import numpy as np

from scipy.integrate import odeint

import matplotlib.pyplot as plt

# Constants

G = 6.67408e-11 # Gravitational constant (m^3 kg^-1 s^-2)

c = 2.99792458e8 # Speed of light (m/s)

a\_radiation = 7.5657e-16 # Radiation constant (J m^-3 K^-4)

R\_gas = 8.3144621 # Universal gas constant (J mol^-1 K^-1)

sigma = 5.670374419e-8 # Stefan-Boltzmann constant (W m^-2 K^-4)

# Stellar Structure Equations

def hydrostatic\_equilibrium(rho: float, M: float, r: float) -> float:

"""

Compute the pressure gradient due to hydrostatic equilibrium at a given radius.

Args:

rho (float): Density at radius r (kg/m^3).

M (float): Mass enclosed within radius r (kg).

r (float): Radius from the center of the star (m).

Returns:

float: Pressure gradient dP/dr (Pa/m).

"""

return -G \* M \* rho / r\*\*2

def energy\_generation\_pp\_chain(T: float, rho: float) -> float:

"""

Compute the energy generation rate per unit mass for the proton-proton chain.

Args:

T (float): Temperature (K).

rho (float): Density (kg/m^3).

Returns:

float: Energy generation rate (W/kg).

"""

# Constants for the PP chain reaction rate (approximate)

epsilon\_pp = 1.07e-7 \* (rho / 1e5) \* (T / 1e7)\*\*4

return epsilon\_pp

def energy\_generation\_cno\_cycle(T: float, rho: float) -> float:

"""

Compute the energy generation rate per unit mass for the CNO cycle.

Args:

T (float): Temperature (K).

rho (float): Density (kg/m^3).

Returns:

float: Energy generation rate (W/kg).

"""

# Constants for the CNO cycle reaction rate (approximate)

epsilon\_cno = 8.24e-26 \* (rho / 1e5) \* (T / 1e7)\*\*19

return epsilon\_cno

def radiative\_gradient(L: float, rho: float, T: float, r: float, kappa: float) -> float:

"""

Compute the temperature gradient due to radiative transfer.

Args:

L (float): Luminosity enclosed within radius r (W).

rho (float): Density (kg/m^3).

T (float): Temperature (K).

r (float): Radius from the center of the star (m).

kappa (float): Opacity (m^2/kg).

Returns:

float: Temperature gradient dT/dr (K/m).

"""

return - (3 \* kappa \* rho \* L) / (16 \* np.pi \* a\_radiation \* c \* T\*\*3 \* r\*\*2)

def equation\_of\_state(rho: float, mu: float, T: float) -> float:

"""

Compute the pressure using the ideal gas law.

Args:

rho (float): Density (kg/m^3).

mu (float): Mean molecular weight.

T (float): Temperature (K).

Returns:

float: Pressure (Pa).

"""

return (rho / mu) \* R\_gas \* T

# Additional functions for mass continuity and luminosity gradient can be added here.

# Example usage and integration routines would follow.

def mass\_continuity(rho: float, r: float) -> float:

"""

Compute the mass gradient at a given radius.

Args:

rho (float): Density at radius r (kg/m^3).

r (float): Radius from the center of the star (m).

Returns:

float: Mass gradient dM/dr (kg/m).

"""

return 4 \* np.pi \* r\*\*2 \* rho

def luminosity\_gradient(rho: float, epsilon: float, r: float) -> float:

"""

Compute the luminosity gradient at a given radius.

Args:

rho (float): Density at radius r (kg/m^3).

epsilon (float): Energy generation rate per unit mass (W/kg).

r (float): Radius from the center of the star (m).

Returns:

float: Luminosity gradient dL/dr (W/m).

"""

return 4 \* np.pi \* r\*\*2 \* rho \* epsilon

def stellar\_structure(y, r, mu):

"""

System of differential equations for stellar structure.

Args:

y (array): Current values [P, M, L, T] at radius r.

r (float): Current radius (m).

mu (float): Mean molecular weight.

Returns:

dydr (array): Derivatives [dP/dr, dM/dr, dL/dr, dT/dr].

"""

P, M, L, T = y

rho = (mu \* P) / (R\_gas \* T) # Using the ideal gas law rearranged

# Total energy generation rate per unit mass

epsilon = energy\_generation\_pp\_chain(T, rho) + energy\_generation\_cno\_cycle(T, rho)

# Opacity (for simplicity, use a constant or implement an opacity law)

kappa = 0.034 # Opacity in m^2/kg (example value)

# Differential equations

dPdr = hydrostatic\_equilibrium(rho, M, r)

dMdr = mass\_continuity(rho, r)

dLdr = luminosity\_gradient(rho, epsilon, r)

dTdr = radiative\_gradient(L, rho, T, r, kappa)

return [dPdr, dMdr, dLdr, dTdr]

# Central conditions (approximations for a solar-mass star)

P\_c = 1.3e16 # Central pressure (Pa)

T\_c = 1.57e7 # Central temperature (K)

mu = 0.61 # Mean molecular weight for fully ionized gas

rho\_c = (mu \* P\_c) / (R\_gas \* T\_c) # Central density (kg/m^3)

# Initial state vector

y0 = [P\_c, 0, 0, T\_c]

r0 = 1e-8 # Starting radius (m)

r\_max = 7e8 # Approximate radius of the star (m)

r\_points = np.linspace(r0, r\_max, 1000)

# Solve the differential equations

from scipy.integrate import odeint

solution = odeint(stellar\_structure, y0, r\_points, args=(mu,)) r0 = 1e-8 # Starting radius (m)

r\_max = 7e8 # Approximate radius of the star (m)

r\_points = np.linspace(r0, r\_max, 1000)

# Solve the differential equations

from scipy.integrate import odeint

solution = odeint(stellar\_structure, y0, r\_points, args=(mu,))

P = solution[:, 0]

M = solution[:, 1]

L = solution[:, 2]

T = solution[:, 3]

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))

# Pressure vs. Radius

plt.subplot(2, 2, 1)

plt.plot(r\_points, P)

plt.xlabel('Radius (m)')

plt.ylabel('Pressure (Pa)')

plt.title('Pressure Profile')

# Mass vs. Radius

plt.subplot(2, 2, 2)

plt.plot(r\_points, M)

plt.xlabel('Radius (m)')

plt.ylabel('Mass (kg)')

plt.title('Mass Profile')

# Luminosity vs. Radius

plt.subplot(2, 2, 3)

plt.plot(r\_points, L)

plt.xlabel('Radius (m)')

plt.ylabel('Luminosity (W)')

plt.title('Luminosity Profile')

# Temperature vs. Radius

plt.subplot(2, 2, 4)

plt.plot(r\_points, T)

plt.xlabel('Radius (m)')

plt.ylabel('Temperature (K)')

plt.title('Temperature Profile')

plt.tight\_layout()

plt.show()

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))

# Pressure vs. Radius

plt.subplot(2, 2, 1)

plt.plot(r\_points, P)

plt.xlabel('Radius (m)')

plt.ylabel('Pressure (Pa)')

plt.title('Pressure Profile')

# Mass vs. Radius

plt.subplot(2, 2, 2)

plt.plot(r\_points, M)

plt.xlabel('Radius (m)')

plt.ylabel('Mass (kg)')

plt.title('Mass Profile')

# Luminosity vs. Radius

plt.subplot(2, 2, 3)

plt.plot(r\_points, L)

plt.xlabel('Radius (m)')

plt.ylabel('Luminosity (W)')

plt.title('Luminosity Profile')

# Temperature vs. Radius

plt.subplot(2, 2, 4)

plt.plot(r\_points, T)

plt.xlabel('Radius (m)')

plt.ylabel('Temperature (K)')

plt.title('Temperature Profile')

plt.tight\_layout()

plt.show()

def kramers\_opacity(rho: float, T: float) -> float:

"""

Calculates the opacity using Kramers' law.

Args:

rho (float): Density (kg/m^3).

T (float): Temperature (K).

Returns:

float: Opacity (m^2/kg).

"""

return 4e25 \* (rho) \* T\*\*(-3.5)

def total\_pressure(rho: float, T: float, mu: float) -> float:

"""

Calculates the total pressure including gas and radiation pressure.

Args:

rho (float): Density (kg/m^3).

T (float): Temperature (K).

mu (float): Mean molecular weight.

Returns:

float: Total pressure (Pa).

"""

P\_gas = equation\_of\_state(rho, mu, T)

P\_rad = (1/3) \* a\_radiation \* T\*\*4

return P\_gas + P\_rad