"Star Soft" Documentation

Ponomarev Daniil

August 2024

Contents

1 Introduction:

"Star Soft" is a python modeling software that allows you to set up and run simulations of stellar cores, including evolution of mass fractions of elements, electron and positron concentrations, temperature of the core and outside neutrino flux.

2 Measuresment system:

Most of the modeling uses measurement system called "SA - Star Astro" with core units being: gramm, year, santimeter, mole, Kelvin, elemental charge, Gauss.

Only exclusions are "Temperature_change.py" where Si is used and method "Resonant_reactions_matrix_update(<...>)" in "Resonant_reactions.py" where Si with MeV for Breit_Wigner's formula are used.

Secondary units are called SA[name], futher are ratios between them and there SI counterparts.

- SAJoul = 10^{22} Joul
- SAWatt = $3,154 \times 10^{29}$ Watt
- SAPascal = 10^{16} Pascal

3 Requiered external libraries:

For stable forkflow Star Soft 4.9 requers a list of external Python libraries:

- matplotlib;
- scipy;
- numpy.

4 Components and structure:

Current version 4.9 of "Star Soft" consists of 17 components, including: main script, 11 modules, output visualiser, 2 input text files and 2 output text files.

- Main script: Folder name: "Star_Soft_main". Performs general management of modules and variables, parcing of general simulation settings, some minor calculations and turning text modeling output into graphs using mathplotlib.pyplot at the end.
- Nuclear decay reactions module: Folder name: "Decay_reactions". Contains a pack of methods for reading and storing information about nuclear decay reactions and calculating there rates.
- "Simulation_output.txt" parcer: Folder name: "Parcer". A script that turns text modeling output into graphs using mathplotlib.pyplot.
- Differential equations solving module: Folder name: "Differentials_solving_methods". Contains methods (in current version 1) for solving systems of differential equations that describe rate of concentrations of components change.
- Display module: Folder name: "Displays". Contains methods for displaying states of different parts of modeling in a comfortable to read way.
- Electron / positron maganing module: Folder name: "Electron_positron_operations". Contains methods for: storing information about reaction‘s effect on electron / positron concentrations, electron / positron initial concentration count, rates of there concentration change.
- Wolrd constants initialization module: Folder name: "World_constants_initializer". Constains a method to parse input file "World_constants.txt" and turn each line constaining a constant info in format "[Name] = [Value] [Measure units]" into global constant with coresponding name and value.

- Elements technical operations module: Folder name: "Elements_opertaions". Contains many technical methods for parcing parts of "Simulation_input.txt" aand inside data as well as some calculating methods.

- Neutrino module: Folder name: "Neutrino_spectrum": Contains methods for: storing information about reaction‘s neutrino emission and calculation neutrino flux / energy spectrum based on rates of reactions.

- Processes module: Folder name: "Processes": Contains a pack of methods for reading and storing information about all types of reactions, calculating and updating there rates, based on methods from article [1] about nuclear reactions rates.

- Resonant reactions module: Folder name: "Resonant_reactions". Contains a pack of methods for reading and storing information about resonant reaction and calculating there cross-sections and rates.

- Temperature module: Folder name: "Temperature_change". Contains a method that calculates energy impacts and loses of ther star, so change of the temperature of its core.

- Unresonant reactions module: Folder name: "Unresonant_reactions". Contains a pack of methods for reading and storing information about unresonant reactions and calculating there rates.

- Configs input file: Folder name: "Simulation_input". Main configuration file of the modeling. Contains parametrs of simulated star as well as all of the reactions and constants that determine there rates.

- Wolrd constants file: Folder name: "World_constants". Contains class with all world constants from which they can be called using there name.

- Elements data file: Folder name: "Elements_data". Contains list of all used elements (there cores) as well as there mass, charge and spin of them.

- Data output file: Folder name: "Simulation_output". Main output file that is used to store modeling data in text format. Not designed to be read manually.

- Logs life: Folder name: "Logs". Secondary output file that contains logs of simulation, designed for troubleshooting.

5 Decay_reactions:

- **Decay_reactions_matrix_create(Elements, R, T, p0, T0)**:

Input variables: Elements [array] - copy of "Elements_data.txt"; R [float] - universal gas constant in SI; T [float] - currect core temperature in Kelvins; p0 [float] - normal atmospherical pressure in pascals; T0 [float] - normal temperature in Kelvin.

Description: Uses "Simulation_input" and "Elements_data.txt" to create a partially filled template for "Decay_reactions":

Decay_reactions for reaction with number i:

- Decay_reactions[0][i] - initial element;
- Decay_reactions[1][i] - mass of initial element;
- Decay_reactions[2][i] - charge of initial element;
- Decay_reactions[3][i] - spin of initial element;
- Decay_reactions[4][i] - first resulting element;
- Decay_reactions[5][i] - mass of first resulting element;
- Decay_reactions[6][i] - charge of first resulting element;
- Decay_reactions[7][i] - spin of first resulting element;
- Decay_reactions[8][i] - second resulting element;
- Decay_reactions[9][i] - mass of second first resulting element;
- Decay_reactions[10][i] - charge of second resulting element;
- Decay_reactions[11][i] - spin of second resulting element;
- Decay_reactions[12][i] - normal half-life of initial element;
- Decay_reactions[13][i] - constant of the reaction (determines dependence from temperature and density of the plasma);
- Decay_reactions[14][i] - type of the reaction (1 - alpha decay, 2 - beta decay (+/-));
- Decay_reactions[15][i] - energetic reaction effect;

- Decay_reactions[16][i] - normal temperature in Kelvin;
- Decay_reactions[17][i] - normal density of the substance from the initial element (as a gas);
- Decay_reactions[18][i] - modified half-life of the initial element;
- Decay_reactions[19][i] - number of reactions per cm^{-3} per s^{-1}.

.

Output variables: Decay_recationss [array] - decay reactions matrix.

- **Decay_reactions_matrix_update(Decay_reactions, ro, T)**:

Input variables: Decay_reactions [array] - decay reactions matrix; ro - current core density in g/cm^3 ; T [float] - currect core temperature in Kelvin.

Description: This method is supposed to adjust half-lifes of elemnts nucleus to current density and temperature of the core. I NOT USED AT SHALL NOT BE TOUCHED!.

Output variables: Decay_reactions [array] - decay reactions matrix.

- **Decay_reactions_count(Decay_reactions, Concentrations)**:

Input variables: Decay_reactions [array] - decay reactions matrix; Concentrations - [array] stores concentrations of elements in format [elements (navigation part)][concentrations] with navigation [1][Number of the element].

Description: Calculates rates of nuclear decay recations (not reactions change) based on formula for nucler decay:

If n is concentration of nuclei with half-life T, time is t, then:

$$n(t) = \frac{n_0}{2^{\frac{t}{T}}}$$

Then we can calculate the derivative of the concentration over time:

$$\dot{n} = \frac{dn}{dt} = \frac{-n_0 \cdot \ln(2)}{T \cdot 2^{\frac{t}{T}}}$$

And since on every step of calculation we act as t = 0, that can be transformed into:

$$\dot{n} = \frac{dn}{dt} = \frac{-n_0 \cdot \ln(2)}{T}$$

.

Output variables: Decay_reactions [array] - decay reactions matrix.

6 Differentials_solving_methods:

- **Courant_diff_solve(Unresonant_reactions, Resonant_reactions, Decay_reactions, Processes, Speed, Time, Smart, M_nuc, Concentrations, Mass_fractions, R_core, T, ro, Average_per_particle_weight, Electron_unresonant_reaction_effect, Positron_unresonant_reaction_effect, Electron_resonant_reaction_effect, Positron_resonant_reaction_effect, Electron_decay_reaction_effect, Positron_decay_reaction_effect, Electron_process_effect, Positron_process_effect, Elements_burning_time, Burning_speed, Elements, Electrons_concentration, Positrons_concentration, T_surface, R_star, i, R, N_a)**

  :

Input variables: Unresonant_reactions [array] - unresonant reactions matrix, Resonant_reactions [array] - resonant reactions matrix, Decay_reactions [array] - decay reactions matrix, Processes [array] - processes matrix, Speed [float] - Courant's number, Time [float] - virtual time from start of the modeling in years, Smart [bool] - parameter that determins if system can adjust Courant's number in the process (dosent work in current version), M_nuc [float] - atomic unit of mass in grams, Concentrations [array] - stores concentrations of elements in format [elements (navigation part)][concentrations] with navigation [1][Number of the element], Mass_fractions - [array] stores mass fractions of elements in format [elements (navigation part)][mass fractions] with navigation [1][Number of elements], R_core [float] - radius of the core of the star in cantimeters, T [float] - current core temperature in Kelvins, ro [float] - current core density in g/cm^3, Average_per_particle_weight [float] - average per particle molecular weight, Electron_unresonant_reaction_effect [array] - stores reaction's impacts on concentrations of electrons (how many reaction produces or consumes) in format [Reactions impact] with navigation [Number of read unresonant reaction], Positron_unresonant_reaction_effect [array] - stores reaction's impacts on concentrations of positrons (how many reaction produces or consumes) in format [Reactions impact] with navigation [Number of read unresonant reaction], Electron_resonant_reaction_effect [array] - stores reaction's impacts on concentrations of electrons (how many reaction produces or consumes) in format [Reactions impact] with navigation [Number of read resonant reaction], Positron_resonant_reaction_effect

[array] - stores reaction‘s impacts on concentrations of positrons (how many reaction produces or consumes) in format [Reactions impact] with navigation [Number of read resonant reaction], Electron_decay_reaction_effect [array] - stores reaction‘s impacts on concentrations of electrons (how many reaction produces or consumes) in format [Reactions impact] with navigation [Number of read decay reaction], Positron_decay_reaction_effect [array] - stores reaction‘s impacts on concentrations of positrons (how many reaction produces or consumes) in format [Reactions impact] with navigation [Number of read decay reaction], Electron_process_effect [array] - stores process‘ impacts on concentrations of electrons (how many process produces or consumes) in format [Process‘ impact] with navigation [Number of read process], Positron_process_effect [array] - stores process‘ impacts on concentrations of positrons (how many process produces or consumes) in format [Process‘ impact] with navigation [Number of read process], Elements_burning_time [array] - stores burning times of elements in format [Elements (navigation part)][Elements burning time] with navigation [1][Number of the element], Burning_speed [array] - stores burning times of elements in format [Elements (navigation part)][Elements burning speed] with navigation [1][Number of the element], Elements [array] - copy of "Elements_data.txt", Electrons_concentration - concentration of electrons, Positrons_concentration - concentration of positrons, T_surface - surface star temperature in Kelvins, R_star, i [int] - degrees of freedom, R [float] - universal gas constant in SI, N_a [float] - Avogadro‘s number in SI.

Description: Calculates rates of all recations using coresponding calculating methods for there types, turns there rates into rates of changes of concentrations of elements (burning speed for short):

$$\frac{dn_i}{dt} = -a \sum_{i=1}^{k} R_i$$

,

where for nuclei i with k reactions that include it where "a" is nuclear factor coresnonding to nuclei "i" (a is positive if nuclei i is in right side of nuclear equation and vice versa).

Then it calculates there burning times using there concentrations. If burning speed of an element is negative then there burning time is simply concentration divided by burning speed:

$$t_i = \frac{n_i}{\dot{n}_i}$$

, where t_i is burning time of nuclei "i", n_i is its current concentration and $\dot{n}$ is its bruning speed.

Else if burning speed of an element is positive:

$$t_i = \frac{n_{max_i} - n_i}{-\dot{n_i}}$$

, where everything is the same despite n_{max} is its maximum possible concentration of nuclei "i" which is calculated like:

$$n_{max} = \frac{\rho}{m_i \cdot M_{nuc}}$$

The same thing (excluding max concentration) is repeated for electrons and positrons and temperature of the core.

If burning speed of electrons is negative:

$$t_{e-} = \frac{n_{e-}}{-\dot{n_{e-}}}$$

otherwise:

$$t_{e-} = \infty$$

Same for positrons:

$$t_{e+} = \frac{n_{e+}}{-\dot{n_{e+}}}$$

otherwise:

$$t_{e+} = \infty$$

And for temperature if its‘ change speed is negative (temperature change here is calculated based on dt with Temperature_change_speed_count(<...>) method.):

$$t_T = \frac{T}{-\dot{T}}$$

otherwise if it is poritive:

$$t_T = \infty$$

Then minimum of burning times of electrons, positrons, elements and core temperature change is multiplied by less then zero ratio called Courant‘s number ("Speed" in code) and the result is the time step of the modeling (dt in code). It also uses modified explicit Euler‘s method but multiplies step of time axis by Courant‘s number between 0 and 1 which helps it to stay stable:

$$t_{step} = min(t) \cdot C$$

.

Output variables: Burning_speed [array] - stores burning times of elements in format [Elements (navigation part)][Concentrations] with navigation [1][Number of the element], Speed [float] - Courant's number, Electrons_burning_speed [float] - burning speed of electrons, Positrons_burning_speed [float] - burning speed of positrons, dT [float] - change of core temperature in Kelvins, dt [float] - time step of the modeling in years, Elements_burning_time [array] - stores burning times of elements in format [Elements (navigation part)][Elements burning time] with navigation [1][Number of the element].

7 Displays:

- **Simulation_state_display(Mass_fractions)**:

Input variables: Mass_fractions [array] - stores mass fractions of elements in format [elements (navigation part)][mass fractions] with navigation [0][Number of element] for element names and [1][Number of element] for mass fraction values.

Description: This function formats the mass fractions of elements into a string for display. Each element's name and its corresponding mass fraction are included in the output string.

Output variables: Answer [string] - a formatted string containing element names and their mass fractions.

- **Concentrations_state_display(Concentrations)**:

Input variables: Concentrations [array] - stores concentrations of elements in format [elements (navigation part)][concentrations] with navigation [0][Number of element] for element names and [1][Number of element] for concentration values.

Description: This function formats the concentrations of elements into a string for display. Each element's name and its corresponding concentration are included in the output string.

Output variables: Answer [string] - a formatted string containing element names and their concentrations.

- **Elements_burning_speed_state_display(Elements_burning_speed)**:

Input variables: Elements_burning_speed [array] - stores burning speeds of elements in format [elements (navigation part)][burning speeds] with navigation [0][Number of element] for element names and [1][Number of element] for burning speed values.

Description: This function formats the burning speeds of elements into a string for display. Each element's name and its corresponding burning speed are included in the output string.

Output variables: Answer [string] - a formatted string containing element names and their burning speeds.

- **Elements_burning_time_state_display(Elements_burning_time)**:

Input variables: Elements_burning_time [array] - stores burning times of elements in format [elements (navigation part)][burning times] with navigation [0][Number of element] for element names and [1][Number of element] for burning time values.

Description: This function formats the burning times of elements into a string for display. Each element's name and its corresponding burning time are included in the output string.

Output variables: Answer [string] - a formatted string containing element names and their burning times.

- **Simulation_state_display_simpler(Mass_fractions)**:

Input variables: Mass_fractions [array] - stores mass fractions of elements in format [elements (navigation part)][mass fractions] with navigation [0][Number of element] for element names and [1][Number of element] for mass fraction values.

Description: This function provides a simplified format for displaying mass fractions of elements. It creates a comma-separated string of mass fraction values without element names.

Output variables: Answer [string] - a comma-separated string of mass fraction values.

- **El_pos_state_display(Electrons_concentration, Positrons_concentration)**:

Input variables: Electrons_concentration [float] - concentration of electrons, Positrons_concentration [float] - concentration of positrons.

Description: This function formats the concentrations of electrons and positrons into a comma-separated string.

Output variables: [string] - a formatted string containing electron and positron concentrations.

- **Time_display(Time, Time_limit)**:

Input variables: Time [float] - current simulation time in years, Time_limit [float] - total simulation time limit in years.

Description: This function formats the current simulation time and calculates the percentage of the simulation completed. It returns a string with the time in years and the percentage completed.

Output variables: [string] - a formatted string containing the current time in years and the percentage of simulation completed.

- **Temperature_display(Temperature)**:

Input variables: Temperature [float] - current temperature in Kelvin.

Description: This function formats the temperature for display.

Output variables: [string] - a formatted string containing the temperature in Kelvin.

- **General_condition_display(Concentrations, Elements_burning_time, Electrons_burning_time, Positrons_burning_time)**:

Input variables: Concentrations [array] - stores concentrations of elements in format [elements (navigation part)][concentrations] with navigation [0][Number of element] for element names and [1][Number of element] for concentration values, Elements_burning_time [array] - stores mass fractions of elements in format [elements (navigation part)][burning_times] with navigation [0][Number of element] for element names and [1][Number of element] for burning time values, Electrons_burning_time [float] - burning time for electrons, Positrons_burning_time [float] - burning time for positrons.

Description: This function uses empericaly estimated boundaries of parameters of simulation to deretmine if it is stable or not. Output of this function is only emperical estimate to warn user about potential problems and shall not be taken as the ultimate verdict..

Output variables: [string] - A status message ("Stable", "Suspicious...", or "Critical!") based on the current state of the simulation..

- **Debug_display(Unresonant_reactions, Concentrations, Electrons_concentration, Positrons_concentration)**:

Input variables: Unresonant_reactions [not used], Concentrations [array] - stores concentrations of elements, Electrons_concentration [float] - concentration of electrons, Positrons_concentration [float] - concentration of positrons.

Description: This function is currently unused. It's designed to print concentrations of elements, electrons, and positrons for debugging purposes.

Output variables: None (prints to console).

8 Electron_positron_operations:

- **Electron_positron_unresonant_matrix_create(R)**:

Input variables: R [int] - number of unresonant reactions.

Description: This function parses unresonant reaction equations from the "Simulation_input.txt" file and creates matrices storing the effect on electron and positron numbers for each reaction.

Output variables: Electron_unresonant_reaction_effect [array] - stores reaction's impacts on concentrations of electrons (how many one reaction produces or consumes) with navigation [Number of unresonant reaction], Positron_unresonant_reaction_effect [array] - stores reaction's impacts on concentrations of positrons (how many one reaction produces or consumes) with navigation [Number of unresonant reaction].

- **Electron_positron_resonant_matrix_create(R)**:

Input variables: R [int] - number of resonant reactions.

Description: This function parses resonant reaction equations from the "Simulation_input.txt" file and creates matrices storing the effect on electron and positron numbers for each reaction.

Output variables: Electron_resonant_reaction_effect [array] - stores reaction's impacts on concentrations of electrons (how many one reaction produces or consumes) with navigation [Number of resonant reaction], Positron_resonant_reaction_e

[array] - stores reaction‘s impacts on concentrations of positrons (how many one reaction produces or consumes) with navigation [Number of resonant reaction.

- **Electron_positron_decay_matrix_create(R)**:

Input variables: R [int] - number of decay reactions.

Description: This function parses decay reaction equations from the "Simulation_input.txt" file and creates matrices storing the effect on electron and positron numbers for each reaction.

Output variables: Electron_decay_reaction_effect [array] - stores reaction‘s impacts on concentrations of electrons (how many one reaction produces or consumes) with navigation [Number of unresonant reaction], Positron_unresonant_reacti [array] - stores reaction‘s impacts on concentrations of positrons (how many one reaction produces or consumes) with navigation [Number of unresonant reaction].

- **Electron_positron_processes_matrix_create(R)**:

Input variables: R [int] - number of processes.

Description: This function parses process equations from the "Simulation_input.txt" file and creates matrices storing the effect on electron and positron numbers for each process.

Output variables: Electron_process_effect [array] - stores process‘ impacts on concentrations of electrons (how many one process produces or consumes) with navigation [Number of process], Positron_process_effect [array] - stores process‘ impacts on concentrations of positrons (how many one process produces or consumes) with navigation [Number of process].

- **Electron_positron_initial_concentration_count(Elements, Concentrations)**:

Input variables: Elements [array] - copy of "Elements_data.txt", Concentrations [array] - stores concentrations of elements.

Description: This function calculates the initial concentration of electrons and positrons based on the assumption of overall electrical neutrality of the star:

If charge of all nucleis from 0 to k is positive (any real situation):

$$n_{e-} = \sum_{i=1}^{k} n_i$$

If it is negative otherwise, then method will create correspondint initial concentration of positrons:

$$n_{e+} = \sum_{i=1}^{k} n_i$$

.

Output variables: Electrons_concentration [float] - initial concentration of electrons, Positrons_concentration [float] - initial concentration of positrons.

- **Electron_positron_burning_speed(Electron_unresonant_reaction_effect, Positron_unresonant_reaction_effect, Unresonant_reactions, Electron_resonant_reaction_effect, Positron_resonant_reaction_effect, Resonant_reactions, Electron_decay_reaction_effect, Positron_decay_reaction_effect, Decay_reactions, Electron_process_effect, Positron_process_effect, Processes):**

  :

Input variables: Electron_unresonant_reaction_effect [array] - stores reaction's impacts on concentrations of electrons (how many reaction produces or consumes) in format [Reactions impact] with navigation [Number of read unresonant reaction], Positron_unresonant_reaction_effect [array] - stores reaction's impacts on concentrations of positrons (how many reaction produces or consumes) in format [Reactions impact] with navigation [Number of read unresonant reaction], Unresonant_reactions [array] - unresonant reactions matrix, Electron_resonant_reaction_effect [array] - stores reaction's impacts on concentrations of electrons (how many reaction produces or consumes) in format [Reactions impact] with navigation [Number of read resonant reaction], Positron_resonant_reaction_effect [array] - stores reaction's impacts on concentrations of positrons (how many reaction produces or consumes) in format [Reactions impact] with navigation [Number of read resonant reaction], Resonant_reactions [array] - resonant reactions matrix, Electron_decay_reaction_effect [array] - stores reaction's impacts

on concentrations of electrons (how many reaction produces or consumes) in format [Reactions impact] with navigation [Number of read decay reaction], Positron_decay_reaction_effect [array] - stores reaction‘s impacts on concentrations of positrons (how many reaction produces or consumes) in format [Reactions impact] with navigation [Number of read decay reaction], Decay_reactions [array] - decay reactions matrix, Electron_process_effect [array] - stores process‘ impacts on concentrations of electrons (how many process produces or consumes) in format [Process‘ impact] with navigation [Number of read process], Positron_process_effect [array] - stores process‘ impacts on concentrations of positrons (how many process produces or consumes) in format [Process‘ impact] with navigation [Number of read process], Processes [array] - processes matrix.

Description: This function calculates the rate of change in electron and positron concentrations based on all types of reactions and processes:

$$\dot{n_{e-}} = -a_{e-} \sum_{i=1}^{k} R_i$$

,

$$\dot{n_{e+}} = -a_{e+} \sum_{i=1}^{k} R_i$$

,

where for k reactions that include electrons or positrons where "a" is factor coresnonding to nuclei them (a is positive if particle is in right side of nuclear equation and vice versa).

.

Output variables: Electrons_burning_speed [float] - rate of change of concentration of electron, Positrons_burning_speed [float] - rate of change of concentration of positron.

- **Read_the_reaction_el_pos(Reaction)**:

Input variables: Reaction [string] - a string representing a reaction equation.

Description: This function parses a reaction equation to determine its effect on electron and positron numbers.

Output variables: Electrons_add [int] - the net change in electron for given reaction, Positrons_add [int] - the net change in positron for given reaction.

- **El_pos_burning_time_count(Electrons_burning_speed, Positrons_burning_speed, Electrons_concentration, Positrons_concentration)**:

Input variables: Electrons_burning_speed [float], Positrons_burning_speed [float] - Rates of change in electron and positron concentrations, Electrons_concentration [float], Positrons_concentration [float] - Current concentrations of electrons and positrons.

Description: This function calculates the burning time for electrons and positrons based on their current concentrations and burning speeds.

If burning speed of electrons is negative:

$$t_{e-} = \frac{n_{e-}}{-\dot{n_{e-}}}$$

otherwise:

$$t_{e-} = \infty$$

Same for positrons:

$$t_{e+} = \frac{n_{e+}}{-\dot{n_{e+}}}$$

otherwise:

$$t_{e+} = \infty$$

.

Output variables: Electrons_burning_time [float] - burning time for electrons, Positrons_burning_time [float] - burning time for positrons.

9 Element_operations:

- **Elements_list_create()**:

Input variables: None (reads from "Elements_data.txt").

Description: Creates an array containing data about elements in the format: [Element, nucleus mass, nucleus charge, nucleus spin].

Output variables: Elements [array] - Elements [array] - copy of "Elements_data.txt".

- **Is_element_new(x, Elements)**:

Input variables: x [string] - element symbol, Elements [array] - here: elements, already read from "Elements_data.txt".

Description: Low-level technical function used when creating the element matrix. Checks if an element is new (not already in the Elements array).

Output variables: [bool] - True if the element is new, False otherwise.

- **Mass_of_element(x, Elements)**:

Input variables: x [string] - element symbol, Elements [array] - copy of "Elements_data.txt".

Description: Technical function to find the nucleus mass of a given element in the Elements array.

Output variables: [float] - mass of the element's nucleus.

- **Charge_of_element(x, Elements)**:

Input variables: x [string] - element symbol, Elements [array] - copy of "Elements_data.txt".

Description: Technical function to find the nucleus charge of a given element in the Elements array.

Output variables: [int] - charge of the element's nucleus.

- **Spin_of_element(x, Elements)**:

Input variables: x [string] - element symbol, Elements [array] - copy of "Elements_data.txt".

Description: Technical function to find the nucleus spin of a given element in the Elements array.

Output variables: [float] - spin of the element's nucleus.

- **Concentration_of_element(x, Concentrations)**:

Input variables: x [string] - element symbol, Concentrations [array] - stores concentrations of elements in format [elements (navigation part)][concentrations] with navigation [1][Number of the element].

Description: Technical function to find the concentration of nuclei of a given element in the Concentrations array.

Output variables: [float] - concentration of the element's nuclei.

- **Mass_fraction_of_element(x, Mass_fractions)**:

Input variables: x [string] - element symbol, Mass_fractions - [array] stores mass fractions of elements in format [elements (navigation part)][mass fractions] with navigation [1][Number of elements].

Description: Technical function to find the mass fraction of nuclei of a given element from the core substance in the Mass_fractions array.

Output variables: [float] - Mass fraction of the element's nuclei.

- **Max_concentration_of_element(x, Elements, ro, M_nuc)**:

Input variables: x [string] - element symbol, Elements [array] - copy of "Elements_data.txt", ro [float] - core density, M_nuc [float] - atomic mass unit in grams.

Description: Technical function to calculate the maximum possible concentration of elements. Used to determine the burning time of an element if its concentration is increasing.

Output variables: Max_concentration [float] - maximum possible concentration of the element.

- **Compound_core(x, y, Elements)**:

Input variables: x, y [string] - element symbols, Elements [array] - copy of "Elements_data.txt".

Description: Technical function for resonant reactions, determining the conditional core that arose before the instantaneous alpha decay in some resonant reactions with He4 output:

then

$$^{A_1}_{Z_1}X + ^{A_2}_{Z_2}Y \rightarrow ^{A_3}_{Z_3}Z$$

where:

$A_1 + A_2 = A_3$

and

$Z_1 + Z_2 = Z_3$.

Altough, a compound core should be listed in "Elements.txt" for this function not to crash

.

Output variables: [string] - symbol of the compound core that would be created if protons and neutrons of cores x and y were fused. .

- **Reduced_mass(m1, m2)**:

Input variables: m1, m2 [float] - masses of two particles.
Description: Technical function for resonant reactions, calculating the reduced mass of the reaction (not a process):

$$m_{ik} = \frac{m_i m_k}{m_i + m_k}$$

.
Output variables: [float] - reduced mass of two particles.

- **Average_per_particle_weight_count(El_con, Pos_con, Elements, Concentrations)**:

Input variables: El_con [float] - electron concentration, Pos_con [float] - positron concentration, Elements [array] - copy of "Elements_data.txt", Concentrations [array] - stores concentrations of elements in format [elements (navigation part)][concentrations] with navigation [1][Number of the element].
Description: Function to calculate the average molecular weight (average mass of any particle):

$$\langle m \rangle = \frac{\sum_{i=1}^{k} n_i \cdot m_i}{\sum_{i=1}^{k} n_i}$$

.
Output variables: Average_per_particle_weight [float] - Average molecular weight.

- **Read_element(x, Elements)**:

Input variables: x [string] - element symbol, Elements [array] - copy of "Elements_data.txt".
Description: Parsing function that combines the results of functions for finding the mass, charge, and spin of an element's nucleus for a given element.
Output variables: x [string], Mass [float], Charge [int], Spin [float] - element symbol, mass, charge, and spin.

- **Read_the_unresonant_reaction(line, Reaction_number, Elements, Unresonant_reactions)**:

Input variables: line [string] - reaction‘s equation from "Simulation_input.txt", Reaction_number [int] - index of the reaction, Elements [array] - copy of "Elements_data.txt", Unresonant_reactions [array] - matrix of unresonant reactions.

Description: Parsing function designed to record part of the data about the reaction in the matrix of unresonant reactions.

Output variables: Unresonant_reactions [array] - updated matrix of unresonant reactions.

- **Read_the_resonant_reaction(line, Reaction_number, Elements, Resonant_reactions)**:

Input variables: line [string] - reaction‘s equation from "Simulation_input.txt", Reaction_number [int] - index of the reaction, Elements [array] - copy of "Elements_data.txt", Resonant_reactions [array] - matrix of resonant reactions.

Description: Parsing function designed to record part of the data about the reaction in the matrix of resonant reactions.

Output variables: Resonant_reactions [array] - updated matrix of resonant reactions.

- **Read_the_decay_reaction(line, Reaction_number, Elements, Decay_reactions)**:

Input variables: line [string] - reaction‘s‘ equation from "Simulation_input.txt", Reaction_number [int] - index of the reaction, Elements [array] - copy of "Elements_data.txt", Decay_reactions [array] - matrix of decay reactions.

Description: Parsing function designed to record part of the data about the reaction in the matrix of decay reactions.

Output variables: Decay_reactions [array] - updated matrix of decay reactions.

- **Read_the_process(line, Reaction_number, Elements, Processes)**:

Input variables: line [string] - process‘ equation from "Simulation_input.txt", Reaction_number [int] - index of the process, Elements [array] - copy of "Elements_data.txt", Processes [array] - matrix of processes.

Description: Parsing function designed to record part of the data about the process in the matrix of processes.

Output variables: Processes [array] - updated matrix of processes.

10 Neutrino_spectrum:

- **Is_neutrino(Components)**:

Input variables: Components [array] - list of components of reaction‘s eqation..

Description: Parsing function designed to identify if reaction emmits a neutrino as a result.

Output variables: [bool] - parament that shows if reaction emmits a neutrino (True) or not (False).

- **Neutrino_flux(Distance, R_core, R)**:

Input variables: Distance [float] - distance to the core of a star from which neutrino flux must be predicted, R_core [float] - radium of the core, R [float] - ammount of recation for which neutrino flux is being calculated per cm^-3 per s^-1.

Description: Calculates neutrino flux for a given reaction based on its rate:

$$\Phi_\nu = \frac{V \cdot R}{S}$$

where V - is a total volume of the core, R - rate of the reaction that emmits neutrino and S - total surface of the star.

(NOTE: Uses simpilification assuming that star is a material point, works well only for observation point with distance to the star way greater then the radium of the star)

.

Output variables: [int] -ammount of neutrino per second with set energy and intensity of their emission.

- **Neutrino_energy(Energy, E_max)**:

Input variables: Energy [float] - we calculate flux of neutrinos that have this energy in MeV, E_max [float] - max possible neutrino energy for given reaction (only for reactions with non-constant neutrino energy).

Description: Calculates probability density for neutrino with set energy using Ferma‘s formula for beta-decay neutrino energy density probability [2]:

$$P(E) = X^2 \cdot (1 - X)^2 \cdot \sqrt{1 - X^2}$$

with

$$X = \frac{E}{E_{max}}$$

where E_{max} is the energy output of the reaction neutrino was produced by.

.

Output variables: [float] - probability density for neutrino with set energy.

- **Neutrino_unresonant_reactions_matrix_create()**:

Input variables: None (reads from "Elements_data.txt").

Description: Uses "Simulation_input" to create an array with information about neutrino emission for "Unresonant_reactions":

Neutrino_unresonant_reactions_matrix for reaction with number i:

- Neutrino_unresonant_reactions_matrix[0][i] - reaction type (0 - neutrino-free reactions, 1 - continuum (neutrinos with different energies), 2 - neutrinos with fixed energy);
- Neutrino_unresonant_reactions_matrix[1][i] - maximum neutrino energy (for type 1 reactions);
- Neutrino_unresonant_reactions_matrix[2][i] - neutrino channel energy for first channel (for type 2 reactions);
- Neutrino_unresonant_reactions_matrix[3][i] - probability of neutrino channel for first channel (for type 2 reactions);
- Neutrino_unresonant_reactions_matrix[4][i] - neutrino channel energy for second channel (for type 2 reactions);
- Neutrino_unresonant_reactions_matrix[5][i] - probability of neutrino channel for second channel (for type 2 reactions).

.

Output variables: Neutrino_unresonant_reactions_matrix [array] - unresonant reactions neutrino matrix.

- **Neutrino_resonant_reactions_matrix_create()**:

Input variables: None (reads from "Elements_data.txt").

Description: Uses "Simulation_input" to create an array with information about neutrino emission for "Resonant_reactions":

Neutrino_resonant_reactions_matrix for reaction with number i:

- Neutrino_resonant_reactions_matrix[0][i] - reaction type (0 - neutrino-free reactions, 1 - continuum (neutrinos with different energies), 2 - neutrinos with fixed energy);
- Neutrino_resonant_reactions_matrix[1][i] - maximum neutrino energy (for type 1 reactions);
- Neutrino_resonant_reactions_matrix[2][i] - neutrino channel energy for first channel (for type 2 reactions);
- Neutrino_resonant_reactions_matrix[3][i] - probability of neutrino channel for first channel (for type 2 reactions);
- Neutrino_resonant_reactions_matrix[4][i] - neutrino channel energy for second channel (for type 2 reactions);
- Neutrino_resonant_reactions_matrix[5][i] - probability of neutrino channel for second channel (for type 2 reactions).

.

Output variables: Neutrino_resonant_reactions_matrix [array] - resonant reactions neutrino matrix.

- **Neutrino_decay_reactions_matrix_create()**:

Input variables: None (reads from "Elements_data.txt").

Description: Uses "Simulation_input" to create an array with information about neutrino emission for "Decay_reactions":

Neutrino_decay_reactions_matrix for reaction with number i:

- Neutrino_decay_reactions_matrix[0][i] - reaction type (0 - neutrino-free reactions, 1 - continuum (neutrinos with different energies), 2 - neutrinos with fixed energy);
- Neutrino_decay_reactions_matrix[1][i] - maximum neutrino energy (for type 1 reactions);

- Neutrino_decay_reactions_matrix[2][i] - neutrino channel energy for first channel (for type 2 reactions);
- Neutrino_decay_reactions_matrix[3][i] - probability of neutrino channel for first channel (for type 2 reactions);
- Neutrino_decay_reactions_matrix[4][i] - neutrino channel energy for second channel (for type 2 reactions);
- Neutrino_decay_reactions_matrix[5][i] - probability of neutrino channel for second channel (for type 2 reactions).

.

Output variables: Neutrino_decay_reactions_matrix [array] - decay reactions neutrino matrix.

- **Neutrino_processes_matrix_create()**:

Input variables: None (reads from "Elements_data.txt").
Description: Uses "Simulation_input" to create an array with information about neutrino emission for "Decay_reactions":
Neutrino_processes_matrix for reaction with number i:

- Neutrino_processes_matrix[0][i] - process type (0 - neutrino-free processes, 1 - continuum (neutrinos with different energies), 2 - neutrinos with fixed energy);
- Neutrino_processes_matrix[1][i] - maximum neutrino energy (for type 1 processes);
- Neutrino_processes_matrix[2][i] - neutrino channel energy for first channel (for type 2 processes);
- Neutrino_processes_matrix[3][i] - probability of neutrino channel for first channel (for type 2 processes);
- Neutrino_processes_matrix[4][i] - neutrino channel energy for second channel (for type 2 processes);
- Neutrino_processes_matrix[5][i] - probability of neutrino channel for second channel (for type 2 processes).

.

Output variables: Neutrino_processes_matrix [array] - decay reactions neutrino matrix.

- **Neutrino_spectrum_create(Neutrino_unresonant_reactions_matrix, Unresonant_reactions, Neutrino_resonant_reactions_matrix, Resonant_reactions, Neutrino_decay_reactions_matrix, Decay_reactions, Neutrino_processes_matrix, Processes, R_core, Distance_neutrino_spectrum)**:

Input variables: Neutrino_unresonant_reactions_matrix [array] - unresonant reactions neutrino matrix, Unresonant_reactions [array] - unresonant reactions matrix, Neutrino_resonant_reactions_matrix [array] - resonant reactions neutrino matrix, Resonant_reactions [array] - resonant reactions matrix, Neutrino_decay_reactions_matrix [array] - decay reactions neutrino matrix, Decay_reactions [array] - decay reactions matrix, Neutrino_processes_matrix [array] - processes neutrino matrix, Processes [array] - processes matrix, R_core [float] - core radium in santimeters, Distance_neutrino_spectrum [float] - distance to the core of a star from which neutrino flux must be predicted.

Description: For all types of reactions: unresonant reactions, resonant reactions, decay reactions and processes method checks type of neutrino emission in coresponding neutrino reactions and acts corespondingly. If reaction is neutrinoless, then it does nothing. If reaction‘s neutrino emission is continuum, then it takes a set ammount of energy values in energy range from 0 to maximum possible neutrino energy, uses Neutrino_energy(Energy, E_max) method to get probability density for each energy values, then applies Neutrino_flux(Distance, R_core, R) method to each of them to get neutrino flux for each energy values based on rate of coresponding reaction and adds (neutrino flux / neutrino energy) array to the graph. If reaction emmits neutinos with set energy or energies, then it does all same actions but uses probabilty of neutrino getting a specific energy instead of Neutrino_energy(Energy, E_max) method.

Output variables: None (draws graph using mathplotlib.pyplot showing neutrino spectrum).

11 Processes:

- **Processes_matrix_create(Elements)**:

Input variables: Elements [array] - copy of "Elements_data.txt".

Description: Uses "Simulation_input" and "Elements_data.txt" to create a partially filled template for "Processes":

Processes for reaction with number i:

- Processes[0][i] - first initial element;
- Processes[1][i] - second initial element;
- Processes[2][i] - third initial element;
- Processes[3][i] - first resulting element;
- Processes[4][i] - second resulting element;
- Processes[5][i] - third resulting element;
- Processes[6][i] - energy effect of the process;
- Processes[7][i] - value of $<\sigma \cdot v>$;
- Processes[8][i] - number of processes per cm^{-3} per s^{-1}.

.

Output variables: Processes [array] - processes matrix.

- **Processes_formulas_container(Processes, T, N_a, t1)**:

Input variables: Processes [array] - processes matrix, T [float] - core temperature in Kelvin, N_a - Avogadro‘s number, t1 - number of process in Processes.

Description: Aplies a complex numerical formula for process‘ cross-section from a container consisting of units with structure: "[Element 1, Element 2, Element 3, Critical temperature (for changing formula from cold to hot), cold formula, hot formula]". Fromulas come from an article [1].

Output variables: Decay_recationss [array] - decay reactions matrix.

- **Processes_matrix_update(Processes, T, N_a)**:

Input variables: Processes [array] - processes matrix, T [float] - core temperature in Kelvin, N_a - Avogadro‘s number.

Description: Applies Processes_formulas_container(Processes, T, N_a, t1) method for each process in Processes.

Output variables: Processes [array] - processes matrix.

- **Processes_count(Processes, Concentrations)**:

Input variables: Processes [array] - processes matrix, Concentrations - [array] stores concentrations of elements in format [elements (navigation part)][concentrations] with navigation [1][Number of the element].

Description: Calculates rates of processes based on Processes using formula:

$$R = n_1 \cdot n_2 \cdot n_3 \cdot \langle \sigma \cdot V \rangle_{gs}$$

.

Output variables: Processes [array] - processes matrix.

12 Resonant_reactions:

- **Integration_trapz(x, y)**:

Input variables: x, y [array] (of equal size) - sets of x and y coordinates of data points of the graph that is being integrated.

Description: This utility function performs numerical integration using the trapezoidal rule. It calculates the area under the curve defined by the x and y values, function iterates through the provided lists, computing the integral by summing up the areas of trapezoids formed by adjacent data points. If there are k data points and step between data points is h then:

$$\int_{x_1}^{x_2} f(x)dx \approx h \cdot \left(\frac{f(x_1) + f(x_2)}{2} + \sum_{i=1}^{k-1} f(x_1 + ih) \right)$$

.

Output variables: Integral - total square under the data curve.

- **Resonant_reactions_matrix_create(Elements, h_, M_nuc, e, k, T)**:

Input variables: Elements [array] - copy of "Elements_data.txt", h_ [float] - reduced Planck's constant in SI, M_nuc [float] - atomic unit of mass in grams, e [float] - Euler's number, k [float] - Boltzman's constant in SI, T [float] - core temperature in Kelvin.

Description: Uses "Simulation_input" and "Elements_data.txt" to create a partially filled template for "Resonant_reactions":

Resonant_reactions for reaction with number i:

- Resonant_reactions[0][i] - first initial element;
- Resonant_reactions[1][i] - second initial element;
- Resonant_reactions[2][i] - mass of first initial element;
- Resonant_reactions[3][i] - mass of second initial element;
- Resonant_reactions[4][i] - charge of first initial element;
- Resonant_reactions[5][i] - charge of second initial element;
- Resonant_reactions[6][i] - spin of first initial element;
- Resonant_reactions[7][i] - spin of second initial element;
- Resonant_reactions[8][i] - reduced reaction mass;
- Resonant_reactions[9][i] - first resulting element;
- Resonant_reactions[10][i] - mass of first resulting element;
- Resonant_reactions[11][i] - charge of first resulting element;
- Resonant_reactions[12][i] - spin of first resulting element;
- Resonant_reactions[13][i] - second resulting element;
- Resonant_reactions[14][i] - mass of second resulting element;
- Resonant_reactions[15][i] - charge of second resulting element;
- Resonant_reactions[16][i] - spin of second resulting element;
- Resonant_reactions[17][i] - partial width of the input channel;
- Resonant_reactions[18][i] - partial width of the output channel;
- Resonant_reactions[19][i] - full resonance width;
- Resonant_reactions[20][i] - resonance energy;
- Resonant_reactions[21][i] - energetic effect of the reaction;
- Resonant_reactions[22][i] - temperature for which the cross-section of the formation of a composite core is calculated;

- Resonant_reactions[23][i] - dependence of the cross-section on the energy of the nuclei (based on Breit-Wigner's formula over an energy range of 50 resonance full widths from resonance energy in both directions):
  - Resonant_reactions[23][i][0] - values of energy in SAJouls that lay within energy range of 50 resonance full widths from resonance energy in both directions (otherwise are 0);
  - Resonant_reactions[23][i][1] - values of cross-section that are corresponding to energy values from Resonant_reactions[23][i][0] (Resonant_reactions[23][i][0][j] and Resonant_reactions[23][i][1][j] is a corresponding pair);
- Resonant_reactions[24][i] - $<\sigma \cdot v>$ of the formation of a composite core;
- Resonant_reactions[25][i] - number of reactions per cm^{-3} per s^{-1}.

.

Output variables: Decay_recations [array] - decay reactions matrix.

- **Resonant_reactions_matrix_update(Resonant_reactions, Concentrations, T, Pi, e, k, M_nuc)**:

Input variables: Resonant_reactions [array] - resonant reactions matrix, T [float] - core temperature in Kelvin, Pi [float] - number π, e [float] - Euler's number, k [float] - Boltzman's constant, M_nuc [float] - atomic unit of mass in grams.

Description: For each resonant reaction in Resonant_reactions in calculates:

$$\langle \sigma \cdot v \rangle_{gs} = \int_0^{\alpha \cdot c} \left(\int_{E_{res}-n\cdot\Gamma}^{E_{res}+n\cdot\Gamma} \sigma(E) \cdot P(E; v_{rel})\, dE \right) v_{rel} \cdot P(v_{rel})\, dv_{rel}$$

Here α and n are technical parameters. α is needed to avoid effects of general relativity and should be kept below 0.1. n determines the width of energy range that is used to calculate cross-sections of resonant reactions (range of n resonance full widths from resonance energy in both directions).

Also, both integrals are normalized using Integration_trapz(x, y) method. But that is only a technical correction with change due to that usually being less then one millioth and quickly decreasing with the increase of Point_of_calculation parameter.

$\sigma(E)$ comes from Breit-Wigner's formula:

$$\sigma(E) = \frac{\pi\hbar^2}{2\mu E} \cdot \frac{(2J+1)}{(2J_i+1)(2J_j+1)} \cdot \frac{\Gamma_a\Gamma_b}{(E-E_R)^2 + \left(\frac{\Gamma}{2}\right)^2}$$

$P(E; v_{rel})$ comes from Maxwell's distribution of probability for a pair of particles to have a certain value of total energy in non-inertial frame of reference based on their relative speed:

$$P(E; v_{rel}) = \begin{cases} \sqrt{E - E_{rel}} \cdot \exp\left(-\frac{E-E_{rel}}{kT}\right), & if\ E \geq E_{rel}, \\ 0, & if\ E < E_{rel}. \end{cases}$$

And $P(v_{rel})$ is Maxwell's probability distribution for relative speed of a pair of particles (not necessarily similar):

$$P(v_{rel}) = 4\pi \left(\frac{m_{ik}}{2\pi kT}\right)^{3/2} v_{rel}^2 \exp\left(-\frac{m_{ik} v_{rel}^2}{2kT}\right)$$

where m_{ik} is reduced mass of a pair of particles:

$$m_{ik} = \frac{m_i m_k}{m_i + m_k}$$

.

Output variables: Resonant_reactions [array] - resonant reactions matrix.

- **Resonant_reactions_reactions_count(Resonant_reactions, Pi, R, T, Elements, Concentrations)**:

Input variables: Resonant_reactions [array] - resonant reactions matrix, Pi [float] - number π, R [float] - universal gas constant in SI, T [float] - core temperature in Kelvin, Concentrations - [array] stores concentrations of elements in format [elements (navigation part)][concentrations] with navigation [1][Number of the element].

Description: Calculates rates of resonant reactions based on Processes using formula:

$$R = n_1 \cdot n_2 \cdot \langle \sigma \cdot v \rangle_{gs}$$

.

Output variables: Processes [array] - processes matrix.

- **Resonant_reactions_matrix_print(Resonant_reactions)**:

Input variables: Resonant_reactions [array] - resonant reactions matrix.

Description: Prints resonant reactions matrix.

Output variables: None (prints into console).

13 Temperature_change:

- **Temperature_change_speed_count(Unresonant_reactions, Resonant_reactions, Decay_reactions, Processes, T_surface, R_star, R_core, Concentrations)**:

Input variables: dt [float] - time step of the modeling, Unresonant_reactions [array] - unresonant reactions matrix, Resonant_reactions [array] - resonant reactions matrix, Decay_reactions [array] - decay reactions matrix, Processes [array] - processes matrix, St_Bol [float] - Stefan-Boltzman's constant in SI, Pi [float] - number π, T_surface - surface star temperature in Kelvins, R_star, i [int] - degrees of freedom, R_core [float] - universal gas constant in SI, N_a [float] - Avogadro's number in SI, Concentrations.

Description: This method calculates total generated energy from all reactions:

$$E_{gain} = \sum_{i=1}^{k} (E_{effect} \cdot R_i) \times \frac{4}{3} \pi {R_{core}}^3$$

,

then calculates total energy loss using Stefan-Boltzman's constant and star's surface:

$$E_{loss} = \sigma \cdot 4 \cdot \pi \cdot R_{star}^2 \cdot T^4$$

Total energy change:

$$E_{change} = E_{gain} - E_{loss}$$

After that it uses Mayer's ratio with constant volume of unit of substance of the core to get heat capacity for perfect gas model which is used for plasma:

$C_V = \frac{R}{\gamma - 1}$, where $\gamma = \frac{5}{3}$ instead of $\frac{4}{3}$ because plasma is not a perfect one-atom gas.

Ammount of particles in moles in a unit of volume for all isotopes:

$$\nu = \sum_{i=1}^{k} \frac{n_k}{N_a}$$

After that it calculates temperature change using star's total volume:

$$T_{change} = \frac{E_{change}}{C_V \cdot \nu \cdot \frac{4}{3} \pi {R_{core}}^3}$$

.

Output variables: Temperature_change [float] - change of core temperature in Kelvin.

- **Temperature_change_time_count(T, Temperature_change_speed)**:

Input variables: T [float] - core temperature in Kelvins, Temperature_change_speed - $\frac{dT}{dt} = \dot{T}$.

Description:

Calculates time it would take for core temperature to drop to 0 Kelvin with current change rate if it‘s charge speed is negative:

$$t_T = \frac{T}{-\dot{T}}$$

.

Output variables: Temperature_change_time - time it would take for core temperature to drop to 0 Kelvin with current change rate if it‘s charge speed is negative..

14 Unresonant_reactions:

- **Unresonant_reactions_matrix_create(Elements)**:

Input variables: Elements [array] - copy of "Elements_data.txt".

Description: Uses "Simulation _input" and "Elements_data.txt" to create a partially filled template for "Unresonant_reactions":

Unresonant_reactions for reaction with number i:

- Unresonant_reactions[0][i] - first initial element;
- Unresonant_reactions[1][i] - second initial element;
- Unresonant_reactions[2][i] - mass of first initial element;
- Unresonant_reactions[3][i] - mass of second initial element;
- Unresonant_reactions[4][i] - charge of first initial element;
- Unresonant_reactions[5][i] - charge of second initial element;
- Unresonant_reactions[6][i] - reduced reaction mass;
- Unresonant_reactions[7][i] - S_0 (scaling factor for cross-section);
- Unresonant_reactions[8][i] - Kronecker symbol (indicates identity or distinct nature of reaction channels);

- Unresonant_reactions[9][i] - τ;
- Unresonant_reactions[10][i] - first resulting element;
- Unresonant_reactions[11][i] - mass of first resulting element;
- Unresonant_reactions[12][i] - second resulting element;
- Unresonant_reactions[13][i] - mass of second resulting element;
- Unresonant_reactions[14][i] - third resulting element;
- Unresonant_reactions[15][i] - mass of third resulting element;
- Unresonant_reactions[16][i] - number of reactions per unit of time;
- Unresonant_reactions[17][i] - dependence on electrons (positrons);
- Unresonant_reactions[18][i] - first electronic (positronic) proportionality coefficient;
- Unresonant_reactions[19][i] - second electronic (positronic) proportionality coefficient;
- Unresonant_reactions[20][i] - number of the determining reaction (if any);
- Unresonant_reactions[21][i] - energy effect of the reaction.

.

Output variables: Unresonant_recationss [array] - unresonant reactions matrix.

- **Unresonant_reactions_matrix_update(Unresonant_reactions, T)**:

Input variables: Unresonant_recationss [array] - decay reactions matrix; T [float] - currect core temperature in Kelvin.

Description: This method updates tau to a new core temperature value.

Output variables: Unresonant_recationss [array] - unresonant reactions matrix.

- **Unresonant_reactions_count(Unresonant_reactions, Concentrations)**:

Input variables: Unresonant_reactions [array] - unresonant reactions matrix; Concentrations - [array] stores concentrations of elements in format [elements (navigation part)][concentrations] with navigation [1][Number of the element].

Description: Uses classical unresonant reactions rate formula [3] to calculate rates for them:

$$R_{ik} = 7.20 \times 10^{-19} \cdot \frac{n_i n_k}{1 + \delta_{ik}} \cdot \frac{S_0}{Z_i Z_k m_{ik}} \cdot \tau^2 e^{-\tau}$$

,

where:

$$\delta_{ik} = 1$$

if i = k, otherwise

$$\delta_{ik} = 0$$

and

$$\tau = 42.49 \left(\frac{Z_i^2 Z_k^2 m_{ik}}{T/10^6} \right)^{1/3}$$

with

$$m_{ik} = \frac{m_i m_k}{m_i + m_k}$$

.

Output variables: Unresonant_recationss [array] - unresonant reactions matrix.

15 World_constants:

Constaints a class "Worldconstants" with names and values of all world constants. Is imported in other modules where its needded as "Wc".

16 Star_soft_main:

1. Library imports (lines 1-14): This section imports the necessary libraries and modules, including datetime for time operations, matplotlib for plotting graphs, numpy for numerical computations, and all other modules for modeling operations.

2. Simulation parameter initialization (lines 17-54): This block reads input parameters from the "Simulation_input.txt" file and initializes them. Important parameters include: - R_{core}: stellar core radius - ro: sore density - T: temperature - $T_{surface}$: surface temperature - R_{star}: stellar radius - Mass fractions of elements - Time limits and step - Solution method (Curant)

3. File opening and creation of main data structures (lines 56-136): This section opens necessary input/output files, creates element lists, initializes physical constants (e.g., π, e, nucleon mass M_{nuc}, electron mass M_e) and creates matrices for various types of reactions (non-resonant, resonant, decay).

4. Initialization of electron and positron concentrations (lines 138-150): Initial concentrations of electrons and positrons are calculated based on the condition of stellar electroneutrality.

5. Main simulation loop (lines 170-287): This is the core of the program, where step-by-step modeling of stellar core evolution occurs. At each step: - Element concentrations are updated - Reaction matrices are updated - A differentiation step is performed (Curant or Runge-Kutta 4) - Element mass fractions, electron and positron concentrations, and temperature are updated - Simulation state information is periodically output

6. Results visualization (lines 290-375): After the simulation is complete, data is read from the output file and a graph of various stellar parameters' evolution (element mass fractions, electron and positron concentrations, temperature) is plotted as a function of time.

17 Simulation_input:

NOTE: All <variables> in this section are presented in format: Elements name [string]: Characteristic [float], ...

- General star parameters:

For graphs: core radius, core density, core temperature, star radius, initial elements concentrations, modleing time limit, time step limit (cirrently is hard-written into code in Curant(<...>) method with relative time step limit), differential equations solving method: method, its mode and coresponding constants (currently only Curant and Force), core temperature change permission, output frequency, output format - must be provided suitable values according to description in the file.

- Reactions settings (RECOMMENDED NOT TO CHANGE):

Reactions constants coresponding to there model (except for "Processes" - complicated formulas for them are hard-written into Processes_formulas_container(<...>)

method).

- Reactions:

Reactions equations with some settings:

Unesonant: Components_(Defying reaction number (result is multiplied by that defying reaction rate))_(Energy output)_(Neutrino emission settings)

Resonant, Decay and Processes: Components_(Energy output)_(Neutrino emission settings)

18 World_constants:

Contains universal constants in SA units measurement system in format:

[Name] = [Value] [Measure units]

List of constants:

- Pi = 3.14159262 - number Pi;
- e = 2.7182818284590 - Euler‘s number;
- M_nuc = 1.66e-24 g - atomis mass unit;
- M_e = 9.1093837015e-28 g - mass of a stationary electron;
- i = 6 - degrees of freedom of particles in stellar plasma;
- St_Bol = 5.67e-8 * 1e+22 * 3.154e+7 * 1e-4 $SAWatt/sm^2 * K^4$ - Stephan-Boltzman‘s constant;
- R = 8.3 * 1e+22 SaJoul / K * Mole - universal gas constant;
- N_a = 6.022e+23 1 / Mole - Avogadro‘s number;

- k = 1.380649e-23 * 1e+22 SAJoul / K - Boltzman‘s constant;
- h = 6.63e-34 * 1e+22 / 3.154e+7 SAJoul * y - Plank‘s constant;
- h_ = 1.0545726e-34 * 1e+22 / 3.154e+7 SAJoul * y - reduced Plank‘s constant;
- p0 = 1e+5 * 1e+16 SAPascal - normal pressure;
- T0 = 273.15 K - normal temperature.

19 Elements_data:

Contains information about nucleis in format:

[Name] [Mass in atomic mass units] [Charge in elemental charges] [Spin]

(Note: Shall not be modified, to set up initial concentrations of elements, use ”Simulation_input.txt”).

20 Simulation_output:

Single output structure:

Passed time of modeling, percentage of time passed (from time limit), <Concentrations>, Electrons concentration, Positrons concentration, Temperature K Time step.

21 Logs:

NOTE: All <variables> in this section are presented in format: Elements name [string]: Characteristic [float], ...

Initial output if no major bugs occure:

```
[Output time]>>>Creating unresonant reaction matrix...
[Output time]>>>Succesfully created unresonant reaction matrix with [int] reactions.
[Output time]>>>Creating resonant reactions matrix...
[Output time]>>>Succesfully created resonant reaction matrix with [int] reactions.
[Output time]>>>Creating decay reactions matrix...
[Output time]>>>Succesfully created decay reaction matrix with [int] reactions.
[Output time]>>>Creating processes matrix...
[Output time]>>>Succesfully created processes matrix with [int] reactions.
[Output time]>>>Starting simulation...
```

Single output structure:

```
[Output time]>>>General info: Mass fractions: Passed time, percentage of time (from time limit), <Concentrations>, El. conc., Posit. conc., Temp.
[Output time]>>>Iteration: [int], Time passed: [float], [float]\%, Time step: [float]y
[Output time]>>>Slowest element: [float] (Burning time), [string] (name of the element)
[Output time]>>>Reactions speed: Unres.: [<Reactions rates>] Res.: [<Reactions rates>] Decay: [<Reactions rates>] Processes: [<Reactions rates>]
[Output time]$>>>$Concentrations: <Concentrations>
[Output time]$>>>$Elements\_burning\_speed: <Elements burning speed>
[Output time]$>>>$Elements\_burning\_time: <Elements burning time>
[Output time]$>>>$Simulation condition: [string]
```

22 Used materials:

[1] C. Angulo et al., Nucl. Phys. A, 656, 3, 1999 [2] Konopinski, E. J. (1943). Beta-Decay. Reviews of Modern Physics, 15(4), 209–245 [3] Star physics, Ivanov, 2018