Intrdouction

Refer the `algebra.pdf` file for the algebra behind the project.

The repo contain the following files and folders

```
user0@mypc:~/NSwDM$ ls
binary          common  exec_binary.sh  gw_ft  purge_data.sh  run_gw_ft.sh
binary_conf.c   data    find_gw_ft.sh   head   readme         SNR
```

The following introduce their basic usage.

Generation of binary evolution data

The basic step for generating binary evolution data is shown below

1. Setting up

First edit the `binary_conf.c` file for setting the binary parameters.

```c
int main()
{
    struct Pa p; // Defined in setting.h header file

    p.M_A = 1.4; // Star A total mass (M+m) in (solar mass)
    p.p_A = 0.0; // Star A m/M ratio
    p.m_A = p.p_A / (1 + p.p_A); // Star A m/(M+m) ratio

    p.M_B = 1.4; // Star B total mass (M+m) in (solar mass)
    p.p_B = 0.0; // Star B m/M ratio
    p.m_B = p.p_B / (1 + p.p_B); // Star B m/(M+m) ratio

    p.n_A = 0.8; // Star A polytropic index
    p.R_0_A = 12.0; // Star A radius without tide and dark matter core in unit of (km)
    p.f_R_A = -2.0; // Irrotation star A as Riemann ellipsoid
    p.r = 8.0; // Initial orbital separation in unit of (R_0_A)

    /* Set to 1 to trigger the corresponding post-Newtonian terms, 0 otherwise.
     * They modifythe post-Newtonian terms in evolution equation
     * and initial orbital frequency
     */
    p.pn.PNc = 1; // Conservation Post-Newtonian terms
    p.pn.PN1 = 1; // PN1 order
    p.pn.PN2 = 1; // PN2 order
    p.pn.PN25 = 1; // Disspiative PN2.5 order
    p.pn.PN3 = 1; // PN3 order

    if (p.pn.PNc == 0 ) {
        p.pn.PN1 = 0;
        p.pn.PN2 = 0;
        p.pn.PN3 = 0;
    }
```

Edit the following 3 lines for Yukawa force (also called dark repulsion in comment)

```c
    p.dark.on = 0; // Set to 1 to trigger dark repulsion, 0 otherise (again affect the initial orbital frequency and evolution eq
uation)
    p.dark.alpha = 0.01; // Dark repulsion coupling constant alpha prime
    p.dark.mV = 20.0; // Dark repulsion mediator range in km
```

When the parameters are set, the `eolve(p);` in the `binary_conf.c` file will generate the evolution data file by solving initial conditions and the ODEs.

```c
    evolve(p); // evolve the binary and produce the data file, comment out this line if choose to loop through using above comme
nted codes
```

2. Execution of `binay_conf.c` file

The execution require the GNU Scientific Library (GSL) package be installed on the machine. See [(https://www.gnu.org/software/gsl/)]

The `binary_conf.c` file in `NSwDM/` folder is copied into the `binary/` folder. The C codes in `binary/` is then compiled and executed. The above operations are done by executing the bash script `exec_binary.sh` :

```
#! /bin/bash

cp binary_conf.c binary/
cd binary/
echo "Compiling with gcc..."
gcc -o run binary_conf.c evolve.c -lgsl -lgslcblas -lm

echo "Evolving the binary..."
./run

# Move the binary configurations file to folder "head/" in ../
echo "Saving the binary configurations file..."
mv HEAD* ../head/

# Copy data in binary/ to the folder "data/" in ../
cp -r data ../

# Cleaning the folder "binary/data/"
rm -vf data/bh/no_dark/*
rm -vf data/bh/dark/*
rm -vf data/dark/*
rm -vf data/dm/*
```

Execution of `exec_binary.sh` : (The distance and step are measured in meters)

```
user0@mypc:~/NSwDM$ ./exec_binary.sh
Compiling with gcc...
Evolving the binary...
Writing Configuration file...
Real time = 0 (s)       binary time = 0.000 (ms)        distance = 96000.000     step = 0.0146226574
Real time = 10 (s)      binary time = 40.733 (ms)       distance = 92580.884     step = 4472.1385712344
Real time = 20 (s)      binary time = 82.675 (ms)       distance = 88764.733     step = 7413.2579154272
Real time = 30 (s)      binary time = 125.407 (ms)      distance = 84007.837     step = 7987.2883631170
Real time = 40 (s)      binary time = 169.887 (ms)      distance = 78277.905     step = 4011.2491794535
Real time = 49 (s)      binary time = 220.261 (ms)      distance = 69406.375     step = 6072.9803353790
Real time = 58 (s)      binary time = 294.219 (ms)      distance = 32030.899     step = 8327.6241379408

Saving data to file...

Total time elapsed = 58 (s)

*****************Program Done*****************
Saving the binary configurations file...
removed 'data/dm/C1P123_D2_MA1.40_mA0.000_MB1.40_mB0.000_RA12.0_nA0.8_spin-2.0_e0.0_rp8.0_pA0.00_pB0.00.txt'
user0@mypc:~/NSwDM$
```

3. The evolution data file

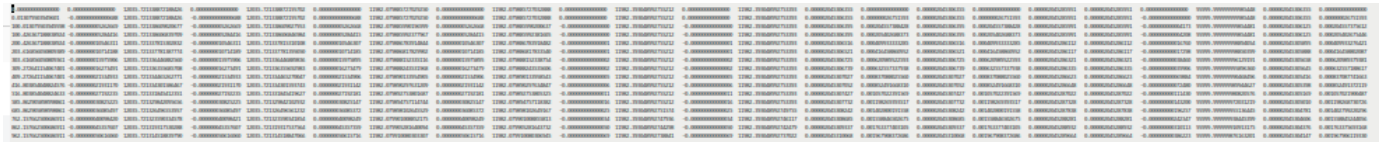
The data file for binary evolution (with / without) Yukawa force is stored in (`data/dark/` / `data/dm/`) folder:

```
user0@mypc:~/NSwDM$ cd data/dm
user0@mypc:~/NSwDM/data/dm$ ls
C1P123_D2_MA1.40_mA0.000_MB1.40_mB0.000_RA12.0_nA0.8_spin-2.0_e0.0_rp8.0_pA0.00_pB0.00.txt
user0@mypc:~/NSwDM/data/dm$
```

The file name is explained as follow:

Name	Meaning		Name	Meaning
C1P123	Conservative PN terms with order 1, 2, 3		D2	Dissipative PN terms with order 2.5
MA1.40	Star A total mass = 1.4 solar mass		mA0.000	Star A DM core percentage = 0%
MB1.40	Star B total mass = 1.4 solar mass		mB0.000	Star A DM core percentage = 0%
RA12.0	Star A reference radius = 12.0 km		nA0.8	Star A polytropic index = 0.8
spin-2.0	Riemanna Ellipsoid		e0.0	initial orbital eccentricity = 0.0
rp8.0	Initial orbital separation = 8.0 (Star A reference radius) = 8 * 12.0km = 96.0 km		pA0.0	Star A DM core to Normal Matter ratio = 0.0
pB0.0	Star B DM core to Normal Matter ratio = 0.0			

The `C1P123_D2_MA1.40_mA0.000_MB1.40_mB0.000_RA12.0_nA0.8_spin2.0_e0.0_rp8.0_pA0.00_pB0.0 0.txt` data file contain 23 columns:



From left to right, the columns are data for (all are in geometric unit)

1. time t
2. $\dot{a}_1(t)$
3. $a_1(t)$
4. $\dot{a}'_1(t)$
5. $a'_1(t)$
6. $\dot{a}_2(t)$
7. $a_2(t)$
8. $\dot{a}'_2(t)$
9. $a'_2(t)$
10. $\dot{a}_3(t)$
11. $a_3(t)$
12. $\dot{a}'_3(t)$
13. $a'_3(t)$
14. $\Omega(t)$
15. $\phi(t)$

16. $\Omega'(t)$
17. $\phi'(t)$
18. $\Lambda(t)$
19. $\Lambda'(t)$
20. $\dot{r}(t)$
21. $r(t)$
22. $\dot{\theta}(t)$
23. $\theta(t)$

4. The binary configuration file

For each generation of binary evolution data file, a header file containing the corresponding binary parameters are generated and stored at `head/` folder

```
user0@mypc:~/NSwDM/head$ ls
HEAD_C1P123_D2_MA1.40_mA0.000_MB1.40_mB0.000_RA12.0_nA0.5_spin-2.0_e0.0_rp7.0_alpha0.01_mV20.0_pA0.00_pB0.00.txt
HEAD_C1P123_D2_MA1.40_mA0.000_MB1.40_mB0.000_RA12.0_nA0.6_spin-2.0_e0.0_rp7.0_alpha0.01_mV20.0_pA0.00_pB0.00.txt
HEAD_C1P123_D2_MA1.40_mA0.000_MB1.40_mB0.000_RA12.0_nA0.8_spin-2.0_e0.0_rp8.0_pA0.00_pB0.00.txt
```

They are stored for the purpose of more binary configuration details.

Calculation of gravitational waves (GW) h_+ and $h_\times$ polarization mode and it's fourier transform (FT)

Using the evolution data file, the orbital acceleration $\ddot{r}$ and $\ddot{\theta}$ can be calculated, which can be used to calculated h_+ and $h_\times$. The calculation is done in the `gw_ft/` folder.

1. Generation of GW and FT data file

To calculate the GWs h_+ and $h_\times$ polarization mode and it's fourier transform for a particular data file (`C1P123_D2_MA1.40_mA0.000_MB1.40_mB0.000_RA12.0_nA0.8_spin2.0_e0.0_rp8.0_pA0.00_pB0.00.txt` in this case), issue the following command in `NSwDM/` folder:

- `./find_gw_ft.sh dm C1P123_D2_MA1.40_mA0.000_MB1.40_mB0.000_RA12.0_nA0.8_spin-2.0_e0.0_rp8.0_pA0.00_pB0.00.txt`

where two argument is needed for the `./find_gw_ft.sh` script:

- `./find_gw_ft.sh <dm/dark> <data file name>`

The first argument specify the folder `data/dm` or `data/dark`.

2. The GW and FT data file

After the `./find_gw_ft.sh` is successfully run. The GW data file is stored in `data/gw/dm` (or `data/gw/dark/`) with file name

- `GW_C1P123_D2_MA1.40_mA0.000_MB1.40_mB0.000_RA12.0_nA0.8_spin-2.0_e0.0_rp8.0_pA0.00_pB0.00.txt`

which consist of 3 columns:

```
0.000000000000000       -0.077106181676697       -0.000021150306855
0.013079503949601       -0.077106181675565       -0.000021154432281
100.013079503949598     -0.077106157257490       -0.000052695568527
100.426367188838924     -0.077106157168292       -0.000052825924185
```

From left to right, they are data for $t(\text{in unit of m}), h_+(t), h_\times(t)$. Now denote their fourier transform by $\tilde{h}_+(f)$ and $\tilde{h}_\times(f)$ respectively. The fourier transform data file is stored in `data/fourier/dm` (or `data/fourier/dark/`) with file name

- `F_GW_C1P123_D2_MA1.40_mA0.000_MB1.40_mB0.000_RA12.0_nA0.8_spin2.0_e0.0_rp8.0_pA0.00_pB0.00.txt`

which consist of 5 columns:

```
4992 16930.4 7.64294e-06 16930.4 2.15277e-05
4993 16933.8 7.64295e-06 16933.8 2.15276e-05
4994 16937.2 7.64295e-06 16937.2 2.15276e-05
4995 16940.5 7.64295e-06 16940.5 2.15276e-05
```

From left to right, they are data for $n(\text{index}), f, |\tilde{h}_+(f)|, f, |\tilde{h}_\times(f)|$. (The 2nd and 4th columns are identical).

- The frequency f is in unit of (Hz) and $|\tilde{h}_+(f)|$ is in unit of (second)
- Note $h_+(t), h_\times(t), \tilde{h}_+(f), \tilde{h}_\times(f)$ are values without multiplying the distance to source and binary reduced mass. i.e. $h_+ = (\dot{r}^2)\cos 2\theta + r\ddot{r}\cos 2\theta - 4r\dot{r}\dot{\theta}\sin 2\theta - 2r^2\dot{\theta}^2\cos 2\theta - r^2\ddot{r}\sin 2\theta$

Signal to noise ratio (SNR)

To calculate the SNR for a particular binary configuration:

1. Go to the `SNR/` folder
2. Execute the compiled C file by `./snr <F_GW_....txt>`, where the `<F_GW_....txt>` is the fourier transform data file.
3. The value of SNR will be printed on the terminal

To calculate SNR for multiple binary configuration, a bash script `snr.sh` in the `SNR/` folder may help. Refer to the `snr.sh` file for details.

- The integrand in SNR calculation is $(|\tilde{h}_+(f)|^2 + |\tilde{h}_\times(f)|^2)/S_n(f)$ where $S_n(f)$ is the one-sided power spectral density (PSD) for noise. Thus, the value of SNR printed in the terminal is the value without factoring the reduced mass and distance to source.

BH-BH binary evolution

To run BH-BH binary evolution, simply replace the `evolve(p);` in `binary_conf.c` to `evolve_BH(p);`. This will generate evolution data file and stored to `data/bh/dark` or (`data/bh/no_dark/`). The resultant data file consist of 5 columns which are $t, \dot{r}, r, \dot{\theta}, \theta$.

- The GW and FT C code/bash script do NOT support the data file for BH-BH.

Others

- The `binary/` and `gw_ft/` folder have 3 common files: `func_lib.h`, `poly_const_table.csv`, `setting.h`, thus, to edit these file, it is suggested to edit them in the `common/` folder and run the `sync_common.sh` script to update the 3 common files in `binary/` and `gw_ft/` folder
- If the whole `NSwDM/` folder become too large, it is suggested copy the `data/` and `head/` folder to another location and then run `purge_data.sh` script to delete all data file in `data/` and `head/`.
- The `poly_const_table.csv` contains the polytropic constants calculated by a python script. The script is not included in the repo. Note only polytropic constants for polytropic index n = 0.5, 0.6, 0.7, ..., 1.4 are calculated.

About solving for initial conditions

We solve 6 equilibrium equations : $f_j(a_1, a_2, a_3, a_1', a_2', a_3'; r)$, with orbital separation r as parameter. As suggested in the reference paper LRS4, before solving it using newton's method, we normalize the variable a_i and parameter r by:

$$\hat{a}_i = \frac{a_i}{a_1 + a_1'}, \quad \hat{a}_i' = \frac{a_i'}{a_1 + a_1'}, \quad \hat{r} = \frac{r}{a_1 + a_1'}$$

Thus we are using the normalized orbital separation $\hat{r}$ to solve for normalized axes $\hat{a}_i$ from $f_j(\hat{a}_i, \hat{a}_i'; \hat{r})$. As an example, from the reference paper LRS4, the ratios of the axes $a_{2/3}/a_1$ are obtained by first specifying a particular value of $\hat{r}$:

TABLE 1
SEQUENCES OF COMPRESSIBLE DARWIN MODELS [a] WITH $K = K'$

$\hat{r}$ [b]	$\bar{r}$	a_2/a_1	a_3/a_1	R/R_0	a_2'/a_1'	a_3'/a_1'	R'/R_0'	$\bar{\Omega}$	$\bar{J}$	$\bar{E}$
				$n = 0, \quad p = 0.8$						
3.0	3.0459	0.9809	0.9702	1.	0.9847	0.9740	1.	0.1089	2.1913	-1.5645
2.5	2.5656	0.9673	0.9499	1.	0.9739	0.9562	1.	0.1410	2.0496	-1.5797

(Note $\hat{r} = 3.0$ or 2.5). After the normalized axes length are found, their physical value can be calculated by the following equations by specifing a value of R_0, this R_0 value also determined the value of R_0':

$$R = (a_1 a_2 a_3)^{1/3}, \quad R' = (a_1' a_2' a_3')^{1/3}, \quad R = R(\hat{a}_i, \hat{a}_i'; \hat{r}, R_0), \quad R' = R'(\hat{a}_i, \hat{a}_i'; \hat{r}, R_0')$$

And now we can calculate the physical value of orbital separation r via :

$$r = \hat{r}(a_1 + a_1')$$

Implementation in the C codes

The physical value of r is specified in `binary_conf.c` file, however, since a simple formula for obtaining $\hat{r}$ from r (without needing to know the value of a_i, a_i') could not be found, we need to solve

the 6 equlibrium euqations f_j by a initial guess of $\hat{r}$, and then proceed to solve f_j and obtain physical value of r. In summary, we need to solve the equation

$$r(\text{solved from } f_j \text{ by a guess } \hat{r}) - r(\text{set in binary configuration c file}) = 0$$

Ths equation is implemented in `func_r` function (capped from `func_lib.h` file in `common/` folder)

```
/* The func_r provide the equation r/R_0(solved from the above newton_6d) - r/R_0(in binary_conf.c) = 0
 * which we seek to solve for r/R_0(solved from the above newton_6d) by secant method
 */
double func_r(double x, void * params)
{
    struct Pa p = * ((struct Pa *) params);
    double aim = p.r; // Set the variable aim = p.r chosen in binary_conf.c
    p.r = x; // Assign a random guess orbital separation x = r/(a1+a1') to p.r

    /* Obtain the equilibrium configuration from the random guessed orbital separation x
     * Compare the solved phyical orbital separation r to aim = p.r chosen in binary_conf.c
     */
    double r_phy = newton_6d(p).r_phy / p.R_0_A;
    return r_phy - aim;
}
```

And solved by secant method with initial guess $\hat{r} = 5.0$ and 6.0:

```
/* Obtain the suitable x = r/(a1 + a1') used in solving the equlibrium axes length of the ellipsoid*/
double secant_r(double func(double, void *), void * params)
{
    struct Pa p = * ((struct Pa *) params);
    double ans, x1, x0, err;
    x1 = 6.0; // Initial guess 1
    x0 = 5.0; // Initial guess 2
    err = fabs(x1 - x0); // Initial error
    int i = 0;
    while (err > 1e-15)
    {
        double f_x1 = func(x1, &p);
        double f_x0 = func(x0, &p);
        double new_x = x1 - f_x1 * (x1 - x0) / (f_x1 - f_x0);

        x0 = x1;
        x1 = new_x;
        err = fabs(x1 - x0); // Update error until it is within the desired range (1e-15)
        i++;
    }

    ans = x1;
    //printf("total iteration = %d suitable r scaled is % .5lf\n", i, ans);
    return ans;

}
```

After a correct value of $\hat{r}$ is obtained, then the initial axes length are obtained by solving f_j

---