Hyperspectral Pyrometry

Naman J. Parikh

July 2, 2024

Motivation and Application

In this document, I will present the method I have used for hyperspectral pyrometry. The goal of this project is to assist in bulk crystal growths performed in the Laser Diode Floating Zone furnace. While the furnace can provide information on the temperature at which it is running, it cannot capture the temperature of the material itself and the complex thermal gradient in the molten zone. Having information of the temperature and thermal gradient of the molten zone can allow researchers to adjust their growths in real time to ensure they create the product they desire and have additional data to aid with reproducibility. We hope to provide this information using a hyperspectral camera mounted in the furnace and taking images which are streamed and analyzed in real time using the following methods.

Note that since this method has been developed for specific conditions, it may require modification or refinement for other uses. In particular, we assume the object imaged has been heated sufficiently so as to exhibit blackbody radiation.

Loading and Correcting Data

The Python package Spectral Python (SPy) allows for easily loading ENVI-formatted hyperspectral images in Python. We can open and load an image from it's `hdr` and `raw` files with the following.

```
import spectral.io.envi as envi
data_ref = envi.open("raw.hdr", "raw.raw")
data_tensor = np.array(data_ref.load())
```

We also need to correct the data against the white and dark references for the image provided by the camera. These references can be similarly loaded.

```
white_ref = envi.open('whiteReference.hdr', 'whiteReference.raw')
white_tensor = np.array(white_ref.load())

dark_ref = envi.open('darkReference.hdr', 'darkReference.raw')
dark_tensor = np.array(dark_ref.load())
```

We then correct the image with the following line.

```
corrected_data = np.divide(
    np.subtract(data_tensor, dark_tensor),
    np.subtract(white_tensor, dark_tensor))
```

Visualizing Data

Printing the reference object `data_ref` will provide metadata for the hyperspectral image. The data tensor can be rendered as an image using the following.

```
from spectral import imshow
imshow(data_tensor)
```

The tensor is a 3-D numpy array. Each individual pixel in the image can be accessed by indexing as one would index a 2-D array. For each pixel, the spectrum is an array of values representing the intensity of each measured wavelength. In order to use the spectrum, an array of wavelength values is needed. This can be acquired by consulting the camera information, or with the following code. This code parses the `hdr` file to extract an array of wavelength values into the array `wavelengths` and the units as a string in `units`.

```
# reading hdr file
file = open("raw.hdr", 'r')
text = file.read()

# parsing for wavelength values and units
start_id = "\nwavelength = {\n"
start_index = text.find(start_id) + len(start_id)
end_id = "\n}\n;AOI height"
end_index = text.find(end_id)
wavelengths = text[start_index:end_index]
wavelengths = np.array(wavelengths.split("\n,"), dtype="float32")
units_id = "wavelength units = "
units_index = text.find(units_id) + len(units_id)
units = text[units_index:text.find(start_id)]
```

A spectrum at pixel (x, y) can then be visualized using

```
import matplotlib.pyplot as plt
plt.scatter(wavelengths, corrected_data[x][y])
plt.xlabel(f"Wavelength [{units}]")
plt.ylabel("Intensity [arbitrary units]")
```

Spectrum Fitting via Non-Linear Least Squares Regression

Theory

We can use machine learning to fit a given spectrum to the theoretical blackbody spectrum equation. In this equation, we have intensity as a function of wavelength given by

$$I(\lambda) = \epsilon \left(\frac{2hc^2}{\lambda^5}\right) \left(\exp\left(\frac{hc}{\lambda k_B T}\right) - 1\right)^{-1}$$

where ϵ is the emissivity, h is Planck's constant, c is the speed of light, k_B is Boltzmann's constant, T is the temperature of the blackbody, and λ is the wavelength.
Since emissivity can in theory vary with wavelength, we can model it as a function of wavelength with a quadratic approximation

$$\epsilon(\lambda) = a_0 + a_1\lambda + a_2\lambda^2$$

Substituting this into the equation for blackbody radiation intensity gives

$$I(\lambda) = (a_0 + a_1\lambda + a_2\lambda^2)\left(\frac{2hc^2}{\lambda^5}\right)\left(\exp\left(\frac{hc}{\lambda k_B T}\right) - 1\right)^{-1}$$

Additionally, it is possible that the blackbody is not the only light source. To account for any stray light source in the image, we can add a constant offset Ω, giving the final model

$$I(\lambda) = (a_0 + a_1\lambda + a_2\lambda^2)\left(\frac{2hc^2}{\lambda^5}\right)\left(\exp\left(\frac{hc}{\lambda k_B T}\right) - 1\right)^{-1} + \Omega$$

We now have a wavelength-dependent model with parameters a_0, a_1, a_2, T. We can usenon-linear least squares regression to fit the data to this model and learn the parameters.

Implementation

We first express the blackbody equation.

```
import numpy as np

# constants
h = 6.626e-34 # Planck's constant
c = 299792458 # Speed of light
k = 1.380649e-23 # Boltzmann constant

# l: wavelength, T: temperature, e: emissivity, offset: constant offset to account for stray
    light in the data
def blackbody(l, T, e, offset):
    return (e * ((2 * h * c**2) / l**5) * (1 / (np.exp((h * c) / (l * k * T)) - 1))) + offset
```

For the least squares regression, we use the Sci-Kit Learn function as shown.

```
from scipy.optimize import least_squares

spectrum = corrected_data[x][y]

# parameters = [a0, a1, a2, offset, T]
def intensity(params, l):
    e = params[0] + (params[1] * l) + (params[2] * l**2)
    return blackbody(l, params[4], e, params[3])

# Residuals of fitting
def residuals(params):
    result = []
    for i in range(len(wavelengths)):
        Si = intensity(params, wavelengths[i] * 1e-9)
        St = spectrum
        result.append(Si - St)
    return np.array(result)

params0 = np.array([1, 1, 1, 0.1, 500]) # initial parameters
result = least_squares(residuals, params0)
```

This returns the final parameters `result.x` and the final fitting cost `result.cost`. In particular, the temperature is given in `result.x[-1]`.

References