Simen Voltersvik

Bachelor's thesis

TOOL FOR ESTIMATING IFRS9 MACRO MODELS

For DNB Bank ASA

June 2020

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Bachelor's thesis

2020

NTNU
Norwegian University of
Science and Technology

Simen Voltersvik

TOOL FOR ESTIMATING IFRS9 MACRO MODELS

For DNB Bank ASA

Bachelor's thesis
June 2020

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

To comply with the regulation outlined in IFRS9 to use forward looking information to estimate expected loss for credit exposures, financial institutions are required to model movements over time in average observed default frequency for subsegments of their portfolio. In DNB, this is done through OLS regression using a credit cycle derived from weighted observed default frequency for specified segments as dependent variable and external macro variables as explanatory variables. The challenge is to find the correct variables to use, and further to know what transformations of these variables to use. Previously this was a time-consuming process in DNB that did not aid sufficiently in finding the optimal model to explain the credit cycle. The program developed for this paper aims to increase the help offered in finding OLS models through brute-force search where all possible combinations of likely explanatory variables are tried. The program also aims to make in-depth analysis of specified models more streamlined and efficient.

Contents

Preliminary study

The preliminary study will provide background information about the problem the IFRS9 Macro Tool is intended to solve, describe DNB's current tool for estimating IFRS9 macro models and the issues related to it, and lastly give a general description of how the new tool is intended to be, and argue why it will solve the business problem better than DNB's current tool.

Background

IFRS9 is an International Financial Reporting Standard meant to provide regulation of the financial industry to minimize the probability of system failures on the scale of the 2008 financial crises. The standard has been developed by the International Accounting Standards Board (IASB) and came into effect from 1st of January 2018 (McGeachin & Tarce, 2019).

One of the key principles of the IFRS9 standard regarding financial instruments accounting is that financial institutions are required to make impairments of expected future credit losses. In the old standard IAS 39 losses were only entered in the accounts when customers defaulted on their loans. By making impairments before the actual defaults happen, the banks are forced to build up capital to avoid systemic shocks to the industry on that scale witnessed in 2008.

To calculate impairments for future credit loss and be compliant with the IFRS9 regulations, DNB has developed a model to calculate expected credit loss (ECL). This approach entails grouping all loan agreements into segments that have similar characteristics. These segments are grouped by both industry and geography. Each of these segments are mapped to an IFRS9 macro model, where segments mapped to the same IFRS9 macro model follow a similar credit cycle. A credit cycle is the cycle of loan default frequencies for a given segment, and is countercyclical to the business cycle of that segment. When the business cycle of a given segment is high, the number of defaults is low, hence the credit cycle is low. The opposite is true when the business cycle is low. The way credit cycles are predicted forward in time for each of DNB's IFRS segments, is to use OLS regression models where historical Z is modelled using historical macro data, and input predictions of these macro variables are used to estimate future Z.

Credit cycles, denoted as Z, are modelled as a normal distribution over a full business cycle. In order to estimate the OLS regression models for each segment, historical data of the observed default frequency (ODF) for all agreements in each segment is used to generate a historical Z time series. For larger segments where DNB has a sufficient amount of agreements to make unbiased a historical Z time series, internal default data is used. For segments where DNB has too few agreements to generate an unbiased historical Z timeline, external default data from the Credit Risk Initiative from

the Risk Management Institute of the National Institute of Singapore (RMICRI) is used to generate the historical Z time series.

In order to predict future Z time series, DNB has developed a tool to transform internal DNB default data and default data from RMICRI to historical Z time series. The tool also does transformation on macro data times series provided by DNB Markets, and aids in finding intuitive OLS models with correlation between the macro data time series and the generated historical Z time series.

DNB is currently in the process of rewriting the bank's ECL model, that takes these macro models as an input, from SAS Enterprise Guide running on inhouse Linux servers to Python running on Amazon Web Servers. The format of how the coefficients of the macro models are stored will be changed in this process, requiring the output of the macro model estimation tool to be changed from the current setup.

Current DNB tool for estimating IFRS9 macro models

Insight into how the current DNB tool for estimating IFRS9 macro models work, and what drawbacks it has, have been acquired through discussions with the developers of the program, and the users of the program.

The current tool DNB is using to estimate IFRS9 macro models is developed in SAS Enterprise Guide. It is structured as a series of individual steps that must be run in a particular order. The analysts using the tool to estimate the IFRS9 macro models must use industry intuition to pick out what macro variables and the transformation of these that are expected to correlate to the Z time series of the particular IFRS segment being modelled, and what time series data points are outliers, and hence require dummy variables. To make the program estimate the model, the analyst must insert the name and transformation of the macro variable, together with the dummies, into the code itself, and run a particular set of steps. The output of the program gives information about r-squared of the model, the p-values and coefficients of the give explanatory variables, and the test results of a number of statistical tests.

The program also has functionality where the analyst can specify a set of macro variables, and the program will return a list of suggestions of models ranked by the r squared value of each model. This functionality does not consider the significance of the explanatory variables of the suggested models.

Drawbacks of DNB's current tool for estimating IFRS9 macro models

The main drawbacks that have been identify are twofold. It is both time consuming for analysts to use and it does not aid sufficiently in finding the optimal macro models.

Because the program is structured as a series of steps the analyst must execute in a particular order, it requires good familiarity of the program from the analyst in order to use the tool correctly. If the steps are run in an incorrect order, the program might crash, or it might produce incorrect output. Because different models are tested by altering the code itself, it also requires analysts to have sufficient programming experience to use efficiently. There is also significant operational risk involved in using this program, as it is not always clear if the analyst has made a mistake when executing the program or what input data actually goes into the program. This might lead to inaccuracies in the output from the program that are hard to detect.

As the program requires the analyst to manually enter the type and transformation of the macro variables expected to correlate with the segment Z time series, and dummies that might capture outliers, it is not given that the optimal combination of variables, variable transformation, and dummies for modelling the Z time series is found. The functionality in the program where it tests different transformations does little to account for this issue, as it only ranks macro models based on r squared, ignoring other important OLS information like p-values of the individual macro variables or if the variable coefficients have an expected impact on the modelled time series. It also lacks the possibility to aid with identifying possible outliers where dummies should be utilized.

Future DNB tool for estimating IFRS macro models

It is a wish from DNB for the new tool to be integrated with the new implementation of their ECL model. For that reason, the new tool for estimating DNB's IFRS9 macro models will be written in Python, and put on DNB's development platform along with the new ECL model to easily allow for the tool and the ECL model to be fully integrated in the future.

The design of the program has not been fully landed yet at his stage, but it is a requirement from DNB that the new design will try to overcome the shortcomings of the program it is replacing. This involves creating the program in such a way that the steps associated with the current tool is eliminated through streamlining the entire flow. To achieve the program will have a user-centered design adhering to the principles outlined by Donald A.Norman(2013). This can be boiled down to making the program in such a way that it is adjusted to the needs and behaviour of the users, and not requiring the users to adjust excessively to the program. The user should only have to input necessary data at the start of the operation. This is intended be solved by having a terminal based interface, as DNB's development platform on AWS is not currently supporting any python GUI. From

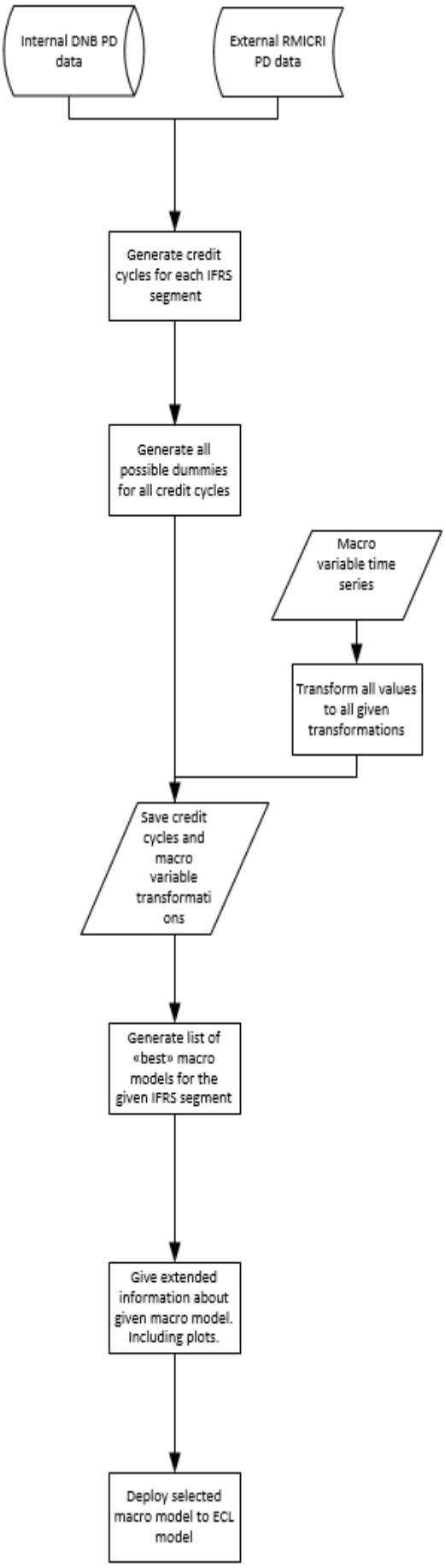
this interface the user will be able to interact with all functionality of the tool, without having to see or alter any python code. The product owner of DNB's development platform on IPA has revealed that the platform might implement support for Dash during the time this project will be developed. If that becomes available, the macro tool will have a web interface developed through the Dash framework.

The different parts of the program will still have to be executed in a designated order, but this will be clearly specified in the program's user guide, and the program will return helpful error messages if the user interact with the program incorrectly. As all interaction with the program will happen from an interface, there will be no need for the users to alter the program code to use the tool. As oppose to the current tool, the new one will have validation of input to the program, something that will minimize the chance of incorrect input altering the output of the program unnoticed, while also making it much easier for the users to locate errors in their input.

The new tool for estimating IFRS9 macro models must also provide more assistance to analysts for finding the optimal models. It will improve upon the functionality from the current tool that generates a list of models ranked by the value of adjusted r squared, by also generating and trying combinations of dummy variables on the models. Like the current model, the new model will also rank the models by value of adjusted r squared, but it will also discard models that have explanatory variables with a p value above a threshold set by the user, models with variables having variable coefficients not adhering to industry intuition, and models with value of adjusted r-squared bellow a threshold set by the user . This change is believed to greatly enhance the aid given by the program to the user in finding an optimal model. The ambition is to reduce the need for great industry intuition in order to find the optimal IFRS9 macro models, and for the users to avoid excessive use of trial and error to find suitable variable transformations.

Future process flow

Together with future users of this tool, a first draft of how the program flow will be has been outlined. Further documentation of the program will be in the requirement analysis and technical documentation.

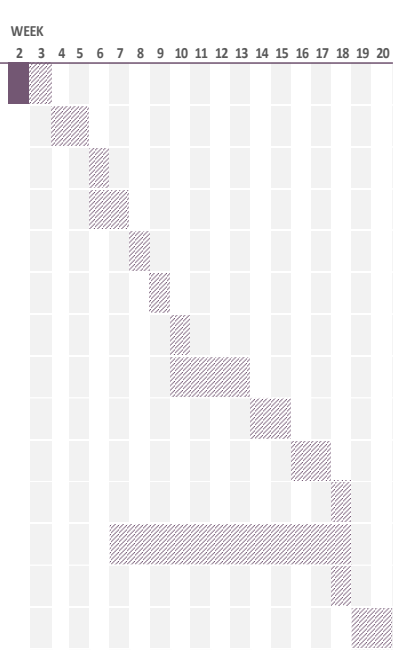


- RMICRI data are excel files analysts can upload to a folder on DNB's development platform. In order to get internal data, a link to DNB's data warehouse is required.
- In order to generate credit cycles, customers must first be mapped to the correct IFRS9 segment. This is trivial for internal data, but requires some logic to map RMICRI data correctly. After mapping, the mean for individual years and the entire time series are calculated to generate credit cycles.
- When the length of the time series is found, all possible dummies for the given time series are calculated.
- Macro variable time series are in excel files analysts can upload to a folder on DNB's development platform.
- These macro variables are all transformed to multiple defined transformations, including lag, moving average and inflation adjustment.
- Credit cycle, macro variable and dummy data are stored in a dedicated folder on DNB's development platform.
- All possible combinations of variable transformations and dummies are tested against the credit cycle time series of the IFRS9 segments. As there are millions or billions of possible combinations of variables and dummies, there must be ways for the users of the program to instruct the program as to what variables not to include in this step for the different IFRS9 macro models. This is to avoid noise in the output to the user, and to drastically reduce execution time. Results that have significant explanatory variables with an expected effect on modelled Z and a high enough adjusted r-squared, are ranked by value of r squared, and saved on DNB's development platform.
- User can select what OLS models to analyse further. What additional information is needed, will be detailed on a later stage.
- When macro model for the given IFRS segment is decided, the model parameters are stored in a format aligned with the new ECL model.

Project timeline

The project's timeline will use week numbers as time values. To account for potential delays, the final delivery for the project is set to week 21, corresponding to 24th of May. This allows for a few weeks delay if some project activities are more time consuming than expected.

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Develop preliminary report	2	2	2		50%
Develop requirements analysis	4	2			0%
Milestone - planning phase finished	6	1			
Code development - Generate credit cycles	6	2			0%
Code development - Generate Dummies	8	1			0%
Code development - Variable transformation	9	1			0%
Milestone - data preparation finished	10	1			
Code development - Generate list of best model fits	10	4			0%
Code development - In-depth analysis	14	2			0%
Final testing and debugging	16	2			0%
Milestone - development phase finished	18	1			
Develop User documentation	7	12			0%
Develop User guide	18	1			0%
Finish paper	19	2			0%



Figur 1 - GANTT diagram of scheduled timeline for project

The project is divided into four phases. The preliminary phase is where the work is focused on getting a proper understanding of the business issue to be solved, and to plan for the development period that follows. There are two activities in the preliminary phase, developing the preliminary study report and developing the requirements analysis. Both of these activities are crucial in order to develop a program that actually solves DNB's business problem. The preliminary phase is scheduled to be completed by week six, corresponding to 2nd of February.

The development phase is divided into two distinct phases. The first phase is focused on developing the code to read, validate, clean, and transform data, and to store this in a format suitable for data analysis. There is some uncertainty as to how time consuming it will be to develop the part of the program generating the credit cycles, as the DNB logic for mapping internal and external data to IFRS9 segments is somewhat complexly implemented in SQL and lacks any documentation. The other activities in this phase are not expected to take more than the allocated time. This phase is expected to be completed by week 9, corresponding to 23rd of February.

The data analytics phase is expected to be the most time-consuming phase, where complex functionality not offered by DNB's current model will be developed. It is allocated a generous amount of time to the activities in this phase, as this is the part of the development with the highest uncertainty regarding how challenging it will be to implement the required functionality. Finishing of the program and final testing will also be part of this phase. This phase is expected to be completed by week 18, corresponding to 26th of April

The last phase is the documentation phase. Development of the technical documentation will be done in parallel with development of the program, but it will be finished in this phase. A user guide will also be developed in this phase, and the paper to be delivered will be finished. It is not expected to be any uncertainties regarding time consumption in this phase. This phase is expected to be finished by week 21, corresponding to 18th of May.

Requirement analysis

The requirement analysis will provide the necessary foundation to develop a program that sufficiently solves DNB's business problem. Expected use of the model will be documented in user stories, to make sure the program can be used in all its intended roles. This will give a general overview of the expected use of the program. Based on the user stories, both functional and non-functional requirements will be documented, and linked to the relevant user stories. This will specify what functionality is expected from the program. The user stories and requirements will also form the basis for later testing of the program.

Both the user stories and the functional requirements were developed in cooperation with future users of the program. All the requirements specifying expected output from the program and execution speed was directly specified by future users in DNB, while others were specified by the author of this paper after discussions with future users. Particularly regarding the OLS suggestion functionality, linked to User story 2, as this was somewhat new functionality in the program, all of the requirements were specified by the author and cross checked with future users of the program for confirmation. The functionality describing the structure of the program, were also developed by the author before and under program development by taking inputs from future users as to what overall wishes for the program were, and discussing it further to distil it down to specific program requirements.

Users of the program

The program will exclusively be used by a small group of analysts. These analysts will not necessarily have great insight into the workings of the program, nor will they have any particular knowledge of Python or programming in general. For that reason, it is particularly important to have extensive validation of all input to the model, and return informative error information to the analyst. The analysts are used to working with similar in-house developed software, and are not dependent on a graphical user interface in order to use the program, as long as the user interface is intuitive and they have a user guide.

General standards of the program

As the ECL model as a whole, including the process to estimate the OLS models, constitutes the majority of the yearly audit of DNB, it is paramount that the logic used in the program is sufficiently easy to follow. This require clearly readable code with descriptive variable, class and function names. It might also involve in some instances to prioritize making the code readable rather than making the code quick. DNB does not have a clear general coding standard one is expected to follow, but as this program will be integrated with the new ECL model currently under development, it is natural that the two programs follow a similar code practice. For the new ECL model, the PEP 8 style guide is used as a general guideline, although some parts of it is less strict. The same style guide will therefore be used for this program.

User stories

User story 1: Generate credit cycles
The user must be able to generate the Z time series for each IFRS9 Macro Model based on internal and external data on observed default frequency (ODF). These Z time series are what will be modelled in the rest of the program.
Acceptance criteria
The user can easily generate Z time series for all DNB IFRS9 Macro Models.
Priority
Low Despite the Z time series are crucial for the working of the program as a whole, this part of the program is not expected to be complicated in terms of code, rather in terms of business logic. For that reason, this part of the program can be developed later by DNB, and the existing solution in SAS can be used to generate the Z values until this part of the program is developed.

User story 2: Generate list of best OLS models
The user must be able to generate a list of suggestions for explanatory variables for a chosen IFRS9 Macro Model. The list of suggestions will be sorted based on the value of adjusted r-squared of the model, while models with insignificant explanatory variables, unintuitive coefficient values and low adjusted r-squared values will be excluded.
Acceptance criteria
The user can easily generate a list of model suggestions given the user input. This list must be sorted on value of adjusted r-squared, and filtered on significance of explanatory variables, adjusted r-squared and variable coefficients.
Priority
High As this most likely will be the most challenging part of the program in terms of actual programming, this is the part of the program where DNB can have the most benefit from external help.

User story 3: Do in-depth analysis and select model
The user must be able to select a specific model for an IFRS9 Macro Model to do in depth analysis on. The program must return all relevant statistical information about the model in order for the analyst to select the optimal model. Lastly the user must have the option to save the chosen model.
Acceptance criteria
The user must be able to get all relevant statistics to have sufficient information to evaluate the robustness of the model and be able to save the model if it is deemed satisfactory.
Priority
High This is also a part of the program with a high degree of complexity where DNB will have good benefit of external help. This part of the program will also have some overlapping functionality with User story 2, so it might not be easily developed by DNB on their own at a later stage.

User story 4: Automatic validation of existing models
An analyst should be able to easily validate the existing OLS models, and models that have a significant decrease in robustness should be highlighted by the program.

Acceptance criteria
The user must be able to get a list of OLS models that have changed significantly in key aspects specified by DNB.
Priority
Medium This is functionality that is nice to have rather than must have. Meanwhile it is functionality that will lead to significant time savings for analysts recalibrating the models, and the complexity of it might mean that it is not easily done by DNB at a later stage. DNB currently lacks a complete dataset of OLS results from current models, and this part of the program will thus be difficult to implement before that is created.

User story 5: Deployment of new models
An analyst should be able to easily get an overview of the new OLS models, and deploy the parameters to the new ECL program
Acceptance criteria
The user must be able to get an overview of the new models and deploy the new parameters in the format required by the new implementation of DNB's ECL model.
Priority
High This functionality is essential for DNB in order to use this program into their new ECL model.

Functional requirements

Name	FR 1: Generate credit cycles from ODF data
Description	The ODF data for each year of the time series must be converted to a normal distribution with a mean of zero and a standard deviation of 1. This is done by these steps: - The average ODF per year is calculated. - The inverse of the normal cumulative distribution function (cdf) is calculated for the average ODF for each year. - The average and standard deviation for the calculated values are estimated. - The final Z value for each year is equal to: (the calculated inverse of the normal cumulative distribution function value – the estimated average

	value) / the estimated standard distribution The output must be saved in an excel file with one sheet for each IFRS9 Macro Model. The format must be an array with two columns, year and Z, and one row for each year of the Z time series. The program must also be able to validate that the input only has ODF values in the range 0 to 1.
Associated user story	User story 1: Generate credit cycles

Name	FR 2: Generate credit cycles from external data
Description	For the IFRS9 Macro Models associated with segments where DNB has insufficient data to accurately generate representative credit cycles, external data from RMICRI is used. These ODF values must be mapped to the correct IFRS Macro Models, and the output from this mapping must be on the same format as the input to FR 1. The external data is mapped to the specific IFRS9 Macro Models based on business rules sat by DNB.
Associated user story	User story 1: Generate credit cycles

Name	FR 3: Generate dummy variables
Description	In order to be able to account for outliers in the time series, dummy variables spanning all possible periods of the time series must be generated. After discussion with DNB analysts, it has been decided to set the default for maximum length of a dummy to five years. This must be easy for DNB to change at a later stage if the business requirement changes.
Associated user story	User story 2: Generate list of best OLS models User story 3: Do in-depth analysis and select model

Name	FR 4: Validation of macro data input

Description	The macro input data from DNB Markets are located in an excel sheet. This data must be validated for errors before being used in the program. The validation should ensure the data subscribe to these requirements: - The macro variable time series must be continuous from first year of data to the last year of data, without gaps. - All variables specified in a meta data table must be present in the input sheet. - All variables expected to have data as specified in the meta data table must have data. - All data must be numerical The program will generate a list of errors if any are detected, and halt further execution of the program.
Associated user story	User story 2: Generate list of best OLS models User story 3: Do in-depth analysis and select model

Name	FR 5: Normalize macro data
Description	The macro data from DNB Markets comes in different varieties. Some are absolute values, some are year-on-year changes, while some have absolute values for historical data and year-on-year changes for predicted data. Some time series also denotes year-on-year changes differently by either denoting the change as a decimal, or as a value representing the decimal. E.G. 4% can be both 0.04 and 4. To make the data consistent, the macro data time series must be normalized into a format specified in a meta data table. The time series for the variable LTV must also be calculated from other time series in the DNB Markets data.
Associated user story	User story 2: Generate list of best OLS models User story 3: Do in-depth analysis and select model

Name	FR 6: Transform macro data
Description	The macro data must be possible to transformed into all possible transformations specified by DNB. This involves having the possibility to do the following transformations, and combinations of these: - Lag

	- Percentage change - Absolute change - Max change - Moving average - Natural logarithm - Inflation adjustment (with both Norwegian and US inflation)
Associated user story	User story 2: Generate list of best OLS models User story 3: Do in-depth analysis and select model

Name	FR 7: Do simplified OLS regression
Description	The program must be able to perform a simplified version of OLS regression, where only value of adjusted r-squared, mean squared error, together with p-values and coefficients for the explanatory variables are calculated. The user will specify threshold values for adjusted r-square and for p-values, and the expected sign of the coefficient. If any of the calculated values are below the threshold values or if any coefficients have the wrong sign, the OLS model is rejected. If not rejected, the regression will return adjusted square, mean squared error, coefficients and variables used. In order to mitigate the problem related to bias stemming from serial correlation in the residuals, a Newey-West HAC estimator must be implemented.
Associated user story	User story 2: Generate list of best OLS models

Name	FR 8: Generate sorted list of IFRS9 Macro Models suggestions
Description	The program must be able to generate two lists of models suggestion sorted on value of adjusted r-squared based on the functionality described in RF 7. One list without dummies and one with dummies. These lists must contain the name of explanatory variables and their transformation, the sign of the coefficients for each explanatory variable, value of adjusted r-squared, mean squared error, and coefficient values.
Associated user story	User story 2: Generate list of best OLS models

Name	FR 9: Extended OLS regression statistics
Description	In addition to the OLS requirements specified in FR 7, the following statistics must also be returned for the regression: - Residuals - Predictions - Confidence intervals with alpha=5% - Durbin Watson statistics - Jarque Bera Statistics - Omnibus test statistics - Breusch-Pagan statistics - Lilliefors test statistics - Kwiatkowski–Phillips–Schmidt–Shin and augmented Dickey–Fuller statistics
Associated user story	User story 3: Do in-depth analysis and select model

Name	FR 10: Do rolling regression
Description	The program must be able to do a rolling regression in order to benchmark the model against the two simplified models as-is and through the cycle. This is done through doing multiple regressions of the same variables with an increasing time series in order to find the estimated coefficients given that particular time series. The program defaults to starting the rolling regression ten years back, and works its way to present day. The coefficients obtained are thereafter used to predict the following year. The residuals from each year is compared to the residuals from the simplified as-is and through the cycle models. Both the mean absolute error(MAE) and root mean squared error(RMSE) are calculated. The as-is model assumes the Z value of next year will be equal to the previous year in the rolling regression. The through the cycle model assumes the Z value is always 0. The user must be able to see the comparison of the MAE and RMSE to the challenger models, the predictions and confidence level of the rolling regression, and the coefficients from each year of the rolling regression. The coefficient values are wanted in order to see if coefficients are relatively stable from year to year.
Associated user story	User story 3: Do in-depth analysis and select model

Name	FR 11: Visualising the regression results
Description	The program must be able to make visualisations of the regression results. The visualisations that are requested are: Comparing rolling regression predicted Z with confidence levels to actual Z Comparing predicted regression with confidence level to actual Z
Associated user story	User story 3: Do in-depth analysis and select model

Name	FR 12: Saving the regression parameters
Description	The user must be able to save the regression parameters when the desired model is found. The parameters that have to be saved are adjusted r-squared, coefficient values, p-values of explanatory variables and name and transformation of explanatory variables.
Associated user story	User story 3: Do in-depth analysis and select model

Name	FR 13: Evaluating existing OLS models
Description	The program must automatically re-estimate the coefficients of the models currently in production with the ECL model, and compare these to a set of business rules to see if the models still are satisfactory. These are the business rules that will flag a model for re-evaluation: - Reduction in adjusted r-squared of more than 10 percentage points compared to previous model - Insignificant variables in model - Change of sign for explanatory variable coefficients - Not outperforming the as-is and through the cycle models in rolling regression.
Associated user story	User story 4: Automatic validation of existing models

Name	FR 14: Evaluating and deploying new models
Description	The user must be able to clearly see an overview of all the IFRS9 Macro Models, with information of which one has been changed and not, and at what time the change was done. The OLS parameters for each model must also be visible. The user can deploy the changes, and the program will convert the information into the format required by the new ECL model, and store the data in the directory system of the ECL model ready for use.
Associated user story	User story 5: Deployment of new models

Name	FR 15: Structure the use of the program into projects
Description	In order to have a proper structure to the inputs and outputs to the program, and to make it easy for auditors to retrace what input was used in the program, the program must have functionality to structure each round of IFRS9 macro model estimation into logical projects in the program. In these project folders, all the input to and output from the program will be stored. The program needs user functionality for creating and selecting these projects in an intuitive way, eliminating the need for the users to come up with a suitable folder structure.
Associated user story	All

Non-functional requirements

Name	NFR 1: Execution speed of program
Description	The perceived execution speed of the program must be instant. There should be no noticeable lag in the interaction between the user and the program. The sole exception being the flow for generating OLS model suggestions, as this flow, depending on the input from the user, can be somewhat time consuming. The flow for generating model suggestions will only have to be done a handful of times for each IFRS9 macro model being estimated, and it is therefore acceptable

	for this flow to have some execution time. Despite this, it should not take more than 10 minutes, assuming a sensible input to the functionality from the user.
Associated user story	All

Name	NFR 2: Flexibility of program
Description	As this program will interact with the new ECL model, it is important to make it sufficiently modular in order to avoid duplicating code and make it maintainable. Because there might be minor changes to the parameters in the future, it is important to avoid hard coding values. This is also important in order to avoid the need for the users of the program to have particular knowledge of the inner workings of the program or of Python in general.
Associated user story	All

Name	NFR 3: User friendliness of the program
Description	As the analysts using this program does not necessarily have knowledge of Python or coding in general, it should under no circumstances be required to alter program code in order to use the program for its intended purposes. Further it should not be required that users need particular knowledge and/or experience with the program in order to use it for its intended purposes. All needed should be general knowledge of statistics and a simple user guide. Only for future development of the program should it be necessary to have a deeper understanding of the program than what is described in the program's user guide.
Associated user story	All

Development process and redefined scope

The Covid-19 pandemic struck society midway during the development of this program. As the writer of this paper and the other subject matter experts related to this program in DNB work in financial risk analysis, and thus was needed for more urgent tasks in the bank, development was abruptly halted for a while. At the same time, the pandemic highlighted weaknesses in some IFRS9 macro

models, in that not all existing models behaved in an expected way given the macroeconomic environment. Developing this tool for use in re-estimating some of the underperforming models was therefore given top priority later in the corona period.

In order to satisfy the urgent needs of DNB, a new scope for the program was agreed upon. The priority was on the modelling itself, corresponding to User story 2, “Generate list of best OLS models”, and User story 3, “Do in-depth analysis and select model”. User story 5, “Deployment of new models” was still kept in scope as it did not require much work, while User story 1, “Generate credit cycles” and User story 4, “Automatic validation of existing models” was taken out of scope. The reasoning behind this is that it was rather obvious at the moment what IFRS9 macro models underperformed without the need for an automated way to detect this, and that generating credit cycles in the existing SAS program was trivial, while at the same time DNB started looking into if the logic behind it should be changed for the future. The inclusion of a web based GUI through Dash was also taken out of scope, as it was deemed unnecessary for the urgent needs of DNB, while at the same time this program was not given priority to use the new Dash functionality.

The scope for User story 3, “Do in-depth analysis and select model” was expanded with more tests and visualisations of the macro model performance forward in time, in order to best account for and avoid unexpected behaviour from the IFRS9 macro models going forward.

As a result of these changes, and experience gained during the development phase, the program differs somewhat in overall structure from the initial design outlined in the preliminary study.

Technical documentation

Techniques and technologies used in the program

This program uses several techniques and technologies to solve the business problem in the best way possible. The following is a brief description on some of these techniques.

Cloud computing

This program is deployed on DNB’s development platform IPA, running on Amazon Web Servers(AWS). This system is a cloud computing solution where DNB rent storage and computing power on demand from an AWS data center in Ireland. The main benefit of cloud computing is the scalability of the hardware resources. When running program on traditional in-house servers, the computing power is limited by the processing power of the server hardware, and an increase in

computing power requirements will have to be resolved by upgrading the in-house servers. This might be justifiable if the general demand for greater computing power in the company is high, but if this demand is only present in smaller parts of the organisation, such an expensive upgrade might be unjustified. With cloud computing on the other hand, this local increase in demand for processing power can be resolved by upgrading the instance running the particular programs demanding greater computational power. The instance this particular program is running on currently has a quadcore CPU, that fulfils the current requirements of the program. But if DNB wants to further develop the OLS suggestion module of this program, by far the most computational heavy functionality in this program, to do more advanced filtration, or if DNB wishes to expand the number of variables to input to the simulation, only having four cores available might lead to an excessive increase in processing time on the current instance. This can easily be resolved on AWS by upgrading the current instance, something that is arguably cheaper and most certainly quicker than upgrading in-house servers.

Use of map and lambda

Several places in this code a combination of map() and lambda functions are used to apply a function to an iterable. The basic syntax for this is:

```
results_list = list(map(lambda x: the_operation_to_be_applied, iterable))
```

The map function is a functionality that map all elements of an iterable, E.G. a list or a dictionary, to a specified function. In functionality it equates a regular Python for loop, but is used extensively in this program as it makes the code more compact without making it noticeably less readable. Another option would be to use list comprehension, which arguable is the Pythonic way to apply a function over iterables. It is claimed by some Python enthusiasts that list comprehension is quicker than map, which again is quicker than a regular for loop (Mamaev 2018). List comprehension was not used extensively in this program as it is the personal opinion of the author of this documentation that the syntax for list comprehension is not intuitive for those not particularly familiar with Python. As the parts of the code using map does not involve iterating over large objects where this increase in execution speed will make a noticeable difference, the focus was kept on having easily readable code. See more examples of how to use map() at w3school.

Lambda functions are small anonymous functions, and is an alternative to writing separate defined functions. These functions are useful when the function is only used once, or a limited number of times, where having a specified function would make the flow of the program more difficult to

follow, and where the functions are simple in nature, as lambda functions don't allow for complex logic. See more examples of the use om lambda expressions at w3schools.

Multiprocessing

By default, programs are executed serially. This means that each step of the program is executed one by one in the order defined by the program code. But in order to take advantage of the multicore and superscalar architecture of modern CPUs, the program must be instructed to process certain tasks in parallel. There are two common ways of doing this, either to use multithreading, or to use multiprocessing. Multiprocessing is where the operating system creates sub processes that operates in isolation, that it then distributes to the CPU cores for processing. Multithreading is similar, with the major difference being that threads in the same process do not operate in complete isolation like regular processes, in that they have access to a shared memory. The program can be instructed to distribute execution of certain parts of the code to individual processes or threads, and thus benefitting from multicore and superscalar CPUs. See Arpaci-Dusseau & Arpaci-Dusseau(2018) for further information regarding parallelization.

Because Python's memory management is not thread-safe, Python has a Global Interpreter Lock that prevents multiple threads from executing Python code simultaneously. The consequence of this is that multithreading on Python is not able to take full advantage of multiprocessing. See Python.org's description of Python GIL. For that reason it was decided to utilize multiprocessing to take benefit of the multicore CPUs available for this program.

In order to benefit from multiprocessing, the different processes must be fully independent from each other, and as there is some overhead for the operating system to initialize multiprocessing, the processing to be done must be of a certain volume. For these reasons, it was decided to only use multiprocessing on the most processing heavy part of this program, the part where a large number of time series combinations are regressed in order to find optimal OLS model. The number of combinations to be tried can exceed one million, and each combination regression is completely independent, making this suitable for multiprocessing.

Python file structure

The program code is located on three different Python files. "Macro_Model_Tool.py" and "Macro_Model_Tool_User_Interface.py" are located in a designated folder named "Macro_Tool", located under the path "code/Modelling". "Macro_Model_Tool_User_Interface.py" contains the logic for the user to interact with the program, and to set up estimation projects, while "Macro_Model_Tool.py" contains the main program code itself. The program also uses functionality

in “Dataprep_Functions.py” located together with the ECL model. The functionality of the program stored in “Dataprep_Functions.py” is the logic that validates, normalize, and transforms the variable time series. Despite being developed as part of this program, it was decided to integrate this functionality with the program code of the ECL model to ensure the Macro tool program and the ECL model have the same validation and transformation logic. This will minimize the consequences if there has been a misunderstanding as to how the transformations should be implemented.

There are certain dependencies concerning the directory structure around the program. To avoid issues, it is important to keep these relative paths unchanged:

- Path from main program to ECL Dataprep program: ../Dataprep
- Path from main program to Macro tool data: ../../../data/Macro_tool_data
- Path from main program to folder with estimation projects: ../../../data/Macro_estimation_projects

If the directory structure must be changed for some reason, make sure to thoroughly test the program after. Also make sure that the different parts of the ECL program have the same relative path to the folder ECL data, where both programs collects information using the same overview of paths. If directory structure is changed, make sure to change the paths in the parameter variable “default_path_dict” of the object of class User in “Macro_Model_Tool_User_Interface.py”. Also change the path to the file with the default paths for the entire IFRS9 project. This is given in the variable “default_paths_path” in the function “get_default_file_paths()” in “Macro_Model_Tool_User_interface.py”. The functions “create_macro_project()” and “set_macro_project()” in “Macro_Tool_User_Interface.py” both uses paths dependent on the current structure of the project. For “set_macro_project()” it is only the variable “macro_projects_path” that must be changed, while for “create_macro_project()” multiple alterations to the code will have to be done.

Data structure and input files

It was decided to use excel files as input and output to and from the program. This was done mainly because the future users of the program are very used to working with and in excel sheets, and most of the input to the program is already stored in such files. The amount of data going into the model is not extensive, so it was decided not to use csv files in order to speed up loading into the program, as this would lead to the files being less readable for humans.

There are two main storage folders for the IFRS9 Macro model tool. Both are located in the folder ‘data’. These are “Macro_tool_data” and “Macro_estimation_projects”.

“Macro_tool_data” is the main data storage for the program itself. It contains the following subfolders:

- Analysis_output_template – containing the excel template the program uses to write out the results from the in-depth analysis. It also contains a folder with old templates. It is important that the name of the template is “OLS_results_template.xlsx” if the template is changed in the future. Make sure the updated version harmonizes with the function “write_in_depth_results_to_excel()” in “Macro_Model_Tool.py”. In order to get dynamically updated graphs accounting for different lengths of time series in the template, option two of this guide is used: Bansal.

- Macro_data – containing the macro data input to be used in the OLS regression. There can be multiple macro files in this folder, and there are no special naming restrictions. When setting up a new macro estimation project, the user setting up the project will be able to select what macro data file to use in that particular project. The program requests three scenarios as input when setting up a new macro estimation project, baseline, low and adverse. Make sure to have a folder for each of these scenarios in the input macro sheet.

- OLS_simulation_input – containing the variable input data for use in OLS model suggestion module. There can be multiple variable input files, and there are no special naming restrictions. When setting up a new macro estimation project, the user setting up the project will be able to select what variable input file to use in that particular project. The selected file must contain one sheet for every macro model that will be part of the project. Each sheet must contain the headers “Variables”, “Inflation adjust”, Ln adjust”, “Expected effect”. For each variable that might be included for the particular IFRS9 macro model, the variable code must be stated, Inflation adjust must be stated(Nor, US, or nothing), Ln adjust must be stated(Y or N), and Expected effect must be given(+ or -).

- Segment_Z – containing the excel files with the Z values to be modelled. There can be multiple segment Z files, and there are no special naming conventions. When setting up a new macro estimation project, the user setting up the project will be able to select what segment Z file to use in that particular project. The selected file must contain one sheet for every macro model that will be part of the project. The selected file must contain the name “Date” in cell A1 and the name “Z” in cell B1. The observation dates and the calculated Z

values must be in the rows following this.

- Suggestion_output_template - containing the excel template the program uses to write out the results from generate OLS suggestions. It also contains a folder with old templates. It is important that the name of the template is OLS_suggestions_template.xlsx if the template is to be changed in the future. Make sure the updated version harmonizes with the function "write_OLS_suggestions_to_excel ()" in "Macro_Model_Tool.py".

"Macro_estimation_projects" is the folder containing the macro estimations projects user will create before estimating new OLS parameters for the IFRS9 macro models. The program will create a subfolder in this folder with the name given by the user creating the project. The program will then create these subfolders with the stated excel files:

- OLS_analysis results – containing a sub folder for each IFRS9 macro model that has been tried modelled in the project. These sub folders again contain the output from the in-depth analysis of that particular IFRS9 macro model based on the template "OLS_results_template.xlsx. The name of the output will be generated automatically by the program, and contains the input variables and a timestamp in order to ensure each file name is unique.

- Project_model_coefficients – containing the files "Changed_models.xlsx" and "Macro_models.xlsx". "Changed_models.xlsx" contains an overview of the models that have been changed in this project. This file is automatically generated and updated by the program. "Macro_models.xlsx" contains one sheet per IFRS9 macro model, with the OLS parameters "variable", "variable transformation" and "coefficient" for each model. When project is created, the program will load the current OLS parameters from the ECL model. When the users finds a suitable new OLS model for an IFRS9 macro model and save these OLS parameters, the corresponding sheet will be updated in "Macro_models.xlsx" and the file "Changed_models.xlsx" will be updated.

- Project_parameters – containing the parameters that will be used in the Macro model tool. The files in the folder are "Macro_data.xlsx", "Macro_sheets.xlsx", "OLS_results_template.xlsx", "OLS_suggestions_template.xlsx", "Simulation_input.xlsx", and "Z_values.xlsx". All of these files are automatically loaded into this folder when project is created, and it is important that the file names stay unchanged. "Macro_data.xlsx" contains

the macro data to be used in the program. “Macro_sheets.xlsx” contains the mapping for the scenarios to be used in the in-depth analysis to the sheets in “Macro_data.xlsx”. “OLS_results_template.xlsx”, which is the output template for the in-depth analysis. “OLS_suggestions_template.xlsx”, which is the output template for the generate OLS suggestions. “Simulation_input.xlsx”, which is the overview of what variables that can be included in the generate OLS model suggestions. “Z_values.xlsx containing the Z_values to be modelled in the project.

- Suggested_OLS_models - containing a sub folder for each IFRS9 macro model that has been tried modelled in the project. These sub folders again contain the output from the generate OLS suggestions of that particular IFRS9 macro model based on the template “OLS_suggestions_template.xlsx. The name of the output will be generated automatically by the program, and contains a timestamp in order to ensure each file name is unique.

The program also uses the file “Segment_info.xlsx” from the ECL model, the file containing the current macro model parameters used in ECL model, and the file containing metadata used to validate and normalize the macro data. These are located together with the input data for the ECL model. See the technical documentation for the ECL model for further information.

Python code documentation

This code documentation focuses mainly on showing the logic and flow of the implemented code, and highlighting considerations taken during development. The code itself is extensively commented, and it has been a strong focus on having reasonably readable code with descriptive names of classes, variables, and functions. In the flow diagrams showing program logic, steps coloured blue have designated flows elsewhere in the documentation detailing how that particular process is implemented. This documentation covers little relating to the data structures in the program, as this is specified clearly in the docstrings of each function, where it states format of input and output. For this reason, it was decided to not include diagrams like class diagrams.

In the design of the program, great emphasis has been put on splitting the program into logically separate pieces to have separation of concerns and enabling the different program flows to utilize common functionality. Examples of this include the separation of the parts of the code enabling and validating user input and the parts of the code executing the different flows, and splitting the program functionality into designated functions with clearly defined inputs and outputs. Designing

the program in this way enables much easier maintenance and further development of the code, as the dependencies in the program flow is clearly defined by the different functions, and common functionality only has to be changed one place. This makes it significantly easier to debug faulty code and enables expansion of the program without running the risk of interfering with existing functionality compared to the approach of creating monolithic applications where entire flows are combined into one single non-modular instruction flow.

Python interpreter and Libraries/packages used in program

The Python interpreter used for this program is Python 3.6.10, the current default interpreter on DNB's development platform.

These are the packages used in this program. For the program, a virtual environment has been created where these packages have been included. If this environment is updated, it is important to test the functionality of the program before putting the new virtual environment into production, as there might be issues concerning the versions of the different libraries. All of these packages have been installed on the DNB development platform itself, and is therefore deemed safe and robust by its administrators.

Name	**Version**	**Link to documentation**
Pandas	1.0.3	https://pandas.pydata.org/
Statsmodels	0.11.1	https://www.statsmodels.org/stable/index.html
Numpy	1.18.4	https://numpy.org/
Itertools	3.6.10	https://docs.python.org/2/library/itertools.html
Multiprocessing	3.6.10	https://docs.python.org/2/library/multiprocessing.html
Time	3.6.10	https://docs.python.org/3/library/time.html
Statistics	3.6.10	https://docs.python.org/3/library/statistics.html
Functools	3.6.10	https://docs.python.org/3/library/functools.html
Scipy	1.4.1	https://www.scipy.org/
Shutil	3.6.10	https://docs.python.org/3/library/shutil.html
Openpyxl	3.0.3	https://openpyxl.readthedocs.io/en/stable/
Os	3.6.10	https://docs.python.org/3/library/os.html
Sys	3.6.10	https://docs.python.org/3/library/sys.html
Warnings	3.6.10	https://docs.python.org/3/library/warnings.html
Xlrd	1.2.0	https://pypi.org/project/xlrd/

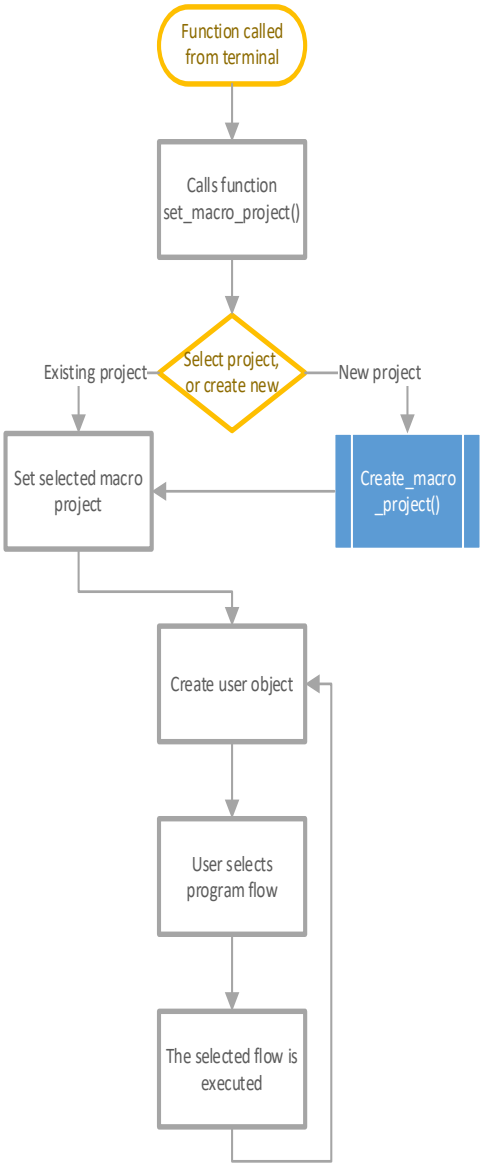
Re	3.6.10	https://docs.python.org/3/library/re.html
math	3.6.10	https://docs.python.org/3/library/math.html

User interface and project administration

The program is structured around working in projects a user creates at the start of a new round of estimations of IFRS9 macro models. When inside a project, the user interacts with the program through functions in the class User, which again calls the main logic of the program in "Macro_Model_Tool.py". This is done to separate user interface and program logic. The project folder contains all input files used, making it easy to see what parameters have been used in a future audit.

Overall flow

The program is launched by calling the function "main()" in "Macro_Model_Tool_User_Interface.py". In that function, this is the overall flow:



This entire flow is inside an infinite loop, and then again inside a try-except. This is in order to handle unexpected exceptions in the program.
The user is prompted to select a macro project to work in, or create a new one.

See designated flow for creating new macro project. When new project is created, the user is navigated into this new project automatically.

When user has selected or created a macro project, an object of class User is created. A dictionary with paths to required files and folders is automatically created.

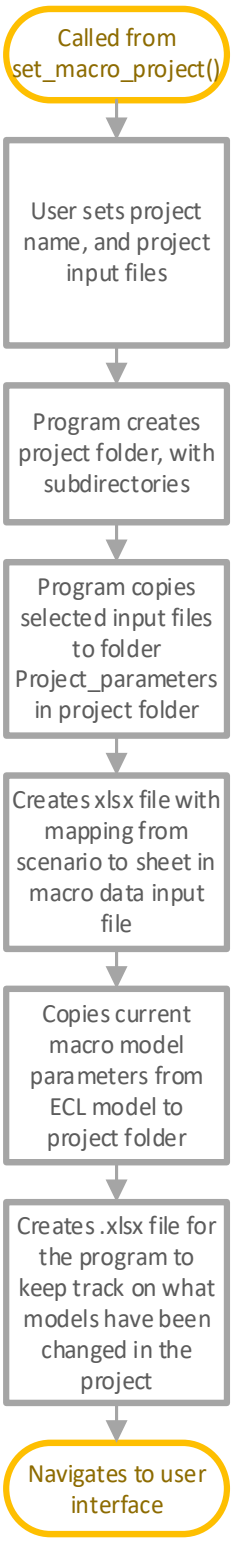
The user can select what program functionality to use from a list.

Based on user selection, the function in the user interface corresponding to that selection is called. The separate functions takes the required input from the user, and calls its designated function in “Macro_Model_Tool.py”. The logic used to get the required input from the user should be fairly easily understood from the program code. See separate flows for the designated functions in “Macro_Model_Tool.py” under.

The flow in-depth analysis gives the user the option to call the function save_model_results() for saving the model parameters for the specified model for later deployment to the ECL model. This function should be easily understood, and hence is not described specifically in this documentation.

The flow for deploying the new IFRS9 macro models to the ECL model calls the function deploy_new_models(). This function should be easily understood from the code, and hence is not described in this documentation. See user guide for further details on this flow.

Create macro project



User is asked to specify name. The program checks if an existing project has the same given name. The user is asked to specify all input files to the program. These are Macro forecast file, segment Z file and file with OLS simulation input. The user is also asked to specify what sheets in the macro file corresponds with the three scenarios; baseline, low and adverse.

The project creates a folder with the given project name, and the subfolders: OLS_analysis_results, Suggested_OLS_models, Project_parameters, and Project_model_coefficients.

Program uses shutil.copy() in order to copy the specified files into the project subfolder Project_parameters. The two templates for output to excel are also copied to this folder.

Program creates an excel file with the mapping from the three scenarios to three sheets in the macro data input file and saves to project subfolder Project_parameters. The file has the headers “Type” and “Name”. The three types “Baseline_sheet”, “Adverse_sheet” and “Low_sheet” are mapped to names of sheets in macro data input file.

File containing the current IFRS9 macro models in use in the ECL model is copied into the project subfolder Project_model_coefficients.

Program creates an excel file to keep track on IFRS9 macro models that have been changed in the project. The excel file has three headers, “Model”, “Changed”, and “Date”, and one row for each IFRS9 macro model specified under “Model”. “Changed is set to “No” for all rows.

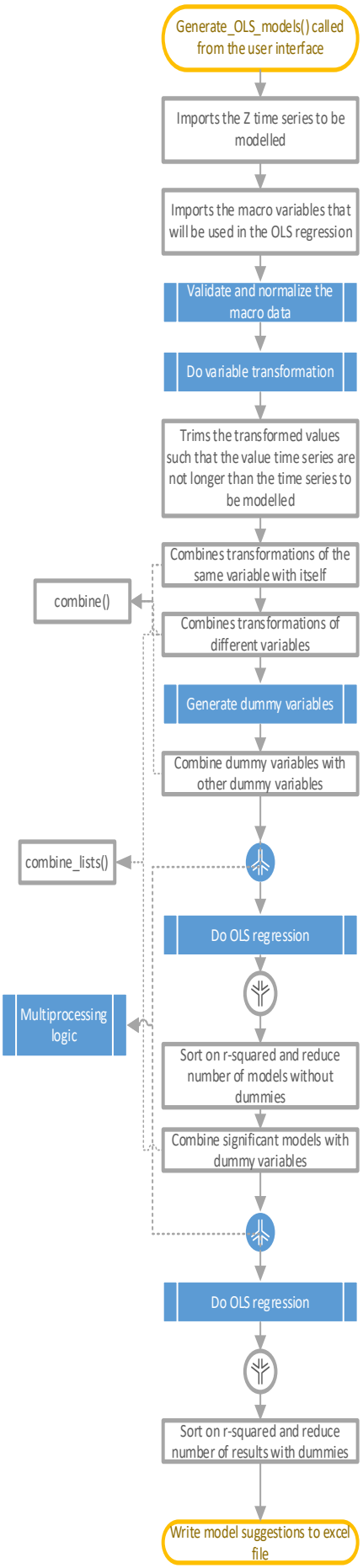
Generate model suggestions

The module to generate proposed OLS models is called from the user interface with the required parameters. As there is a limit on processing power, several measures had to be taken in order to get the processing time down to an acceptable level. These includes utilizing multiprocessing where parallelization was possible, setting threshold values for the OLS regression in order to limit the overall number of regressions that have to be done, and have a program flow where all the combinations first were tested without including dummy variables, and thereafter only adding dummy variable combinations to the combinations that showed promise after that. This means that dummy variables will only be added to combinations that had OLS parameters above the threshold set for adjusted r-squared and below the threshold set for significance level.

These alterations to the original plan significantly lowered processing time from weeks to below the threshold of 10 minutes set by DNB when a sensible number of input variable and transformations are used.

To make combinations to be tested in OLS regression, two special functions were written to avoid duplication of code. combine() takes a list of variables to be combined, and depending on the input parameter max_combinations, combines the elements in the list to all possible combinations using the function combinations() from the itertools package. Combine_lists() takes a list of lists as input, and combines the different elements of the lists with each other in all possible ways by using the function product() from the itertools package. Both of these functions should be easy to understand by reading the code, and therefore does not have dedicated visualisations of their flows.

Overall flow



The function is called with parameters specified in the function docstring in the code.

Z to be modelled is imported from file in project parameter folder.

The macro variables to be used in modelling is imported from file in project parameter folder.

The macro data to be used is validated and normalized. Se separate flows under *common functionality*. If errors are found, program execution is halted.

All specified transformations for each specified variables are done. Se separate flow.

As Z time series might be shorter than variable time series, time series are shortened such that only observation dates that are in Z for the different variable time series are kept.

Calls function combine() in order to combine transformations of the same variable with itself.

Calls first combine() to combine the different variables, and calls combine_list() to combine the variable transformations of all variables.

Generate dummy variables for the Z time series. See separate flow.

Combine dummy variables with other dummy variables by calling combine()

In order to speed up processing, multiprocessing is done. See separate flow.

OLS regression is done with all the combinations of variables, but without dummies for the first round of regression. See separate flow.

The list with OLS models passing the tests in the OLS function is sorted on r-squared and capped to 1000 models. This was done to reduce processing time, and after conferring with future users.

The list of successful models is combined with list of dummy combinations.

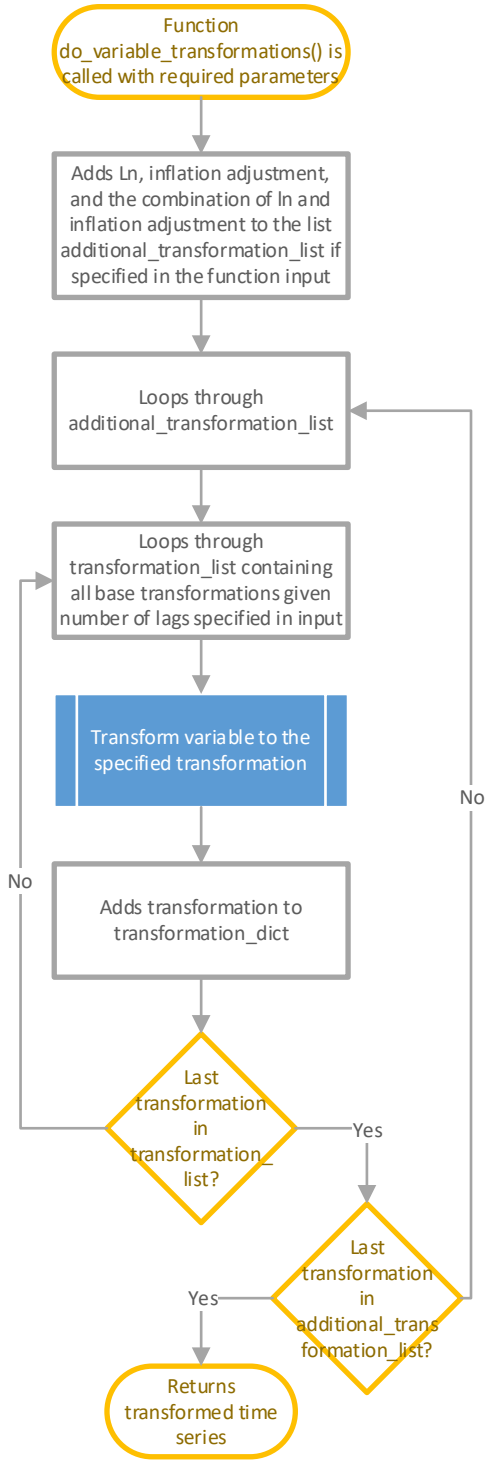
Multiprocessing is done for second round of regression.
See separate flow.

OLS regression is done for all models passing first round of regression, and that was not cut in the following step. See separate flow.

The list of models passing the second round of regression is sorted by r-squared and capped to 10000 models.

The results are written to excel by calling write_OLS_suggestions_to_excel(). This code should be unproblematic to follow.

Do variable transformation



do_variable_transformations() is called with parameters specified in the function docstring.

Based on information in simulation input file in project folder Project_parameters, a list is filled with Ln, inflation adjustment codes and “”. “” represent no Ln or inflation adjustments of the other transformations.

Loop through all elements in list created in previous step.

Given the lag-input, the program loops through all base transformations(Pct1, Lag1, Movavg2, etc.).

For each combination of the two lists, the program calls transform_variable() in class Ols_parameter_transformation in Dataprep_functions.py. See separate flow under Common functionality.

Each transformed time series is added to a dictionary with key=transformation code, value=transformed time series.

Continuous the nested loop through both lists.

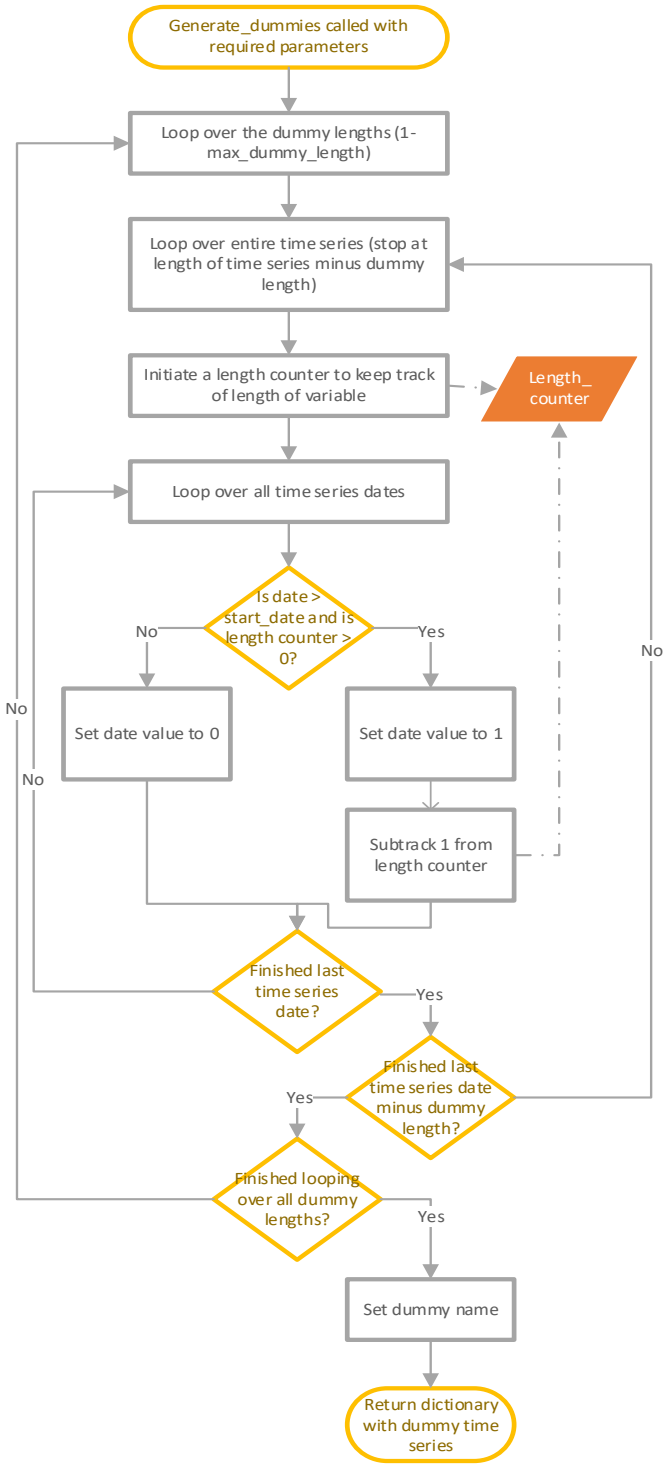
Returns a dictionary with the transformed time series for all the variables.

Generate dummy variables

This function generates all possible dummies based on the Z to be modelled, and based on maximum length of dummy variables(defaulted to five years). This function has three nested loops, where the

first loop iterates over all possible dummy lengths based on the max_dummy_length input, and thereafter loops over all possible starting dates given the dummy length. Lastly the program loops over the entire timeseries of Z to actually generate the dummy time series.

This is the flow of the function:



generate_dummies() is called with parameters specified in the function docstring.

Loops over all the dummy lengths specified in the input. So first generating dummies with length one year, thereafter dummies for two years, etc.

Loop over all observation dates minus length of dummies, to find all possible start dates given the dummy length.

Initiate a length counter to keep track of how many more years to add to dummy time series.

Loop over entire time series in order to generate the actual dummy time series for the particular dummy variable.

If date in last loop is over start date identified earlier, and if the length counter is above 0, the value 1 is added to dummy time series, and 1 is subtracted from the length counter. If these conditions are not true, meaning date is before start of dummy period or the particular length of dummy har already been created, 0 is added to the dummy time series.

Continuous to loop through the three nested loops.

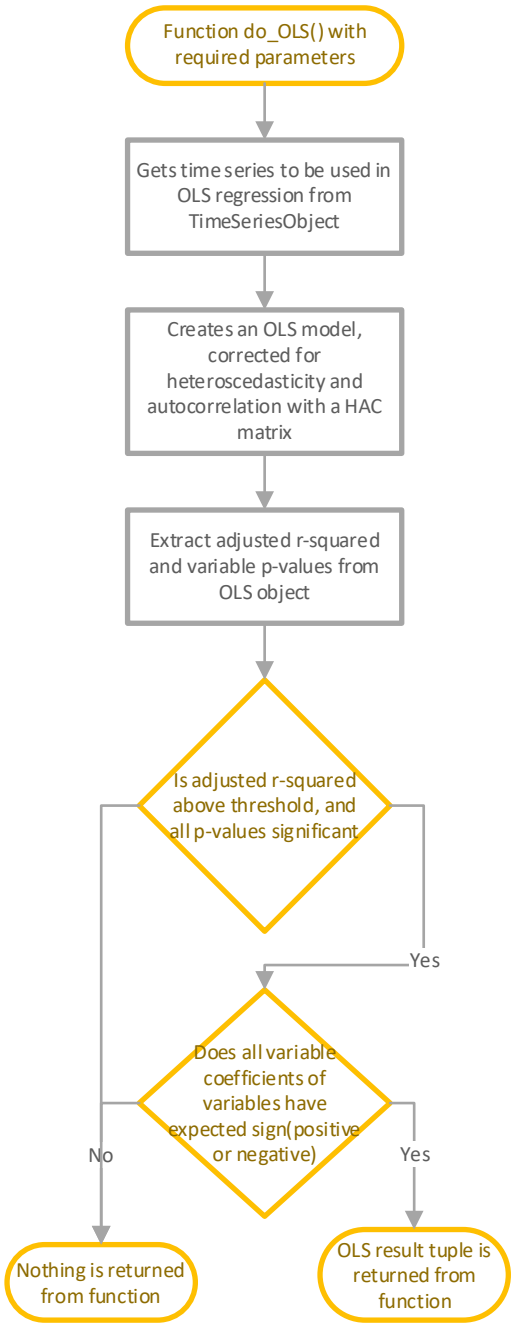
Dummy name is set. If start and end date are the same, this variables becomes the dummy name. If they are different, “start_date – end_date” becomes the variable name.

A dictionary with key = dummy name, and value = dummy time series is returned.

Do OLS regression

This function is where the actual OLS regression on the time series from earlier in the program is done. If the time series for each individual regression was added to the tuple that is an input to this function, the program had major memory issues because of the sheer number of combinations to do regression simulations on. To get around this issue, the class TimeSeriesLookup was created. An object of this class is created before the OLS simulation, and a pointer to it is added to the input to this function. This object contains all the different time series that will be included in the OLS simulation, Z, variable and dummy time series, and by calling the class function time_series() with a tuple of the variables to be used in the OLS simulation, the function trims the time series to the shortest of the time series for that particular regression, and returns them.

This is the main flow of the function:



The function do_OLS() is called with the parameters described in the function docstring.

Gets the trimmed time series to use in OLS regression by calling the function time_series() on the TimeSeriesLookup-object.

Transforms the time series into the format required by the OLS function, and calls the function OLS() from the package Statsmodels to create an OLS model. Corrects for heteroscedastic and autocorrelation by calling the function get_robustcov_results() from Statsmodels.

Extracts adjusted r-squared and p-values of explanatory variables from OLS object returned by the Statsmodels package.

Checks if adjusted r-squared is above the defined threshold, and check if all explanatory variables are significant based on the significance level defined.

Checks if all variables, not dummies, have the expected coefficients (positive or negative) as defined in the simulation input file stored in the project folder Project_parameters.

If the model passes the above tests, a tuple containing OLS results are returned. If the model does not pass, nothing is returned from this function.

Multiprocessing logic

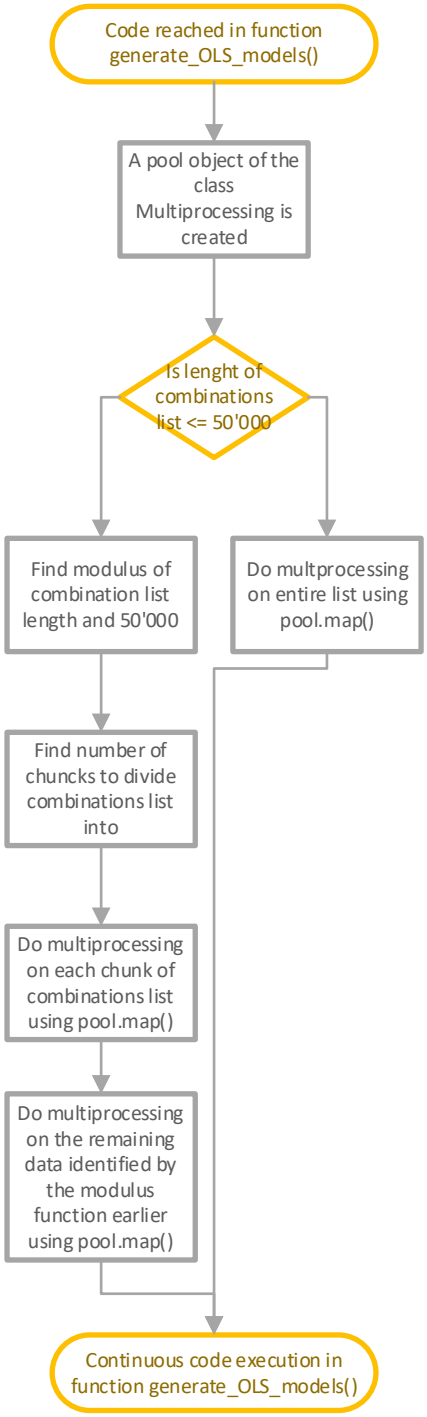
As Python has a global interpreter lock (GIL), using multithreading for parallelism would limit the computer to only using one CPU, despite having a multi-core setup. For that reason multiprocessing is implemented for parallelisation in this program. See chapter on Multiprocessing under Techniques and technologies used in the program for further information on this topic.

It would require a disproportionate amount of overhead for the computer if very big iterables are passed to the multiprocessing-functionality. For that reason, the list of combinations to be passed to

the multiprocessing-logic is divided into chunks of maximum 50’000 elements. This size is the result of timing of different chunk sizes. This increased execution speed considerably.

This logic is not contained in a designated function, but is a part of the function generate_OLS_models() two separate places.

The main flow of the multiprocessing logic is:



Create a pool object from the class Multiprocessing in the package with the same name.

Checks if length of combinations list is above the chunk size of 50’000, and therefore if it needs to be divided into chunks before doing multiprocessing.

If dividing into chunks is not required, multiprocessing is done on the entire combinations list. If it has to be divided before doing multiprocessing, (length of combinations list)mod(50’000) must be found to make the remaining list divisible by 50’000.

Find our how many chunks to divide the combinations list into before doing multiprocessing.

Does multiprocessing with each chunk of the combinations list.

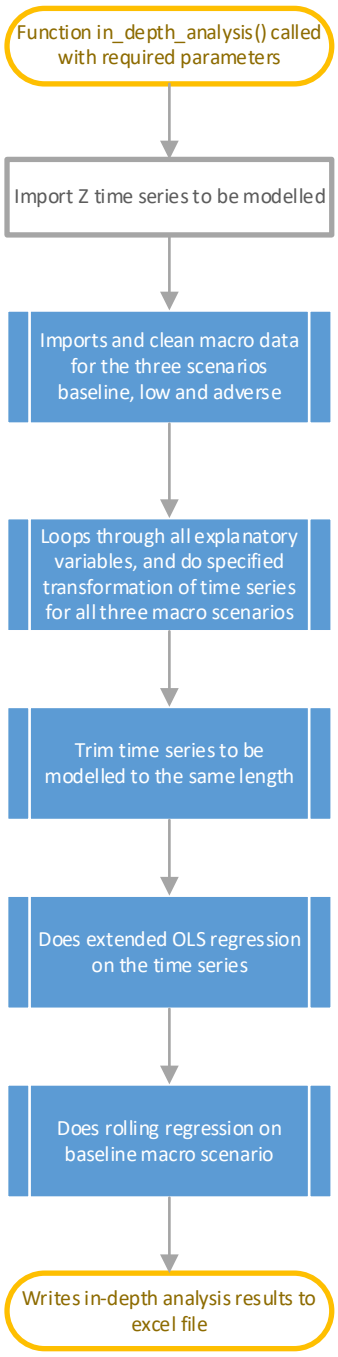
Does multiprocessing on the remainder of the combinations list. On the part of the list not divisible by 50’000.

In-depth analysis

The in-depth analysis returns in-depth statistical information of a chosen OLS model for a given IFRS9 macro model. The statistical information this will write to an excel file is:
adjusted r-squared, mean absolute error of model, root mean squared error of model, KPSS and ADF of residuals, Durbin Watson test statistics, Jarque-Bera normality test statistics, Omnibus normality test statistics, Lillefors' test statistics, Breusch-Pagan Lagrange Multiplier test for heteroscedasticity, MAE and RMSE of rolling regression for model and the challenger models as-is and TTC, p-values and coefficients of explanatory variables, KPSS and AFD for explanatory variables time series, year on year coefficients of explanatory variables from rolling regression, all time series used in OLS regression, predicted historical Z for baseline scenario with 95% confidence interval, predicted future Z for baseline, low and adverse scenario, and predicted Z from rolling regression with 95% confidence interval.

In order to simplify storage of data, the custom class OLS_result_object is used. An object of this class has get and set methods, for an organized way of storing and retrieving data during calculation and during writing the results of the in-depth analysis to excel.

Overall flow



Function in_depth_analysis() in Macro_Model_Tool.py is called with parameters described in the function docstring.

Calls custom function get_Z_values() to retrieve Z time series to be modelled.

Imports macro data from all three scenarios, and validates it using the function Check_Macro_Input() in Dataprep_Functions.py. See separate flow under Common Functionality in this documentation. If errors are found, execution of program is stopped. If no errors, normalization of the data is done, see separate flow under Common Functionality.

Loops through all explanatory variables. If variable is not dummy, transformation of macro data time series is done by calling transform_variable() in class Ols_parameter_transformation in Dataprep_functions.py. See separate flow under Common functionality.

Time series of all macro scenarios are trimmed to the same length by calling function trim_time_series(). See separate flow.

Calls function do_extended_OLS() to generate OLS regression statistics. See separate flow.

Calls function rolling_regression() to get rolling regression statistics. See separate flow.

Calls function write_in_depth_results_to_excel() to write OLS results to a copy of the excel template OLS_results_template stored in the project folder under Project_parameters. This code should be easy to follow as it just extracts data from the OLS_result_object object and writes it to cells in the excel file through a combination of direct cell references and loops using offsets.

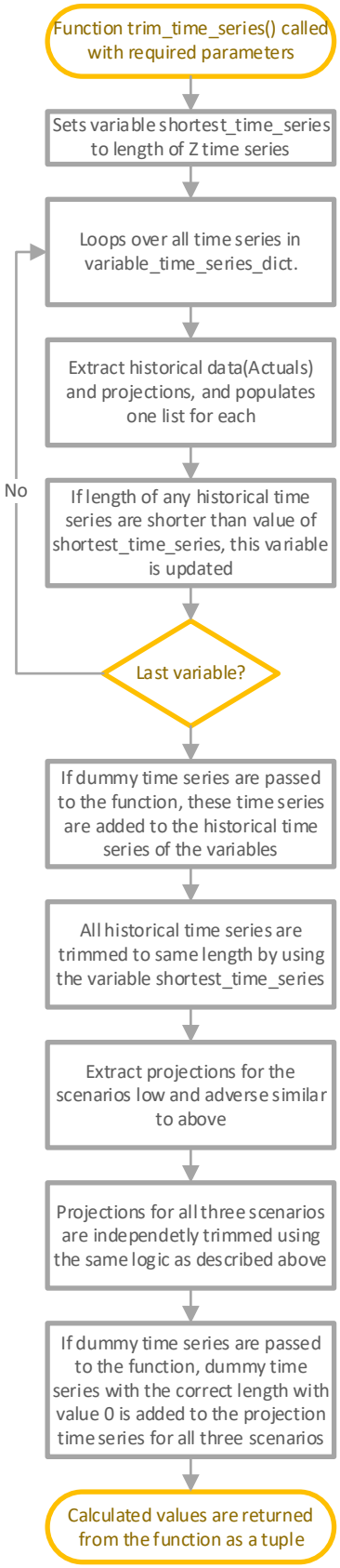
Trim time series

Despite the flow generating OLS model suggestion has functionality for trimming time series to the same lengths in order to be able to do OLS regression, it was decided to make dedicated functionality

for this for the in-depth analysis flow. The reason is that the data to be handled and the requirements relating to execution speed differ to such an extent that to make common functionality to satisfy the requirements for both flows, would require complex code potentially not satisfying the requirements of either flow satisfactory and potentially being hard to understand and maintain for future development of the program. While the functionality for trimming time series is executed inside the OLS function in the flow for generating OLS models when fetching variable time series, the trimming of the time series must be done before calling the OLS function in the flow in-depth analysis.

This function was subjected to multiple rounds of rewriting during development, stemming from new requirements to the output of the entire flow deemed necessary in light of the corona situation. For that reason, and a general lack of time and unwillingness to risk breaking the parts of the function proven to be working and thereby jeopardizing the work being done in estimating new OLS parameters for one of DNB's IFRS9 macro models, the program features some duplication of code not aligning with best coding practice. Under Suggestions for future improvements, it is recommended to refactor the code in this function to avoid code duplication and generally making it more readable.

This is the flow of the function:



Function trim_time_series() is called with parameters specified in function docstring.

Sets a variable with integer value equal to length of Z time series. This variable will be used to shorten time series to same length.

Loops over all time series in variable_time_series_dict, representing the baseline macro variable time series.

Extract historical and projected data into separate lists. This is done by using the strings "_Actuals" and "_Projections" stored in a tuple with the time series value of that specific observation date.

Finds the shortest time series. Compares the length of each historical time series to integer stored in variable shortest_time_series.

Loops until last macro variable time series.

If dummy variables are passed to the function in dummy_dict, these time series are added to the list of historical time series.

All historical time series are cut to same length by using the variable shortest_time_series and cutting the start of the time series. It is assumed that the time series have the same end date, but can have different starting dates.

Extract the projection time series for all three macro scenarios using the same logic as above.

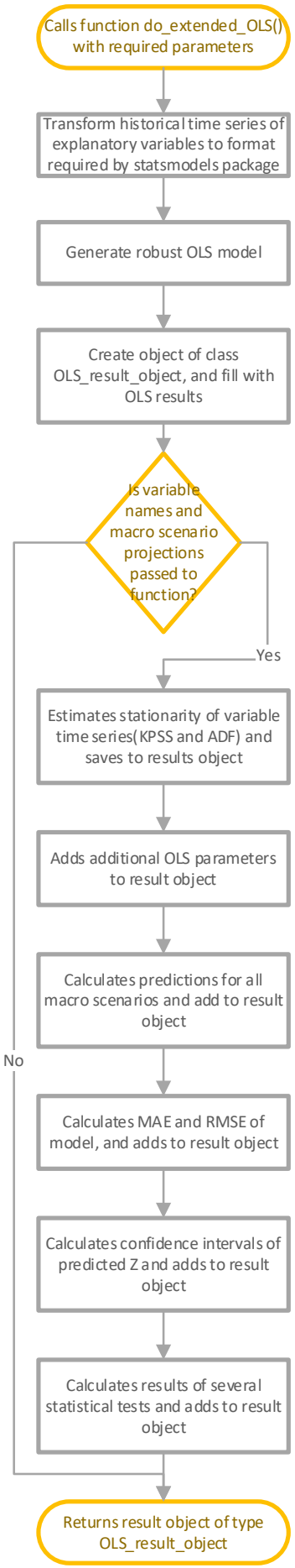
Projection time series are independently trimmed to the same length using a similar logic as above. This time the end of the projections are trimmed, as it is assumed that projections have the same start date, but can have different ending dates.

If dummy time series are passed to the function, projected dummy time series are added to projected time series with value 0, as these will have no effect on projections. As we know length of projected time series, the program adds the correct length of dummy time series.

Do extended OLS regression

Despite the flow for generating suggested OLS models also has a function for estimating OLS parameters, it was decided to create a dedicated function for the in-depth analysis flow for the same reasons as it was decided to create a dedicated flow for trimming time series. Having a separate function for this ensures the functions are optimized for the individual flows, and makes the code much more readable and easy to maintain and expand. This function was made flexible enough to be able to use in a simplified way for rolling regression in addition to the more extensive analysis required to generate all necessary statistical parameters.

This is the overall flow of the function:



Function do_extended_OLS() is called with parameters specified in the function docstring.

The variable time series are transformed into a numpy array, transposed and a constant is added.

Estimates model first by calling function OLS(), and then by calling function get_robustcov_results() to correct for heteroscedastic and autocorrelation.

Creates result object and store model object, model coefficients, Z time series and explanatory variables time series to object.

Checks if these parameters are passed. If not, no further analysis is done on the model, and result object is returned from function. If the parameters are passed, more extensive analysis is done.

Loops through explanatory variables and calculates stationarity statistics(KPSS and ADF) of time series. Adds this to result object.

Adds adjusted r-squared, r-squared, p-values, residuals and baseline historical predictions to result object.

Calculates future predictions for all macro scenarios and adds to result object.

Calculates mean absolute error(MAE) and root mean squared error and adds to result object.

Calculates 95% confidence interval of historical predicted Z, and adds to result object.

Calculates statistics from the tests Durbin-Watson, Jarque-Bera, Omnibus, Breusch-Pagan and Lillefors, and adds to result object.

Returns result object.

Do rolling regression

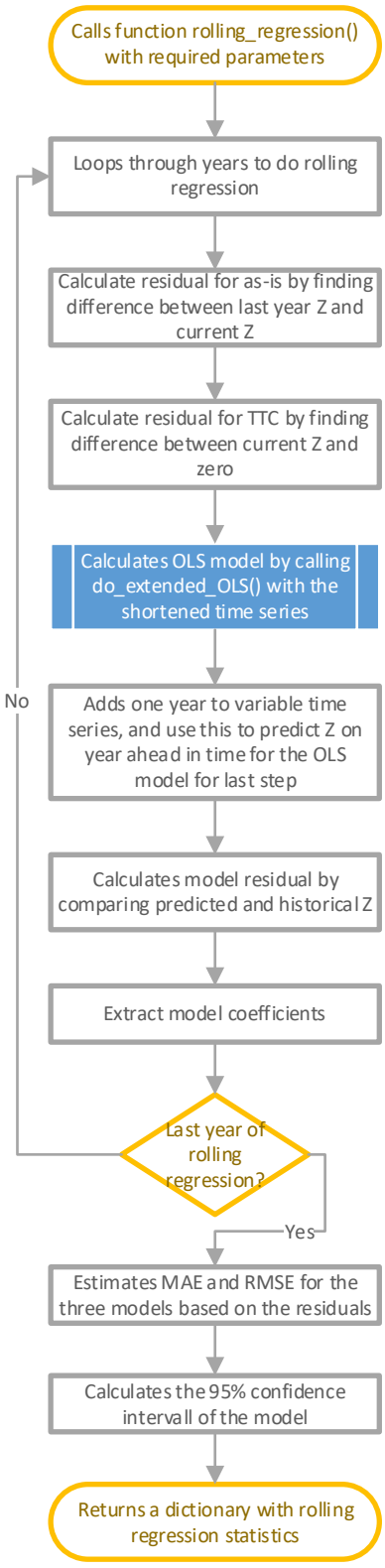
Rolling regression is a way to determine the forecasting properties of the model over time. It does ten separate OLS regression on an ever-increasing time series length in order to see how well the OLS regression of the shortened time is able to predict one year ahead in time. So it start by shortening the time series by 11 years to see how well it predicts the Z 10 years ago, then shortens the time series by 10 years to see how well it predicts the Z 9 years ago, etc.

The forecasting properties of the two simplified challenger models as-is and TTC are also calculated. As-is assumes Z one year ahead in time is equal to the current Z, while TTC assumes no cyclicality of Z, and therefore that all future predictions of Z are zero.

Mean absolute error(MAE) and root mean squared error(RMSE) of all three models for the entirety of the rolling regression period is calculated for the user to compare out of sample model performance and forecasting properties. Coefficients of the tested model are also saved, for the user to analyse if they fluctuate in an unpredictive way.

The statsmodels package has functionality to do rolling regression, but it did not satisfy all requirements of DNB. For that reason, it was decided to create a custom function for this, that also can be extended in future iterations of the program.

This is the overall flow of the function:



Function rolling_regression() is called with parameters specified in function docstring.

Years to do rolling regression is specified in input. Loops downwards from this integer to 1.

As-is model assumes next year will be equal to current year. Residual for that year is found by the difference in current Z and next year Z.

TTC model assumes no cyclicality in Z. Residual for that year is found by the difference between zero and next year Z.

Calls function do_extended_OLS(). As no prediction variables are included, only a simplified version of the function is done. See separate flow.

Use model from last step to calculate predictions for next year by passing time series with one additional observation date and fetching last predicted Z.

Calculates residual by the difference in predicted and actual Z.

Extracts and saves coefficients from estimated model

Loops through all years of rolling regression

Estimates MAE and RMSE for all three models based on the estimated residuals.

Estimates 95% confidence interval for the specified model.

Returns a dictionary with rolling regression results.

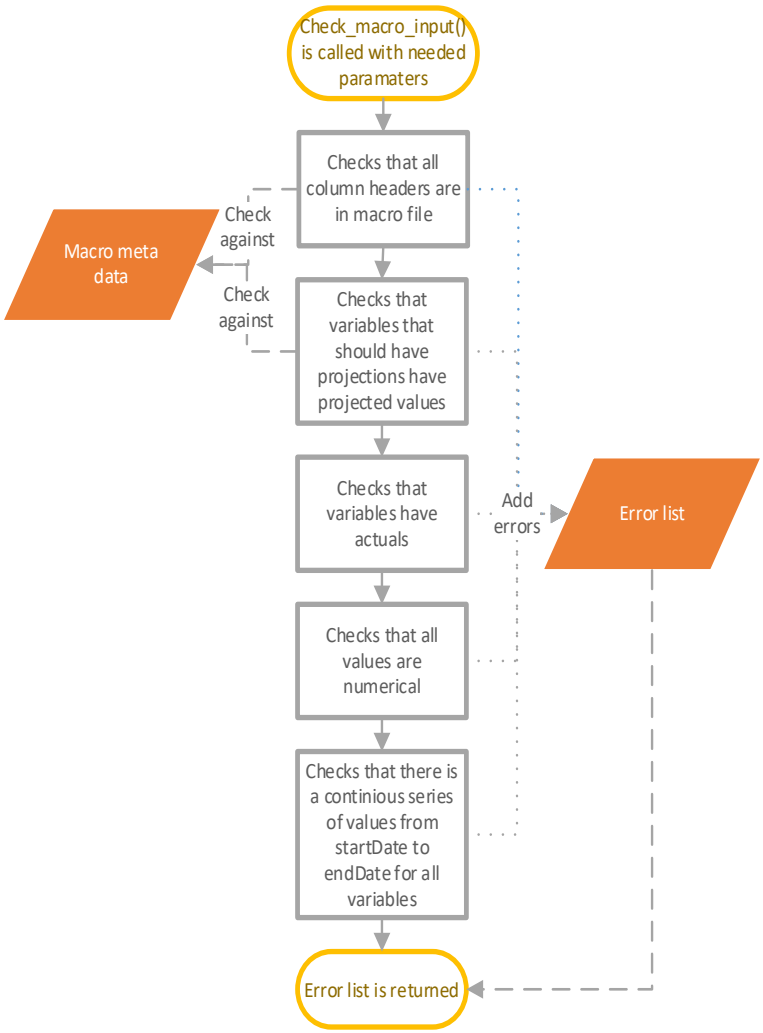
Common functionality

This is general functionality used by multiple program flows.

Validate macro data

This step aims to stop execution of program if obvious errors in the macro file input are found, and returns a list of errors that should aid the user as to how to fix the macro file. This validation will not detect all possible errors in the macro input file, and should be improved on as DNB gains experience into what are typical errors not detected in current program. It is also recommended to extend the functionality to also fix minor, predictable errors in the input macro file, see more on this under "Suggestions for future improvements". To validate the input, the program uses an excel file stored on the ECL model with macro metadata. This file contains a list of all the expected variables in the macro input, and information on what is expected of historical and projected data for each variable. This program code was incorporated into the code of the new ECL model, and is stored in Dataprep_Functions.py. To follow the structure of the new ECL model, this functionality is organized in the class Datavalidation.

This is the flow of the validation:



The main function of the class is called with parameters specified in the function docstring.

Program loops through all variables in macro input file and in macro metadata file, and compares.

Checks if variables that are expected to have projections, based on metadata file, actually have projections.

Checks if variables that are expected to have historical data, based on metadata file, actually have projections.

Checks that all data values are numerical.

Checks if there is a continuous time series from first data of historical data to last date of projected data, to avoid non-continuous time series.

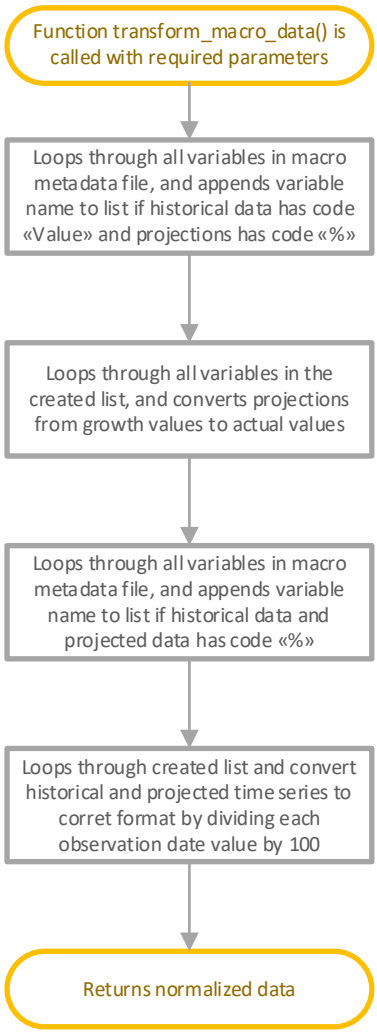
Returns error list. If no errors are found, this will be empty.

Normalize macro data

In accordance with DNB wishes, this functionality was incorporated with the new ECL model, and put in the file Dataprep_Functions.py, and stored together with the ECL program code. To follow the structure of the new ECL model, this functionality was organized in the class DataNormalization, with the two functions transform_macro_data(), that actually normalizes the data, and generate_LTV_hh(), that generates a new macro variable, loan to value for households, based on other macro data provided by DNB Markets. Data must be normalised as the macro input data lacks a common format. There are two cases treated by this function. One is that actual values are given in the historical data, while growth rate is given for projections. These time series will be transformed to actual data also for projections. The other case is where both historical data and projected data are given as a percentage, but where the value given is 100 times as high as the format wanted for the macro data. E.G. 4% is denoted as 4, instead of 0.04. These time series will be divided by 100 for each observation date.

The calculation of LTV_hh is fairly straightforward, and should be easy to read from the code itself, and does not have a visualised flow in this documentation. The formula for the variable is (C2_hh / C2_hh_divider) / (HPI / HPI_divider). C2_hh and HPI are both in the macro input, while C2_hh_divider and HPI_divider are found in the sheet IndexYears in the macro metadata file stored on the ECL project.

The main flow of transform_macro_data() is:



Function transform_macro_data() is called with required parameters specified in function docstring.

Loops through list in macro metadata file to find variables where projections must be transformed from growth values to actual values.

Converts projections to actual values by multiplying previously observed value in the time series by (1 + growth rate given in projections).

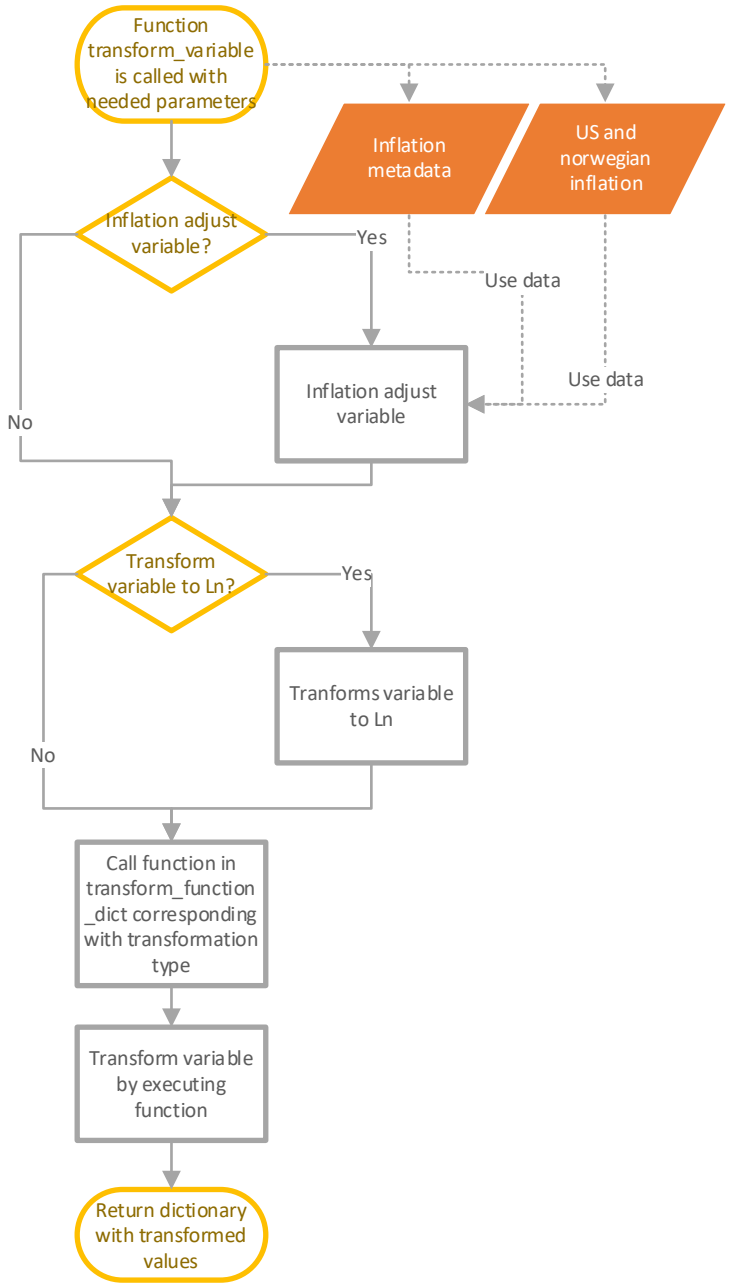
Loops through list in macro metadata file to find variables where entire time series must be transformed into correct format for percentage/growth.

Converts time series to correct format by dividing each observation by 100.

Returns the same data frame that was taken as input, but with normalised values.

Transform variable time series

In accordance with DNB wishes, this functionality was incorporated with the new ECL model, and put in the file Dataprep_Functions.py, and stored together with the ECL program code. To follow the structure of the new ECL model, this functionality was organized in the class Ols_parameter_transformation. It has the main function transform_variable(), which calls other supporting functions corresponding to each supported transformation type.



Function transform_variable is called with required parameters specified in function docstring.

If transformation type contains "Real", inflation adjustment is done using inflation data and inflation metadata specified in input, and inflation code is removed from transformation type.

If transformation type contains "Ln", the time series is converted using the natural logarithm, and Ln code is removed from transformation type.

Uses transformation type as key in dictionary with lambda functions referencing all transformation functions, and combinations of these.

Variable time series are transformed, and a dictionary of the transformed time series.

Testing

The program has been used by DNB to estimate some of the IFRS9 macro models that had model performance that was deemed unsatisfactory given the macroeconomic situation created by the corona virus. In this process all output from the macro model tool has been validated against output given by the current program in SAS. This involved comparing estimated r-squared values, variable p-values, variable coefficients, variable transformations, rolling regression results, predicted Z values for the different scenarios, confidence ranges, and results from the various statistical tests on residuals and time series. OLS regression and statistical tests has also been done manually in excel in

order to ensure the correctness of the output from this program, and time series transformations has been manually calculated to ensure the transformations were implemented correctly in the program.

Future users of the program have been involved in the development from the start of the project, all of whom has little to no experience with Python or working in a terminal window. This has helped to uncover faults and insufficiencies in the user interaction with the program during development. The feedback from future users has been that the final version of the program is very easy to interact with and intuitive to use.

Assessment of program

Feedback from DNB

The feedback from DNB on this program has been overwhelmingly positive. The program functionality has been used extensively by DNB before the program was even finished in order to address IFRS9 macro models that underperformed during the economic shock. The feedback from analysts benefitting from the functionality in this program has been that it would be unlikely that these IFRS9 macro models would be possible to re-estimate in due time before 2020 quarter two reporting had it not been for this program. Both because of the great advantage of having a list of OLS models generated by the OLS suggestion functionality of the program, and the structured and accurate output from the in-depth analysis functionality.

Own assessment of program

Despite some of the program code show signs of the immense time pressure and changes in program requirements brought on by the corona pandemic, the author of this paper is overall very content with the final version of this program.

In terms of programming quality, the functionality generating the suggestions for OLS models is the centrepiece of this program, and is the result of countless iterations of trial and error, and utilizing increasingly complex programming approaches and structural considerations rooted in an understanding of the problem that needed to be solved, in order to get the processing time down from weeks to minutes. This approach to finding IFRS9 macro models also signals a shift in the approach to finding the models from just using industry intuition and manual trial and error, to using a combination of industry intuition and raw computing power to find optimal models, yielding both better models and a significant reduction in time this process takes for involved analysts. By further

expanding this functionality in the program and include other model criteria, DNB can use this approach to find even better IFRS9 macro models in a shorter time in the future.

The storage structure behind the program is also a feature that should be highlighted. By automating how the output from the program is stored, and thus not relying on user of the program following guidelines on how to store and structure output, the probability of having a robust and sustainable system for storage of output has greatly increased. This in stark contrast to how the output from most in-house models are stored, which is often having all output files from a program being saved to a common directory.

The last feature to highlight is the rudimentary way this program has an interface and visualisation of the output, despite being confined to a terminal and directory-based interface. The user interface certainly is not pretty or exciting, but the feedback has been that it is intuitive and a major step up from the system-centred design of the old macro tool program. The output files are in a format that can be attached to model change proposals being sent to internal audit, a far cry from the extensive copy-paste that had to be done from the old program during model change evaluations.

User guide

In order to structure the use of the model, the users are required to create a macro estimation project for that round of IFRS9 macro model estimation before gaining access to the program functionality. It is strongly suggested that only one project is created and used for each round of OLS estimation. Do not create designated projects for different models and/or for different users of the programs. This might lead to errors when deploying the new model parameters to the ECL model.

There are also instruction videos showing some of these operations.

Setting up a new macro estimation project

1. Make sure you have uploaded these files to the program under “Macro_tool_data” before setting up a new project:

    a. The macro data from DNB Markets to be used in modelling. Make sure you have separate sheets for the baseline scenario, the low scenario, and the adverse scenario. It is not important that these sheets have these specific names, but you must know what sheet corresponds to what scenario. Also make sure what is in these sheets is just the macro data itself, and not comments etc. The macro data will be validated by the program to ensure it is in the correct format before use.

b. Z values to be modelled. This file must contain a sheet for each IFRS9 macro model to be modelled in the program, and the sheet name must correspond exactly to the name of the IFRS9 macro models. Cell A1 must contain “Date” and cell B1 must contain “Z”. In the following rows the observation dates and the estimated Z values for each date must be given.

c. List of input variables to be used for suggestion module. This file must contain a sheet for each IFRS9 macro model to be modelled in the program, and the sheet name must correspond exactly to the name of the IFRS9 macro models. It must contain, from cell A1 to D1, “Variables”, Inflation adjust”, “Ln adjust”, “Expected effect”. For each variable to be used, the name of the variable, inflation adjustment(“Nor”, “US” or nothing), Ln adjust(“Y” or “N”) and expected effect(“+” or “-“) must be stated.

2. When starting up the program, you are told to either choose an existing macro estimation project or create a new project. Type “new” in order to create a new project.

3. Type in the name of the new project. The program will not allow the same name as an existing project.

4. Select the macro file containing the macro variables to be used in modelling.

5. Select the three sheets corresponding to baseline, low and adverse.

6. Select the file with the Z timeseries to be modelled.

7. Select the file with the variable input suggestion module in the project.

8. Project is created, and all project files and folders are automatically generated.

9. User is automatically routed into newly created project.

Generate OLS model suggestions

This flow will generate a list of suggested OLS models for the specified IFRS9 macro model. These models are filtered on significance of variables and adjusted r-squared, and are sorted by adjusted r-squared.

1. When starting the program, you are prompted to select a project to work on. If no project has been created for this round of macro model estimation, see the above guide as to how to create a new project.

2. Select “Generate model suggestions” by typing 2.

3. Select the IFRS9 macro model to generate suggestions for.

4. Select variables to include. These variables are from the file “Simulation_input.xlsx” stored under “Project_parameters” in the project folder. To select multiple variables, separate the variable numbers with a comma. The program will validate input and stop execution if input format is incorrect. Program also displays the selected variables and prompts you to confirm the selection with “Y” or “N”.

5. Select maximum number of lags to use in variable transformation. The program supports zero to three lags. Shorter lags in variable transformation generally means the model reacts quicker to economic shocks.

6. Select(Y/N) if you want to include an aim for first year of prediction, including this observation in OLS simulation. If aim is included, specify a target percentile for first year of predictions higher than 0 and lower than 1.

7. Select(Y/N) if you want to use default parameters for OLS regression. If “N”, all parameter values must be set by user. Default parameters are:
    a. Maximum variables per regression, excluding dummies: 3
    b. Maximum number of dummies per regression: 1
    c. Maximum number of times a variable can be used in one particular regression: 1
    d. Significance level used to discard OLS models with insignificant explanatory variable: 0.05

e. Minimum adjusted R-squared required for model not to be discarded: 0.65

8. Program will run, printing start of first and second round of regression, and printing timings for both rounds.

9. Results are written out to the project folder, in the folder “Suggested_OLS_models”.

Do in-depth analysis

This flow will generate extensive statistical information for the given IFRS9 macro model and given explanatory variables.

1. When starting the program, you are prompted to select a project to work on. If no project has been created for this round of macro model estimation, see the above guide as to how to create a new project.

2. Select “Do in-depth analysis of specified model” by typing 3.

3. Select IFRS9 macro model to work on.

4. Select input option. This only has one option as of today, but this is suggested to be increased in future iterations to the program.

5. Type the variable and variable transformations to be used in the regression in accordance with the format specified in the terminal. You can also copy-paste this information from the output from suggested OLS models. The format from both column B and C is supported.

6. The results are saved to the project folder under the folder “OLS_analysis_results”.

7. Some model results are printed to the terminal. This is to minimize any chance of deploying the wrong OLS model. The user is prompted to state if OLS parameters are to be saved for future deployment to the ECL model(Y/N). If “Y”, the user is prompted to confirm this by typing “Save”.

Deploy new model parameters to the ECL model

When all IFRS9 macro models for the particular round of estimation are completed, this flow will save the variables to the ECL model. To minimize the chance of deploying the models by accident, a flow must also be done in the ECL model itself.

1. When starting the program, you are prompted to select a project to work on. If no project has been created for this round of macro model estimation, see the above guide as to how to create a new project.

2. Select “Create new model parameters” by typing 5.

3. The program will print name of all IFRS9 macro models that have been changed, and date of change for each model. This is done as a safety mechanism to avoid deploying incorrect or unexpected model parameters.

4. The user is prompted to select if new model coefficients should be deployed to the ECL model(Y/N).

5. If “Y”, the user is prompted to type “Deploy” to confirm saving of parameters to the ECL model.

Change project parameters after model estimation project is created

Project parameters are loaded into the estimation project when the project is created. If there is a need to change these parameters after the project is created, E.G. if new macro projections should be added or if new variables to use in the functionality that generates model suggestions should be included, this can be done by altering the parameter files located in the project folder under “Project_parameters”. Make sure the file names correspond to the technical documentation requirements specified under “Data structure and input files”, “Project parameters”.

Suggestions for future improvements

During the development of this program, the scope has been somewhat decreased, some code has been sub-optimally implemented, the need for a minor rethinking relating to structuring of the code has arisen, and new potentially useful functionality has been identified. This is a summary of what

has been identified as useful future improvements to the program, with some suggestions as to how it can be done.

Implement User story 1: Generate credit cycles

This was originally a part of the scope of the program, but was taken out of scope due to resource limitations stemming from the corona virus outbreak. It was also taken out of scope due to a discussion around how exactly the Z value for certain IFRS9 macro models should be calculated in the future, and a new structure for mapping the data to the different IFRS9 macro models. This will involve the inclusion of several stakeholders in DNB, and will be part of a greater project. The program is written in a way that should easily allow for this functionality to be implemented into the Macro tool program.

Implement user story 4: Automatic validation of existing models

This was originally a part of the scope of the program, but was taken out of scope due to resource limitations stemming from the corona virus outbreak. It was also taken out of scope as the automated tests used to evaluate the performance is up to consideration in light of experience gained from the performance of some IFRS9 macro models during the corona virus outbreak. The program is written in a way that should easily allow for this functionality to be included into the Macro tool program.

Add a web-based user interface using Dash

During development of this program, DNB started rolling out the Dash-framework on their development platform. This project was not prioritized for the first round of projects being allowed to test out this functionality. When this framework becomes available for everyone on the DNB development platform, it is advised to implement a web GUI for this project in addition to the current terminal based user interface. It will make the program available to more users, as having a terminal based interaction with a program can discourage users unfamiliar with terminal based program interaction from using the program. It would be advisable to visualise the different graphs in the web interface, and make it an option to write out the results from in-depth analysis to an excel file in order to reduce unnecessary storage on the development platform.

Expand OLS suggestions by allowing the user to input ranges the baseline predictions forward in time have to fall within

A problem discovered when using the program to actually model a credit cycle for an IFRS9 macro model was that it was very hard to find models that have reasonable predictions given the extreme economic shock stemming from the corona virus outbreak. This is due to the fact that we lack comparable observations in the modelled time series. Several models with an apparent high degree of fit, measured by the value of adjusted r-squared, either underestimated or greatly overestimated the expected consequences of the economic setback caused by the corona virus outbreak. This was initially thought to be redeemed by setting an aim for the first year of predictions. This did not yield the results that was hoped. A suggested solution is therefore to include an option for the users to set a range of values the predicted Z must be within for the first n number of years of prediction, and discard models with predictions outside of these ranges. This can be done with minor adjustments to the function “do_extended_OLS()” in “Macro_Model_Tool.py”. An even more refined approach is to create a weighing system where model performance is defined as a product of both r-squared and how close it is to an expected range, where the weight of the two parts can be given as an input by the user. This might especially aid in finding good dummy variables, as the program no longer just aims to reduce the residuals, but also adjusts coefficients of explanatory variables to make future predictions more reasonable.

Expand OLS suggestions by filtering out variables not passing the KPSS test

One of the requirements when modelling Z, is that the time series of the explanatory variables are stationary around a mean. This is due to the nature of how Z is calculated. The in-depth analysis returns p-values from the statistical tests Kwiatkowski–Phillips–Schmidt–Shin(KPSS) test and augmented Dickey–Fuller(ADF) test that will help the analyst determine if the variable time series are stationary around a trend. As the KPSS test is the one out of the two tests with the lowest likelihood of determining that a time series is not stationary around a trend, an improvement to the program would be to allow for the user to determine if the OLS suggestion module should automatically discard OLS models containing non-stationary time series. This can be done with minor adjustments to the function “do_extended_OLS()” in “Macro_Model_Tool.py”.

Have a decomposition of the effect from each variable

In model selection it is useful to see what effect the individual explanatory variables have on the

predicted Z. This involves using actual and predicted time series of the explanatory variables, and the variable coefficients. The required information is already stored in variables in the function "write_in_depth_results_to_excel()" in "Macro_Model_Tool.py". The piece of information needed for this not already written to the output excel is the projected time series for the explanatory variables. These are stored in the variables "baseline_macro_predictions", "adverse_macro_predictions" and "low_macro_predictions".

Link macro tool and ECL project to see effect of new model

Link the Macro Tool and the ECL project in such a way that models tested in in-depth analysis can be tried on the ECL model in order to see the effect the new model will have on ECL on the portfolio using that particular model compared to the current model used by the ECL program. This will require some alteration of code in the ECL model in order to make it callable from outside its own interface, and it will require alteration of the Macro model tool. This functionality is suggested to be put into a designated function in the existing function "in_depth_analysis ()" in "Macro_Model_Tool.py". As this will significantly increase execution time of in-depth analysis, it is important that this is an option for the user, and not a default. It must also be decided how the results should be presented. Either aggregated ECL figures can be included into the current output file from in-depth analysis, or more granular data can be exported to a designated excel file. Granular data can also be included into a designated sheet in the current output from in-depth analysis, but it should be taken into consideration that this might lead to performance issues in the excel file itself because of the large amounts of data the ECL model returns.

Add possibility to select variables and transformations used in in-depth analysis from list

The only way of selecting what variables and transformations to use as input into the in-depth analysis is to type or paste the variables in a specific format. This approach should work well for most users, but in order to make the program easier to use, an option to select one variable followed by the wanted transformation from a list, and repeat until all variables are selected could be implemented. A suggestion is to use the file "Simulation_input.xlsx" under "Project_parameters" in the project folder for each project to avoid having to write out all variables in the macro input sheets. All supported transformations are in dictionary "transform_functions_dict" in "Dataprep_functions.py", located in the ECL project itself. This option to select variables should be

added to the function “in_depth_analysis()” in “Macro_Model_Tool_User_Interface.py”. Add the option to the dictionary “input_options” and code that is called if user_input ==”2”.

Refactor duplicated code in trim_time_series()

As the scope relating to the inclusion of more macro scenarios with varying degree of pessimism used for predicting Z forward in time was expanded when working on estimating the macro model in light of the corona virus outbreak, the emphasis was to write code that yielded the required information rather than write optimal code. For that reason, it is some duplicated code in the function “trim_time_series()” in “Macro_Model_Tool.py”. In order to get cleaner, more robust code, this logic should be rewritten in such a way that baseline, low and adverse time series should be trimmed by the same code, and avoid the code duplication currently in the program. This was not done for the first version because of time constraints.

Add validation when adding macro sheets to estimation project

In the first version of the program, the validation of the macro data input is done when running the flows “Generate model suggestions” and “Do in-depth analysis of specified model”. This validation should also be done when adding macro data sheets to a project. It should be kept for when doing the individual runs using the macro data as well, in case users update the macro data after creating the project, but it would identify issues with the macro data earlier if the validation is done when the project is created. This will involve using the function ”import_and_clean_macro_input()” in “Macro_Model_Tool.py” in the function “create_macro_project()” in “Macro_Model_Tool_User_Interface.py”. Because of time constraints, this is not implemented in the first version of the program.

Optimize validation of input macro sheet

During development, some common issues with the macro input was discovered that could be handled by the program automatically. The validation in place will reject these macro sheets, but a better solution would be for the program to get around these issues. The two identified issues that could be handled by the program is difference in end date of actual time series, and cells with information outside the frame with macro data. Difference in end data of actuals time series happen if an old macro scenario is used, which sometimes happen for low and adverse scenario. A solution for this is to use the actuals time series of the baseline for all actuals time series in adverse and low

scenario, and thereby avoiding the problem all together. The problem with cells outside of the frame with macro data containing information stems from certain calculations and comments inserted in the excel sheets for other processes in the bank. A way to get around this is to trim the pandas data frame with the macro data in order to remove cells outside of the macro area. As the number of columns in the macro input is constant, the slice on the width of the data frame is known. In order to know how to slice the length of the data frame, one could count the number of cells in column A, containing observation dates, to know what the area of the sheet actually contains macro data, and slice away rows under last observation date.

Restructure the program into classes

While the part of the program that eventually was included into the ECL model was structured into the classes Datavalidation, Ols_parameter_transformation and DataNormalization, and the user interface was structured into the class User, this was not done for the rest of the program. This was deemed unnecessary during development, but it has become obvious that the program will benefit from a structure where all code is in well defined classes. As the program is set to be expanded beyond the initial scope in the future, such a partition of the code will greatly help with the overall structure of the program. Proposed classes are:

"OLS_suggestion_generator" – containing all the code specific to the flow generate OLS suggestions.

"in-depth_analysis" – containing all the code specific to the flow in-depth analysis.

"Macro_tool_functions" – containing all the code that is used by multiple flows.

"Project_administration" – containing the code that has to do with creating and setting macro estimation projects.

Sources

A. McGeachin & A. Tarce, 2019, Guide to new Standards IFRS 9, IFRS 15, IFRS 16 and research opportunities, viewed June 1st 2020, https://www.ifrs.org/-/media/feature/news/2019/june/basics-of-new-ifrs-standards-and-research-eaa-paphos.pdf

S. Bansal, TrumpExcel, How to Create a Dynamic Chart Range in Excel, viewed June 16th 2020, https://trumpexcel.com/dynamic-chart-range/#Using-Excel-Formulas.

D. Norman 2013, The Design of Everyday Things: Revised and Expanded Edition, Basic books

Python.org, GlobalInterpreterLock, viewed June 16th 2020, https://wiki.python.org/moin/GlobalInterpreterLock

M. Mamaev 2018, If you have slow loops in Python, you can fix it…until you can't, freeCodeCamp, https://www.freecodecamp.org/news/if-you-have-slow-loops-in-python-you-can-fix-it-until-you-cant-3a39e03b6f35/#:~:text=Of%20Python's%20built%2Din%20tools,efficient%20than%20recursive%20function%20calls

R. Arpaci-Dusseau & A Arpaci-Dusseau 2018, Operating Systems: Three Easy Pieces, CreateSpace Independent Publishing Platform

w3schools, Python Lambda, viewed June 16th 2020, https://www.w3schools.com/python/python_lambda.asp

w3schools, Python map() function, viewed June 16th 2020, https://book.pythontips.com/en/latest/map_filter.html

Appendix

1. Program source code.
2. PowerPoint presentation of project.
3. The two templates used for program output.
4. Example of program output.
5. Videos showing program functionality.
6. Time sheet and Gantt chart of project timeline