Delay Time Distribution of binay Compact Objects

Students:
Aidin Attar
Roya Joulaei Vijouyeh
Bahador Amjadi
Mojtaba Roshana

Prof. Michela Mapelli

27 June 2022

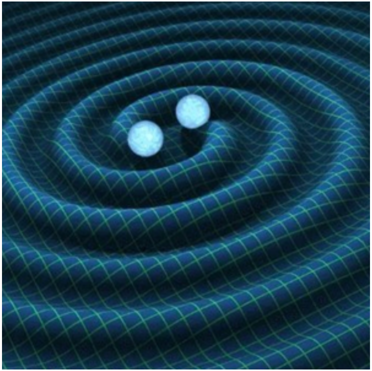
Università degli Studi di Padova

The first gravitational wave has been detected in September 14, 2015. However, "Gravitational waves were first predicted by Albert Einstein a century ago on the basis of his general theory of relativity."

Binary Compact Objects are binary systems composed of two compact objects (black holes or neutron stars) originating from the evolution of a massive binary star

The **Delay Time** is the time between the formation of a binary star and the merger of the two black holes that form from the binary star.

- The merger is caused by **gravitational-wave** emission
- The delay time depends on the **black hole masses**, **semi-major axis** and **eccentricity**
- **To Calculate the Delay Time** $\rightarrow$ integrate the system of two ODE



Gravitational Waves

Frequency GRAVITATIONAL WAVES from a BINARY STAR are MONOCHROMATIC with frequency = 2 orbital frequency

$$\omega_{GW} = 2\omega_{orb} = 2\sqrt{\frac{G(M+m)}{a^3}} \tag{1}$$

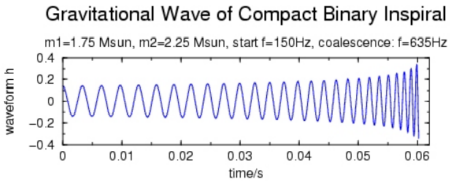
Polarization

$$h_{+} \sim \frac{16\, G^2}{c^4} \frac{M^2}{r\, a} \cos\left(2\,\omega\,(t - r/c)\right)$$
$$h_{\times} \sim \frac{16\, G^2}{c^4} \frac{M^2}{r\, a} \sin\left(2\,\omega\,(t - r/c)\right)$$

AMPLITUDE of GWs from binary systems:

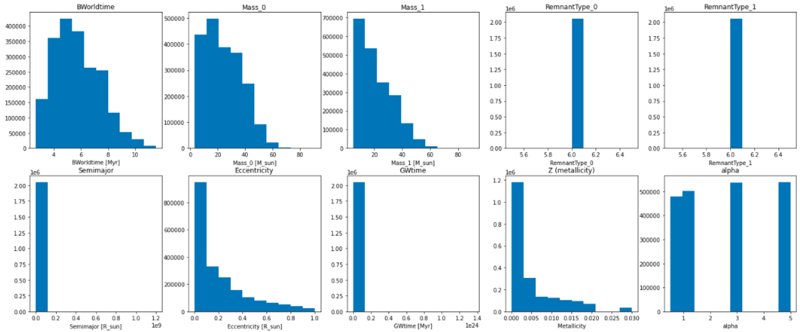
$$h = \frac{1}{2}(h_+^2 + h_x^2) \sim \frac{8G^2M^2}{c^4ra} \qquad (2)$$

- the bigger the amplitude (strain), the easier the detection
- the farther the binary, the smaller the amplitude
- the larger the masses, the larger the amplitude
- the smaller the semi-major axis, the larger the amplitude

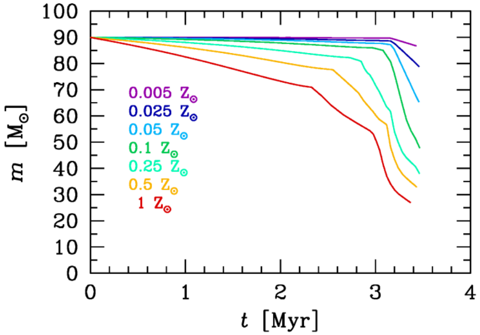


THE DATA

The data which is provided in this research is come from simulations of binary compact object formation.

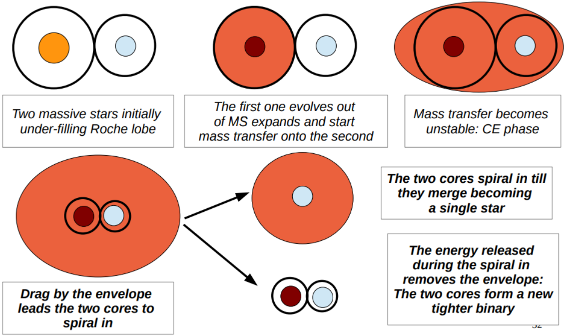


MASS LOSS DEPENDS ON **METALLICITY**



COMMON ENVELOPE

One of the most important stages in the evolution of close binary stars is common envelope (CE) evolution. It is commonly thought to occur when the expanding primary star transfers mass to its companion at a too high rate that the companion cannot accrete it. This leads to the companion star being engulfed by the envelope of the primary. The orbital energy and angular momentum of the orbiting components are then transferred into the CE, resulting in the orbital decay and spiral-in of the star



Università degli Studi di Padova

The main goals of the Projects are as following:

- Calculation of the delay time distribution and plotting the their distribution
- Obtaining a fitting formula for the resulting distribution
- Training a random forest to learn the delay time distribution

The System of Ordinary Differential Equations:

$$\frac{da}{dt} = -\frac{64}{5}\frac{G^3 Mm(M+m)}{c^5 a^3 (1-e^2)^{7/2}}(1+\frac{73}{24}e^2+\frac{37}{96}e^4) \tag{3}$$

$$\frac{de}{dt} = -\frac{304}{15}e\frac{G^3 Mm(M+m)}{c^5 a^4 (1-e^2)^{5/2}}(1+\frac{121}{304}e^2) \tag{4}$$

a: semi-major axis
e: orbital eccentricity
M: primary mass
m: secondary mass
G: gravity constant
c: speed of light

Equations obtained by **Peters** (1964) combining **Keplerian celestial mechanics** with the **quadrupole formula** to obtain the equations for the GW induced evolution of orbital parameters.

Euler Scheme

The general form of a first-order one-variable **ordinary differential equation** is

$$\frac{dy}{dx} = f(y, x) \tag{5}$$

Considering the **Taylor expansion**

$$y(x+h) = y(x) + \frac{dy}{dx}h + \frac{1}{2}\frac{d^2y}{dx^2}h^2 + \cdots = y(x) + f(y,x)h + \mathcal{O}(h^2)$$

and neglecting the terms of order **higher than** h.

$$y(x+h) = y(x) + f(y,x)h \tag{6}$$

Equation 6 defines the **explicit Euler's method**. This is a **first-order method**: the errors scale as h^2.

```python
def ODE_EU( xin, yin, h, M, m ):
    ...
    dydx = deriv( xin, yin, M, m )
    for i in range(N):
        yout[ i ] = yin[ i ] + h * dydx[ i ]

    return yout
```

Python implementation of the Euler Method.

Fourth Order Runge-Kutta method

Skipping intermediate methods such as the **Midpoint method**, we consider the **fourth order** version of the **Runge-Kutta methods**, obtained by considering the terms further than h in the Taylor Expansion.

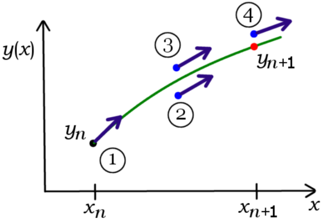
The equations look like:

$$\begin{aligned}
Y_1 &= y_i \\
Y_2 &= y_i + f(Y_1, t_i)\frac{\Delta t}{2} \\
Y_3 &= y_i + f(Y_2, t_i + \Delta t/2)\frac{\Delta t}{2} \\
Y_4 &= y_i + f(Y_3, t_i + \Delta t/2)\Delta t \\
y_{i+1} &= y_i + [f(Y_1, t_i) + 2f(Y_2, t_i + \Delta t/2) + 2f(Y_3, t_i + \Delta t/2) + f(Y_4, t_i)]\frac{\Delta t}{6}
\end{aligned} \tag{7}$$

Equation 7 defines the **4th order Runge Kutta scheme**. This is a **fourth-order method**: the errors scale as h^5.

The 4th-order RK requires four evaluations of the righthand side per step h

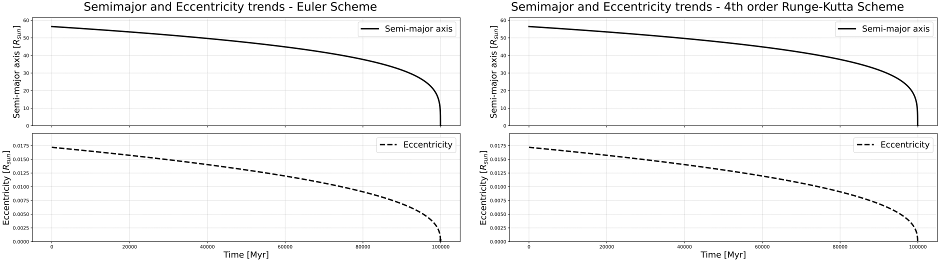
- k1 is the slope at the beginning of the time step
- k2 and k3 are the derivatives at the trial midpoint
- k4 is the slope at the trial endpoint




```python
def ODE_RK( xin, yin, h, M, m ):
    ...
    dydx = deriv( xin, yin, M, m )
    for i in range(N):
        k1[ i ] = h              * dydx[ i ]
        yt[ i ] = yin[ i ] + 0.5 *   k1[ i ]

    dydx = deriv( xh, yt, M, m )
    for i in range(N):
        k2[ i ] = h              * dydx[ i ]
        yt[ i ] = yin[ i ] + 0.5 *   k2[ i ]

    dydx = deriv( xh, yt, M, m )
    for i in range(N):
        k3[ i ] = h              * dydx[ i ]
        yt[ i ] = yin[ i ] +         k3[ i ]

    dydx = deriv( xin, yt, M, m )
    for i in range(N):
        k4  [ i ] = h * dydx[ i ]
        yout[ i ] = yin[ i ]  + k1 [ i ] / 6. + k2 [ i ] / 3. \
                              + k3 [ i ] / 3. + k4 [ i ] / 6.
    return yout
```

Python implementation of the fourth order Runge-Kutta.

Plot of the results



Integration path for the semi-major axis and the eccentricity of the orbit for a random sample.

	$M[M_{SUN}]$	$m[M_{SUN}]$	$a_0[R_{SUN}]$	$e_0[R_{SUN}]$	$t_{del}[Myr]$	Computation time
Euler	21.09631	18.34303	0.01719063	56.46466	99934.8	$\sim$ 11s
RK4	21.09631	18.34303	0.01719063	56.46466	99934.2	$\sim$ 46s

Parameters and results for the two methods.

Fixed time step computation time

Considering an average time of $\sim$ 15s for the Euler scheme:

$$Time_{comp} = 15s \times 2 \cdot 10^6 entries \simeq 1yr$$

We want to choose a value of h that is a good compromise between accuracy and speed of calculation.

- a **smaller value** of h means **higher accuracy**, but an exceedingly small h might result in a **computational challenge**
- a **larger value** of h means **lower accuracy** (negative a and e as soon as gravitational waves start being important in our case), but an **fast computation**

Solution

Adapt the step size to the region's characteristics:

- **flatter** regions: **larger** step size
- **stiffer** regions: **smaller** step size

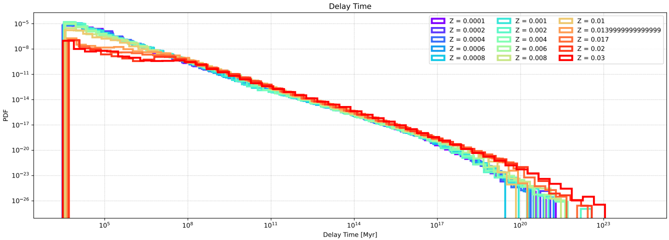
Strategy for our problem

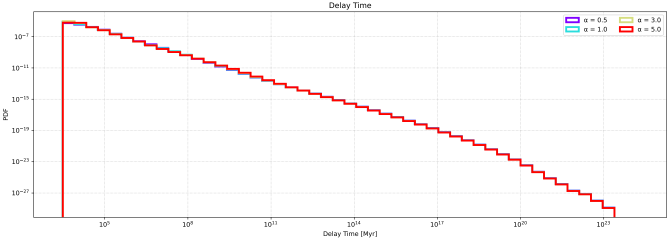
An option is that we require the relative variation of the semi-major axis to be nearly constant, or at least smaller than a chosen tolerance, during the integration.

```python
def delay_time(row, function, h, t):
    ...
    while a > r_sc:
        a_new, e_new = function( t, (a, e), h, M2, M1 )

        if abs( a_new - a )/a < (0.1*tol):
            h *= 2
            a_new, e_new = function( t, (a, e), h, M2, M1 )

        elif abs(a_new - a)/a > tol:
            while abs(a_new - a)/a > tol:
                h /= 10.
                a_new, e_new = function( t, (a, e), h, M2, M1 )

        a, e = (a_new, e_new)
        t += h

    return pd.Series([t, e])
```

Python implementation of the Adaptive step size.

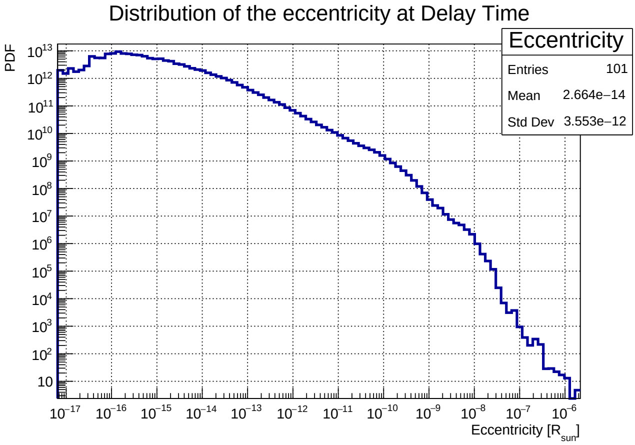
Distribution of the delay time





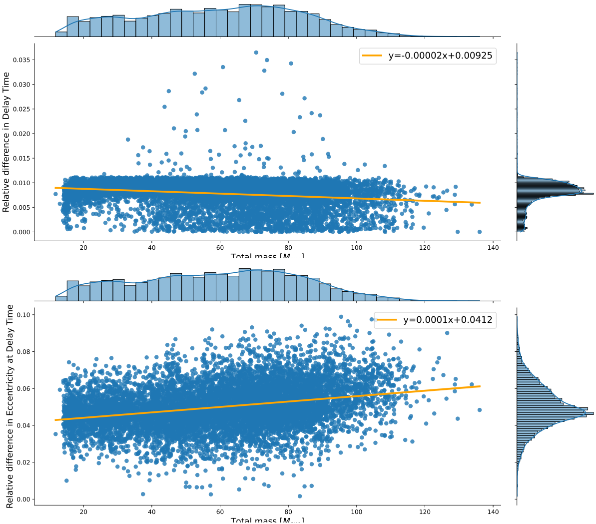
Distribution of the delay time for different values of metallicity (top) and efficiency of the common envelope (bottom).

A study was done also for the value of Eccentricity when the two compacts merge.



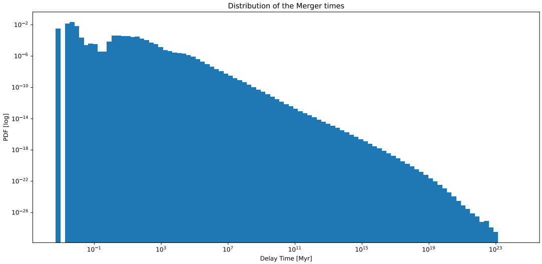
Value of eccentricity of the orbit at the delay time.

Euler vs. RK4



Relative difference in Delay Time and Eccentricity at the Delay time between the RK4 and Euler method for a subsample the dataset.

We have to fit a curve on some distributions of the mergers time which we computed



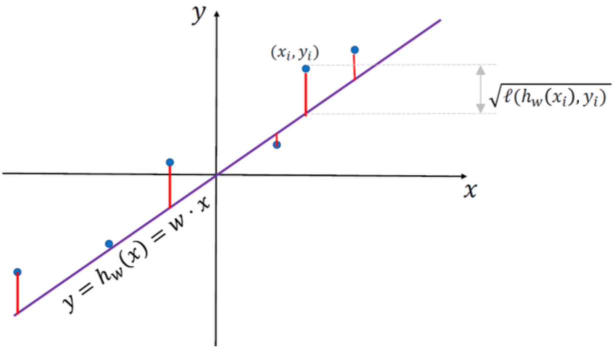
We tried tow packages to fit a curve:

- **Linear Regrassion scipy (on logaritmic histogram)**
- **ROOT**

Università degli Studi di Padova

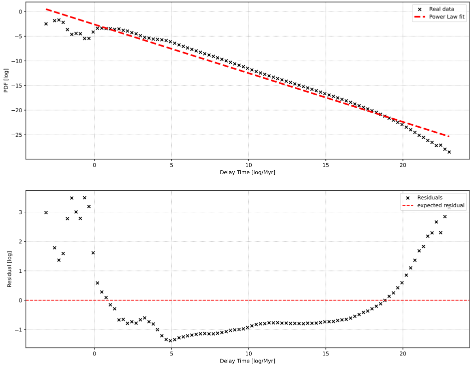
Optimizing w to find best line which can be fit on our data to get minimum error

$$\mathcal{X} = \mathbb{R}^1 \; \mathcal{Y} = \mathbb{R}$$



fitting a curve on DelayTime numerical computation of ODE:

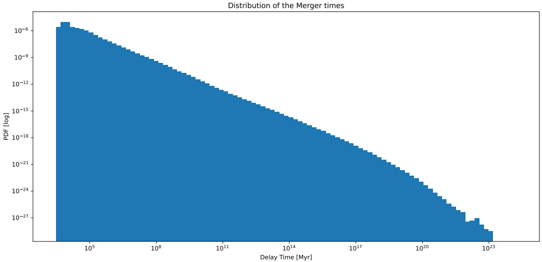
- slope : -0.98 ± 0.03
- intercept : -2.6 ± 0.5
- χ^2 statistic: 28.6
- $R^2 = 0.967$



Fitting : Add BWtime

Here we added age of the systems to the DelayTime computed which we measured before

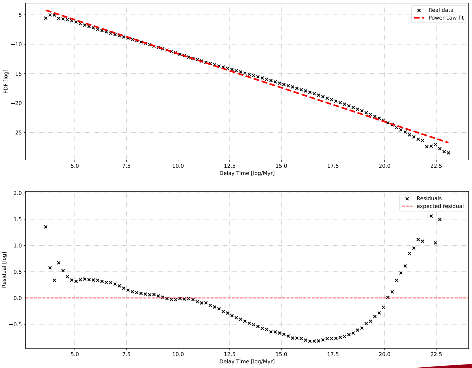
- **BWtime : age of the system**
- **DelayTime computed : The time which we get by computing the ODE**



DelayTime computed + BWtime

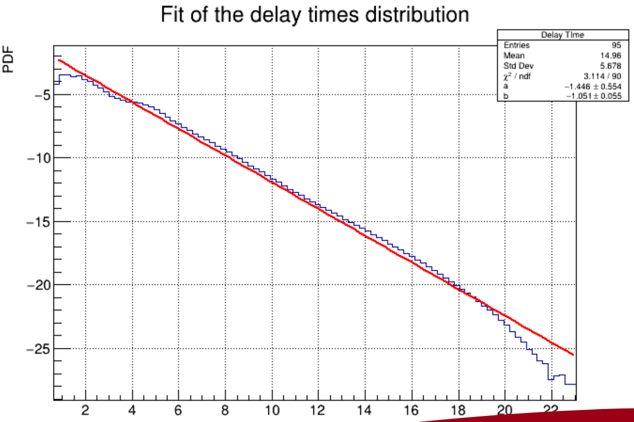
At first we tried to fit a power law on this distribution by linear regression (scipy) but it not behave like a power law exactly

- slope : -1.16 ± 0.02
- intercept : -0.06 ± 0.34
- χ^2 statistic: 2.32
- $R^2 = 0.990$



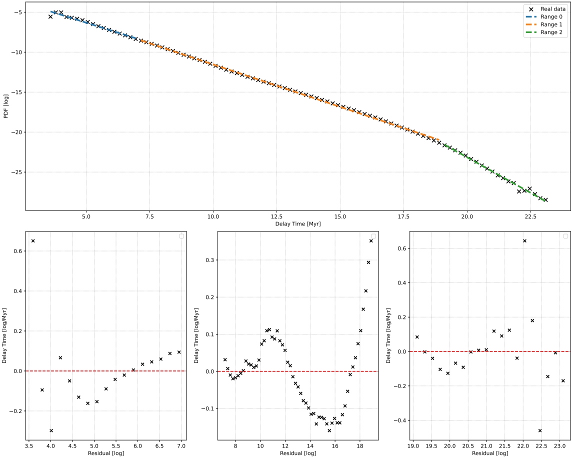
Also it has been done by PyROOT At the short merger time and long merger time distribution

- slope : -1.12 ± 0.06
- intercept :-0.5 ± 0.8
- χ^2 statistic: 1.66



Fitting : Different ranges

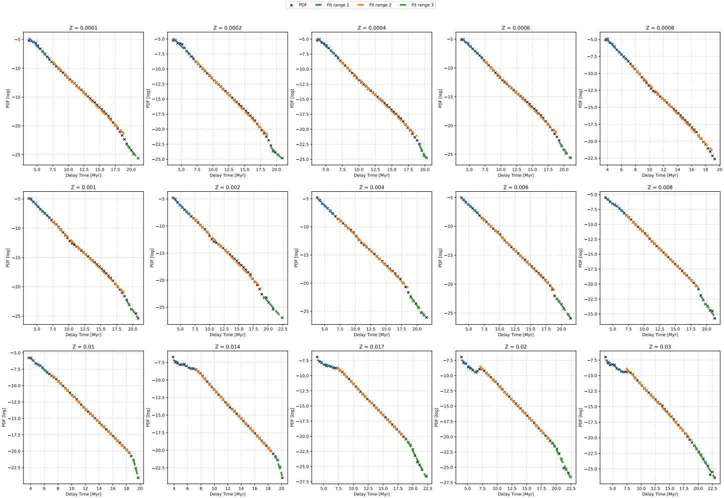
The PDF does not behave like a power law at first and the last parts, therefore we split it in three different ranges



We got better fits for for different ranges (0, 10^7, 10^{19}, 10^{30}):

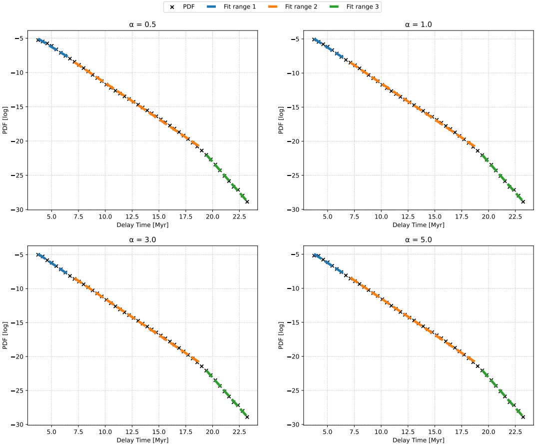
- First range[0, 10^7]:
  - slope : -0.996 $\pm$ 0.103
  - intercept : -1.332 $\pm$ 0.557
  - χ^2 statistic: 0.11322
  - R^2 : 0.965
- Middle range[10^7, 10^{19}]
  - slope : -1.060 $\pm$ 0.008
  - intercept : -0.941 $\pm$ 0.113
  - χ^2 statistic: 0.037
  - R^2 : 0.999
- Last range [10^{19}, 10^{30}]
  - slope : -1.776 $\pm$ 0.081
  - intercept : 12.368 $\pm$ 1.727
  - χ^2 statistic: 0.029
  - R^2 : 0.991

We fit some power law curves on merger distributions for the mergers which has different metallicity (Z)



Fitting : Alpha

And also do the same for different alpha (efficiency of common envelope)



Some parameters which obtained from fit for different Z and Alpha:

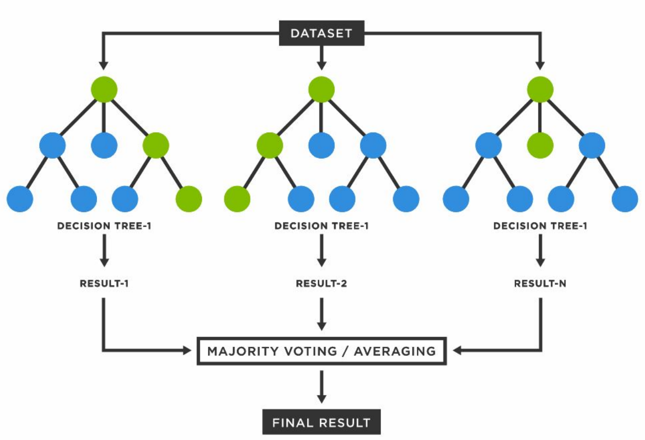
- $z = 0.0001$:
  - slope=-1.147
  - intercept=-0.197
- $z = 0.03$:
  - slope=-0.985
  - intercept=-2.22
- $alpha = 0.5$:
  - slope=-1.158
  - intercept=0.066
- $alpha = 5.0$:
  - slope=-1.164
  - intercept=0.066

The function is :

$$f(x) = x^a 10^b \tag{8}$$

Random Forest

In the last part $\rightarrow$ Training a **random forest** to predict the delay time given some features



Tools and libraries:

- **RandomForestRegressor** from **Sklearn**
- **XGBRFRegressor** from **Xgboost**
- **GridSearchCV** and **RandomizedSearchCV** from **Sklearn**
- **Tensorflow Decision Forests** from **TensorFlow**

Università degli Studi di Padova

Loading and pre-processing the data:
We keep the essential features and drop the other ones. The resulting schema looks like the following:

	Mass_0	Mass_1	Semimajor	Eccentricity	Z	alpha	Delay_Time
0	18.34303	21.09631	56.46466	0.017191	0.0004	0.5	9.993410e+04
1	50.99943	51.78028	118460.30000	0.415099	0.0004	0.5	5.520443e+16
2	33.98611	30.83786	135.10660	0.098669	0.0004	0.5	7.108337e+05
3	42.61480	33.31328	305.57600	0.003112	0.0004	0.5	1.214966e+07
4	49.74939	45.91471	1759.20200	0.159172	0.0004	0.5	5.995947e+09

- Normalizing the input/output based on different functions:

Input	$F(x)$	**Output**	$F(x)$
Standardization	$\frac{x - \overline{x}}{\sigma}$	**Log.**	$\log x$

- Splitting the data into train/test sets: 80% training, 20% test

Starting with **Scikit-learn** Random Forest Regressor

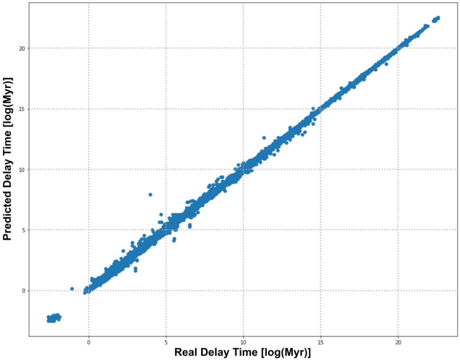
Parameters of the Model → **default parameters**

- Training the model on the training set
- Evaluating the model on the test set
- **No** hyper-parameter tuning or GridSearch for this part
- **Training Score**: 0.999
- **Test Score**: 0.999
- **Test Mean squared error**: 0.001
- **Test Mean squared log error**: 0.004

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': -1,
 'oob_score': False,
 'random_state': None,
 'verbose': 1,
 'warm_start': False}
```

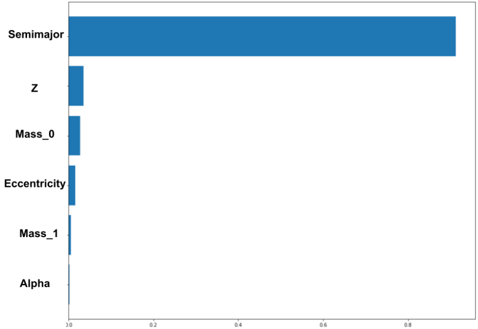
default parameters of Scikit-learn random forest regressor

Plotting the predicted delay times vs the real ones leads to:



predicted delay times vs the real delay times using Scikit random forest regressor with default parameters

Feature importance refers to the methods trying to find a score for each feature representing the relative importance of that feature. higher score $\rightarrow$ larger effect on the outcome



The feature importance of random forest model using Scikit library with default parameters

XGBoost Random Forests

This time we use XGBRFRegressor from XGBRF library

We perform **hyper-parameter tuning**

First using **RandomizedSearchCV** → to specify the range of suitable parameters

- RandomizedSearchCV:

```
random_grid = {
    'n_estimators': [100, 200, 300, 500, 1000],
    'max_depth': [None, 3, 5, 7, 10],
    'min_child_weight': [None, 1, 3, 5],
    'subsample': [None, .1, .5, .7],
    'colsample_bytree': [None, .1, .5, .7],
    'grow_policy': ['depthwise', 'lossguide'],
    'booster':['gbtree', 'gblinear', 'dart']}
```

```
rf = XGBRFRegressor()
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
                               n_iter = 50, cv = 3, verbose=10,
                               random_state=42, n_jobs = -1)
```

Then using **GridSearchCV** to find the best parameters in a specific range

- GridSearchCV:

```
param_grid = {
    'n_estimators': [450, 500, 550],
    'max_depth': [9, 10, 11],
    'min_child_weight': [2, 3, 4],
    'subsample': [.6, .7, .8],
    'colsample_bytree': [None],
    'grow_policy': ['lossguide'],
    'booster':['gbtree']}
```

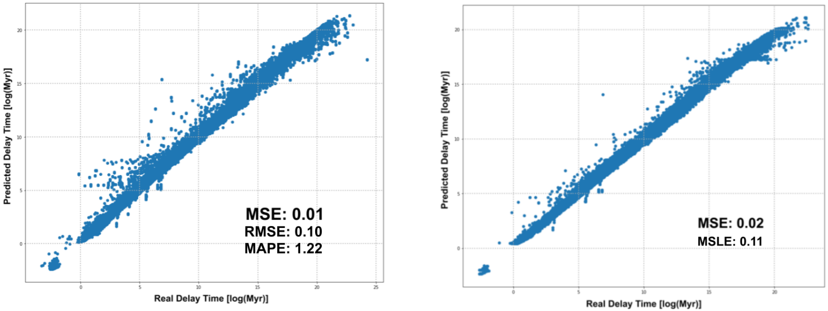
```
rf = XGBRFRegressor(n_jobs = -1, verbosity = 1)
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 10)
```

- **Best parameters found:**

```
subsample=0.7, n_estimators=500, min_child_weight=3, max_depth=10
```

Random Forest

The prediction results using **XGBoost** (best parameters) and **Tensorflow**(default parameters) are as following:

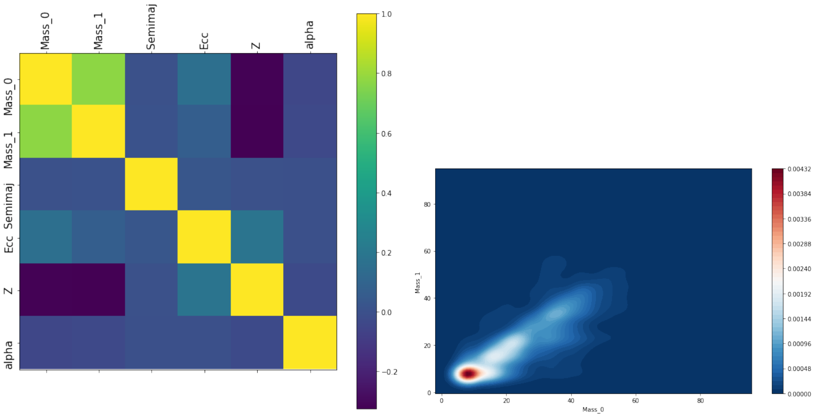


The results of Tensorflow with default parameters (left) vs XGBoost with best parameters (right)

Conclusion

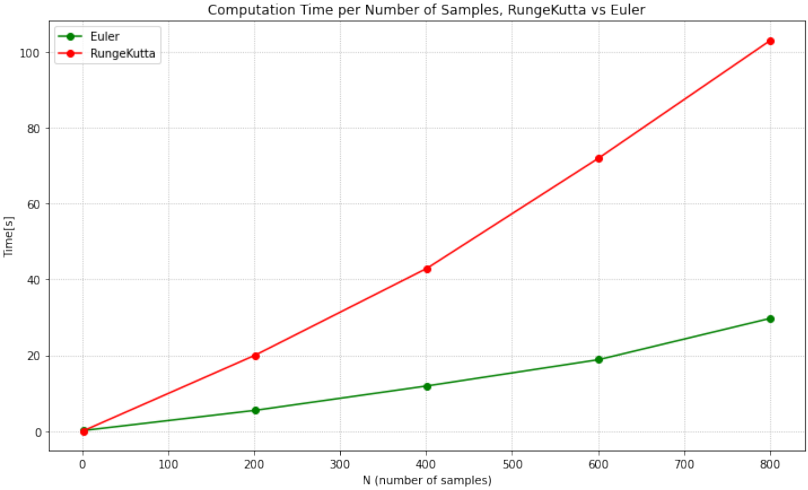
- Calculation of the delay times by **integrating** over system of ODEs using **Euler** and **fourth order Runge-Kutta** schemas and **adaptive time-step**
- **Fitting curve** to the **distribution** of the delay times for different values of **metallicity** and **efficiency of the common envelope**
- Training a **random forest** using **Sklearn**, **Xgboost** and **Tensorflow** to predict the delay times

Fin



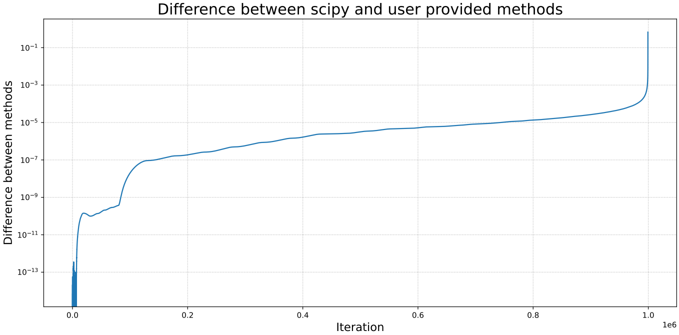
Correlation matrix for the features (1:Mass_1, 2:Mass_2, 3:Semimajor, 4:Eccentricity, 5:Metallicity, 6:Alpha) and correlation between the masses

Computation time



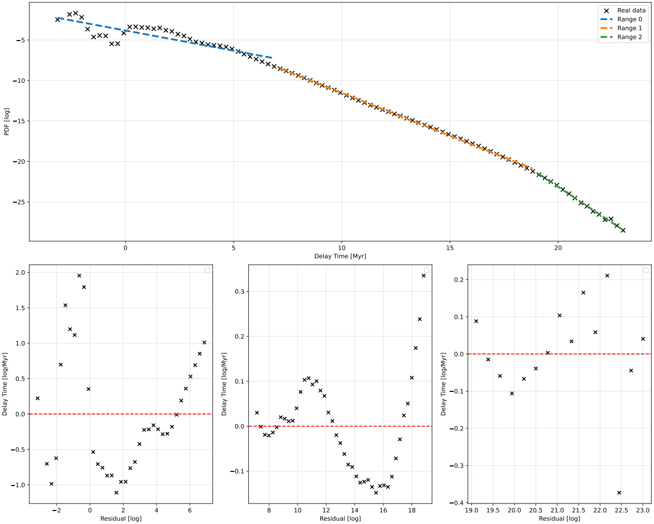
Computation time with adaptive time step for Euler and RK4 methods

Integration: Scipy vs User Provided



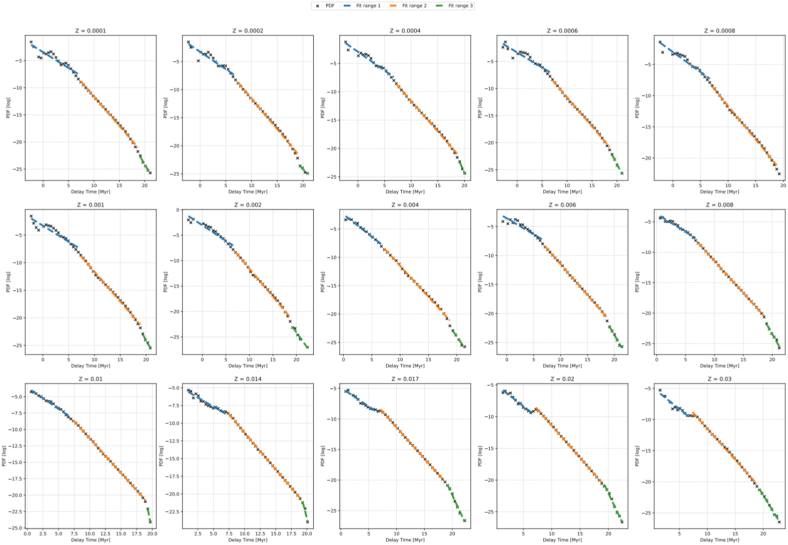
Difference in Semimajor-Axis value using lsoda (though scipy) and the user provided method

Fits without BWorldtime



Fits for the merger times

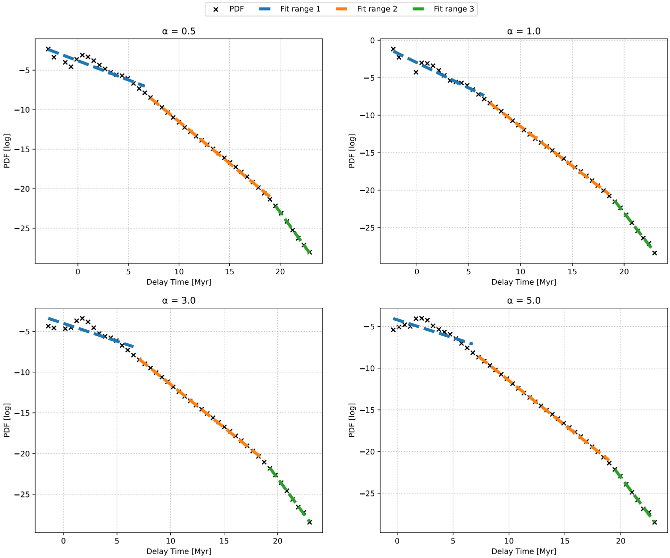
Università degli Studi di Padova



Fits for the merger times for different values of metallicity

Fits without BWorldtime

Università degli Studi di Padova



Fits for the merger times for different values of efficiency of the common envelope