**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`**

| **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) |
| **effective\_index\_profile** |  | 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** | Differential Evolution, Basin Hopping, Dual Annealing (see Scipy) | 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) |
| **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 |
| **plot\_literature\_comparison** | Bar chart of reflectance values from this work and literature | Each bar: Reflectance value; X-axis: Method; Y-axis: Reflectance (%) | Visual comparison is clear and widely accepted in scientific reporting. | Comparison with literature |
| **plot\_angular\_response** | *R*(*θ*) vs. *θ*, with error bars from uncertainty analysis | *R*(*θ*): Reflectance at angle; *θ*: Angle of incidence; Error bars: Standard deviation | Shows AR performance at different angles, critical for solar applications. | Standard AR coating analysis |
| **plot\_sensitivity\_heatmap** | 2D grid: *R*(*p*1​,*p*2​) | *R*(*p*1​,*p*2​): Reflectance as a function of two parameters | Visualizes sensitivity to parameter changes, helps identify robust designs. | Standard practice |
| **plot\_3d\_reflectance\_surface** | 3D surface: *R*(*p*1​,*p*2​) | As above, but shown as a 3D surface plot | Shows joint parameter effects, useful for optimization insight. | Standard visualization |
| **plot\_parallel\_coordinates** | Parallel coordinates plot of normalized parameters and reflectance | Each line: One parameter set; Axes: Each parameter and reflectance; Color: Best reflectance highlighted | Visualizes high-dimensional optimization results, highlights best solutions. | Standard ML/data visualization |
| **generate\_txt\_summary** | Text report, includes all results and parameter tables | All terms as above, formatted for clarity | Ensures reproducibility and clarity for examiners/readers. | This work |
| **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\_environmental\_impact** |  | *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\_quality\_metrics** | Derived metrics: aspect ratio, yield, etc. | Aspect ratio: Height/width; Yield: Fraction of successful devices; Others: As defined in code | Standard for evaluating manufacturability and performance. | This work, standard fabrication analysis |
| ML model selection | Random Forest, XGBoost | Random Forest: Ensemble of decision trees; XGBoost: Gradient-boosted trees; Neural Network: Multi-layer perceptron; MSE: Mean squared error | Both RF and XGBoost 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) |
| **plot\_learning\_curve** | Plot train/test error vs. sample size | X-axis: Number of training samples; Y-axis: Error (MSE); Curves: Train and test error | Standard for diagnosing ML model performance and overfitting. | Standard ML practice |
| **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 |
| **calculate\_cooling\_factor** | Cooling=a+b⋅(aspect ratio) | Cooling: Estimated cooling factor; *a*,*b*: Empirical constants; Aspect ratio: Height/width | Models heat dissipation, relevant for solar cell operation. | This work, empirical model |
| **calculate\_surface\_energy** |  | Surface energy: Energy per unit area; Roughness: RMS surface roughness; Aspect ratio: Height/width | Models anti-soiling and hydrophobicity. | This work, empirical model |
| **calculate\_contact\_angle** |  | *θ*: Contact angle (deg); *c*: Empirical constant; Aspect ratio: Height/width | Models hydrophobicity, relevant for self-cleaning surfaces. | This work, empirical model |
|  |  |  |  |  |