Atmospheric conditions

FK8029 - Computational Physics

Andreas Evensen

Department of Physics
Stockholm University
Sweden

April 4, 2024

Contents

1 Introduction

Why does planets look the way they do? Why does life exist on certain planets, and not on others? How can life thrive on a planet such as Earth but not on another planet such as Venus? There are many questions we can ask ourselves in pursuit of the answers, however, one of the most fundamental answers to these questions is the atmosphere.

Life exists due to an atmosphere, and our atmosphere is what makes life possible on Earth. The atmosphere keeps the planet warm enough such that liquid water can exist, and it also protects us from hazardous radiation. Thus, in this report we will investigate the atmospheric properties of Earth via a simple radiation balance model.

2 Theory & Method

The atmosphere is a rather complex system, and thus in this model we will simply the atmosphere into a series of cells/layers. The incoming solar radiation, which is both in the infra-red spectrum and visible spectrum, will be partially reflected when it encounters the atmosphere. Some will be transmitted through each cell, whilst some will be attenuated by the atmospheric cells. Each cell will then emit infra-red radiation, which will be attenuated in the same manner as the incoming solar radiation. A schematic of the model is shown below[1].

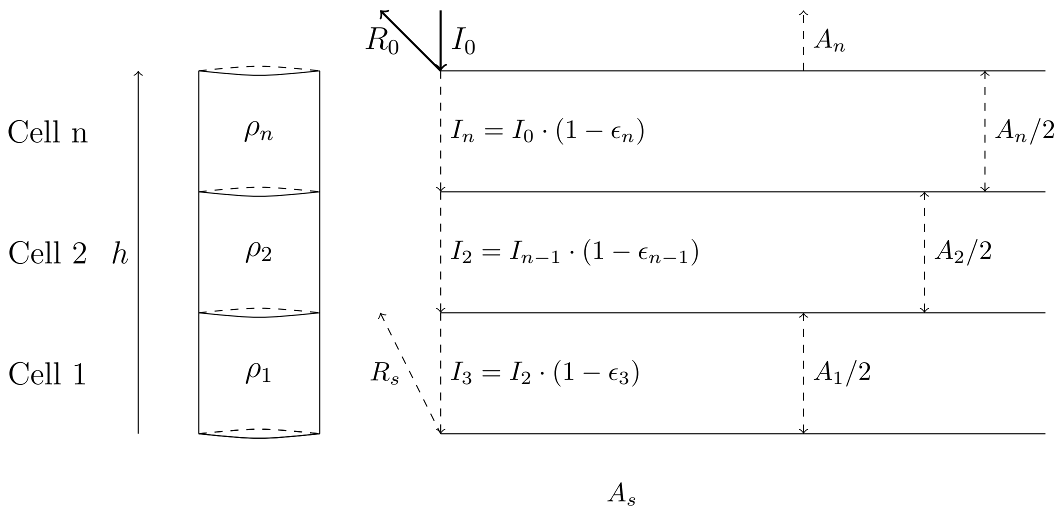


Figure 1: Schematic of the model

The densities in each layer is derived from barometric formula, and is thus given by:

$$\rho(z) = \rho_0 \cdot \exp\left[\frac{-gMz}{RT_0}\right], \tag{1}$$

where z is the height above sea level, ρ_0 is the density at sea level, g is the acceleration due to gravity, M is the molar mass of the atmosphere, R is the ideal gas constant, and T_0 is the temperature at sea level. The pressure is given by the same exponential formula but by replacing the constant ρ_0 by the pressure at sea level, P_0. From there, we can derive the cross-section for the visible and infra-red radiation as follows:

$$\sigma^{vis}(z) = \frac{\alpha_{vis}}{\rho(0)},$$
$$\sigma^{inf}(z) = \frac{\alpha_{inf}}{\rho(0)},$$

where α_{vis} and α_{inf} are scaling factors. The model can be solved iteratively, where radiation balance requires that the incoming solar radiation is equal, and its reflection is equal to the outgoing radiation, i.e. $I_0 - \sum_i R_i = A_n$, where A_n is the outgoing radiation from the last cell/space. The model can thus be described by the following equations:

$$T_i^\beta = \left(T_{i+1}^\beta + \delta_{\beta,inf}\frac{E_{i+1}}{2}\right) \cdot \epsilon_i^\beta \tag{2}$$

$$K_i = \left(K_{i-1} + \frac{E_{i-1}}{2}\right) \cdot \epsilon_i^{inf}, \tag{3}$$

$$E_i = \left(\frac{E_{i+1} - E_{i-1}}{2} + K_{i-1} + T_{i+1}^{inf}\right) \cdot \left(1 - \epsilon_i^{inf}\right) + T_{i+1}^{vis}\left(1 - \epsilon_i^{vis}\right). \tag{4}$$

In the above equations T is the transmitted radiation K is the outgoing radiation and E is the accumulated energy in the cell, in the notation above β is either vis or inf. Note that we can divide the outgoing radiation into two parts, one part that is the reflected radiation in the visible spectrum, and one part that is the emitted radiation in the infra-red spectrum.

Moreover, in each cell the emitted energy is equal to the absorbed energy, and we assume that each cell emits with equal probability in both directions. This in total leads to a set of seven equations of which must be solved. The coefficients ϵ_i^β are exponential decaying coefficients that describe the transmission in each cell i, and are defined by:

$$\epsilon_i^\beta = \exp\left[-\sigma_i^\beta \rho_i \Delta z\right],$$

where again β is either vis or inf. The transmission in each cell then corresponds to the absorption in that cell, i.e. $T_{i+1}^\beta(1-\epsilon_i^\beta)$, which is what is depicted in the above schematic 1.

From the above set of equations, it's possible to find the temperature of each layer by Stefan-Boltzman's law:

$$F = \sigma T^4, \tag{5}$$

where σ is a constant and T is the temperature. The flux, F, the net radiation upwards in the cell.

3 Result & Discussion

The above theory was implemented in a Python script. Below is a table showing the various initial parameters used to solve the system of equations above, eq (2) – (4).

Table 1: Atmosphere parameters

	P_0 [kPa]	T_0 [K]	g [m/s]	z [km]	I_0 [W/m^2]
Earth	101.3	288	9.81	100	344

Using known parameters, such as the pressure at the surface and the gravitational acceleration at the surface, one derived the density and pressure at the different layers.

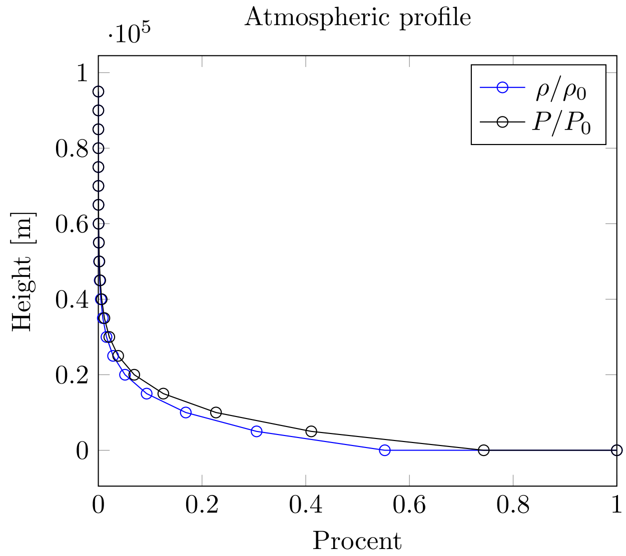


Figure 2: Atmospheric profile of the Earth

The above figure shows how the normalized pressure and density changes with height. As the height increases, the pressure and density decreases, which is expected. This indicates that the density and the pressure decreases significantly with height, as compared to temperature which decreases more slowly.

The surface temperature, is determined by the flux from the surface upwards, and thus is given by A_s is the above schematic 1, and using eq (5), we find the surface temperature as a function of number of cells, which is shown below in fig 3.

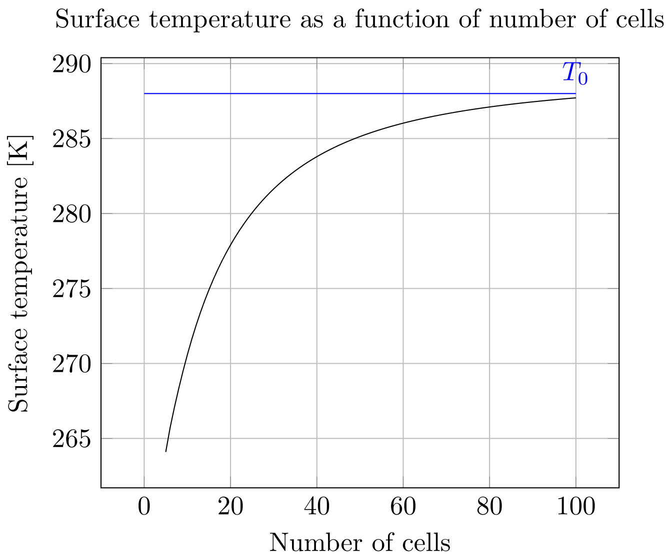


Figure 3: Surface temperature with scaling factors $\alpha_{vis} = 1 \cdot 10^{-4}$, and $\alpha_{inf} = 1.07 \cdot 10^{-3}$.

We see that the surface-temperature converges towards the true value with increasing number of cells. This implies that liquid water can exist, and thus as an extension – life. The temperature increases inversely proportional to the number of cells, and is thus not highly dependent on the number of cells. This is expected, as the attenuation in each cell is dependent on the height of the cell, and it's density; with an increasing number of cells, the height of each cell decreases, but the density is better resolved, and thus the attenuation is better resolved.

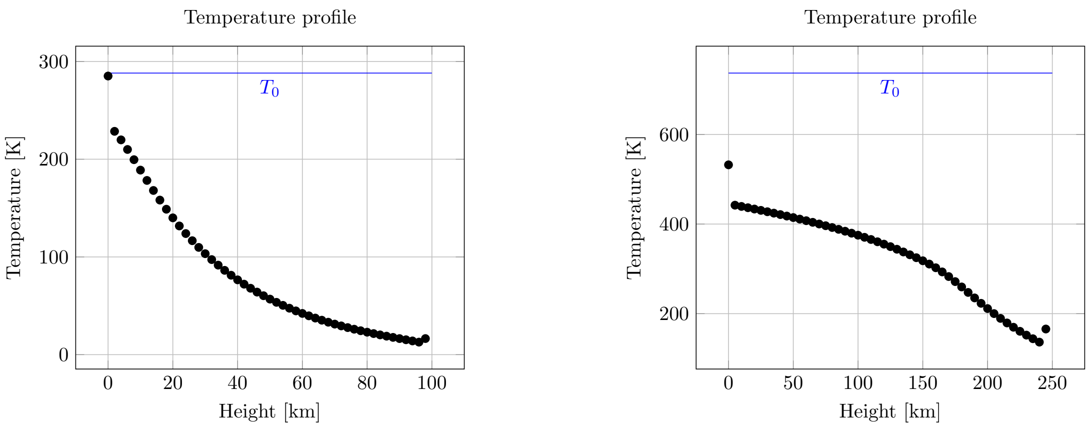
The temperature profile of the earth is shown below in fig 4a. As per contrast to the normalized pressure and density, the temperature decreases more slowly with height, as previously mentioned. This implied that approximations made with the barometric formula, eq (1), to some extent, is valid. Moreover, the temperature profile is plausible, as the temperature decreases with height and converges close to zero at the top of the atmosphere. This is reasonable, as the temperature of empty space is close to absolute zero, i.e. 0 K.

The model was also applied to the planet Venus, and the temperature profile of Venus is shown below in fig 4b. The following parameters were used to solve the system of equations for Venus[2]:

Table 2: Atmosphere parameters

	P_0 [MPa]	T_0 [K]	g [m/s]	z [km]	I_0 [W/m^2]
Venus	9.3	737	8.87	250	655.5

The scaling factors α_{vis} and α_{inf} were set to $1 \cdot 10^{-6}$ and $5 \cdot 10^{-1}$, respectively. This was done to better approximate Venus dense atmosphere. Although the attempts to take into account Venus dense atmosphere, the computed surface temperature of Venus is significantly lower than its true surface temperature. To better approximate, both the Earth's atmosphere and Venus Atmosphere, one would need to take into account the composition of the atmosphere, and the greenhouse effect. Although our model is simple, and underestimates the temperature of Venus, it still states that the temperature of Venus is significantly higher than the Earth. This in itself implies that life, as we know it, cannot exist on Venus, since the temperature is too high.



(a) Temperature profile of the Earth

(b) Temperature profile of the Venus

Figure 4: Temperature profile of the Earth 4a and Venus 4b.

4 Conclusion

The model was able to improve upon the results attained by Stefan-Boltzman's law, by taking into account that the radiation attenuates in the atmosphere and being redistributed. However, the model is very simple in that the radiation is banded into two categories, visible and infra-red light, instead of being banded into a 'continuous' spectrum. This would allow for a more accurate representation of the atmosphere, since the different contents of the atmosphere would absorb different wavelengths of radiation. Moreover, this model does not take into account non-radiant heat transfer, such as convection which is a significant factor in the atmosphere.

The model was solved iteratively, Appendix 1, where difficulties arose in the implementation of the model. The system of equations, eq (2) – (4), were not trivial to solve in the sense of being solved 'iteratively'. Nevertheless, the equations were solved using a simple iterative method, and the results were plausible, with room for improvement.

References

[1] R. T. Pierrehumbert, "Infrared radiation and planetary temperature", Physics Today **64**, 33–38 (2011).

[2] Wikipedia contributors, *Atmosphere of venus — Wikipedia, the free encyclopedia*, [Online; accessed 4-April-2024], 2024.

5 Appendix

```python
import numpy as np
from dataclasses import dataclass
import os
import sys

sys.setrecursionlimit(10000)

# Constants
R = 8.3144598
STEFANBOLTZMAN = 5.67 * 10**(-8)


@dataclass
class Atmosphere:
    """
        A dataclass representing the atmosphere of a planet

        params:
            incomingFlux: float - the incoming flux in W/m^2
            height: float - the height of the atmosphere in meters
            surfacePressure: float - the surface pressure in pascals
            gravity: float - the gravity at the surface in m/s^2
            groundTemperature: float - the temperature at the surface in Kelvin
            deltaHeight: float - the height of each cell in the atmosphere
            numberOfCells: int - the number of cells in the atmosphere
            surfaceDensity: float - the density at the surface in kg/m^3
            mAir: float - the molar mass of air in kg/mol
            planetAlbedo: float - the albedo of the planet

        returns:
            Atmosphere - the atmosphere of the planet
    """
    incomingFlux: float
    height: float
    surfacePressure: float
    gravity: float
    groundTemperature: float
    deltaHeight: float
    numberOfCells: int
    surfaceDensity: float
    mAir: float = 0.0289644
    planetAlbedo: float = 0.04


def computePressure(h: float) -> float:
    """
        Computes the pressure given the height

        params:
            h: float - height in meters
        returns:
            float - pressure in pascals
    """
    return atm.surfacePressure * np.exp(-atm.gravity * h * atm.mAir/ (R * atm.groundTemperature))

def computeDensity(atm: Atmosphere, h: float) -> float:
    """
        Computes the density given the height

        params:
            atm: Atmosphere - the atmosphere of the planet
            h: float - height in meters
        returns:
            float - Density in kg/m^3
```

```
    """
    return atm.surfaceDensity * np.exp( -atm.gravity * atm.mAir * (h + atm.deltaHeight) / (R * atm.groundTemperature))




def iterate(iteration: int, absorbed, transmittedDownInf, transmittedDownVis, transmittedUpVis, transmittedUpInf, emittedUp, emittedDown, cellFlux: np.ndarray) -> np.ndarray:
    """
        Iterates through the cells to calculate the energy balance, via recursion

        params:
            iteration: int - the current iteration
            absorbed: np.ndarray - the absorbed energy in each cell at the current iteration
            transmittedDownInf: np.ndarray - the transmitted IR radiation going down in each cell at the current iteration
            transmittedDownVis: np.ndarray - the transmitted visible radiation going down in each cell at the current iteration
            transmittedUpVis: np.ndarray - the transmitted visible radiation going up in each cell at the current iteration
            transmittedUpInf: np.ndarray - the transmitted IR radiation going up in each cell at the current iteration
            emittedUp: np.ndarray - the emitted energy in each cell going up at the current iteration
            emittedDown: np.ndarray - the emitted energy in each cell going down at the current iteration
            cellFlux: np.ndarray - the cell flux at the current iteration, used for radiation balance

        returns:
            np.ndarray - the cell flux

        Error:
            Exception - if the solution does not converge within 5000 iterations
    """

    # Top Down
    for i in range(len(absorbed) - 2, 0, -1):
        #Visible contribution
        transmittedDownVis[ i ] += transmittedDownVis[i + 1] * epsilonV[ i ]

        #IR contribution
        transmittedDownInf[ i ] += (transmittedDownInf[i + 1] + emittedDown[i + 1]) * epsilonI[ i ]

        # Attenuated in cell
        absorbed[ i ] += transmittedDownVis[i + 1] * (1 - epsilonV[ i ]) + (transmittedDownInf[i + 1] + emittedDown[i + 1]) * ( 1 - epsilonI[ i ] )

        # Emitting IR energy 50% goes up and 50% goes down
        emittedDown[ i ] += absorbed[ i ] / 2
        emittedUp[ i ] += absorbed[ i ] / 2

        # Keeping track of the cells
        cellFlux[ i ] += emittedUp[ i ] + transmittedUpInf[i] + transmittedUpVis[i] - (emittedDown[i + 1] + transmittedDownInf[ i + 1] + transmittedDownVis[i + 1])  # new

        # Reset, we have 'used' the energies in this iteration
        transmittedDownVis[i + 1] = 0.0
        transmittedDownInf[i + 1] = 0.0
        emittedDown[i + 1] = 0.0
        absorbed[ i ] = 0.0

```

```
    # Ground

    # Visible contribution from reflection
    transmittedUpVis[ 0 ] = transmittedDownVis[ 1 ] * atm.planetAlbedo

    # Absorbed energy
    absorbed[ 0 ] = transmittedDownVis[ 1 ] - transmittedUpVis[ 0 ] + emittedDown[
    1 ] + transmittedDownInf[ 1 ]

    # Kirchhoff's law
    emittedUp[ 0 ] = absorbed[ 0 ]

    # fSurface += cells[0].absorbed
    cellFlux[ 0 ] += absorbed[ 0 ]

    # Reset
    absorbed[ 0 ] = 0.0
    transmittedDownVis[ 1 ] = 0.0
    transmittedDownInf[ 1 ] = 0.0
    emittedDown[ 1 ] = 0.0

    # Bottom up
    for i in range(1, len(absorbed) - 1):
        # Visible contribution
        transmittedUpVis[ i ] += transmittedUpVis[i - 1] * epsilonV[ i ]

        # IR contribution
        transmittedUpInf[ i ] += (transmittedUpInf[i - 1] + emittedUp[i - 1]) *
    epsilonI[ i ]

        # Absorbed energy
        absorbed[ i ] += transmittedUpVis[i - 1] * (1 - epsilonV[ i ]) + (
    transmittedUpInf[i - 1] + emittedUp[i - 1]) * (1 - epsilonI[ i ])

        # Emitting IR energy 50% goes up and 50% goes down
        emittedDown[ i ] += absorbed[ i ] / 2
        emittedUp[ i ] += absorbed[ i ] / 2

        # Keeping track of the cells
        cellFlux[ i ] += emittedUp[ i ] + transmittedUpVis[ i ] + transmittedUpInf[
     i ] - (emittedDown[i + 1] + transmittedDownInf[ i ] + transmittedDownVis[ i ])
      # new

        # Reset, we have 'used' the energies in this iteration
        transmittedUpVis[i - 1] = 0.0
        transmittedUpInf[i - 1] = 0.0
        emittedUp[i - 1] = 0.0
        absorbed[ i ] = 0.0


    # Radiation into space
    absorbed[ -1 ] = transmittedUpVis[ -2 ] + transmittedUpInf[ -2 ] + emittedUp[
    -2 ]

    # Reset, we have 'used' the energies in this iteration
    transmittedUpVis[ -2 ] = 0.0
    transmittedUpInf[ -2 ] = 0.0
    emittedUp[ -2 ] = 0.0

    cellFlux[ -1 ] += absorbed[ -1 ] # Emitted energy into space

    if iteration > 5000:
        raise Exception("Did not converge within 5000 iterations")

    if abs(atm.incomingFlux * 0.7 - cellFlux[-1])> 0.001: # Radiation balance
    condition
        return iterate(iteration + 1, absorbed, transmittedDownInf,
```

```
    transmittedDownVis, transmittedUpVis, transmittedUpInf, emittedUp, emittedDown, cellFlux)
    else:
        return cellFlux


if __name__ == '__main__':
    # Global variables
    global atm
    global epsilonI
    global epsilonV

    # Initialize the atmosphere
    numberOfCells: int = 50

    planet = input("Enter the planet name: ")

    if planet == "earth":
        height: int = 100_000 # m

        atm = Atmosphere(
            incomingFlux = 340, #  W/m^2
            height = height, # m
            surfacePressure = 101_325, # Pa
            gravity = 9.81, # m/s^2
            groundTemperature = 288, # K
            deltaHeight = height / (numberOfCells), # m
            numberOfCells = numberOfCells,
            surfaceDensity = 1.225   # kg/m^3
        )

        # Initialize the data that we need to keep track of
        absorbed = np.zeros(numberOfCells + 1, dtype = float)     # Used to keep track of the absorbed energy in each cell, for radiation balance and temperature calculation
        transmittedDownVis = np.zeros(numberOfCells + 1, dtype = float)   # Used to keep track of the transmitted visible radition going down
        transmittedUpVis = np.zeros(numberOfCells + 1, dtype = float)     # Used to keep track of the transmitted visible radiation going up
        transmittedDownInf = np.zeros(numberOfCells + 1, dtype = float)   # Used to keep track of the transmitted ir radiation going down
        transmittedUpInf = np.zeros(numberOfCells + 1, dtype = float)     # Used to keep track of the transmitted ir radiation going up
        emittedUp = np.zeros(numberOfCells + 1, dtype = float)    # Used to keep track of the emitted energy in each cell going up
        emittedDown = np.zeros(numberOfCells + 1, dtype = float)  # Used to keep track of the emitted energy in each cell going down

        # Compute the attenuation coefficients for each layer
        densities = np.zeros(numberOfCells + 1, dtype = float)
        densities[0] = atm.surfaceDensity
        pressures = np.zeros(numberOfCells + 1, dtype = float)
        pressures[0] = atm.surfacePressure

        #fileLayer = open('layerInfo.dat', 'w') # Uncomment the file writing for densities and pressures plot
        #fileLayer.write('z\trho\tp\n')
        #fileLayer.write('0\t{}\t{}\n'.format(densities[0]/densities[0], pressures[0]/pressures[0]))
        Z: np.ndarray = np.zeros(numberOfCells + 1, dtype = float)
        for i in range(0, numberOfCells):
            z = i * atm.deltaHeight
            Z[i + 1] = z
            densities[i + 1] = computeDensity(atm, z)
            pressures[i + 1] = computePressure(z + atm.deltaHeight / 2)
            #fileLayer.write('{}\t{}\t{}\n'.format(z, densities[i+1]/densities[0], pressures[i+1]/pressures[0]))
```

```
        #fileLayer.close()

        # Attenuation for visible and IR radiation
        attV: float = 1e-4 / densities[0]
        attInf: float = 1.07e-3 / densities[0]

        # Attenuation for visible and IR radiation
        epsilonV: np.ndarray = np.exp( -attV  * densities * atm.deltaHeight)
        epsilonI: np.ndarray = np.exp( -attInf  * densities * atm.deltaHeight)

        # Initizalize the incoming radiation
        inc: float = atm.incomingFlux * 0.7     # 30% of the incoming radiation is reflected
        transmittedDownVis[ -1 ] = inc * 0.25   # 25 % of the incoming is visible
        transmittedDownInf[ -1 ] = inc * 0.75   # 75 % of the incoming is IR

        cellFlux = np.zeros(numberOfCells + 1, dtype=float) # Used to keep track of the emitted energy in each cell, for radiation balance and temperature calculation

    elif planet == "venus":
        height: int = 250_000 # m

        atm = Atmosphere(
            incomingFlux = 2622/4, #  W/m^2
            height = height, # m
            surfacePressure = 9.3 * 10*6, # Pa
            gravity = 8.87, # m/s^2
            groundTemperature = 737, # K
            deltaHeight = height / (numberOfCells), # m
            numberOfCells = numberOfCells,
            surfaceDensity = 67 # kg/m^3
        )

        # Initialize the data that we need to keep track of
        absorbed = np.zeros(numberOfCells + 1, dtype = float)     # Used to keep track of the absorbed energy in each cell, for radiation balance and temperature calculation
        transmittedDownVis = np.zeros(numberOfCells + 1, dtype = float)   # Used to keep track of the transmitted visible radition going down
        transmittedUpVis = np.zeros(numberOfCells + 1, dtype = float)     # Used to keep track of the transmitted visible radiation going up
        transmittedDownInf = np.zeros(numberOfCells + 1, dtype = float)   # Used to keep track of the transmitted ir radiation going down
        transmittedUpInf = np.zeros(numberOfCells + 1, dtype = float)     # Used to keep track of the transmitted ir radiation going up
        emittedUp = np.zeros(numberOfCells + 1, dtype = float)    # Used to keep track of the emitted energy in each cell going up
        emittedDown = np.zeros(numberOfCells + 1, dtype = float)  # Used to keep track of the emitted energy in each cell going down

        # Compute the attenuation coefficients for each layer
        densities = np.zeros(numberOfCells + 1, dtype = float)
        densities[0] = atm.surfaceDensity
        pressures = np.zeros(numberOfCells + 1, dtype = float)
        pressures[0] = atm.surfacePressure

        #fileLayer = open('layerInfo.dat', 'w') # Uncomment the file writing for densities and pressures plot
        #fileLayer.write('z\trho\tp\n')
        #fileLayer.write('0\t{}\t{}\n'.format(densities[0]/densities[0], pressures[0]/pressures[0]))
        Z: np.ndarray = np.zeros(numberOfCells + 1, dtype = float)
        for i in range(0, numberOfCells):
            z = i * atm.deltaHeight
            Z[i + 1] = z
```

```
            densities[i + 1] = computeDensity(atm, z)
            pressures[i + 1] = computePressure(z + atm.deltaHeight / 2)
            #fileLayer.write('{}\t{}\t{}\n'.format(z, densities[i+1]/densities[0], pressures[i+1]/pressures[0]))

        #fileLayer.close()

        # Attenuation for visible and IR radiation
        attV: float = 1e-6 / densities[0]
        attInf: float = 5e-1 / densities[0]

        # Attenuation for visible and IR radiation
        epsilonV: np.ndarray = np.exp( -attV  * densities * atm.deltaHeight)
        epsilonI: np.ndarray = np.exp( -attInf  * densities * atm.deltaHeight)
        print(densities)

        # Initizalize the incoming radiation
        inc: float = atm.incomingFlux * 0.7     # 30% of the incoming radiation is reflected
        transmittedDownVis[ -1 ] = inc * 0.25   # 25 % of the incoming is visible
        transmittedDownInf[ -1 ] = inc * 0.75   # 75 % of the incoming is IR

        cellFlux = np.zeros(numberOfCells + 1, dtype=float) # Used to keep track of the emitted energy in each cell, for radiation balance and temperature calculation
    else:
        print("Invalid planet name")
        sys.exit(1)


    # Iterate until the solution converges or the maximum number of iterations is reached
    flux = iterate(0, absorbed, transmittedDownInf, transmittedDownVis, transmittedUpVis, transmittedUpInf, emittedUp, emittedDown, cellFlux)

    # Convert the flux to temperature via the Stefan-Boltzman law
    temperature = ( flux / STEFANBOLTZMAN ) ** 0.25

    file = open("testoutput{}.dat".format(planet), "w") # For the temperature at different heights
    for i in range( len( flux ) - 1 ):
        current_height = i * atm.deltaHeight / 1000 # Convert to km
        file.write(f"{current_height} {temperature[i]}\n")

    file.close()

    #file = open("ground_temperature2.dat", "a+")
    #file.write('{}\t{}\n'.format(numberOfCells, temperature[0]))
    #file.close()

    print("-"*50)
    print("Temperatures [K]")
    print(temperature)
```

Listing 1: Code for the model