epf
ENGINEERING SCHOOL
Creating the future together

SolPOC

User Guide

SolPOC v0.9.7

Index

List of figures

List of Tables

I. Introduction

SolPOC, Solar Performance Optimization Code is a simple and fast code running under Python 3. The code is designed to solve Maxwell's equations in a multilayered thin film structure in 1 dimension. The package is specifically designed for research, industrial and academic research in optical coatings, thin film deposition or in solar energy applications (thermal, photovoltaic, etc.). The SolPOC code use a stable method (Abélès matrix) to quickly calculate the optical behavior (reflectivity, transmissivity, and absorptivity) from a stack of thin films deposited on a solid substrate over a full solar spectrum just from complex refractive indices of real materials. SolPOC comes with several optimization methods, specific cost functions for optic or solar energy applications and a comprehensive database of refractive indices for real materials.

In the end, SolPOC is simple to use for no-coder users thanks to main script, which regroup all necessary variables and automatically saves important results in text files and PNG images. Thanks to Python and the use of a multiprocessing pool most problems can be solved in a couple of minutes. This code can be used for scientific research or academic education. The present document aim is to be User Guide. It describes who the code works, how to use if, understand the major results and provides several examples. To assist users who are simply looking for specific information, this document is intentionally redundant. We still hope that it will be useful and enjoyable to read.

1.1. License

This program is free software: you can redistribute it and/or modify it under the terms of the **GNU General Public Licens**e as published by the Free Software Foundation, either version 3 of the License, or any later version.

This program is distributed in the hope that it will be useful, but without any a warranty, without even the implied warranty of merchantability of fitness for a particular purpose. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

1.2. Current version

The actual version of SolPOC is 0.9.7. This version is still under continuous development, and new "sub" version can be update (as 0.9.8 or other), when we added new functionality or bug fix. SOLPOC are tested under Windows, using Spyder, Visual Studio or PyCharm as IDE. Please report to the main author any bug or ideas for further implementation.

We propose the following logo, feel free to use it if relevant.

Figure 1 : SOLPOC logo

1.3. Code Origin and Purpose

The code has been developed on Scilab (version 5.6) during the main author's Ph.D. Thesis at PROMES CNRS (Perpignan, 66, France) between 2014 and 2018. The A.Grosjean Ph.D. Thesis has been defended with success March 7, 2018 [1]. The main aim of the Ph.D. thesis was to explore multiple pathways to improve the performance and if possible, reduce cost of the three types of surfaces encountered in solar thermal collectors: reflectors, antireflective windows and selective absorbers with thin film. For this purpose, the code was developed to study and maximize solar performance of the thin films used the thermal solar collectors.

The major contributions of this code, compared to existing ones, are to work across a wide spectral range, ranging from UV to infrared (280 nm – 30 nm), and to consider real material refractive indices (already included in a database). This makes SolPOC code particularly relevant for solar applications. Additionally, the code operates based on a relevant optimization algorithm for thin-film stacks, enabling it to identify and evaluate a variety of functional solutions. These solutions are highly applicable and sometimes counterintuitive compared to classical optical theory, thus ensuring significant innovation in optimized thin-film solutions.

Between 2018 and 2023, the code (previously under Scilab and named COPS) continued to be utilized and valued by the author and the PROMES – CNRS laboratory in France. Its effectiveness and "user-friendly" interface contributed to its success in local research teams. The code was directly referenced in several scientific publications and 2 book chapters, and it served as a valuable tool in different thesis conducted at the PROMES – CNRS laboratory [2–4]. Given the positive feedback on the usefulness of the code and a new research project initiated by the PROMES – CNRS laboratory, a decision was made in January 2023 to migrate the code to Python, introduce new functionalities, and release it as open-source software. This led to the current version of the code, named SolPOC (v0.9.7).

1.4. About the code

The current version of SolPOC offers the following features:

- Quicker and stable calculation of reflectivity, transmissivity, and absorptivity of thin layers stack using a vectorized (using NumPy package) Abélès formalism method [5].
- Working with a full solar spectral range including infrared (e.g. : 280 nm to 30 µm) [6]
- Use refractive index data of real materials found in peer-reviewed papers [7].

- Evaluate thin layers stack’s solar properties.
- Use Effective Medium Approximation methods (EMA) to model the optical behavior of material mixtures (dielectric mixtures, metal-dielectric, porous materials) [8].
- Optimize stack optical performances according to a large panel of cost functions, including cost functions for solar energy systems, building and solar thermal uses.
- Propose 6 different optimization methods based on evolutionary algorithms, such as PSO or Differential Evolution.
- Highly quality parallel code allows us to be working with multiprocessing.
- Automatically results output (.txt files and .PNG images) to a folder and propose a simplified user interface, bringing together useful variables in a few lines of code.

1.5. Dependency and installation

The actual version SolPOC (0.9.7) run under Python (version 3.11). All informations are presented on GitHub https://github.com/SolPOCandCo/SolPOC.

Concerning the installation, a full description is provided on GitHub.

1.6. Why use thin layer stacks in optics?

A thin film coating is a surface treatment widely used in various research and industrial sectors, including optic and solar energy applications. These treatments involve one or more thin layers of material (ranging from nanometers to micrometers) deposited on a substrate. The thin film stack modifies the near substrate’s surface and imparts specific and optimal properties for intended applications, such as optical properties, scratch resistance, deformation resistance, oxidation resistance, etc. while still benefiting from the mechanical properties of the substrate. Thin film coatings are particularly prevalent in the field of optics, including solar thermal and photovoltaic collectors. Practically, the overall performance of PV solar panels and thermal solar collectors heavily relies on the optical properties provided by thin film coatings (with thicknesses in nanometers or micrometers), rather than the bulk materials (with thicknesses in millimeters or centimeters).

In fact, just a few hundred nanometers of thin film materials deposited on the surface can drastically alter the optical behavior of a bulk substrate. Therefore, thin film coatings are often chosen for cost reduction and efficiency improvement. For instance, silver (Ag) is one of the most reflective metals. It's easy to understand the advantage of using thin film coatings: a few tens of nanometers of silver thin film deposited on a rigid substrate (glass as an example) will have the same optical as a solid silver mirror but at a much lower cost and with improved mechanical and ageing properties.

1.7. Examples of uses:

SolPOC can be used in all scientific domains where light is coming on one or several thin layer stacks deposited on a substrate. This software is very relevant for research, development, and education in solar energy, including solar thermal, photovoltaics or eyeglasses [9,10]. SolPOC can be used for several purposes, but not limited:

- advanced reflective coatings, using metallic and/or dielectric layers.
- antireflective coatings for human eye vision, PV cells or solar thermal applications.
- coatings for optical instruments, such as Bragg mirrors.
- radiative cooling coatings.
- low-e coatings and solar control glass for building applications.
- selective and absorber coatings for solar thermal applications.
- Spectral-splitting coating for PV-CST hybridization

See the paper and the Jupyter NoteBook for more details. We are assured that SolPOC will continue to be valuable asset to the solar community and can be readily adapted and applied to other communities in the future.

II. How SOLPOC evaluate optical properties?

Here we are describing the physics used in SolPOC. To illustrate usage through step-by-step instances, we have prepared Jupyter Notebooks accessible on GitHub. Additionally, pre-filled SolPOC launch files for various examples are present in the GitHub repository. Finally, we provide a description of the physics used in the code.

2.1. Energy Conservation Law in Optic

The optical properties derive from the law of energy conservation. The energy within an isolated system remains constant over time. This principle applies to radiation incidents on a material, which can either be reflected (*R*), absorbed (*A*), or transmitted (*T*) (Eq. 1). the radiation arrives at a solid angle defined by an angle of incidence θ and an azimuthal angle ψ, at a given wavelength λ. Also, the material is at a fixed temperature T_e. This leads to the following equation:

$$A(T_e,\theta,\psi,\lambda) + R(T_e,\theta,\psi,\lambda) + T(T_e,\theta,\psi,\lambda) = 1 \qquad \textit{Eq. 1}$$

The system stores energy by absorbing a part of the incident flux. If the body under study is in thermal equilibrium with its environment (constant temperature T_e), it necessarily redistributes the available energy to its surroundings. The emitted flux is called emissivity (represented as E) and is related to the absorbed flux by Kirchhoff's radiation law (Eq. 2).

$$A(T_e,\theta,\psi,\lambda) = E(T_e,\theta,\psi,\lambda) \qquad \textit{Eq. 2}$$

2.2. Complex refractive index

SolPOC system operates by employing complex refractive index to describe the optical behavior of materials. At present, it is not feasible to directly input material descriptions into SolPOC using other methods such as dielectric permittivity (often denoted $\varepsilon = \varepsilon_r + \boldsymbol{i}\varepsilon_i$, where ε_r

is the real part and *ε* ε_i is the imaginary part) or models like Drude, New-Amorphous, Brendel-Bormann, etc. Our approach involves utilizing complex refractive indexes to align with practices of our research community and to facilitate the direct application of following refractive index measurements via ellipsometry. Complex refractive indexes (*N*) are structured as two distinct elements: the real part n, commonly referred to as refractive index, and the imaginary part k, known as the extinction coefficient (Eq. 3). these two values are dimensionless and vary according to the wavelength λ.

$$N(\lambda) = n(\lambda) + \mathbf{i} \cdot k(\lambda) \qquad \textit{Eq. 3}$$

2.3. RTA: solve the Maxwell's equations.

The RTA function is the main function of SolPOC, it enables the simulation of the optical behavior of a stack, whether simple or highly complex, composed of one or multiple thin layers of materials deposited on substrate (Figure 2), using the complex refractive indexes *Nj* of the constituent materials and the thickness of the thin layers *dj*. The reflectivity (*R*) and the transmissivity (*T*) of a stack of thin film deposited on a substrate are obtained from the complex refractive indexes. If the most common formalism is based on the Transfer Matrix Method (TMM), other formalisms are available with their own advantages and inconvenient such as the Scattering Matrix, the Abélès formalism (with is different than TMM), the Admittance method or more recently an adaptation called the Direchlet-to-Neunmann maps. A complete deep-review and comparison of these different formalisms have been provided recently by D. Langevin et al. [11].

2.3.1. Abélès formalism

Based on their work, we have chosen the Abélès formalism for SolPOC as the best compromise between time and stability instead of TMM [12]. Moreover, similarly as A.Luce et al with TMM-Fast, the Abélès formalism allow us to use the package NumPy library witch strongly reduce the calculation time per CPU [13]. We have looked for the highest efficiency for the NumPy implementation by optimizing the code structure. The characteristic matrices M calculation are voluntary only 3D dimensions, to be of shape [2, 2·L, λ] where L the number of thin layers and λ the wavelengths for avoiding RAM use abuse. SolPOC is based on classical optical theory and solves Maxwell's equations using the Abélès formalism. For a reasonable number of layers (<150), this approach strikes a favorable balance between speed and stability [8]. Absorbance (A) is deduced through the law of conservation energy (A = 1 – R – T), while emissivity is calculated in accordance with Kirchhoff's radiation law $E(\lambda, T, \theta) = A(\lambda, T, \theta)$.

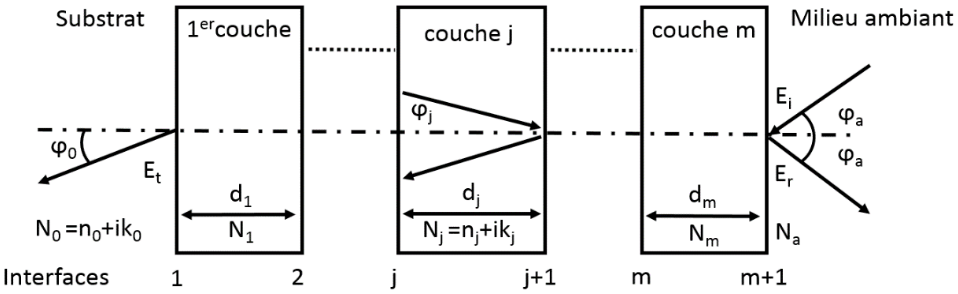


Figure 2. Multilayers stack on substrate

The specificity of the RTA function in the SolPOC code is its avoidance of repetitive calculations for each wavelength to generate spectral functions R(λ), T(λ), and/or A(λ). The complete spectrum is computed as a single operation, thanks to a specific coding structure, which means a saving in calculation time of around 500 times for a complete solar spectrum (280-2500 nm).

2.3.2. Detailed formalism

The Appendix 1 by Denis Langevin describes this formalism, which is used without restriction in all versions of SolPOC, including version 0.9.7.

2.3.3. Incidence Angle

SolPOC has the capability to consider the angle of incidence of radiation on the stack. The RTA function incorporates an optional parameter for the angle of incidence, initially set at 0° relative to normal. To adjust the value of the angle of incidence parameter (expressed in degrees and defined relative to the stack's normal), you can modify the value of the variable "Ang".

In SolPOC, the incidence angle is in degres

2.3.4. Benchmark of optical properties

To illustrate the accuracy of SolPOC compared to community-validated codes, we present a test case involving the reflectivity calculation of a 20-period Bragg mirror, an optical structure well-known for achieving reflectivity values close to 0 or 1 across a spectral range. Figure 3 demonstrates that the reflectivity spectra computed independently by SolPOC, TMM-Fast, Solcore and PyMoosh are visually indistinguishable, as evidenced by the perfect overlap of the four curves. We observe a maximum deviation of $1.6\cdot10^{-12}$ between SolPOC and PyMoosh (blue curve), reflecting the use of an identical formalism and corresponding in practice to the rounding of the last digit of a floating-point number. Both TMM-Fast and Solcore employ the transfer matrix method, which introduces slightly larger deviations. Specifically, the average relative error between SolPOC and TMM-Fast (green curve) is approximately 2.1·10−8 with even lower errors (below 10^{-11}) in regions of high reflectivity (around 700 nm), and higher errors (up to 10−6) in regions of very low reflectivity. The relative error between SolPOC and Solcore (red curve) is higher, in order of $3.6\cdot10^{-5}$ and can reach up to 10^{-2} at wavelengths with minimal reflectance. Such relative errors in the computation of very low reflectivity values (e.g., 10−5) at monochromatic wavelengths are still entirely acceptable.

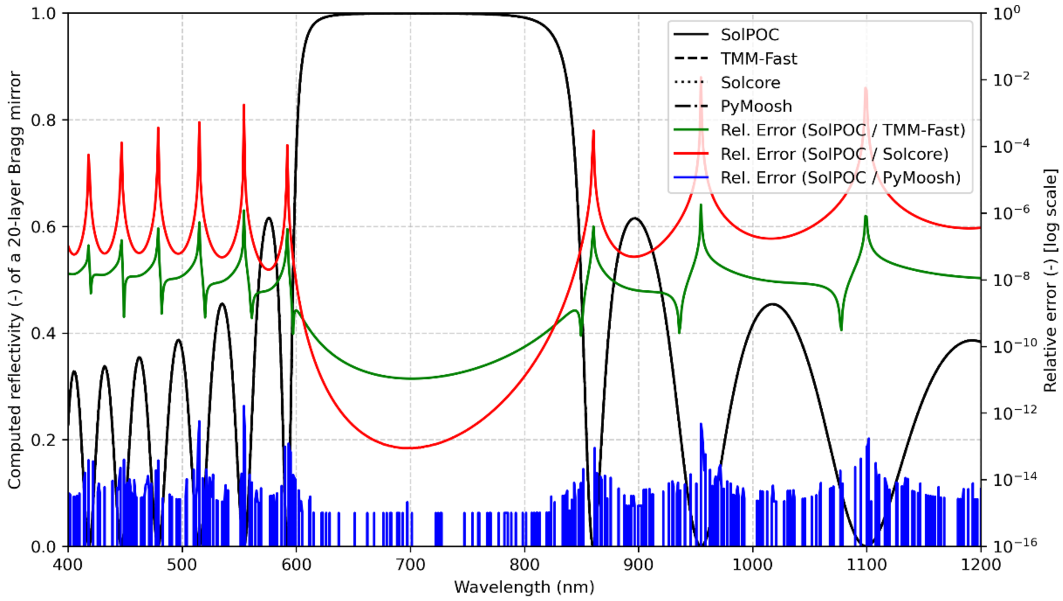


Figure 3 : Computed reflectivity of 20 layers Bragg mirror with different optical Python package

Figure 4 illustrates the absolute error in the average reflectivity of the same structure calculated over the 400–1200 nm spectral range. It demonstrates that computational accuracy is maintained across a wide range of stack sizes, from 2 to 50 thin-film layers. We observe results that are strictly identical between SolPOC and PyMoosh, up to numerical precision limits. The average absolute deviation is approximately 2·10-9 when compared to TMM-Fast, and 7.3·10^{-8} when compared to Solcore. In practical terms, considering the largest absolute erreor (SolPOC & PyMoosh vs. Solcore) an absolute error of 10^{-8} means that computed average reflectivity values agree at least up to the seventh decimal place, far exceeding the precision of current experimental characterization techniques. While not all optical structures could be compared due to obvious constraints in time, resources, and scope, we have strong confidence in the reliability of the results. Nevertheless, it is possible that unconventional layer arrangements may reveal numerical instabilities, which we would be happy to investigate and resolve.

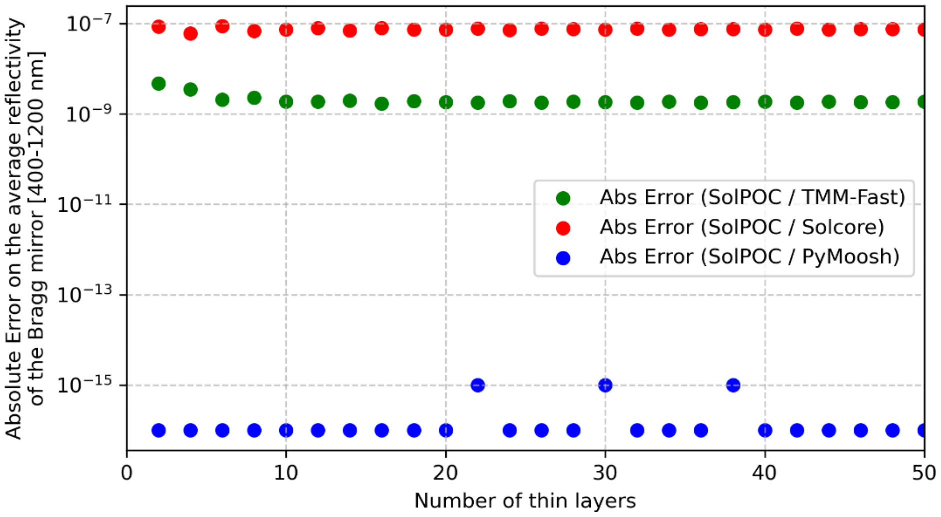


Figure 4 : Computed average reflectivity of Bragg mirror with different optical Python package, for 2 to 50 thin layers

2.3.5. Benchmark of time calculation

For many optimization processes, the runtime of the cost function is a critical factor. We therefore compared the execution speed of SolPOC for optical property calculations against three widely used optical packages—TMM-Fast, Solcore, and PyMoosh—using the test case described previously. Each package has specific features that should be highlighted when interpreting runtime comparisons. Solcore offers flexibility to treat incoherent layers, which inevitably impacts computational speed. Both PyMoosh and TMM-Fast compute optical properties for a single polarization at a time; thus, their functions must be executed twice to cover the use case relevant to solar energy applications, where both polarizations are required. In contrast, TMM-Fast provides additional capabilities, such as calculating the optical properties of multiple stacks, incidence angles, and wavelengths in a single function call. Moreover, it is the only package among the three to offer GPU acceleration through PyTorch, as well as an autograd functionality.

Figure 5 illustrates the benchmarking results obtained on a standard laptop without GPU acceleration, equipped with an Intel Core i7-1165G7 processor. This test therefore reflects the situation of a user without access to GPU computation, which is generally less favorable to TMM-Fast. We focus here on our main use case: computing the reflectance, transmittance, and absorptance for both polarizations (TE and TM) of a coherent multilayer stack over the full solar spectrum (280–2500 nm, 5 nm step size) at a given incidence angle. Such a calculation is necessary for the cost functions used in optimization routines that adjust layer thicknesses, material choices, and other stack parameters. Under this scenario, we observe that SolPOC (black curve) achieves the highest number of evaluations per second, followed by PyMoosh (blue curve, on average 7× slower), and then Solcore and TMM-Fast (green and red curves, respectively, both ~30× slower than SolPOC depending on the number of layers). TMM-Fast exhibits a more stable execution time, only weakly affected by the number of layers, which can be advantageous in certain contexts. Similarly, in applications requiring evaluation at multiple incidence angles, TMM-Fast is less affected, since SolPOC requires the calculation to be repeated n times. The curves with star markers represent performance when including 45 incidence angles, a practical discretization. In this case, SolPOC's speed (black curve with star) is reduced by a factor of 45, yet it remains on average about 3× faster than TMM-Fast under CPU-only execution (green curve, with star). We therefore conclude that SolPOC provides a clear runtime advantage for CPU users performing optical property calculations over two polarizations, across 450 wavelengths, and for 1–45 incidence angles. In other contexts, particularly when GPU acceleration is available, we encourage users to perform their own benchmarks.

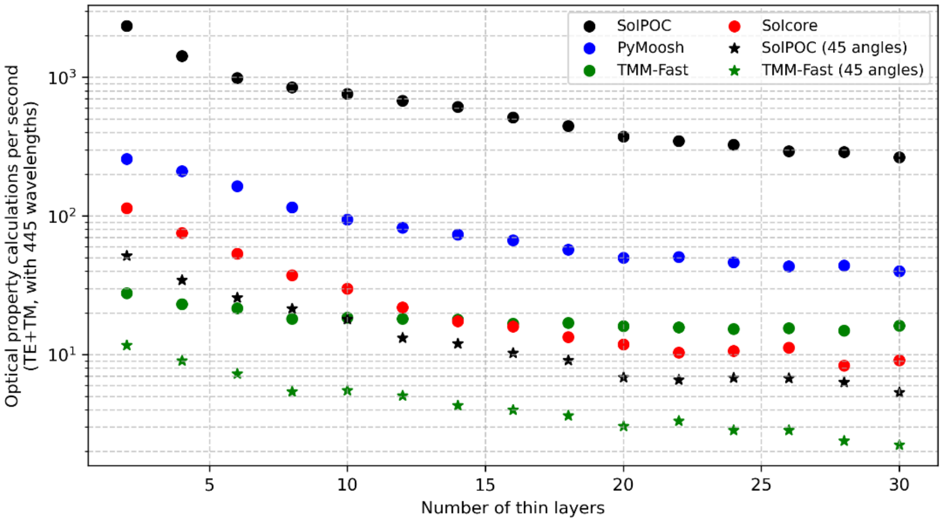


Figure 5 : Number of optical properties calculation per second on 445 different wavelengths on both polarizations (TE and TM), for different packages, using cpu

III. Global implementation

To understand how SolPOC works, we strongly recommend going through the provided Jupyter Notebooks. In addition, two example scripts can help you get started:

1. Basic_example_solpoc.py
2. Curve_RTA.py

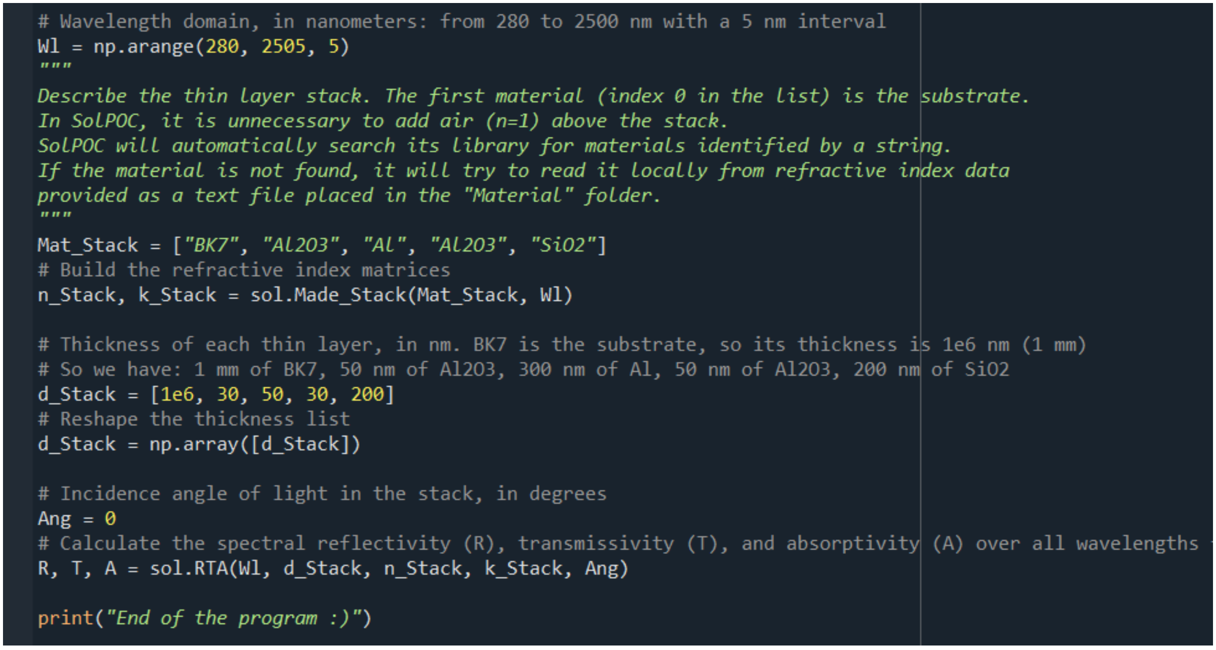
Below, we outline some fundamental concepts. The key idea is that SolPOC operates like a nested structure, where the core function is RTA. This function performs the main task: calculating the optical properties (Reflectivity, Transmissivity, and Absorptivity). All other parts of the code are essentially designed to prepare and supply the correct input parameters to this central function through various subroutines.

3.1. Cumput the reflectivity, transmissivity and absorptivity

In the example illustrated in Figure 6, the goal is to calculate the reflectivity, absorptivity, and transmissivity of a thin-film stack. To perform this calculation, the following inputs are required:

1. Wavelengths (Wl) defined using np.arange, expressed **in nanometers (nm).**

2. Refractive indices composed of a real part (*n_Stack*) and an imaginary part (*k_Stack*) (representing absorption).
3. Incidence angle (Ang), the angle of incoming light with respect to the stack, **in degrees**.
4. Layer thicknesses (d_Stack), a list of layer thickness values, also expressed in **nanometers (nm).**



```
# Wavelength domain, in nanometers: from 280 to 2500 nm with a 5 nm interval
Wl = np.arange(280, 2505, 5)
"""
Describe the thin layer stack. The first material (index 0 in the list) is the substrate.
In SolPOC, it is unnecessary to add air (n=1) above the stack.
SolPOC will automatically search its library for materials identified by a string.
If the material is not found, it will try to read it locally from refractive index data
provided as a text file placed in the "Material" folder.
"""
Mat_Stack = ["BK7", "Al2O3", "Al", "Al2O3", "SiO2"]
# Build the refractive index matrices
n_Stack, k_Stack = sol.Made_Stack(Mat_Stack, Wl)

# Thickness of each thin layer, in nm. BK7 is the substrate, so its thickness is 1e6 nm (1 mm)
# So we have: 1 mm of BK7, 50 nm of Al2O3, 300 nm of Al, 50 nm of Al2O3, 200 nm of SiO2
d_Stack = [1e6, 30, 50, 30, 200]
# Reshape the thickness list
d_Stack = np.array([d_Stack])

# Incidence angle of light in the stack, in degrees
Ang = 0
# Calculate the spectral reflectivity (R), transmissivity (T), and absorptivity (A) over all wavelengths
R, T, A = sol.RTA(Wl, d_Stack, n_Stack, k_Stack, Ang)

print("End of the program :)")
```

Figure 6 : script basic_example_solpoc

The refractive index (n) and extinction coefficient (k) of each material are obtained from the *SolPOC* material library or local data files. Finally, Reflectivity (R), Transmissivity (T), and Absorptivity (A) are computed for all wavelengths and the specified incidence angle.

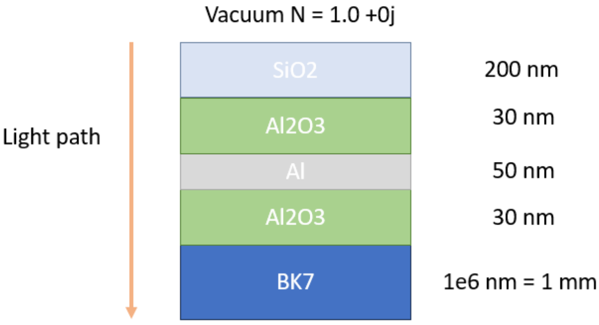


Figure 7 : thin layer stack expressed in Figure 6

Note 1 : The real part of the refractive index (n, stored in n_Stack) and the imaginary (complex) part (k, stored in k_Stack) are kept in two separate variables, whereas some codes combine them into a single complex variable. We chose this separation because it significantly speeds up subsequent RTA calculations.

In SolPOC **is not necessary** to include the ambient medium above the thin layer stack. It's automatically included by the code in the RTA function.

3.2. The thin layer : d_Stack

The *d_Stack* variable defines the thickness of each layer in the stack, expressed in nanometers (nm).

- ➢ The order of values in the list matches the order of materials in Mat_Stack, so that each thickness corresponds to the correct layer.

For convenience and computational efficiency, the list is converted into a NumPy array (np.array([d_Stack])) before further calculations, which speeds up operations when passed to SolPOC's functions.

3.3. Made_Stack

The *Made_Stack* function allows converting a sequence of material names, expressed as a list of strings representing the thin-film stack, into two matrices *n_Stack* and *k_Stack* that can be used almost directly in RTA. This function is responsible for:

1. Checking if a layer contains a composite material, i.e., a material name containing a dash (-), such as air-SiO_2 (porous materials) or W-Al_2O_3 (cermet).
2. Loading the corresponding material(s) using the open_material function.
3. Interpolating and extrapolating the optical constants over the wavelength domain using the interpolate_with_extrapolation function.
4. Repeating this process for each material in the stack.

The dimensions of *n_Stack* and *k_Stack* differ depending on whether the stack contains at least one composite layer:

1. If the stack contains no composite layers, n_Stack and k_Stack have shape of a 2-dimensional array : len(Wl), len(Mat_Stack).
2. If the stack contains at least one composite layer, n_Stack and k_Stack have shape of a 3-dimensional array (len(Wl), len(Mat_Stack), 2) to store the two materials in the composite layer.

Figure 8 present a schematic representation of how thin layer stacks (i.e a list of character string in Mat*_Stack*) produces different matrices *n_Stack* and *k_Stack*. With an analogously to a table:

a. columns correspond to the material of each thin-film layer,
b. rows correspond to wavelengths.
c. The optional third dimension corresponds to the presence of two materials per layer (i.e., a composite layer). They are the pseudo code of *Made_Stack* function is present in the appendix.

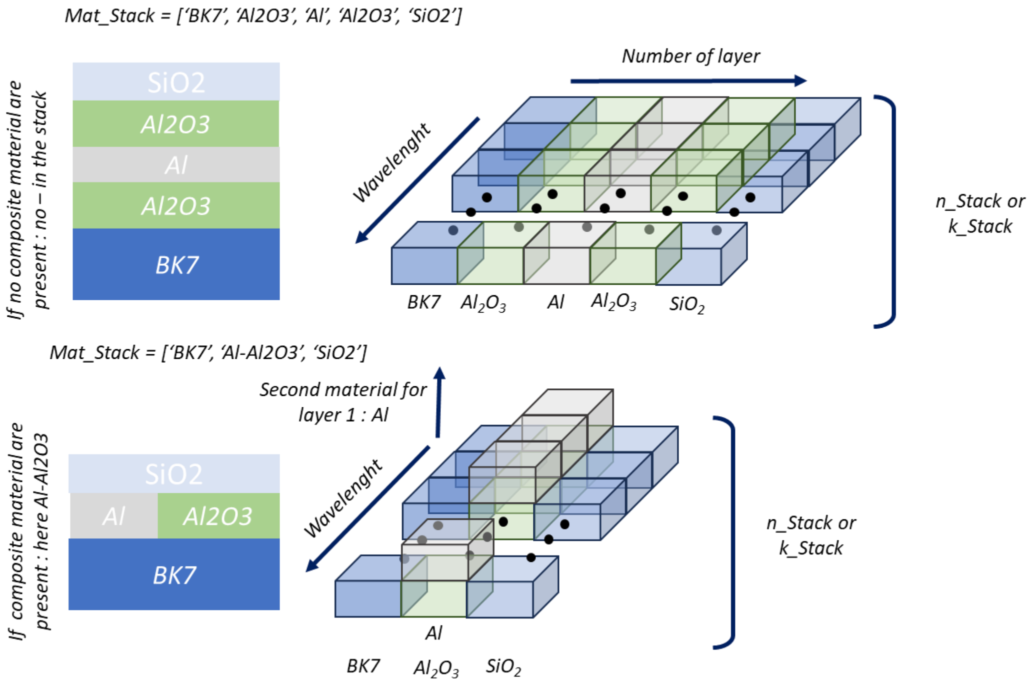

Mat_Stack = ['BK7', 'Al2O3', 'Al', 'Al2O3', 'SiO2']
If no composite material are present : no – in the stack
SiO2
Al2O3
Al
Al2O3
BK7
Number of layer
Wavelenght
n_Stack or k_Stack
BK7
Al_2O_3
Al
Al_2O_3
SiO_2
Mat_Stack = ['BK7', 'Al-Al2O3', 'SiO2']
If composite material are present : here Al-Al2O3
SiO2
Al
Al2O3
BK7
Second material for layer 1 : Al
Wavelenght
n_Stack or k_Stack
Al
BK7
Al_2O_3
SiO_2

Figure 8 : Schematic representation of n_Stack and k_Stack provide by Made_Stack function. The upper case has no composite material, and lower-case composite material (Al-Al2O3) in the stack

3.4. Made_Stack_vf

As explained earlier, the *Made_Stack* function converts a list of material names into two arrays, *n_Stack* and *k_Stack*, which contain the real and imaginary parts of the refractive index, respectively. However, when composite materials are present (resulting in 3-dimensional arrays), these matrices cannot be directly used by the RTA function, as it's requires *n_Stack* and *k_Stack* to be two-dimensional.

This transformation is performed by the *Made_Stack_vf* function, which applies an Effective Medium Approximation (EMA) to convert a layer composed of two materials into a single effective material. To do this, it requires a parameter called the volume fraction (vf), representing the proportion of each constituent material. The different types of EMA models and the role of the volume fraction are detailed in Section EMA: Effective Medium Approximation. Figure 9 give a schematic representation of the *Made_Stack_vf* functions.

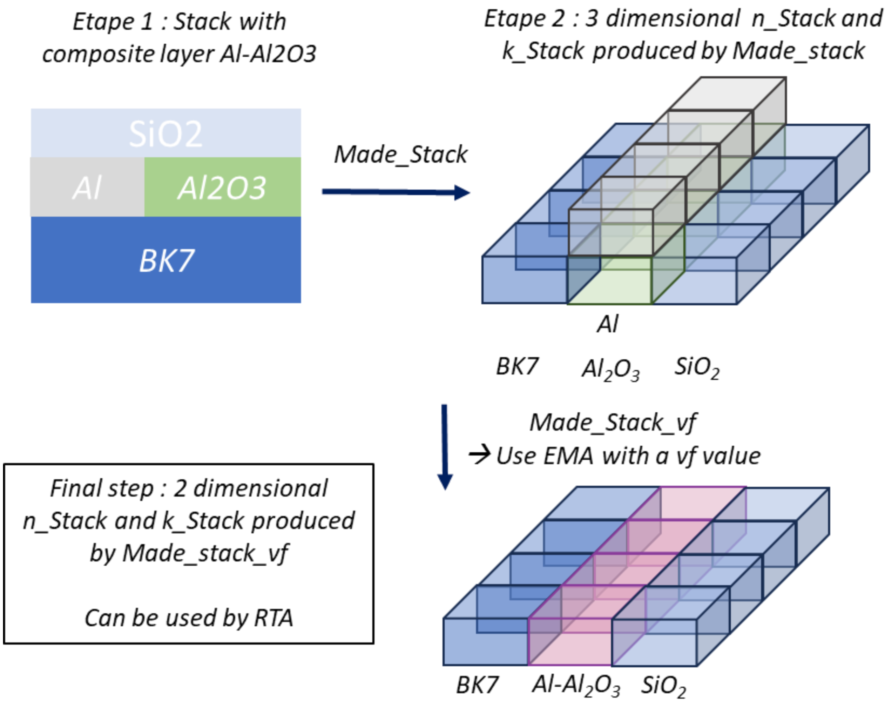

Etape 1 : Stack with composite layer Al-Al2O3
SiO2
Al
Al2O3
BK7
Made_Stack
Etape 2 : 3 dimensional n_Stack and k_Stack produced by Made_stack
Al
BK7
Al_2O_3
SiO_2
Made_Stack_vf
→ Use EMA with a vf value
Final step : 2 dimensional n_Stack and k_Stack produced by Made_stack_vf
Can be used by RTA
BK7
$Al\text{-}Al_2O_3$
SiO_2

Figure 9 : Schematic representation of n_Stack and k_Stack provide by Made_Stack function. The upper case has no composite material, and lower-case composite material (Al-Al2O3) in the stack

This final step is not handled by *Made_Stack*, because the need for homogenization may arise at different stages of the workflow. For example, *Made_Stack* is typically executed at the beginning when building the stack, while *Made_Stack_vf* may be used later, during the optimization process, if the goal is to optimize the value of vf.

3.4.1. EMA: Effective Medium Approximation

The complex refractive index of composite layers, such as cermet (W-Al_2O_3, mixture of dielectrics and metal) or porous materials (such as mixture of air and dielectric, like air-SiO_2) were estimated by applying an Effective Medium Approximation (EMA) method. These methods consider a macroscopically inhomogeneous medium where quantities such as the dielectric function vary in space and are often used in material sciences. Different EMA theories have been reported in the literature, such as Bruggeman and Maxwell-Garnett. The Bruggeman method is used in the code [8]. The Bruggeman theory was selected early for the creation of COPS, as already discussed in several papers [3,4]. Briefly, this theory makes no hypothesis of a major constituent is necessary and it allows simulating high volume fractions. At each wavelength, the complex dielectric function ε_{eff} of the materials mixture is deduced from the dielectric matrix ε_m and inclusions ε_i with a volume fraction of inclusions, noted *vf* in the code.

$$vf\,\frac{\varepsilon_i-\varepsilon_{eff}}{\varepsilon_i+2\varepsilon_{eff}}+(1-vf)\frac{\varepsilon_M-\varepsilon_{eff}}{\varepsilon_M+2\varepsilon_{eff}}=0 \qquad \text{Eq. 4}$$

3.4.2. Use composite material in a stack.

Using composite material in the code is straightforward. This is achieved by associating the name of two materials present in the database with a hyphen (symbol -) in a string of characters. For instance, writing "air-SiO2" in the stack definition instructs SolPOC to use an EMA to model the refractive index of a mixture of air and SiO_2 (a porous layer). Similarly, users may define "W-SiO2" for a cermet (metal–dielectric composite), or even combinations such as "SiO2-TiO2" or "W-Ag". In these cases, the optimizer adjusts both the thickness and the mixture ratio (the volume fraction, noted vf). In our view, this approach qualifies as a real material optimization, since it relies on the refractive indices of real, experimentally measured materials.

The list below shows an example of a stack of three composite thin films.

a. W-Al2O3 is a cermet layer : a mixture of metallic inclusion into a dielectric matrix. In "W-Al2O3", the inclusions are "W" and the matrix is "Al2O3".
b. ZnO-TiO2' is a mixture of two dielectric layers, which produce a refractive index range between the two materials as shown in [14] [1] or [15].
c. air-SiO2' is a porous material, which produce a refractive index range between air (n=1.0) and the material used (here SiO2). Others laws exist for modeled the refractive index of porous mixture, such as Maxwell-Garnet or Yoldas formula, and Bruggemann model give similar results [16].

Mat_Stack = ['Fe ', 'W-Al2O3', 'ZnO-TiO2', 'air-SiO2']

3.4.3. EMA function and time calculation

In the current version of the code the Brugmann function is vectorized using numpy, and do not slow the code considerably. Since SolPOC v0.9.6 optimizing with composite materials as not a significative impact of time-consuming.

3.5. RTA_curve

The figure summarizes the process used to compute the optical properties through several key steps:

1. Step 1 – Input definition: The description of the materials in the stack (Mat_Stack), their thicknesses (d_Stack), and, if applicable, their volume fractions (vf) are required.All these quantities are organized as lists, ensuring that each index corresponds to the same layer in the stack.
2. Step 2 – Optical data generation: The function Made_Stack creates the variables n_Stack and k_Stack by retrieving the refractive index data either from online databases or from the user's local library. It automatically handles composite materials using a dash (-) notation (see EMA: Effective Medium Approximation).

[1] Update : the article is retracted due to use of the same figure in several paper. However, the fact that a dielectric mixture can produce refractive index range between the two materials is still demonstrated. We actually looking for proper reference in remplacement

3. Step 3 – Effective medium calculation : When necessary, the function Made_Stack_vf applies EMA models to generate n_Stack and k_Stack as two-dimensional variables, incorporating the volume fraction information.
4. Step 4 – Optical properties computation: Finally, the RTA function computes the reflectivity, transmissivity, and absorptivity of the entire thin-film stack.

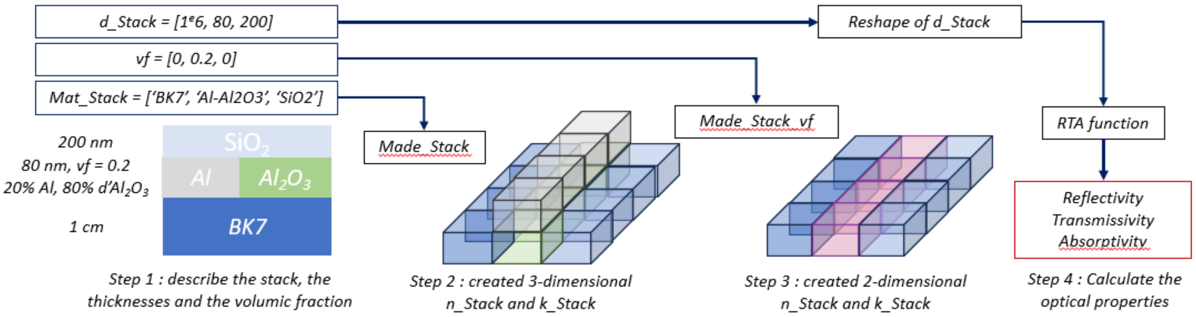

d_Stack = [1e6, 80, 200]
vf = [0, 0.2, 0]
Mat_Stack = ['BK7', 'Al-Al2O3', 'SiO2']
Reshape of d_Stack
Made_Stack
Made_Stack_vf
RTA function
200 nm
80 nm, vf = 0.2
20% Al, 80% d'Al_2O_3
1 cm
SiO_2
Al
Al_2O_3
BK7
Reflectivity
Transmissivity
Absorptivity
Step 1 : describe the stack, the thicknesses and the volumic fraction
Step 2 : created 3-dimensional n_Stack and k_Stack
Step 3 : created 2-dimensional n_Stack and k_Stack
Step 4 : Calculate the optical properties

Figure 10 : schematic representation of different step between a stack description and the calculation of optical properties

The sequence of steps described above closely corresponds to the process implemented in the RTA_curve function. The only "difference" lies in the use of the Individual_to_Stack function, which is described earlier. In short, this function reconstructs the variables d_Stack and vf from a single list. Indeed, during the optimization process, the two lists — d_Stack and vf — are merged into a single list (forming what we refer to as an individual), as this unified representation is required for the optimization algorithm.

3.6. Incoherent layer: RTA_curve_inco

An enhanced version of the *RTA_curve* function is available and is, for example, used in the script Curve_RTA.py. This function is very similar to *RTA_curve*, but it additionally allows the treatment of incoherent thin-film layers, expanding the range of physical configurations that can be simulated.

3.6.1. About incoherent layer:

In thin-film optics, layers are classified as coherent or incoherent depending on how the optical phase of light is preserved during propagation within each layer.

- ➢ A coherent layer is one whose physical thickness is comparable to or smaller than the coherence length of the light source, so that interference effects between multiple reflections inside the layer are maintained. This is typically the case for thin dielectric coatings such as anti-reflection or interference filters, where constructive and destructive interference determine the optical response.
- ➢ In contrast, an incoherent layer is much thicker than the coherence length, meaning that the phase relationship between reflected and transmitted waves is lost due to random phase variations. As a result, interference effects average out, and such layers are treated using intensity-based (rather than amplitude-based) calculations, often by combining reflectance and transmittance incoherently.

In multilayer simulations, coherent and incoherent layers can coexist, requiring hybrid models that handle interference only where it is physically meaningful.

Figure 11 illustrates the optical properties of a BK7 glass layer (which can be approximated as silica with a refractive index n = 1.5) depending on whether the layer is treated as coherent or incoherent. In this example, the glass layer has a thickness of 1 µm.

1. Left panel – Coherent layer without a second interface: The layer is coherent, but there is no interface behind it. The reflection is about 0.05, corresponding to a single refractive index change between air (n = 1.0) and glass (n = 1.5).
2. Middle panel – Coherent layer with two interfaces (thin glass plate): This case represents what one would observe when looking through a 1 µm-thick glass plate. Since the thickness of the glass layer is comparable to the incident wavelength (280 – 2500 nm), interference patterns appear due to multiple reflections within the layer.
3. Right panel – Incoherent glass layer: Here, the glass layer is considered incoherent, meaning that no interference occurs. The resulting reflection is roughly 0.1, which is approximately twice the single-interface reflection (0.05). This value arises from the two refractive index transitions: first from air (n = 1.0) to glass (n = 1.5), and then from glass back to air as the radiation exits the layer.

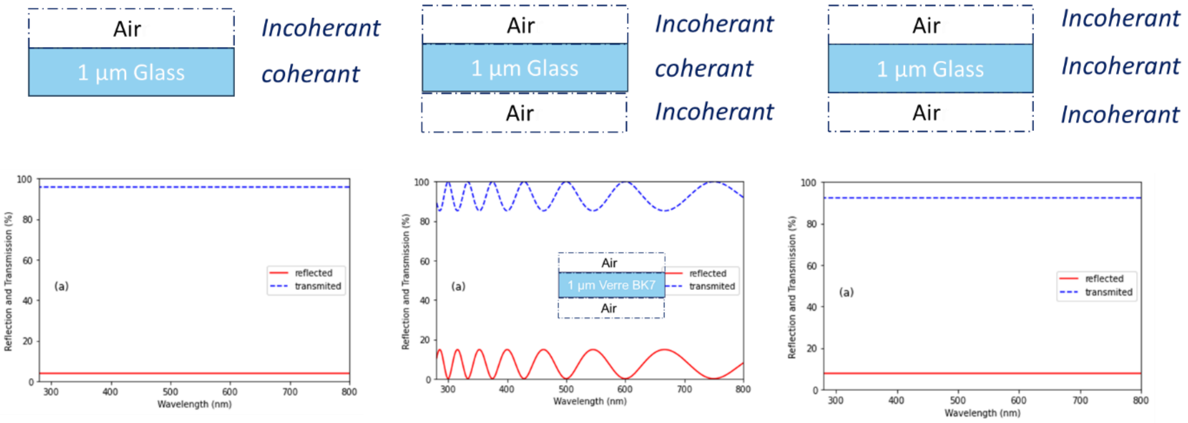


Figure 11 : Example of coherent or incoherent thin layer of BK7

3.6.2. Optical properties with incoherent layer: SolCore package

In the current version of SolPOC (v0.9.7), the RTA function does not yet support the treatment of incoherent layers — similarly to other packages such as PyMoosh or TMM-fast. To perform such calculations, we rely on SolCORE, which, to our knowledge, is the only Python package capable of handling incoherent thin-film layers.

In summary:

1. If the stack contains only coherent layers:

➔ Use the RTA function from the SolPOC package.

2. If the stack contains at least one incoherent layer:

- ➔ Use the calculate_rta function from SolPOC. The Made_SolCORE_Stack function serves as the bridge between the SolPOC formalism and SolCORE.

It is therefore necessary to define what is meant by coherent and incoherent layers.

By default, SolPOC uses a threshold value called coherency_limit, set to 2000 nm

This value can be modified by the user through the coherency_limit variable within the parameters dictionary. The function Stack_coherency uses this threshold to determine whether the stack should be treated as coherent or incoherent.

3.7. Individual: stack description

In the context of the code, an "individual" refers to a potential stack of thin layers during an optimization process. An individual describes the thin layers stack and includes at least the description of thicknesses of each thin layer. It can also include the thin layers composition for composite or theorical layers. An individual results from an optimization function. Although similar, it should not be confused (as it might differ) with the list of thin layer thicknesses *d_Stack* used for the RTA function (see paragraph RTA: solve the Maxwell's equations, p13).

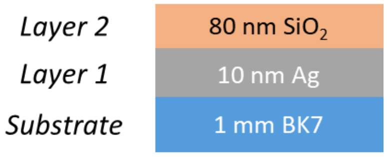
1 individual = 1 stack description = 1 result according to a cost function = 1 probable solution to the problem

The *Individual_to_Stack* function acts as a bridge between the optimization parameters (stored in an "individual") and the optical model used by SolPOC. It transforms an individual, that is, a list of numerical parameters such as layer thicknesses, volume fractions or refractive index into separate variables (*d_Stack*, *vf*, *n_Stack*, *k_Stack*) ready for further computation.

3.7.1. Individual with only thicknesses

In the case of a stack with no theoretical thin layer (paragraph Individual with theoretical material, p27) and no composite layer (Individual with composite material, p26). This means that each individual is an array with a length equal to the number of thin layers, including the substrate. The number of thin layers and the material is described in *Mat_Stack*. Every value in the array describes a thin-film thickness in nanometers. In this context, an individual is identical to *d_Stack*. An example is presented in Figure 12 for a stack of two thin layers (Ag et SiO_2) deposited on a glass substrate (BK7), forming a BK7/Ag/SiO_2 stack.

Mat_Stack = ['BK7', 'Ag', 'SiO2']



Individual: output of optimization function
Example of individual : array([1000000, 10, 80])

Thickness of thin layers stack : input of RTA function
d_Stack : array([1000000, 10, 80])

Figure 12 : Thin layers stack description, with Individual et d_Stack

The individual array consists of thickness values (in nanometer) starting from the substrate towards the outer layers. Table 1 is an example that illustrates the correspondence between thicknesses and the stack materials described in Figure 12.

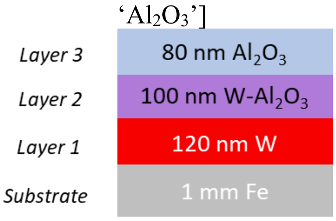
Table 1 : description of stack described in Figure 12.

Index of Individual	0	1	2
Description	Thickness of Substrate	Thickness of Layer 1	Thickness of Layer 2
Material	BK7	Ag	SiO_2
Value	1 000 000	10	80

3.7.2. *Individual with composite material*

In the case of a thin layer stack that includes at least composite layers (Individual with composite material, p26), each stack's description requires the thickness of each layer and volumetric fractions. Each individual is still represented by an array containing thicknesses, but added with the volume fractions, which described the composition of the composite layer (in fact the percentage of inclusion in the host matrix). A volume fraction is number between 0 and 1. In this context, an individual is not identical to *d_Stack*. Here's an example in Figure 13 for a stack of three thin layers (W, W-Al_2O_3 et Al_2O_3), including a composite thin layer, here W-Al_2O_3. All layers are deposited on an iron substrate (Fe), forming Fe/W/W-Al_2O_3/Al_2O_3 stack.

Mat_Stack = [‘Fe, ‘W’, ‘W-Al_2O_3’, ‘Al_2O_3’]



Individual: output of optimization function
Example of individual: array ([1000000, 120, 100, 80, 0, 0.25, 0])

Thickness of thin layers stack: input of RTA function
d_Stack : array ([1000000, 120, 100, 80])

Figure 13 : Stack description including composite layer(s), with Individual and d_Stack

The array *individual* consists of thickness values (still in nanometer) starting from the substrate towards the outer layers, followed by the volume fractions (*VF*, number between 0 and 1) of each layer which is also optimized. Since the layer n°1 (the tungsten layer, W) and the layer n°2 (Al_2O_3) are not composite layers, the volumetric fraction (VF) value is by default 0. Even if another value is present as vf for Layer 1 and 3, it's has no impact. Table 2 is an example that illustrates the correspondence between refractive indexes, thicknesses, and the materials of the thin layers as well as the substrate.

Table 2 : description of stack described in Figure 13

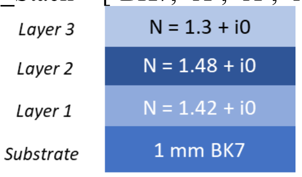
Index	0	1	2	3	4	5	6
Description	Thickness of Substrate	Thickness of Layer 1	Thickness of Layer 2	Thickness of Layer 3	VF of Layer 1	VF of Layer 2	VF of Layer 3
Material	Fe	W	W-Al_2O_3	Al_2O_3	W	W-Al_2O_3	Al_2O_3
Value	1 000 000	120	100	80	0	0.25	0

As a reminder, a composite thin layer is composed of two distinct materials, separated by a hyphen (symbol “- “). The first material is the inclusion and the second the matrix host (see paragraph about the Effective Medium Approximation). For example, porous layer (mixture of air inclusion in with a dielectric matrix, like SiO_2) can be noted “air-SiO2" or a mixture of two dielectric can be noted “SiO2-Al2O3”.

3.7.3. Individual with theoretical material

This case involves an *individual* derived from optimization with theoretical materials. The introduction of theoretical material is defined by the variable *nb_layer*, which adds theoretical thin layer above the stack determined in *Mat_Stack*. A theoretical thin layer is one for where both the real part of the refractive index (assumed to be constant across different wavelengths) and the thickness are optimized concurrently.

Mat_Stack = [‘BK7, ‘X’, ‘X’, ‘X’]



Individual: output of optimization function
Individual : array ([1000000, 47, 38, 120, 1.42, 1.48, 1.3])

Thickness of thin layers stack: input of RTA function
d_Stack : array ([1000000, 47, 38, 120])

Figure 14 : Stack description including theorical layer(s), with Individual and d_Stack

The individual array consists of thickness values starting from the substrate towards the outer layers, followed by the real part of the refractive index for each thin layer, which is also optimized. Table 3 is an example that illustrates the correspondence between refractive indexes, thicknesses and the materials of the thin layers as well as the substrate.

Table 3 : description of stack described in Figure 14

Index	0	1	2	3	4	5	6
Description	Thickness of Substrate	Thickness of Layer 1	Thickness of Layer 2	Thickness of Layer 3	*N* of Layer 1	*N* of Layer 2	*N* of Layer 3
Material	Fe	-	-	-	-	-	-
Value	1 000 000	47	38	120	1.42	1.48	1.30

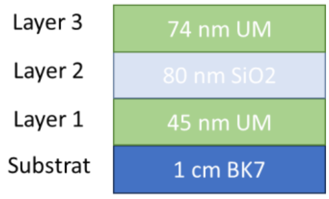
3.7.4. Individuals with unknown material

The latest version of SolPOC led us to develop a new module that optimizes both the thickness of a stack and the choice of materials from a predefined list of three options. This corresponds to cases where one or more layers in the stack have unknown materials (referred to as *Unknown Material*, or UM). This type of optimization is discussed in Section Optimizing a stack with a list . and implemented in the script *template_optimization_with_materials.py*. This development required us to update the structure used to define an *individual* in the optimization process. The list of parameters to be optimized now contains two types of elements:

1. Layer thickness, expressed in nanometers
2. Continuous numerical values ranging from –1 to 1, which represent the material choice.

These continuous values allow the optimization algorithm to automatically explore all possible material combinations. A dedicated function, *choose_material*, converts each value in the range [–1, 1] into a specific material from the predefined list *Mat_Option*.

Mat_Stack = [‘BK7, ‘UN’, ‘SiO2’, ‘UM’]
Mat_Option = [‘ZnO’, ‘TiO2’]



Individual: output of optimization function
Individual : array ([1000000, 45, 80, 74, -0.98, 0.12, 0.95])

Thickness of thin layers stack: input of RTA function
d_Stack : array ([1000000, 45, 80, 74])

Figure 15 : Stack description including layers where the materials is Unknow (UM)

Table 4 : description of stack described in Figure 14

Index	0	1	2	3	4	5	6
Description	Thickness of Substrate	Thickness of Layer 1	Thickness of Layer 2	Thickness of Layer 3	X of Layer 1	X of Layer 2	X of Layer 3
Material	BK7	Unknow	SiO2	Unknow	-	-	-
Value	1 000 000	45	80	74	-0.98	0.12	0.95

3.7.5. Individual with more than one

In the current version of SolPOC (v0.9.7), a stack **cannot include** simultaneously

1. theoretical thin layer and composite layer.
2. theoretical thin layer and unknow material
3. composite layer and and unknow material

This limitation arises directly from the current structure of the code, which we are fully aware of and are considering addressing in future versions. On the other hand, we have never encountered a case requiring such a large number of variations within the stack.

3.7.6. Stack_to_Individual Flowchart

3.8. Materials refractive index database

One advantage of SolPOC is to provide the user with a large database of refractive indices for all types of materials, particularly those suitable for solar energy applications (including the most used metals, ceramics or oxide material). This database is derived from a critical review of the scientific literature (e.g., refractiveindex.info database) and technical catalogs (e.g., technical catalog from the glass industry), which allowed preselecting the most relevant data (e.g., measurements on thin films rather than bulk materials, measurements rather than modeling/simulation/extrapolation, numerous measurement points to minimize reliance on interpolation/extrapolation by the code, etc.), corresponding to a broad spectral range directly compatible with thermal solar applications. These studies have been selected because they cover a large spectral domain, from the solar range 280 – 4000 nm, and often the IR range necessary for radiative losses calculation. They also present a good accuracy in the solar range, needed for a good estimation of solar performance. Also, the data was measured on actual thin film samples fabricated by deposition techniques similar to that used in CSP industries.

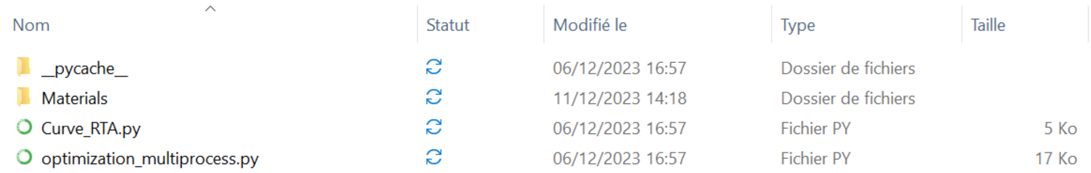
3.8.1. How to use the material database?

The materials refractive index database of SolPOC “came” with the installation of the SolPOC package. Indeed, during the installation of the package, a local copy of the materials database is created. Each materials file is a text files named according to the materials, with the refractive index data (*n* and *k* in the wavelength, *Wl*).

During the utilization SolPOC process in that order for have the refractive index data of the materials present in the thin stack:

1. SolPOC try to read the refractive index data present in the installation package. Original text files can be found in GitHub in solpoc/Materials.
2. If the materials chosen in the stack is not available in the database SolPOC try to read it in the text files in a local folder named “Materials”. Figure 6 provide an example of the folder created using the !solpoc-init command. Place your personal materials in the empty folder named “Materials”

Note that once installed, you don't need an Internet connection. The database is present locally.



Nom	Statut	Modifié le	Type	Taille
__pycache__	⟳	06/12/2023 16:57	Dossier de fichiers	
Materials	⟳	11/12/2023 14:18	Dossier de fichiers	
Curve_RTA.py	⟳	06/12/2023 16:57	Fichier PY	5 Ko
optimization_multiprocess.py	⟳	06/12/2023 16:57	Fichier PY	17 Ko

Figure 16 : Example of folder ProjectSolPOC created using the !solpoc-init command. Place your personal materials in the empty folder “Materials”

3.8.2. Interpolation

The complex refractive indexes of materials need to be interpolated (with a similar step to the wavelength) and extrapolated by the code to the chosen wavelengths. This ensures that vectors and tables have the same dimensions. In the code a linear interpolation method, while simple, has been favored, as it gives the best results. Other interpolation methods, such as cosine or polynomial, could potentially introduce outliers (example. Negative *k*-values between two zero *k*-values). The code includes several test procedure functions to assess the accuracy of interpolation and notify the user in case of substantial errors.

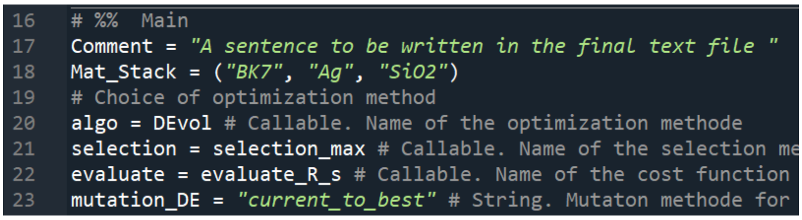
3.8.3. Add new material.

Adding new refractive indexes ($N = n + \mathbf{i}k$) for new materials to the database is a straightforward process. Simply add a text file (.txt) to the Material folder, respecting the data formatting. A readme.txt file is also present in the Material folder, in addition to the numerous examples already included.

1. The text file must be named in accordance with the material’s name. Indeed, the character string you use in the stack description (variable *Mat_Stack*) in SolPOC is

employed to open the corresponding text file. It's crucial to ensure a precise correspondence between the character string and the filename.

Example: The names of the materials are described by the variable *Mat_Stack*. The files that SolPOC will attempt to open in the Materials folder will be "*BK7.txt*," "*Ag.txt*," and "*SiO2.txt*".



```
# %%  Main
Comment = "A sentence to be written in the final text file "
Mat_Stack = ("BK7", "Ag", "SiO2")
# Choice of optimization method
algo = DEvol # Callable. Name of the optimization methode
selection = selection_max # Callable. Name of the selection me
evaluate = evaluate_R_s # Callable. Name of the cost function
mutation_DE = "current_to_best" # String. Mutaton methode for
```

Figure 17 : Illustration of the script

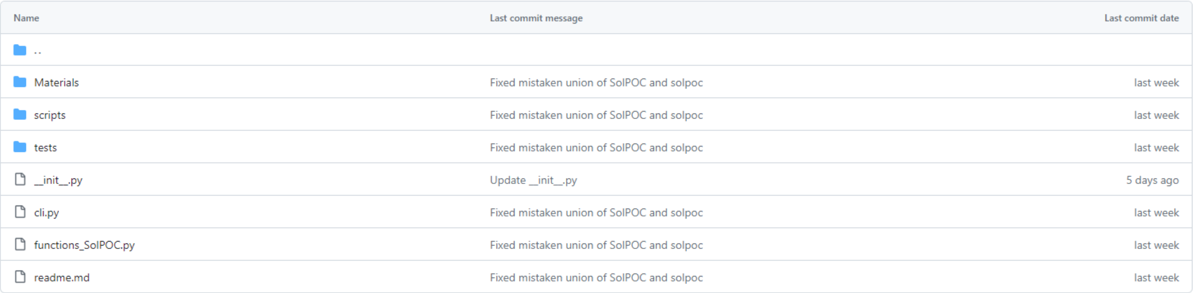
2. The text file contains only 3 columns. The index 0 column corresponds to the wavelengths (in nm), the index 1 column is the real part of the refractive index (n), and the index 2 column is the complex part (k). If the complex part is strictly zero, it is necessary to write the value 0.

We have some good practice reminders:

- Material names should not include a hyphen (symbol: -) or number in indexes or numbers in subscript or superscript (write *SiO2.txt* and no *SiO₂.txt*)
- We've used the underscore or underline symbol to bring more nuance to material names, for example with author names.
- For non-English-speaking users, the decimal separator in the files should be a dot, not a comma.

3.8.4. Consult the database.

The full database is available in folder "solpoc/Materials" on GitHub, and come from of the RefractiveInde.info website [7]. The website shares refractive index of materials in peer-reviewed papers. You can consult the current materials and their data simply by browsing the Materials folder. If necessary, you can make a local copy.



Name	Last commit message	Last commit date
..		
Materials	Fixed mistaken union of SolPOC and solpoc	last week
scripts	Fixed mistaken union of SolPOC and solpoc	last week
tests	Fixed mistaken union of SolPOC and solpoc	last week
__init__.py	Update __init__.py	5 days ago
cli.py	Fixed mistaken union of SolPOC and solpoc	last week
functions_SolPOC.py	Fixed mistaken union of SolPOC and solpoc	last week
readme.md	Fixed mistaken union of SolPOC and solpoc	last week

Figure 18 : The Material folder, available on GitHub, contain all the database of materials.

This database can be participative. You can add materials yourself by contacting us or by making a push from GitHub. Likewise, feel free to contact us in the event of an error or request for clarification on the database.

The refractive index database is also available as Zenodo Archive : Grosjean, A. (2025). SolPOC_Python_Package_v0.9.7_Materials_database (0.9.7) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.17208300 [17]

3.8.5. List of materials

Here is the list of main materials present in the package :

- Ag.txt
- Ag_Babar.txt
- Ag_Hagemann.txt
- Ag_McPeak.txt
- Ag_Rakic.txt
- Ag_RBB.txt
- Ag_RLD.txt
- Ag_W.txt
- air.txt
- Al.txt
- Al2O3.txt
- AlN.txt
- Al_Hagemann.txt
- Al_McPeak.txt
- Al_Orda.txt
- Al_Rakic.txt
- Al_RBB.txt
- Al_RLD.txt
- Au.txt
- Au_RLD.txt
- BaF51.txt
- BaK1.txt
- BaK2.txt
- BaK4.txt
- BaK4N.txt
- BalF4.txt
- BaLF4N.txt
- BaSF2.txt
- BK10.txt
- BK7.txt
- CaF2.txt
- Ch_Rakic.txt
- Co.txt
- Cr.txt
- Cr2O3.txt
- Cr_RLD.txt
- Cu.txt
- F2.txt
- F5.txt
- Fe.txt
- FK51A.txt
- GaAs.txt
- ITO.txt
- K10.txt
- K10N.txt
- K7.txt
- K7N.txt
- KF9.txt
- KF9N.txt
- LaF2.txt
- LaF35.txt
- LaK14.txt
- LaK21.txt
- LaK34.txt
- LaK7.txt
- LaSF.txt
- LaSF9N.txt
- LF5.txt
- LLF1.txt
- MgF2.txt
- MgO.txt
- Mo.txt
- Ni.txt
- Ni_Rakic.txt
- PC.txt
- PK52A.txt
- PSK3.txt
- PSK3N.txt
- PSK53A.txt
- PSK53AN.txt
- Pt.txt
- Ra.txt
- RbF.txt
- SF2.txt
- SF2N.txt
- SF6.txt
- SF6N.txt
- Si.txt
- Si3N4.txt
- SiC.txt
- SiO.txt
- SiO2.txt
- SK2.txt
- SK2N.txt
- SSK2.txt
- SSK2N.txt
- SSK5.txt
- SSK8.txt
- Ta.txt
- Ta2O3.txt
- Ti.txt
- TiN.txt
- TiO2.txt
- V.txt
- vaccum.txt
- W.txt

- WO3.txt
- Zn.txt
- ZnO.txt
- Zr.txt
- ZrO2.txt

For some materials, several data sets are available. This is the case for gold (Au) and silver (Ag), for example. To differentiate between data for different materials or different measurements, we personally add the underscore symbol "_", followed by the information.

By Ag_Rakic, we mean silver refractive index data proposed by Rakic et al., while Ag_McPeak means silver refractive index data proposed by McPeak [18,19].

We don't have all the scientific publications corresponding to each material. Most of the refractive index data in the SolPOC package comes from the refractive index website [7]. Please refer to this site (or other) to obtain the publications and references corresponding to the materials present. **Finally, if you're using SolPOC for a publication, we strongly advise you to search and include by yourself for the refractive index that correspond to your studying case.** The refractive indices of material vary according to their purity, deposition conditions, thin-film thickness, temperature... Refractive index data may therefore be relevant in one case and erroneous in another.

3.9. Theorical Materials

In the code, it's possible to optimize a stack (always according to a cost function) without using one or more real materials. In this case, we use the term theoretical materials. In COPS, a theoretical material is defined by:

1- A real part of the refractive index, constant over wavelengths
2- A complex part of refractive index equal to 0.

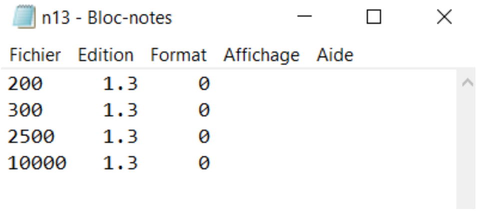
We then find the following formalism, which brings our theoretical materials closer to theoretical dielectric materials:

$$N_{th} = n + i0 \, with \, \frac{dn}{d\lambda} = 0 \qquad \textit{Eq. 5}$$

The use of theoretical materials allows us to simplify the problem, by searching for the best dielectric-type materials to deposit in order to obtain the best cost function.

3.9.1. Use a Theorical Material without optimize the refractive index

When writing stack materials, it's possible to refer to text files in the *Materials* folder that meet the conditions set out above. Some examples are already present, and creating new ones is very easy. These are the text files named "*n13.txt*, *n17.txt*, *n23.txt*", as example. Each file contains a few lines describing the material's refractive index at the following wavelengths. The Figure 19 gives an example, where the file "*n13.txt*" describes a material with a refractive index equal to 1.3 at 200, 300, 2500 and 10000 nm with k=0. The few values presented are sufficient, thanks to interpolation (see Interpolation paragraph, p. 14).



```
200     1.3     0
300     1.3     0
2500    1.3     0
10000   1.3     0
```

Figure 19 : Example of theoretical material present in the Materials folder.

3.9.2. Theorical Material for thicknesses and refractive index optimization .

An example can be found in Tutorial 3 : Optimize Stack Thicknesses With Theoretical Material where we look for the best refractive indices to design a 3-layer anti-reflective lens for the human eye.

3.10. Summary of optical properties calculations

Now, we can conduct an initial synthesis. *Figure* 20 illustrates the process by which the code derives the optical properties (reflectivity, transmissivity, and absorptivity) of a stack of thin films. Initially, the code utilizes complex refractive indices, either acquired from a database or directly calculated in the case of theoretical thin films. In the presence of composite materials, such as cermet or porous materials, the code employs an EMA method through the Bruggeman model to assess the refractive index of the thin film. At the end of this step, the code constructs a 1D stack comprising 0 to 150 thin films deposited on a substrate. Subsequently, the code applies the Abélès formalism to acquire the optical properties. The M matrices are structured as [2, 2·L, λ] to expedite calculations, utilizing the NumPy package (with *L* the thin layers numbers). This capability enables the code to compute optical properties across the entire solar spectrum in a matter of milliseconds.

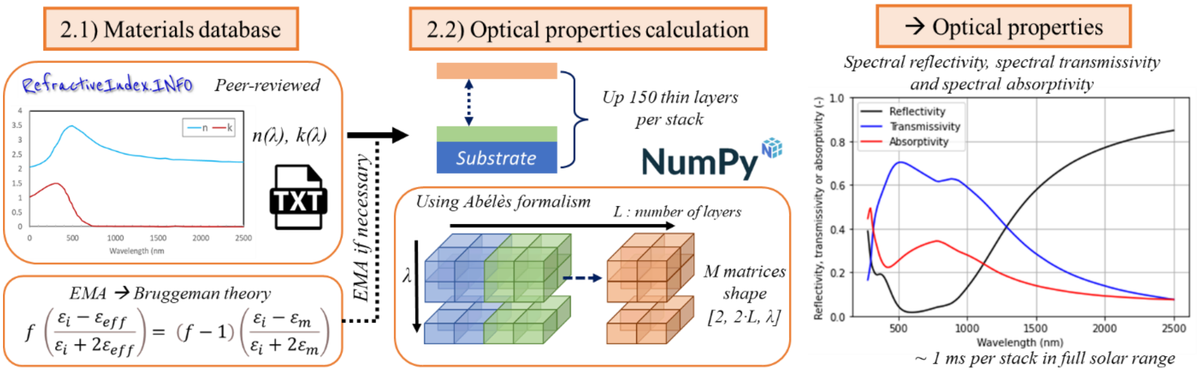

2.1) Materials database
RefractiveIndex.INFO
Peer-reviewed
n(λ), k(λ)
TXT
EMA → Bruggeman theory
$f\left(\frac{\varepsilon_i - \varepsilon_{eff}}{\varepsilon_i + 2\varepsilon_{eff}}\right) = (f-1)\left(\frac{\varepsilon_i - \varepsilon_m}{\varepsilon_i + 2\varepsilon_m}\right)$
EMA if necessary
2.2) Optical properties calculation
Up 150 thin layers per stack
Substrate
NumPy
Using Abélès formalism
L : number of layers
λ
M matrices shape [2, 2·L, λ]
→ Optical properties
Spectral reflectivity, spectral transmissivity and spectral absorptivity
Reflectivity
Transmissivity
Absorptivity
Reflectivity, transmissivity or absorptivity (-)
Wavelength (nm)
~ 1 ms per stack in full solar range

Figure 20 : SolPOC use refractive index from peer reviewed studies, added, if necessary, with EMA theory for created a thin layer stack. The optical properties are calculated using a Abélès formalism, using NumPy package for reduce time calculation.

3.11. Parameters and get_parameters

Many functions in SolPOC require a wide set of variables to perform their calculations. Instead of passing a long list of arguments to each function call, some of which are only used by internal subfunctions, SolPOC adopts a clearer and more structured approach: the parameters dictionary.

The parameters dictionary is a centralized container that groups together all the physical and numerical variables required for the optical simulation of a thin-film stack in SolPOC. It consolidates all the data describing a case study : material properties, stack geometry, solar spectrum, and simulation conditions into a single, coherent structure.

This dictionary can then be passed as a single argument to many functions in the module (such as *RTA_curve_inco*, *evaluate_RTA_s*, etc.), allowing them to directly access the necessary information. This approach improves code readability, reduces the risk of errors when passing variables, and simplifies the reuse or modification of simulation parameters.

3.11.1. Get_parameters

SolPOC provides a dedicated initialization function for the dictionary: *get_parameters*. This function automatically builds a complete dictionary containing all the data required for simulation and, when applicable, for optimization. It also includes a “self-filling mechanism”: if certain variables are missing or not specified by the user, the function automatically assigns appropriate default values to ensure that all necessary parameters are defined.

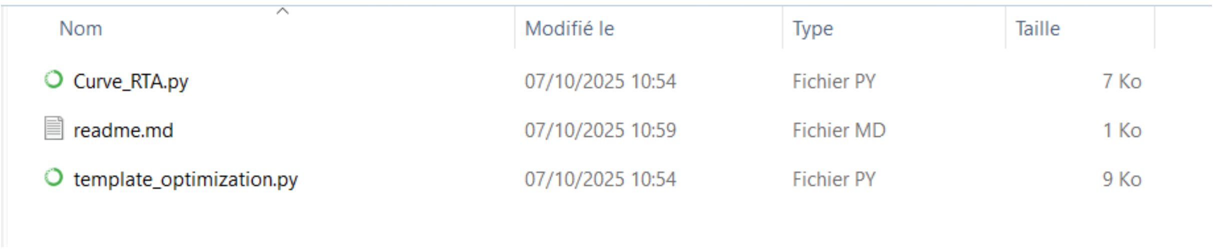
This system guarantees that every SolPOC function has access to a consistent and complete set of physical, optical, and algorithmic variables, while greatly simplifying the process of passing arguments within the code.

The Appendix 2 lists all variables included in the dictionary along with their descriptions and default values.

IV. Description of template for Optimization

The advantage of SolPOC is that the information required for optimization is grouped together at the beginning of the script. The main Python script for optimization is called “template_multiprocess”.

If you have use the !solpoc-init command, those files are automatically present in the folder Projet_SolPOC once it has been created



Nom	Modifié le	Type	Taille
Curve_RTA.py	07/10/2025 10:54	Fichier PY	7 Ko
readme.md	07/10/2025 10:59	Fichier MD	1 Ko
template_optimization.py	07/10/2025 10:54	Fichier PY	9 Ko

Figure 21 : the main script “template_multiprocess” is automatically present in the Projet_SolPOC folder

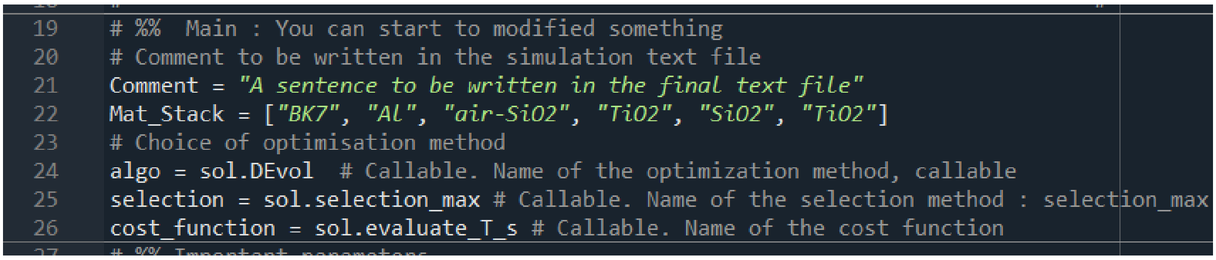
The variables are already prewritten, so all you need to do is change their values. Once the values have been modified, the code executes in a single stroke, without any further action required by the user. Execution includes:

- Declaration of variables, functions and import modules.
- Optimizing the problem several times, taking advantage of multiprocessing
- Processing and formatting results
- Saving the main results as images and text files.

The main code can be divided into 4 different parts, represented by cells marked with the symbol #%%. Here's a description of the different code cells.

4.1. Cell #1: Describe the Stack and the Optimization Method

The cell #1 is shown in Figure 22. This cell contains several callable. The principle of a callable is described in What is a callable? page 40.



```
# %%  Main : You can start to modified something
# Comment to be written in the simulation text file
Comment = "A sentence to be written in the final text file"
Mat_Stack = ["BK7", "Al", "air-SiO2", "TiO2", "SiO2", "TiO2"]
# Choice of optimisation method
algo = sol.DEvol  # Callable. Name of the optimization method, callable
selection = sol.selection_max # Callable. Name of the selection method : selection_max
cost_function = sol.evaluate_T_s # Callable. Name of the cost function
```

Figure 22 : Stack description and choose the optimization method.

- *Comment*: it is a character string used by the user to describe the problem, the context and the aim of this optimization. This sentence will then be written to a file containing the main information.
- *Mat_Stack*: This is a list of character strings. Each character string corresponds to the name of a text file in the *Materials* folder, which contains the refraction indices.
- *algo* is callable. This is the name of the function defined in SolPOC package which is used for optimization.
- selection is callable. This is the name of the function defined in SolPOC package, which is used to select desirable results, for example by minimizing or maximizing the cost function.
- Cost_function is callable. This is the name of the function defined in SolPOC package which calculates the cost function, i.e. the optical performance of a stack.

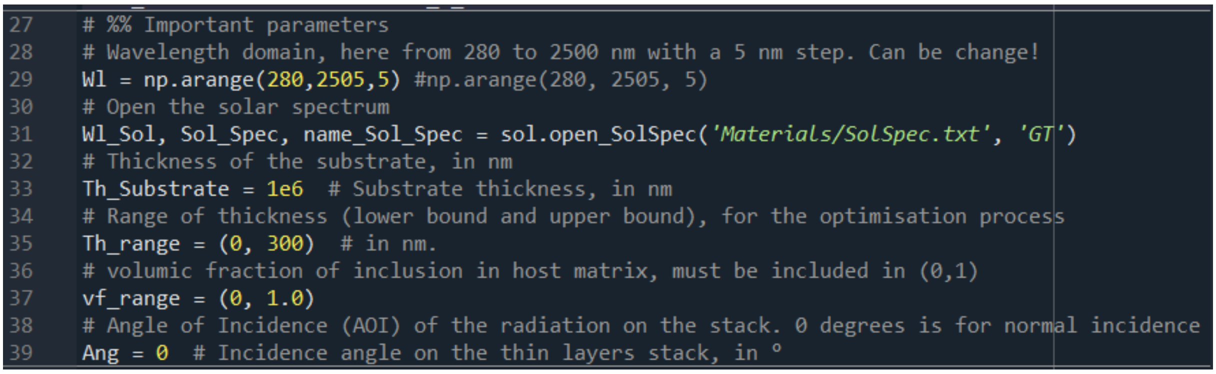
4.2. Cell #2 : Important Parameters

Cell #2 is depicted in Figure 23. This cell encompasses the primary parameters essential for SolPOC code. These include:

- *Wl*: Describing the spectral range in nm, it is an array declared using the NumPy library. In the given example, np.arange(320, 2505, 5) specifies a starting wavelength of 320 nm, an ending wavelength of 2505 nm (exclusive), with a step size of 5 nm.

Consequently, an array (similar to a list) with 437 elements is generated: [320, 325, …, 2495, 2500].

- *Th_Substrat*: Representing the thickness of the substrate in nm.
- *Th_range*: Indicating the range of admissible thicknesses for thin film layers in nm. It defines the lower and upper bounds for the algorithm's exploration. For instance, *Th_range* = (0, 200) implies that the thicknesses will be optimized between 0 and 200 nm.
- *vf_range*: Representing the range of admissible volumetric fraction of the refractive index for optimization with composite thin films (cermet, porous materials, see paragraph EMA: Effective Medium Approximation). It defines the lower and upper bounds. For instance, vf_range = (0, 1.0) indicates that the volumetric fraction used un Bruggman model will be optimized between 0 and 100%.
- *Ang*: Denoting the value of the angle of incidence relative to the normal of the irradiation on the stack, expressed in degrees. A value of 0 implies that the irradiation is perpendicular to the stack.



```
# %% Important parameters
# Wavelength domain, here from 280 to 2500 nm with a 5 nm step. Can be change!
Wl = np.arange(280,2505,5) #np.arange(280, 2505, 5)
# Open the solar spectrum
Wl_Sol, Sol_Spec, name_Sol_Spec = sol.open_SolSpec('Materials/SolSpec.txt', 'GT')
# Thickness of the substrate, in nm
Th_Substrate = 1e6  # Substrate thickness, in nm
# Range of thickness (lower bound and upper bound), for the optimisation process
Th_range = (0, 300)  # in nm.
# volumic fraction of inclusion in host matrix, must be included in (0,1)
vf_range = (0, 1.0)
# Angle of Incidence (AOI) of the radiation on the stack. 0 degrees is for normal incidence
Ang = 0  # Incidence angle on the thin layers stack, in °
```

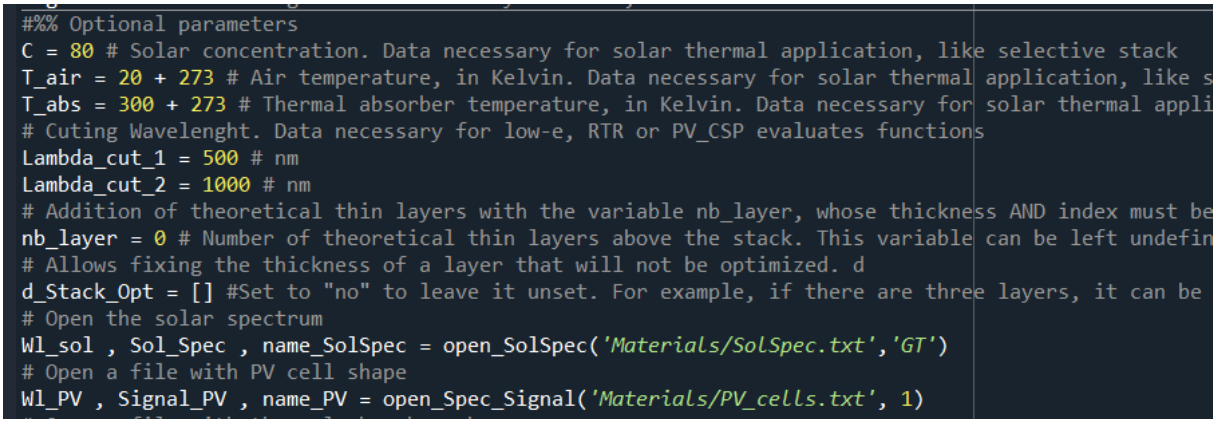
Figure 23 : Important parameters for SolPOC

4.3. Cell #3: Other important Parameters

Cell #2 is depicted in Figure 23, containing the parameters necessary for various cost functions or advanced SolPOC functions. For a description of a cost function, see paragraph Cost functions: the callable , p 45. The other parameters can be :

- *C*: Represents the solar concentration rate, crucial for solar thermal absorber calculations.
- *T_air*: Denotes the temperature of the air surrounding the thin-film stack, a requisite for solar thermal absorber calculations.
- *T_abs*: Represents the temperature of the absorber, specifically the stack of thin films and the substrate. This parameter is essential when calculating solar thermal absorbers. Note: Although material optical properties can change with temperature, this aspect is not considered here. The stack temperature does not directly influence the refractive indices of thin films.
- *Lambda_cut_*1: Signifies a cut-off wavelength, measured in nm. This value is useful in certain cost functions, notably in the cost fucntion *evaluate_low_e*.

- *Lambda_cut_2*: Represents a second cut-off wavelength, also in nm, with the condition Lambda_cut_2 > *Lambda_cut_1*. This value proves beneficial in certain cost functions often concerning spectral splitting, such as *evaluate_RTR* or *evaluate_TRT*
- *n_range*: Signifying the range of admissible real part of the refractive index for theoretical thin film optimization (see paragraph Theorical Materials). It sets the lower and upper bounds. For example, n_range = (1.3, 3.0) means that the real part of the index will be optimized between 1.3 and 3.0 (MgF_2 to AlAs as example).
- *Nb_layer*: An optional variable that may not be defined (deleted or commented out). Nb_layer represents the theoretical number of thin layers deposited on top of the stack. Refer to paragraph Theorical Materials for details.
- *d_Stack_Opt*: Another optional variable that may not be defined (deleted or commented out). d_Stack_Opt is a list of strings and numbers used to set one or more thicknesses. See paragraph Tutorial 4 : Optimize Stack Thicknesses with a Thickness Fixed for further clarification.
- *Wl_sol*, *Sol_Spec*, and *name_SolSpec*: This line initiates a solar spectrum using the open_SolSpec function. Wl_sol then contains the lengths of the solar spectrum in nm, SolSpec represents its irradiance in W/m^2nm^{-1}, and name_SolSpec is a string denoting the name of the solar spectrum.
- *Wl_PV*, *Sol_PV*, and *name_PV*: This line opens a signal, a spectrum function in 0 and 1 wavelengths. This signal is applied to the solar spectrum BEFORE being applied to the stack. This allows, for instance, consideration of the selectivity of a PV cell. *Wl* contains the lengths of the solar spectrum in nm, Signal is a spectrum function.



```
#%% Optional parameters
C = 80 # Solar concentration. Data necessary for solar thermal application, like selective stack
T_air = 20 + 273 # Air temperature, in Kelvin. Data necessary for solar thermal application, like s
T_abs = 300 + 273 # Thermal absorber temperature, in Kelvin. Data necessary for solar thermal appli
# Cuting Wavelenght. Data necessary for low-e, RTR or PV_CSP evaluates functions
Lambda_cut_1 = 500 # nm
Lambda_cut_2 = 1000 # nm
# Addition of theoretical thin layers with the variable nb_layer, whose thickness AND index must be
nb_layer = 0 # Number of theoretical thin layers above the stack. This variable can be left undefin
# Allows fixing the thickness of a layer that will not be optimized. d
d_Stack_Opt = [] #Set to "no" to leave it unset. For example, if there are three layers, it can be
# Open the solar spectrum
Wl_sol , Sol_Spec , name_SolSpec = open_SolSpec('Materials/SolSpec.txt','GT')
# Open a file with PV cell shape
Wl_PV , Signal_PV , name_PV = open_Spec_Signal('Materials/PV_cells.txt', 1)
```

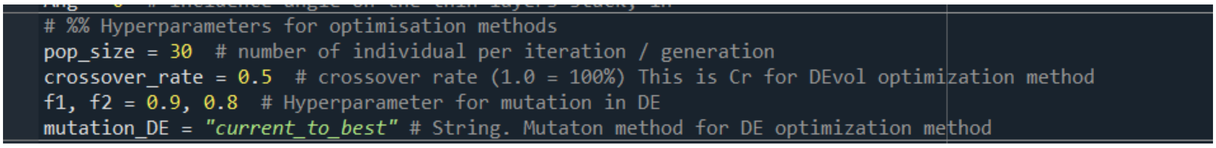
Figure 24 : Example of other optional parameters for SolPOC

4.4. Hyperparameters for optimization methodes

Cell #3 contains the hyperparameters required for the optimization methods. A hyperparameter is a quantity that governs how an optimization algorithm operates. The parameters presented in Figure 25 correspond to the DEvol (Differential Evolution) method. They include:

- *Pop_size*: The population size, i.e., the number of candidate agents or “parents.”
- *Crossover_rate* = 0.5 (ranging from 0 to 1): The crossover probability, which controls the mixing of parent solutions.

- *F1 and F2*: Hyperparameters influencing the creation of the mutant vector in the Differential Evolution process, also known as the differential weights.
- *Mutation_DE*: The mutation strategy applied in the DE algorithm.



```
# %% Hyperparameters for optimisation methods
pop_size = 30  # number of individual per iteration / generation
crossover_rate = 0.5  # crossover rate (1.0 = 100%) This is Cr for DEvol optimization method
f1, f2 = 0.9, 0.8  # Hyperparameter for mutation in DE
mutation_DE = "current_to_best" # String. Mutaton method for DE optimization method
```

Figure 25 : Example of other optional parameters for SolPOC

These hyperparameters play a crucial role in the algorithm's performance and convergence, and their adjustment can have a significant impact on the results obtained (see Optimizations , p 39 for more information). Table 4 presents typical values and indicates where to modify them. There are two main types of locations:

1. Directly from the SolPOC launch script. The most common hyperparameters are found in cell no. 4. This avoids the need to modify values directly in functions. This method is available for *DEvol*, *Optimize_ga*, and *Strangle*.
2. For less frequently used methods such as *(1+1)-ES*, *PSO*, and *simulated annealing*, the hyperparameters are written directly into the functions.

Table 5 : Hyperparameters and optimization methods

Name of the algorithm function	Where are the hyperparameters?	Typical value for the hyperparameter

DEvol	In the code main files	pop_size = 30
		mutation_rate = 0.5
		f1, f2 = 0.9, 0.8
		mutation_DE = "current_to_best"

(1+1)-ES	In the function, present in the package	initial_step_size = 10
		step_size_factor = 0.99

Optimize_ga	In the code main files	pop_size = 30
		crossover_rate = 0.5
		evaluate_rate = 0.3
		mutation_rate = 0.5
		mutation_delta = 15
		precision_AlgoG = 1e-5
		nb_generation = 50

Optimize_Strangle	In the code main files	pop_size = 30
		evaluate_rate = 0.3
		precision_AlgoG = 1e-5
		nb_generation = 50

PSO	In the function present in the package	inertia_weight = 0.8
		cognitive_weight = 1.5
		social_weight = 1.5

Simulated_annealing	In the function present in the package	initial_temperature = 4500
		cooling_rate = 0.95

4.4.1. About the seed

The optimization routines use non–gradient-based algorithms, such as evolutionary or stochastic methods. These algorithms inherently rely on a random number generator, meaning that two runs of the same script can produce slightly different results. If the optimization process is stable and well-tuned, these variations should remain minimal — but they still exist.

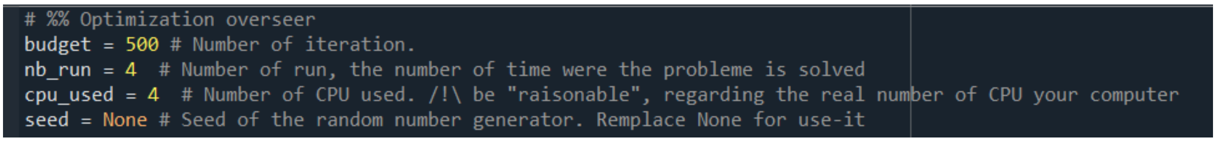
- ➢ To ensure full reproducibility of results, it is necessary to fix the seed (i.e., the initial value of the random number generator) at the start of the optimization.

In each template script, you can specify an integer value in the following line: seed = None # Seed of the random number generator. Replace None with an integer to fix it. Replacing None with a specific integer (e.g., seed = 12345) guarantees that the random sequence, and therefore the optimization results, are reproducible. The remaining variables are hyperparameters crucial for the algorithms to operate. Hyperparameters are external parameters to the optimization algorithms, not inherently learned by the algorithm itself, and must be set by the user before initiating the optimization process.

4.5. Cell #4 : Optimization overseer

Cell #4, shown in Figure 27, defines the global context of an optimization process. Those parameters concern all optimization methods.

- *Budget*: Specifies the number of iterations allowed for each optimization method.
- *nb_run*: Defines the number of independent optimizations runs. For example, if nb_run = 10, the optimization problem is solved independently ten times in succession.
- *cpu_used*: Indicates the number of logical cores used for parallel execution (see Multiprocessing, p. 38).
- *Seed*: Sets the seed for the random number generator. Replace None to activate reproducible behavior.



```
# %% Optimization overseer
budget = 500 # Number of iteration.
nb_run = 4  # Number of run, the number of time were the probleme is solved
cpu_used = 4  # Number of CPU used. /!\ be "raisonable", regarding the real number of CPU your computer
seed = None # Seed of the random number generator. Remplace None for use-it
```

Figure 26 : Optimization oversee

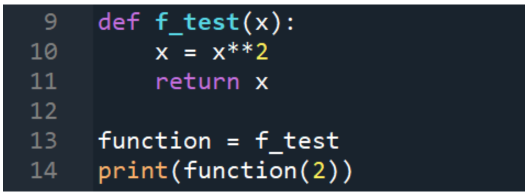
V. Run an Optimization

To launch an optimization with SolPOC, all you need to do is run the main SolPOC script. The whole script is already written and contains all the different variables to describe a problem, solve it and save the relevant results in an automatically created folder.

5.1. Optimizations method

5.1.1. What is a callable?

In the code, we assign callable to the variables *algo*, *selection* and *evaluate*. Here, a callable is the name of a previously defined function. An example is shown in the image below.



```python
def f_test(x):
    x = x**2
    return x

function = f_test
print(function(2))
```

Figure 27 : Example of callable

Using the function variable as a function is an example of a callable. In Python, a callable is an object that can be called a function. This includes user-defined functions, built-in functions, class methods and so on. In this example, the variable function is assigned to the function *f_test*. Consequently, function becomes a reference to this function and can be called using parentheses as if it were a function. Calling function (2) executes function *f_test* with argument 2, squaring 2 and returning 4.

The callable usable in *algo* corresponds to the various thin-film stack optimization algorithms. Each optimization algorithm has 2 callable variables as inputs:

1. *evaluate*, which are also callable. They correspond to the name of the cost function and are called "algo" in the function.
2. *selection*, which corresponds to the search for the minimum or maximum of the cost function. Depending on the optimization function, selection may be callable.
3. The last object is a dictionary containing all the necessary information. It is used to easily transmit relevant information to the program's various functions.

The aim of the algorithm is to provide an optimized solution (i.e. a stack of thin layers) to the problem, according to the cost function written in the evaluate callable. The solution is chosen (minimized or maximized, for example) thanks to callable selection. All three callable work as described in Figure 28, which also presents inputs and outputs.

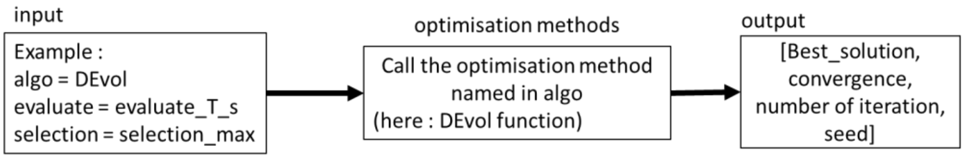


Figure 28: Input and output of the optimization methods

A total of 6 different optimization algorithms are currently available in SolPOC. For a good description of the various algorithms and their usefulness for thin-film stack optimization, we recommend reading Bennet's Ph.D. and his work [20,21]. These algorithms are well known in the scientific community and many descriptions, examples and tutorials can be found on the Internet.

5.1.2. Dictionary: parameters

In the code, a dictionary named "parameters” is being used to pass parameters between different functions. It avoids giving an important number of parameters for each function, especially for the cost function. As an example, the parameters present in the container are the wavelength vector, the stack refractive index (*Mat_Stack*, *n_Stack*, *k_Stack*) etc. During the program, most functions just read the different parameters present in the dictionary.

5.2. Definitions

Optimization domain: A structure is represented by a set of parameters (the materials and their thickness). The authorized values of those parameters, set by the users, form the optimization domain. Typically, larger is the optimization domain, more difficult is the optimization process.

Cost function: Each structure is associated with a cost function representing the ideal optical property we want. In our results, upper the cost function value is, better is the performance of the structure.

Local and global optima: Photonics cost functions are complex and presents many local optima, rendering the search for the global optimum (the point of the optimization domain where the value is the upper) very difficult. Global optimization algorithms, in particular the evolutionary or genetic ones (which are one of the most popular global optimization methods), explore the optimization domain by generating several initial structures, called the initial population, in the optimization domain. Then, the population is evolving by creating new structures and keeping those who are better (with a upper cost function) than those in the actual population. On the contrary, local optimization methods, often based on gradient descent, are very efficient to find local minima but is detrimental to the global optimum search.

5.3. Which one use?

The various algorithms can be employed to optimize the thickness of a stack of thin layers. However, SolPOC incorporates two additional functions:

1. optimizing thickness and the real part of the refractive index (refer to Theorical Materials, p32)
2. optimizing thickness and volume fraction (refer to EMA: Effective Medium Approximation, p21)

These two functionalities are not universally present in all algorithms due to a lack of demand. Table 6 outlines the various optimization algorithms and their functionalities. The absence of functionality is merely due to a lack of necessity and time constraints, with DEvol and optimize providing complete satisfaction. It is feasible to implement the various functionalities in all the algorithms.

Name of algorithm	Thickness	Thickness with theoretical material	Thickness with volumic fraction
DEvol	X	X	X
(1+1)-ES	X		
optimize_ga	X	X	X
optimisze_Strangle	X		
PSO	X		
Simulated_annealing	X		

Table 6 : Optimization algorithm and their functionalities

If you have no idea which algorithm to use, we recommend DEvol, with the hyperparameters proposed in Table 5. These values come from the Ph.D. work of P.Bennet [20]. If DEvol doesn't give good results, it's often necessary to increase the budget, which here means increasing the value of the *nb_generation* variable.

5.3.1. DEvol : Different Evolution from P.Bennet et al [20].

The typical Differential Evolution (DE) is a population-based optimization algorithm designed for global optimization in continuous search spaces. In the most used algorithm in the code. In DE, the individuals are like vectors. We propose a short description of the algorithm:

1. Initialize a population of random solutions (vectors) in the search space.
2. Generate trial vectors by combining and perturbing selected individuals using differential mutation and crossover technique.
3. Evaluate the fitness of the trial vectors with a cost function.
4. Replace individuals in the population with their respective trial vectors if they are superior.

DE iteratively evolves a population of solutions, encouraging exploration and exploitation of the search space by repeating step 2-4. By applying differential mutation and selection mechanisms, the algorithm efficiently navigates towards optimal or near-optimal solutions in complex, multi-dimensional spaces.

In the code, we use a specific variant of DE algorithm named here « DEvol », which was been developed by A. Moreau and P. Bennet for numerical optimization of photonic structures. In essence, their research has demonstrated that DEvol effectively addresses thin film stacking optimization problems, as we do in SolPOC The current implementation of DEvol is also integrated into PyMoosh[2], a numerical code available in GitHub [5]. To gain a better understanding of DEvol, we highly recommend referring to their published works [20,22].

5.3.2. One_plus_One_ES

The (1+1)-Evolution Strategy (named One_plus_One_ES in the code) is a simple optimization algorithm used to find local optima in continuous search spaces. The (1+1)-ES is a type of evolutionary strategy that explores the solution space by gradually adapting the step

[2] DEvol is named « *differential_evolution* » in PyMoosh

size based on the success of generating better solutions. It is a simple but effective optimization method for local search problems.

5.3.3. Optimize_ga

Genetic Algorithm (GA) is a powerful optimization technique inspired by the process of natural selection. GA iteratively evolves a population of solutions over generations, with fitter individuals having a higher chance of contributing to the next generation. This process mimics the principles of natural evolution, leading the algorithm towards better solutions in complex search spaces. There are many different versions of genetic algorithms, as shown by a concise description of the algorithm:

1. Initialize a population of potential solutions (chromosomes) randomly or using domain knowledge.
2. Evaluate the fitness of each chromosome based on an objective function.
3. Select individuals from the population to create a new generation based on their fitness, favoring better solutions.
4. Apply genetic operators: crossover (recombination) and mutation, to create offspring with variations.
5. Replace the old population with the new generation of individuals.

By repeating the 2-5 steps for a predefined number of generations or until convergence to an optimal solution. The method for each step defines a particular specific type of genetic algorithm. We propose a particular method in SolPOC without having the guarantee that this method is the best one.

5.3.4. PSO: Particle Swarn Optimization

The Particle Swarm Optimization (PSO) algorithm is a population-based optimization technique inspired by the social behavior of birds flocking or fish schooling. In PSO, a group of particles (potential solutions) moves through the search space to find the optimal solution. The PSO algorithm is iterative and relies on the collective information sharing among particles to explore and exploit the search space efficiently, converging towards an optimal or near-optimal solution.

5.3.5. Simulated_annealing.

Simulated Annealing is a probabilistic optimization algorithm used to find global or near-global optimal solutions in complex search spaces. Simulated Annealing mimics the annealing process in metallurgy, where a material is slowly cooled to minimize defects and achieve a stable structure. Similarly, it explores the solution space by allowing "bad" moves early on but gradually becomes more selective, converging towards an optimal solution.

5.3.6. Strangle

The algorithm referred to here as "strangle" refers to the algorithm present in the 1st version of the code, named COPS (in French : Code d'Optimisation des Performances Solaire) and described in several research articles [2–4] . The algorithm proceeds by progressively reducing the admissible set of problem variables in order to identify a solution. The aim is to obtain at the end of the process the thickness of each thin layer of the stack described, to optimize the chosen performance criterion. Figure 29 summarizes the algorithm, here for selective stacks. Although simple, this algorithm gives good results, particularly in avoiding local minima.

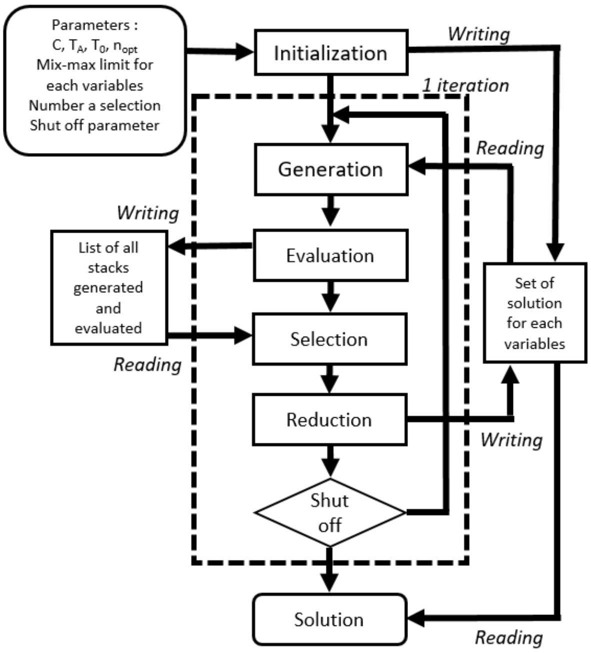


Figure 29 Strangle algorithm optimization method.

5.4. Default optimization method: DEvol

The primary algorithm under investigation in our research endeavors is Differential Evolution (DE), proposed by A.Moreau and P.Bennet [23]. Given the current state of knowledge, we wholeheartedly recommend its adoption when confronted with the dilemma of algorithm selection. The hyperparameters for DE, as elucidated in Table 5, exhibit a high degree of qualitative efficacy in the realm of thin-film optimization.

One pivotal consideration pertains to the determination of the number of generations, a parameter that proportionally influences the computational budget, indicative of the optimization process duration. For the sake of elucidation, we offer an order of magnitude for the requisite budget to achieve a high-quality optimization. This estimate is concomitantly associated with the computational time on a standard laptop equipped with an Intel Core i7-1165G7 processor operating at 2.80GHz and 16GB of RAM, based on 10 independent runs.

For a more comprehensive understanding and exemplification of our proposed methodology, additional instances can be found on section Examples: the VIII, p62 of this document. To access detailed hyperparameter information, kindly refer to the launch files or the text file titled "*optimization.txt*"

The data is provided solely for informational purposes. Numerous factors can influence the budget required for effective optimization and the computation time.

We recall that budget = *pop_size* * *nb_generation*

Table 7 :Typical Budget values and time calculation for different problems (v0.9.0)

Type of coating : (example)	**Budget**	**Time calculation (10 run)**
Antireflective coating, 3 layers ($TiO_2/Al_2O_3/SiO_2$)	600	7 s
Antireflective coating, 3 porous SiO_2 layers	900	233 s
Antireflective coating, 6 theoretical layers	1800	20 s
Cermet Selective coating, 3 layers with 1 cermet layer	750	120 s
Cermet Selective coating, 6 layers with 3 cermet layers	1200	550 s
Silvered Low-e coating, 6 layers. Double $Ag/TiO_2/SiO_2$	2400	27 s
Bragg mirror on glass, 10 layers $BK7/(TiO_2/SiO_2)_5$	3000	30 s
Bragg mirror on glass, 20 layers $BK7/(TiO_2/SiO_2)_{10}$	9000	140 s

5.5. Cost functions: the callable *evaluate*.

All functions are present in SolPOC package. They are also present in the files named *function_SolPOC*, present on GitHub. The *evaluate()* functions are callable representing cost functions utilized in the program. An *evaluate()* function takes an individual and the container as input. An individual is at a minimum a list of thicknesses, i.e., a stack, meaning a possible solution (refer to paragraph Individual: stack description, p9). Note that an individual may be more than just a list of thin layer thicknesses, possibly including:

- The addition of a volume fraction (vf) if one of the thin layers is a composite layer, i.e., a mixture of two materials (e.g., a $W\text{-}Al_20_3$ cermet or a porous layer).
- The addition of refractive indices if optimizing both thickness and the real refractive index of a thin layer. It is assumed that the real refractive index is constant, and k = 0 at all wavelengths.

In all cases, an *evaluate()* function returns the individual's performance, i.e., a score between 0 and 1. The current list and description of cost functions present in the code are provided below. Table 4 synthesizes the relationship between cost functions and parameters.

Name of the evaluate function	Optional parameters
Evaluate_R_s, Evaluate_A_s, Evaluate_T_s	Wl_sol and Sol_Spec
Evaluate_rh	C, T_air, T_abs, Wl_sol and Sol_Spec
Evaluate_T_pv Evaluate_A_pv	Wl_sol and Sol_Spec, Wl_PV and Signal_PV
Evaluate_T_vis	Wl_sol and Sol_Spec, Wl_H_eye and Signal_H_eye
Evaluate_low_e	Wl_sol and Sol_Spec Lambda_cut_1
Evaluate_RTR Evaluate_TRT	Wl_sol and Sol_Spec Lambda_cut_1 Lambda_cut_2
Evaluate_netW_PV_CSP	Wl_sol and Sol_Spec Lambda_cut_1 Lambda_cut_2 Poids_PV
Evaluate_fit	Signal Signal_2

Table 8 : List of symbols used in cost functions.

- C : solar concentration factor, from the solar collector.
- $E_{BB}(T_A)$: thermal emittance using a black body law and the temperature of the absorber (T_{abs})
- I : solar irradiation of the solar spectra used
- J(λ) : solar spectra irradiance, en W/m^2
- R(λ) : Reflectivity of the stack, for each wavelength
- r_H : heliothermal efficiency, also called helio to thermal efficiency.
- S_{PV} (λ) : The "signal" of the PV cell, i.e., its ability to absorb solar radiation based on wavelengths.
- S_{Th} : (λ) : The "signal" of the thermal absorber, i.e., its ability to absorb solar radiation based on wavelengths.
- T(λ) : Absorptivity of the stack, for each wavelength.
- T(λ) : Transmissivity of the stack, for each wavelength.
- T_0 : Ambient temperature (in K), a parameter defined by the user.
- $T_{A\,bs}$: temperature (in K) of the thermal absorber ;
- λ_1 et λ_2 : spectral domain for the solar spectra, often 320 nm for λ_1 and 2500 nm for λ_2
- λ_{cut_1} et λ_{cut_2} cut-off wavelengths
- σ i: s the Stefan-Boltzmann constant

5.5.1. evaluate_R

The callable *Evaluate_R* calculates the stack's average reflectivity in wavelengths, defined by the vector Wl. No weighting is applied: all wavelengths are equally important. The solar (or other) spectrum is of no importance in calculation. The equation used is as follows:

$$\hat{R} = \frac{1}{n}\sum_{i=1}^{n} R(\lambda)d\lambda$$

Eq. 6

5.5.2. evaluate_T

The callable *Evaluate_T* calculates the average reflectivity of the stack in wavelengths, defined by the vector Wl. No weighting takes place: all wavelengths are equally important. The solar (or other) spectrum is of no importance in calculation. The equation used is as follows:

$$\hat{T} = \frac{1}{n}\sum_{i=1}^{n} T(\lambda)d\lambda \quad \textit{Eq. 7}$$

5.5.3. evaluate_R_s

The callable *Evaluate_R_s* calcul the solar reflectance (R_S). In solar reflectance is the stack reflectance spectrum $R(\lambda)$ weighed by a solar spectrum $J(\lambda)$ and integrated over wavelength, to calculate the total solar power (in W/m^2) reflected by the stack. This value is divided by the total power received from the Sun, to obtain the solar-weighted reflectance R_S. The solar reflectance is the capacity to reflected sun irradiance. As an example, a mirror with a solar reflectance of 0.95 means that the mirror reflects 95% of all the sunlight flux density, per unit of surface. This value can directly be calculated with the function *SolarProperties*. Note according *Rs*, *Ts*, *As :* Read page 52 for more information about the solar spectrum and for choose λ_1 and λ_2. We recommend λ_1 = 320 nm and λ_2 = 2500 nm with a 5 nm step.

$$R_S = \frac{\int_{\lambda 1}^{\lambda 2} R(\lambda)\cdot J(\lambda)\cdot d\lambda}{\int_{\lambda 1}^{\lambda 2} J(\lambda).d\lambda} \quad \textit{Eq. 8}$$

5.5.4. evaluate_R_s_AOI

The callable evaluate_R_s_AOI extends the previous function Evaluate_R_s by introducing the effect of the angle of incidence (AOI) on the solar reflectance. While *Evaluate_R_s* computes the solar-weighted reflectance at normal incidence, this function evaluates the same property over a range of incident angles (typically 0°, 10°, 20°, 30°, and 40°) and returns the average value. The purpose is to obtain a more realistic estimation of the coating's optical performance under varying illumination conditions, representative of real solar operation.
The reflectance spectrum *R(λ)* is then calculated via RTA, and the corresponding solar reflectance is derived using SolarProperties, which integrates *R(λ)* over the solar spectrum *J(λ)*. The final indicator, R̅_s, represents the mean solar-weighted reflectance over all considered angles:

$$\bar{R}_s = \frac{1}{N}\sum_{i=1}^{N} \frac{\int_{\lambda_1}^{\lambda_2} J(\lambda)\cdot R_i(\lambda)\cdot d\lambda}{\int_{\lambda_1}^{\lambda_2} J(\lambda)\cdot d\lambda}$$

This quantity expresses the coating's global ability to reflect solar irradiance over a realistic range of incidence angles. A value close to 1 indicates a highly reflective behavior even under oblique illumination, an important feature for solar mirrors and RTR-type coatings exposed to varying solar positions throughout the day.

5.5.5. evaluate_T_s

The callable *Evaluate_T_s* calcul the solar transmittance (T_S), such as the solar reflectance. This value can directly be calculated with the function *SolarProperties*. Note according *Rs*, *Ts*, *As :* Read page 52 for more information about the solar spectrum and for choose λ_1 and λ_2. We recommend λ_1 = 320 nm and λ_2 = 2500 nm with a 5 nm step.

$$T_S = \frac{\int_{\lambda 1}^{\lambda 2} T(\lambda)\cdot J(\lambda)\cdot d\lambda}{\int_{\lambda 1}^{\lambda 2} J(\lambda).\, d\lambda} \tag{Eq. 9}$$

5.5.6. evaluate_A_s

The callable Evaluate_A_s calcul the solar transmittance (A_S), such as the solar reflectance. This value can directly be calculated with the function *SolarProperties*. Note according *Rs*, *Ts*, *As :* Read page 52 for more information about the solar spectrum and for choose λ_1 and λ_2. We recommend λ_1 = 320 nm and λ_2 = 2500 nm with a 5 nm step.

$$A_S = \frac{\int_{\lambda 1}^{\lambda 2} A(\lambda)\cdot J(\lambda)\cdot d\lambda}{\int_{\lambda 1}^{\lambda 2} J(\lambda).\, d\lambda} \tag{Eq. 10}$$

5.5.7. evaluate_T_pv and evaluate_A_pv

The callable *Evaluate_T_pv* evaluate the solar transmittance, for a PV cells. As solar transmittance, the stack transmittance spectrum $T(\lambda)$ is first weighed by a solar spectrum $J(\lambda)$ and in second weighted by a PV cell response ($S_{PV}(\lambda)$). In need, a PV cell cannot convert all wavelenght into electricity. Typical PV cells response in wavelength is present in the *PV_cells.txt* file. Notes that the wavelength domain can be reduced, depending on the PV cells used.

$$T_{PV} = \frac{\int_{\lambda 1}^{\lambda 2} T(\lambda)\cdot S_{PV}(\lambda)\cdot J(\lambda)\cdot d\lambda}{\int_{\lambda 1}^{\lambda 2} S_{PV}(\lambda)\cdot J(\lambda)\cdot d\lambda} \tag{Eq. 11}$$

The *evaluate_A_pv* cost function is very similar: the transmissivity curve is replaced by the absorptivity curve. This function can be used to maximize the antireflective coating on a opaque PV cells.

$$A_{PV} = \frac{\int_{\lambda 1}^{\lambda 2} A(\lambda)\cdot S_{PV}(\lambda)\cdot J(\lambda)\cdot d\lambda}{\int_{\lambda 1}^{\lambda 2} S_{PV}(\lambda)\cdot J(\lambda)\cdot d\lambda} \tag{Eq. 12}$$

5.5.8. evaluate_T_vis

Evaluate_T_vis calcul the Visible Solar Transmittance, according to a human eye sensitivity to wavelength. In need, a human eye is not equally sensitive to all wavelengths, so we need Normalized relative spectral distribution for the calculation of the Visible Solar Transmittance (*Tvis*). Typical human eye sensitivity is present in a text files *Human_eye.txt*.. Notes that the wavelength domain can be reduced, depending of the PV cells used.

$$T_{vis} = \int_{370\,nm}^{780\,nm} T(\lambda) \cdot S_{vis}(\lambda) \cdot d\lambda \qquad \text{Eq. 13}$$

5.5.9. evaluate_rh

Globally, the heliothermal efficiency R_h represents the capacity for a coating to be a good candidate or a not for solar thermal conversion at high temperature ($T_A >> T_0$). This cost function is necessary for selective coating, used in solar concentrated system. This value quantifies the capacity of the absorber to convert incident solar radiation into heat, to be transferred to a heat transfer fluid. These values are the ratio of absorbed solar flux density, minus the radiating thermal losses (due to the radiating exchange between the cold environment and the hot absorber, given by Stefan-Boltzmann law), divided by the total concentrated solar flux density received by the absorber [17], [22]. Notes than convective and conductive thermal losses are also present for real thermal absorbers, but they are neglected here i) compared to much higher radiating losses (σT^4) and ii) most solar thermal absorber at high temperature operate under vacuum/low pressure.

$$R_h = \frac{Absorbed\ Flux - Radiative\ losses}{Total\ flux} = A_S - \frac{E_{BB} \cdot \sigma(T_A^4 - T_0^4)}{C \cdot I \cdot \eta_{opt}} \qquad \text{Eq. 14}$$

Different parameters are present in SolPOC for the calculation of heliothermic efficiency (rh, here calculated with a function, named *helio_th*. The solar absorptance A_S and thermal emittance $E_{BB}(T_A)$), which are both derived from spectral reflectance $R(\lambda)$ are calculated respectively with the function *SolarProperties()* and the function named *E_BB()* (Emissivity calculated from a Black Body). The optical performance of the concentrator η_{opt} represents an average value that includes several factors such as the mirror solar reflectance, protective glass transmittance (if any), soiling of optical components, cosines effects and shadowing effects, etc. We selected a value $\eta_{opt} = 0.70$ from literature [24].

5.5.10. evaluate_low_e

The function *Evaluate_low_e* calculates the optical performance of the stack to generate a low-e profile. Such thin-film coatings are utilized in building glazing to manage solar gain and minimize heat loss. The objective of a low-e coating is to:

- Remain transparent from the beginning of the solar spectrum (often 280 nm) to a cut-off wavelength λ_{cut_1}, maximizing solar gains and enhancing visual comfort for occupants by allowing natural light penetration.
- Become reflective from the cut-off wavelength λ_{cut_1}. A highly reflective behavior implies low infrared emission (hence the name low-e glass; refer to equation Eq. 15) and, consequently, limited heat loss through radiation.

Figure 30 taken from the bibliography, illustrates the ideal reflectivity and transmissivity spectrum of a low-e glass (figure on the left) and an example of treatment (figure on the right) [25]. As example and from the left Figure a low-e glasses with an “idealized spectra for hot climate coating applications" should have a value of λ$_{cut\_1}$ ≈ 1000 nm. Concerning coatings for "Idealized spectra for cold climate coating applications", the λ$_{cut\_1}$ value is ≈ 2.5 µm).

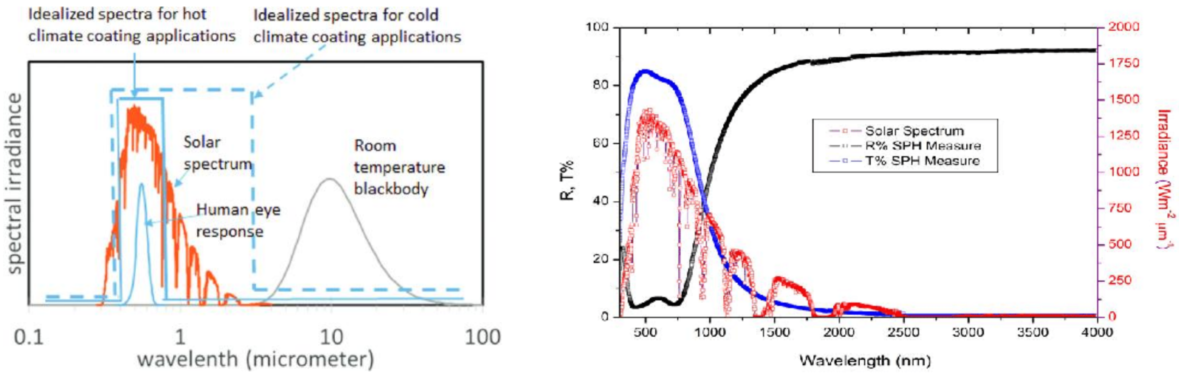


Figure 30 : spectrum of low-e coating from [25,26]

For this application, we have written the following equation, which is used in the *evaluate_low*_e function. The parameter λ$_{cut\_1}$ is fixed before the optimization process and must be placed in the "*parameters*" Dictionary: parameters, p41.

$$\eta_{low-e} = \frac{\int_{\lambda_1}^{\lambda_{cut\_1}} J(\lambda)\cdot T(\lambda)d\lambda + \int_{\lambda_{cut\_1}}^{\lambda_2} J(\lambda)\cdot R(\lambda)d\lambda}{\int_{\lambda_1}^{\lambda_2} J(\lambda)\cdot d\lambda} \qquad \textit{Eq. 15}$$

5.5.11. evaluate_RTR and TRT

Evaluate_RTR takes over and completes the previous function*, evaluate_low_e().* We now seek to reflect the radiation in the short wavelength range to obtain a profile: Reflector - Transparent - Reflector. The ideal treatment is now :

1. Transparent reflector from the beginning of the solar spectrum (often 280 nm) to a cut-off wavelength λ$_{cut\_1}$
2. Transparent between two cut-off wavelengths λ$_{cut\_1}$ and λ$_{cut\_2}$.
3. Reflector beyond the second cut-off wavelength λ$_{cut\_2}$.

To achieve this, we've written the following function. The two parameters λ$_{cut\_1}$ and λ$_{cut\_2}$ are fixed during calculation and optimization. They are placed in the *parameters* dictionary and should normally be defined when SolPOC is launched.

$$\eta_{RTR} = \frac{\int_{\lambda_1}^{\lambda_{cut\_1}} J(\lambda)\cdot R(\lambda)d\lambda + \int_{\lambda_{cut\_1}}^{\lambda_{cut\_2}} J(\lambda)\cdot T(\lambda)d\lambda + \int_{\lambda_{cut\_2}}^{\lambda_2} J(\lambda)\cdot R(\lambda)d\lambda}{\int_{\lambda_1}^{\lambda_2} J(\lambda)\cdot d\lambda} \qquad \textit{Eq. 16}$$

The opposite cost function is named TRT. It's correspond to profile: Transparant - T – Reflector – Transparent. The ideal treatment is now :

1. Transparent from the beginning of the solar spectrum (often 280 nm) to a cut-off wavelength λ_{cut_1}
4. Reflector between two cut-off wavelengths λ_{cut_1} and λ_{cut_2}.
5. Transparent beyond the second cut-off wavelength λ_{cut_2}.

$$\eta_{TRT} = \frac{\int_{\lambda_1}^{\lambda_{cut\_1}} J(\lambda)\cdot T(\lambda)d\lambda + \int_{\lambda_{cut\_1}}^{\lambda_{cut\_2}} J(\lambda)\cdot R(\lambda)d\lambda + \int_{\lambda_{cut\_2}}^{\lambda_2} J(\lambda)\cdot T(\lambda)d\lambda}{\int_{\lambda_1}^{\lambda_2} J(\lambda)\cdot d\lambda} \qquad \text{Eq. 17}$$

5.5.12. evaluate_netW_PV_CSP

evaluate_netW_PV_CSP extends the logic of the previous evaluation functions to simultaneously assess the optical and energetic performance of a coating in the context of a combined Photovoltaic–Concentrated Solar Power (PV–CSP) system. The goal is to quantify how effectively a multilayer structure (defined by its thicknesses and optical constants) can transmit the useful part of the solar spectrum to the PV cells while reflecting the infrared portion toward the thermal receiver. The evaluation is based on the spectral power balance between the transmitted and reflected fluxes, each weighted according to their role in the system:

- the transmitted solar flux within the PV sensitivity range (weighted by the parameter *poids_PV*),
- and the reflected solar flux in the thermal range contributing to CSP performance.

The function converts the optimization variable (*individual*) into a physical multilayer stack, computes its spectral reflectance (*R*) and transmittance (*T*), and integrates these quantities over wavelength using the solar spectrum and predefined spectral signals (*Signal_PV* and *Signal_Th*).
The resulting metric, η_net_PV_CSP, represents the global energetic efficiency of the coating, defined as:

$$\eta_{net\_PV\_CSP} = \frac{\int J(\lambda)\cdot T(\lambda)\cdot S_{PV}(\lambda)d\lambda + \int J(\lambda)\cdot R(\lambda)\cdot S_{Th}(\lambda)d\lambda}{\int J(\lambda)\cdot S_{PV}(\lambda)d\lambda + \int J(\lambda)\cdot S_{Th}(\lambda)d\lambda}$$

This performance indicator balances both photovoltaic and thermal contributions, guiding the optimization toward realistic RTR-like coatings that maximize overall solar energy utilization instead of behaving like perfect mirrors.

5.5.13. evaluate_fit

The callable named fit (as *evaluate_fit_RT*, *evaluate_fit_R*, *evaluate_fit_T*) are designed for optical reverse engineering, where the goal is to retrieve the thicknesses of thin-film layers that best reproduce an experimental optical response. The function compares the simulated reflectance (*R(λ)*) and transmittance (*T(λ)*) spectra of a multilayer stack with the corresponding experimental measurements. Each *individual* represents a possible configuration of layer thicknesses proposed by the optimization algorithm.

After converting the optimization vector into a physical stack via *Individual_to_Stack*, the function computes the spectral reflectance and transmittance using RTA. The deviation between the calculated and experimental spectra is then quantified using a normalized mean squared error (MSE), ensuring that the cost function is scaled between 0 and 1, regardless of signal amplitude. Two costs are computed — one for *R(λ)* and one for *T(λ)* — and the final fitness value corresponds to their average:

$$\text{cost} = \frac{1}{2}\left[\text{MSE}_{\text{norm}}(R, R_{exp}) + \text{MSE}_{\text{norm}}(T, T_{exp})\right]$$

A value close to 0 indicates a nearly perfect match between simulated and measured data, meaning the retrieved layer thicknesses faithfully reproduce the experimental optical behavior. This fitting process is fundamental for characterizing unknown coatings, validating optical models, or refining material parameters based on real spectral measurements.

5.6. Note according Rs, Ts, As: the solar spectrum

For solar performances, such as solar reflectance (R_S), solar transmittance (T_S) or solar absorptance (A_S), we need a solar spectrum, which cannot be replaced by a black body. The solar spectra used by default in SolPOC are the ASTM G173-03.

5.6.1. The ASTM G173-03 solar spectra

The chosen by default solar spectrum is the ASTM G173-03 AM 1.5 defined between 280 and 4000 nm, also known as the AM 1.5 solar spectrum [27,28]. The Air Mass (AM) factor represents the atmosphere thickness through by the sunlight at ground level. With a value of 1.5, this solar spectrum is representative of sun light on the United States. The specific value of 1.5 has been selected in the 1970s for standardization purposes and is still in use today.

The AM 1.5 solar spectrum can be split in three:

- The extraterrestrial solar spectrum, which include the direct irradiance from the sun above the atmosphere, for example at altitude superior to 100 km. It the AM0 solar spectrum with a total irradiance value between 280 to 4000 nm at 1366 W/m^2.
- the Global Tilt (GT) solar spectrum, which includes direct irradiance from the sun and diffuse sunlight coming from the ground or clouds. This solar spectrum cannot be concentrated in optical systems. The total irradiance value between 280 to 4000 nm is 1000.4 W/m^2.
- the Direct and Circumsolar (DC) solar spectrum, which includes only direct irradiance from the sun and its corona. This solar spectrum can be concentrated in optical systems, such as mirrors or lenses. The total irradiance value between 280 to 4000 nm is 900.8 W/m^2.

Note than : GT = DC + Diffuse (Global Tilt equal Direct and Circumsolar plus diffuse Solar Spectrum)

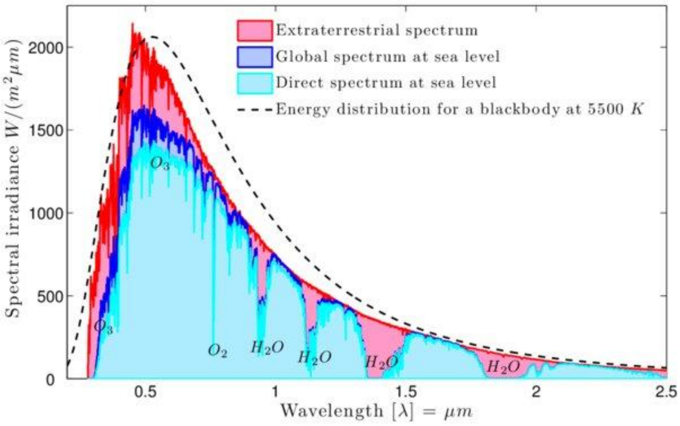


Figure 31 : Illustration of the different solar spectrum

5.6.2. Witch one should I use?

If you are uncertain about which solar spectrum to use, either DC or GT, a straightforward approach is to consider the question, "*Does my device or my coating incorporate optical concentration*?"

- If the answer is no, the Global Tilt solar spectrum can be employed. This is suitable for the majority of applications, such as anti-reflective coatings for vision, low-emissivity glazing, and similar cases.
- If the answer is yes, the Direct and Circumsolar spectrum should be used. This is applicable, for instance, to optical surfaces utilizing concentrated solar thermal energy.

In cases of uncertainty, it's important to note that this paragraph provides an initial approach. For more detailed guidance, please refer you to the relevant bibliography of your domain.

5.6.3. Integration process and location of the file

In the integration process, the spectral ranges from 280 to 320 nm and from 2500 to 4000 nm can be ignored, due to the low irradiance in these ranges: they represent less than 1% of the total solar incident power. This reduced spectral range of 320-2500 nm is in fact recommended by SolarPACES organization in solar reflectance guidelines [29]. The document also recommends the use of a wavelength step $d\lambda$ = 5 nm, often used in the code. The all ASTM G173-03 solar spectra are present in the text file (*SolSpec.txt*) located in the *Materials* folder. The files must be understood like this: the first column is the wavelength, in nm. The column n°2 to n°4 are respectively the DC solar spectrum, the extraterrestrial solar spectrum and the GT solar spectrum, are all in W/m^2nm^{-1}.

The solar spectrum file can be easily open with the function *open_SolSpec*, with include on optional parameters for selected the type of solar spectrum between “DC”, “GT” and “Extr”.

```
# Open the solar spectra, here DC
Wl_sol , Sol_Spec , name_SolSpec = open_SolSpec('Materials/SolSpec.txt' , ‘DC’)
# Interpolate the solar spectrum
```

```
Sol_Spec = np.interp(Wl, Wl_sol, Sol_Spec) # Interpolate the solar spectrum
```

5.7. The callable *selection*

The *selection* callable is employed to either maximize or minimize the cost function defined in the evaluate function. This callable can be utilized in two different ways:

1. If the optimization algorithm is "*Optimize_agn*" or "*Strangle*," i.e., two genetic algorithms, the callable is used to invoke a function. We find either the "selection_max" function, which returns a share of individuals with the highest score according to the cost function (named in *evaluate*), or the "*selection_min*" function, which returns a share of individuals with the lowest scores. This method allows for the implementation of various selection functions, for instance, by modifying the number of individuals that will serve as "parents" for the next generation.

2. For other optimization algorithms, we use only the function name, like a Boolean. We look for "selection_min" or "selection_max," and no other functions are planned. We utilize an if loop.

    a. If the callable is *selection_min*, the optimization method optimize according to the cost function.
    b. If the callable is *selection_max*, the optimization method optimize to 1 minus the cost function.

This approach is justified, as all other algorithms (*DEvol*, *One_plus_One_ES*, *PSO*, or *simulated annealing*) are only coded solely to **minimize the cost function**. It the cases for most optimization method, which allow us to only minimize the cost function. But in other cases, our wish is to maximize.

In SolPOC maximizing one cost function is therefore equivalent to minimizing 1 minus one cost function. See the Jupyter Notebook for more details

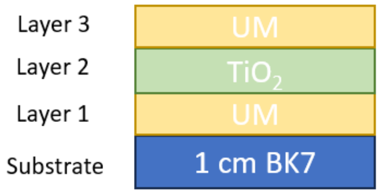
VI. Optimizing a stack with a list potential material

In the design of thin-film optical stack, the choice of material critically affects device performance by nature. In some situations, the selection of the materials in the thin layer stack can be deducted by the user, but it's not allowing the case. In some other situations, the optimal combination of materials and thicknesses is unknown a priori. Exploring all possible combinations manually or via brute-force simulation is computationally expensive, especially as the number of layers or candidate materials increases.

Heuristic optimization algorithms, such as Differential Evolution (DE), are very effective at adjusting continuous numerical parameters, like layer thicknesses, to find high-performance designs. However, these algorithms are not naturally suited for categorical variables, such as material selection, which is inherently discrete (e.g., choosing between Material A, B, or C).

6.1. Proposed methodology

To overcome the fact that heuristic methods can only deal with continuous numerical parameters, we adopt an approach that maps discrete material choices to continuous numerical variables. Each material is represented by a number (typically in the range [-1, 1]), which is then probabilistically interpreted as a choice of material. Figure 32 show an example of a 3 thin layer stack, which contains two Unknow Materials (noted '*UM*'). The Unknow Materials can be ZnO or Al2O3. During the optimization process each layer is described by two numbers: the thickness (noted in d_Stack) and X in [-1, 1] range which is describing the probability of choosing one material instead one or two others in the material list option *Mat_Option* through a function. As other cases, such as EMA, the function *Individual_to_stack* function assures that bride, completed by the function *fill_material_stack*.



```
Mat_Stack = ['BK7', 'UM', 'TiO2', 'UM']
Mat_Option = ['ZnO', 'Al2O3']
Individual = [1 mm,  120 nm, 58 nm, 129 nm, -0.916, 0.628, 0.372, 0.474]
d_Stack = [1 mm,  120 nm, 58 nm, 129 nm,]
X = [ -0.916, 0.628, 0.372, 0.474]
```

Figure 32 : Example of stack with Unknow Material

For example, -1 corresponds to a 100% probability of selecting Material A, 1 corresponds to 100% probability of selecting Material B, and intermediate values encode mixed probabilities. Additional details are provided in the section Individuals with unknown material.

This encoding allows the optimizer to simultaneously explore both layer thicknesses and material combinations within a single numerical framework. This approach provides several advantages:

- It preserves the efficiency and convergence properties of continuous optimization algorithms.
- It enables smooth exploration of the design space, avoiding the combinatorial explosion associated with discrete searches.
- It allows extending the method to more than two materials, using probabilistic mappings such as linear interpolation, sigmoids, or Gaussian-based transitions.

Ultimately, this method bridges the gap between discrete material selection and continuous numerical optimization, enabling fully automated exploration of complex thin-film stack designs with unknown optimal compositions.

6.2. Global owerview

The following section provides a description of the optimization process and its main steps:

1. Step 1 – Define the stack structure: Specify the thin-film stack by listing the materials for each layer. Known materials are written explicitly (e.g., SiO2, Al2O3), while layers with unknown materials are indicated by writing 'UM' (Unknown Material). It is therefore possible to define a stack containing both known and unknown layers in the same structure.

2. Step 2 – Define the list of candidate materials: Provide in the variable Mat_Option a list of three possible materials from which the algorithm can select during the optimization process.

Figure 33 illustrates an example taken from the stack defined in the script template_optimization_with_materials.py, located in the optimization folder. In this example, the substrate, and layers 1, 2, and 4 are known, whereas layers 3, 5, and 6 are unknown. The optimization algorithm searches among three possible candidates for the unknown layers: SiO_2, ZnO, and TiO_2.

```
Mat_Stack = ["BK7", "SiO2", "TiO2",
"UM", "TiO2", "UM", "UM"]

Mat_Option = ["SiO2", "ZnO", "TiO2"]
```

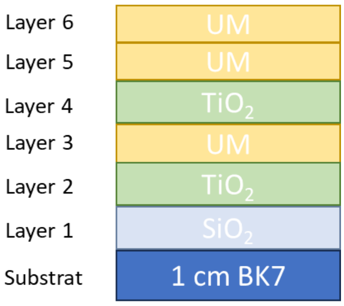


Figure 33 : Example available in the template_optimization_with_materials.py

The use of 'UM' and *Mat_Option* within *Mat_Stack* has two main effects on the code:

1. A material named 'UM' actually exists in the database. Its refractive index values are irrelevant — it simply allows the *Mat_Stack* function to load a placeholder material and to build a temporary version of n_Stack and k_Stack without causing errors.
2. The presence of 'UM' in the *Mat_Stack* list, together with the definition of a *Mat_Option* list, enables the code to adopt the correct structure for creating an individual, that is, a combined list of layer thicknesses and continuous variables ranging from [-1, 1].

During the optimization process:

3. The optimizer receives an individual. It must evaluate this individual by ensuring that the *RTA* function receives the correct arguments*: d_Stack, n_Stack*, and *k_Stack*.
4. The *individual_to_stack* function separates the thickness values from the material-choice values contained in the individual. To do this, it checks for the presence of a material-choice list by detecting the *Mat_Option* key in the parameters dictionary. The output of this step is *d_Stack* (the list of layer thicknesses). However, at this stage, the refractive index matrices *n_Stack* and *k_Stack* do not yet correspond to the correct materials.
5. The *fill_material_stack* function then updates the *Mat_Stack* list, which describes the materials of the stack. It uses the *choose_material* function to replace each 'UM' entry in *Mat_Stack* with one of the candidate materials defined in *Mat_Option.*

> The Mat_Stack generated by *choose_material* is inherently stochastic, as the values *X*, represent only a probability distribution for selecting a specific material.
>
> Consequently, two separate calls to the function will generally produce two different stacks.

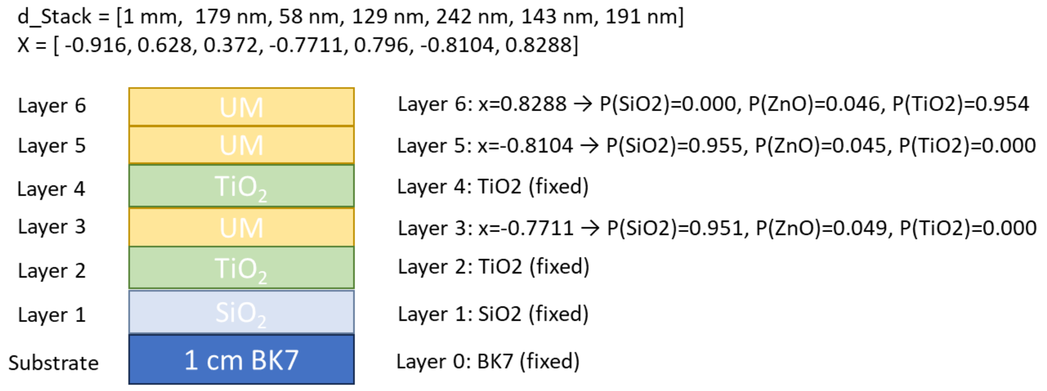
6. The function *fill_material_stack* thus returns a list of materials (*Mat_Stack*), which can then be processed by Made_Stack to obtain the corresponding *n_Stack* and *k_Stack*.

➢ At this stage, the stack can be fully evaluated, as all necessary quantities, *d_Stack*, *n_Stack*, and *k_Stack*, are available.

7. The entire process is repeated multiple times by the optimizer, which adjusts both the thicknesses and the X parameters for each layer.

At the end of the optimization, the optimizer returns the best solution (an individual), which consists of a list of thicknesses and X parameters. In the current version of SolPOC (v0.9.7), the specific materials corresponding to the optimized stack are not directly retained within the optimization, although they could be easily retrieved if needed. The function *print_material_probabilities* allows users to display the probability of each material being selected for each layer by performing a series of trials.

Figure 34 illustrates the output of template_optimization_with_materials.py with the seed set to 3963690230. It can be observed that layer 3 has a 95.1% probability of being SiO2, layer 5 has a 95.5% probability of being SiO2, and layer 6 has a 95.4% probability of being TiO2. The optimizer thus identifies a stack structure of [SiO2, TiO2, SiO2, TiO2, SiO2, TiO2], which corresponds precisely to the expected result in this example.



```
d_Stack = [1 mm,  179 nm, 58 nm, 129 nm, 242 nm, 143 nm, 191 nm]
X = [ -0.916, 0.628, 0.372, -0.7711, 0.796, -0.8104, 0.8288]
```

```
Layer 6: x=0.8288 → P(SiO2)=0.000, P(ZnO)=0.046, P(TiO2)=0.954
Layer 5: x=-0.8104 → P(SiO2)=0.955, P(ZnO)=0.045, P(TiO2)=0.000
Layer 4: TiO2 (fixed)
Layer 3: x=-0.7711 → P(SiO2)=0.951, P(ZnO)=0.049, P(TiO2)=0.000
Layer 2: TiO2 (fixed)
Layer 1: SiO2 (fixed)
Layer 0: BK7 (fixed)
```

Figure 34 : Output of the template_optimization_with_material.py scripts, with the seed 3963690230

In the template, we finalize the optimization process by performing a second optimization that adjusts only the thicknesses of the five stacks generated by the previous solution. This step allows for fine-tuning of the results, once the materials have been nearly fully identified.

6.3. Probability distribution for choose materials

The *choose_material* function transforms a value X in the range $[-1,1]$ into a probabilistic distribution that allows selecting between two materials (A and B) or three materials (A, B, or C). We implemented three probability distribution methods with two guiding principles in mind:

1. Deterministic selection: For each material, there must exist a value of X such that the probability of selecting that material is 100%, allowing the optimization method to fix the choice of a material.
2. Probabilistic choice: Conversely, there must exist intervals of X where the probability of selecting one material is equal to that of another, enabling the optimization method to explore different material options.

The three implemented distribution methods, which can be provided as an argument to the choose_material function, are:

1. **Linear:** Produces a linear probability profile between two or three materials.
2. **Sigmoid:** Produces a sharp transition between two materials, with a plateau ensuring an almost 100% probability for one material over a wide interval of X. The parameter k controls the steepness of the slope; by default, $k = 10$ for two materials and $k = 15$ for three.

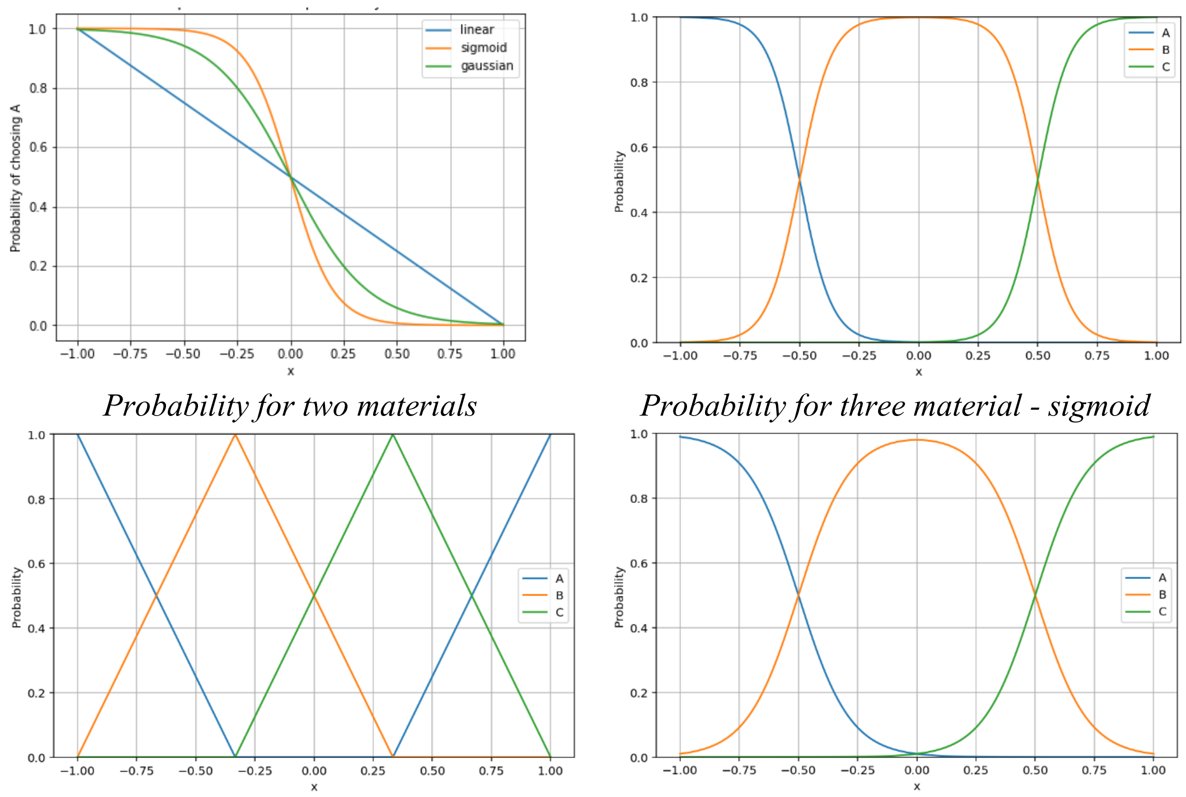
The sigmoid distribution uses the following equations implemented in the code

$p_A = \frac{1}{1+e^{kx}}, p_B = 1 - p_A$	2 materials
$p_A = \dfrac{1}{1+e^{k(x+0.5)}}, p_C = \dfrac{1}{1+e^{-k(x-0.5)}}, p_B = 1 - p_A - p_C$	3 materials

3. **gaussian**. Produces a gradual transition between two materials, with a plateau ensuring an almost 100% probability for one material over a narrow interval of X. By default, $\sigma = 0.6$ for two materials and $\sigma = 0.33$ for three.

$g_A = \exp\left(-\dfrac{(x+1)^2}{2\sigma^2}\right), g_B = \exp\left(-\dfrac{(x-1)^2}{2\sigma^2}\right)$ $p_A = \dfrac{g_A}{g_A + g_B}, p_B = \dfrac{g_B}{g_A + g_B}$	2 materials
$g_A = e^{-\frac{(x+1)^2}{2\sigma^2}}, g_B = e^{-\frac{x^2}{2\sigma^2}}, g_C = e^{-\frac{(x-1)^2}{2\sigma^2}}$ $p_A = \dfrac{g_A}{g_A + g_B + g_C}, p_B = \dfrac{g_B}{g_A + g_B + g_C}, p_C = \dfrac{g_C}{g_A + g_B + g_C}$	3 materials

Figure 35 illustrate different probability distributions for selecting between two materials (A and B, top left) and three materials (A, B, or C). The script *template_optimization_with_materials.py* allowed us to quickly test the effect of the three methods. In this script, the optimization method must identify three unknown materials (denoted UM) in the stack $["BK7","SiO2","TiO2","UM","TiO2","UM","UM"]$to maximize solar reflectance. The solution, already reported in the literature, is to alternate SiO2 and TiO2, i.e., $["BK7","SiO2","TiO2","SiO2","TiO2","SiO2","TiO2"]$. Among the tested distributions, we observed that the sigmoid profile yielded the best optimization performance.



Probability for two materials

Probability for three material - sigmoid

Probability for three material - linear

Probability for three material - gaussian

Figure 35 : Illustration of distribution probability

Based on this example, we currently recommend using the **sigmoid profile**.

However, we strongly encourage users to perform additional tests. As a first step, we suggest verifying that the optimization method can reproduce solutions already reported in the literature before attempting more complex cases.

VII. How SOLPOC use multicore CPU?

7.1. Multiprocessing

SolPOC allows you to work with the multiprocessing library. Multiprocessing is the ability for a code to run independent calculations on several cores of the same processor at the same time. Each core works independently, which is ideal for spreading the workload and saving time. This makes it possible to exploit the full capacity of recent processors.

7.1.1. Compatible computers

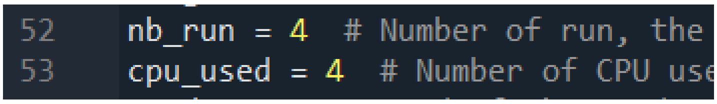
Normally, all today's computers incorporate a processor with several cores, at least 2. Every computer should therefore be compatible and benefit from multiprocessing code. To find out the number of cores in your processor:

1- Open the Control Panel and search for your processor type.
2- In Python, the *cpu_count* command in the multiprocessing library returns the number of "cores" detected.

To be precise, the number and type of cores in a processor can sometimes be complicated and separated between cores and threads. At this actual version, we don't have a formal answer on the use of different types of cores (core vs. thread) in Python via the multiprocessing library and the use of a pool. During the code testing and development, we successfully used a VM running Windows 10, running in a rack of 2 x Xeon Gold 5220r for a total of 48 real cores / 96 logical cores with 128 GB of DDR3 RAM.

7.2. Use the multiprocessing.

To use several cores during optimization, which is recommended for several runs, you just need to write an integer value (an int) in the "*cpu_used*" line.



```
nb_run = 4  # Number of run, the
cpu_used = 4  # Number of CPU use
```

Figure 36 : How fix the number of CPU used

When the code is launched, the script writes the number of cores detected (via *cpu_count*) and the number of cores used (via *cpu_used*) to the console. There's no security against the user: you can launch more cores than the number available. Normally, this does not result in an error. Once the code has been run, there is little difference from code without multiprocessing, except in terms of total computation time. The result score is displayed via a printout as soon as a core has finished. Note that the time displayed corresponds to the time taken by the core. The core then takes on the next problem in the list (the pool) in no chronological order. To get an idea of calculation time: Table 7.

7.3. Saving time

Based on our current observations and understanding, it's existe an difference between vectorization and multiprocessing.

1. Vectorization consists in reformulating computations to take advantage of optimized matrix operations, often implemented in C or Fortran. This approach reduces the overhead associated with Python loops and significantly improves performance when handling large datasets.

- ➢ **It's provided using the NumPy package**

2. In contrast, parallelization using the multiprocessing module relies on distributing tasks across several independent processes. While this enables the use of multiple CPU cores, it also introduces additional overhead related to process creation and inter-process communication.

- ➢ **It's provided on SolPOC using the Multiprocessing package**

Consequently, vectorization is particularly well suited for numerically intensive operations on arrays, whereas parallelization is more relevant for independent, complex tasks that cannot easily be expressed in a vectorized form.

7.3.1. Vectorization

Similar to other packages, SolPOC relies on vectorization to efficiently compute optical properties through optimized matrix operations, thereby avoiding explicit loops over wavelengths. PyMoosh adopts a similar strategy, whereas TMM-fast applies vectorization simultaneously over wavelengths, incidence angles, and layer thicknesses. This design can indeed be advantageous when combined with PyTorch and GPU acceleration; however, we have observed that performance may degrade on CPUs when certain dimensions become large (e.g., multiple incidence angles or multiple stacks). In such cases, a trade-off between vectorization and explicit looping appears necessary to avoid excessive memory load. While our observations suggest these trends, we cannot claim with absolute certainty that vectorization always outperforms multiprocessing in every possible scenario.

They are list vectorized function in SolPOC

a. RTA3C (simplified version of RTA) and RTA for optical properties calculation
b. Bruggeman
c. BB, for Black Body calculation and E_BB, for emissivity
d. SolarProperties

7.3.2. Multiprocessing

The time saved by using multiprocessing depends on many factors. The information presented here is not definitive. In fact, the time saved depends on the processor, RAM, other applications outside Python, etc. The time saving is not guaranteed if you only run the calculation on one or two cores, and the time saving is not strictly linear with the number of processors. Following tests on a simple cost function (6-period Bragg mirror over the full solar spectrum), our Amdahl's Law code was found to be 97.7% parallelizable per fit (R2: 0.9969) [30]. This estimation was calculated through a fitting process ($R^2 = 0.9969$). This speedup is coupled with the benefits of the NumPy-based implementation of the Abélès formalism, allowing us to strongly decrease the calculation time. As an example, if 1 run on 1 CPU takes approximately 36s of calculation time, 48 parallelized runs on 48 CPUs take only 77s.

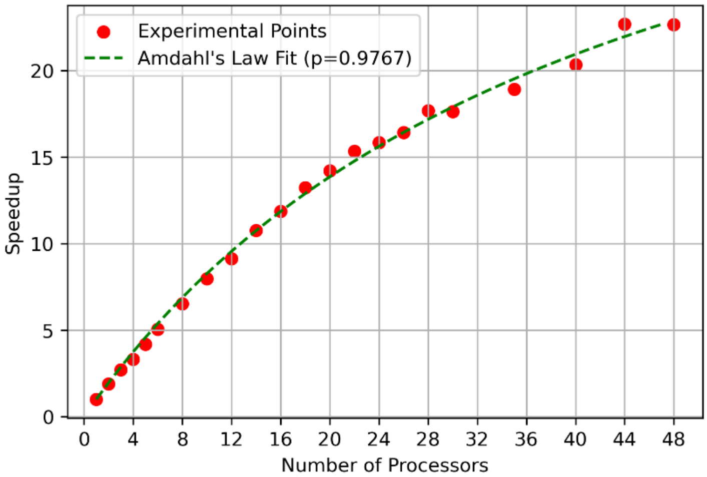


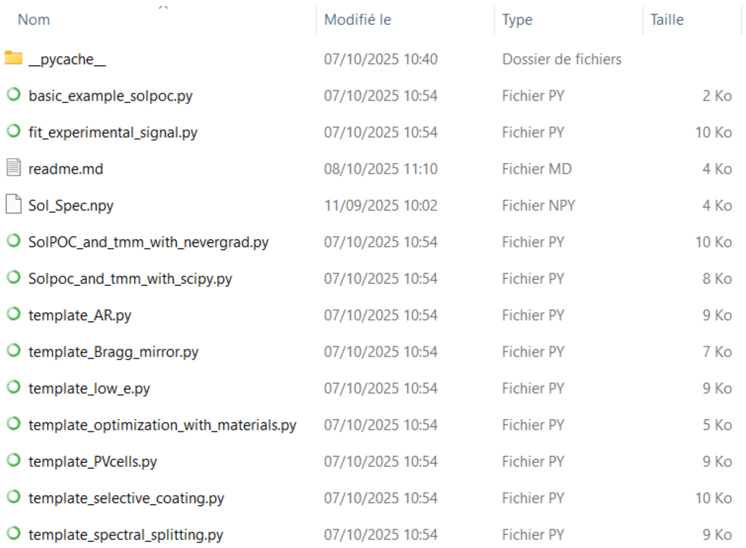
Figure 37 : Amdawl's law for SolPOC (v0.9.6) on 2 Intel Xeon Gold 5220r. If 1 run on 1 CPU takes approximately 36s of calculation time (example), 48 parallelized runs on 48 CPUs take only 77s.

Advice:

- Launch a number of runs that is an integer proportional to your number of cores, e.g., 8, 16, 24 runs for 8 cores computer or 6, 12, 24 runs for 6 cores computer
- Keep a share of resources available and monitor core workloads via the performance manager.
- Make sure your PC is well ventilated, as for more demanding applications (video games / photo editing) and monitor your battery for laptops.
- Information in the console may not be displayed as and when required.
- On servers with a large number of cores (we encountered an error on a 96-core server), it may not be possible to use all of them. The code then returns an error on launch.

VIII. Examples: the template Scrips

We provide a series of examples for the code, proposed as template scripts. These are 'template_multiprocess.py' files preconfigured to match practical cases. They are located in the folder Examples.



Nom	Modifié le	Type	Taille
__pycache__	07/10/2025 10:40	Dossier de fichiers	
basic_example_solpoc.py	07/10/2025 10:54	Fichier PY	2 Ko
fit_experimental_signal.py	07/10/2025 10:54	Fichier PY	10 Ko
readme.md	08/10/2025 11:10	Fichier MD	4 Ko
Sol_Spec.npy	11/09/2025 10:02	Fichier NPY	4 Ko
SolPOC_and_tmm_with_nevergrad.py	07/10/2025 10:54	Fichier PY	10 Ko
Solpoc_and_tmm_with_scipy.py	07/10/2025 10:54	Fichier PY	8 Ko
template_AR.py	07/10/2025 10:54	Fichier PY	9 Ko
template_Bragg_mirror.py	07/10/2025 10:54	Fichier PY	7 Ko
template_low_e.py	07/10/2025 10:54	Fichier PY	9 Ko
template_optimization_with_materials.py	07/10/2025 10:54	Fichier PY	5 Ko
template_PVcells.py	07/10/2025 10:54	Fichier PY	9 Ko
template_selective_coating.py	07/10/2025 10:54	Fichier PY	10 Ko
template_spectral_splitting.py	07/10/2025 10:54	Fichier PY	9 Ko

Figure 38 : Template scripts proposed in the example folde

Each example explains a real case of thin film stack and how it can be solved using SolPOC. The proposed examples complement the Jupyter Notebooks. The primary objective of SolPOC is to optimize the thickness of each thin layer in the stack.

- The choice of the optimization algorithm is specified in the callable "algo."
- The optimization objective is defined in the callable "evaluate," which thereby represents the cost function utilized by the code. This function can be minimized or maximized through the callable "selection."

To conduct these optimizations, we will employ a Dell laptop equipped with an Intel Core i7 processor featuring 4 cores and 8 GB of RAM. We use a laptop instead of a dedicated computing server for illustrative purposes; however, SolPOC can seamlessly run on a standard laptop.

An folder with the output of those scrips can be found as : Grosjean, A. (2025). SolPOC v0.9.7 correct output of exemples [Data set]. Zenodo. https://doi.org/10.5281/zenodo.17287049 [31]

8.1. Tutorial 1 : Optimize Stack Thicknesses

In this first series of two examples, we will optimize the thicknesses of two stacks.

Example 1a: Bragg Mirror

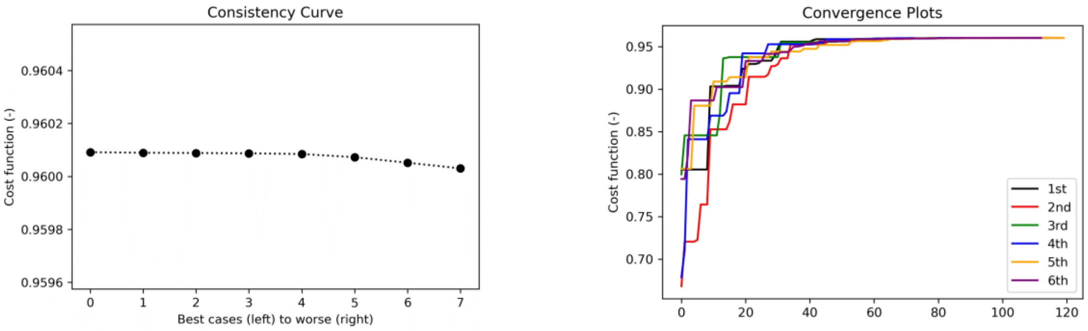
In the first example, we propose the optimization of a Bragg mirror, which is undoubtedly one of the most well-understood optical structures. It is a periodic multilayer structure composed of alternating layers of two materials with different refractive indices, each having optical thicknesses of a quarter wavelength (λ/4). Our objective is to recreate a Bragg mirror consisting of 4 periods of SiO_2/TiO2 (so a total of 8 thin layers) deposited on a BK7 glass substrate. We aim to achieve the highest average reflectivity between 500 and 650 nm, provide by the cost function "*evaluate_R_Brg*". To achieve this, here are the key optimization parameters of SolPOC and their justifications. The entire set of files generated by the code is present in the folder.

Code	Justification
Mat_Stack = ("BK7", "SiO2", "TiO2", "SiO2", "TiO2", "SiO2", "TiO2", "SiO2", "TiO2")	Description of the stack consisting of 8 thin layers of SiO2 and TiO2 deposited on glass.
Wl = np.arange(400 , 805, 5)	Spectral range from 400 nm to 800 nm, exceeding the specified range (500 – 650 nm).
Th_range = (0, 200)	The optimization seeks a solution with thin layer thicknesses ranging from 0 to 200 nm

algo = DEvol
selection = selection_max
cost_function = evaluate_R_Brg
	We employ the DEvol optimization algorithm. In line with our objective, the cost function is "evaluate_R_Brg," which we aim to maximize using "selection_max."

nb_run =8
cpu_used = 8
	The optimization will be conducted 10 times. To reduce computation time, 10 CPUs are utilized in parallel.

After initiating the optimization, the results are automatically saved in a folder created by the code. The Consistency Curve graph (Figure 20) demonstrates that the problem is well-resolved: the algorithm consistently rediscovers the highest achievable performance (8 runs out of 8), which is 96.01%. The Convergenceplots.png graph illustrates that the performance of the top 6 stacks (y-axis) converges towards the optimum as the optimization progresses (depicted on the x-axis). All other files created by SolPOC are available in the docs/example folder on GitHub.

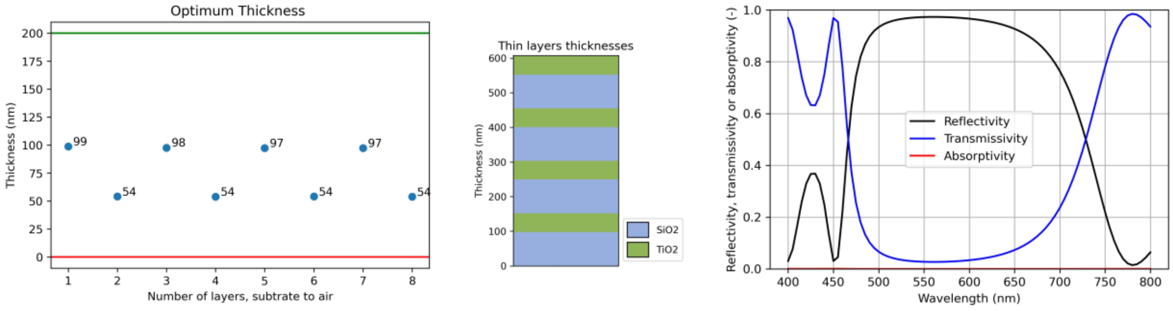


ConsistencyCurve.png *ConvergencePlots2.png*

Figure 39 : Consistency curve of the entire set of 8 runs and convergence curves of the top 6 solutions, in their final order of arrival.

The "*Stacks.txt"* text file contains the thicknesses of all 8 solutions provided by each of the 8 runs. The 8 solutions/stacks are each written on a different line. The most performant stack among the 8 launches, i.e., the solution from our optimization, is depicted in the figure "*Optimum_Thickness_Stack.png*," illustrated in Figure 21.

Optimum_Thickness_Stack.png depicts the thickness of each thin layer (in nanometers) represented by their order in the stack: layer #1 is the layer deposited on the substrate, and layer #8 is the one that terminates the stack, in contact with the air. The red and green lines represent the lower and upper limits specified in the *Th_range* variable, defining the solution space explored by the algorithm. In Figure 21, the thicknesses are periodic, as expected for a Bragg mirror. None of the optimized thicknesses are close to the green or red curves, indicating that the space explored by the algorithm is sufficient. The reflectance and transmittance of the most performant stack are plotted in the two images "*Optimun_Reflectivity.png*" and "*Optimun_Transmissivity.png*" respectively. The picture *OpticalStackResponse.png* here illustrated on the right in Figure 21, describe the full optical behavior of the stack. At the end, *Stack_plot.png* propose a schematic representation of the stack.



Optimun_Thickness_Stack.png *Stack_plot.png* *OpticalStackResponse.png*

Figure 40 : Optimization Result: On the left: stack descriptions from COPS. The central image illustrates the same stack. On the right: reflectivity curve of the best solution.

Example 1b : PV Cell with SolarSpectrum

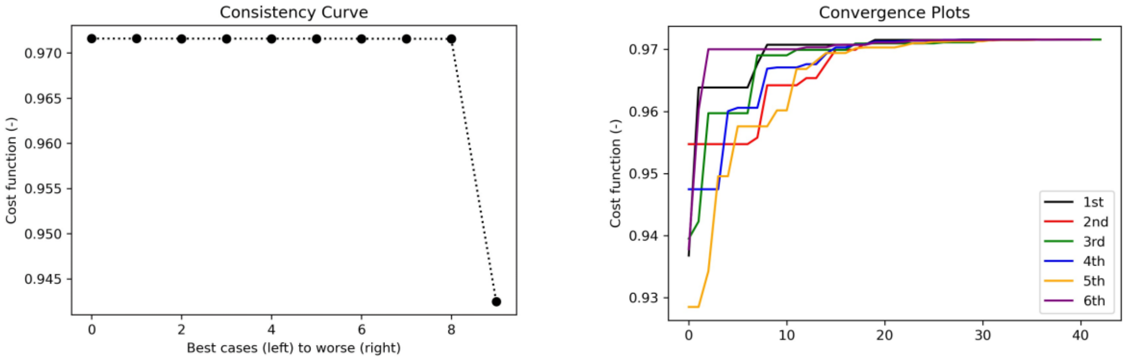
In this second example, we will optimize an antireflective coating for a silicon-based photovoltaic cell. The PV cell will be represented by 1 mm of silicon. As silicon is primarily opaque, the goal of optimizing the antireflective layer will be to maximize absorptance. For this purpose, we will use the cost function "*evaluate_A_pv*" (see paragraph 5.5.7, page 48). This cost function considers a solar spectrum (here ASTM G173-03 GT) and the normalized spectral response of a silicon cell. Here are the key optimization parameters of SolPOC and their justifications. All files created by SolPOC are available in the docs/example folder on GitHub.

Code	Justification
Mat_Stack = ["Si", "TiO2", "ZnO", "Al2O3"]	Description of the stack consisting of 3 thin layers of TiO_2, ZnO, and Al_2O_3.
Wl = np.arange(280, 1505, 5)	Spectral range from 280 nm to 1500 nm to match the efficiency of the PV cell.
Th_range = (0, 200)	The optimization seeks a solution with thin layer thicknesses ranging from 0 to 200 nm.

algo = DEvol
selection = selection_max
cost_function = evaluate_A_pv
	We employ the DEvol optimization algorithm. In line with our objective, the corresponding cost function is "*evaluate_A_pv*," which we aim to maximize using "*selection_max*."

nb_run = 8
cpu_used = 8
	The optimization will be conducted 8 times. To reduce computation time, 8 CPUs are utilized in parallel.

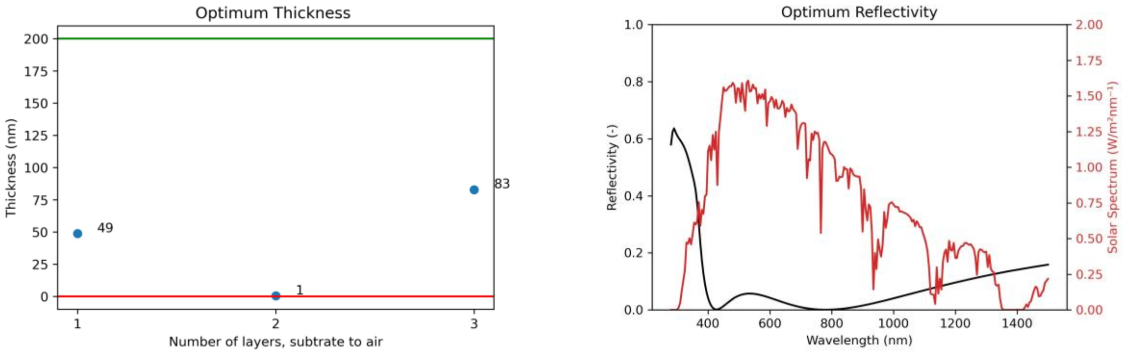
The *ConsistencyCurve.png* graph shows that the problem is well-resolved, though not perfectly. Figure 22 demonstrates that the algorithm successfully rediscovered a stack with a value of 0.9715 according to the cost function in 7 out of 8 runs. The 8th launch proposes a solution with a score of 0.942, likely indicating a local optimum from which the algorithm struggled to escape. This figure highlights the necessity of conducting multiple optimizations to have confidence in the provided solution. The *Convergenceplot.png* graph reveals that the top 6 stacks converge well towards the same extreme value.



ConsistencyCurve.png *ConvergencePlots2.png*

Figure 41 : Illustration of problems resolution by the algorithm. On the left: the Consistency Curve illustrates the 8 solutions ranked from best to worst. On the right: an illustration of the cost function during optimization.

Figure 42 illustrates the main results. The left figure describes the thickness of each thin layer in the stack for the best solution. Note that layer #2, ZnO, has a thickness of 1 nm or less: the algorithm removed it from the stack to ensure the best performance according to the cost function. The right figure (*Optimun_Reflectivity.png*) illustrates the reflectivity spectrum of the best stack, with the solar spectrum. Note it should be interesting to multiply the solar spectrum by the normalized spectral response of the cell, resulting in zero irradiance from 1150 nm onwards.



Optimun_Thickness_Stack.png *Optimun_Reflectivity.png*

Figure 42 : Result of the optimization of an anti-reflective coating for a Si cell. On the left: thickness of the thin layers described in the stack. On the right: reflectivity with the solar spectrum.

Figure 43 give a schematic representation of the stack. Note than the color used are refractive index dependent. As example, low refractive index such as SiO2 are un blue when medium refractive index materials are in green. High refractive index materials are in orange / red. The thicknesses are correct, and we easily see than the ZnO layers has been "removed" from the stack by the optimization method.

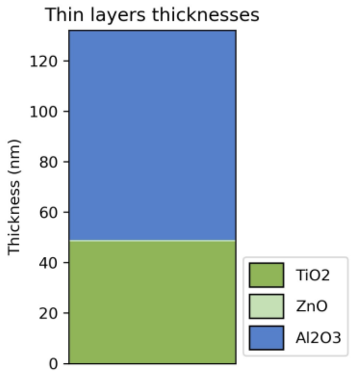


Figure 43 : Schematic representation of the stack with Stack_plot.png.

8.2. Tutorial 2 : Optimize Stack Including Composite Materials

It is possible to optimize a stack using composite materials, representing a mixture of two materials such as cermet (dielectric-metal mix) or porous materials. In SolPOC, composite materials are declared in the stack using a hyphen between the two materials. The Bruggeman's law is used to calculate the effective refractive index of the medium (see paragraph 3.4.1 pages 21), based on the mixture ratio between the two materials, known as the volume fraction. SolPOC will optimize both the thickness of each thin layer and the volume fraction, if necessary. The range of volume fractions explored during optimization is specified in the variable "vf_range" which is an optional variable. The volume fraction is a percentage, so the definition range of "vf_range" is [0 - 1], which can be reduced if needed. It is important to note that optimization with composite materials is slower.

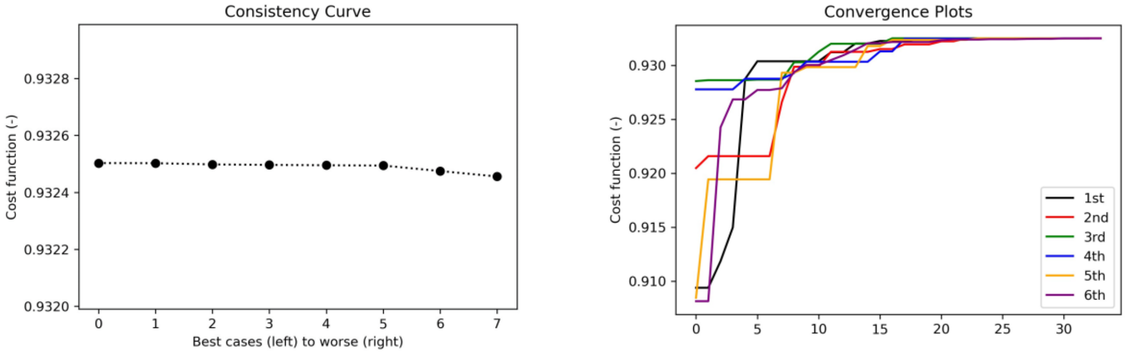
Example : selective coating.

We present an example of optimizing a stack that includes a composite layer, specifically a W-Al_2O_3 cermet. The goal is to optimize a selective coating for a solar thermal collector. In summary, we aim for high solar absorptance in the solar spectrum (280 to 2500 nm) and high reflectance in the infrared domain (2.5 to 30 µm). While such coatings can be achieved in various ways, the use of cermet is common, for instance, in a stack W/W-Al_2O_3/Al_2O_3 deposited on an iron substrate. Here are the main parameters of SolPOC optimization and their justifications. All files created by SolPOC are available in the docs/example folder on GitHub.

Code	Justification
Mat_Stack = ("Fe", "W", "W-Al2O3", "Al2O3")	Thin layers stack : W/W-Al2O3/Al2O3 on a Fe substrat
Wl = Wl_selectif()	Domain specially designed for selective treatments
Th_range = (0, 300)	Optimization uses thin films between 0 and 200 nm
Vf_range = (0, 1.0)	Optimization seeks a percentage of W inclusion in an Al2O3 matrix of between 0 and 100%.

algo = DEvol
selection = selection_max
cost_function = evaluate_rh
	We use the DEvol algo. According to our objective, the corresponding cost function is "evaluate_rh", which we seek to maximize via "selection_max".

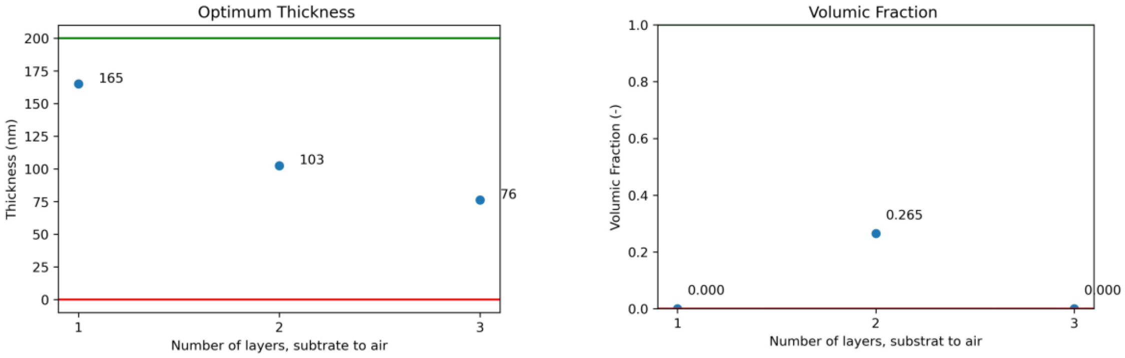
The Consistency Curve graph (available in the folder) demonstrates that the problem is well-solved: the algorithm consistently finds the maximum performance achieved, which is 0.9325.



ConsistencyCurve.png *ConvergencePlots2.png*

Figure 44 : ConsistencyCurve and ConvergencePlots2, for a 3 layers selective coating.

The "*Stacks.txt*" file contains all the results, with the best one illustrated in the "*Optimun_Thickness_Stack.png*" and "*Optimun_VolumicFraction.png*" figures. Note than the figure Optimum_VolumicFraction.png" was no present in the previous example. In the " *Optimun_VolumicFraction.png* " image, all thin layers are represented, even if they consist of a single material. For this reason, the volumic fraction of layer #1 and layer #3 are 0, because the are respectively W and Al_2O in the W/W-Al_2O_3/Al_2O_3 stack. A red line and a green (here not visible) are also added for represent the limit give in the parameter *Vf_range*.

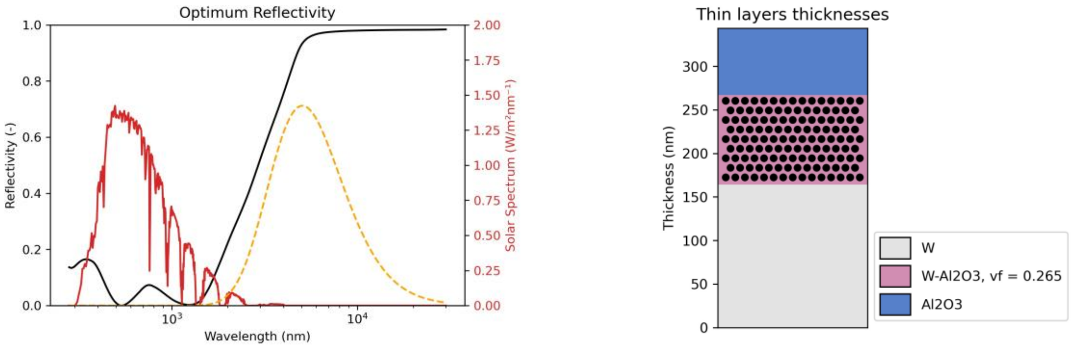


Optimun_Thickness_Stack.png *Optimun_VolumicFraction.png*

Figure 45 : Result of the optimization of a selective stack. On the left: thin layers thicknesses. Ohe right: optimized volumetric fraction for each thin layer.

Figure 46 illustrate the reflectivity of the stack and the schematic illustration of the stack. Please note that the images "*Optimum_Reflectivity.png*" and "S*tack_plot.png*" are adaptive for the selective coating. If the evaluate function is "*evaluate_rh*," the "*Optimum_Reflectivity.png*" includes a blackbody curve (in orange), calculated with the temperature T_{abs}. Note that the blackbody curve is illustrative; the shape is normalized to have the same maximum as the solar spectrum (in red). The units (in $W.m^2.nm^{-1}$) on the y-axis are incorrect, but the x-axis (the wavelength, in nm) is correct. We made this choice because, for high temperatures, the black body is higher than the solar spectrum, creating a graph that is not easy to read. Regarding the

"*Stack_plot.png*," the cermet layers are represented in pink to purple color, depending on the volumetric fraction value.



Optimun_reflectivity.png *Stack_plot.png*

Figure 46 : At left, the reflectivity curve in the Optimiun_reflectivity.png, with a black body. A right: the Stack_plot, with a cermet layer

8.3. Tutorial 3 : Optimize Stack Thicknesses With Theoretical Material

It is possible to optimize a stack using theoretical materials to simultaneously optimize both thickness and refractive index (see section 3.7.3 pages 27). Usually, for optimize only the thickness (refractive index), you need to add a text file describing your material to the *Materials* folder. In this example we go to optimize the thicknesses AND the refractive index of thin layer, by using theoretical materials. Theses thin layers are added on top of the stack declared in the *Mat_Stack* variable.

> Please note it is not yet possible to include them under a thin layer of a conventional material. Figure 47 illustrates the achievable cases, with an "X" symbolizing a thin layer of theoretical materials.

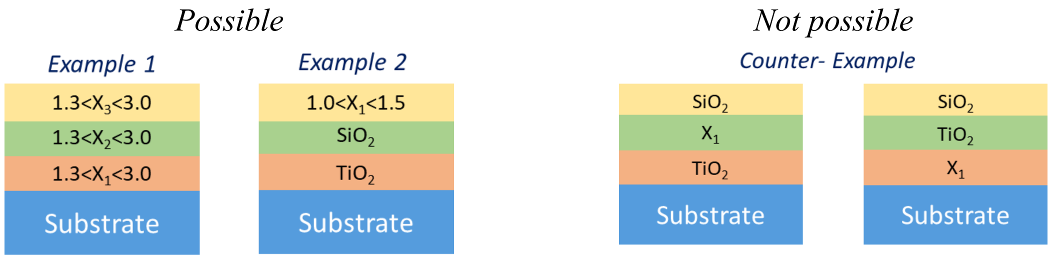


Figure 47 : Example of possible and not possible stack using theoretical material

Here are the variable declarations for this type of stack:

- Example 1 : Mat_Stack = ("BK7") , nb_layer = 3, n_range = (1.3 , 3.0)
- Example 2 : Mat_Stack = ("BK7", "TiO2", "SiO2"), nb_layer = 1, n_range = (1.0 , 1.5)

The number of theoretical thin layers is declared with the variable *nb_layer*. This variable is optional; the code can function correctly if it is not defined. During optimization, the code will optimize the real part of the refractive index between two extremes, defined in the variable *n_range*. Common values are between 1.3 and 3.0. The literature shows that materials with a refractive index lower than 1.3 and higher than 3.0 are rare. The lower range can be adjusted towards 1.0 (close to the refractive index of air) by using porous materials.

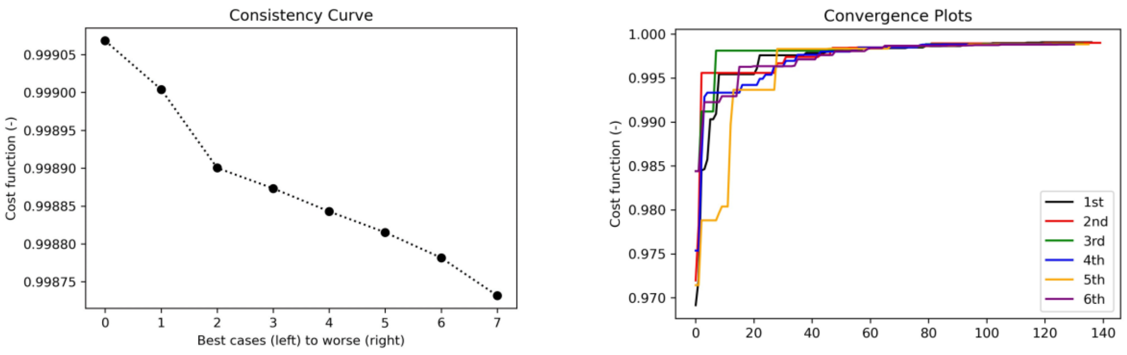
Example : Search for refractive indices for an antireflection coating

The goal is to optimize a three-layer antireflection coating for the human eye deposited on glass (BK7), searching for the thicknesses and materials to be deposited on the substrate. In this case, we are uncertain about the thicknesses and refractive indices of the materials to be used and the order in which they are deposited in the stack. We will use this functionality in SolPOC Here are the main optimization parameters and their justifications. All files created by SolPOC are available in the docs/example folder on GitHub.

Code	Justification
Mat_Stack = ["BK7"]	Substrate: BK7 glass (n=1.42)
Wl = np.arange(300 , 805, 5)	Wavelength range: 300 to 800 nm to encompass the human eye's sensitivity domain.
nb_layer = 3	Three theoretical thin layers are added on the substrate.
Th_range= (0, 200)	Optimization involves thin layers with thicknesses ranging from 0 to 200 nm.
n_range= (1.442 , 2.42)	Theoretical thin layers have refractive indices (n) between 1.442 (MgF2 index at 587 nm) and 2.42 (TiO2 index at 587 nm).

algo = DEvol
selection = selection_max
evaluate = evaluate_T_vis
	DEvol algorithm is used for optimization. The corresponding cost function is "evaluate_T_vis", and we aim to maximize it using the "selection_max" callable.

Figure 48 show the optimization quality for particular problem, with two variables per thin layers: the thicknesses and the refractive index (assumed constant d*n*/dλ = 0 and with k = 0 over the wavelengths). After launching the optimization, the results are automatically saved in a folder created by the code. The *ConsistencyCurve.png* graph shows that the problem is solved: the algorithm consistently achieves high performance (0.99905), and the lowest point has a performance close to the maximum (0.99875. However, it's worth noting that two points (representing two runs with different initializations / starting point) do not have the exact same value. Therefore, while the problem is well solved, the optimum value cannot be considered a "global optimum." This is observed despite each optimization having converged, as shown in *ConvergencePlots2.png*.



ConsistencyCurve.png *ConvergencePlots2.png*

Figure 48 : ConsistencyCurve.png and ConvergencePlots2.png for this example cases

Figure 30 depicts the results. As usual, the *Optimun_Thickness_Stack.pn*g figure displays the thicknesses of the best run in nanometers, here the only point with a performance of 0.99905. For this same stack, the *Optimun_RefractiveIndex_Stack.png* figure shows the optimized refractive indices, for each layer. The red and green lines illustrate the lower and upper bounds of the possibilities explored by the stack. We conclude that the best stack with three thin layers for our problem consists of the following thicknesses: 1 mm of BK7 / 54 nm, with *n*=1.86 / 89 nm with *n*=2.33 / 89 nm with *n*=1.44. A literature search shows that the real materials that would correspond most closely are Al_2O_3 (n=1.67 at 587 nm) MgF_2, and TiO2.

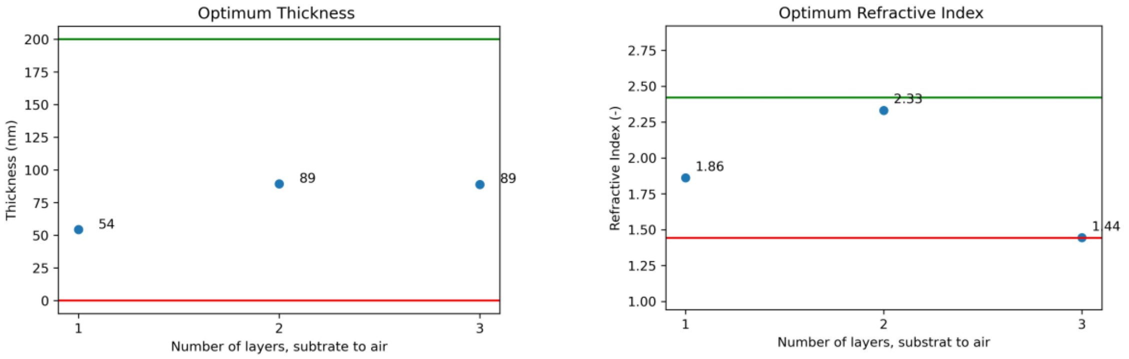


Figure 49 : On the left, the thickness of each thin layer. On the right, the refractive index of each thin layer.

To conclude this example, Figure 50 shows the schematic representation of the stack proposed by SolPOC with the figure *Stack_plot.png*. The refractive indices of each thin layer are indicated, and the color of each thin layer is a function of the refractive index.

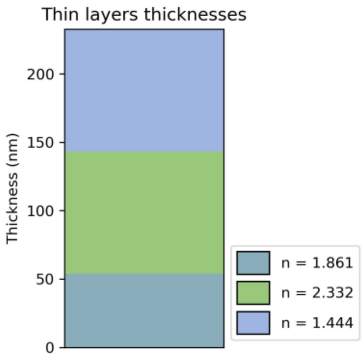


Figure 50 : On the left, the thickness of each thin layer. On the right, the refractive index of each thin layer.

8.4. Tutorial 4 : Optimize Stack Thicknesses with a Thickness Fixed

Here's an example of how to fix the thickness of one or more thin layers in the case of optimization. You need to write a number in the variable *d_Stack_Opt*, which is a list. Each element of the list with index i corresponds to the thin layer with index i. Here's an example for a silver layer and a SiO2 layer deposited on a glass substrate. We want to fix the thickness of the silver layer at 6 nm. The thickness of the SiO2 layer is free and will be optimized.

```
Mat_Stack = ("BK7", "Ag", "SiO2")
d_Stack_Opt =  [6, "no"]
```

The variable *d_Stack_Opt* is optional. If it is not declared or if the list is empty, the code considers that all thicknesses should be optimized. For thin layers that need to be optimized, we recommend writing a string (the code optimizes all thin layers that are not written with an int or a float).

This option is available only in two optimization methods: *DEvol* and *optimise_ga.*

Example : low_e glasses

We present an example of optimization with a low-e coating for a building glazing. Here, we aim to replicate a result presented in a study. The goal of a low-e solar glazing is to be transparent in the visible part of the solar spectrum (up to 800 nm, to benefit from natural light) and then reflective in the IR part of the solar spectrum (800 - 2500 nm) to limit thermal losses. In their studies, M. Sebastiani et al propose the typical stack of a silver-based low-emissivity (low-E) [32]. Their studies specify that the two ZnO layers serve an adhesion purpose, and the silver layer is approximately 10 nm thick. Figure 27 illustrates the stack of thin layers.

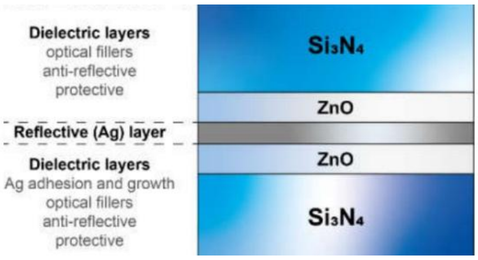


Figure 51 : Low-E glass stack with a thin layer of silver [32]

Here are the main optimization parameters of SolPOC and their justifications. All files created by SolPOC are present in the docs/example folder. To set the thickness of the thin silver layer, we declare the variable *d_Stack_Opt* and write a number at index #2 in the d_*Stack_Opt* variable. We recall, then the first value of the list is index #0. Secondly, the first value of *d_Stack_Opt* corresponds to the first thin layer of the stack, here Si_3N_4.

Code	Justification
Mat_Stack = ["BK7","Si3N4, "ZnO", "Ag", "ZnO", "Si3N4"]	Substrate: BK7 glass (n=1.42)
Wl = np.arange(280 , 1505, 5)	Wavelength range: 280 to 1500 nm
Th_range = (0, 200)	Optimization using thin films ranging from 0 to 200 nm
Lambda_cut_1 = 800 # nm	Cutoff wavelength for the cost function *evaluate_low_e is* 800 nm
d_Stack_Opt = ["no", "no", 10, "no", "no"]	The 3rd layer in the stack, here the silver, has a fixed thickness of 10 nm.

algo = DEvol
selection = selection_max
evaluate = evaluate_low_e
	DEvol algorithm is used. According to our objective, the corresponding cost function is "evaluate_T_vis". We aim to maximize using the selection_max callable

After launching the optimization, the results are automatically saved in a folder created by the code. We compare the result by fixing the thickness of silver with a conventional optimization where all thicknesses are optimized. Figure 52 illustrates the optimized thicknesses of each thin layers, with a fixed value at 10 nm for the silver layer (left) and a free value of silver (right). We notice that a silver thickness of 10 nm is the right order of magnitude.

Without fixing the silver thickness, the code found a value of 11 nm (right Figure). This thin metallic layer, in a stack that must transmit light from 280 to 800 nm, has a strong influence on the layers behind it (the layer #1 of Si_3N_4 and layer #2 of ZnO). In both cases, the thicknesses of the second ZnO layers (index #4) should have no impact on the stack optical performances. In both first case (left) the algorithm wants to remove this layer, by choosing a value of 3 or 0 nm. The thicknesses of thin layer #1 (Si3N4) and thin layer#2 (ZnO) are different in the two cases. But we can remark than the sum of the two thicknesses is similar 149 nm + 47 nm = 196 nm for the left case and 82 nm + 115 nm = 197 nm in the right case.

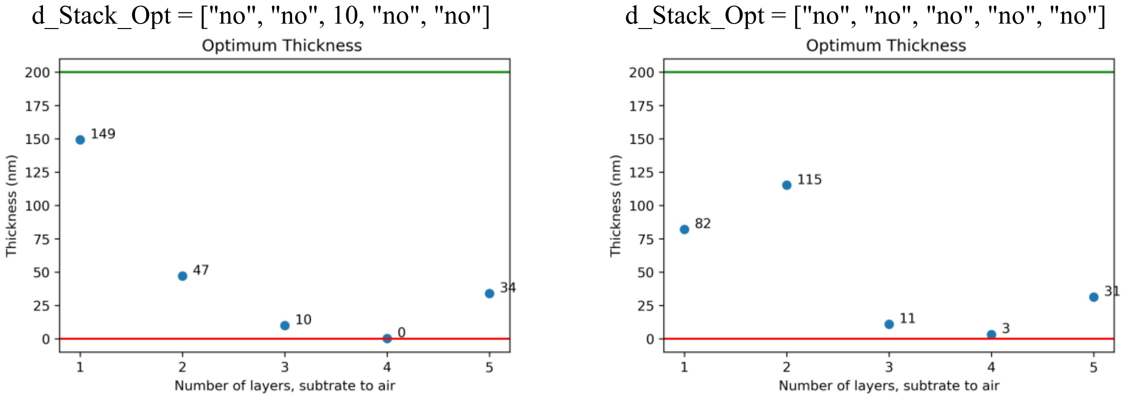
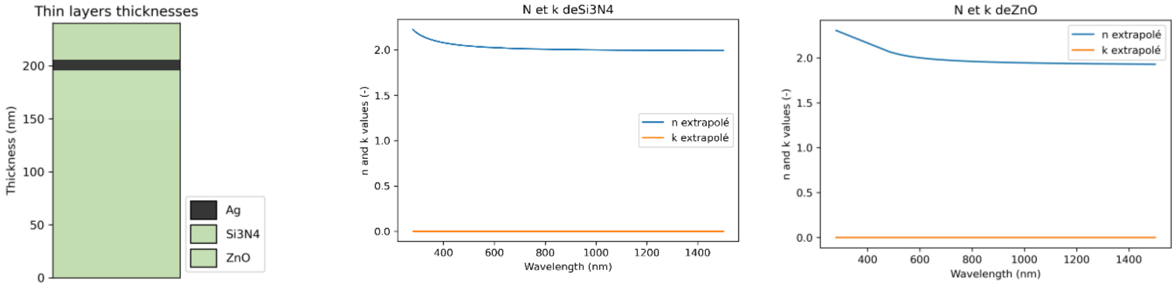


Figure 52 : Thicknesses of thin layers in a low-E glass. On the left: the thickness of silver is 10 nm (layer #3). On the right, all thicknesses are optimized.

The explanation for the constant sum of thicknesses for layers 1 and 2 can be easily found in the additional graphs provided by SolPOC. For example, the colors (light green) assigned to Si_3N_4 and ZnO in *Stack_plot.png* suggest that their refractive indices are close. This assumption can be easily confirmed by checking the refractive index graphs that SolPOC generates for each material used in the stack. Here it's, *refractive_indexZnO.png* and *refractive_indexSi3N4.png* and/or by examining the values in the corresponding text files (*ZnO.txt* and *Si3N4.txt*). It becomes evident that the refractive indices of ZnO and Si3N4 are very close, leading to the observed results during the optimization process. If the ZnO layers serve an adhesion purpose, they have the value than the Si_3N_4 layers for SolPOC, as they have a very similar refractive index.



Stack_plot.png *refractive_indexSi3N4.png* *refractive_indexZnO.png*

Figure 53 : Additional figures created by SolPOC as Stack_plot and refractive_index can help to clearly understand the proposed results.

The two *ConsistencyCurve.png* figures for a fixed thickness of silver (Figure 54 on the left) and a free thickness of silver (Figure 54 on the right) are shown. In this specific case (which should not be generalized), it is observed that fixing the thickness of silver reduces the gap between the best and least performing solutions. The consistency curve ranges from 0.77775 to 0.77755 with a fixed silver thickness of 10 nm. If the silver thickness is free (Figure 31 on the right), the algorithm needs to find a value close to 10 nm in addition to optimizing the other thin layers. However, with a sufficient budget (here, 7500 evaluations), the algorithm manages to optimize the thickness of the silver layer to exactly 11.3 nm. This allows 4 out of 8 runs to achieve higher performance (0.77775 vs. 0.78346) compared to the previous case.

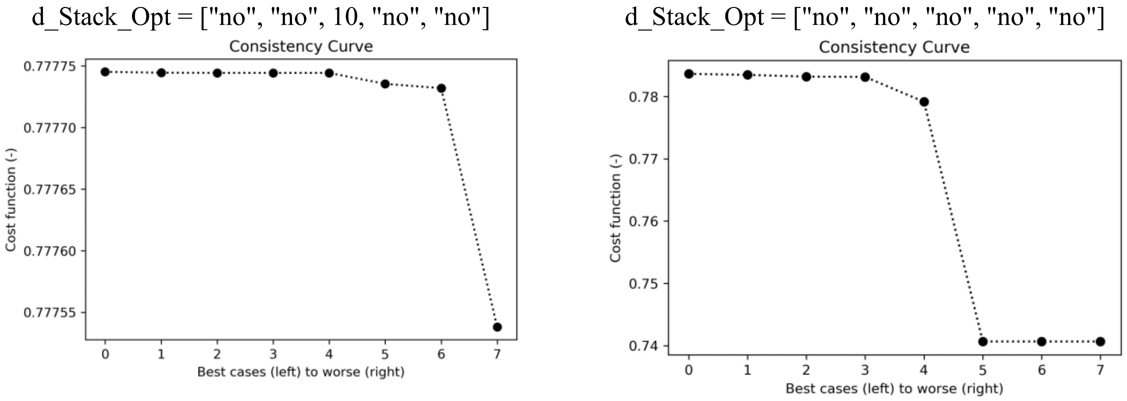


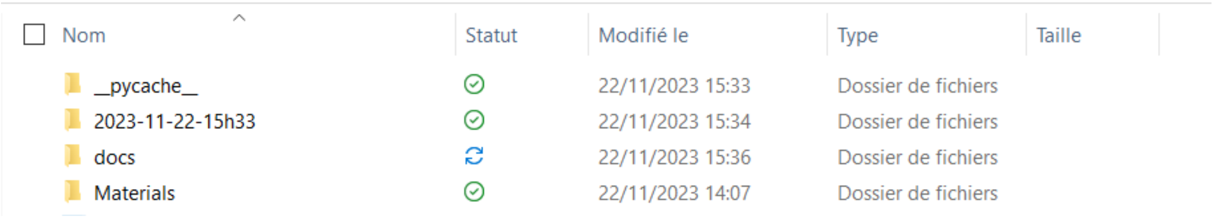
Figure 54 : Thin film thicknesses for a low-e glass. On the left: the thickness of silver is 10 nm (layer #3). On the right, all thicknesses are optimized.

IX. Output Folder

To make it easier to use SolPOC and avoid having to manually type the same commands to save data, we've given the code the ability to save the main results. We describe here the various files automatically created by SolPOC at the end of its execution. These files are purely informative, and it's relatively easy to create your own.

9.1. Folder With the Saved Results

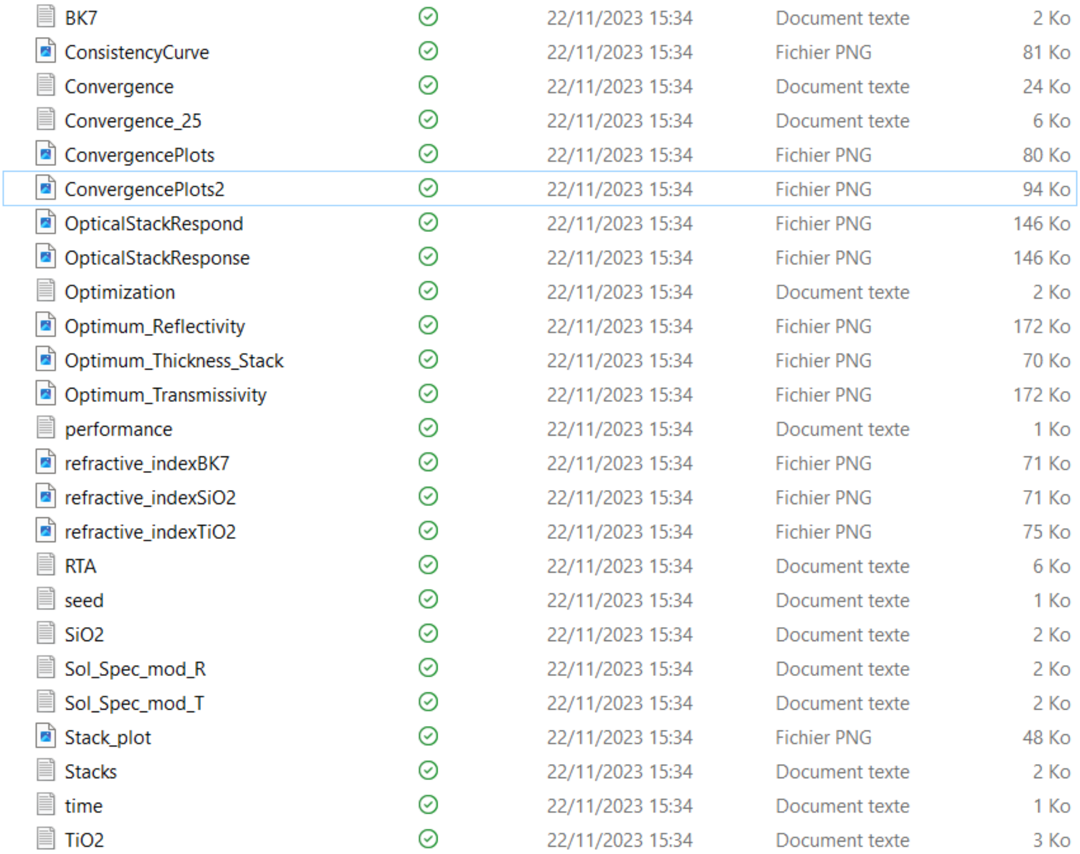
When the code is launched, SolPOC automatically creates a folder, named according to the launch date and time. The formalism is: "YYYY-MM-DD-HHhMM", as in the following example: "2023-11-22-15h33". The date and time are those of the computer clock, obtained via the datetime library. An example is shown in Figure 55.



□ Nom	Statut	Modifié le	Type	Taille
__pycache__	✓	22/11/2023 15:33	Dossier de fichiers	
2023-11-22-15h33	✓	22/11/2023 15:34	Dossier de fichiers	
docs	⟳	22/11/2023 15:36	Dossier de fichiers	
Materials	✓	22/11/2023 14:07	Dossier de fichiers	

Figure 55 : Example of folder named "2023-11-22-15h33"

The folder is used to store various backup files proposed by code users. You can easily add or remove information from this folder to suit your needs.



BK7	✓	22/11/2023 15:34	Document texte	2 Ko
ConsistencyCurve	✓	22/11/2023 15:34	Fichier PNG	81 Ko
Convergence	✓	22/11/2023 15:34	Document texte	24 Ko
Convergence_25	✓	22/11/2023 15:34	Document texte	6 Ko
ConvergencePlots	✓	22/11/2023 15:34	Fichier PNG	80 Ko
ConvergencePlots2	✓	22/11/2023 15:34	Fichier PNG	94 Ko
OpticalStackRespond	✓	22/11/2023 15:34	Fichier PNG	146 Ko
OpticalStackResponse	✓	22/11/2023 15:34	Fichier PNG	146 Ko
Optimization	✓	22/11/2023 15:34	Document texte	2 Ko
Optimum_Reflectivity	✓	22/11/2023 15:34	Fichier PNG	172 Ko
Optimum_Thickness_Stack	✓	22/11/2023 15:34	Fichier PNG	70 Ko
Optimum_Transmissivity	✓	22/11/2023 15:34	Fichier PNG	172 Ko
performance	✓	22/11/2023 15:34	Document texte	1 Ko
refractive_indexBK7	✓	22/11/2023 15:34	Fichier PNG	71 Ko
refractive_indexSiO2	✓	22/11/2023 15:34	Fichier PNG	71 Ko
refractive_indexTiO2	✓	22/11/2023 15:34	Fichier PNG	75 Ko
RTA	✓	22/11/2023 15:34	Document texte	6 Ko
seed	✓	22/11/2023 15:34	Document texte	1 Ko
SiO2	✓	22/11/2023 15:34	Document texte	2 Ko
Sol_Spec_mod_R	✓	22/11/2023 15:34	Document texte	2 Ko
Sol_Spec_mod_T	✓	22/11/2023 15:34	Document texte	2 Ko
Stack_plot	✓	22/11/2023 15:34	Fichier PNG	48 Ko
Stacks	✓	22/11/2023 15:34	Document texte	2 Ko
time	✓	22/11/2023 15:34	Document texte	1 Ko
TiO2	✓	22/11/2023 15:34	Document texte	3 Ko

Figure 56 : Example of the different results files present in folders created by SolPOC.

Descriptions of files in the folder are as follows, with additional details provided in the dedicated paragraph:

- *ConsistencyCurve.png.* The pictures *ConsistencyCurve.png* image illustrates the cost function (performance) on the y-axis for all launches, arranged in descending order on the x-axis. The best stack is on the left, and the worst on the right, offering an overall view of optimization quality.
- *Convergence.txt & convergence_25.txt.* These text files contain the cost function values for each launch during the optimization process, organized as arrays. Rows correspond to different launches, and columns to the values of the function throughout the process. The initial cost function value is present at the end of the list.
- *OpticalStackResponse.* The image de depicts the reflectivity, the transmissivity and the absorptivity of the best stack, combining all launches. The x-axis corresponds to the wavelengths used, and the default solar spectrum is shown on the second y-axis.
- *Optimization*: The solution text file contains all the information necessary to reproduce the simulation or optimization, encompassing variable values, stacking materials, the optimization function used, etc.
- *Optimun_Reflectivity.png*. This image displays the reflectivity of the best stack, combining all launches. The x-axis corresponds to the wavelengths used. The solar spectrum used is also presented on the image, on the second y-axis.

- *Optimun_Transmissivity.png*. The image depicts the transmissivity of the best stack, combining all launches. The x-axis corresponds to the wavelengths used, and the default solar spectrum is shown on the second y-axis.
- *performance.txt*: The performance file contains the value of the cost function for each launch.
- *RTA.txt*: This text file includes the reflectivity (column no. 1), transmissivity (column no. 2), and absorptivity (column no. 3) of the best stack, combining all launches, across wavelengths (column no. 0) in nm.
- *Seed.txt*: The seed text file includes the seed initiated at the start of each optimization algorithm, if it utilizes a random number generator (see the seed paragraph).
- *Stacks.txt*: Each line in the stack file contains a description of the stack, with thicknesses shown in nanometers. Noted the stack can include the volumic fraction or the refractive index data of the ideal materials.
- *Stack_plot.txt*. This picture is a schematic presentation of the stack.
- *Thickness*: The thickness image visually represents the thickness of each thin layer (in nanometers) in their order within the stack, with the first layer being the one deposited on the substrate and the last layer ending the stack, in contact with the air. Red and green lines represent the upper and lower limits specified in the variable *Plage_ep*.
- *Time.txt* : The time file contains the time, in seconds, taken by each core for each solution.

It is important to note that in the files containing all the results, the values are **not sorted in ascending or descending order**. They are written in the same order in all the other files. The writing order directly corresponds to the order of variables in SolPOC**.**

For example, the first line of the *performance.txt* file, the *convergence_dev.txt* file, the *seed.txt* file, and the *stacks.txt* file all represent the same stack.

9.2. ConsistencyCurve.png

9.2.1. The need of the Consistency Curve

The "*ConsistencyCurve.png"* figure represents the performance (according to a cost function) of multiple runs of the same optimization that have converged. Each point represents the result of a complete optimization (the value of each optimization can be founded in *performance.txt* text files). All results are then sorted in ascending order of performance to assess whether the optimization algorithm consistently finds similar solutions. The need for the Consistency Curve arises from :

1. the use of non-deterministic algorithms
2. the use of cost functions rich in local minima.

In the optimization of stacks with numerous thin layers, each execution provides a potentially different solution, even if convergence is achieved for each run. In optical design optimization, the convergence of an optimization algorithm does not guarantee the quality of the solution. A solution is deemed acceptable only if multiple runs converge to the same result. This allows us to conclude that this solution was not reached by chance, increasing confidence in identifying the global optimum. The purpose of the *ConsistencyCurve.png* is to provide a visual aid for this analysis.

9.2.2. Improve the Consistency Curve

To improve the appearance of the Consistency Curve, the first method to consider is increasing the optimization time, which mean increase the number of iterations. The method for increasing the optimization time varies for each algorithm. Taking *DEvol* as an example: for this algorithm, it is necessary to increase the number of generations, which, in turn, increases the budget. For recall, in SolPOC:

$$\boxed{\text{Budget} = pop\_size \cdot nb\_generation}$$

Figure 57 illustrates different Consistency Curves for an SiO_2/TiO_2 Bragg mirror with 8 periods (maximization of the cost function *evaluate_R_Brg*) under different budgets, which are proportional to the computation time. It can be observed that increasing the budget improves the consistency curve and thus the quality of the response. With sufficient computation time, the algorithm consistently finds the optimum 10 times (black curve, budget of 9000). It is also noticeable that for the cyan curve (budget: 6000), the algorithm identified a low-quality value once, highlighted in black. It may have been trapped in a local optimum, while launches for 3000 and 4500 luckily avoided it. This risk is inherent in optimization methods, and that's why we recommend running multiple instances, and accordingly, SolPOC is coded to take advantage of multiprocessing. In any case, this value can be ignored because the other 9 runs at the same budget correctly identified the global optimum.

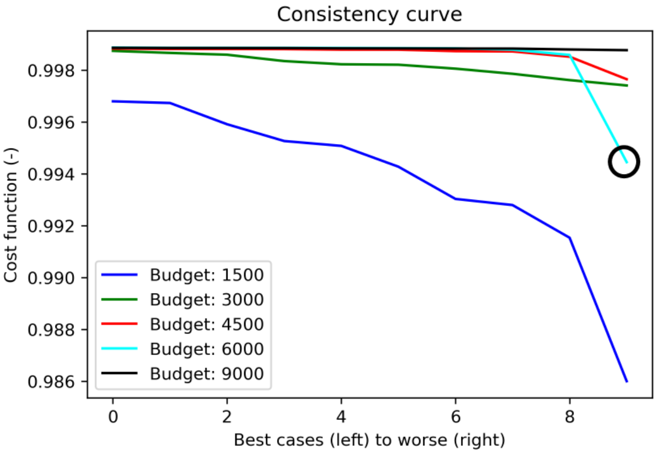


Figure 57 : Consistency Curve for a Bragg mirror SiO2/TiO2 for 8 periods, for different budget, using DE

There are other methods to improve the appearance of the Consistency Curve. Here are some suggestions:

- Modify Optimization Algorithm Hyperparameters: Adjusting the hyperparameters of optimization algorithms can impact their convergence behavior. Experimenting with parameters like mutation rates, crossover rates, and population sizes (for genetic algorithms) can be considered.

- Increase the Number of Runs: While this may not necessarily improve the appearance, increasing the number of runs can enhance the chances of consistently finding the extremum. Running the optimization algorithm multiple times provides a better understanding of the solution space.
- Change Optimization Method/Algorithm: Different optimization algorithms have different strengths and weaknesses. Trying alternative optimization methods or algorithms might lead to better convergence behavior or the identification of different solutions.

In conclusion, it's crucial to question whether the goal is to identify a global optimum or if a locally optimal solution is satisfactory. In the case of complex thin-film stack optimizations for deposition purposes, it may not always be necessary (in our opinion) to focus on achieving excessively high-quality optimization. Balancing computational cost and solution quality is often a practical approach in real-world applications.

9.3. Convergence

To represent the quality of the optimization, *ConvergencePlots.png* and *ConvergencePlots2.png* graphs depict the value of a cost function during the optimization progresses (the values used for theses graph are in the *Convergence.txt* and *Convergence_25.txt* files. This ensures that the optimization has converged to a solution, indicating that it was pursued for a sufficient duration. For clarity, the evolution of the performance of the top 3 stacks is shown for *ConvergencePlots.png* and the top 6 for *ConvergencePlots2.png*. Figure 40 illustrates both graphs for the "Example 1b : PV Cell with SolarSpectrum". It is observed that :

1. Each curve ends with a plateau, indicating that the algorithm has converged.
2. The 6 curves converge to the same point, indicating that the top 6 optimizations find the same result.

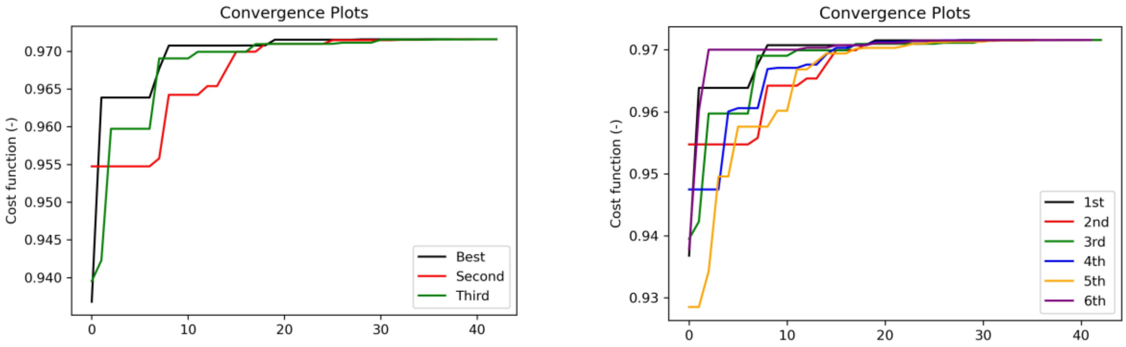


Figure 58 : Convergence plots from « Example 1b : PV Cell with SolarSpectrum »

All the values are written in the text files *Convergence.txt* and with *Convergence.txt* (at 25 equidistant points), and in the text file dev.txt (for all the data). For these Convergence files, the values of the same run are on the same line. The cost function value at the end of the optimization is written in the first column. The stacks are not sorted in ascending order: the first row corresponds to the first solution returned by the code, not the best stack of all runs.

Note: The file dev.txt contains the cost function values in the case of a problem where we seek to minimize (*selection* = selection_min). In the case where we seek to maximize (*selection* = selection_max), it contains 1 minus the value of the cost function.

9.4. Refractive_index.png

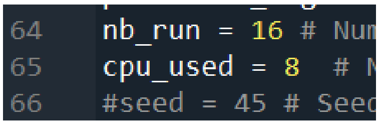
On Figure 56, we can remark several files as: *refractive_indexBK7.png*, *refractive_indexSiO2.png*, *refractive_indexTiO2.png* and *BK7.txt*, *SiO2.txt* and *TiO2.txt*. These files are refractive indices of materials used in simulation. Is the example of Figure 56 we have 3 different materials (SiO_2, BK7 and TiO_2). The PNG files are graphically representation of the refractive index, after linear extrapolation across wavelengths. The text file are the raw data corresponding to the PNG Figures. These data can be pertinent, for instance, when the range of calculation wavelengths differs from the data obtained from measurements. The provided examples mention refractive indices for specific materials within certain wavelength ranges, but it might be necessary to work in a broader spectral range. For instance:

Example: The refractive indices of TiO2 in the study by S.V. Zhukosky are provided from 211 nm to 1690 nm. However, it may be necessary to work over the entire solar spectrum, for example, from 280 nm to 2500 nm [33].

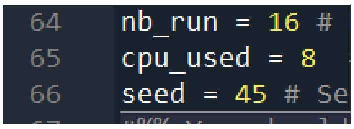
Example: The refractive indices of SiO2 in the study by F. Lemarchand are provided from 250 to 2500 nm. However, it may be necessary to work in the infrared range, for example, from 280 nm to 30 µm [34].

9.5. Seed.txt

In the context of random number generation, a seed is a starting point used by the underlying random number generator algorithm. For random number SolPOC use the **NumPy package**. Numpy uses various random number generator algorithms to generate random numbers. These algorithms always take value as input to initialize their internal state. If the value is not provided by the user, several methods are implemented in NumPy, like read the clock hour or use OS-specific randomness source. In SolPOC, most evaluate function return the seed value used. It's allowing you to reproduce the same sequence of random numbers every time you run the code, which can be helpful for debugging, testing, and ensuring result reproducibility. To fix the seed value, uncomment the line below '*cpu_used*' in the main script.



```
nb_run = 16 # Nu
cpu_used = 8   # N
#seed = 45 # Seed
```



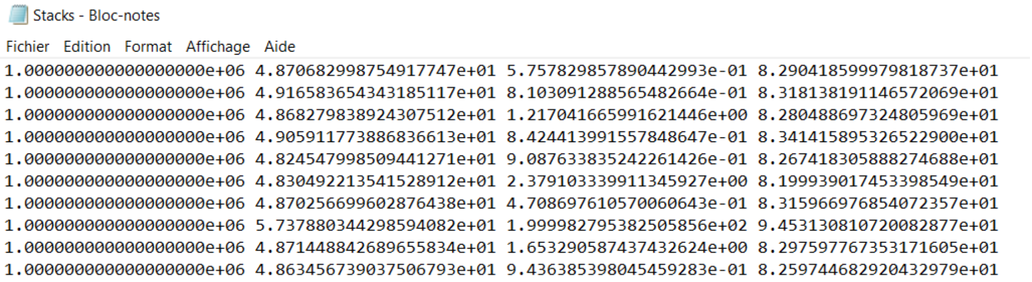
```
nb_run = 16 #
cpu_used = 8
seed = 45 # Se
```

Figure 59 : To left, seed unfixed, each run is different. To right : seed fixe, each run are strongly identical.

By assigning a specific value to the seed yourself, you guarantee the reproducibility of your executions, even when multiprocessing is enabled. The seed value you define acts as the seed for a number generator integrated into the main code. This will generate a series of random numbers which will then be given one by one as an additional argument to the multiprocessing function: each launch of the function therefore receives a different number. Each time the function is run, the unique number given as an argument serves as the corresponding seed, producing a separate series of random numbers for each optimization process. For example, if you run the function with 8 cores in parallel, you'll get 8 different results. However, a subsequent run with the same seed will return the same 8 results, to ensure constant reproducibility.

9.6. Stacks.txt

The "Stacks.txt" text file contains the solutions for each optimization run. In SolPOC, a solution refers to a stack of thin layers, described by their thicknesses and potentially the volumetric fraction or refractive index. The file contains one line per solution, meaning one line per optimization run. Each solution is written on a single line, where spaces represent different thin layers, and the values are written in nanometers. The example in Figure 60 from Example 1b : PV Cell with SolarSpectrum. This example consists of 3 thin layers on a 1 mm stack, with 8 runs, resulting in 8 solutions. For an illustration and an example of formatting, Table 8 reprises the results in a tabular format.



```
1.000000000000000000e+06 4.870682998754917747e+01 5.757829857890442993e-01 8.290418599979818737e+01
1.000000000000000000e+06 4.916583654343185117e+01 8.103091288565482664e-01 8.318138191146572069e+01
1.000000000000000000e+06 4.868279838924307512e+01 1.217041665991621446e+00 8.280488697324805969e+01
1.000000000000000000e+06 4.905911773886836613e+01 8.424413991557848647e-01 8.341415895326522900e+01
1.000000000000000000e+06 4.824547998509441271e+01 9.087633835242261426e-01 8.267418305888274688e+01
1.000000000000000000e+06 4.830492213541528912e+01 2.379103339911345927e+00 8.199939017453398549e+01
1.000000000000000000e+06 4.870256699602876438e+01 4.708697610570060643e-01 8.315966976854072357e+01
1.000000000000000000e+06 5.737880344298594082e+01 1.999982795382505856e+02 9.453130810720082877e+01
1.000000000000000000e+06 4.871448842689655834e+01 1.653290587437432624e+00 8.297597767353171605e+01
1.000000000000000000e+06 4.863456739037506793e+01 9.436385398045459283e-01 8.259744682920432979e+01
```

Figure 60 : Example of Stack.txt files, from « Example 1b : PV Cell with SolarSpectrum »

	Substrate	Layer #1	Layer #2	Layer #3
Stack n°1	1000000 nm	49 nm	1 nm	83 nm
Stack n°2	1000000 nm	49 nm	1 nm	83 nm
Stack n°3	1000000 nm	49 nm	1 nm	83 nm
Stack n°4	1000000 nm	49 nm	1 nm	83 nm
Stack n°5	1000000 nm	48 nm	1 nm	83 nm
Stack n°6	1000000 nm	48 nm	2 nm	82 nm
Stack n°7	1000000 nm	49 nm	0 nm	83 nm
Stack n°8	1000000 nm	57 nm	200 nm	95 nm
Stack n°9	1000000 nm	49 nm	2 nm	83 nm
Stack n°10	1000000 nm	49 nm	1 nm	83 nm

Table 9 : Data from Figure 60

9.7. Performance.txt

The *performance.txt* text file contains the cost function values at the end of the optimization for each of the launches. As a reminder, the values are not written in ascending order. The line numbers correspond between the other text files. For example, the stack thicknesses on line No. 3 described in *Stacks.txt* has a performance value written on line #3 of the *performance.txt* file, with a computation time present on line # of the *time.txt* file.

The highest or lowest value in the performance.txt file represents the best solution among all our launches.

9.8. Optimization.txt

The Optimization.txt text file represents a summary of all the necessary information for the optimization. It is a summary of the .py script used to launch the optimization process. It normally contains all the information needed to recreate the stack, the materials, the name of the optimization algorithm, the cost function, and all the necessary parameters, such as the wavelength range, angle of incidence, etc. This file helps avoid the need to take manual notes or keep a large number of Python scripts to retain information about resolved optimizations. All the main parameters and variables are written, although they may not always be relevant to the cost function used. We hope it proves useful to you. Feel free to modify it as needed.

Conclusion

The deployment of renewable energies such as photovoltaics or solar thermal systems requires innovative and highly efficiency surface coatings to efficiently harness and convert the abundant energy from the sun. Industry and academics need a free, easily tunable, and efficient code for modeled and optimized thin layer stack for solar energy. After several years of development and internal use in PROMES-CNRS laboratories, we propose to make SolPOC (Solar Performances Optimization Code) freely available to the community.

SolPOC is a Python package designed to solve Maxwell's equations in a multilayered thin film structure. The code is specifically designed for research in coatings, thin film deposition, and materials research for solar energy applications (thermal and PV). The code uses a stable method to quickly calculate reflectivity, transmissivity, and absorptivity from a stack of thin films over a full solar spectrum. SolPOC comes with several optimization methods, a multiprocessing pool, and a comprehensive database of refractive indices for real materials. In the end, SolPOC is simple to use for no-coder users thanks to main script, which regroup all necessary variables and automatically save important results in text files and PNG images.

The package relies on proven optical theories with different evolutionary optimization algorithms and cost function specially designed for solar energy utilization. We want to emphasize the critical importance of open science, advocating for the transparent sharing of codes and methods, as no similar code exists. This approach stands as the most effective means to surmount the inevitable challenges encountered in the development of advanced coatings for solar energy systems.

Even if SolPOC is quite simple code, this is a research-grade program. We actually do research with it. Do not hesitate to contact us, for help, academic project or cited to current version of our work.

Bibliographie

[1] A. Grosjean, Etude, modélisation et optimisation de surfaces fonctionnelles pour les collecteurs solaires thermiques à concentration, 2018. http://www.theses.fr/2018PERP0002/document.

[2] A. Grosjean, A. Soum-Glaude, L. Thomas, Replacing silver by aluminum in solar mirrors by improving solar reflectance with dielectric top layers, Sustainable Materials and Technologies 29 (2021) e00307. https://doi.org/10.1016/J.SUSMAT.2021.E00307.

[3] A. Grosjean, A. Soum-Glaude, L. Thomas, Influence of operating conditions on the optical optimization of solar selective absorber coatings, Solar Energy Materials and Solar Cells 230 (2021) 111280. https://doi.org/10.1016/J.SOLMAT.2021.111280.

[4] A. Grosjean, A. Soum-Glaude, P. Neveu, L. Thomas, Comprehensive simulation and optimization of porous SiO2 antireflective coating to improve glass solar transmittance for solar energy applications, Solar Energy Materials and Solar Cells 182 (2018) 166–177. https://doi.org/10.1016/J.SOLMAT.2018.03.040.

[5] Moreau Antoine, Bennet Pauline, Langevin Denis, Wiecha Peter, PyMoosh, (2023). https://github.com/AnMoreau/PyMoosh (accessed September 12, 2023).

[6] Flamant Gilles, Matériaux pour le solaire à concentration, in: Le Solaire à Concentration, ISTE, 2021.

[7] M. N. Polyanskiy, Refractive index database, (2023). https://refractiveindex.info (accessed September 11, 2023).

[8] D.A.G. Bruggeman, Berechnung verschiedener physikalischer Konstanten von heterogenen Substanzen. I. Dielektrizitätskonstanten und Leitfähigkeiten der Mischkörper aus isotropen Substanzen, Ann Phys 416 (1935) 636–664. https://doi.org/https://doi.org/10.1002/andp.19354160705.

[9] D. Ngoue, A. Grosjean, L. Di Giacomo, S. Quoizola, A. Soum-Glaude, L. Thomas, Y. Lalau, R. Reoyo-Prats, B. Claudet, O. Faugeroux, C. Leray, A. Toutant, J.-Y. Peroy, A. Ferrière, G. Olalde, 3 - Ceramics for concentrated solar power (CSP): From thermophysical properties to solar absorbers, in: O. Guillon (Ed.), Advanced Ceramics for Energy Conversion and Storage, Elsevier, 2020: pp. 89–127. https://doi.org/https://doi.org/10.1016/B978-0-08-102726-4.00003-X.

[10] A. Grosjean, E. Le Baron, Longtime solar performance estimations of low-E glass depending on local atmospheric conditions, Solar Energy Materials and Solar Cells 240 (2022) 111730. https://doi.org/10.1016/J.SOLMAT.2022.111730.

[11] D. Langevin, P. Bennet, A. Khaireh-Walieh, P. Wiecha, O. Teytaud, A. Moreau, PyMoosh : a comprehensive numerical toolkit for computing the optical properties of multilayered structures, (2023). http://arxiv.org/abs/2309.00654.

[12] F. Abelès, La théorie générale des couches minces, Journal de Physique et Le Radium 11 (1950) 307–309. https://doi.org/10.1051/jphysrad:01950001107030700.

[13] A. Luce, A. Mahdavi, F. Marquardt, H. Wankerl, TMM-Fast, a transfer matrix computation package for multilayer thin-film optimization: tutorial, Journal of the Optical Society of America A 39 (2022) 1007. https://doi.org/10.1364/josaa.450928.

[14] A.D. Rakic´, R. Rakic´, A.B. Djuriš, J.M. Elazar, M.L. Majewski, Optical properties of metallic films for vertical-cavity optoelectronic devices, 1998.

[15] K.M. McPeak, S. V. Jayanti, S.J.P. Kress, S. Meyer, S. Iotti, A. Rossinelli, D.J. Norris, Plasmonic films can easily be better: Rules and recipes, ACS Photonics 2 (2015) 326–333. https://doi.org/10.1021/ph5004237.

[16] Bennet Pauline, Optimisation numérique des structures photoniques, 2022.

[17] J. Rapin, P. Bennet, E. Centeno, D. Haziza, A. Moreau, O. Teytaud, Open Source Evolutionary Structured Optimization, in: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, Association for Computing Machinery, New York, NY, USA, 2020: pp. 1599–1607. https://doi.org/10.1145/3377929.3398091.

[18] O. Teytaud, P. Bennet, A. Moreau, Discrete global optimization algorithms for the inverse design of silicon photonics devices, Photonics Nanostruct 52 (2022). https://doi.org/10.1016/j.photonics.2022.101072.

[19] P. Bennet, Optimisation numérique des structures photoniques, 2022. http://www.theses.fr/2022UCFAC052/document.

[20] A. Soum-Glaude, I. Bousquet, L. Thomas, G. Flamant, Optical modeling of multilayered coatings based on SiC(N)H materials for their potential use as high-temperature solar selective absorbers, Solar Energy Materials and Solar Cells 117 (2013) 315–323. https://doi.org/10.1016/j.solmat.2013.06.030.

[21] G. Ding, C. Clavero, Silver-Based Low-Emissivity Coating Technology for Energy-Saving Window Applications, in: Modern Technologies for Creating the Thin-Film Systems and Coatings, InTech, 2017. https://doi.org/10.5772/67085.

[22] M. Ferrara, Low emission sputtered coatings for smart glazing. How to manage the upcoming light in energy efficient buildings by means of AlN-Ag based sputtered optical filters, 2016. https://www.researchgate.net/publication/303864848.

[23] C.A. Gueymard, The SMARTS spectral irradiance model after 25 years: New developments and validation of reference spectra, Solar Energy 187 (2019) 233–253. https://doi.org/10.1016/J.SOLENER.2019.05.048.

[24] C.A. Gueymard, D. Myers, K. Emery, Proposed reference irradiance spectra for solar energy systems testing, Solar Energy 73 (2002) 443–467. https://doi.org/10.1016/S0038-092X(03)00005-7.

[25] Aránzazu Fernández-García, Florian Sutter, Marco Montecchi, Fabienne Sallaberry (CENER), Anna Heimsath (Fraunhofer ISE), Carlos Heras, Estelle Le Baron, Audrey Soum-Glaude, Guidelines Parameters and Methode to Evaluate the Reflectance Properties OF Materials for Concentrating Solar Power Technology Under Laboratory Conditions, Official Reflectance Guideline Version 3.1 April 2020, 2020.

[26] D.P. Rodgers, Improvements in Multiprocessor Sgstem Design, 1985.

[27] M. Sebastiani, E. Rossi, M. Zeeshan Mughal, A. Benedetto, P. Jacquet, E. Salvati, A.M. Korsunsky, Nano-Scale Residual Stress Profiling in Thin Multilayer Films with Non-Equibiaxial Stress State, Nanomaterials 10 (2020). https://doi.org/10.3390/nano10050853.

[28] S. V. Zhukovsky, A. Andryieuski, O. Takayama, E. Shkondin, R. Malureanu, F. Jensen, A. V. Lavrinenko, Experimental Demonstration of Effective Medium Approximation Breakdown in Deeply Subwavelength All-Dielectric Multilayers, Phys Rev Lett 115 (2015). https://doi.org/10.1103/PhysRevLett.115.177402.

[29] L. Gao, F. Lemarchand, M. Lequime, Exploitation of multiple incidences spectrometric measurements for thin film reverse engineering, Opt Express 20 (2012) 15734. https://doi.org/10.1364/oe.20.015734.

Appendix A : Abélès formalims used in SolPOC package

denis.langevin

October 10th 2025

SolPOC uses a conventional method known as the Abélès formalism for the calculation of the stack optical properties (reflectance, rransmittance, and absorptance) at all wavelengths and incidence angles. The document presented here describes this formalism, which is used without restriction in all versions of SolPOC, including version 0.9.7

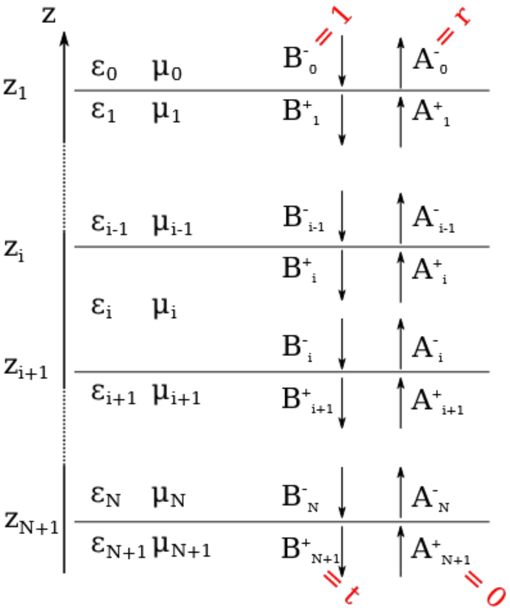


Figure 1: Multilayer structure and definition of important variables. We consider the light coming from the top of the image. ϵ_i and μ_i are respectively the permittivity and the permeability of layer i. A_i (resp. B_i) are defined as the amplitude of the upward propagating (resp. downward propagating) field in layer i, and the $+$ (resp. $-$) superscript indicates that the field value is taken at the top (resp. bottom) of layer i.

We have a multilayer stack of materials defined by their relative permittivities ϵ_i and relative permeabilities μ_i (see Fig. 1). Light is shone on this multilayer with a certain wavelength λ, defining also the wavevector $k_0 = 2\pi/\lambda$. The angle of incidence θ lets us define the wavevector component in the layer plane: $k_x = k_0 \sin(\theta)$).

Thus, we can define the wavevector component perpendicular to the interface planes:

$$\gamma_i = \sqrt{\mu_i \, \epsilon_i \, k_0^2 - k_x^2} \, , \tag{1}$$

with which we also define $\psi_i = \frac{\gamma_i}{\mu_i}$ for all layers.

Contrary to other formalisms, the Abélès formalism computes continuity conditions with the E_y field between each layer. The matrices will therefore be written in terms of $E_{y,i}$ and $\partial z E_{y,i}$. However, because other formalisms (S-matrices and T matrices) decompose the field as upwards and downwards components, as presented on Fig. 1, we will use these to define the starting continuity conditions. Taking for instance the field on the upper side of the layers and using the field continuity, we can write:

$$\begin{cases} E_{y,i}(z_i) & = A_i^+ + B_i^+ \, , \\ \partial z E_{y,i}(z_i)/\mu_i & = i\psi_i(A_i^+ - B_i^+) \, , \\ E_{y,i+1}(z_{i+1}) & = A_i^+ e^{-i\gamma_i h_i} + B_i^+ e^{i\gamma_i h_i} \, , \\ \partial z E_{y,i+1}(z_{i+1})/\mu_{i+1} & = i\psi_i(A_i^+ e^{-i\gamma_i h_i} - B_i^+ e^{i\gamma_i h_i}) \, . \end{cases} \tag{2}$$

These relations and the continuity conditions at the interface at z_{i+1} lead to the matrix equations:

$$\begin{pmatrix} E_{y,i+1} \\ \partial z E_{y,i+1}/\mu_{i+1} \end{pmatrix} = M_i \begin{pmatrix} E_{y,i} \\ \partial z E_{y,i}/\mu_i \end{pmatrix} \tag{3}$$

$$= \begin{pmatrix} \cos(\gamma_i h_i) & -\frac{\sin(\gamma_i h_i)}{\psi_i} \\ \psi_i \sin(\gamma_i h_i) & \cos(\gamma_i h_i) \end{pmatrix} \begin{pmatrix} E_{y,i} \\ \partial z E_{y,i}/\mu_i \end{pmatrix} . \tag{4}$$

Once the matrix product $M = \prod_i M_i$ is computed, we get:

$$\begin{pmatrix} E_{y,N+1} \\ \partial z E_{y,N+1}/\mu_{N+1} \end{pmatrix} = M \begin{pmatrix} E_{y,0} \\ \partial z E_{y,0}/\mu_0 \end{pmatrix} \tag{5}$$

and the r and t coefficients are defined as:

$$E_{y,N+1} = t \tag{6}$$

$$E_{y,0} = r + 1 \tag{7}$$

$$\partial z E_{y,N+1}/\mu_{N+1} = -i\psi_{N+1} t \tag{8}$$

$$\partial_z E_{y,0}/\mu_{i=0} = i\psi_0(r - 1) \tag{9}$$

The final coefficients are written:

$$
\begin{aligned}
r &= -\frac{(M_{1,0} - j\psi_0 M_{1,1}) + j\psi_{N+1}(M_{0,0} - j\psi_0 M_{0,1})}{(M_{1,0} + j\psi_0 M_{1,1}) + j\psi_{N+1}(M_{0,0} + j\psi_0 M_{0,1})} \\
t &= M_{0,0}(r+1) + j\psi_0 M_{0,1}(r-1)
\end{aligned}
$$

Key	Description	Type / Unit	Default Value / Activation Condition
Mat_Stack	List of materials forming the optical stack (from substrate to air)	list[str]	**Required**
n_Stack, k_Stack	Real and imaginary parts of the refractive index for each layer	ndarray	**Required**
Wl	Wavelength range for simulation	ndarray (nm)	280–2500 nm with 5 nm step
Ang	Incidence angle on the stack	float (°)	0°
Sol_Spec	Solar spectrum interpolated on Wl	ndarray	ASTM G173-03 Global Tilt (AM 1.5)
name_Sol_Spec	Name of the selected solar spectrum	str	“Unknown”
Th_Substrate	Substrate thickness	float (nm)	First value of d_Stack, or 1 mm
d_Stack	List of layer thicknesses (including substrate)	list[float]	
vf, vf_range	Volume fractions and allowed range for the Effective Medium Approximation (EMA)	list[float], tuple	(0, 1) if material mixing applies; None otherwise
Th_range	Range of possible layer thicknesses (for optimization)	tuple (nm)	(0, 300)
coherency_limit	Thickness threshold for coherent/incoherent layer treatment	float (nm)	2000 nm
Lambda_cut_1, Lambda_cut_2	Spectral cut-off wavelengths separating reflective/transmissive zones	float (nm)	Depends on cost function (evaluate_RTR, low_e, etc.)
nb_layer	Number of additional layers to optimize	int	0
n_range	Allowed range of refractive index values n	tuple	Required if nb_layer > 0, mean the stack contain theorical layers
d_Stack_Opt	Specifies which layer thicknesses are fixed or optimized	`list[str	float]`
budget	Total number of evaluations of the cost function	int	— (required for optimization)
pop_size	Population size for heuristique optimization method	int	30
algo	Optimization algorithm used	function	DEvol by default
cost_function / evaluate	Cost function to minimize or maximize	function	— (required for optimization)
selection, selection_name	Selection function for evolutionary algorithms	function, str	selection_max
f1, f2, mutation_DE	Differential Evolution parameters	float, str	f1 = 1.0, f2 = 1.0, mutation = "rand_1"
crossover_rate	Crossover rate for DE/GA	float	0.5 (DE) / 0.9 (GA)
precision_AlgoG	Genetic or Strangle algorithm parameters (precision, mutation range, evaluation rate)	float	1.00E-05
mutation_delta	Genetic or Strangle algorithm parameters (precision, mutation range, evaluation rate)	float	15
evaluate_rate	Genetic or Strangle algorithm parameters (precision, mutation range, evaluation rate)	float	0.3
seed, seed_list	Random seed and list of seeds for multiprocessing	int, list[int]	Automatically generated
Mat_Option	Optional list of available materials for optimization	list[str]	
Mode_choose_material	Material selection mode (‘linear’, ‘sigmoid’, or ‘gaussian’)	str	"sigmoid"
C	Solar concentration factor for heliothermal efficiency (evaluate_rh)	float	80
T_air, T_abs	Air and absorber temperatures (for heliothermal efficiency)	float (K)	293 K / 573 K
Signal_PV, Signal_Th	PV and thermal absorber spectral response signals	ndarray	Default files: PV_cells.txt, Thermal_absorber.txt
Signal_H_eye	Human eye sensitivity curve	ndarray	File Human_eye.txt
Signal_fit, Signal_fit_2	Reference data for fitting or comparison	ndarray	—
poids_PV	Weighting factor for PV contribution in PV/CSP optimization	float	3
nb_run	Number of optimization repetitions	int	1