E08 Lay table automation

This example demonstrates how the OrcaFlex automation tools can be used to create installation lay tables.

A basecase model is analysed to determine the range of top tension values within which the lay operation can proceed. To find the limiting tensions, a range of pipe (or umbilical) spans from ship to touchdown are analysed statically to find the cases where one or more of the acceptance criteria is infringed. Once the limits are found, a set of intermediate cases are analysed, and results are reported to a lay table, which clearly indicates the acceptable / unacceptable cases.

It is assumed that the user is familiar with the use of OrcaFlex, and either the OrcaFlex spreadsheet or Python (depending on the preferred automation tool).

This example includes files that can be modified by the user to suit their needs. Note that whilst the files include representative data, they are examples only, and should be modified and checked for accuracy as required to suit the individual needs of an application or project. **Orcina will not be held responsible for any inaccuracies in data as a result of using these example files.**

Although two methods are presented here, the use of Python provides a much more efficient and robust automation, and this is the preferred method. Even if the user has little or no experience of Python coding, this does not need to be a barrier as it is a logical and user-friendly language. Python is available as a free download and, in addition to the information found within the OrcaFlex API help, there are a great deal of related resources available on the internet.

The two automation methods are described below.

Lay table automation using Python

Introduction

The Python automation example requires the following three files to run:

- *CreateLayTable.py* – the main Python code
- *Basecase.dat* – the basecase OrcaFlex file
- *PipelayConfig.yml* – the file containing user input data.

To allow the example to run without modification, the above files should all be saved in the same folder (different locations can be used for each file, but the code would have to be modified accordingly).

When the example is run (by double-clicking file *CreateLayTable.py*), this will open the basecase file, cycle through a variety of load cases where the anchor position of the *Pipe* in the model is modified, establish the near and far cases, save a series of OrcaFlex dat files, and then automatically generate and open a lay table in Excel format.

Pre-requisites

The code is written in Python 3. A suitable version of Python will need to be installed before you can run this script. See the Python interface | Installation page of the OrcFxAPI help file for details.

As well as Python, the following 3rd party Python modules will need to be installed to allow the code to run:

- **`XlsxWriter`** (for writing and formatting data in an Excel spreadsheet)
- **`PyYaml`** (to allow access to a separate config file containing user inputs)

It is recommended that a good source code editor such as Notepad++ is used for viewing and editing the Python code. Notepad++ provides useful features such as syntax highlighting, colour coding, file compare, etc.

Excel data is written in .xlsx format, which dictates the use of Excel 2007 or later.

User input data

Whilst the main code will need to be modified to suit each application, the most common data that will require user modification is contained within a separate file, *PipelayConfig.yml*. This allows changes to be made to the acceptance criteria without needing to edit the Python code. In the example, these are the suggested user inputs:

- **Basecase name** – the name of the file containing the base OrcaFlex model.
- **Max allowable tension** (kN).
- **Min allowable tension** (kN).
- **Min allowable constraint clearance** (m) – minimum allowable distance to the constraint defined above the stinger in the basecase.
- **Max allowable bollard pull** (kN) – the maximum horizontal component of force exerted on the vessel.
- **Increment** (m) – the distance between anchor points from one load case to the next, used when finding the near and far extremes.
- **Number of cases to analyse** – the number of cases required in the lay table, including the near and far extremes.
- **Code check reference** – the code check that will be referenced in the lay table.

Python code

The code contains descriptive text where necessary (indicated with the '#' prefix) to help the user understand the purpose of each section.

The following list is a summary of the main steps that occur when the code is run:

- The config file (*PipelayConfig.yml*) is opened and user data (including acceptance criteria) is retrieved.
- The basecase file is opened and the key objects are identified.
- The user is notified via command line that the process has started.
- Statics is run on the basecase file and results are compared to the defined acceptance criteria.
- Assuming all acceptance criteria are achieved with the basecase file, the anchor position is then reduced systematically by the defined increment from the default position and results assessed, until the near limit load case is identified (as soon as one of the acceptance criteria is not achieved, this is identified as the near limit).
- Once the near limit has been achieved, the anchor position is then increased by the same increment from the default position until the far limit is identified.
- The user is notified via command line that the limits have been found.

- Load cases are generated at equal increments between the near and far limits, with the quantity of load cases defined by the config file. The resulting dat files are saved in the same folder as the Python file.
- A new Excel file is created, and global settings (such as paper size and orientation) are defined.
- The formatting is defined for all sections of the Excel file (font name, size, cell colour, etc.).
- The following sections are inserted into the Excel file:
    - Title box
    - Basecase data
    - Acceptance criteria
    - Comments sections
    - Headers for the main lay table
    - Document footer.
- The previously saved dat files are re-opened one by one, statics run, and the necessary results generated.
- While the above step is happening, the user is notified via the command line as each file is being processed, and the necessary results are then inserted into the Excel lay table.
- If all acceptance criteria are achieved for a load case, the relevant line in the lay table is coloured green. For the near and far limits (i.e. when one or more acceptance criteria are not achieved), the line is NOT coloured green, and the cells related to the failed values are coloured red.
- Once all data has been inserted, formatting is applied to the remaining empty cells in the spreadsheet to remove grid lines and colour the cells white.
- The spreadsheet is saved, and the user is notified via command line that the process is complete.
- Excel is started automatically, and the lay table opened.

Notes

- Some typical user modified variables have been placed in the separate 'config' file. However, it is expected that the main code will also need to be edited to suit the needs of each particular application.
- The acceptance criteria include limits from code check API RP 1111. These will need to be modified if another code check is required.
- Part of the code launches all dat files that are present in the root folder (using the `glob` module). If the number of cases is reduced and the code is re-run, the old dat files will still be there and will be included in the lay table (e.g. If the code is run with 20 cases, 20 dat files will be created and saved; if it is then re-run with 16 cases, dat files 17, 18, 19 & 20 will need to be deleted prior to the lay table being created, otherwise the table will be incorrect). This process could of course be automated, but we have not done so in this instance to ensure that the code does not inadvertently delete any other files that the user had stored in the same folder!
- If a previous version of the created spreadsheet is open when the code is run, it will fail at the point it tries to create the new spreadsheet. This is because Excel cannot create two files with the same name in the same folder.

Lay table automation using the OrcaFlex spreadsheet

Introduction

OrcaFlex is supplied with a special Excel add-in which enables you to automate the extraction of simulation results into your own spreadsheet. You can then use the normal Excel calculation facilities to do your own customised post-processing. For users who are unfamiliar with, or do not have access to Python, this spreadsheet example has been created.

Warning: The lay table created using this method is highly dependent on the content and location of data from a number of set-up sheets (pre-processing, post-processing, and 'raw' results). Care should be taken when changes are made to any of these sheets as it can introduce errors into the lay table. The use of Python automation (as described earlier) is preferred, as it completely removes this risk.

The OrcaFlex spreadsheet automation example requires the following files to be able to run:

- *OrcaFlex Pipelay Results.xlsx* – the OrcaFlex spreadsheet which contains pre- and post-processing, raw results, and final lay table.
- *Basecase.dat* – the basecase OrcaFlex model.

The basic steps to use this method are as follows:

- Complete pre-processing sheet with estimated range of load cases and required increment (this may need to be modified once the results have been obtained and limits identified).
- Create and run batch script file to create the dat files. Note: if Distributed OrcaFlex is available, there is an option available called *save, run and submit*, which also generates sim files, but this should not be used in this instance as it creates dynamic sim files (we require static sim files only).
- Using OrcaFlex batch processing (or Distributed OrcaFlex if available), create, run and save statics sim files for all the previously created dat files. Note, both methods offer a check box option (*skip dynamics*) which, if ticked, means that only a static analysis would be performed for each load case in this example.
- Complete and process required results using the post-processing sheets.
- Complete the necessary acceptance criteria on the final results sheet and check if the appropriate near and far limits have been used. If not, modify and re-run all steps to generate the new set of sim files. Repeat this cycle until the first and last results in the lay table are the near and far extremes (i.e. they are the only lines in the table with a failed result).

Pre-requisites

This spreadsheet requires the OrcaFlex Excel add-in to be installed. This should be automatically installed with OrcaFlex, and successful installation is indicated by the presence of an additional 'OrcaFlex' tab in Excel when the spreadsheet is open. See the OrcaFlex help file section *Automation* for further details.

Pre-processing

The first tab in the spreadsheet shows the load cases that will be generated. Using this method, the limits aren't established automatically, so a certain amount of trial and error will be needed prior to settling on the final line end positions to include in the lay table. In this example (to allow

comparison to the equivalent Python code) there are 18 load cases including the near and far extremes.

Post-processing

Post-processing in the spreadsheet is split up into two tabs – one for range graph results (multiple results for each load case) and one for all other results (where a single value is returned for each load case). The post-processing commands can all be contained in the same sheet but splitting them in this way makes the generation and formatting of the subsequent raw results sheets slightly easier.

It is generally best practice with post-processing to output results to a 'raw' results sheet and then add a further final 'filtered' results sheet with all necessary formatting, etc. This is because the post-processing commands automatically populate the results sheet they reference, and as such there is a risk of over-writing formatting, formulae etc. when modifying or adding to the post-processing commands.

Ordinarily, to avoid the risk of referencing old data, a *Clear* command would also be added at the start of the post-processing to remove all data from the raw results sheet. In this case, however, a formula is necessary in the raw results sheet for calculating whether the maximum stress is in overbend or sagbend. For this reason, the *Clear* command has been omitted from the *Range Graphs Post-processing* sheet.

Raw results

The raw results, as with the post-processing instructions, are split into two sheets: one for range graph results (*Range Graph Results*, results in column format) and one for all other results (*General Results*, results in row format).

When duplicating instructions for multiple load cases, care should be taken to ensure the appropriate row / column offset is applied. The *Final Results* sheet (i.e. the lay table) is dependent on the raw results always being in the same location.

The *Range Graph Results* sheet includes formulae under the *Declination Direction* column, necessary for calculating whether the maximum stress is in the overbend or sagbend. If extra load cases are required, it will be necessary to copy and paste the formulae in this column for those extra cases.

Final results – lay table

If the preceding information has all been set correctly, the only user inputs required in the lay table are the code check reference and the acceptance criteria (cells highlighted in yellow). Stress limit acceptance criteria are not highlighted as these are automatically pulled in from the basecase file.

If another code check is required, this will affect the acceptance criteria used, and the *Final Results* table will need to be modified accordingly.

The lay table includes rows for 18 load cases. If this is modified for more load cases, the formatting and formulae will need to be copied down for the required extra rows. Similarly, if fewer load cases are required, the unused rows should be deleted.

The columns for maximum stress (overbend and sagbend) contain array formulae, and as such, if the cells are modified, it is necessary to hold *Shift* and *Ctrl* when pressing *Enter*, otherwise the formula will be incorrect. Additionally, these columns reference a specific number of rows in the

Range Graph Results sheet (currently rows 4 to 385). Note that if the number of line segments is increased in the basecase file, this will increase the number of nodes, and therefore the number of range graph results recorded. Consequently, it will be necessary to widen the range of rows that are referenced, otherwise the maximum values returned may not be correct (i.e. if a maximum value occurred in one of the new rows not referenced).

The first column A is hidden by default. This column contains ascending numbers identifying the row and is necessary for all 'OFFSET' formulae within the lay table. If new rows are added, this column should also be added to with the next ascending number, otherwise none of the results will populate.

References

For further information relating to the use of Python with OrcaFlex, please refer to the API help file. This can be accessed from the Help | API documentation menu item in OrcaFlex.

The same resource can be found in the OrcFxAPI sub-folder of the OrcaFlex installation directory, or on the OrcaFlex documentation page of our website. The same page offers an introduction to the Python interface to OrcaFlex; including a PDF document, summarising a short introduction to the Python interface to OrcaFlex, along with a separate folder containing a series of example scripts.

For further information relating to the use of Excel for pre- and post-processing with OrcaFlex, please refer to the *Automation* section of the OrcaFlex help file.