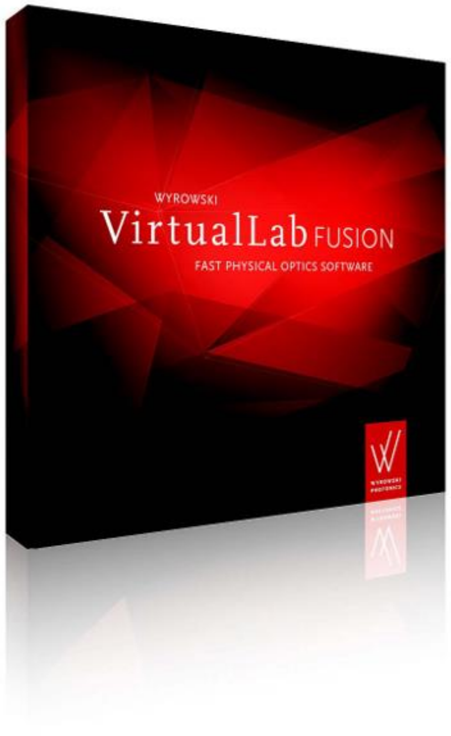
Cross-Platform Optical Modeling and Design with VirtualLab Fusion and Python

Abstract



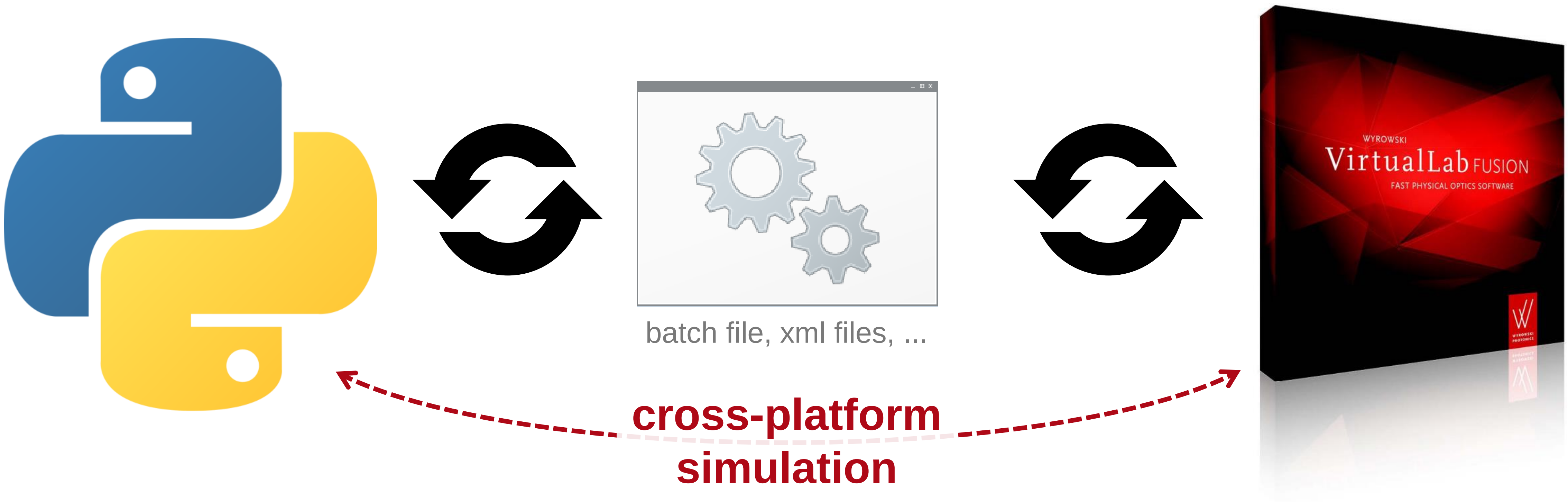
VirtualLab Fusion allows external access to its solvers. This is helpful if data processing or optimization tools other than those of VirtualLab should be used. Via the standard batch mode, we demonstrate how to use Python to trigger VirtualLab in the background to run optical simulations and output their results which can then be further processed and visualized with Python's capabilities. As example rigorous grating analyses and parametric scanning are shown.

Workflow Overview

Python
- interactive access to batch mode files
- external mathematical functions and tools

Batch Mode Files
- execution of simulations
- optical parameters and simulation result storage

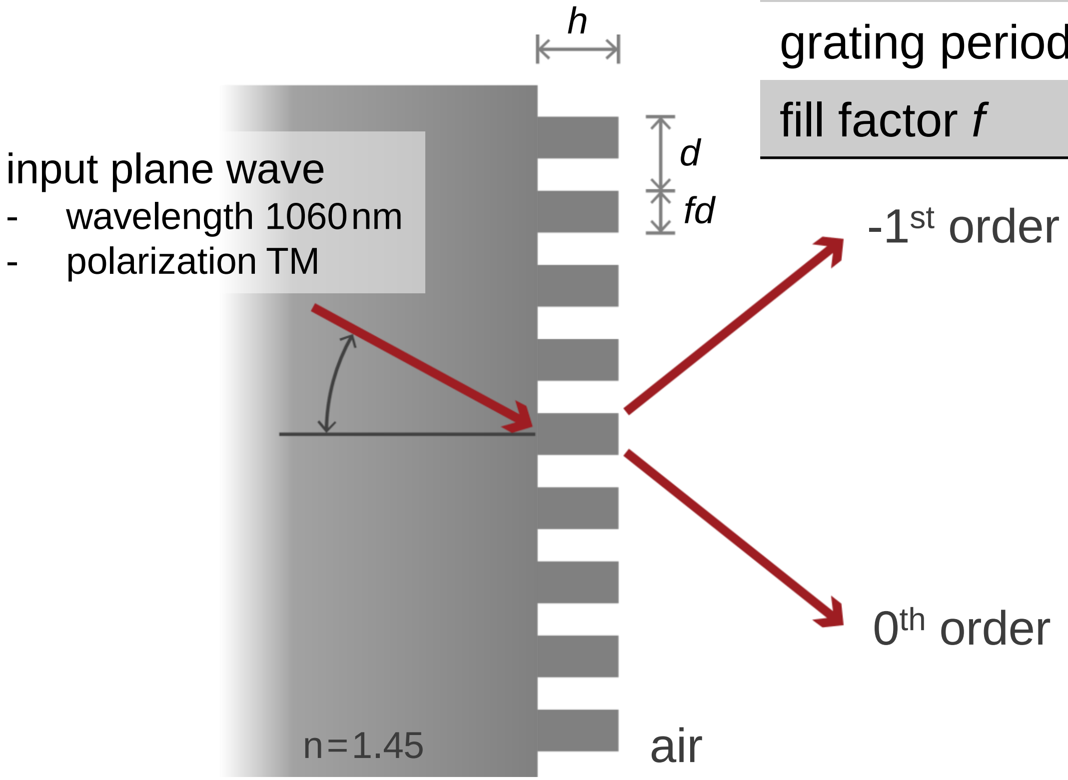
VirtualLab Fusion
- optical setup definition
- kernel simulation engine



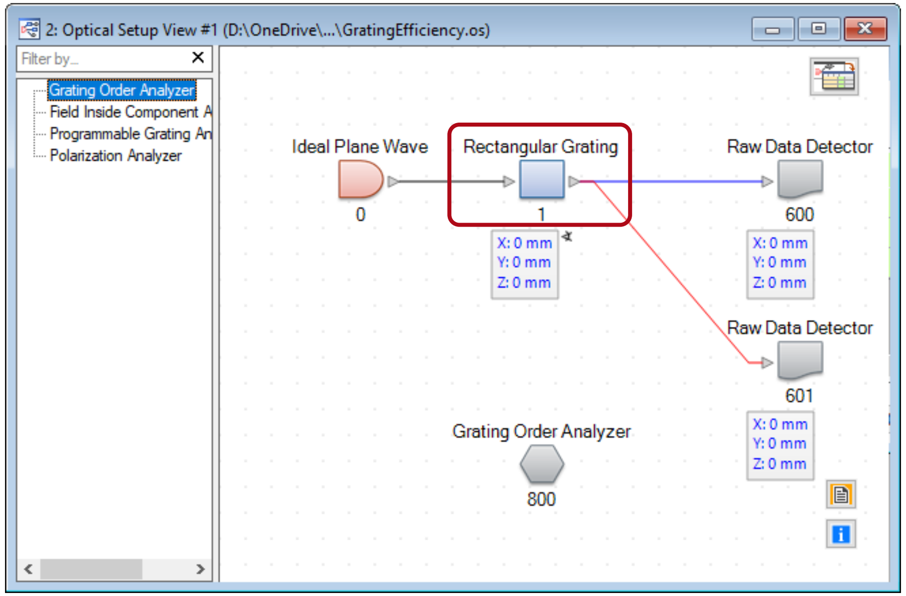
Define Optical Setup in VirtualLab Fusion

initial grating parameters

Parameter	**Value**
grating depth h	1.85 µm
grating period d	1060 nm
fill factor f	50 %

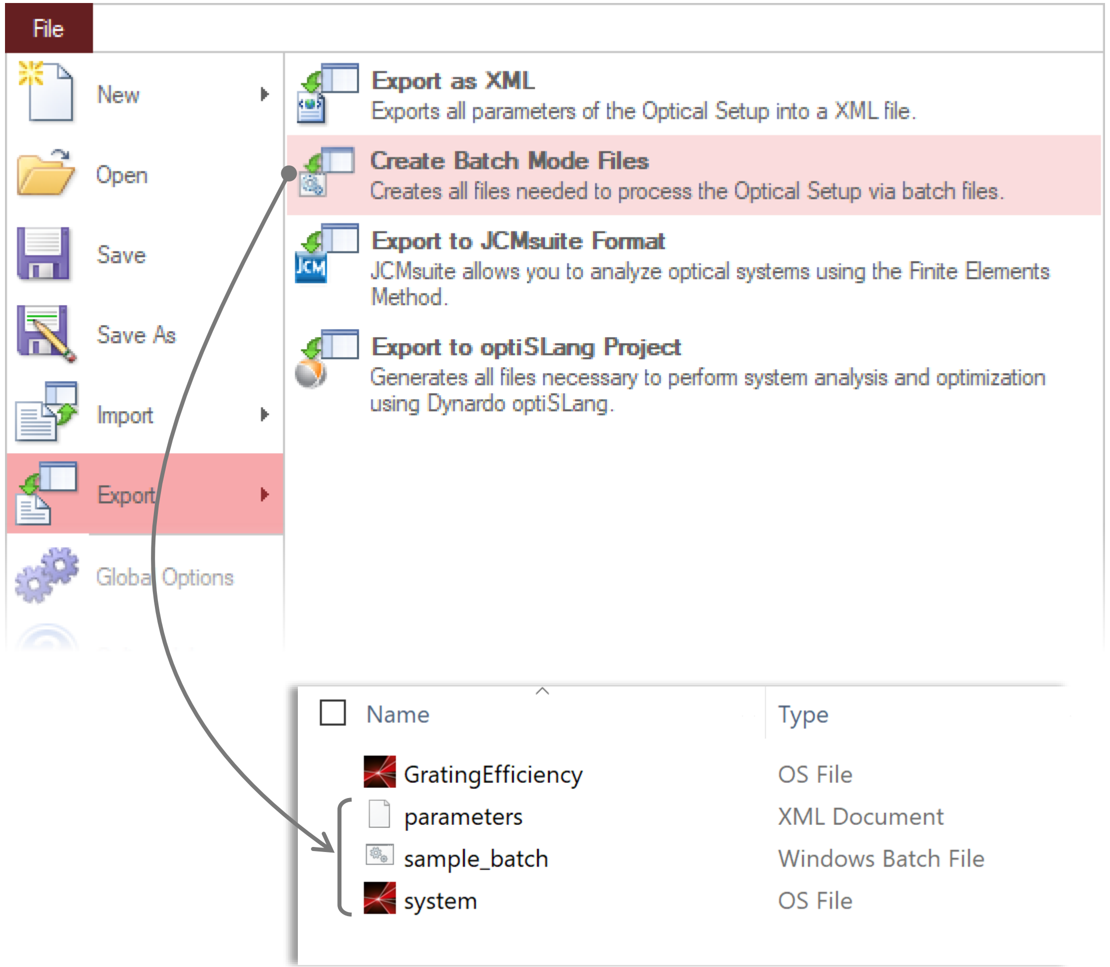


corresponding optical setup generated in VirtualLab



Create Batch Mode Files

- First we create batch mode files for a selected optical setup.
- In the selected folder, three new files are generated
  1. parameters.xml
     xml file containing all parameters of the optical setup from VirtualLab
  2. sample_batch.bat
     batch file containing commands intended to be executed
  3. system.os
     os file (VirtualLab file format) containing the original optical setup



Batch File Content

The batch file can be opened with any editor like program.

After the generation of the batch file, there will be as many commands listed to trigger a VirtualLab Fusion simulation as simulation engines are available in the optical setup document, e.g.

- Field Tracing
- Classic Field Tracing
- Ray Tracing
- Ray Tracing System Analyzer

Typically not all type of simulations are required and also not all optional arguments, e.g. the generation of a subfolder where the results are input.

The command looks like the following:

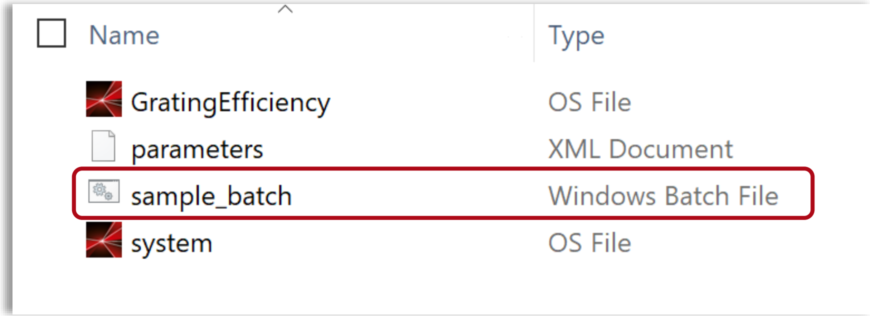
ARGUMENT(S)	DESCRIPTION
-performLPD {1}	The mandatory argument `-performLPD` must be followed by the path and file name of the Optical Setup to be simulated.
{2}	The output folder. Results are written into a `results.xml` file. Complex documents which cannot be saved into this XML file are stored as separate documents, the `results.xml` file then contains only a reference to that file. If warnings or errors occur during the simulation, they are written into a `ProcessingInfo.log` file. (If *Pop up Error Messages* or *Pop up Warning Messages* is activated in the Global Options dialog (↪Sec. 6.22), the corresponding messages are also shown in a message box.)
-parameters {3}	With this optional argument you can specify a XML file with parameter values. These values are then used for the simulation instead of the original parameter values. You can use File > Export > Create Batch Mode Files to create a sample XML file named `parameters.xml` with the correct format.
-engine {4}	With this optional argument you can specify the simulation engine to be used. "0" refers to Classic Field Tracing, "1" to Field Tracing, "2" to Ray Tracing, and "4" to Ray Tracing System Analyzer. Other numbers refer to the index of the analyzer to be used for the simulation. If this parameter is not specified, the *Default Simulation Engine* from the Global Options dialog (↪Sec. 6.5) is used. For Laser Resonator Optical Setups always the Eigenmode Analyzer is used.
-subfolder	If this optional parameter is specified, a subfolder in the output folder `{2}` is generated where the result and logging files are stored. In this way consecutive calls of the `virtuallab.exe` do not overwrite already calculated results. The name of the subfolder is `<Name of simulation engine> (<Date and Time>)`.

```
virtuallab.exe -performLPD {1} {2} [-parameters {3}] [-engine {4}] [-subfolder]
```

Modify Batch File

Open batch file (e.g. with an editor)

1. choose simulation engine
   (in this example only the Grating Order Analyzer is used)
2. delete the output option
   (the presented example works without subfolder)




```
"%VirtualLab_AppDir%\VirtualLab.exe" -performLPD "C:\VLF-Python\system.os" "C:\VLF-Python" -parameters "C:\VLF-Python\parameters.xml" -engine 2 -subfolder & REM Classic Field Tracing
"%VirtualLab_AppDir%\VirtualLab.exe" -performLPD "C:\VLF-Python\system.os" "C:\VLF-Python" -parameters "C:\VLF-Python\parameters.xml" -engine 800 -subfolder & REM Grating Order Analyzer
```

[original batch file]

1. delete the line for Classic Field Tracing →

~~-engine 2 -subfolder & REM Classic Field Tracing~~

-engine 800 ~~-subfolder~~ & REM Grating Order Analyzer

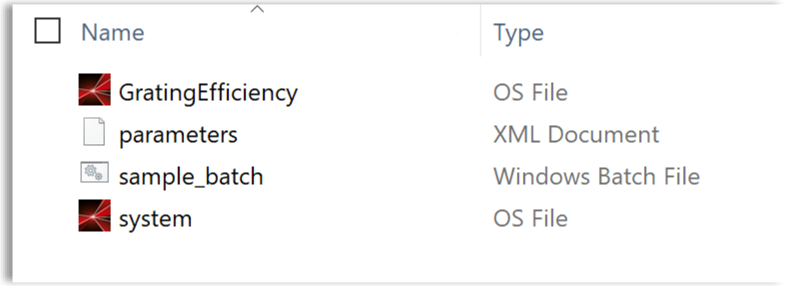
2. delete subfolder option

[modified batch file]

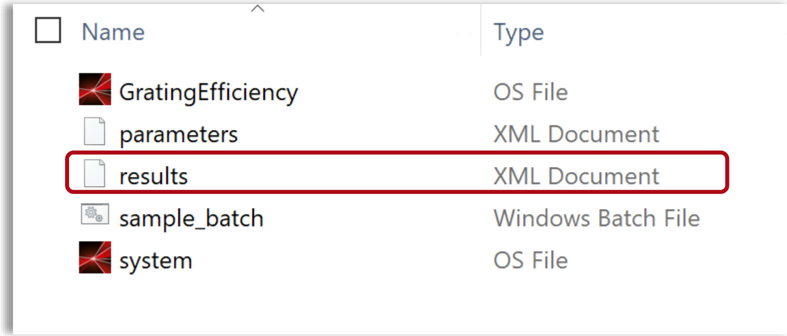
```
-engine 800 & REM Grating Order Analyzer
```

Execute Simulation Using Batch File

- It is recommended to execute the batch file first (e.g. by double click in the MS Explorer window), as a pre-check for the complete workflow.
- After execution, a new file is generated
  - results
    (xml file containing the result values)
- One may also open the result.xml file to check the result values.



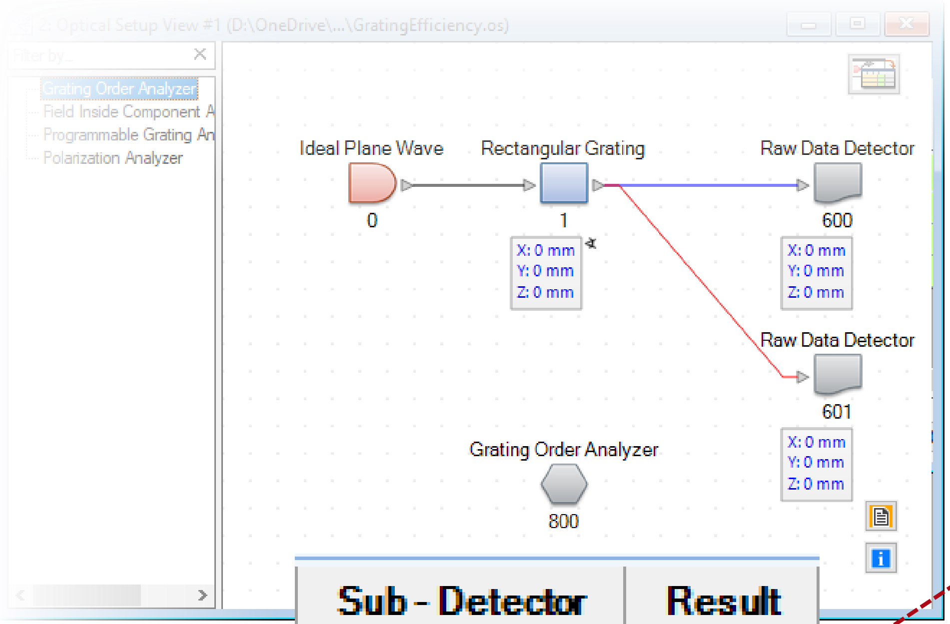
before executing batch file



after executing batch file

Checking Simulation Results Generated by Batch File

- Results in VirtualLab Fusion



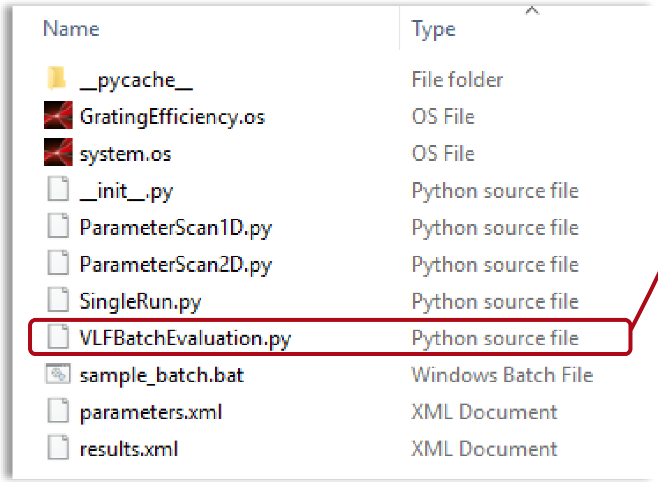
Sub - Detector	Result
Efficiency T[-1; 0]	87.41 %
Efficiency T[0; 0]	10.331 %
Efficiency T[+1; 0]	0 %

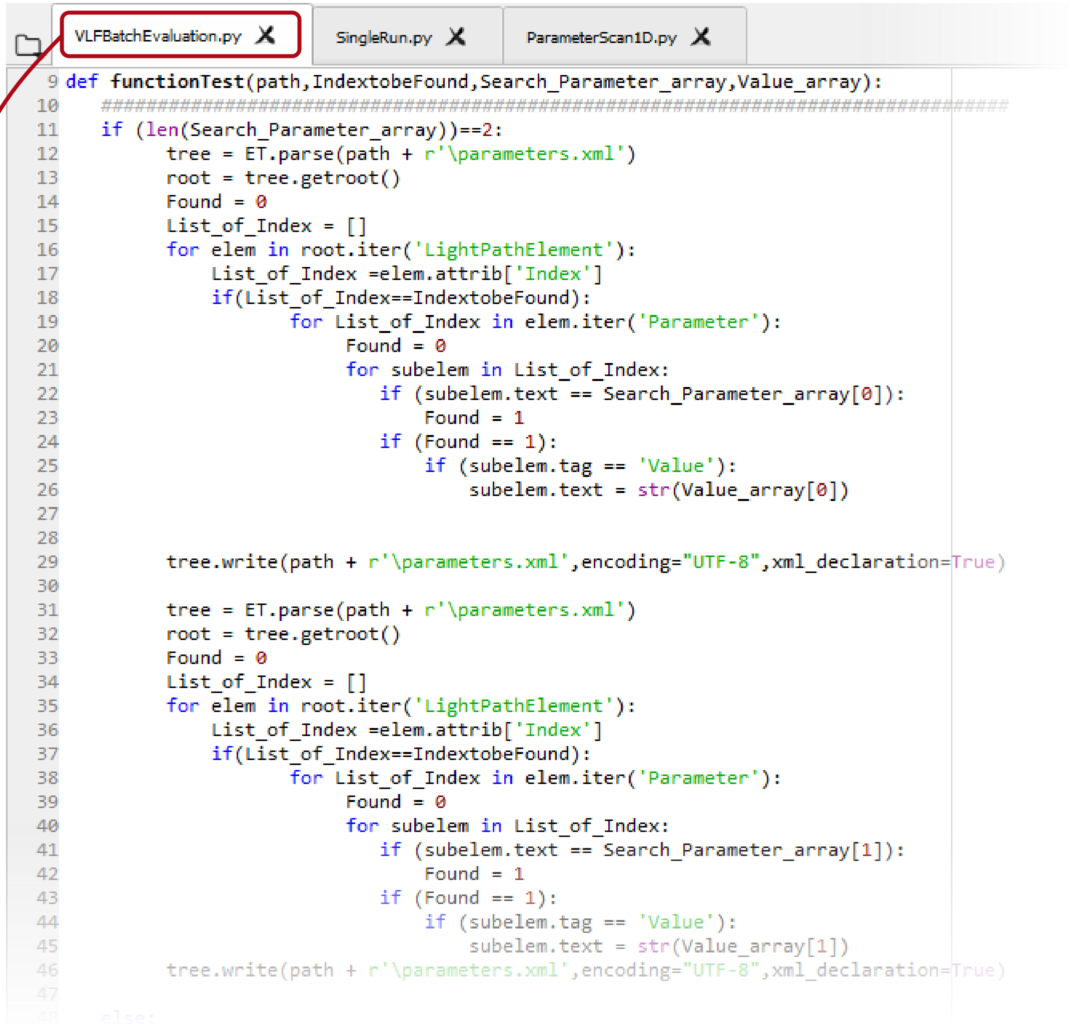
- Results in xml file (can be viewed e.g. in simpled text editor or internet browser)

<?xml version="1.0" encoding="UTF-8"?>
- <Detector_Results Engine="Grating Order Analyzer">
<VLVersion Version="2020.2.2.22"/>
- <Detector_Result Type="List of Physical Values" Name="“Grating Order Analyzer” (# 800) (Results for Individual Orders)">
- <Result Index="0">
<Name>Efficiency T[-1; 0]</Name>
<Value>87.409800037125478</Value>
<Unit>%</Unit>
</Result>
- <Result Index="1">
<Name>Efficiency T[0; 0]</Name>
<Value>10.330826275349573</Value>
<Unit>%</Unit>
</Result>
- <Result Index="2">
<Name>Efficiency T[+1; 0]</Name>
<Value>0</Value>
<Unit>%</Unit>
</Result>
</Detector_Result>
</Detector_Results>

Execute Simulation Using Python (via Batch)

- A basic Python function has been prepared for executing the batch file and interacting the related xml files.
- Copy **"VLFBatchEvaluation.py"** file directly to the working folder.





Execute Simulation Using Python (via Batch)

- In this example, one can execute the Python function below `FunctionTest(Path,IndexToBeFound, Search_Parameter_ ...)`
- A Python file **"SingleRun.py"** is prepared for executing the function.

VLFBatchEvaluation.py × | SingleRun.py ×

```python
import numpy as np
import sys
sys.path.append(r"C:\VLF-Python\VLFBatchEvaluation.py")
from VLFBatchEvaluation import*
path = r"C:\VLF-Python"
####################################################################
#default Value
IndextobeFound = '1'
Search_Parameter='Stack #2 (Rectangular Grating) | Surface #1 (Rectangular Grating Interface
Value =1.85e-6 # given in meters
value_array=np.array([Value])
Search_Parameter_array=np.array([Search_Parameter])
efficiency=functionTest(path,IndextobeFound,Search_Parameter_array,value_array)
```

Name	Type
__pycache__	File folder
GratingEfficiency.os	OS File
system.os	OS File
__init__.py	Python source file
ParameterScan1D.py	Python source file
ParameterScan2D.py	Python source file
SingleRun.py	Python source file
VLFBatchEvaluation.py	Python source file
sample_batch.bat	Windows Batch File
parameters.xml	XML Document
results.xml	XML Document

In this example, the -1st order efficiency is displayed after executing the function

IPython console

Console 1/A

```
In [7]: runfile('C:/VLF-Python/SingleRun.py', wdir='C:/VLF-Python')
Reloaded modules: VLFBatchEvaluation
87.409800037318078

In [8]:
```

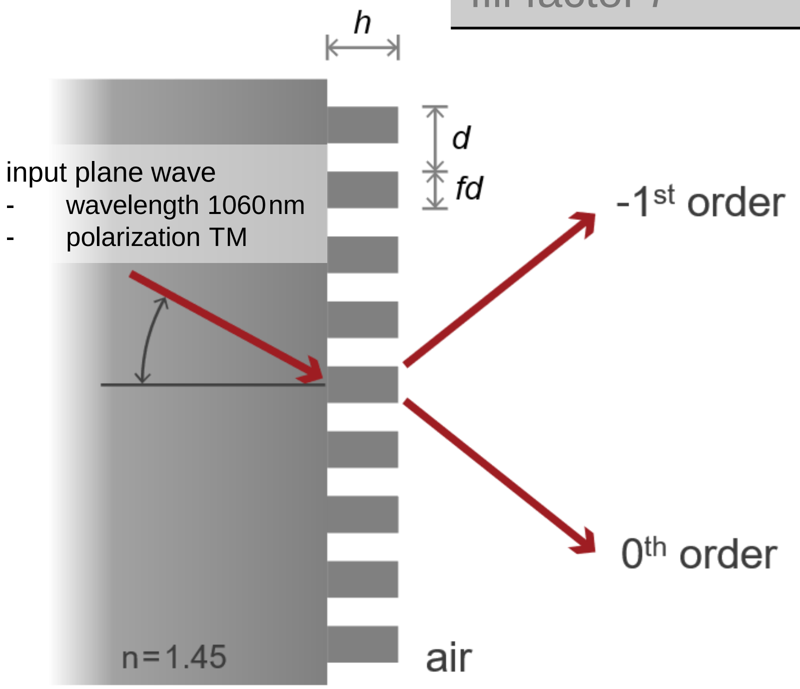
-1st diffraction order

Parameter Scanning – Varying Single Parameter

- The basic Python file can be used as a sub-function in another Python file as well.
- As an example, we demonstrate how to scan a selected parameter in the optical setup and to check the influence on the result.
- In this example the grating depth is varied, and the transmitted diffraction efficiency of the -1st order is evaluated.

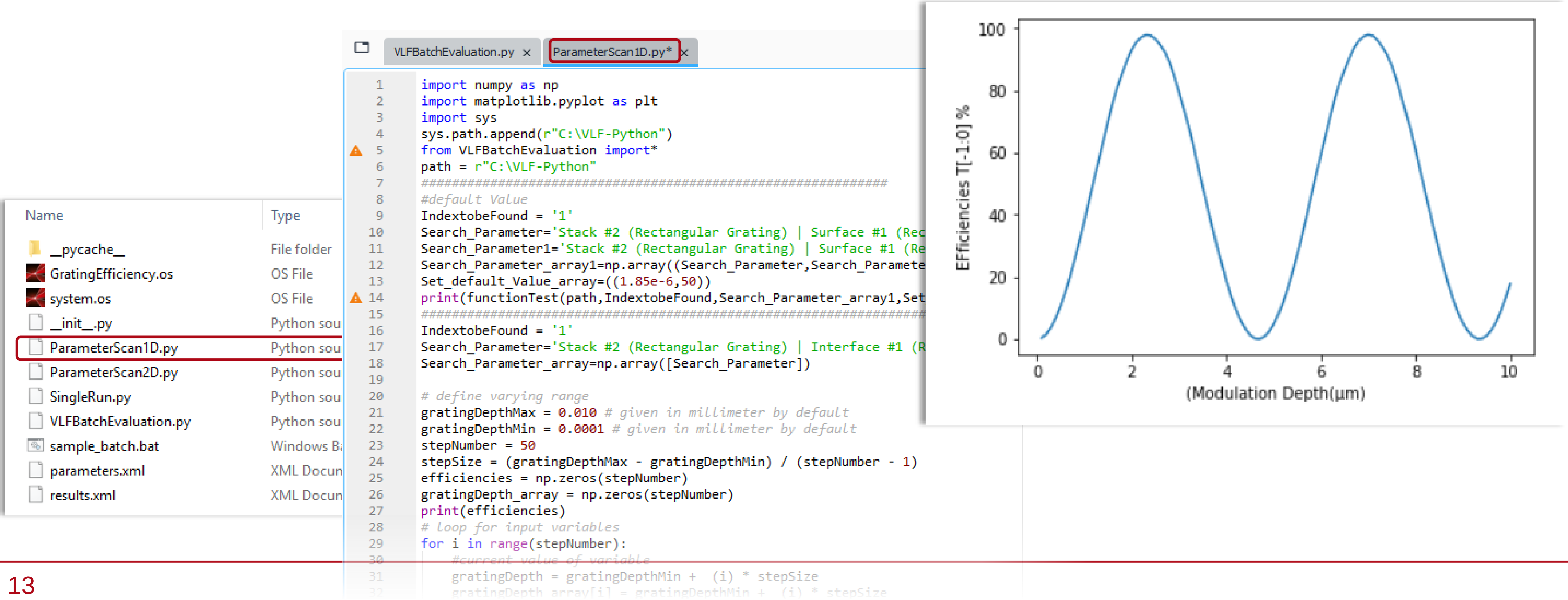
grating parameters

Parameter	Value
grating depth h	**[0.1; 10.0] µm**
grating period d	1060 nm
fill factor f	50 %



Parameter Scanning – Varying Single Parameter

To use the example file, directly copy the Python file "ParameterScan1D.py" into the working folder, adjust the working path, and then execute it.

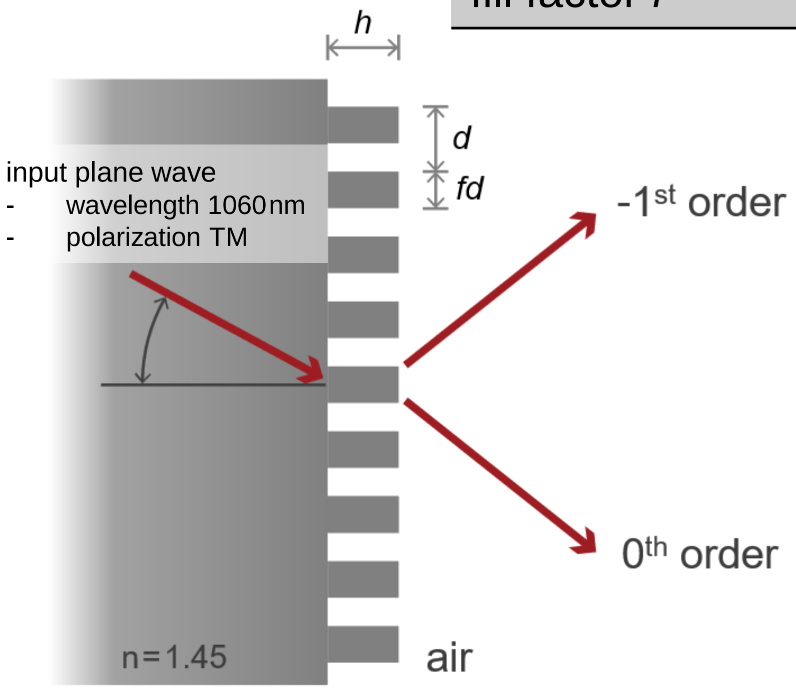


Parameter Scanning – Varying Multiple Parameters

- The basic Python file can be applied in a flexible way.
- For example, one can vary multiple variables and make a multi-dimensional scan over the parameter space.
- In this example, both the grating depth and the fill factor are varied, and again the diffraction efficiency of the -1st order is under investigation.

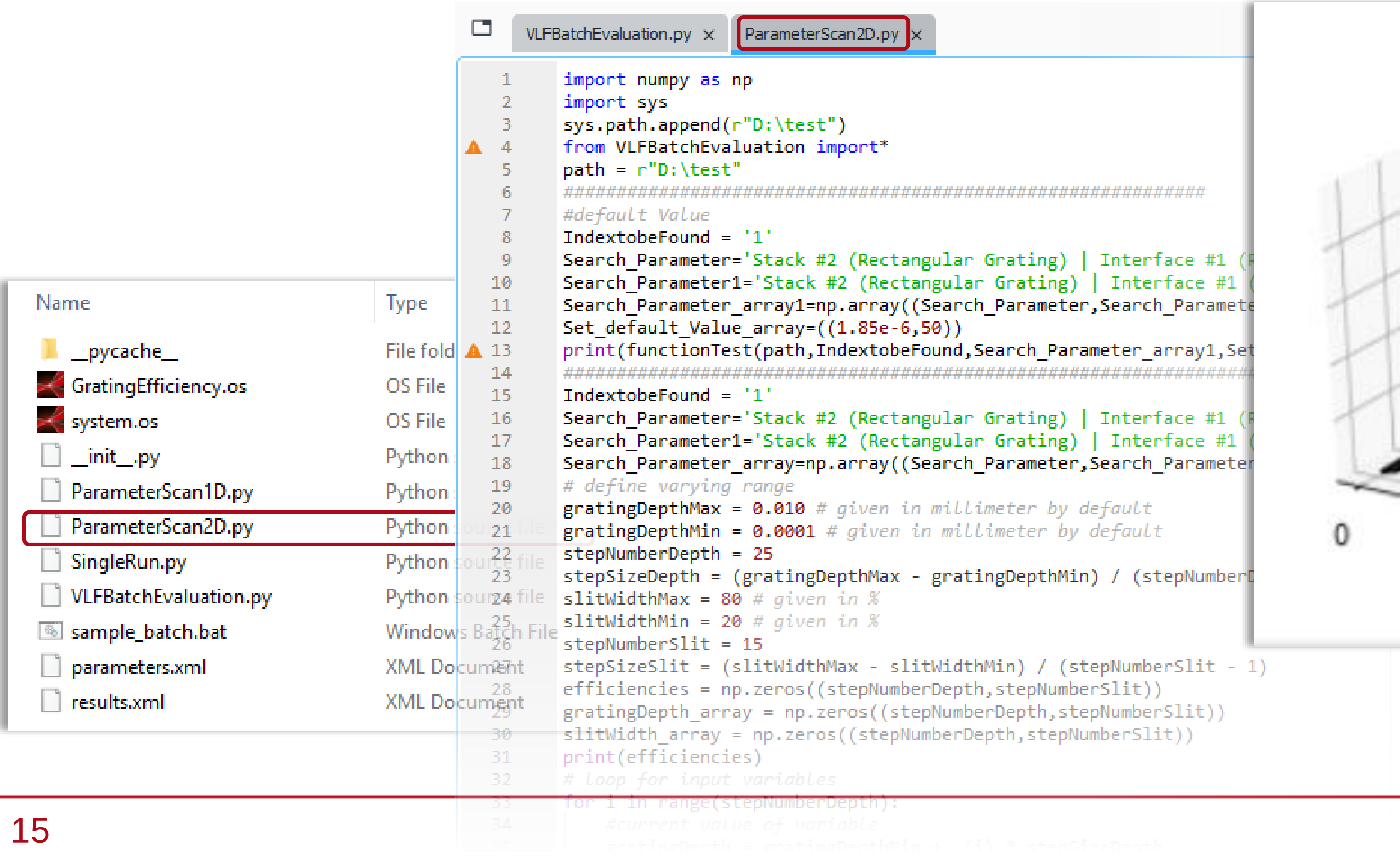
grating parameters

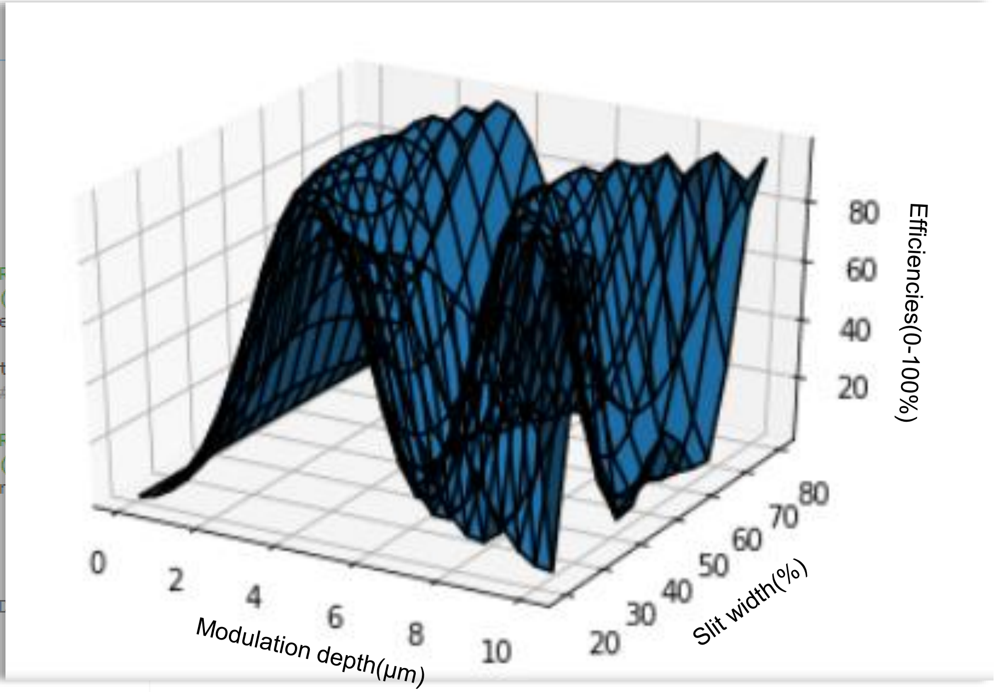
Parameter	Value
grating depth *h*	**[0.1; 10.0] µm**
grating period *d*	1060 nm
fill factor *f*	**[20; 80] %**



Parameter Scanning 2D – Varying Multiple Parameters

To use the example file, directly copy the Python file "ParameterScan2D.py" into the working folder, adjust the working path, and then execute it.





Document Information

title	Cross-Platform Optical Modeling and Design with VirtualLab Fusion and Python
document code	CPF.0002
version	2.0
toolbox(es)	(depending on situation; Grating Toolbox used for this example)
– VLF version – Python version	– VirtualLab Fusion 2020.2 (Build 2.22) – Python 3.7.1
category	Feature Use Case
further reading	– Cross-Platform Optical Modeling and Design with VirtualLab Fusion and MATLAB