**code/materials.py**

* Imports the pandas library and gives it the alias `pd`.
* Purpose: Pandas is used for reading and handling tabular data, such as CSV files.
* Imports the `interp1d` function from the `scipy.interpolate` module.
* Purpose: `interp1d` is used to create interpolation functions, which allow you to estimate values between known data points.
* Defines a new class called `Material`.
* Purpose: This class will represent a material (like silicon or SiO₂) and provide methods to get its optical properties at any wavelength.
* Defines the constructor (`\_\_init\_\_`) for the `Material` class.
* Takes one argument: `csv\_path`, which is the path to a CSV file containing material data.
* Reads the CSV file at the given path into a pandas DataFrame called `data`.
* Purpose: The CSV is expected to have columns for wavelength, n (refractive index), and k (extinction coefficient).
* Extracts the 'wavelength' column from the DataFrame and stores it as a NumPy array in `self.wavelengths`.
* Purpose: These are the wavelengths (in nanometers) at which n and k are measured.
* Extracts the 'n' column (refractive index) and stores it as a NumPy array in `self.n`.
* Extracts the 'k' column (extinction coefficient) and stores it as a NumPy array in `self.k`.
* Creates a linear interpolation function for the refractive index (`n`) as a function of wavelength.
* Purpose: Allows you to get the refractive index at any wavelength, even if it's not in the original data, by interpolating between known values.
* `bounds\_error=False, fill\_value='extrapolate'`: If you ask for a wavelength outside the data range, it will extrapolate rather than error.
* Creates a linear interpolation function for the extinction coefficient (`k`) as a function of wavelength.
* Same logic as above, but for k.
* Defines a method `get\_nk` that takes a wavelength in nanometers as input.
* Uses the interpolation function to get the refractive index (`n`) at the requested wavelength.
* Uses the interpolation function to get the extinction coefficient (`k`) at the requested wavelength.
* Returns the interpolated values of `n` and `k` as a tuple.

Summary

* This file defines a `Material` class that loads wavelength-dependent optical data (n, k) from a CSV file.
* It provides a method to get the refractive index and extinction coefficient at any wavelength using interpolation.
* This is essential for accurate optical simulations, as real materials have wavelength-dependent properties.

**solar\_spectrum.py**

* + Imports the pandas library and gives it the alias `pd`.
  + Purpose: Pandas is used for reading and handling tabular data, such as CSV files.
  + Imports the `interp1d` function from the `scipy.interpolate` module.
  + Purpose: `interp1d` is used to create interpolation functions, which allow you to estimate values between known data points.
  + Defines a function called `load\_solar\_spectrum` that takes one argument: `csv\_path`, which is the path to a CSV file containing solar spectrum data.
  + Reads the CSV file at the given path into a pandas DataFrame called `data`.
  + Purpose: The CSV is expected to have columns for wavelength and intensity.
  + Extracts the 'wavelength' column from the DataFrame and stores it as a NumPy array in `wavelengths`.
  + Purpose: These are the wavelengths (in nanometers) at which the solar spectrum is measured.
  + Extracts the 'intensity' column from the DataFrame and stores it as a NumPy array in `intensity`.
  + Purpose: These are the spectral intensities (in watts per square meter per nanometer) at each wavelength.
  + Creates a linear interpolation function for the solar spectrum intensity as a function of wavelength.
  + Purpose: Allows you to get the solar intensity at any wavelength, even if it's not in the original data, by interpolating between known values.
  + `bounds\_error=False, fill\_value=0`: If you ask for a wavelength outside the data range, it will return 0 instead of raising an error.
  + Returns the interpolation function so it can be used elsewhere in your code to get solar spectrum values at arbitrary wavelengths.

Summary

* + This file defines a function to load solar spectrum data from a CSV file.
  + It returns an interpolation function that gives the solar intensity at any wavelength (in nm).
  + This is essential for weighting reflectance calculations by the real solar spectrum in your optical simulations.

**code/test\_moth\_eye.py**

* + Imports:
  + Defines the main test function for reflectance and related features.
  + Creates an instance of your main simulation class.
  + Defines parameters for a nearly flat interface (should behave like a simple Fresnel interface).
  + Defines typical/optimal parameters for a moth-eye nanostructure.
  + Calculates and prints the reflectance for the flat interface.
  + Checks that the calculated reflectance matches the Fresnel formula for a flat interface.
  + Asserts the difference is very small (less than 0.01).
  + Calculates and prints the reflectance for the moth-eye structure.
  + Asserts that the moth-eye structure has lower reflectance than the flat interface.
  + Calculates and prints the solar-weighted reflectance for the moth-eye structure.
  + Asserts the weighted reflectance is within a reasonable range for a good moth-eye AR structure.
  + Calculates and prints reflectance for traditional AR coatings: single-layer, double-layer, and gradient-index.
  + Asserts that double-layer and gradient-index coatings perform better than single-layer.
  + Loops through all supported profile types and checks that each has lower reflectance than the flat interface.
  + Tests edge cases:
  + Negative height (should raise an error or warning).
  + Extremely high refractive index (should not crash).
  + Checks that all expected output files (plots, summaries, etc.) are present in the `results` folder.
  + Prints a final success message if all tests pass.
  + Comment: Explains why test values may differ from optimized results.
  + Runs the test suite if the script is executed directly.

**Summary**

* + The script validates the core reflectance calculations, profile types, and output files.
  + It does not test ML models (which is fine, as those are covered in main workflow).
  + It ensures simulation code is robust, correct, and produces all expected outputs.

**`moth\_eye\_project.py`**

* **XGBoost** short form for eXtreme Gradient Boosting is an advanced machine learning algorithm designed for efficiency, speed and high performance.

It is an optimized implementation of [Gradient Boosting](https://www.geeksforgeeks.org/ml-gradient-boosting/) and is a type of [ensemble learning](https://www.geeksforgeeks.org/a-comprehensive-guide-to-ensemble-learning/) method that combines multiple weak models to form a stronger model. XGBoost uses [decision trees](https://www.geeksforgeeks.org/decision-tree/)as its base learners and combines them sequentially to improve the model’s performance. Each new tree is trained to correct the errors made by the previous tree and this process is called [boosting](https://www.geeksforgeeks.org/boosting-in-machine-learning-boosting-and-adaboost/).

How XGBoost Works?

It builds decision trees sequentially with each tree attempting to correct the mistakes made by the previous one. The process can be broken down as follows:

1. Start with a base learner: The first model decision tree is trained on the data. In regression tasks this base model simply predicts the average of the target variable.
2. Calculate the errors: After training the first tree the errors between the predicted and actual values are calculated.
3. Train the next tree: The next tree is trained on the errors of the previous tree. This step attempts to correct the errors made by the first tree.
4. Repeat the process: This process continues with each new tree trying to correct the errors of the previous trees until a stopping criterion is met.
5. Combine the predictions: The final prediction is the sum of the predictions from all the trees.

* **Neural networks** are machine learning models that mimic the complex functions of the human brain.

1. Neurons: The basic units that receive inputs, each neuron is governed by a threshold and an activation function.
2. Connections: Links between neurons that carry information, regulated by weights and biases.
3. Weights and Biases: These parameters determine the strength and influence of connections.
4. Propagation Functions: Mechanisms that help process and transfer data across layers of neurons.
5. Learning Rule: The method that adjusts weights and biases over time to improve accuracy.

* **Linear Transformation:** Each neuron in a layer receives inputs which are multiplied by the weights associated with the connections. These products are summed together and a bias is added to the sum. This can be represented mathematically as:

*z=w1x1+w2x2+…+wnxn+bz=w1​x1​+w2​x2​+…+wn​xn​+b*

* **Batch Normalization** is used to reduce the problem of [internal covariate shift](https://www.geeksforgeeks.org/internal-covariant-shift-problem-in-deep-learning/) in neural networks. It works by normalizing the data within each mini-batch. This means it calculates the mean and variance of data in a batch and then adjusts the values so that they have similar range. After that it scales and shifts the values so that model learn effectively.
* **Rectified Linear Unit (ReLU)** is a popular activation functions used in neural networks, especially in deep learning models. The ReLU function is a piecewise linear function that outputs the input directly if it is positive; otherwise, it outputs zero.
* **Leaky ReLU** is a modified version of ReLU designed to fix the problem of dead neurons. Instead of returning zero for negative inputs it allows a small, non-zero value.
* [**Dropout**](https://www.geeksforgeeks.org/dropout-in-neural-networks/) is a regularization technique which involves randomly ignoring or "dropping out" some layer outputs during training, used in deep [neural networks](https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/) to prevent [overfitting](https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/).
* **Adam (Adaptive Moment Estimation)** optimizer combines the advantages of Momentum and RMSprop techniques to adjust learning rates during training. It works well with large datasets and complex models because it uses memory efficiently and adapts the learning rate for each parameter automatically.
* In [**K-Fold Cross Validation**](https://www.geeksforgeeks.org/r-language/k-fold-cross-validation-in-r-programming/) we split the dataset into k number of subsets known as folds then we perform training on the all the subsets but leave one (k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.
* The **Transfer Matrix Method (TMM)** is a technique used to analyze the behavior of waves propagating through layered or stratified media.
* **Snell's Law**, also known as the law of refraction, describes the relationship between the angles of incidence and refraction when a wave (like light) passes from one medium to another.
* **Weighted reflectance** is a measure of how much light is reflected across a range of wavelengths, weighted according to the relative importance or intensity of each wavelength (such as based on the solar spectrum).
* The **solar spectrum** refers to the full range of electromagnetic radiation emitted by the Sun, including ultraviolet (UV), visible, and infrared (IR) wavelengths.
* **Spectral reflectance** is the ratio of reflected to incident light at a specific wavelength.
* **Angular performance** refers to how well a moth-eye structured solar panel maintains its light absorption or efficiency when sunlight strikes it at different angles throughout the day or year.

| **Function/Area** | **Formula/Model** | **Term Explanations** | **Why This Model?** | **Citation/Comment** |
| --- | --- | --- | --- | --- |
| **transfer\_matrix** |  | *M*: Total transfer matrix for N layers;  Phase thickness;  *nj*​: Refractive index;  *dj*​: Thickness;  *θj*​: Angle;  λ: Wavelength;  *qj*​: Impedance term;  *M*11​,*M*21​: Matrix elements; *R*: Reflectance | TMM is the gold standard for multilayer thin films, accurately models interference and phase effects. | Sun et al. (2008), Dong et al. (2015) |
| **Bruggeman effective medium theory** |  | neff(z)*:* Effective refractive index at depth z;  *f*: Volume fraction;  *n*1​,*n*2​: Refractive indices | EMT models subwavelength structures as a graded index, enabling fast, accurate simulation. | Khezripour et al. (2018) |
| **single\_layer\_reflectance** |  | *R*: Reflectance; *n*0​: Incident medium index; *n*1​: Layer index | Fresnel equations are the standard for single interface reflectance. | Born & Wolf, "Principles of Optics" |
| **double\_layer\_reflectance** | Use TMM for two layers (see above, N=2) | As above, for two layers | Captures interference in double-layer AR coatings, standard in optics. | Born & Wolf |
| **gradient\_index\_reflectance** | TMM/EMT applied to a stack with varying *n*(*z*) | *n*(*z*): Refractive index as a function of depth | Models graded-index AR coatings, more realistic for moth-eye structures. | Khezripour et al. (2018), Sun et al. (2008) |
| **weighted\_reflectance** | ​ | *R*weighted​: Solar-weighted reflectance; *R*(*λ*): Reflectance at wavelength; *S*(*λ*): Solar spectrum | Solar-weighted reflectance is the industry standard for PV performance. | IEC 60904-3, Standard PV analysis |
| **multi\_objective\_score** |  | *wi*​: Weight for objective; *R*: Reflectance; *C*: Cost; *Y*: Yield | Weighted sum allows flexible trade-offs; widely used in engineering optimization. | This work; standard multi-objective optimization |
| **multi\_objective\_optimize** | Random Sampling (Stochastic Parameter Search) | Randomly samples parameter space within bounds and selects the best by score | Simple, robust baseline for multi-objective optimization; no local minima bias |  |
| **advanced\_optimize** | Differential Evolution | Uses population-based or stochastic search to minimize the multi-objective score | Robust global optimizers for non-convex, multi-modal problems | Storn & Price (1997), Wales & Doye (1997), Xiang et al. (2010) |
| **adaptive\_bounds** | Bounds = f(previous best, ± margin, global limits) | Bounds: Parameter search limits; previous best: best parameters from last iteration; margin: allowed variation; global limits: absolute min/max | Dynamically narrows or shifts parameter search space based on previous optimization results, improving efficiency and convergence. | This work; adaptive optimization literature |
| **uncertainty\_analysis** | compute *R* for each, report mean ± std | *pi*​: Parameter; N(1,*σ*): Normal random variable; *R*: Reflectance; mean ± std: Average and standard deviation | Monte Carlo is standard for propagating uncertainty in nonlinear models. | Taylor (1997), Standard uncertainty propagation |
| **calculate\_temperature\_impact** |  | *RT*​: Reflectance at temperature; *R*base​: Base reflectance; *α*: Temp. coefficient; *T*: Temperature; *T*0​: Reference temp. | Linear models are standard for first-order temperature effects in AR coatings. | Literature/industry standard |
| **calculate\_manufacturing\_cost** | Empirical formula: Cost = f(feature size, aspect ratio, method) | Cost: Estimated manufacturing cost; Feature size: Smallest pattern dimension; Aspect ratio: Height/width; Method: Fabrication technique | Reflects real-world fabrication constraints; can be adapted to new data. | This work or cite if from literature |
| **calculate\_lifetime\_performance** | Lifetime = base × (environmental factors, material) | Lifetime: Expected operational years; Base: Nominal lifetime; Environmental factors: Rain, dust, UV; Material: Material durability | Standard in PV/AR literature; reflects real-world durability. | IEC 61215, industry standard |
| **calculate\_comprehensive\_lifetime** | Lifetime = base × (all environmental factors, material, manufacturing) | Lifetime: Expected operational years; base: Nominal lifetime; environmental factors: rain, dust, UV, humidity, temperature; material: durability; manufacturing: yield, cost | Integrates all relevant degradation and manufacturing factors for a holistic, realistic lifetime estimate. | IEC 61215, Smith et al. (2012), this work |
| **calculate\_environmental\_impact** | Renv = R.*f*(rain, dust, UV, humidity, temperature) | *R*env​: Reflectance after environmental exposure; *R*: Initial reflectance; *f*: Degradation factor | Captures real-world degradation, important for outdoor applications. | Smith et al. (2012), literature on environmental effects |
| **calculate\_humidity\_impact** |  | *R*hum​: Reflectance after humidity exposure; *R*: Initial reflectance; *f*hum​: Humidity degradation factor | Models increased reflectance due to humidity-induced degradation | Literature on humidity effects in AR coatings |
| **calculate\_uv\_impact** |  | *R*uv​: Reflectance after UV exposure; *R*: Initial reflectance; *f*uv​: UV degradation factor; *r*deg​: Degradation rate; *t*exp​: Exposure time | Models long-term UV-induced degradation of AR coatings | Literature on UV stability of AR coatings |
| **calculate\_rain\_impact** | Kinetic Energy: KE = 0.5 × ρ × (4/3 × π × (d/2)³) × v²  Erosion Factor: EF = 1.0 + 0.001 × (KE × AR × Y) / 10⁶  Hardness Factor: HF = 1.0 - 0.1 × (H / 10)  Total Impact: TI = EF × HF | **KE:** Kinetic energy of raindrops (J  **ρ:** Water density (1000 kg/m  **d:** Raindrop diameter (2×10⁻³ m)  **v:** Raindrop velocity (9 m/s)  **AR:** Annual rainfall (1000 mm/year)  **Y:** Years of exposure (25)  **H:** Material hardness (Mohs scale) | Physics-based erosion model using actual raindrop kinetic energy. Accounts for material hardness and cumulative exposure over 25-year lifetime. | Based on materials science erosion models. Raindrop impact causes micro-cracking and surface roughening, increasing reflectance over time. |
| **calculate\_dust\_impact** | Dust Thickness: DT = (AD × Y / DD) × 10⁻⁶  Dust Factor: DF = 1.0 + 0.1 × (DT / SH)  Impact: I = BR × DF | AD: Annual dust accumulation (100 g/m²/year)  Y: Years of exposure (25)  DD: Dust density (1500 kg/m³)  DT: Calculated dust thickness (m)  SH: Structure height (m)  BR: Base reflectance | Accumulation model based on field studies. Dust layer creates additional optical interface that increases reflectance proportionally to thickness. | Based on solar panel soiling studies. Dust accumulation creates optical layer that increases reflectance, reducing anti-reflection effectiveness. |
| ML model selection | Random Forest, XGBoost, Neural Network | Random Forest: Ensemble of decision trees; XGBoost: Gradient-boosted trees; Neural Network: Multi-layer perceptron; MSE: Mean squared error | All three are state-of-the-art for tabular regression; best model (lowest MSE) is selected for final predictions. | Breiman (2001), Chen & Guestrin (2016), Goodfellow et al. (2016) |
| **train\_nn** |  | *N*: Number of samples; *yi*​: True value; *y*^​*i*​: Predicted value | MSE is standard for regression; NN captures nonlinear relationships. | Goodfellow et al. (2016) |
| **calculate\_manufacturing\_yield** |  | Yield: Fraction of devices meeting specs; Feature size, aspect ratio, process: As above | Models real-world fabrication success rates. | This work, standard fabrication analysis |
|  |  |  |  |  |