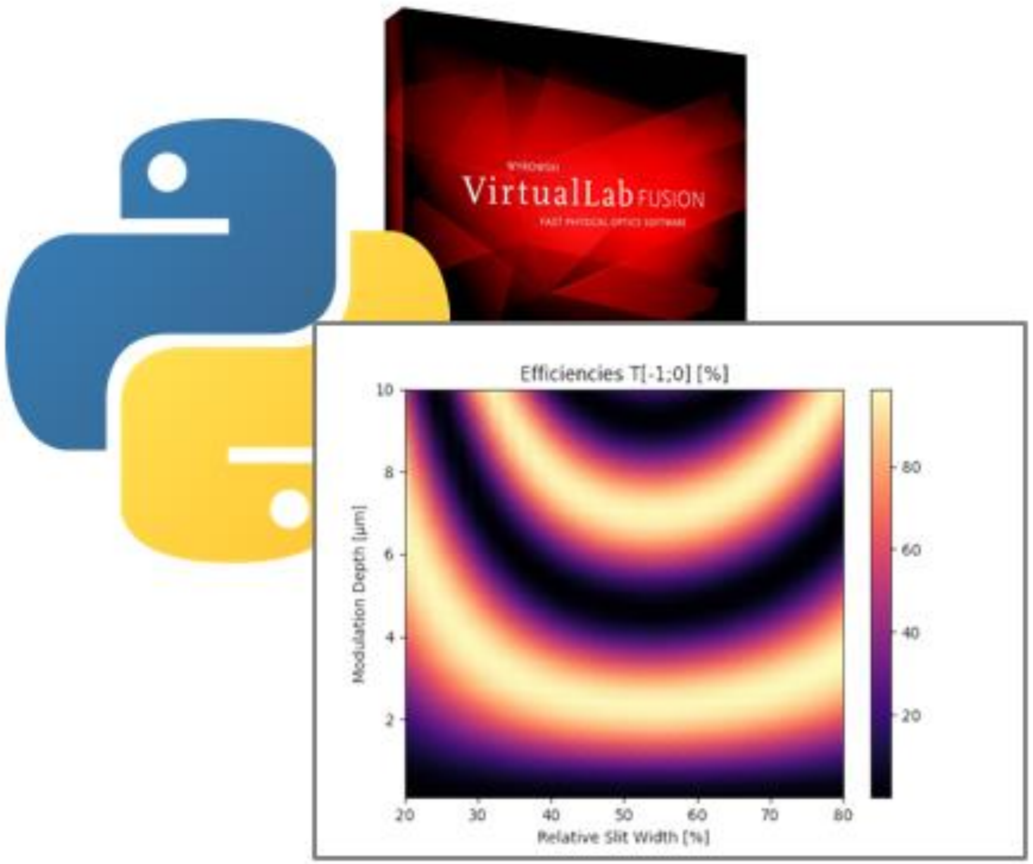
Cross-Platform Parameter Sweep with Python

Abstract



VirtualLab Fusion allows external access to its modeling technology, solvers and results. This is helpful in order to be able to apply other data processing or optimization tools to further investigate optical simulations. In this example, we demonstrate how to run a parameter sweep using a Python script and how to collect the results, which can be further processed with all the capabilities offered by Python. Exemplarily, the diffraction efficiency of a grating is analyzed rigorously.

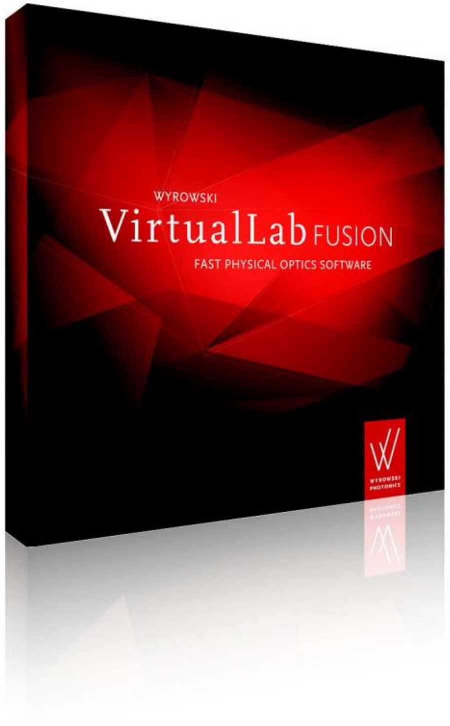
This Use Case Shows…

Python

- external functions

VirtualLab Fusion

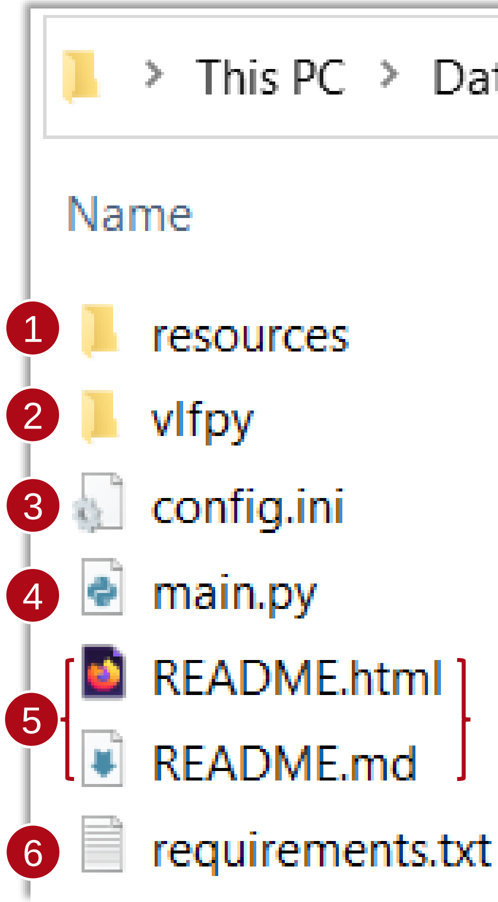
- optical setup definition
- fast physical optics simulation engine



cross-platform simulation

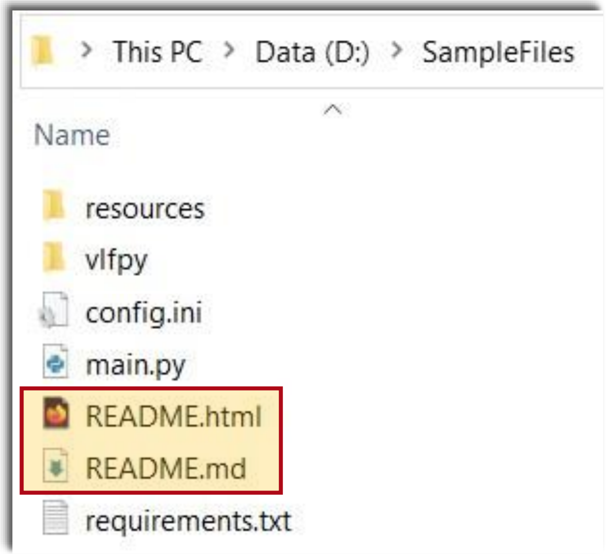
Where to Find The Files

The user can find all files in the folder *SampleFiles*. This archive with the files can be downloaded from our website.

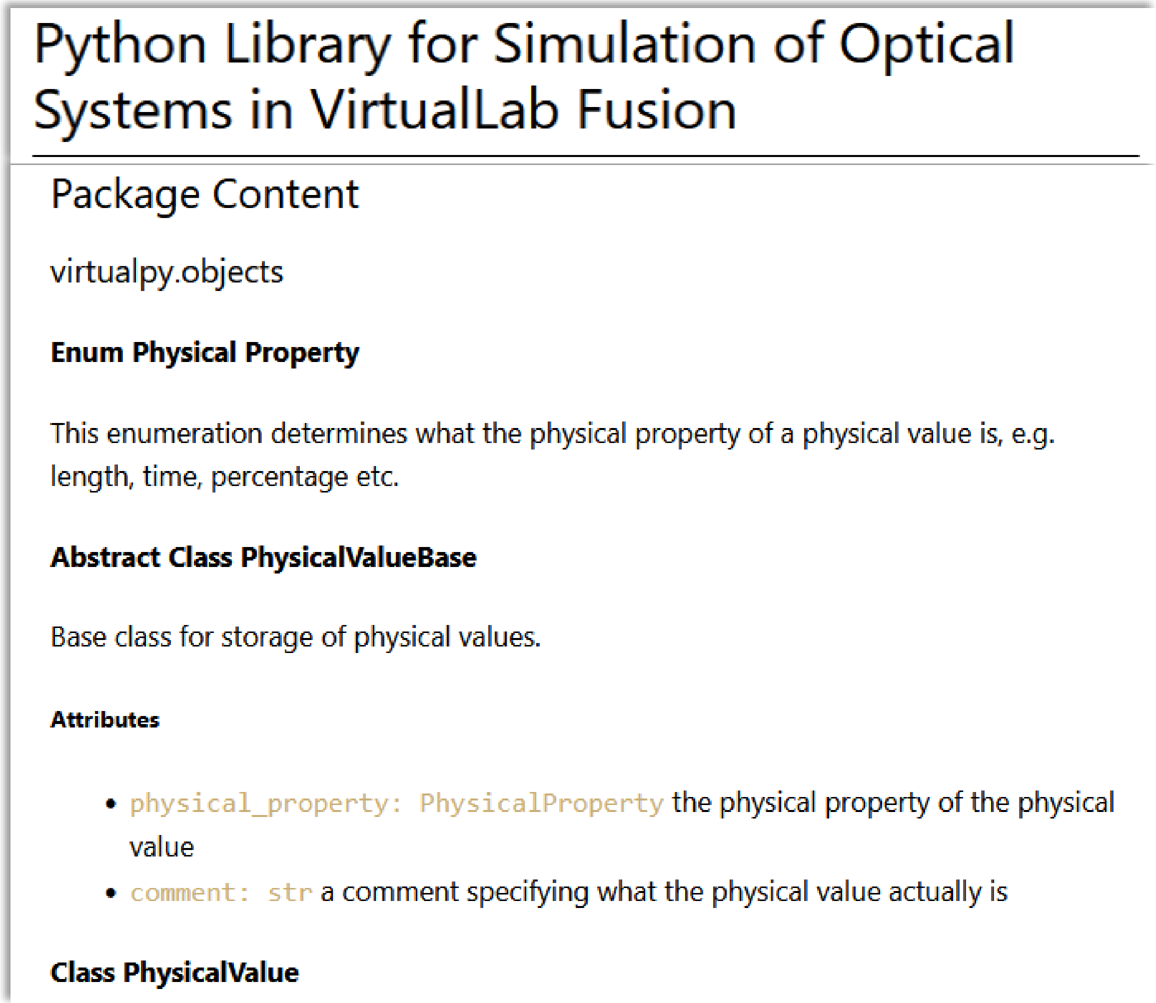


1. *The optical setup to be run is saved here*
2. *The Python library for simulation of optical setups*
3. *Save the directory of your VirtualLab Fusion here*
4. *Use this file to run your simulations*
5. *Important information*
6. *Get the required packages here*

README File



In the README file, you can find a brief overview of the configuration settings and the functions used in the Python library for this use case.



Python Library for Simulation of Optical Systems in VirtualLab Fusion

Package Content

virtualpy.objects

Enum Physical Property

This enumeration determines what the physical property of a physical value is, e.g. length, time, percentage etc.

Abstract Class PhysicalValueBase

Base class for storage of physical values.

Attributes

- `physical_property: PhysicalProperty` the physical property of the physical value
- `comment: str` a comment specifying what the physical value actually is

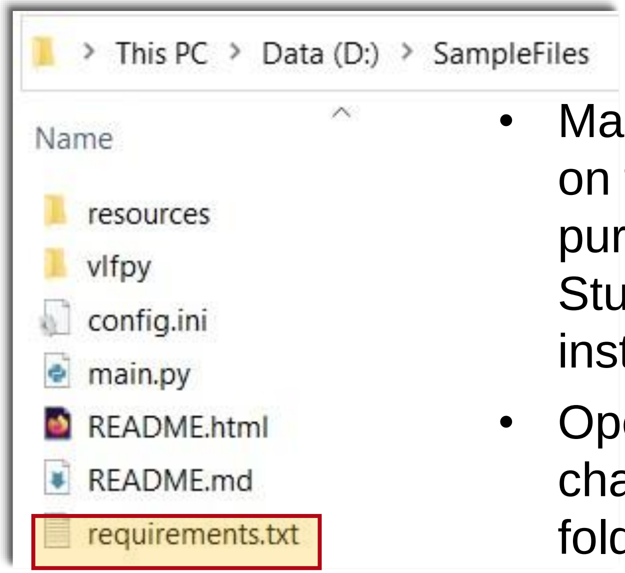
Class PhysicalValue

Prepare Python

Make sure that **Python*** is installed on the computer. Notice that the option **Add python.exe to PATH** should be selected for installation.

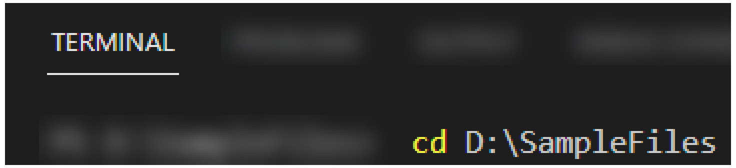
* This Use Case has been created with Python 3.11.0.
Python Release Python 3.11.0 | Python.org

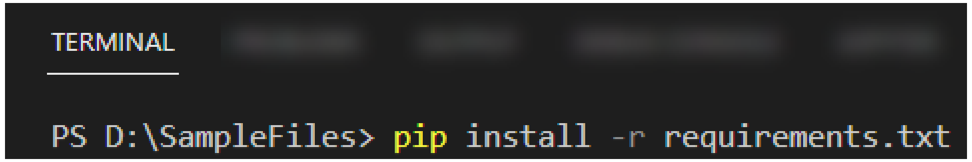
Prepare Python



- Make sure that **Python 3.11.0** is installed on the computer. For demonstration purposes, we use the code editor Visual Studio Code as it offers a user-friendly installation workflow*.
- Open the Python command line and change directory to the *SampleFiles* folder.
- The names of all required packages are saved in the file requirements.txt, run the following command to make sure that all these packages are installed:
  pip install -r requirements.txt

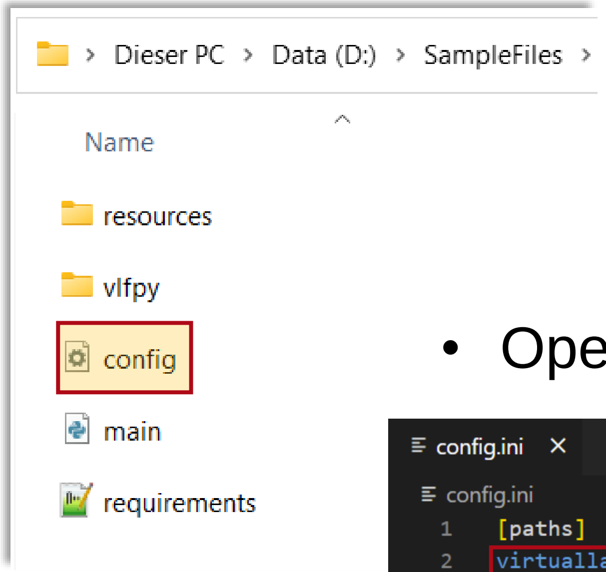
For a step-by-step tutorial on how to connect Python to VirtualLab Fusion, please see:
[Execute an Optical Simulation in VirtualLab Fusion with Python]



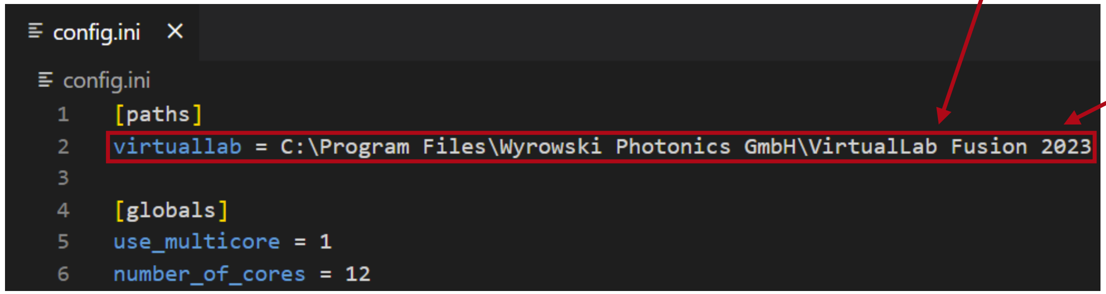


* For further information of the code editor Visual Studio Code for Python please read:
https://code.visualstudio.com/docs/python/python-tutorial
Note that here we demonstrate the installation of the required packages in the global environment. For users working with multiple Python projects, it is recommended to use project-specific virtual environments. Please also refer to the tutorial in the link above to create a virtual environment and install the required packages.

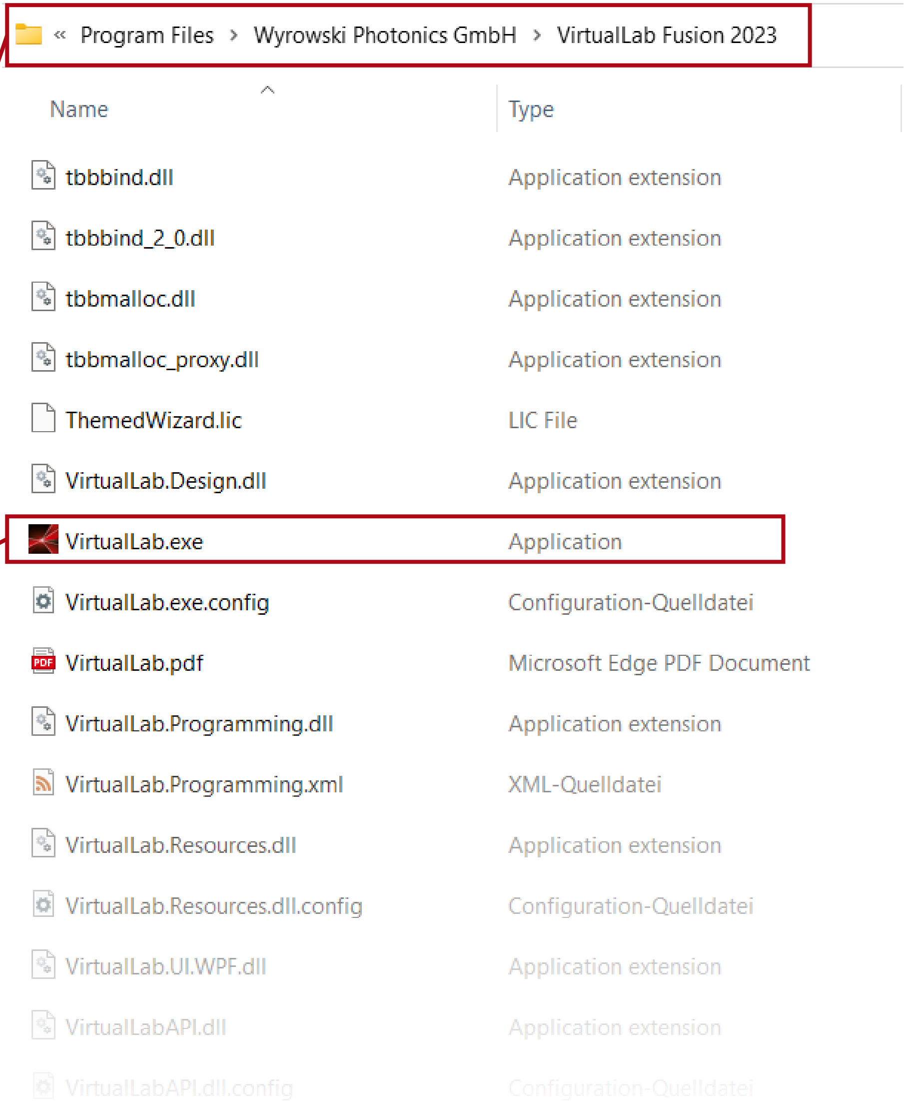
Configure the Path



- Open the config.ini file.



- Set the directory of your VirtualLab Fusion installation (the folder which contains the VirtualLab.exe).



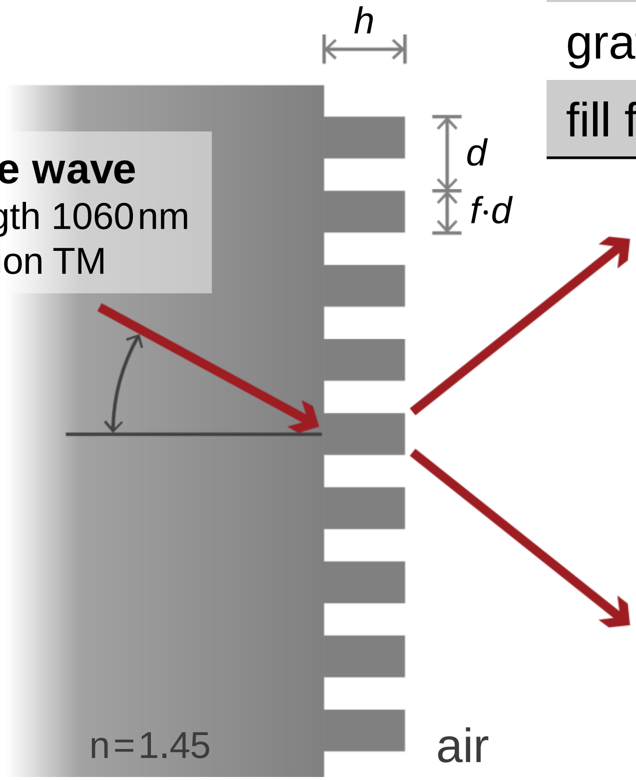
Define an Optical Setup in VirtualLab Fusion

See the full example:
[Stretching or Compression of Ultrashort Pulses with Highly Efficient Transmission Gratings]

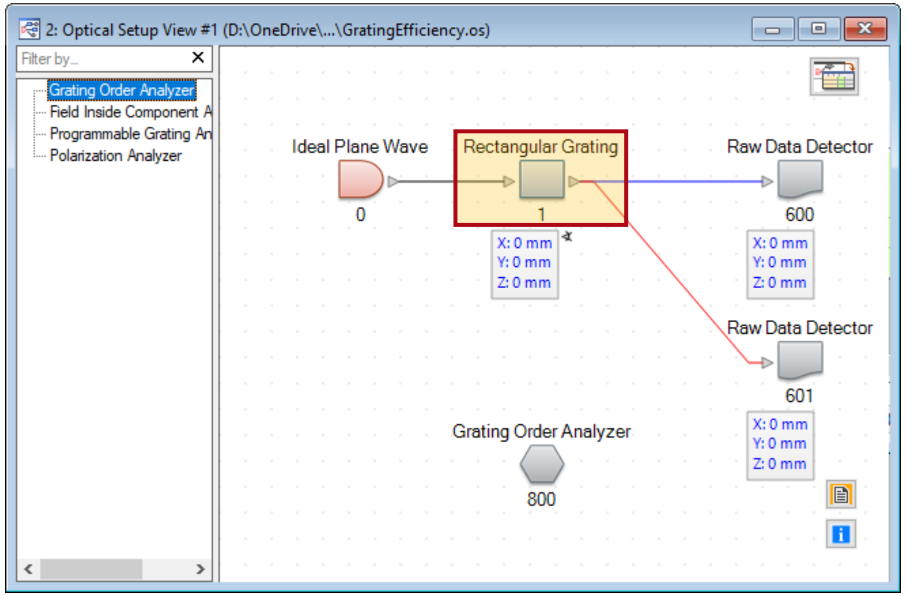
initial grating parameters

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

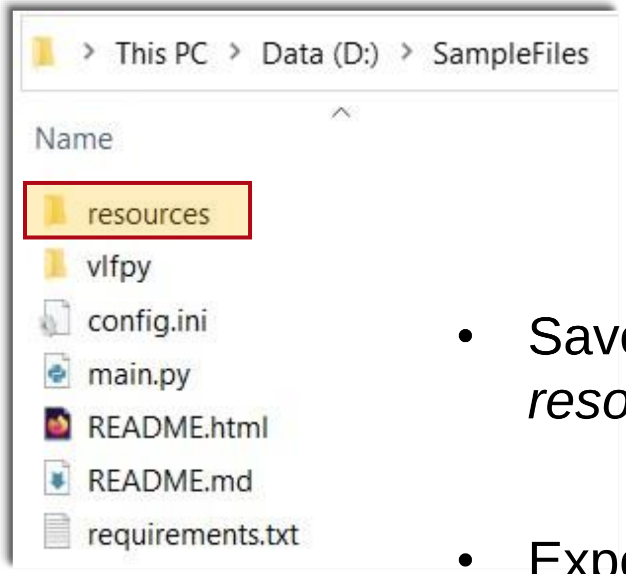
input plane wave
- wavelength 1060 nm
- polarization TM



corresponding optical setup generated in VirtualLab

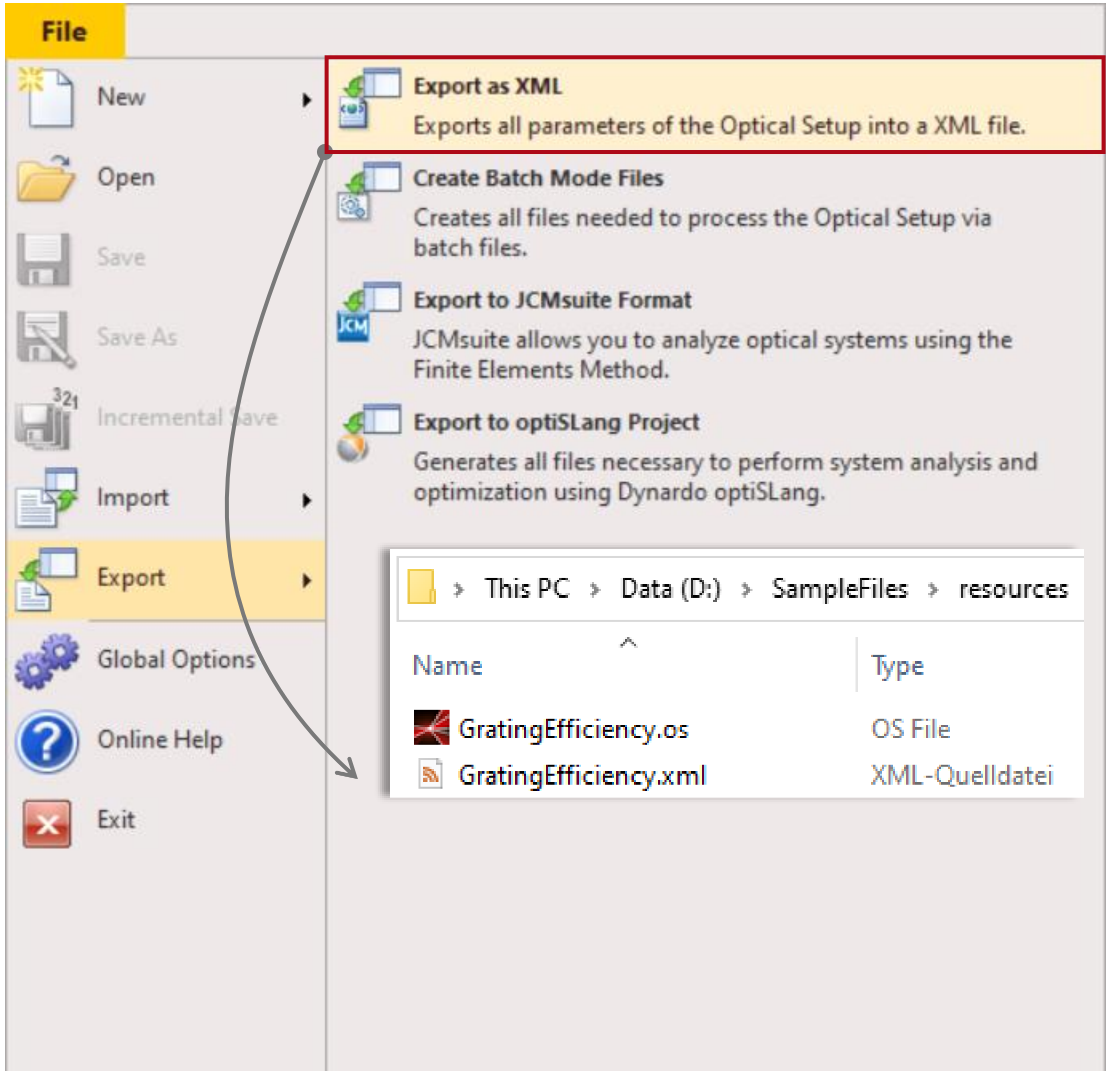


Save the Optical Setup and Export Parameters Into an XML File

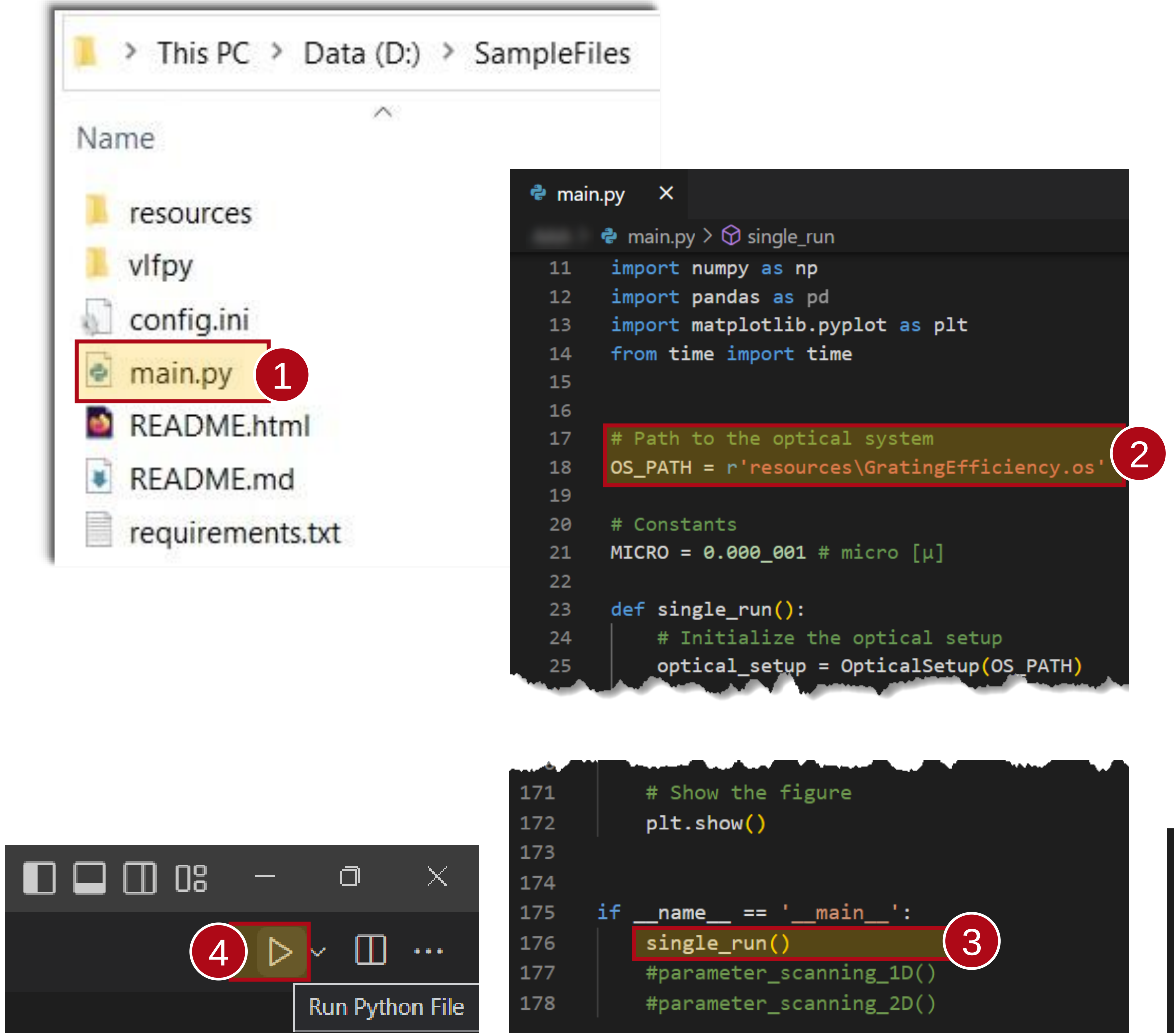


- Save the .OS - file in the *resources* folder.
- Export the .XML file to the *resources* folder.

  GratingEfficiency.xml
  xml file containing all parameters of the optical setup from VirtualLab



Run the Simulation



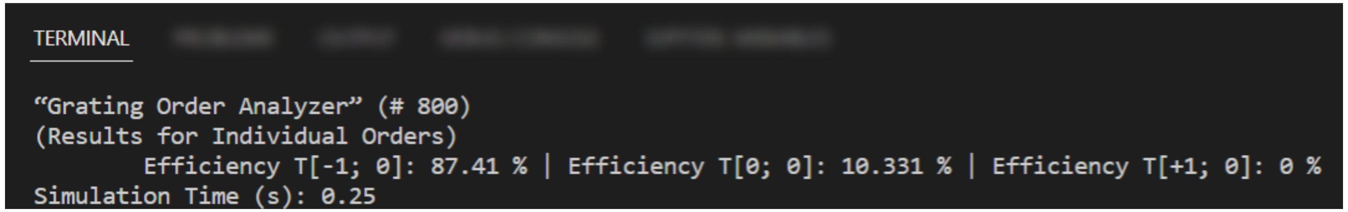
1.) Open the main.py file.

2.) Set the path to that of the optical setup to be evaluated. In this case, as mentioned in the previous page, the optical setup is saved in the *resources* folder.

3.) single_run() is uncommented to simulate the optical setup.

4.) Press the play button at the upper right corner of the window to run the code.

In this example, the -1st, 0th and 1st order efficiencies are displayed after executing the function.

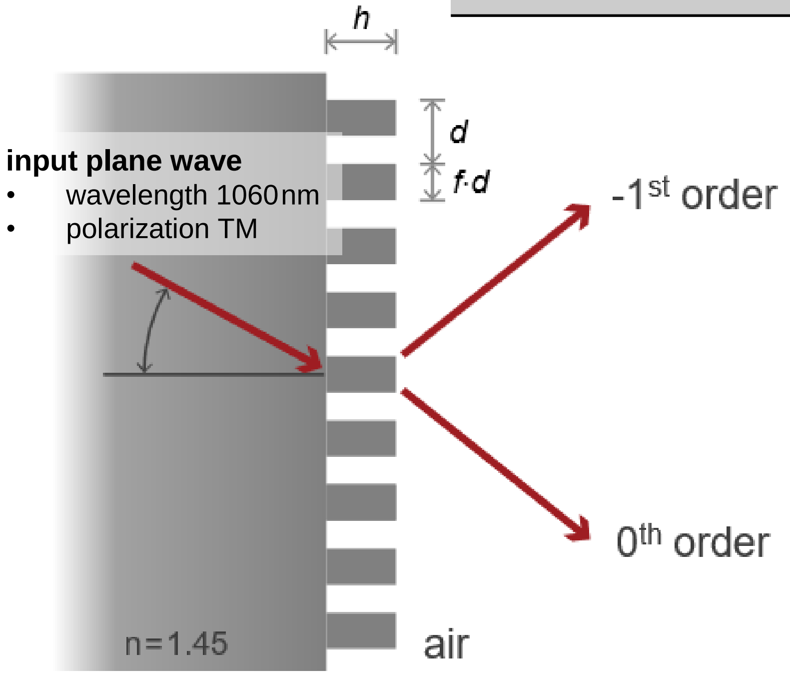


Parameter Scanning – Varying Single Parameter

- As an example, we demonstrate how to scan a selected parameter in the optical setup and to check its impact on the results.
- In this example, the grating depth is varied and the diffraction efficiency of the -1st and 0th transmission orders are evaluated.

grating parameters

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



See the full Use Case:
[Stretching or Compression of Ultrashort Pulses with Highly Efficient Transmission Gratings]

Parameter Scanning – Varying Single Parameter

main.py ×

[redacted] main.py > parameter_scanning_1D

```python
def parameter_scanning_1D():
    # Initialize the optical setup
    optical_setup = OpticalSetup(OS_PATH)

    # Define values for modulation depth, here: 0.1 µm - 10 µm, 100 steps
    values = np.linspace(0.1 * MICRO, 10 * MICRO, 100)

    # Define the parameter set (further information in README.html)
    # 1. parameter: Index attribute of the LightPathElement, here: Rectangular Grating
    # 2. parameter: ID tag of the parameter to be scanned
    # 3. parameter: the already defined values
    params = ParameterSet(1, 'Stack2.LayersAsArray[0].Interface.ModulationDepth', values)
```

- Set the range of values and the number of steps for scanning here.
- Set the index of the LightPathElement and ID of the parameter to be scanned here.

```xml
<LightPathElement Index="1" Name="Rectangular Grating">

        <Name>Stack #2 (Rectangular Grating) | Surface #1 (Rectangular Grating Interface) | Modulation Depth</Name>
        <ID>Stack2.LayersAsArray[0].Interface.ModulationDepth</ID>
        <ShortName>Modulation Depth</ShortName>
        <Value>1.8500000000000001e-06</Value>
        <Unit>m</Unit>
```

- The information of LightPathElement index and parameter ID can be found in xml file saved in the *resources* folder.

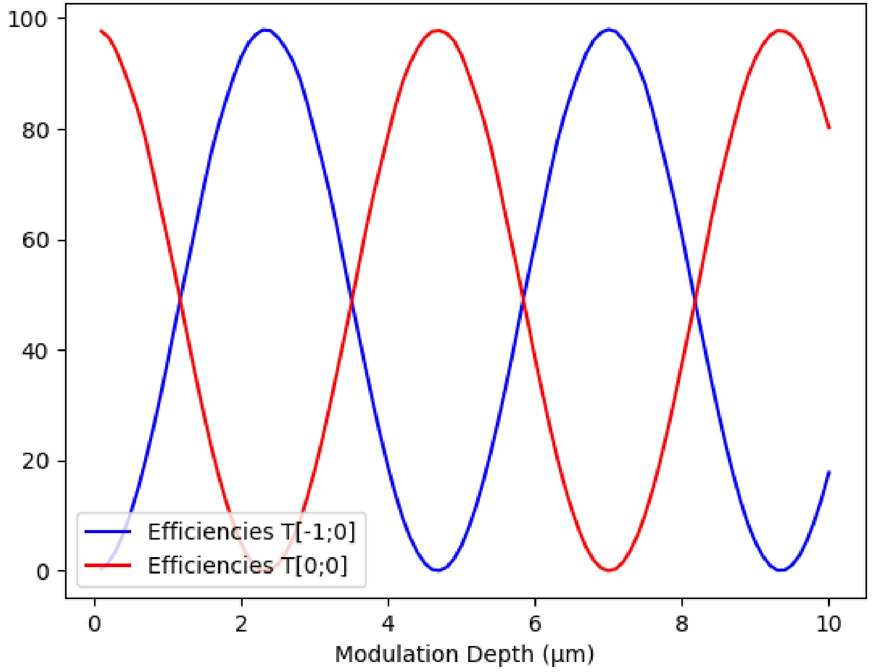
Parameter Scanning – Varying Single Parameter

main.py

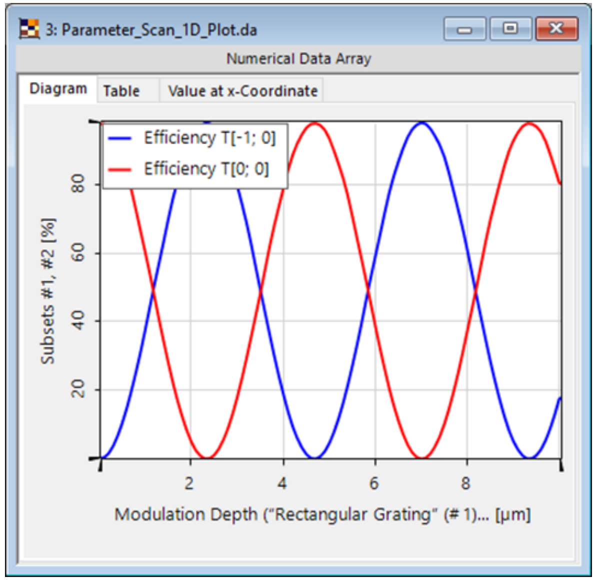
```python
    # Show the figure
    plt.show()


if __name__ == '__main__':
    #single_run()
    parameter_scanning_1D()
    #parameter_scanning_2D()
```

parameter_scanning_1D() in main.py file is uncommented to perform a 1D parameter scanning.
In this example, the grating depth is varied and the diffraction efficiency of the -1st and 0th transmission orders are evaluated.



For comparison, this is the result if the parameter run is performed in VLF directly:

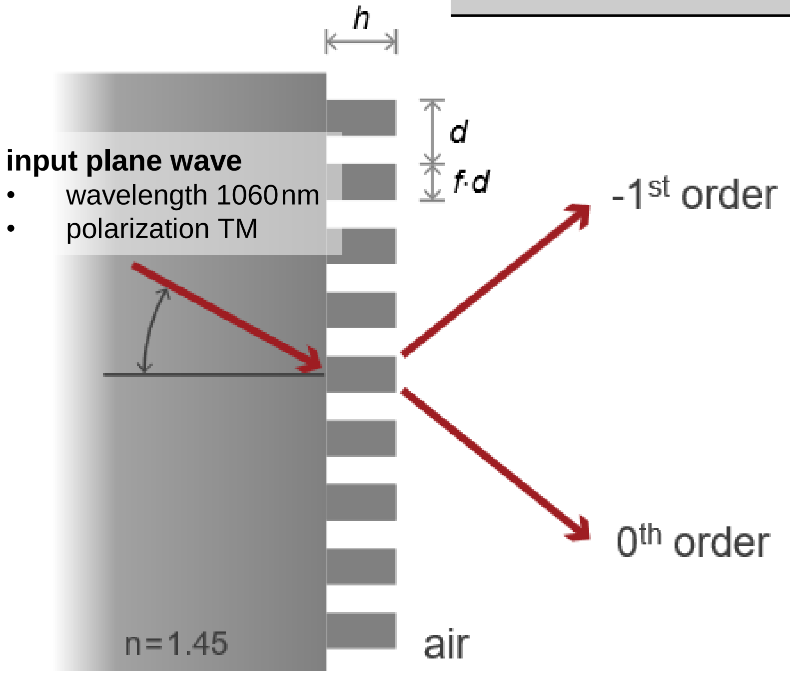


Parameter Scanning – Varying Multiple Parameters

- One can also 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 the -1st order is investigated.

grating parameters

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



See the full Use Case:
[Stretching or Compression of Ultrashort Pulses with Highly Efficient Transmission Gratings]

Parameter Scanning – Varying Multiple Parameters

main.py

main.py > parameter_scanning_2D

```python
def parameter_scanning_2D():
    # Initialize the optical setup
    optical_setup = OpticalSetup(OS_PATH)

    # Define steps for modulation depth
    md_steps = 31

    # Define steps for relative slit width
    sw_steps = 31

    # Define values for modulation depth, here: 0.1 µm - 10 µm, 31 steps (+ 0.33 µm per step)
    md_values = np.linspace(0.1 * MICRO, 10 * MICRO, md_steps)

    # Define values for relative slit width, here: 20 % - 80 %, 31 steps (+ 2 % per step)
    sw_values = np.linspace(0.2, 0.8, sw_steps)

    # Print the number of iterations to the console
    print(f'Number of Iterations = {len(md_values) * len(sw_values)}\n')

    # Define the parameter set (further information in README.html)
    # 1. parameter: Index attribute of the LightPathElement, here: Rectangular Grating
    # 2. parameter: ID tag of the parameter to be scanned
    # 3. parameter: the already defined values
    md_params = ParameterSet(1, 'Stack2.LayersAsArray[0].Interface.ModulationDepth', md_values)
    sw_params = ParameterSet(1, 'Stack2.LayersAsArray[0].Interface.RelativeSlitWidth', sw_values)
```

- Set the range of values and the number of steps for scanning here.
- Set the index of the LightPathElement and ID of the parameter to be scanned here.
- The information of LightPathElement index and parameter ID can be found in xml file saved in the “resources” folder.

Parameter Scanning – Varying Multiple Parameters

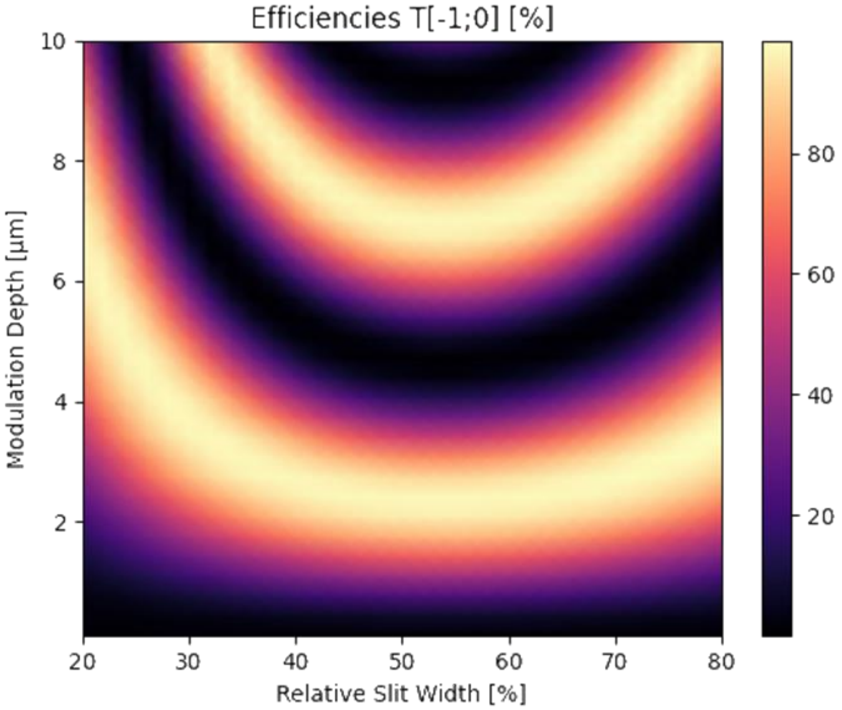
main.py

main.py > ...

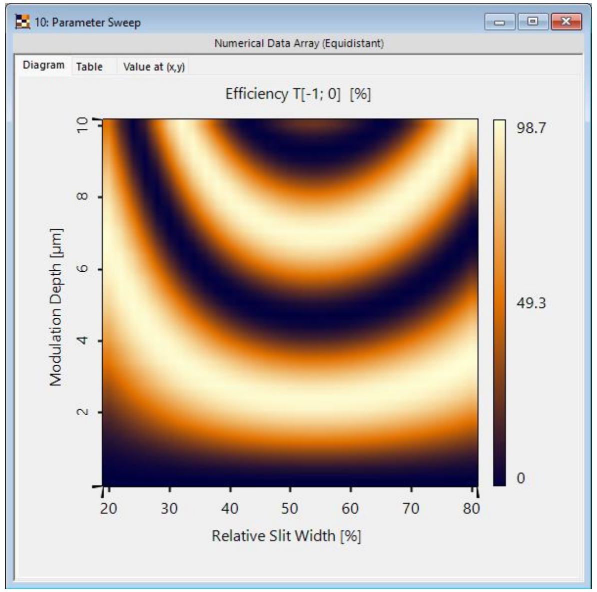
```python
    # Show the figure
    plt.show()


if __name__ == '__main__':
    #single_run()
    #parameter_scanning_1D()
    parameter_scanning_2D()
```

parameter_scanning_2D() in main.py file is uncommented to perform a 2D parameter scanning.
In this example, the grating depth and the fill factor are varied, the diffraction efficiency of the -1st order is evaluated.



For comparison, this is the result if the parameter run is performed in VLF directly:



Document Information

title	Cross-Platform Parameter Sweep with Python
document code	CPF.0005
version	1.0
toolbox(es)	(depending on optical setup; for this example VirtualLab Fusion Advanced)
• VLF version • Python version	• VirtualLab Fusion 2023.1 (Build 1.556) • Python 3.11.0
category	Feature Use Case
further reading	• Cross-Platform Optical Modeling and Design with VirtualLab Fusion and MATLAB • Execute an Optical Simulation in VirtualLab Fusion with Python • Stretching or Compression of Ultrashort Pulses with Highly Efficient Transmission Gratings