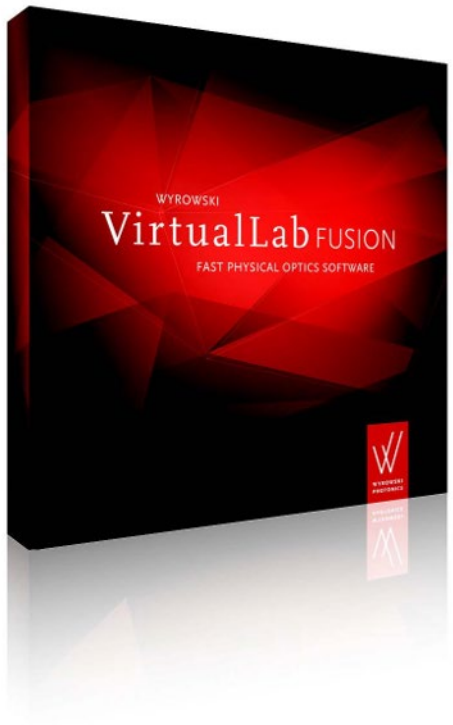
LIGHTTRANS

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

Abstract



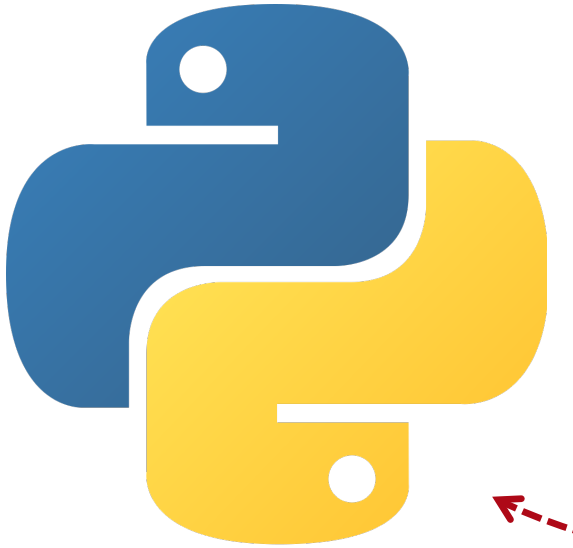
Modeling and design of complex optical systems often requires the use of multiple softwares together, since a single software can hardly provide the needed functionalities for different fields under investigation. Via the standard batch mode, we demonstrate how to use Python to access the field solvers from VirtualLab Fusion and perform optical simulation with Python. Examples on rigorous grating analysis and parametric scanning are shown.

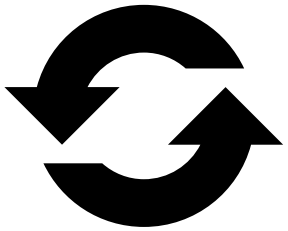
Workflow Overview

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

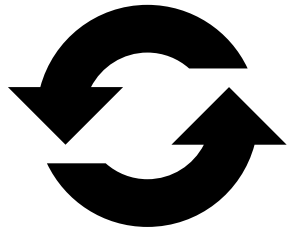
Batch mode files
- execution of simulations
- optical parameters and simulation result storage

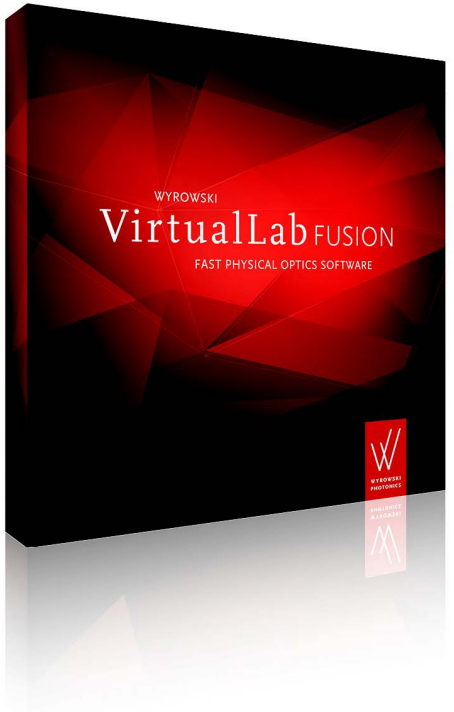
VirtualLab Fusion
- optical setup definition
- kernel simulation engine





batch file
xml files
...



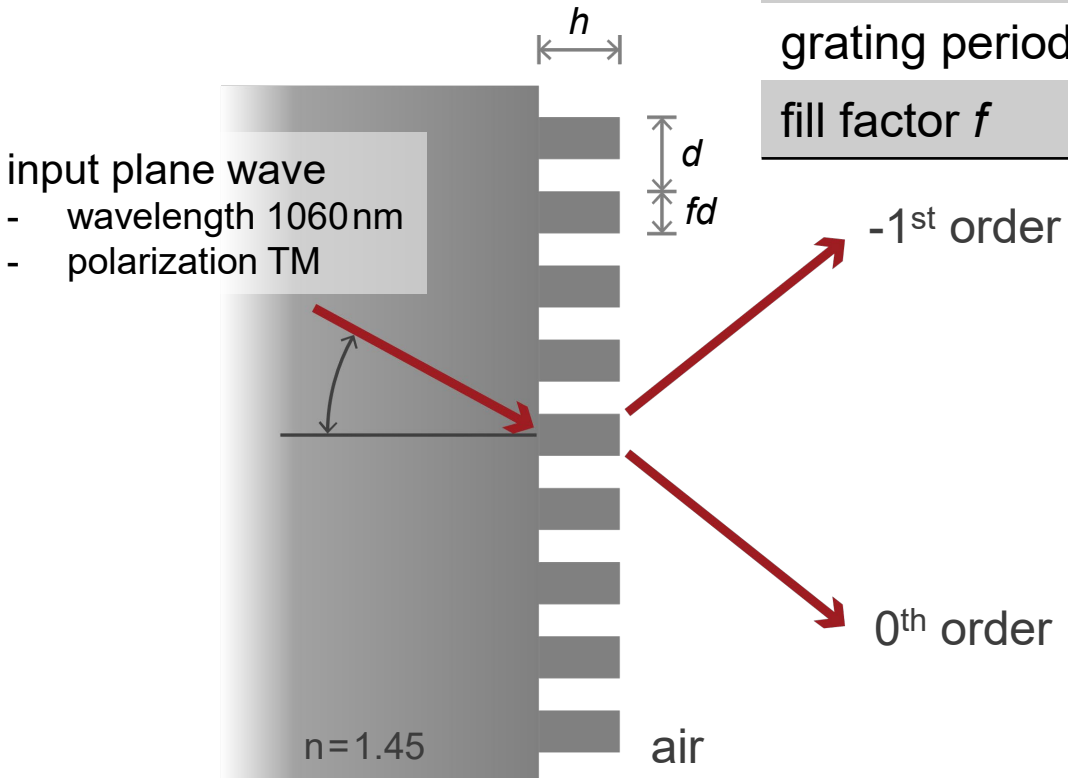


cross-platform simulation

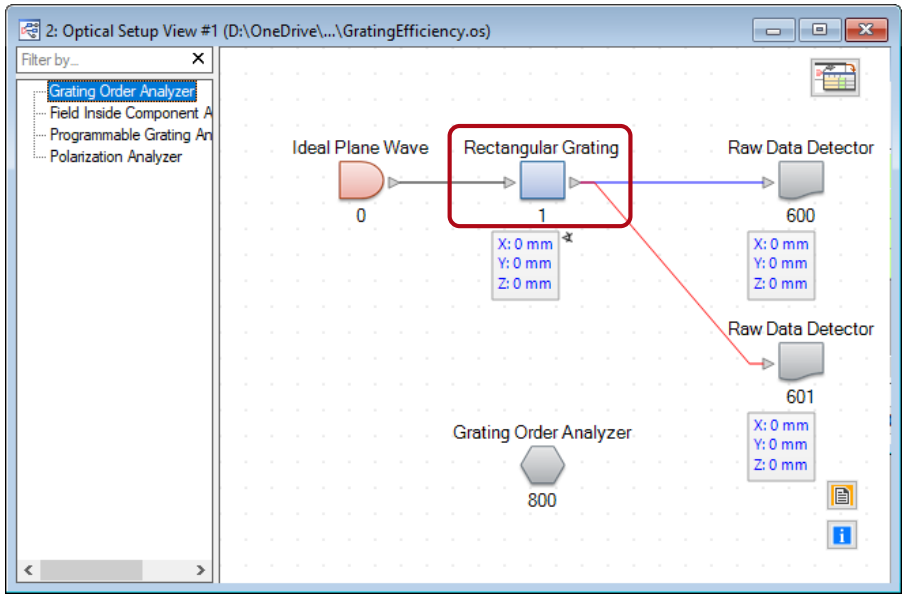
Define Optical Setup in VirtualLab Fusion

initial grating parameters

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

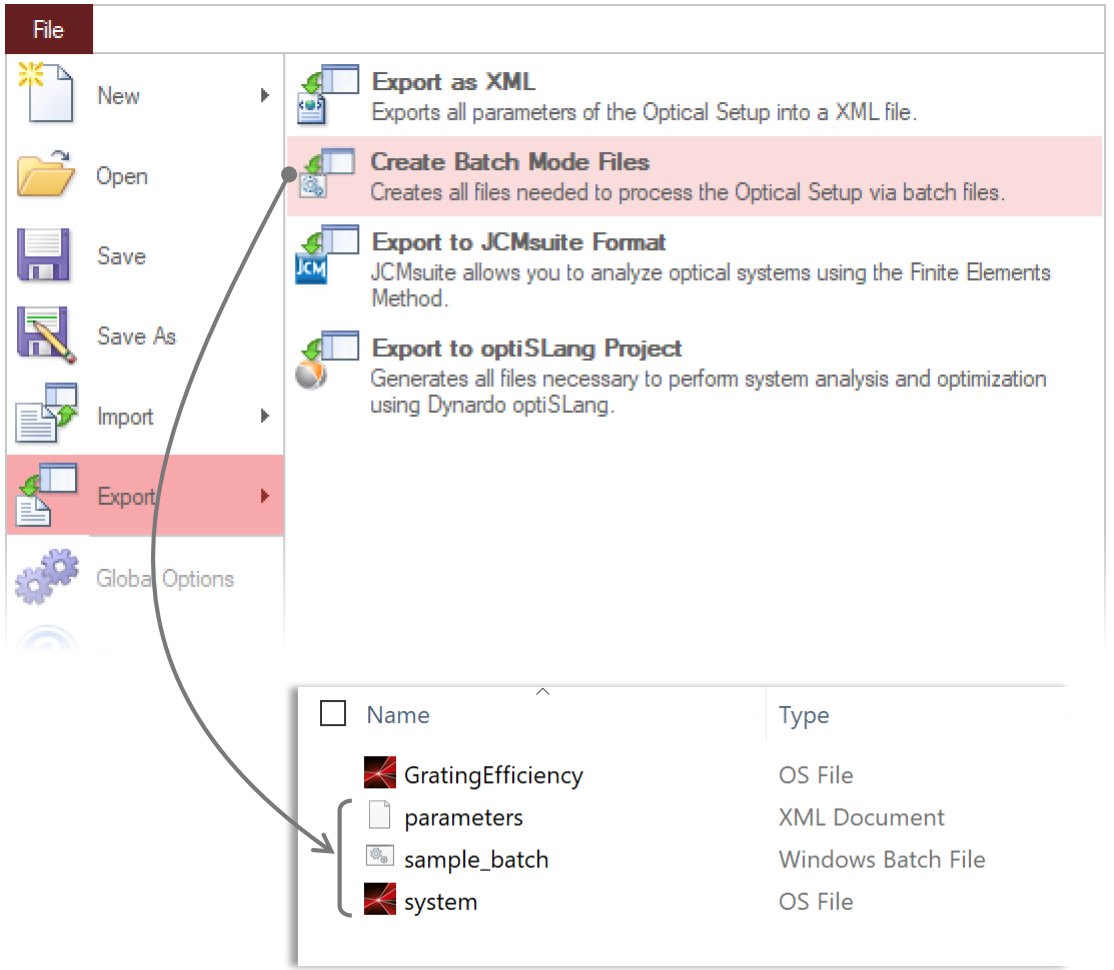


corresponding optical setup generated in VirtualLab



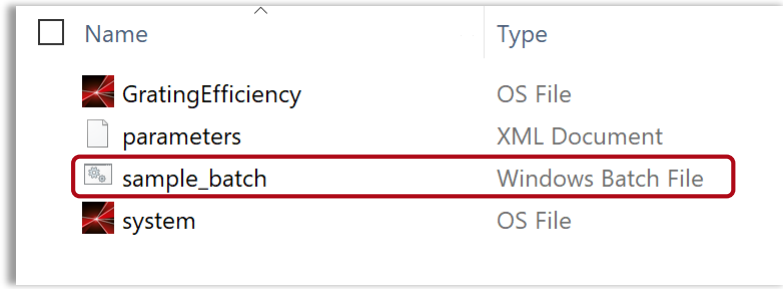
Create Batch Mode Files

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



Modify Batch File

- Open the batch file in e.g. Notepad
  - delete the output option
    (in this example, no subfolder)
  - and modify simulation engine
    (in this example, only use Grating Order Analyzer)



delete the line for Classic Field Tracing →

sample_batch - Notepad [original batch file]

File Edit Format View Help

~~:\SingleSetupExample\parameters.xml" -engine 2 -subfolder & REM Classic Field Tracing~~

:\SingleSetupExample\parameters.xml" -engine 800 ~~-subfolder~~ & REM Grating Order Analyzer

delete subfolder option

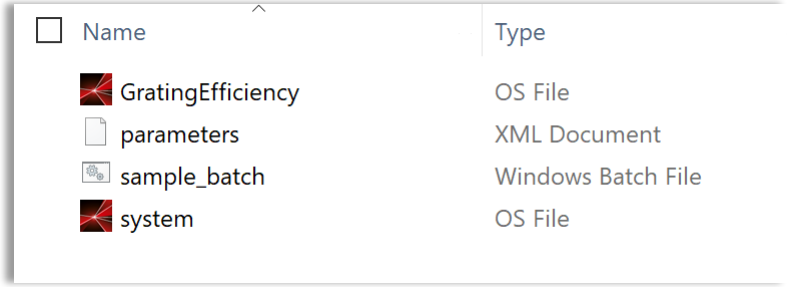
sample_batch - Notepad [modified batch file]

File Edit Format View Help

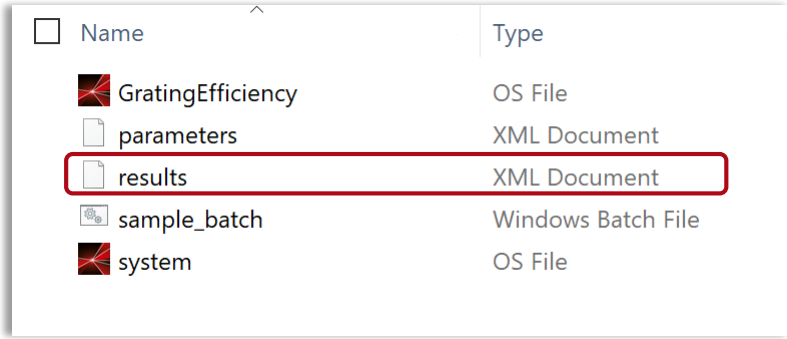
'C:\MatVLF\SingleSetupExample\parameters.xml" -engine 800 & REM Grating Order Analyzer

Execute Simulation Using Batch File

- It is recommended to execute the batch file first, 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.



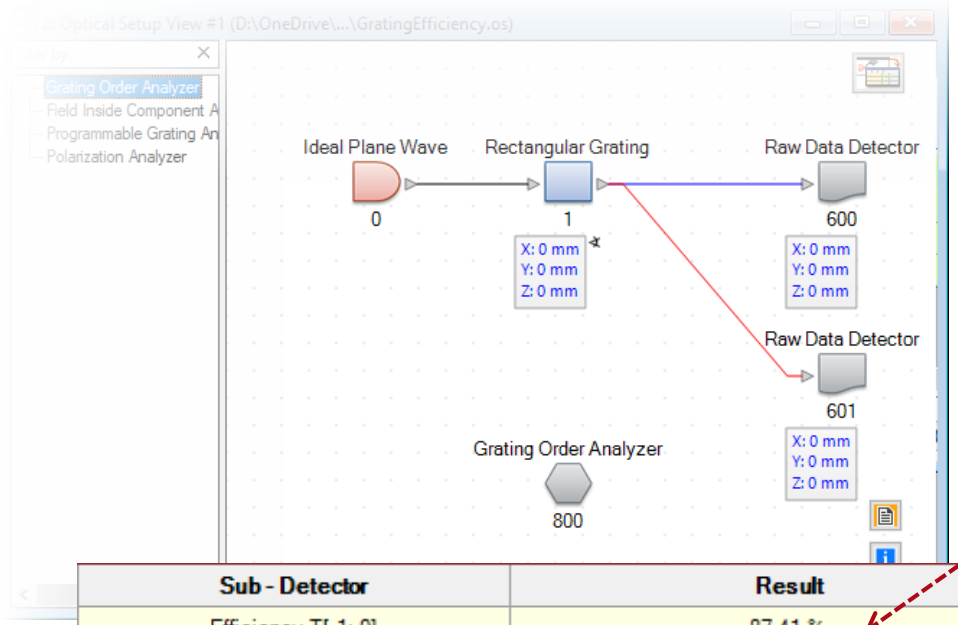
before executing batch file



after executing batch file

Execute Simulation Using Batch File

- Results in VirtualLab Fusion



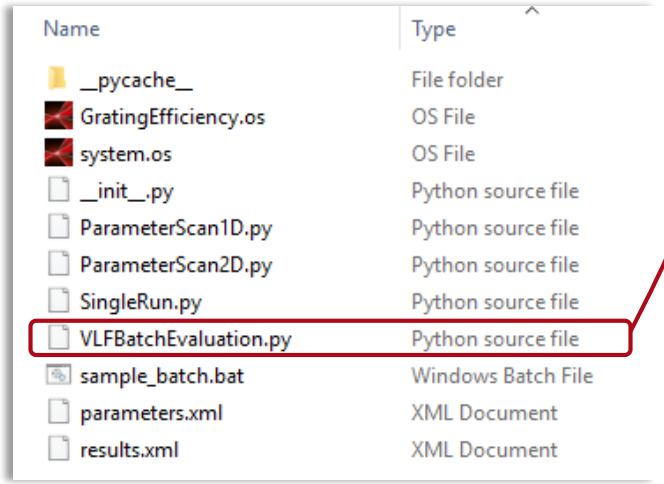
Sub - Detector	Result
Efficiency T[-1; 0]	87.41 %
Efficiency T[0; 0]	10.331 %
Efficiency T[+1; 0]	0 %

- Results in xml file

```
<?xml version="1.0" encoding="UTF-8"?>
- <Detector_Results Engine="Grating Order Analyzer">
    <VLVersion Version="7.5.0.158"/>
  - <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>87.409800037149679</Value>
        <Unit>%</Unit>
      </Result>
    - <Result Index="1">
        <Name>Efficiency T[0; 0]</Name>
        <Value>10.330826275343453</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 interacting and executing the batch file and related xml files.
- Copy VLFBatchEvaluation.py file directly to the working folder.

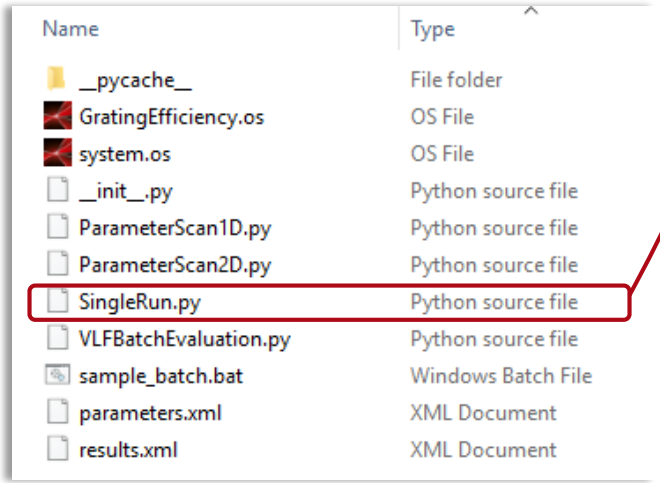


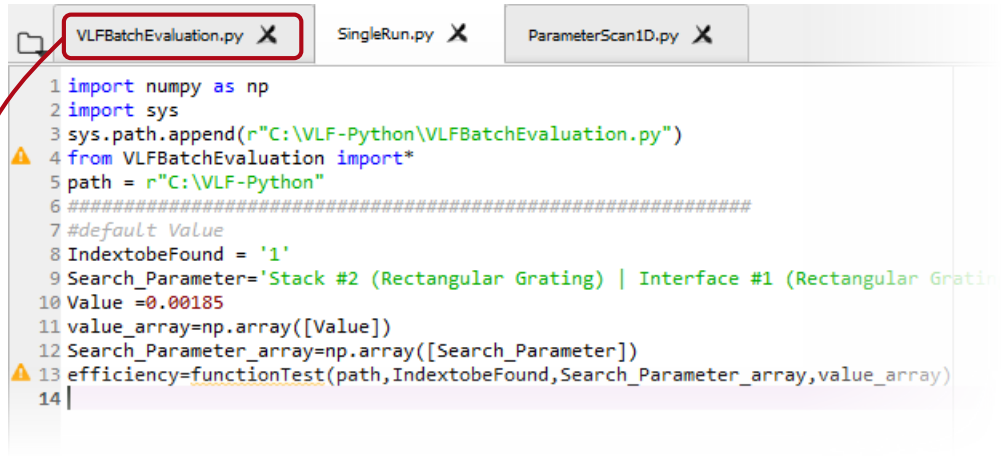
VLFBatchEvaluation.py | SingleRun.py | ParameterScan1D.py

```
def functionTest(path,IndextobeFound,Search_Parameter_array,Value_array):
    ##########################################################################
    if (len(Search_Parameter_array))==2:
        tree = ET.parse(path + r'\parameters.xml')
        root = tree.getroot()
        Found = 0
        List_of_Index = []
        for elem in root.iter('LightPathElement'):
            List_of_Index =elem.attrib['Index']
            if(List_of_Index==IndextobeFound):
                    for List_of_Index in elem.iter('Parameter'):
                        Found = 0
                        for subelem in List_of_Index:
                            if (subelem.text == Search_Parameter_array[0]):
                                Found = 1
                            if (Found == 1):
                                if (subelem.tag == 'Value'):
                                    subelem.text = str(Value_array[0])


        tree.write(path + r'\parameters.xml',encoding="UTF-8",xml_declaration=True)

        tree = ET.parse(path + r'\parameters.xml')
        root = tree.getroot()
        Found = 0
        List_of_Index = []
        for elem in root.iter('LightPathElement'):
            List_of_Index =elem.attrib['Index']
            if(List_of_Index==IndextobeFound):
                    for List_of_Index in elem.iter('Parameter'):
                        Found = 0
                        for subelem in List_of_Index:
                            if (subelem.text == Search_Parameter_array[1]):
                                Found = 1
                            if (Found == 1):
                                if (subelem.tag == 'Value'):
                                    subelem.text = str(Value_array[1])
        tree.write(path + r'\parameters.xml',encoding="UTF-8",xml_declaration=True)
```

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.





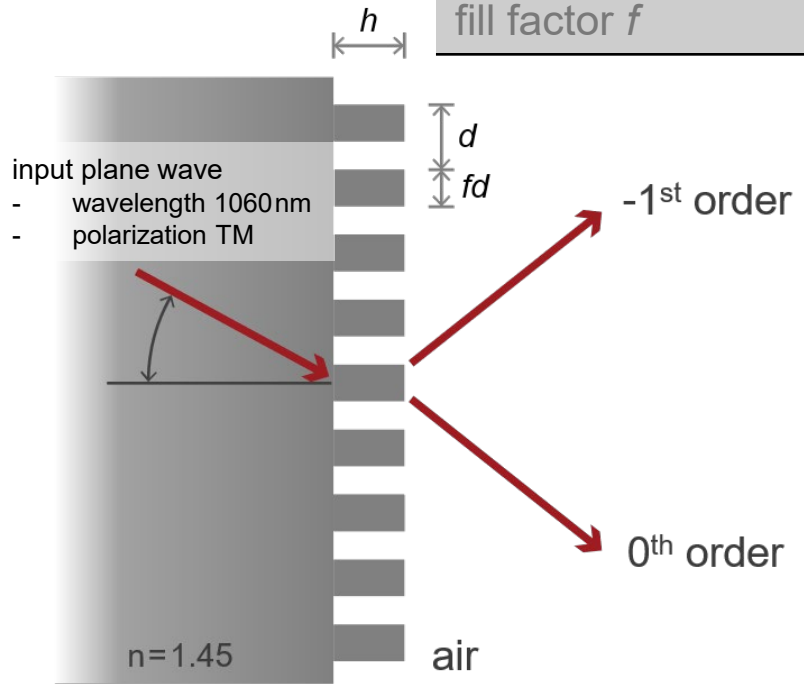
In this example, the -1st order efficiency is displayed after executing the function

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 scanning a selected parameter in the optical setup, and to check the influence on the result.
- In this example, grating depth is varied, and the diffraction efficiency of -1st order is under investigation.

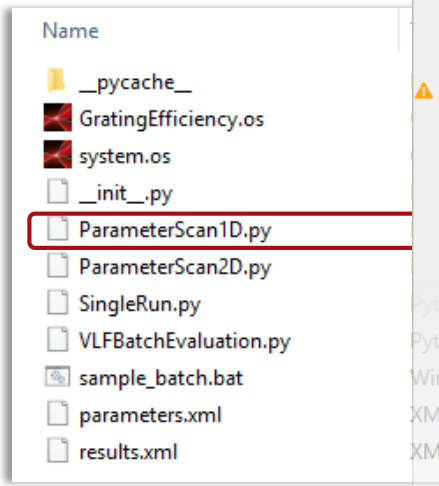
grating parameters

Parameter	Value
grating depth *h*	**0.1-10 µm**
grating period *d*	1060 nm
fill factor *f*	50 %



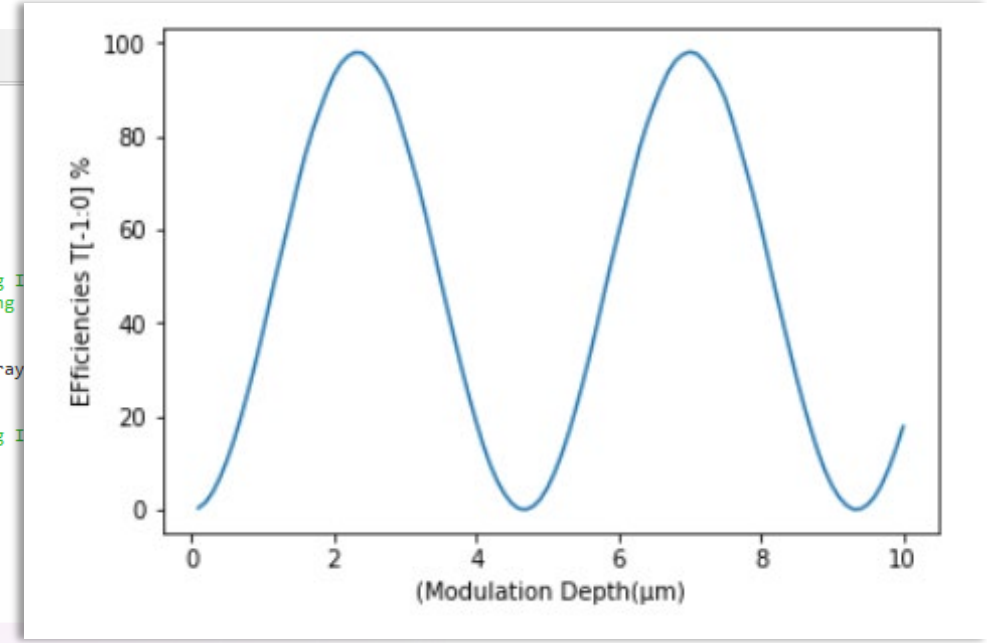
Parameter Scanning – Varying Single Parameter

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



VLFBatchEvaluation.py | SingleRun.py | ParameterScan1D.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) | Interface #1 (Rectangular Grating I
Search_Parameter1='Stack #2 (Rectangular Grating) | Interface #1 (Rectangular Grating
Search_Parameter_array1=np.array((Search_Parameter,Search_Parameter1))
Set_default_Value_array=((1.850000000000001,50))
print(functionTest(path,IndextobeFound,Search_Parameter_array1,Set_default_Value_array
#################################################################
IndextobeFound = '1'
Search_Parameter='Stack #2 (Rectangular Grating) | Interface #1 (Rectangular Grating I
Search_Parameter_array=np.array([Search_Parameter])

# define varying range
gratingDepthMax = 0.010 # given in micrometer
gratingDepthMin = 0.0001 # given in micrometer
stepNumber = 50
stepSize = (gratingDepthMax - gratingDepthMin) / (stepNumber - 1)
efficiencies = np.zeros(stepNumber)
gratingDepth_array = np.zeros(stepNumber)
print(efficiencies)
# loop for input variables
for i in range(stepNumber):
    #current value of variable
    gratingDepth = gratingDepthMin +  (i) * stepSize
    # call function
    print("Modulation depth", gratingDepth)
    Pass_Value=np.array([gratingDepth])
    print("Pass_Value", Pass_Value)
    print(len(Pass_Value))
    efficiencies=[functionTest(path,IndextobeFound,Search_Parameter_array,Pass_Value)]
```

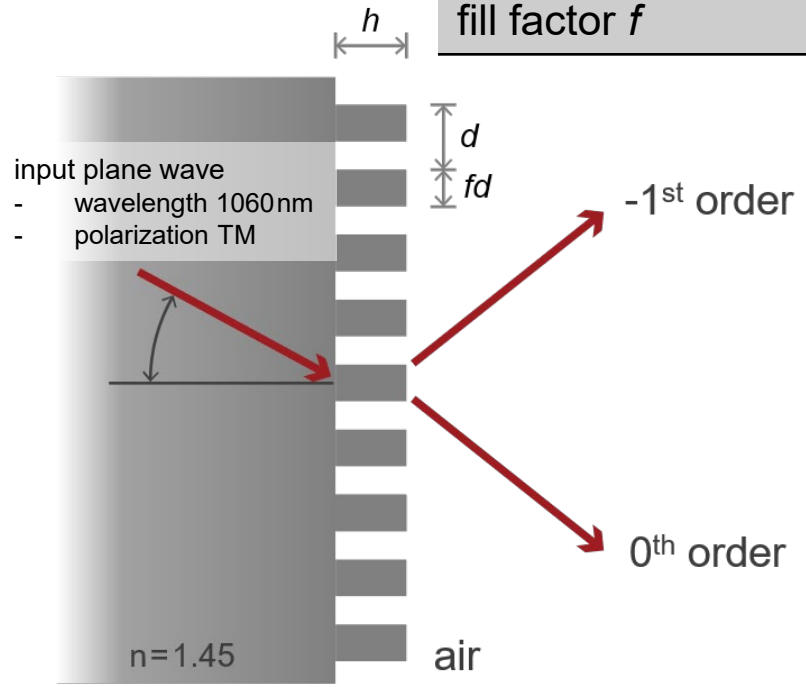


Parameter Scanning – Varying Multiple Parameters

- 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 the diffraction efficiency of -1st order is under investigation.

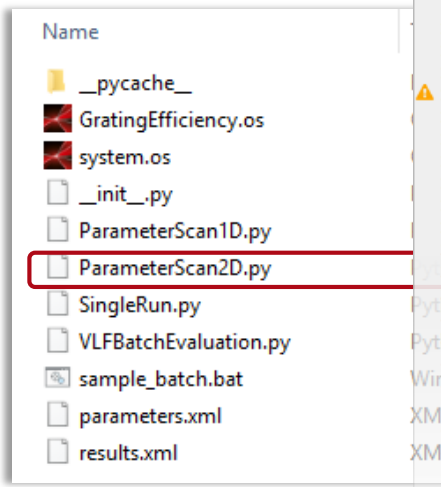
grating parameters

Parameter	Value
grating depth *h*	**0.1-10µm**
grating period *d*	1060nm
fill factor *f*	**20-80%**



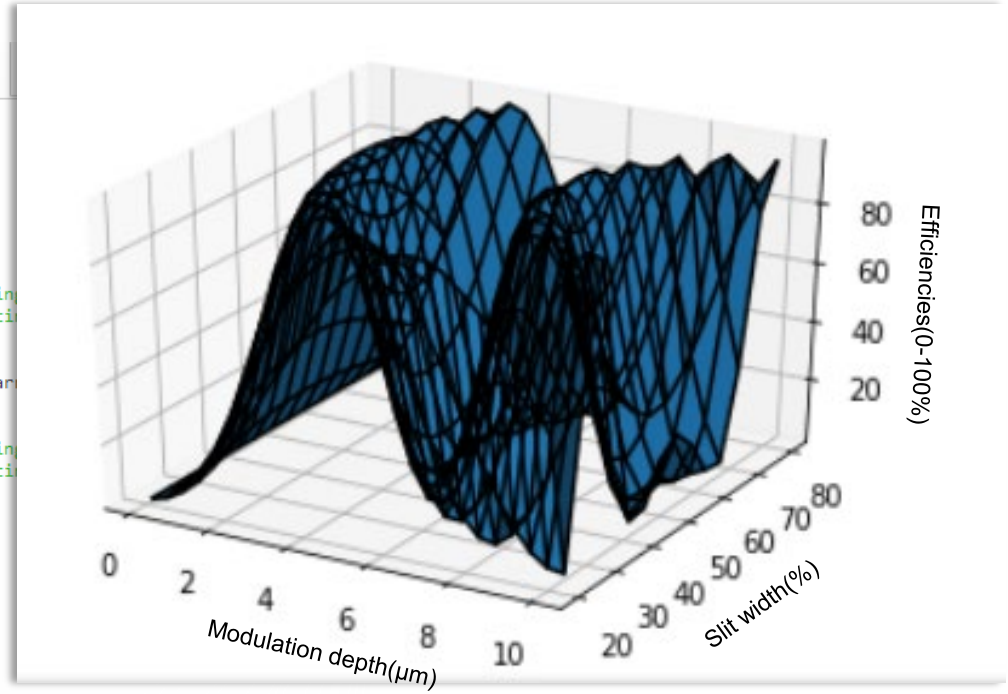
Parameter Scanning 2D – Varying Multiple Parameters

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



VLFBatchEvaluation.py | SingleRun.py | ParameterScan1D.py | ParameterScan2D.py

```
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) | Interface #1 (Rectangular Gratin
Search_Parameter1='Stack #2 (Rectangular Grating) | Interface #1 (Rectangular Grati
Search_Parameter_array1=np.array((Search_Parameter,Search_Parameter1))
Set_default_Value_array=((1.850000000000001,50))
print(functionTest(path,IndextobeFound,Search_Parameter_array1,Set_default_Value_ar
################################################################
IndextobeFound = '1'
Search_Parameter='Stack #2 (Rectangular Grating) | Interface #1 (Rectangular Gratin
Search_Parameter1='Stack #2 (Rectangular Grating) | Interface #1 (Rectangular Grati
Search_Parameter_array=np.array((Search_Parameter,Search_Parameter1))
# define varying range
gratingDepthMax = 10 # given in micrometer
gratingDepthMin = 0.1 # given in micrometer
stepNumberDepth = 25
stepSizeDepth = (gratingDepthMax - gratingDepthMin) / (stepNumberDepth - 1)
slitWidthMax = 80 # given in %
slitWidthMin = 20 # given in %
stepNumberSlit = 15
stepSizeSlit = (slitWidthMax - slitWidthMin) / (stepNumberSlit - 1)
efficiencies = np.zeros((stepNumberDepth,stepNumberSlit))
gratingDepth_array = np.zeros((stepNumberDepth,stepNumberSlit))
slitWidth_array = np.zeros((stepNumberDepth,stepNumberSlit))
print(efficiencies)
# loop for input variables
for i in range(stepNumberDepth):
```



Document Information

title	Cross-Platform Optical Modeling and Design with VirtualLab Fusion and Python
document code	CPF.0002
version	1.0
toolbox(es)	(depending on situation; Grating Toolbox used for this example)
VL version used for simulations	VirtualLab Fusion Spring Release 2019
category	Feature Use Case
further reading	- Cross-Platform Optical Modeling and Design with VirtualLab Fusion and MATLAB