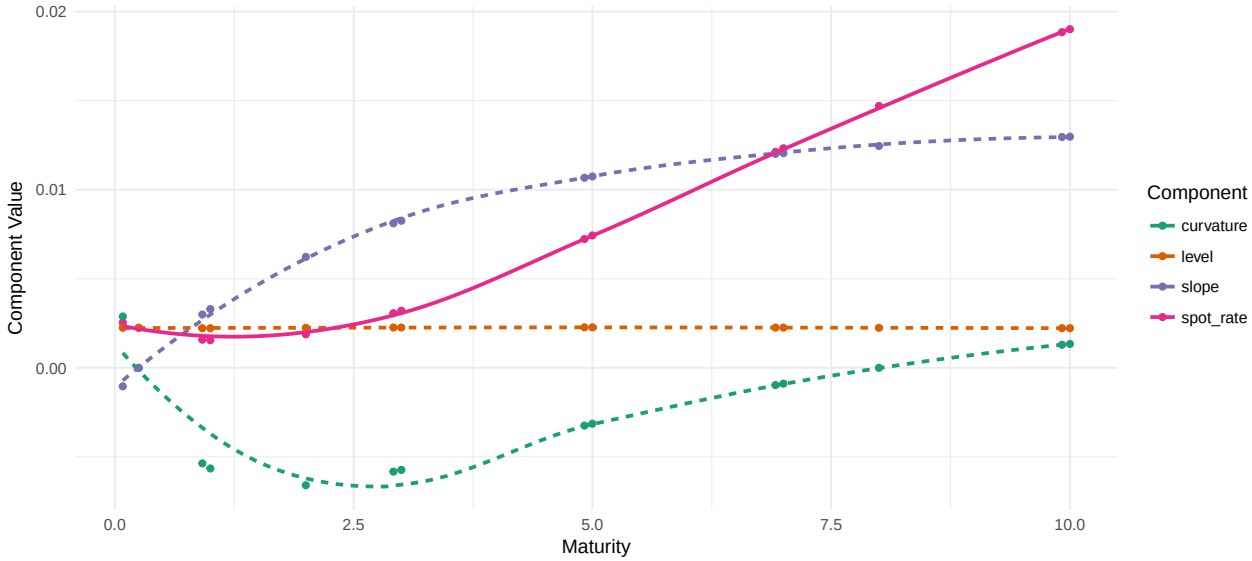


Figure 5.4: Decomposed Spot Rate for January 2012

```
##  2   0.25    2.51e-18 0.00225  1.25e-18   0.00225
##  3   0.917  -5.37e- 3 0.00222  3.00e- 3   0.00158
##  4   1      -5.65e- 3 0.00222  3.30e- 3   0.00155
##  5   2      -6.59e- 3 0.00225  6.23e- 3   0.00188
##  6   2.92   -5.83e- 3 0.00226  8.12e- 3   0.00307
##  7   3      -5.73e- 3 0.00226  8.26e- 3   0.00320
##  8   4.92   -3.24e- 3 0.00227  1.07e- 2   0.00723
##  9   5      -3.14e- 3 0.00227  1.07e- 2   0.00743
## 10   6.92   -9.69e- 4 0.00226  1.20e- 2   0.0121
## 11   7      -8.88e- 4 0.00226  1.21e- 2   0.0123
## 12   8       1.24e-17 0.00225  1.25e- 2   0.0147
## 13   9.92    1.30e- 3 0.00223  1.30e- 2   0.0188
## 14  10       1.34e- 3 0.00223  1.30e- 2   0.0190
```

This extends naturally to charting the decomposed spot rate.

```
# A charting function with custom themes
chart_decomposed_spot_rate <- function(.decomposed_spot) {
  .decomposed_spot %>%
    mutate(linetype = case_when(
      line == "spot_rate" ~ "a", # this works by alphabetical. first is solid, then dashed
      TRUE ~ "b"
    )) %>%
    ggplot(aes(x = maturity, y = value, color = line, linetype = linetype)) +
    geom_point() +
    geom_smooth(method = "loess", se = FALSE) +
    theme_minimal() +
    scale_color_brewer(palette = "Dark2") +
    labs(x = "Maturity", y = "Component Value", color = "Component") +
    guides(linetype = FALSE)
}
```

The spot rate in January of 1981 was definitely an interesting time period! It's essentially inverted, with lower maturity bonds having higher spot rates than longer maturity bonds.

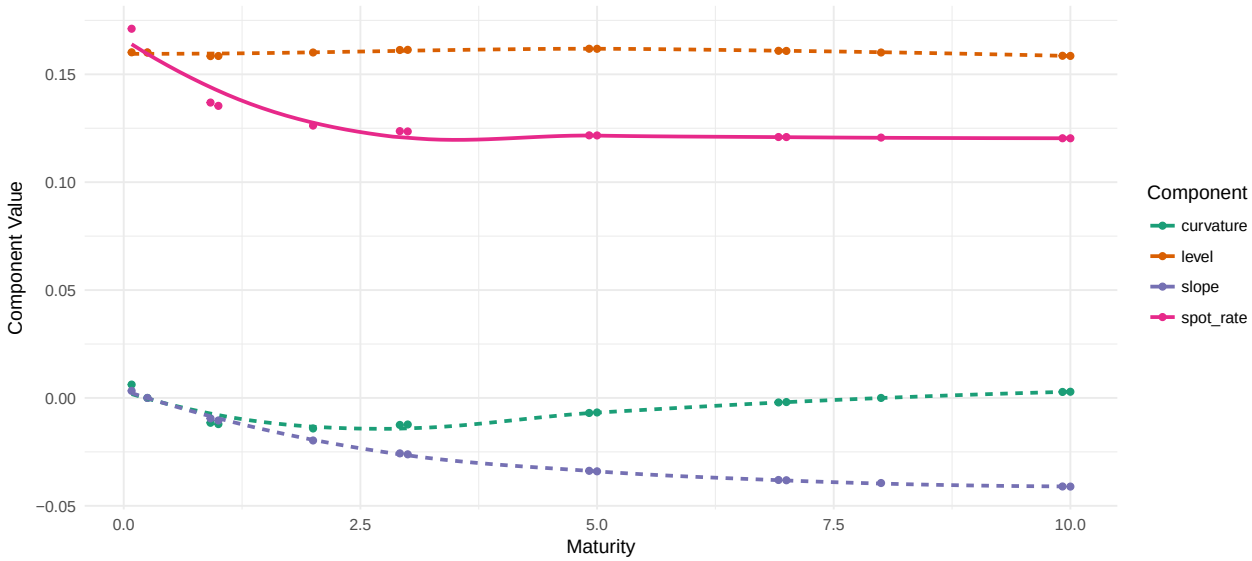


Figure 5.5: Decomposed Spot Rate for January 1981

5.6 Coefficient Stability

Another question worth asking is how stable the coefficients are throughout time. We can test this by running the same regression as before, but with a *rolling window*. This works by calculating the regression `spot_rate ~ level + slope + curvature` for the first 100 days, then shift forward 1 day and drop the last day, calculate the regression again, and repeat this for the length of the series. The `rsample` package provides a number of helpers for doing analysis exactly like this.

First, let's split up our data into the rolling subsets. This results in an extremely useful and compact format for our modeling purposes.

```
all_splits <- spot_rates_q2 %>%
  # Add on the curve factors
  left_join(yield_curve_factors, by = "date") %>%
  group_by(maturity) %>%
  select(-maturity_nm) %>%
  nest() %>%
  # For each maturity, split the data into subsets of 100 data points
  mutate(data_splits = map(data, ~rolling_origin(.x, initial = 100, assess = 1, cumulative = FALSE)))

all_splits
```

```
## # A tibble: 14 x 3
##    maturity data              data_splits
##       <dbl> <list>            <list>
##  1   0.0833 <tibble [459 x 5]> <tibble [359 x 2]>
##  2   0.25   <tibble [459 x 5]> <tibble [359 x 2]>
##  3   0.917  <tibble [459 x 5]> <tibble [359 x 2]>
##  4   1      <tibble [459 x 5]> <tibble [359 x 2]>
##  5   2      <tibble [459 x 5]> <tibble [359 x 2]>
##  6   2.92   <tibble [459 x 5]> <tibble [359 x 2]>
##  7   3      <tibble [459 x 5]> <tibble [359 x 2]>
##  8   4.92   <tibble [459 x 5]> <tibble [359 x 2]>
##  9   5      <tibble [459 x 5]> <tibble [359 x 2]>
```

```
## 10   6.92   <tibble [459 x 5]> <tibble [359 x 2]>
## 11   7      <tibble [459 x 5]> <tibble [359 x 2]>
## 12   8      <tibble [459 x 5]> <tibble [359 x 2]>
## 13   9.92   <tibble [459 x 5]> <tibble [359 x 2]>
## 14  10      <tibble [459 x 5]> <tibble [359 x 2]>
```

If we were to dig into just 1 of the `data_splits` we would find another tibble containing the slices.

```
all_splits$data_splits[[1]]
```

```
## # Rolling origin forecast resampling
## # A tibble: 359 x 2
##   splits       id
##   <list>       <chr>
## 1 <S3: rsplit> Slice001
## 2 <S3: rsplit> Slice002
## 3 <S3: rsplit> Slice003
## 4 <S3: rsplit> Slice004
## 5 <S3: rsplit> Slice005
## # ... with 354 more rows
```

Each of the `rsplit` objects here contain all of the info needed to run the linear model on that subset. We access that split's 100 data points with `analysis()`.

```
analysis(all_splits$data_splits[[1]]$splits[[1]])
```

```
## # A tibble: 100 x 5
##   date       spot_rate  level    slope curvature
##   <date>         <dbl>  <dbl>    <dbl>     <dbl>
## 1 1980-01-31    0.126  0.124  -0.0176    0.00755
## 2 1980-02-29    0.136  0.140  -0.0193   -0.0179
## 3 1980-03-31    0.161  0.159  -0.0412   -0.00205
## 4 1980-04-30    0.116  0.114  -0.00975   0.0137
## 5 1980-05-30    0.0795 0.0816  0.0173    0.00118
## # ... with 95 more rows
```

This notation might all look complicated, but it turns out to be incredibly useful and scalable. To prove that, let's run the model on every split, for every maturity.

```
plan(multiprocess)

all_splits <- all_splits %>%
  mutate(
    model_coef = future_map(data_splits, ~{ # For each maturity...

      maturity_splits <- .x

      map_dfr(maturity_splits$splits, ~ { # For each split...

        split.x <- .x
        date <- assessment(split.x)$date

        # Run the model
        mod <- lm(spot_rate ~ level + slope + curvature, data = analysis(split.x))

        # Tidy up
        tidy_mod <- mod %>% tidy()
```

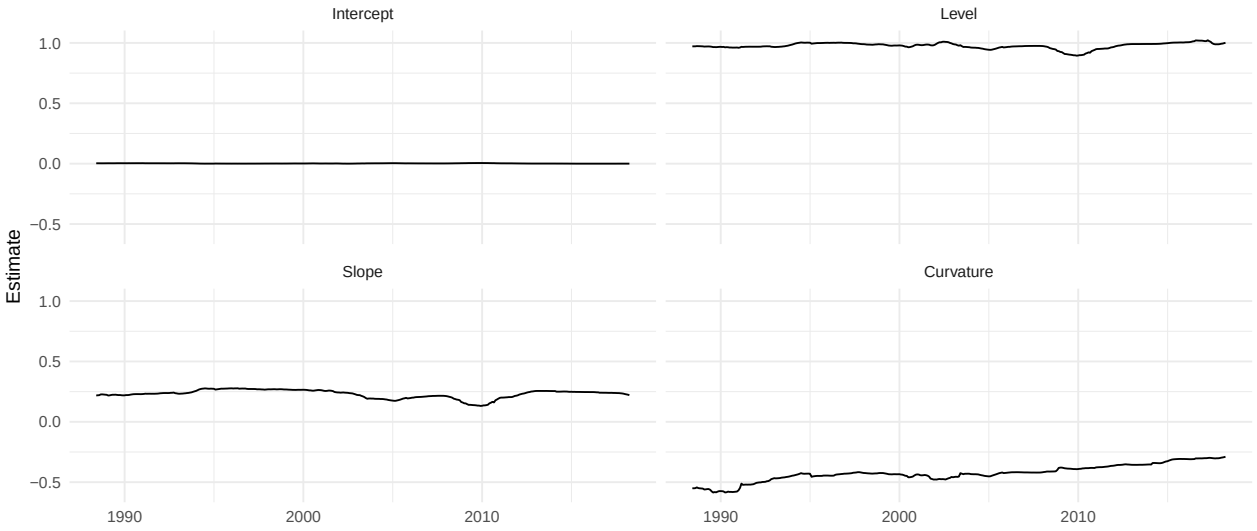


Figure 5.6: Coefficient stability for the 1 year spot rate using a 100 day rolling window

```
        mutate(tidy_mod, date = date) %>%
          select(date, term, estimate)
      })
    })
  )

all_splits
```

```
## # A tibble: 14 x 4
##    maturity data              data_splits        model_coef
##       <dbl> <list>            <list>             <list>
##  1   0.0833 <tibble [459 x 5]> <tibble [359 x 2]> <data.frame [1,436 x 3]>
##  2   0.25   <tibble [459 x 5]> <tibble [359 x 2]> <data.frame [1,436 x 3]>
##  3   0.917  <tibble [459 x 5]> <tibble [359 x 2]> <data.frame [1,436 x 3]>
##  4   1      <tibble [459 x 5]> <tibble [359 x 2]> <data.frame [1,436 x 3]>
##  5   2      <tibble [459 x 5]> <tibble [359 x 2]> <data.frame [1,436 x 3]>
##  6   2.92   <tibble [459 x 5]> <tibble [359 x 2]> <data.frame [1,436 x 3]>
##  7   3      <tibble [459 x 5]> <tibble [359 x 2]> <data.frame [1,436 x 3]>
##  8   4.92   <tibble [459 x 5]> <tibble [359 x 2]> <data.frame [1,436 x 3]>
##  9   5      <tibble [459 x 5]> <tibble [359 x 2]> <data.frame [1,436 x 3]>
## 10   6.92   <tibble [459 x 5]> <tibble [359 x 2]> <data.frame [1,436 x 3]>
## 11   7      <tibble [459 x 5]> <tibble [359 x 2]> <data.frame [1,436 x 3]>
## 12   8      <tibble [459 x 5]> <tibble [359 x 2]> <data.frame [1,436 x 3]>
## 13   9.92   <tibble [459 x 5]> <tibble [359 x 2]> <data.frame [1,436 x 3]>
## 14  10      <tibble [459 x 5]> <tibble [359 x 2]> <data.frame [1,436 x 3]>
```

We've just run `359 * 14 = 5026` linear models. For me, it runs in parallel on 4 cores in ~8 seconds.

At this point, we can pick a maturity, and look at it's coefficients over time. For example, the 1 year spot rate has fairly consistent model terms, but the curvature has begun to rise up from `-0.5` to around `-0.25`. This change over time isn't refected in the `-0.428` curvature estimate we get from running the model over the full time period, and might offer other interesting insights.

The 5 year rate, on the other hand, shows the opposite result, with the curvature term decreasing over time.

Finally, the 10 year shows a similar trend as the 1 year, but starts higher initially.

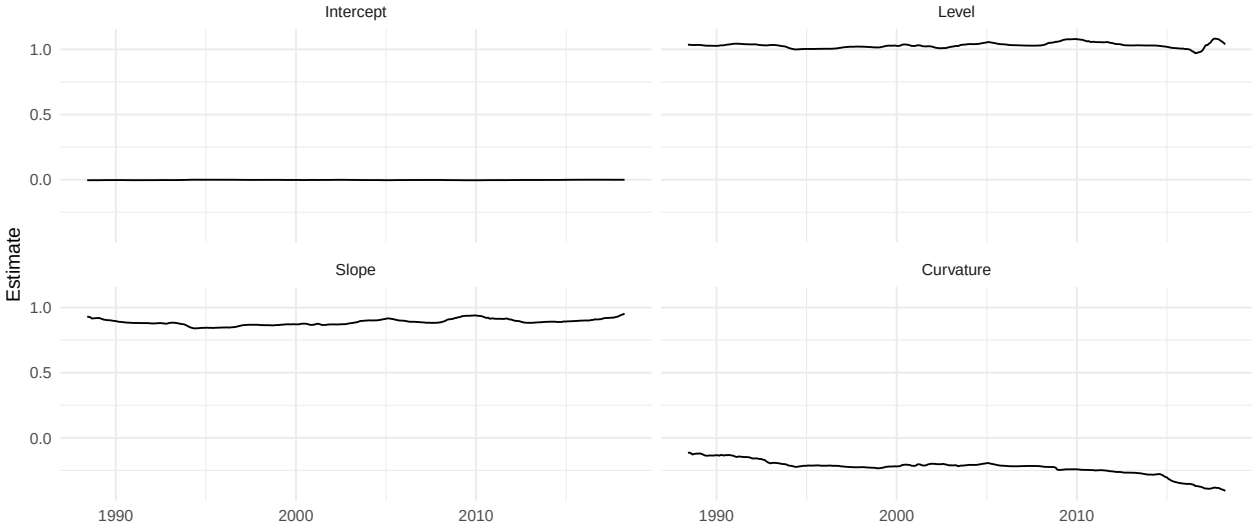


Figure 5.7: Coefficient stability for the 5 year spot rate using a 100 day rolling window

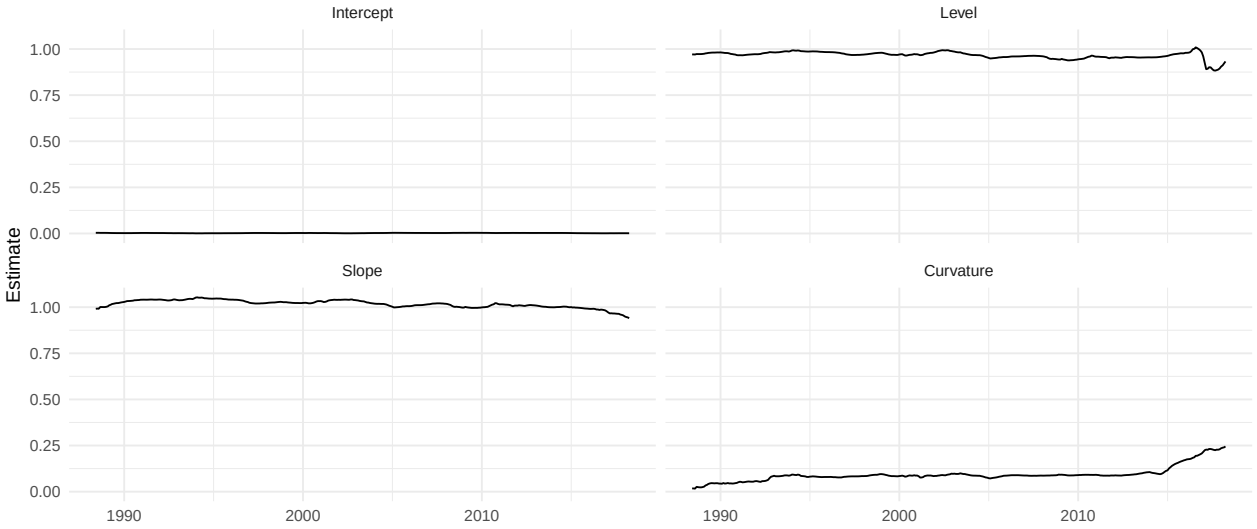


Figure 5.8: Coefficient stability for the 10 year spot rate using a 100 day rolling window

5.7 Do We Need All Three Factors?