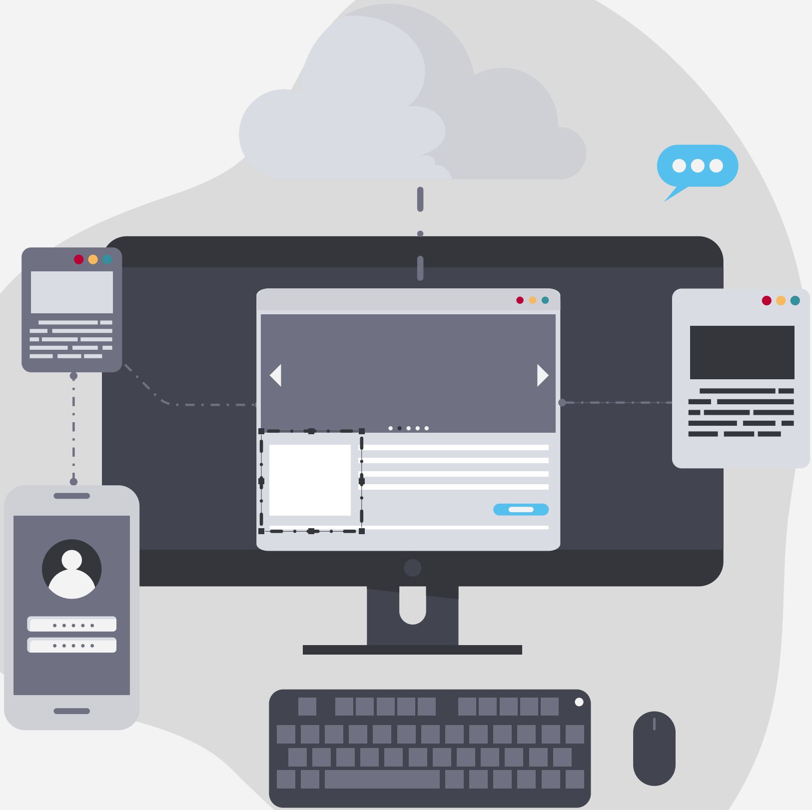
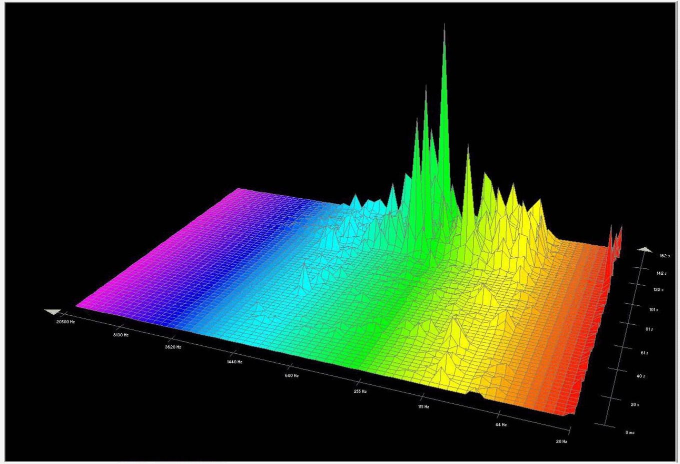
Spectrum optimizer



Nikulin Dmitry
Nemchinov Valentin

About the task

To develop a Python program for selecting the minimum set of artificial light sources, whose combined spectrum reproduces the reference solar spectrum (AM1.5G) as accurately as possible with an acceptable deviation of <15% in total at all wavelengths.



Stack
Implemented in python

scipy – used for non-negative least squares optimization (nnls) to find the optimal coefficients for combining light source spectra in order to approximate the target solar spectrum

numpy – used for numerical computations, including array operations, generating synthetic spectra using Gaussian distributions, adding noise, and performing interpolation and normalization of spectral data

pandas – used for loading, processing, and manipulating spectral data in tabular format, including reading CSV files, handling wavelengths and intensities, and organizing results

optimizer.py

optimizer.py > SpectrumOptimizer > __init__

```python
import numpy as np
import pandas as pd
from scipy.optimize import nnls
from tqdm import tqdm

class SpectrumOptimizer:
    def __init__(self, target_spectrum, sources, wavelength_range=(380, 1100), max_sources=10, tolerance=0.15):
        """
        Основной класс для подбора минимального набора источников света,
        чей комбинированный спектр воспроизводит эталонный спектр AM1.5G
        """
        # Фильтруем эталонный спектр по диапазону
        self.target_spectrum = target_spectrum[
            (target_spectrum["Wavelength"] >= wavelength_range[0]) &
            (target_spectrum["Wavelength"] <= wavelength_range[1])
        ].reset_index(drop=True)

        self.sources = sources
        self.max_sources = max_sources
        self.tolerance = tolerance
        self.target_wavelengths = self.target_spectrum["Wavelength"].values
        self.target_values = self.target_spectrum["Spectral_irradiance"].values

        # Нормализуем эталонный спектр для согласованности масштабов
        self.target_max = self.target_values.max()
        self.target_values_normalized = self.target_values / self.target_max

        print(f"📊 Эталонный спектр: мин={self.target_values.min():.4f}, макс={self.target_values.max():.4f}")
        print(f"📊 После нормализации: мин={self.target_values_normalized.min():.4f}, макс={self.target_values_normalized.max():.4f}")
```

Key steps:

1. **Normalization:** Both the target and source spectra are normalized to ensure c scaling.
2. **Correlation ranking:** Light sources are ranked based on their correlation with t spectrum.
3. **Iterative selection:** At each step, one source is added to the combination, and N used to compute optimal non-negative coefficients that minimize the spectral e
4. **Error evaluation:** The relative error between the combined and target spectru computed, and the process stops if the error falls below a given tolerance thres the max number of sources is reached.

Goal : Minimize the number of light sources while achieving an acceptable match to the spectrum.

generate.py

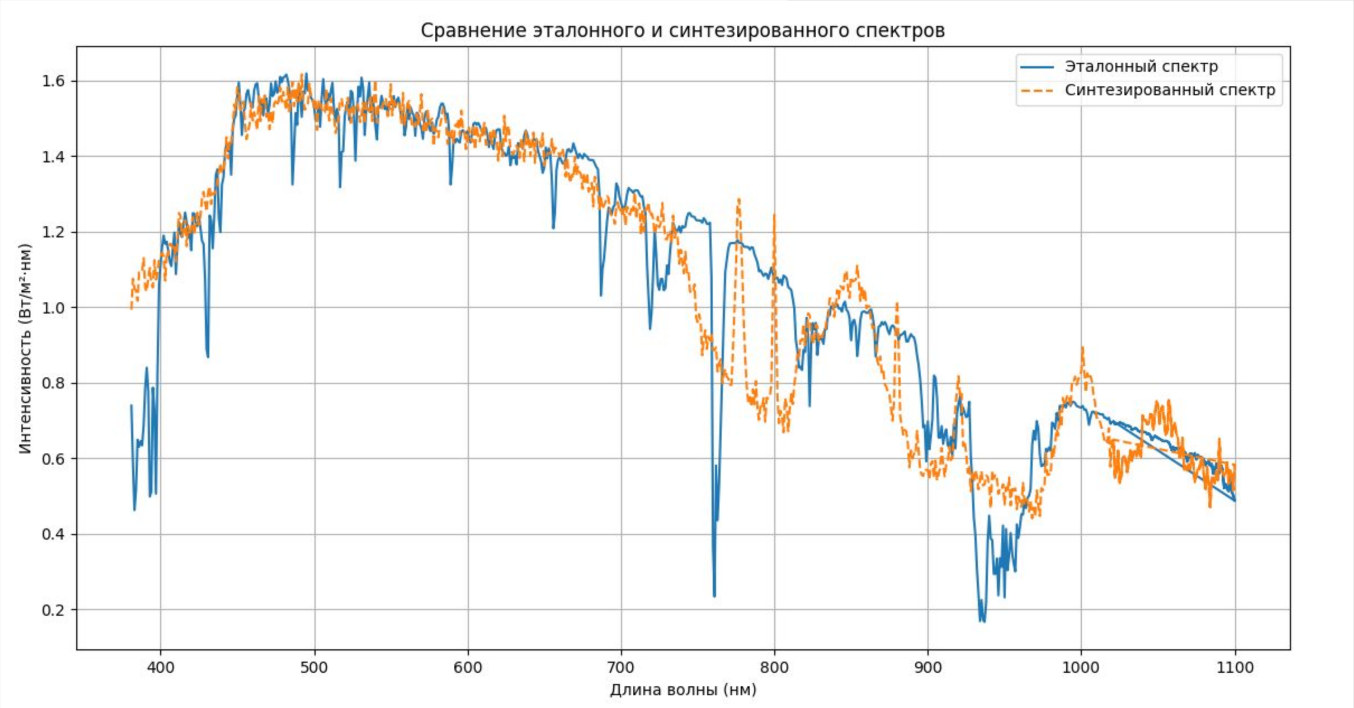
```
def gaussian(x, mu, sigma, amp=1.0):
    """Гауссово распределение для моделирования спектра"""
    return amp * np.exp(-((x - mu) ** 2) / (2 * sigma ** 2))

def generate_led_spectrum(peak, sigma, amp, name):
    """Генерация синтетического спектра с шумом и нормализацией"""
    intensity = gaussian(WAVELENGTH_RANGE, peak, sigma, amp)
    intensity = np.clip(intensity, 0, None)

def main():
    print(f"Генерация спектров для {len(LED_TYPES)} источников...")
    for led in LED_TYPES:
        generate_led_spectrum(
            peak=led["peak"],
            sigma=led["sigma"],
            amp=led["amp"],
            name=led["name"]
```

The algorithm selects the minimum number of light sources whose spectra, combined with non-negative coefficients (found through NNLS), most accurately reproduce the reference solar spectrum of AM1.5G with a given accuracy.

Result



- Average error: 0.1313 (13%)
- Maximum error: 2.7820 (278%)
- Number of sources: 61