```python
#!/usr/bin/env python

from __future__ import print_function

import os

import pyfits
import numpy as np
import matplotlib.pyplot as pl
from matplotlib.ticker import MaxNLocator
from scipy.interpolate import interp1d
from scipy.integrate import simps


emission_lines = [
                    (r'OII',        3727.,        0),
                    (r'H\gamma',    4341.6803,    0),
                    (r'H\beta',     4862.6778,    0),
                    (r'OIII',       5008.1666,    12),
                    (r'H\alpha',    6564.6127,    0),
                    (r'SII',        6732.5382,    12),
                ]


def load_filter(band):
    fn = os.path.join(os.path.dirname(__file__), "filters",
            "sdss_jun2001_{0}_atm.dat".format(band))

    # Load the response curve.
    data = []
    with open(fn) as f:
        for line in f:
            if line[0] != "#" and len(line) > 1:
                data.append(line.split())
    data = np.array(data, dtype=float)

    # Build a spline representation.
    spline = interp1d(data[:, 0], data[:, 1], bounds_error=False,
                                               fill_value=0.0)

    # Compute the normalization integral up to a constant.
    y = data[:, 1] / data[:, 0]  # R(Lambda) / Lambda
    norm = simps(y, data[:, 0])

    return spline, norm


def load_filters():
    filters = {}
    for b in "ugriz":
        filters[b] = load_filter(b)
    return filters


def load_spectrum(fn):
    hdus = pyfits.open(fn)

    # Get the spectrum.
    spec = hdus[1].data
    loglam = np.array(spec["loglam"], dtype=float)
    flux = np.array(spec["flux"], dtype=float)

    # Get the redshift.
    z = float(hdus[2].data["Z"])

    hdus.close()
    return loglam, flux, z


def plot_spectrum(log10lam, flux, ax=None, z=0.0, smooth=0, **kwargs):
    if ax is None:
```

```python
        ax = pl.figure().add_subplot(111)

    x = 10 ** (log10lam - np.log10(1 + z))
    y = flux

    if smooth > 0:
        r = np.arange(-3 * smooth, 3 * smooth + 1)
        f = np.exp(-0.5 * r ** 2 / smooth ** 2) / np.sqrt(2 * np.pi * smooth)
        f /= np.sum(f)
        y = np.convolve(flux, f, mode="same")

    ax.plot(x, y, color="k", **kwargs)

    return x, y


def get_magnitude(log10lam, flux, filter_response, filter_norm):
    # Get the magnitude of in a band up to a constant across bands.
    lam = 10. ** log10lam
    y = flux * filter_response(lam) * lam
    integral = simps(y, x=lam)

    if integral == 0.0:
        return np.inf

    return -2.5 * np.log10(integral) + 2.5 * np.log10(filter_norm)


hcoverk = 1.43878e8  # h * c / k in units of [Angstroms] * [Kelvin]
twohc2 = 1.191043e-16 * 1e-3 * (100) ** 4  # ergs cm^2 / s


def black_body(lam, T):
    return twohc2 / lam ** 5 * (1e8) ** 4 / (np.exp(hcoverk / lam / T) - 1)


def part3():
    filters = load_filters()
    loglam, flux, z0 = load_spectrum("data/part3.fits")

    # De-redshift the spectrum.
    loglam -= np.log10(1 + z0)

    # Compute the colors as a function of redshift.
    Npts = 50
    zrange = [0.07, 0.53]
    data = np.empty((Npts, 3))
    for i, z in enumerate(np.linspace(zrange[0], zrange[1], Npts)):
        l = loglam + np.log10(1 + z)
        mags = dict([(b, get_magnitude(l, flux, f[0], f[1]))
                            for b, f in filters.iteritems()])
        data[i, :] = [z, mags["g"] - mags["r"], mags["r"] - mags["i"]]

    # Generate the plot.
    fig = pl.figure(figsize=(4, 7))
    fig.subplots_adjust(left=0.2, bottom=0.1, right=0.96, top=0.98,
            hspace=0.01)

    ax = fig.add_subplot(211)
    ax.plot(data[:, 0], data[:, 1], "k", lw=2)
    ax.xaxis.set_major_locator(MaxNLocator(5))
    ax.set_xticklabels([])
    ax.set_xlim(zrange)
    ax.set_ylim([0.48, 2.5])
    ax.set_ylabel(r"$g - r$")

    ax = fig.add_subplot(212)
    pl.plot(data[:, 0], data[:, 2], "k", lw=2)
    ax.xaxis.set_major_locator(MaxNLocator(5))
    ax.set_xlim(zrange)
    ax.set_ylim([0.2, 1.1])
    ax.set_ylabel(r"$r - i$")
    ax.set_xlabel(r"$z$")
```

```python
    pl.savefig("part3.pdf")


def part4():
    lamrange = [2800, 8700]

    # The chi-by-eye temperatures.
    temps = [4500, 5200, 4750, 5200, 7300]
    amplitudes = np.zeros(len(temps), dtype=float)
    distances = np.zeros(len(temps), dtype=float)

    # Set up the figure.
    fig = pl.figure(figsize=(6, 12))
    fig.subplots_adjust(left=0.04, bottom=0.07, right=0.96, top=0.95,
            hspace=0.0)

    for i in range(len(temps)):
        ax = fig.add_subplot(len(temps), 1, i + 1)

        # Load the data and plot it.
        loglam, flux, z0 = load_spectrum("data/part4/{0}.fits".format(i + 1))
        lam, fsmooth = plot_spectrum(loglam, flux, z=z0, ax=ax, smooth=4)

        # Compute the redshift distance.
        distances[i] = z0 * 3e5 / 70.0  # Mpc

        # Get these y limits.
        ylim = ax.get_ylim()

        # Plot the unsmoothed spectrum.
        plot_spectrum(loglam, flux, z=z0, ax=ax, alpha=0.3)

        # Plot the black body "fit".
        bb = black_body(lam, temps[i])
        amplitudes[i] = fsmooth[np.argmax(bb)] / np.max(bb)
        ax.plot(lam, bb * amplitudes[i], "k", lw=3,
                alpha=0.8)

        # Plot the spectral lines.
        for n, (k, l, yoff) in enumerate(emission_lines):
            ax.axvline(l, ls="dotted", color="k")
            if i == 0:
                yoff += 8
                ax.annotate("$\mathrm{{{0}}}$".format(k),
                        xy=[l, ylim[-1]], xycoords="data",
                        xytext=[0, yoff],
                        textcoords="offset points",
                        ha="center", size=12)

        # Label the temperature.
        ax.annotate(r"${0}$ K".format(temps[i]), xy=[0, 1],
                xycoords="axes fraction", va="top",
                xytext=[10, -10], textcoords="offset points")

        # Clean up the axis.
        ax.set_ylim(ylim)
        ax.set_xlim(lamrange)
        ax.yaxis.set_major_locator(MaxNLocator(4))
        ax.set_yticklabels([])

        if i < len(temps) - 1:
            ax.set_xticklabels([])
        else:
            ax.set_xlabel(r"$\lambda \, (\mathrm{\AA})$")

    fig.savefig("part4.pdf")

    # Part 5 starts here.
    sigma = 5.6704e-5  # erg / s / cm^2 / K^4
    lamgrid = 10 ** np.linspace(2, 10, 50000)
    for i in range(len(temps)):
        bb = black_body(lamgrid, temps[i])
```

```
        flux = amplitudes[i] * simps(bb, x=lamgrid)
        D = distances[i]  # in Mpc
        print(D * np.sqrt(flux / sigma / temps[i] ** 4))


if __name__ == "__main__":
    part4()
```