CITS 2401
Computer Analysis and Visualisation

THE UNIVERSITY OF WESTERN AUSTRALIA

Assignment 4
Astronomy

Worth: 4% of the unit

Submission: Answer the questions on the quiz server.
Deadline: 20 October 2020 11:59pm
Late submissions: late submissions attract 5% penalty per day up to 7 days. Afterwards, the mark will be 0 (zero). Also, any plagiarised work will be marked zero.

1. Outline

In this assignment, you will interpret outputs of *soft-particle hydrodynamical* simulations. The data has been provided by **Professor Kenji Bekki** from the International Center for Radio Astronomy Research (ICRAR). It describes the evolution of a globular cluster within a host galaxy. Globular clusters are a dense conglomeration of stars which are found in and around galaxies. They were discovered over 200 years ago and yet today, no one knows how they came about. You do not need to know the background behind these simulations (also the vocabulary) in order to complete this assignment, but the assignment may make you feel like an astronomer☺.

In the provided data file, you are looking at particle data. The file have the structure as shown in Fig 1.

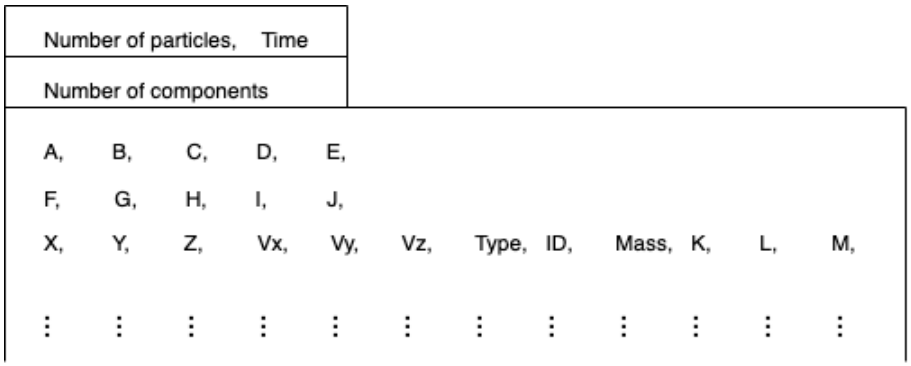

Figure 1: Data structure format.

The X, Y and Z components represent each particles position in space, and V_x, V_y and V_x are the velocities in each direction. Each particle has a unique ID number and an associated mass. Variables A to M are not needed (i.e., ignore them in the analysis). This pattern repeats itself for another two time steps (i.e. three in total – scroll through the whole file).

CITS 2401
Computer Analysis and Visualisation

There are five different types of particles, as shown in Table 1.

Table 1. Types of particles in the dataset

Type	Identifier
Dark matter	0
Disc stars	1
Gas	2
Old stars	3
New stars	4

Dark matter, disc stars and gas particles belong to the galaxy whereas the old and new stars are associated with the globular cluster.

The data is currently in simulation units and must be transformed back into physical units before performing any analysis. To do this, use the conversions using values in Table 2. How to use them will be outlined in the tasks.

Table 2. Parameter values

Parameter	Scaling value
Time	4.7×10^7 years
Mass	$1 \times 10^8 M_\odot$ (solar mass)
Velocity	20.74 km/s
Distance	1×10^3 pc (parsecs)

A skeleton code has been provided to you on the Quiz Server to get you started.

CITS 2401
Computer Analysis and Visualisation

THE UNIVERSITY OF WESTERN AUSTRALIA

2. Tasks

Use the skeleton code provided on the quiz server.

Task 1: Initialise the array
Write a function `init_array()` which returns 5 empty lists as a tuple.

Task 2: Converting to NumPy
Write a function `convert_to_np(components)` which takes an input parameter `components` (a Python list type), and returns a new list with all items in the components converted to NumPy arrays. The `elements` in the `components` are of list type.

Task 3: Scaling values in array
Write a function `scale_arrays(array)` which takes an input parameter `array` (a numpy array type), and applies scales as follows:

- All X, Y and Z values are scaled by `SCALE` global constant value (i.e., multiplied by `SCALE` value).
- All Vx, Vy and Vz values are scaled by `VELOCITY` global constant value (i.e., multiplied by `VELOCITY` value).
- The particle type is casted to integer (see `astype` method).
- All Mass values are scaled by `MASS` global constant value (i.e., multiplied by `MASS` value).

Finally, the function returns the scaled array.

Please note, the input array may be empty (i.e., the size is 0), in which case the passed in array should be returned as is.

Task 4: Generating the time stamps
Write a function `generate_timestamp(components)` which takes an input parameter `array` (a 3D Python list type (i.e., a nested, nested list)), and does the following:

- Converts the input `components` using `convert_to_np` function
- After the conversion, for each item in `components`, apply scaling using `scale_arrays` function

Finally, this function returns the resulting array as a list (i.e., a Python list).

Task 5: Generate 4D data
First, you should understand about what you will be returning from this function. They are (1) data4d and (2) time_array.

data4d

data4d is a 4-dimensional list, where the most outer layer is Python list, while the three inside layers are of type numpy arrays. Simply, the structure of the data4d is:

```
data4d[time step][particle][row][column]
```

For example, if we wish to access all Vx values (index 3) of gas (index 2, determined by its ID) at the first time step (index 0), then you would write:

```
data4d[0][2][:, 3]
```

Note: because the two most inner arrays are of numpy type, we can index in numpy matrix style.

time_array

This is a Python list containing time values from the data (normally 3 values). Initial time is always 0.0, while subsequent time values may vary. For low_res_1.dat file (the one provided), the time values are:

[0.0, 47176249.99999999, 94176250.0]

Now, write a function `generate_4d_data(data_file)` that takes an input parameter `data_file` (a string specifying the file name) that is of the input file described in the instructions. This function does the following:

1. Create an empty list for each particle type (see Table 1 in the instructions) - hint: `init_array` function would be useful.
2. Create a list to save time steps.
3. Iterate over each line from the file (hint: use np.fromstring method to save each row).
    - Keep a list of all time steps (i.e., the row size is 2 as outlined in the instructions), which is the time variable scaled by `SIMTIME` global constant value (i.e., the time * `SIMTIME`).
    - Ignore lines containing variables A-J (i.e., the row size is 5).
    - Keep a list of data line (i.e., the row size is 12) for each particle created in step 1.
4. If the next time step is found, generate the timestamp using the accumulated particles for the time step - hint: use the `generate_timestamp` function.
    - A list should be passed in that includes particles in the order of dark matter, galaxy disc stars, gas, old star and then new star.
    - You should keep the list of timestamps generated.
    - You should reset the list for each particle to be empty.
5. If an unknown ID is found (i.e., ID other than 0-4), then print "`WARNING: unknown ID argument`" and terminates the program by returning None.
6. Finally, return the tuple (list of data4d, list of time steps)

Note: In the provided dataset file `low_res_1.dat`, the next time step is at line 31000, and then another at 62000.

Task 6: Converting to pandas

Write a function `convert_to_pd(col_header, data4d)` which takes two input parameters `col_header` (found in the main function) and `data4d` (from `generate_4d_data` function). Using `pandas` (i.e., imported as pd), you will create 4 pandas data frame objects (hint: use pandas DataFrame method) for the last time step:

- `old`: old star data from data4d
- `new`: new star data from data4d
- `gas`: gas data from data4d
- `gal`: galaxy disc star data from data4d

Remember, their ID represents their index in data4d. See info from the previous task to check how data4d is structured.

For each particle, assign columns attribute to `col_header` (i.e., `columns=col_header`).

Finally, the function returns a tuple containing `old`, `new`, `gas` and `gal` (in this particular order).

CITS 2401
Computer Analysis and Visualisation

THE UNIVERSITY OF WESTERN AUSTRALIA

Task 7: Statistics on particles

Write a function `particle_stats(particle, col_name)` which takes two input parameters `particle` (pandas dataframe object) and `col_name` (Python string). This function will return the minimum, average and maximum values associated with the particle at column `col_name`. If the `col_name` doesn't exist, the function returns `None`.

Below tasks are visualisation questions

Task 8: plotting XY for a single particle

Write a function `plot_single_particle_xy(data4d, time_step, particle, color)` which takes four input parameters `data4d` (from generate_4d_data function), `time_step` (the time step, an integer value), `particle` (an integer value) and `color` (a string) and generates a scatter plot. Your function must return the `plt` object (i.e., `return plt`). Below is instruction on how the scatter plot is setup.

Plot the particle at the time step specified using `data4d`. In particular, this function plots x and y values only (i.e., ignore z for now). Also, you should set scatter plot attributes as following:

- `c`: this is set to `color`, the input parameter.
- `s`: this is set to 1 always.
- `alpha`: this is set to 0.3 always.

Finally, return the `plt` object.

Please note that, the particle value may not be valid (i.e., not between 0 - 4), in which case you should print "`invalid particle id`" and finish (i.e., return None). However, you can assume `data4d`, `time_step` and `color` parameters are all valid.

Task 9: plotting XY for multiple particles

Write a function `plot_multiple_particle_xy(data4d, time_step, particles, colors, alphas)` which takes five input parameters `data4d` (from generate_4d_data function), `time_step` (the time step, an integer value), `particles` (a list of integer values representing particles), `colors` (a list of strings representing a colour), and `alphas` (a list of floats to assign alpha value). This function then generates a scatter plot of all the particles on the same graph. Your function must return the `plt` object at the end (i.e., `return plt`), unless there is an error. Below is instruction on how the scatter plot is setup.

Plot the particle at the time step specified using `data4d`. In particular, this function plots x and y values only (i.e., ignore z for now). Also, you should set scatter plot attributes as following:

- `c`: this is set to color from the `colors` list in order.
- `s`: this is set to 1 always.
- `alpha`: this is set to alpha from the `alphas` list in order.

For example, we have:

```
particles = [1, 2, 3]
colors = ['red', 'blue', 'yellow']
alphas = [0.1, 0.5, 1.0]
```

Then particle 1 (i.e., disc star) gets colour red and alpha value 0.1, particle 2 (i.e., gas) gets colour blue and alpha value 0.5, and finally particle 3 (i.e., old star) gets color yellow and alpha value 1.0.

Finally, return the `plt` object.

Please note that, the particle value may not be valid (i.e., not between 0 - 4), in which case you should print "`invalid particle id`" and finish (i.e., return None). However, you can assume all other input parameter values are valid.

Task 10: Visualising stars

Write a function `star_formation_visualisation(data4d)` which takes one input parameter `data4d` (from generate_4d_data function). This function then generates scatter subplots of XY and XZ graphs for each `timestep` (i.e., 2 rows x 3 columns chart, top row is XY plots, and bottom row is XZ plots). Your function must return the `plt` object at the end (i.e., `return plt`), unless there is an error. Below is instruction on how the scatter plot is setup.

<u>At time step 0:</u>

- Plot XY (row 0) and XZ (row 1) for gas and new star particles (in this order).
- Set gas particles colour to `gas_col` (e.g., `c=gas_col`), `s` = 1, and `alpha` to 0.3
- Set new star particles colour to `new_col`, `s` = 1, and `alpha` = 0.1

<u>At time step 1:</u>

- plot XY (row 0) and XZ (row 1) for gas, old star and new star particles (in this order).
- Set gas and new star particles the same as time step 0.
- Set old star particles with colour `old_col` and `s` = 1 (not specifying alpha to use default value).

<u>At time step 2:</u>

- plot XY (row 0) and XZ (row 1) for gas, old star and new star particles (in this order).
- Set all particles the same as time step 1.

For all charts, set the axis limits as:

```
axe[row, col].set_xlim(-ax_lims, ax_lims)
axe[row, col].set_ylim(-ax_lims, ax_lims)
```

CITS 2401
Computer Analysis and Visualisation

THE UNIVERSITY OF WESTERN AUSTRALIA

Task 11: Compute center for displaying particles

Write a function `compute_center(particles, center)` which takes two input parameters `particles` (a list of particles old star, new star, gas and galaxy disc star in that order) and `center` (numpy array of XYZ positions of the first old star particle in the last time step observed). This function then generates new columns in the particles with `x_scale`, `y_scale` and `z_scale` columns.

For each particle, define new columns `x_scale`, `y_scale` and `z_scale` set to existing `x`, `y` and `z` values minus the `x`, `y` and `z` center values from `center`.

That is, the `x_scale` value is set to `particle.direction - center[direction]`

Here, you will find the `assign` method of the pandas dataframe (type of each particle) useful.

Finally, the function returns the tuple containing old star, new star, gas and galaxy disc star as a tuple (in this specific order).

Once complete, you can pass these particles to the function individual_particles (already implemented for you) along with data4d

i.e., call: `individual_particles(data4d, particles)`

That's the last assignment folks!