# Newton’s Law of Universal Gravitation

# Simulating N-body Problems

**Deep Learning Project**

Physics Background

Numerical Methods

GUI

Model Training

**AGH University of Science and Technology**

**Faculty of Physics and Applied Computer Science**

**September 2023**

**Adam Młyńczak**

**Kacper Duda**

## Abstract

Astronomers for centuries have been wondering about mechanisms of planets' movement. At some point Newton discovered that all bodies attract themselves with a force of a given value:

$$F_g = G\frac{m_1 m_2}{r^2}\text{[i]}. \qquad (1)$$

He has also discovered the laws of classical mechanics allowing to predict future positions of bodies and interactions between them.

But can a deep learning model also see a connection between planets' masses and distance corelation with force? We would like to explore even further – can a DLM predict next positions of a given bodies with only initial positions and their masses as parameters?

Machine Learning is an amazing tool for sciences that has already made great discoveries. We have been inspired with such successes as reducing over 100k differential equations to just 4 (it's pronounced "four"!)[ii] or tracking particles in hadron collider[iii].

## 1. Physics Background

### 1.1. Physical Models Simplifications

We know that the laws of classical mechanics and the law of universal gravitation are not correct from the perspective of modern physics[iv] but we also know that it is still accurate because planets don't move with high – compared to light's speed – velocities and their mass is very big. This means that quantum and relativistic effects are not significant here. Because of that we will use classical mechanics and classical gravity force for calculations.

### 1.2. Calculating Force Value and Force Vector

As mentioned in equation (1), to compute force we need only gravitational constant, masses of bodies and their distance.

To calculate the components of the gravity force vector, trigonometric functions should be used as follow:

$$\begin{aligned} F_x &= F \cdot \cos\varphi, \\ F_y &= F \cdot \sin\varphi, \end{aligned} \qquad (2)$$

where following symbols mean:

- $F$ – value of force
- $F_x, F_y$ – x-axis and y-axis force components
- $\varphi$ – angle between $\vec{F}$ and $\overrightarrow{F_x}$ forces

Following figure describes the situation and relation between the force and its compounds:

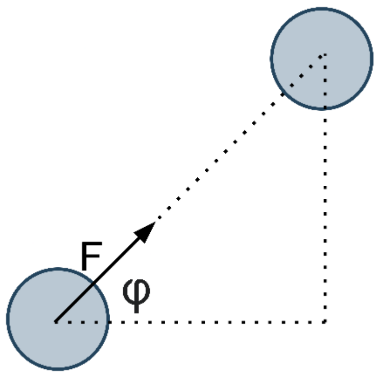

*Figure 1 Position of bodies and analysis of gravitational force applied to one of them.*

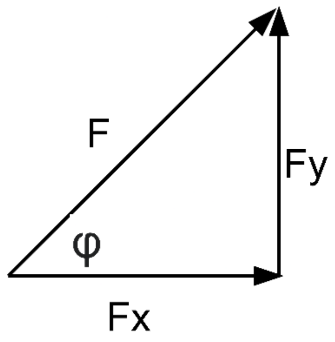

*Figure 2 Components of the force and relation between them.*

Trigonometric functions can be replaced with following formulas:

$$\begin{aligned} F_x &= F \cdot \frac{\Delta x}{r}, \\ F_y &= F \cdot \frac{\Delta y}{r}, \end{aligned} \qquad (3)$$

where $\Delta x$ and $\Delta y$ are x- and y-coordinates distance vectors from body that is attracted to body that attracts. Using equations (3) we can predict future movement of bodies.

## 1.3. Theoretical considerations of classical mechanics

Classical mechanics provide us functions of time describing position, velocity and acceleration:

$$\begin{gathered}\vec{r} = \vec{r}(t), \\ \vec{v} = \frac{d}{dt}\vec{r}(t) \\ \vec{a} = \frac{d}{dt}\vec{v}(t)\end{gathered} \tag{4}$$

And any given time acceleration depends on resultant force and mass of a given body:

$$\vec{a} = \frac{\overrightarrow{F_w}}{m}, \tag{5}$$

where $m$ is a mass and $\overrightarrow{F_w}$ is resultant force calculated with a following formula:

$$\overrightarrow{F_w} = \sum_{i=1}^{n} \overrightarrow{F_\iota}, \tag{6}$$

where $\overrightarrow{F_\iota}$ is gravitational force from another plant, and n is number of planets.

Knowing that we want to predict position vector for a given time period ($t_0 \leq t \leq t_k$) it is possible to reverse equations:

$$\begin{gathered}\vec{r}(t) = \overrightarrow{r_0} + \int_{t_0}^{t_k} \vec{v}(t)dt, \\ \vec{v}(t) = \overrightarrow{v_0} + \int_{t_0}^{t_k} \vec{a}(t)dt\end{gathered} \tag{7}$$

where $\overrightarrow{r_0}$ and $\overrightarrow{v_0}$ are initial position and velocity vectors.

Introducing new variable $\Delta t = t_k - t_0$ and simplifying equations we have:

$$\vec{r}(t) = \overrightarrow{r_0} + \overrightarrow{v_0} \cdot \Delta t + \int_{t_0}^{t_k} \int_{t_0}^{t_k} \vec{a}(t)dt\,dt. \tag{8}$$

Unfortunately, we do not have an analytical formula for acceleration at any given moment in time, as acceleration depends on the positions of other bodies. In order to solve the problem, numerical simplifications will be used to generate following positions of bodies.

## 1.4. Defining a proper gravitational constant

Defining new physics! (Why not?)

The order of magnitude of numbers used in calculating the movement of real planets is too small or big for large neural networks.

We cannot use numbers comparable to real G, real masses and real radiuses. All masses, distances must be scaled to numbers close to one. That means that it is important to accurately adjust G constant. Too small constant will make the model static and to big will make it almost impossible to observe in "human time".

We have picked a situation that made possible calculating such constant.

Two planets with mass $m = m_1 = m_2 = 1kg$ (units will not be included in further calculations because we will be using only SI units) are orbiting each other with constant speed and full period of $T = 5s$. Distance between them is $d = 2m$ so the radius of the circular trajectory is $r = 1m$.

Simple comparison of force of gravity and centripetal force allows us to calculate the G constant.

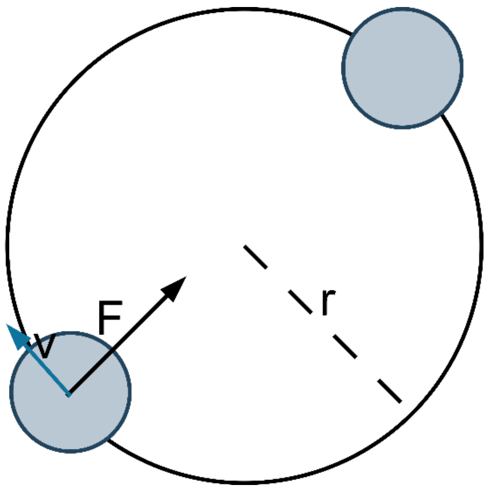

*Figure 3 Two planets orbit each other.*

Force of gravity is equal to centripetal force.

$$F_g = F_d$$

$$G\frac{m \cdot m}{d^2} = \frac{mv^2}{r}$$

$$(v = r\omega)$$

$$G\frac{m}{d^2} = r\omega^2 \quad (9)$$

$$\left(\omega = \frac{2\pi}{T}\right)$$

$$G = \frac{4\pi^2 r d^2}{mT^2}$$

$$G \approx 6.3165 \ldots \frac{Nm^2}{kg^2}$$

A constant of such value should ensure sufficient dynamics while maintaining relatively low velocities.

### 1.5. Additional Rules for Generating Examples

DLMs require lots of examples that are going to be generated by program written by us program.

We also want to maintain relatively close to 0 x and y coordinates in any given time or at least for first few seconds of simulation – it will be explained later.

To make that we have decided to generate models with planets that meet two following assumptions:

- initial center of mass is at origin,
- initial momentum is equal to 0.

Second point makes sure that center of mass stays at the origin.

Steps that are made to fulfil those two points are explained in numerical part.

## 2. DLM Training Data Format

### 2.1. Required Amount of Information to Predict the Future

Equation (8) tells us that to predict next positions we must have initial position, initial velocity, time difference and acceleration.

To know acceleration, we must know force. To know force, we must know masses of objects and their relative distances.

### 2.2. Limitations of Discrete Input

All previous equations assume that time flows continuously. There is still debate about possibility that the universe can be quantized (both time and space) into smaller discrete pieces. But that is not our problem.

DLM must be provided with a 1D array of data – each element symbolizes one neuron. The same is with output. We must divide initial state of planet model into the array and predict next positions and shape it into this format too.

### 2.3. Format of Data

Velocity can be replaced by the difference quotient. That is why we must just to provide two initial positions that are separated with a $\Delta t$ time. Next positions are also separated with that fixed amount of time. It means that input data format and output data format is similar.

Our proposal of input and output shape:

$$\text{X} = [\Delta t, m_1, m_2, \ldots, m_n, f_0, f_1], \quad \text{Y} = [f_2, f_3, f_4, f_5, \ldots f_{of-1}] \quad (10)$$

where X – input array, Y – output array, $m_i$ – mass of a planet, $f_i$ – frame, of – number of output frames.

Frames have given format (they are a part of 1D array, there are no arrays in arrays).

$$f_i = x_{1i}, y_{1i}, x_{2i}, y_{2i}, \ldots, x_{ni}, y_{ni} \quad (11)$$

They serve role of a “photo” of a state of planets.

According to equation (8) that should be enough data to correctly foreseen close future.

We have decided that it is better to predict more than one frame ahead, because model can adjust itself by seeing connections between successive positions of bodies. This is our assumption.

## 3. Numerical Methods

Generating learning data for DLM.

### 3.1. Predicting Next Positions

All planets represented in a model are objects containing their own positions (present and in all previous frames), velocity and mass.

It is impossible to obtain acceleration as a general function of time. That is why equation (8) has do be transformed. We split all time to simulate into very small deltas (computational delta) which value is about 0.00001s – there are 100k frames per second. We assume that acceleration can be approximated to be constant between two consecutive frames (changes too slow to be accurate).

So, at the beginning of each frame there is computed acceleration of each planet. Using that we calculate their next positions and next velocities. The process repeats until all frames are generated.

When assumed that acceleration is constant through a small period, we can compute next position and velocity of each planet with a given equations:

$$\begin{gathered}\overrightarrow{r_{i+1}} = \vec{r_i} + \vec{v_i} \cdot \Delta t + \frac{1}{2}\vec{a_i} \cdot \Delta t^2 \\ \overrightarrow{v_{i+1}} = \vec{v_i} + \vec{a_i} \cdot \Delta t\end{gathered} \tag{12}$$

We cannot use matrix methods to solve those equations because there are many planets that influence each other's movement.

Main loop works is following:

1. calculate resultant force and acceleration,
   a. for each connection (not for each planet) there is calculated value of force,
   b. for one planet there are calculated $F_x$ and $F_y$ compounds,
   c. the other planet in connection receives force compounds with opposite signs,
   d. the results are added to resultant compounds,
   e. after each connection is processed, the resultant force is divided by the planet's mass and the acceleration is calculated,
2. for each planet applying equation (12) to get next coordinates and velocities,
3. repeat until all frames are generated.

### 3.2. Generating Initial Positions

As mentioned, plants initial positions and velocities are not completely random – they are generated in a way that positions values are between [-2.5 and 2.5].

Masses are generated using bell curve:

m – N (0.5, 0.2).

But for positions and velocities the algorithm is a bit different.

For all <u>except one</u> positions and velocities are generated the same way as mass:

$$\begin{gathered}x,y - \mathrm{N}\ (0, 0.5), \\ v_x,v_y - \mathrm{N}\ (0, 0.2).\end{gathered} \tag{13}$$

But the last planet's kinetic values are calculated in a way:

$$z_n = -\frac{\sum_{i=1}^{n-1} z_i \cdot m_i}{m_n} \tag{14}$$

where $z$ is all of values presented in (13) – all coordinates.

This makes sure that two of points mentioned in 1.5. are fulfilled.

At the end there is a check whether any of $z$ calculated is close to 0. We cannot accept that last (even small) planet's coordinates are (100,100) because all planets were in bottom right corner.

If any of $z$ is $|z| \geq 1$, process of choosing planets starts again.

**3.3. Shaping And Transport Data Between Programs**

Each planet object after generation contains all their positions. All that data is converted to format used for machine learning.

Machine learning data format contains many iterations – different simulations with a given number of planets and frames predicted.

Machine learning data format can be converted to GUI format.

All data are transferred between programs via JSON format.

**4. Presenting Visual Data**

After generating data and converting it into „GUI format" we can visualize each iteration on screen. The idea of program is simple – the data is loaded, properly stored and then displayed. As it was said earlier, code accepts files with JSON extension and then groups data into right places – number of iterations, number of objects in each iteration, mass of each planet and positions in each frame which will be displayed. It creates vector containers for all this data. When all the needed information is in the right place, program starts displaying. We used SFML library in C++ to get this done. The changing of frames in iterations is done using *sf::Clock* class and delta parameter passed in entry file. Then the program checks if left arrow button or right arrow button is pressed – using this two we can navigate between iterations. Information with short instruction about it shows in the top left corner. Then the most important aim of this application is done – displaying the objects. First, for visual effects the small dots are shown – „stars". Then the program creates our planets – knowing where the data is stored, it gets necessary information and write information to already created *sf::Circle* class objects – radius, colour and position. The radius is connected to mass of planet. When all information is assigned to each object, window draws each of them. Next to all of planet's program shows mass of object. In the right top corner, we can see number of iterations, and in top left, as it was already said, instruction on how to operate between iterations. After all this things, one of three things can happen:

- If left arrow is clicked, program starts displaying previous iteration.
- If, right arrow is clicked, program starts displaying next iteration.
- If nothing is pressed, the next frame in iteration is displayed (in this case for each object the new position is assigned).
- At any point we can just close window and stop application.

**5. Model**

**5.1. Setup**

There were many challenges on the way to successfully running TensorFlow with GPU support. This included installing specific drivers, updating CUDA using files from NVIDIA's website, and installing TensorRT. Ultimately, we received a message stating that there is one available GPU for training the model.

**5.2. Training**

After many attempts to choose the best configuration, we have finally decided to use configuration that is in source code in file: *model/main.py*

**5.3. Results**

Model became overfitted. It has successfully found connections between input and output in training set. Generalisation has not been reached.

**6. Analysis**

It is still very impressive that our model could have predicted correctly positions of training set. There were three planets with their masses and two positions. The model has found a way to predict future states! It got 16 inputs and have predicted almost 500 outputs!

It is still interesting that model could partially correctly predict future states in validation dataset – in particular the closest future states. It is not unusual that model can predict the closest future states more accurately than those further away.

Model has been trained with frames that have separated with time $\Delta t = 0.025s = 25ms$. But frames have been generated using computational delta (there were in-between frames separated with $\Delta t = 0.00001$s, 80fps AI vs. 100000fps with numerical methods). That means that the model could not know exactly numerical models. And we know that computational delta changed to make 10000fps makes different results than the one used to predict. That means that probably a great part of information was lost.

But model understands relative positions and velocities of bodies. It can predict – somehow – movement of bodies close to each other. It understands that two planets close to each other that are many times bigger orbit each other while they usually catapult the smaller one away. (Fun fact: Jupiter and Saturn are accused of having launched Neptun and Uranus very far away from sun, so it is not that uncommon.) That is very realistic!

## 7. Conclusion

We did not expect that model can (even overfitted) predict movement of bodies that accurate. It may not be great for general examples because gravitational models are very sensitive to initial values.

Model can achieve some accuracy but there are still many topics to be explored before being able to solve this problem using DLMs. There is not enough information to accurately predict future by the model.

[i] Newton's law of universal gravitation
[ii] Artificial intelligence reduces a 100,000-equation quantum physics problem to only four equations
[iii] Performance of a geometric deep learning pipeline for HL-LHC
[iv] Modern physics