Astrophysical Simulations Group 1

Code for the N-body simulation. All the files present in this folder, along with other files including scaling test and performance tests, can be found in the following github repository:
https://github.com/sanderdemeyer/Astrofysische_Groep_1

N_body_sim.cpp

This is the final code for the N-body simulation. The units used in the simulations are:

- length: [AU]
- time: [yr]
- mass: $[M_\odot]$
- G = 39.473107 $[AU]^3 M_\odot^{-1} [yr]^{-2}$

The classes are defined in the file **classes.cpp**. The initial conditions are read from the **Initial_conditions** folder. The filenames should not contain any underscores as they are used as separators. The initial condition files have the given format for each line:

$$m_n \quad x_n \quad y_n \quad z_n \quad v_n \quad v_n \quad v_n$$

The folders to save the trajectories and energies are automatically created and therefore require C++17 or above. A folder **dist_mercury** is also created which save the distance and angle of all the bodies in the system with respect to the second body. This is used to study the Solar System and isn't relevant for other initial conditions. The trajectories and energies for some given initial conditions are stored in folders with the names of the corresponding systems in the **traj** en **energy** folders respectively. The naming convention used in each subfolder is: `SystemName_integrator_tmax_h(_adaptive).txt`. The energy files contain two columns with the time and energy. The format of each line of the trajectory files is:

$$t_m \quad x_{1,t_m} \quad y_{1,t_m} \quad z_{1,t_m} \quad x_{2,t_m} \quad y_{2,t_m} \quad z_{2,t_m} \quad \cdots \quad x_{n,t_m} \quad y_{n,t_m} \quad z_{n,t_m}$$

Two types of integrators have been defined

1. `friend`: These integrators are defined using recursive formula's and are defined as a `friend void` class for the `NSystem` class in **classes.cpp**. This is the default integrator type and is used in the `integrate` function (see further).
2. `general`: These are integrators defined as a new class: `General_integrator`. These integrators are defined using Butcher tableau's and used in the `integrate_general` function (see further).

Some functions are defined to run the simulation:

- `integrate`: This function takes some given N-body initial conditions and calculates their trajectories and total energy for a given maximum time using a given integrator and initial timestep. All integrators can be used with an adaptive timestep and if ADAPTIVE_RK45 is true then RK45 will be used. The available integrators are:
  - Forest Ruth
  - Forward Euler
  - Heun
  - Heun3

    - Leapfrog
    - Position Verlet
    - PEFRL
    - Ralston
    - Ralston3
    - RK2
    - RK3
    - RK4
    - Velocity Verlet
    - Yoshida 4
    - RK45: called by a boolean, any of the above given integrators can be given for the `integrator` parameter as the function won't work without this!
- `integrate_general`: This function takes some given N-body initial conditions and calculates their trajectories and total energy for a given maximum time using a given integrator and initial timestep. All integrators can be used with an adaptive timestep. This is essentially the same function as `integrate` with the only difference being that here the integrators are defined using Butcher tableau's. The available integrators are:
    - 3 over 8
    - Ralston4
    - RK4
    - RK5
    - RK6
    - RK8
    - SSPRK3
    - Wray3
- `loop_h`: This function takes some given N-body initial conditions and runs the `integrate` function for fixed timestep using different timesteps in the given range.
- `loop_h_general`: This function takes some given N-body initial conditions and runs the `integrate_general` function for a number of fixed timesteps in the given range.

To use the code, the initial conditions, integrator type, integrator, (initial) timestep and the maximum time to simulate should be given. Along with whether to use adaptive timestep and whether to use RK45 integrator which has embedded adaptive timestep. All integrators can be used with adaptive timestep. To use RK45 the boolean `ADAPTIVE_RK45` should be set to true. To use `ADAPTIVE_RK45` the integrator type needs to be 'friend'. An example to integrate the Burrau initial conditions for 70 years using an initial timestep of 0.001 year using the RK4 integrator of the type `General_integrator` with adaptive timestep is:

```
// File with the initial conditions to be read from the `Initial_conditions`
folder
std::string in_cond = "Burrau.txt";
 // The type of integrator to be used. By default the `friend void` type
integrators are used.
std::string type_integ = "general";
// he type of integrator to be used. For each type, available integrators are
listed in the README file.
std::string integrator = "RK4";
// The (initial) timestep.
double h = 0.001;
```

```
 // Total time considered in the simulation
double tmax = 70;
// Whether or not to use RK45 embedded adaptive timestep. Only implemented in the
`integrate` function.
bool ADAPTIVE_RK45 = false;
// Whether or not to use an adaptive timestep
bool ADAPTIVE_TIME_STEP = true;
// The following should only be changed to loop over run the simulation for
different timesteps.
bool h_loop = false;
// the maximum timestep
double hmax = 0.1;
// the factor by which to loop from `h` to `hmax`. Should be greater than 1.
int step = 10;
```

Only these need to be changed in the code to run other simulations. To loop over different timesteps `h_loop` should be set to true and `hmax` and `step` should be given.

Regularization

The files **`main_2D.cpp`** and **`classes_regularization.cpp`** can be used to simulate 2D systems with regularization. By default, z = 0.

The parameters to be used are the same as for **`N_body_sim.cpp`**, with the exception of:

- `type_integ`: this is always "function"
- `ADAPTIVE_RK45`: this is always false.
- `h_loop`: This is always false.
- `transform_distance`: this is an extra parameter that denotes below which distance 2 bodies are regularized.

Disclaimers: - The code is always 2-dimensional, with $z = 0$ and $v_z = 0$. - The system loses energy upon each regularization. This means that the code is not fully correct. - when transform_distance = 0.0, this is equivalent to **`N_body_sim.cpp`** and works in 3D too. - The trajectories and energies are saved in the folders **`traj_reg`** and **`energy_reg`** respectively which are automatically created upon running the code.