Hand-in Assignment

Quantitative Methods in Fixed Income, Michel van der Wel, Erasmus U. Rotterdam

IMPORTANT INFORMATION

Below is the hand-in assignment. You will receive a grade on a scale of 1-10 (0 if you hand in nothing or in case of plagiarism). This grade for your hand-in assignment counts for 15% for your overall course grade. The assignment is a CodeGrade assignment, available through Canvas, with ultimate deadline of Friday December 5 at 17:59. Make the assignment on your own; there will be a plagiary scan. Despite of the late deadline, you can start making this assignment already very early on in the course, even starting October 30. Please manage your time well. Even for the last part you have eight working days to complete it, but if you begin only after Lecture 9 with the hand-in assignment you won't make the deadline.

ASSIGNMENT

Coding You can make this assignment with either Octave,[1] Python or R. For each of the five parts given below, there is one function that needs to be programmed. The number of points for each of the parts is given in square brackets. Program the functions first on your own computer. Make sure to use the exact names for each of the functions, with arguments in the given order! Then, after you are happy with the result, you can simply drag and drop the file(s) to the CodeGrade environment.[2] Immediately after the assignment is available, you can in fact already upload the file(s) and test the functions on CodeGrade. Using some pre-set input values, CodeGrade then checks whether each function returns the right value and if it doesn't provide an error. It is strongly advised to test your code! You can upload the file(s) as many times as you want. This can be done one function at a time, so even in week 1 after the second lecture you can try Part I and already earn part of your grade.[3] The file(s) last-uploaded before the deadline will be used for grading the assignment. If you are working in Octave, you can upload the files one function at a time and you should give the file for each function the name of that function. For Python and R, give the file with all functions the name "HIAforQMiFI."

Data The data set to be used for the assignment is available in the assignment on Canvas. It has already been pre-loaded on the root folder on CodeGrade, so you don't need to upload it there yourself. The data are a subset of the Liu and Wu (2021) data.[4] We consider maturities from one month through ten years, with a monthly maturity grid, for a period from January 1972 through

[1] Octave is the free version of Matlab. However, some functions differ between the two languages and also the random number generators have different seeds. For a Matlab programmer, Octave is the most natural choice for this assignment.

[2] For Octave you upload five files, one for each of the functions. For Python and R you upload one file.

[3] For Octave, it is necessary to upload empty function files for the functions you have not completed yet, if you want to upload some functions early.

[4] Based on Liu and Wu (2021), "Reconstructing the Yield Curve", Journal of Financial Economics, 142(3), 1395-1425. Full data available from `https://sites.google.com/view/jingcynthiawu/yield-data`. Downloaded September 12, 2025.

December 2024. Yields have been converted such that 1% is 0.01 in the data. The data set is needed for Parts I-III only.

Part I: Stylized Facts [1 point] (can be made after lecture 2) We start by creating a function that calculates one of the summary statistics, such that it could be used to check the stylized facts. Specifically, create a function that returns the (time series) mean for yield j. In your function, load the data, and consider the full time series. For loading the data, use `xlsread` for Octave, `read_excel` from the pandas package in Python, and `read_excel` in R. Name your function `getTimeSeriesMean`. The function should have as argument an index j that can range from 1 through 120, which decides which of the maturity indices the time series mean is calculated.[5]

Part II: Nelson-Siegel Forecast [2 points] (can be made after lecture 3) Next, we create a function that provides a yield curve forecast based on the Nelson-Siegel model. In the function there are the following steps:

1. Load the data.
2. For a given subsample provided by one of the function arguments i, run the Nelson-Siegel model (via OLS with fixed $\lambda = 0.0609$) to obtain the level, slope and curvature time series. The index i represents the last observation that is used for the subsample and the subsample always starts from the first observation. E.g., if $i = 12$, the first twelve months are used as subsample.[6] Run the regression using `regress` in Octave, `LinearRegression()` in Python and `lm` in R.[7]
3. Estimate independent AR(1) models for each of the three obtained times series using the subsample only.
4. Then, depending on another function argument, forecast the level, slope and curvature h periods ahead using the output from Step 3.
5. Finally, using the Nelson-Siegel model, scale up the level, slope and curvature forecasts to obtain a yield curve forecast and let the function return the forecast for maturity index j, which is also a function argument.

Name your function `getNelsonSiegelForecast`. The function should have as argument an index i that can range from 12 through 612, which decides through what point the subsample runs (so from row index 1 through i), a number h which runs from 1 through 24 which indicates how many steps ahead the forecast should be taken, and an index j that can range from 1 through 120, which decides which of the maturity indices the h-step ahead yield curve forecast is returned. The arguments should be provided in that order.

[5] Indexing differs between Octave and R, and Python. In the latter language indexing starts from zero while in the other two it starts at one. In your function, make sure $j = 1$ refers to the first maturity, so one month, irrespective of the language you've used. Do this similarly for Parts II and III.

[6] Keep maturities measured in months for this value of λ. If you would like to use maturities measured in years, use the value $12 \times 0.0609 = 0.7308$ in your code.

[7] An old version of the assignment required the use of `reg` in R, but this caused some issues. The answers on CodeGrade have been obtained using the `lm` version. Other ways for linear regression may be used (there is no specific requirement for this anymore for R), but can cause different answers and hence the advice is to use `lm`.

Part III: Expectations Hypothesis Test [2 points] (can be made after lecture 4) Using the data, we will now examine whether or not the expectations hypothesis holds. Specifically, create a function that returns the estimated $\beta(\tau)$ for the first expectations hypothesis test as given on Slide 14 of Lecture 4. In your function, load the data, and consider the full time series. Name your function `getEHtest`. The function should have as argument an index j that can range from 2 through 120, which decides which of the maturity indices the test is performed (note that as we are testing the yield curve slope, for index 1 there is a trivial test and we thus don't consider that as possible value for j).[8]

Part IV: Vasicek Function [1 point] (can be made after lecture 6) We now program a function that returns the price of a zero-coupon bond given input values of κ, μ, σ, r_t and τ for the one-factor Vasicek model. These should be, in the given order, the arguments of the function. Use the closed-form solution of slide 21 from lecture 6 with $\lambda = 0$.[9] Name this function `getVasicekPrice`. [This function does not require the data set as input and will be used as input for Part V.]

Part V: Derivative Simulation [3 points] (can be made after lecture 9) And finally, we create a function to simulate the price of a European call bond option, for a setting where the underlying bond is modelled using a 1-factor Vasicek model. [Also this function does not require the data set as input.] Input variables for the pricing problem are:

- Strike price K
- Expiry of bond option T_0
- Maturity of bond T_1
- Short-rate speed of mean reversion κ
- Long-run mean short-rate μ
- Short-rate volatility σ
- Number of replications in simulation study R
- Euler step-length Δ

In terms of timing, at the time of expiry of the bond, given by T_0, the time to maturity of the bond will be $T_1 - T_0$. The bond price is modelled using the 1-factor Vasicek model, for which a closed-form solution exists. Hence, the short-rate needs to be simulated only through T_0 to

[8] In the lecture we took simple discrete-time steps of one year. We are now looking at actual data sampled at the monthly level, so where the next data entry $t+1$ is one month from time t. The relevant test in this case is

$$y_{t+1}\left(\tau - \tfrac{1}{12}\right) - y_t(\tau) = c(\tau) + \beta(\tau)\frac{y_t(\tau) - y_t\left(\frac{1}{12}\right)}{\tau - \frac{1}{12}} + \epsilon_{\tau,t+1}.$$

Make sure in your code that maturities τ are now given in months and have to be adjusted to be in years for the test to hold (so a maturity τ of 2 months should be 2/12 for the test).

[9] We set $\lambda = 0$ as the function will be used as input for Part V, when it is used for pricing an interest rate derivative.

calculate both the discounting as well as the short-rate at this time as input for the bond pricing function. Simulate under the risk-neutral measure (thus take $\lambda = 0$) and initialize with $r_0 = \mu$. Your function should return the simulated bond option price given input values of κ, μ, σ, r_0, R, Δ, T_0, T_1 and K (make this, in the given order, arguments of the function). [Note that in the assignment we have $r_0 = \mu$, but it is good to program this function as general as possible. Do not hard-code this relation.] Assume T_0 and T_1 are always given in full years. Name this function `getSimBondOptionPrice`. Make sure you call the Vasicek bond price function from Part IV in the bond option price calculation code!

We will now structure the bond option price calculation code a bit such that, given a certain random seed set in CodeGrade, only one price will be returned by your code and your assignment can thus be easily graded. Use a two for-loop structure as in the examples from slides 14 and 19 of lecture 9. Note that other ways of implementation are possible, but that has the risk of getting different random numbers used and thus a different answer. To calculate the simulated short rate, use `randn` in Octave, `numpy.random.normal(0,1)` in Python and `rnorm(1)` in R as the random number generator function to obtain scalar random numbers. We are thus drawing from the standard normal distribution, which can be multiplied by the standard deviation to get draws from a normal distribution with the right variance.

Code Neatness [1 point] In addition, 1 point is earned for the quality of your coding; for which CodeGrade will provide detailed feedback. To give you a little bit of leeway for the code neatness, your overall grade will be rounded up to the nearest decimal point (so 9.91 will become a 10.0).

Good luck and (hopefully) have fun programming and linking the course material to data and code! Treat it as your first assignment as 'quant.'