Journal of the Optical Society of America A
OPTICS, IMAGE SCIENCE, AND VISION

TMM-Fast, a transfer matrix computation package for multilayer thin-film optimization: tutorial

ALEXANDER LUCE,[1,2,*] ALI MAHDAVI,[2] FLORIAN MARQUARDT,[1,3] AND HERIBERT WANKERL[2,4]

[1]Friedrich-Alexander Universität Erlangen-Nürnberg, Erlangen, Germany
[2]Osram Opto-Semiconductors, Regensburg, Germany
[3]Max Planck Institute for the Science of Light, Erlangen, Germany
[4]Universität Regensburg, Regensburg, Germany
*Corresponding author: alexander.luce@ams-osram.com

Received 12 January 2022; revised 14 March 2022; accepted 11 April 2022; posted 15 April 2022; published 11 May 2022

Achieving the desired optical response from a multilayer thin-film structure over a broad range of wavelengths and angles of incidence can be challenging. An advanced thin-film structure can consist of multiple materials with different thicknesses and numerous layers. Design and optimization of complex thin-film structures with multiple variables is a computationally heavy problem that is still under active research. To enable fast and easy experimentation with new optimization techniques, we propose the Python package Transfer Matrix Method - Fast (TMM-Fast), which enables parallelized computation of reflection and transmission of light at different angles of incidence and wavelengths through the multilayer thin film. By decreasing computational time, generating datasets for machine learning becomes feasible, and evolutionary optimization can be used effectively. Additionally, the subpackage TMM-Torch allows us to directly compute analytical gradients for local optimization by using PyTorch Autograd functionality. Finally, an OpenAI Gym environment is presented, which allows the user to train new reinforcement learning agents on the problem of finding multilayer thin-film configurations.

https://doi.org/10.1364/JOSAA.450928

1. INTRODUCTION

Although the existence of a globally optimal multilayer thin film has been proven [1–3], the optimization of multilayer thin films concerning reflectivity and transmittivity over wavelength, incidence angles, and many other targets such as phase control remains an interesting field for the scientific community [4–6]. Since the commercial and scientific applications for fine tuned thin films are vast, many dedicated and specialized programs have been developed [7–11] that exploit specialized optimization procedures for thin films, such as the needle method [12–15] or the Fourier method [13,16], which can optimize thin films without the intervention of the user. In the recent past, numerous global optimization techniques have been introduced to the field of thin-film optimization [17–21] including various methods based on reinforcement learning [22,23] and machine learning [24,25]. Especially more complicated thin-film target functions that include scattering and phase control push traditional optimization algorithms to its limits. Researchers are, therefore, investigating new techniques including machine learning and deep search to tackle new and complicated problems [26,27]. Naturally, the need for a standardized, encompassing environment arises to make future research more comparable, consistent, and simple. Here, we propose a comprehensive Python package that provides functionality to researchers to simulate and handle multilayer thin films to test and compare new experimental optimization methods. By relying only on open-source Python packages, automation and interoperability with any software that provides a Python interface can be easily achieved. At its core, the proposed Transfer Matrix Method - Fast (TMM-Fast) package is a parallelized and re-implemented revision of the existing TMM code that was initially published by Byrnes [28]. It implements the Abèles TMM [29] in Python to calculate the optical response to an incident plane wave through a slab of layered thin-film materials with different thicknesses. Given a broad, discretized spectrum of light irradiating a thin film under particular incident angles, the optical properties must be calculated for each contributing wavelength at each angle of incident, e.g., the coefficients of transmission and reflection for a specific wavelength and angle of incidence depend on the thicknesses and dispersive and dissipative refractive index of the layers. Those coefficients can deviate significantly at wavelengths that differ only slightly for the same multilayer thin film. An intuitive approach to increase the computational speed of the response of a multilayer thin film is to parallelize the pairwise independent computations regarding wavelengths and angles. The parallelization is implemented via matrix operations based on the open-source numeric algebra package NumPy [30] and

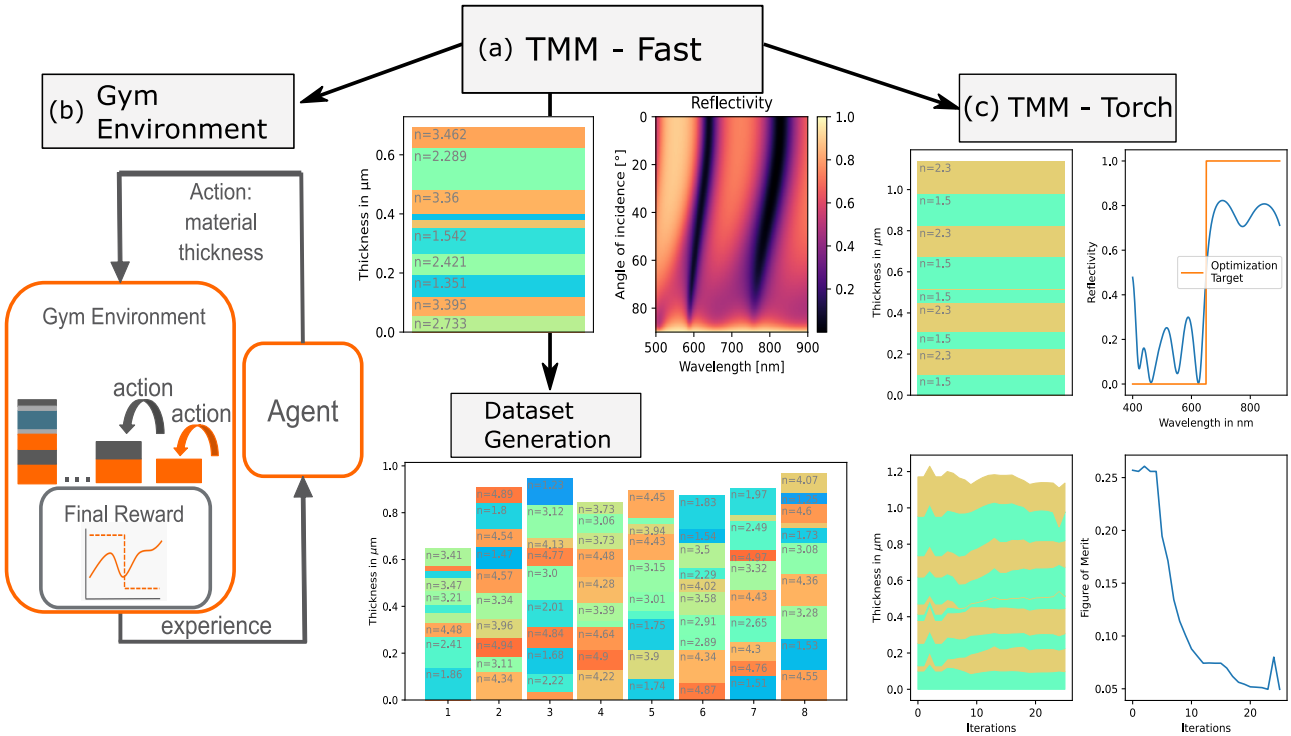

Fig. 1. Schematic overview of the contents of the TMM-Fast package. The package consists of three subdivisions: Part (a) encompasses the core functionality of computing reflectivity and transmissivity of a thin film. Part (b) encompasses an OpenAI Gym environment for reinforcement learning on the task of designing multilayer thin film. Finally, part (c) encompasses a PyTorch implementation of the transfer matrix method to be able to automatically compute gradients and allow for backpropagation through the computation.

the thread-management package Dask. The parallelization was shown to reduce the computational time by ∼100× compared to the original implementation and reaches computational speeds in the same order as state-of-the-art software. By using Dask, the computation of many thin films can be accelerated with an additional factor on the order of available CPU cores. Additionally, by using PyTorch Autograd [31], analytical gradients for any specified thin-film parameter can be computed for local optimizations. Since the implementation is done via PyTorch methods, it can be integrated into advanced neural networks and can be GPU accelerated if necessary. Moreover, the parallelized TMM-Fast code is used to implement an OpenAI Gym environment [32] that can be used by physicists and reinforcement learning researchers. In the environment, the optimization of multilayer thin films is introduced as a sequence generation process and is, thus, considered as a parameterized Markov decision process [33], shown in Section 7. An overview of the scope of the package is shown in Fig. 1. All code is available on GitHub under MIT license [34].

2. OVERVIEW OVER THE CONTENTS

The package is organized in three subpackages: Fig. 1(a) contains the core functionality of the TMM package, computing the optical response of a multilayer thin film quickly via the TMM over a broad range of wavelengths and incident angles. Another part of the core functionality of solving multilayer thin films is the possibility to generate huge datasets with >1e5 data samples for machine learning models.

Figure 1(b) encompasses an OpenAI Gym environment, which allows easy comparisons of different reinforcement learning agents. Here, the environment state is given by the total layer number, layer thicknesses, and material choice. The agent takes an action that specifies the material and thickness of the layer to stack next. The environment implements the multilayer thin-film generation as consecutive conduction of actions and assigns a reward to a proposed multilayer thin film based on how close the actual (solid orange line) fulfils a desired (dashed orange line) characteristic—in this case, a one-dimensional optical characteristic, e.g., reflectivity over wavelength for perpendicular incidence. The experience accumulated by the taken actions of the agent is used to adapt subsequent actions in order to increase the reward and, thus, generate more and more sophisticated multilayer thin films. An investigation of a reinforcement learning agent that was trained with the proposed Gym Environment is given by Wankerl *et al*. [23].

Figure 1(c) encompasses an implementation of the TMM via PyTorch functions, which allow for backpropagation through the entire computation. This enables easy differentiation of the parameters of interest and allows gradient-based and quasi-Newton optimization methods to be used for optimization of multilayer thin films. In the example shown, a random thin film is optimized with respect to the given optimization target, weighted by the mean squared error. Gradients via autograd are used with the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm [35]. The iterative evolution of the layer thicknesses and the correspondingly decreasing figure of merit are shown below. For the given initial values, the optimization converged in about 25 iteration steps.

3. PHYSICAL BACKGROUND OF THE TRANSFER MATRIX METHOD

Light of a particular wavelength passing from one into another material experiences a sudden change of the refractive index n_1 to n_2 that results in reflection and transmission of the incoming wave. A light wave with a wave vector $\mathbf{k}$ that is not parallel to the surface normal experiences refraction. The ratio of angle of incidence θ_1 and angle of refraction θ_2 is given by Snell's law $\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1}$. Given the Fresnel equations, reflection and transmission coefficients r and t can be computed, where a distinction is made between s and p polarization by using a subscript,

$$t_s = \frac{2n_1\cos\theta_1}{n_1\cos\theta_1 + \frac{\mu_{r1}}{\mu_{r2}}n_2\cos\theta_2},$$
$$r_s = \frac{n_1\cos\theta_1 - \frac{\mu_{r1}}{\mu_{r2}}n_2\cos\theta_2}{n_1\cos\theta_1 + \frac{\mu_{r1}}{\mu_{r2}}n_2\cos\theta_2}, \tag{1}$$
$$t_p = \frac{2n_1\cos\theta_1}{\frac{\mu_{r1}}{\mu_{r2}}n_2\cos\theta_1 + n_1\cos\theta_2},$$
$$r_p = \frac{\frac{\mu_{r1}}{\mu_{r2}}n_2\cos\theta_1 - n_1\cos\theta_2}{\frac{\mu_{r1}}{\mu_{r2}}n_2\cos\theta_1 + n_1\cos\theta_2}, \tag{2}$$

with μ_r being the respective magnetic permeability. Note that, for $\theta_1 = 0°$, i.e., vertical incidence, $r_s = r_p$ and $t_s = t_p$. Based on the reflection and transmission coefficients, the reflectivity and transmittivity R and T can be computed as shown in Eq. (3):

$$R_{\lambda,\theta_1} = r_i^2, \quad T_{\lambda,\theta_1} = \frac{n_2\cos\theta_2}{n_1\cos\theta_1}t_i^2, \tag{3}$$

where the subscript i indicates the polarization. Note that, for absorptionless materials, $T + R = 1$ holds.

Now, consider a multilayer thin film with $L \in \mathbb{N}$ layers when the individual layers are denoted by $l \le L$. The light enters from an injection layer of semi-infinite thickness $l = 0$ with a relative amplitude of 1 and exits the multilayer thin film in the outcoupling layer of semi-infinite thickness with $l = L + 1$. From the outcoupling layer, no light enters the thin film. The transmitted part of the light in layer l that travels in "forward" direction, i.e., toward the layer with $l = l_{\text{current}+1}$, is denoted by v_l, and the reflected part that travels in "backward" direction is given by w_l. By using the reflection and transmission coefficients r and t, the response of any layer can be written as

$$\begin{pmatrix} v_l \\ w_l \end{pmatrix} = \begin{pmatrix} e^{-i\delta_l} & 0 \\ 0 & e^{i\delta_l} \end{pmatrix} \begin{pmatrix} 1 & r_{l,l+1} \\ r_{l,l+1} & 1 \end{pmatrix} \frac{1}{t_{l,l+1}} \begin{pmatrix} v_{l+1} \\ w_{l+1} \end{pmatrix}$$
$$= M_l \begin{pmatrix} v_{l+1} \\ w_{l+1} \end{pmatrix}, \tag{4}$$

where $\delta = d_l k_z$ is the accumulated phase of the light wave when traveling through a layer with a specific thickness d_l and with wave vector k_l.

Eventually, the total characteristic matrix of the multilayer thin film is given by

$$\tilde{M} = \prod_{i=0}^{L-1} M_i. \tag{5}$$

Finally, to compute the reflection and transmission of the entire multilayer thin film, one needs to evaluate

$$\begin{pmatrix} 1 \\ r \end{pmatrix} = \tilde{M} \begin{pmatrix} t \\ 0 \end{pmatrix}. \tag{6}$$

The transmission and reflection coefficients are separated,

$$r_{\lambda,\vartheta} = \frac{\tilde{M}_{10}}{\tilde{M}_{00}}, \quad t_{\lambda,\vartheta} = \frac{1}{\tilde{M}_{00}}, \tag{7}$$

and allow us to compute the reflectivity and transmittivity via Eq. (3). Note that one can easily calculate the partial transmission and reflection coefficients by only multiplying Eq. (5) up to $L - l$. The initial angle of incidence in the first layer is given by ϑ.

4. IMPLEMENTATION IN NUMPY

The key contribution to the core functionality of the TMM package is the parallelized handling of the characteristic matrix that reduces computational time. The matrix M_l consists of three separate matrices: matrix A, which encompasses the accumulated phase, and the two matrices holding the coefficients of reflection and transmission, respectively. They are of shape $[N_\lambda, N_\vartheta, L, 2, 2]$, where N_λ and N_θ represent the number of wavelengths and incident angles, respectively. To get the characteristic matrix M_l, NumPy's *einsum* method allows us to specify multiplication and contractions of different dimensions easily:

```
1  M_l = np.zeros((num_lambda, num_angles,
   num_layers, 2, 2), dtype = complex)
2  F = r_list[:, 1:]
3  M_l[:,:,1:-1,0,0] = np.einsum('hji,ji->
   jhi', 1/A, 1/t_list[:, 1:])
4  M_l[:,:,1:-1,0,1] = np.einsum('hji,ji->
   jhi', 1/A, F/t_list[:, 1:])
5  M_l[:,:,1:-1,1,0] = np.einsum('hji,ji->
   jhi', A, F/t_list[:, 1:])
6  M_l[:,:,1:-1,1,1] = np.einsum('hji,ji->
   jhi', A, 1/t_list[:, 1:])
7  Mtilde = np.empty((num_angles, num_lambda,
   2, 2), dtype = complex)
8  Mtilde[:, :] = make_2x2_array(1, 0, 0, 1,
   dtype = complex)
9  for i in range(1, num_layers-1):
10 Mtilde = np.einsum('ijkl,ijlm->ijkm',
   Mtilde, M_l[:,:,i])
```

Finally, $\tilde{M}$ is computed by multiplying out the thickness dimensions.

The entire function is called coh_tmm_fast or by coh_tmm_fast_disp. Both methods differ since the former assumes dispersionless materials whereas the latter accepts dispersive materials. An example is give in Appendix A.1.

5. CORE FUNCTIONALITY SPEEDUP AND DATASET GENERATION

To verify the speedup that the TMM-Fast package provides through parallelization compared to the native implementation of Byrnes, 10 multilayer thin films are generated with 21 layers and are evaluated in a spectral range from 400 to 700 nm at 300 equally spaced points. The angles of incidence range from 0° to 90° based on 40 equally spaced supporting points. The original TMM method requires a computation time of 17.5 ± 0.2 s for the evaluation while our proposed method computes the coefficients of the multilayer thin film in 0.266 ± 0.003 s, which corresponds to an acceleration of $\sim 100\times$. Additionally, to compare the TMM package, the same evaluation was performed by the stackrt function from Lumerical [36], which performed the task in 0.155 ± 0.003 s. The commercial software has an edge over the NumPy implementation, which is expected since it relies on compiled code for the computation.

Finally, the Python package Dask manages parallel threads of the CPU to distribute the computation of different independent computations on all available CPU cores. The application is straightforward: by calling the *coh_tmm_fast* function implicitly inside the *delayed()* function of Dask, a list of all necessary computations is created. Then, the entire list is executed implicitly to create a computational graph for Dask, which is required to orchestrate the parallel threads efficiently. Lastly, the *compute()* method triggers the actual computation, and the result is returned. By running coh_tmm_fast on all available threads, an additional speedup of the dataset creation on the order of the number of available CPU cores is possible. However, the benefit might decrease for very large computational clusters with many cores since the management of the parallel threads creates computational overhang. By calling *multithread_coh_tmm* from the TMM-Fast package, the computation is easily started. A dataset with 1 million samples of 9-layer multilayer thin films at 100 wavelengths and 10 angles of incidence can be created in approximately 40 min on an 8-core machine. Computing more layers, wavelengths points, and angles can increase the computational time significantly. An example is shown in Appendix A.2.

6. TMM-TORCH: MULTILAYER THIN-FILM GRADIENTS VIA AUTOGRAD

For optimization, computing gradients enable gradient-based optimization algorithms such as gradient descent [35]. These gradient-based optimization methods generally converge to local minima with fewer iterations than other non-gradient-based algorithms such as the Nelder–Mead downhill simplex algorithm [37]. The optimization procedure is shown in pseudocode in Algorithm 1. The Python package PyTorch implements matrix multiplication methods, which allow parameters to be automatically differentiated (*Autograd*) via the chain rule [31]. Autograd enables the user to compute the gradients of the input parameters without the necessity of deriving the gradient analytically and can be dynamically adapted to the problem. By using the TMM-Torch subpackage of TMM-Fast, the gradients for a multilayer thin film can be readily computed. An example is shown in Appendix A.3. The application of the transfer matrix takes slightly longer by using the PyTorch routines. Therefore, using the regular TMM-Fast algorithms is still advised to reduce computational time if gradients are not necessary.

Algorithm 1. Pseudocode for the optimization of a thin film with the TMM-Torch subpackage

```
Initialize thin-film parameters
while loss > tolerance do
    compute optical response x⃗ via tmm-torch(parameters)
    compute ℒ(x⃗)
    compute gradients w.r.t. the parameters via backpropagation
  Optimizer(parameters, loss, gradients) → update parameters
end while
return parameters
```

The user needs to specify the *loss function* $\mathcal{L}: \vec{x} \mapsto l$, where $\vec{x}$ denotes the optical properties of interest of the thin film. $\mathcal{L}$ measures the performance of the thin film such that the requirements are fulfilled if $l = 0$. Note that $\mathcal{L}$ must be differentiable and implemented by using PyTorch functionality to allow backpropagation. Additionally, the user must specify an existing or custom *Optimizer*, for example, gradient descent, which updates the thin-film parameters according to the gradient.

7. ENVIRONMENT FOR REINFORCEMENT LEARNING

Reinforcement learning [38] is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize a notion of reward. The proposed code implements such an environment, where agents can stack, characterize, and optimize multilayer thin films. Therefore, the generation of multilayer thin films is considered as a parameterized Markov decision processes [33] and is, thereby, implemented as a sequence generation process: Beginning from $l = 1$, an agent subsequently executes parameterized actions $a_l = (d_l, m_l)$ that specify the thickness d_l and material index m_l of the lth layer. These actions determine which material of which thickness to stack next, thereby consecutively forming a multilayer thin film as illustrated in Fig. 1. The stacked l layers and the optical characteristics of these intermediate multilayer thin films are provided to the agent as the environmental state s. Given this state, the agent takes the next action a_{l+1} until the predefined maximum number of layers is reached or the agent decides to terminate stacking. The optical characteristic of the final proposed multilayer thin film of L layers, e.g., regarding reflectivity $R_{\lambda,\vartheta}(\mathbf{I}, \mathbf{d})$ over wavelength λ and angle ϑ of incidence, is computed via the proposed TMM-Fast method. Here, $\mathbf{I} \in \mathbb{C}^{L \times N_\lambda}$ refers to the matrix of (dispersive and dissipative) refractive indices of the material of each layer. Each material is identified by the material identifier index m_l. $\mathbf{d} \in \mathbb{R}^L$ denotes the vector of layer thicknesses. When the agent starts to stack a thin film, $\mathbf{d}$ and $\mathbf{I}$ are initialized with zeros and get filled according to the taken actions with the layer thicknesses and (dispersive) refractive indices, respectively. The observed reflectivity is compared to a user-defined, desired reflectivity $R^{\text{target}}_{\lambda,\vartheta}$, in order to derive a notion of numeric reward, e.g., an inverted reconstruction error,

$$-\sum_{\lambda,\vartheta} |R^{\text{target}}_{\lambda,\vartheta} - R_{\lambda,\vartheta}(\mathbf{I}, \mathbf{d})|.$$

Based on this reward, the agent learns—for example, based on *Q*-learning [22,39]—to distinguish between good and bad actions and, thus, derive an optimal thin-film design.

Whereas the contained physical methods are well-studied and known for decades, the contribution of the code lies in the implementation of an OpenAI Gym-related environment. Here, the intention is to enable machine-learning researchers without optical expertise to solve the corresponding parameterized Markov decision processes based on common code in the future.

8. CONCLUSION

In this tutorial, the comprehensive Python package TMM-Fast for multilayer thin-film computation, optimization, and reinforcement learning is presented. At its core, the package comprises revised and speedup TMM code from the original TMM package [28]. TMM-Fast enables the user to compute multilayer thin films with NumPy and PyTorch methods and gives full control over the data to the user to enable automation and interoperability. Since the TMM-Fast package evaluates multilayer thin films especially fast, it can be used to generate datasets for machine learning. The reduced computational time also enables evolutionary optimization to be executed in a reasonable amount of time. The TMM-Torch implementation allows the user to compute analytical gradients via automatic differentiation. Quasi-Newton or gradient-based optimization algorithms such as gradient descent can then be used and converge faster to local minima by using an analytical gradient. Finally, an OpenAi Gym environment is proposed, which allows researchers to easily test and experiment with new reinforcement agents on solving the multilayer thin-film problem. All code proposed in this paper is open-source and available on GitHub under the MIT license [34].

APPENDIX A

A.1. TMM-Fast Example

To demonstrate the functionality, a minimal example is given for a multilayer thin film with random thicknesses and refractive indices.

```
import tmm_fast as tmmf
L = 12 # number of layers
d = np.random.uniform(20, 150, L)*1e-9 # thicknesses of the layers
d[0] = d[-1] = np.inf # set first and last layer as injection layer
n = np.random.uniform(1.2, 5, L) # random constant refractive index
n[-1] = 1 # outcoupling into air
wl = np.linspace(500, 900, 301)*1e-9
theta = np.deg2rad(np.linspace(0, 90, 301))
# here s and p polarization is computed and averaged to simulate incoherent light
result = (tmmf.coh_tmm_fast('s', n, d, theta, wl)['R'] + tmmf.coh_tmm_fast('p', n, d, theta, wl)['R'])/2
```

The result of the computation can then be further evaluated. By using the *plot_stacks* function from the TMM-Fast package, the multilayer thin film can be visualized. An example is shown in Fig. 2.

A.2. Dataset Generation

A dataset containing tens of thousands of datasamples can be created by using the *multithread_coh_tmm* function. The external package *tqdm* is used to display a progress bar in order to keep track of the generation progress. In this example, a dataset with 1e6 thin films at 100 wavelengths should be created for vertical incidence.

```
import numpy as np
import tqdm
n_samples = 1e6
n_lambda = 100
n_layers = 12
wl = np.linspace(1000, 1700, n_lambda)*1e-9 # wavelengths
theta = np.array([0]) # vertical incidence
stack_layers = np.random.uniform(5, 180, (n_samples, n_layers))*1e-9
stack_layers[:,0] = stack_layers[:,-1] = np.inf # injection and outcoupling layer
optical_index = np.array([2.5]+[2.0, 1.4]*5+[1.])
optical_index = np.tile(optical_index, (n_samples, 1))
n = 10000 # the dataset is computed in steps of 1e5
dataset = np.empty((n_samples, n_lambda))
for i in tqdm.tqdm(np.array(range(n_samples))[::n]):
tmm_res = np.empty((n, n_lambda))
tmm_res = multithread_coh_tmm('s', optical_index[i:i+n], stack_layers[i:i+n], theta, wl, TorR='R').squeeze()
dataset[i:i+n] = tmm_res
```

Now the dataset can be saved, for example, by using .hdf or NumPy's .npz format.

A.3. TMM-Torch Example

Here, an example of how to compute gradients via automatic differentiation with the TMM-Torch package is shown.

```
import torch
import tmm_fast_torch as tmmt
n_layers = 12 # number of layers
stack_layers = np.random.uniform(20, 150, n_layers)*1e-9 # thicknesses of the layers
stack_layers[0] = stack_layers[-1] = np.inf # set first and last layer as injection layer
```

(Table continued)

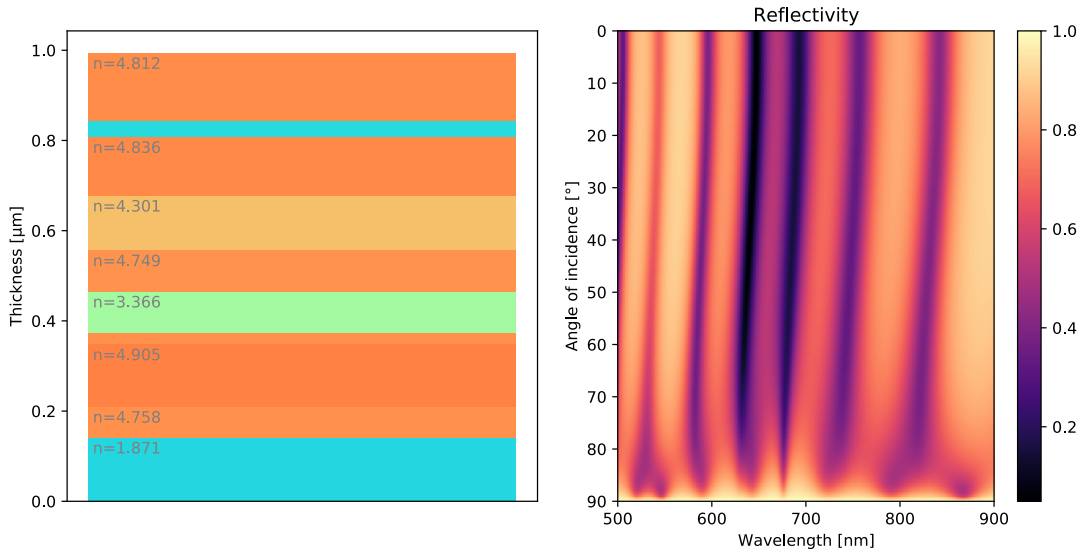

Fig. 2. Multilayer thin film with random layer thicknesses and refractive index under unpolarized illumination. In this example, the materials are dispersionless and dissipationless. The injection region, which is below the thin film in this depiction, possesses a refractive index of $n = 2$. The outcoupling region above the thin film possesses a refractive index of $n = 1$. The reflectivity is computed over a wavelength range of 500–900 nm and from 0° to 90° on a 300 × 300 grid.

```
optical_index = np.random.uniform(1.2, 5, n_layers) # random constant refractive index
optical_index[-1] = 1 # outcoupling into air
stack_layers = torch.tensor(stack_layers, requires_grad=True)
wl = np.linspace(500, 900, 301)*1e-9
theta = np.deg2rad(np.linspace(0, 90, 301))
result = tmmt.coh_tmm_fast('s', optical_index, stack_layers, theta, wl)['R']
mse = torch.nn.MSELoss()
error = mse(result, torch.zeros_like(result)
error.backward()
gradients = stack_layers.grad
```

Since the error is computed with respect to zero reflectivity for all wavelengths at all incidences, the gradient points toward the steepest descent for a broadband anti-reflection coating. By using the minimize function from the Python package SciPy and a gradient-based optimization algorithm, for example, "L-BFGS-B" [40], a local optimum is easily found.

Disclosures. The authors declare no conflicts of interest.

Data availability. Data and code underlying the results presented in this paper are publically available at [34].

REFERENCES

1. A. V. Tikhonravov and J. A. Dobrowolski, "Quasi-optimal synthesis for antireflection coatings: a new method," Appl. Opt. **32**, 4265–4275 (1993).
2. A. V. Tikhonravov, "Some theoretical aspects of thin-film optics and their applications," Appl. Opt. **32**, 5417–5426 (1993).
3. M. Ebrahimi and M. Ghasemi, "Design and optimization of thin film polarizer at the wavelength of 1540 nm using differential evolution algorithm," Opt. Quantum Electron. **50**, 1 (2018).
4. S. W. Anzengruber, E. Klann, R. Ramlau, and D. Tonova, "Numerical methods for the design of gradient-index optical coatings," Appl. Opt. **51**, 8277–8295 (2012).
5. H. Becker, D. Tonova, M. Sundermann, H. Ehlers, S. Günster, and D. Ristau, "Design and realization of advanced multi-index systems," Appl. Opt. **53**, A88–A95 (2014).
6. H. M. Liddell and H. G. Jerrard, *Computer-Aided Techniques for the Design of Multilayer Filters* (A. Hilger, 1981).
7. "OptiLayer," OptiLayer GmbH, https://www.optilayer.com/products-and-services/optilayer.
8. "Thin Film Center," Thin Film Center Inc, https://www.thinfilmcenter.com/essential.php.
9. "RP coating," RP Photonics AG, https://www.rp-photonics.com/coating.html.
10. "TFCalc," Software Spectra, Inc, http://www.sspectra.com/support/index.html.
11. "Film wizard," Scientific Computing International, https://sci-soft.com/product/film-wizard/.
12. B. T. Sullivan and J. A. Dobrowolski, "Implementation of a numerical needle method for thin-film design," Appl. Opt. **35**, 5484–5492 (1996).
13. S. Larouche and L. Martinu, "OpenFilters: open-source software for the design, optimization, and synthesis of optical filters," Appl. Opt. **47**, C219–C230 (2008).
14. A. V. Tikhonravov, M. K. Trubetskov, and G. W. DeBell, "Optical coating design approaches based on the needle optimization technique," Appl. Opt. **46**, 704–710 (2007).
15. A. V. Tikhonravov and M. K. Trubetskov, "Modern design tools and a new paradigm in optical coating design," Appl. Opt. **51**, 7319–7332 (2012).
16. J. A. Dobrowolski and D. Lowe, "Optical thin film synthesis program based on the use of fourier transforms," Appl. Opt. **17**, 3039–3050 (1978).
17. C. P. Chang, Y. H. Lee, and S. Y. Wu, "Optimization of a thin-film multilayer design by use of the generalized simulated-annealing method," Opt. Lett. **15**, 595–597 (1990).
18. W. Paszkowicz, "Genetic algorithms, a nature-inspired tool: a survey of applications in materials science and related fields: Part II," Mater. Manuf. Processes **28**, 708–725 (2013).
19. C. Yang, L. Hong, W. Shen, Y. Zhang, X. Liu, and H. Zhen, "Design of reflective color filters with high angular tolerance by particle swarm optimization method," Opt. Express **21**, 9315–9323 (2013).

20. X. Guo, H. Y. Zhou, S. Guo, X. X. Luan, W. K. Cui, Y. F. Ma, and L. Shi, "Design of broadband omnidirectional antireflection coatings using ant colony algorithm," Opt. Express **22**, A1137–A1144 (2014).
21. S. Martin, J. Rivory, and M. Schoenauer, "Synthesis of optical multilayer systems using genetic algorithms," Appl. Opt. **34**, 2247–2254 (1995).
22. A. Jiang, Y. Osamu, and L. Chen, "Multilayer optical thin film design with deep Q learning," Sci. Rep. **10**, 12780 (2020).
23. H. Wankerl, M. L. Stern, A. Mahdavi, C. Eichler, and E. W. Lang, "Parameterized reinforcement learning for optical system optimization," J. Phys. D **54**, 305104 (2021).
24. R. S. Hedge, "Accelerating optics design optimizations with deep learning," Opt. Eng. **58**, 065103 (2019).
25. J. Roberts and E. W. Wang, "Modeling and optimization of thin-film optical devices using a variational autoencoder," Tech. Rep. (Stanford University, 2018).
26. M. Trubetskov, "Deep search methods for multilayer coating design," Appl. Opt. **59**, A75–A82 (2020).
27. M. Fouchier, M. Zerrad, M. Lequime, and C. Amra, "Design of multilayer optical thin-films based on light scattering properties and using deep neural networks," Opt. Express **29**, 32627–32638 (2021).
28. S. J. Byrnes, "Multilayer optical calculations," arXiv:1603.02720 (2019).
29. F. Abelès, "La théorie générale des couches minces," J. Phys. Radium **11**, 307–309 (1950).
30. C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy," Nature **585**, 357–362 (2020).
31. A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *31st Conference on Neural Information Processing Systems* (2017).
32. "Gym," OpenAI, https://gym.openai.com/.
33. W. Masson, P. Ranchod, and G. Konidaris, "Reinforcement learning with parameterized actions," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence* (2016), pp. 1934–1940.
34. A. Luce and H. Wankerl, "TMM-Fast," GitHub (2021) https://github.com/MLResearchAtOSRAM/tmm_fast.
35. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proc. IEEE **86**, 2278–2324 (1998).
36. "Lumerical," Ansys Canada Ltd., https://www.lumerical.com/products/stack/.
37. J. Nelder and R. Mead, "A simplex method for function minimization," Comput. J. **7**, 308–313 (1965).
38. R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning* (MIT, 1998).
39. C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. thesis (King's College, 1989).
40. "minimize(method='L-BFGS-B')," SciPy, https://docs.scipy.org/doc/scipy/reference/optimize.minimize-bfgs.html.