ON SOCIAL SIMULATION IN 4D RELATIVISTIC SPACETIME

A PREPRINT

Lai Kwun Hang
Centre for Science and Technology Studies (CWTS)
Leiden University
k.h.lai@cwts.leidenuniv.nl

February 15, 2022

ABSTRACT

Agent-based models (ABM) and simulations are becoming more prominent in social science. A major advantage of ABM is that it allows social scientists to explore hypothetical scenarios. If we stretch our imagination, one of the interesting social science scenarios would be our interstellar future. However, modeling an interstellar society requires relativistic physics, which is not straightforward to implement in existing ABM frameworks. In this paper, we present the mathematics and algorithmic details needed for simulating ABM in 4D relativistic spacetime. These algorithms form the basis of our open-source computational framework, "Relativitization" [1].

Keywords Special relativity · Agent-based model · Software framework

1 Introduction

An agent-based model (ABM) is a simulation model bridging microscopic behaviours of agents and macroscopic observations. Depending on the context of the model, an agent can be an individual, an organization, or as big as a country. Assumption about the social behaviours of the agents are made, then the agents are placed and evolved in a computational environment. Modeling computational agents enforces social scientists to make explicit assumptions, which helps formalizing the discussion on mechanisms and phenomena in the society.

In some research projects, if data collection is viable, ABM can be used to explain and predict the data. In many other cases, it is not practical to expect model calibration with data, where ABM can still be used for illustration and theoretical exposition in these cases [2]. In this paper, we are interested in modeling interstellar society, and it certainly belongs to the latter category.

The scientific and technological advancement in the last century greatly increases our understanding of the universe. Nowadays, we are able to build giant telescope and observe astronomical objects billions of light years away. Apart from deeping our scientific understanding, the astronomical knowledge also stimulate our imagination of interstellar civilizations. A lot of great science fictions have been written, and scientists proposed ideas like Fermi paradox [3], Dyson sphere [4] and Kardashev scale [5]. While many of the ideas are physically plausible, it would be interesting to discuss about these ideas in the context of social science. Due to the highly hypothetical nature of the problem, we suggested that ABM can be used to generate formal academic discussion on interstellar society.

To model agents in an interstellar space, supposed we only consider a scale with normal stellar objects where we can ignore the effects of general relativity, such as universe expansion and black holes, we still have to consider the effect of special relativity. In the context of ABM, where we the simulation is computed under a single inertial frame, we can simplify the theory of relativity into two core phenomena: speed of light as the upper bound of the speed of information travel, and time dilation relative to any stationary observer in the inertial frames. This also implies that we have to take care of four dimensions: three space dimension, plus one time dimensions.

Typically, an ABM is constructed on top of an ABM framework to facilitate model development and communication. There are a lot existing ABM framework, to name a few, NetLogo [6], mesa [7], and Agents.jl [8], see [9] for a detailed review. While it is possible to build a 4D relativistic model in some existing ABM frameworks, it is not easy to

enforce the relativistic effect, and it can be error-prone. Therefore, we have developed a simulation framework we call "Relativitization" [1], to help social scientists to build their ABM in relativistic spacetime. In this paper, the mathematics and the algorithms under the framework will be presented.

2 Definitions

In Relativitization, an agent is called a "player". Players live in a universe, and all the computation is done according to an inertial frame of the universe. The spatial coordinates of a player are represented by floating-point numbers x, y and z where the time coordinate of the player is represented by a floating-point number t. To simplify computation and visualization, the universe is partitioned into unit cubes. A player with floating-point coordinates (t, x, y, z) is located at the cube with integer coordinates $T = \lfloor t \rfloor$, $X = \lfloor x \rfloor$, $Y = \lfloor y \rfloor$, $Z = \lfloor z \rfloor$, note that the computations of a simulation are done at unit time steps and we can actually assume $T = t$. Denote the speed of light as c. In vector notation, define $\mathbf{s} = (t, \overrightarrow{u}) = (t, x, y, z)$, and $\mathbf{S} = (T, \overrightarrow{U}) = (T, X, Y, Z)$.

2.1 Interval and time delay

The spacetime interval between coordinates $\mathbf{s}_i$ and $\mathbf{s}_j$ is

$$\|\mathbf{s}_i - \mathbf{s}_j\| = c^2(t_i - t_j)^2 - (x_i - x_j)^2 - (y_i - y_j)^2 - (z_i - z_j)^2. \tag{1}$$

If $\|\mathbf{s}_i - \mathbf{s}_j\| < 0$, it is called a spacelike interval, and events happen at the two coordinates are not causally connected because no information can travel faster than the speed of light c.

It is often needed to compute the interval in integer coordinates. We define the spatial distance between $\overrightarrow{U_i}$ and $\overrightarrow{U_j}$ as the maximum distance between all points in the cubes at $\overrightarrow{U_i}$ and $\overrightarrow{U_j}$

$$|\overrightarrow{U_i} - \overrightarrow{U_j}| = (X_i - X_j + 1)^2 + (Y_i - Y_j + 1)^2 + (Z_i - Z_j + 1)^2. \tag{2}$$

Suppose there is a signal sent from $\overrightarrow{U_i}$ to $\overrightarrow{U_j}$, to ensure that the information travels slower than the speed of light, the integer time delay $td(\overrightarrow{U_i}, \overrightarrow{U_j})$ is computed as

$$td(\overrightarrow{U_i}, \overrightarrow{U_j}) = \left\lceil \frac{|\overrightarrow{U_i} - \overrightarrow{U_j}|}{c} \right\rceil. \tag{3}$$

2.2 Group id

From Eq.2, even if $\overrightarrow{U_i} = \overrightarrow{U_j}$, the time delay is non-zero. However, when two players are really closed to each other, the time delay between the players should be zero.

To allow zero time delay, we define a group id $g(\overrightarrow{u}, \overrightarrow{U})$ of a player as

$$\begin{aligned} n_e &= \left\lceil \frac{1}{d_e} \right\rceil, \\ n_x &= \left\lfloor \frac{x - X}{d_e} \right\rfloor, \\ n_y &= \left\lfloor \frac{y - Y}{d_e} \right\rfloor, \\ n_z &= \left\lfloor \frac{z - Z}{d_e} \right\rfloor, \end{aligned}$$

$$g(\overrightarrow{u}, \overrightarrow{U}) = n_x n_e^2 + n_y n_e + n_z, \tag{4}$$

where d_e is the edge length of a smaller cube inside the unit cube.

If two players have same integer coordinates and same group id, they are both located in a smaller cube defined by d_e, and we say the players belong the same group and the time delays between the players are zero.

2.3 Player data

A player is characterized by a set of data:

- player id i,
- integer coordinates (T_i, X_i, Y_i, Z_i),
- a historical record of integer coordinates $H_i = \{(T_i', X_i', Y_i', Z_i'), T_i' < T_i\}$,
- floating-point coordinates (t_i, x_i, y_i, z_i),
- dilated time residue r_i, a floating point variable to store the effect of time dilation,
- dilation action b_{ri} is a boolean variable to determine the impact of time dilation,
- group id g_i,
- floating-point velocities $\overrightarrow{v_i} = (v_{ix}, v_{iy}, v_{iz})$,
- other data D_i relevant to the model.

2.4 Command

Interaction in ABM is often presented as one player asking another player to do something. Because the speed of information travel is bounded by c, a player cannot simply ask other players to do something immediately. Instead, interactions are mediated by commands. Whenever player i wants to interact with player j, player i send a command to j.

A command is characterized by:

- i_{to}, the id of the player to receive this command,
- i_{from} the id of the target player who sent this command,
- $\mathbf{s}_{\text{from}}$ the coordinates when the player sent this command,
- f_{self} a function to modify data of the sender when this is sent,
- f_{target} a function to modify data of the target player when this is received.

Commands travel at the speed of light c. The amount of time needed for a command to reach the target depends on the trajectory of the target player i_{to} and the sender coordinates $\mathbf{s}_{\text{from}}$,

2.5 Universe data

Universe is a simulation model which aggregates all necessary data and functionalities. An universe has:

- a current universe time T_{current},
- a 4-dimension array of player data list L_{TXYZ}, player data that belongs to (T, X, Y, Z) should be stored at the corresponding list,
- a map from player id to command M_c, storing lists of commands sending to players,
- other universe global data D_G relevant to the model,
- an AI function f_{AI}, given an environment observed by a player, the function determine the commands sent from the player at every turn,
- a regular mechanism function f_{rm}, to modify the player data and generate commands at every turn,
- a dilated mechanism function f_{dm}, to modify the player data and generate commands at turns that are not affected by time dilation,
- a global mechanism function f_{gm}, to modify the global data.

The AI function f_{AI} and the regular mechanism function f_{rm} have similar functionalities. In fact, it is possible to ignore either f_{AI} or f_{rm} in many simple models. The major difference between the two is that AI function is more restricted, i.e., it can only modify its own player data through commands, where the regular mechanism function can do whatever it wants to modify the player data. In a more complex model, it is clearer to separate the cognitive decision of a player (f_{AI}) and the passive process from mechanisms (f_{rm} and f_{dm}).

2.6 3D view

At any moment, a player can only observe a 3D slice of the universe instead of viewing the full 4D array L_{TXYZ}, Denote V_i as the 3D view of the universe from the viewpoint of player i, where the player's time is equal to the current universe time $T_i = T_{\text{current}}$. V_i is characterized by:

- a map from player id to player data, these are the player data in the universe that player i should see,
- the part of the universe global data D_G that should be exposed to players.

Computation of V_i can be decomposed into 2 stages. Firstly, if we ignore the fact that the time delay between players with the same group id and same integer coordinates is zero, all players in a cube share the same 3D view of the universe. Algorithm 1 compute the shared 3D view at the coordinates (T_i, X_i, Y_i, Z_i). Then, V_i can be computed from the shared 3D view by Algorithm 2. In fact, all players in the cube with the same group id has exactly the same 3D view.

Algorithm 1 Algorithm to compute a map of player id to player data viewed from the coordinates (T_i, X_i, Y_i, Z_i)

Input
  T_i, X_i, Y_i, Z_i position of the viewing location
  L_{TXYZ} 4D array of player data list
Output
  Map from player id to player data
Data
  M map from player id to player data
for all X_j, Y_j, Z_j **do**
  $T_j \leftarrow T_i - td(\overrightarrow{U_i}, (X_j, Y_j, Z_j))$
  for all data of player k in $L_{T_j X_j Y_j Z_j}$ **do**
    if M has key k **then**
      if T of $M[k] < T_k$ **then**
        Replace data of player k in $M[k]$
      end if
    else
      Store data of player k to $M[k]$
    end if
  end for
end for
return M

3 Simulation step

Supposed we have defined an universe model, a complete step in a simulation looks like:

1. update the global data (Sec. 3.1),
2. compute time dilation effects for all players (Sec. 3.2),
3. process mechanisms for each player (Sec. 3.3,
4. process the command map (Sec. 3.4),
5. compute and process AI inputs (Sec. 3.5),
6. move players, add afterimages, and update time (Sec. 3.6).

The simulation can be ran for a fixed amount of steps, or stop until a stopping condition is matched.

3.1 Update global data

The universe global data D_G can be separated into 4 parts:

- immutable, not observable by players: such as data saved directory,
- immutable, observable by players: such as the dimension of the universe,

Algorithm 2 Algorithm to compute view V_i of player i

Input
    i the id of the target player
    L_{TXYZ} 4D array of player data list
Output
    V_i 3D view of player i
Data
    M map from player id to player data
Compute M by Algorithm 1
for all player data in $L_{T_iX_iY_iZ_i}$ **do**
    if same group id as player i **then**
        if M has key k **then**
            if T of $M[k]$ < T_k **then**
                Replace data of player k in $M[k]$
            end if
        else
            Store data of player k to $M[k]$
        end if
    end if
end for
Construct V_i from M and the relevant part of D_G
return V_i

- mutable, not observable by players: such as statistics of the model,
- mutable, observable by players: such as variables in some of the mechanisms.

Immutable data does not need to be updated. Mutable data that is not observable by players can be updated without any restriction. The only part where we need to pay extra attention to is the mutable data that is observable by players. If f_{gm} depends on some player data to update the global data, and the effect is observable by players, we need to ensure that no information is transferred faster than the speed of light via the global data update.

For example, if f_{gm} updates a data if "all" player data satisfy a condition, we have to be careful about what do we mean by "all" here. In the universe, the maximum time delay equals $td_{\text{max}} = td((0,0,0),(max(X),max(Y),max(Z)))$. To fulfill the speed of light requirement, f_{gm} has to check whether all player data in L_{TXYZ}, where $T_{\text{current}} - td_{\text{max}} \leq T \leq T_{\text{current}}$, $0 \leq X \leq max(X)$, $0 \leq Y \leq max(Y)$, and $0 \leq Z \leq max(Z)$, satisfy the condition.

3.2 Compute time dilation

Relative to a stationary observer i in an inertial frame, special relativity predicts that a moving observer j experiences a time dilation effect:

$$\gamma_j = \frac{1}{\sqrt{1 - \frac{v_j^2}{c^2}}}, \tag{5}$$

$$\Delta t_j = \frac{1}{\gamma_j} \Delta t_i, \tag{6}$$

where γ_j is called the Lorentz factor.

Ideally, computation should be done in every local frame following all players, and the computation results can be synchronized by Lorentz transformation. However, this will make the framework and the model substantially more complex. Therefore, all computations in Relativitization are done in a single inertial frame. To account for the time-dilation effect, the dilated time residue r_i, and the dilation action variable b_{ri} are updated by Algorithm 3 every turn for every player. b_{ri} will then affect the mechanism processing in Sec. 3.3.

Algorithm 3 Algorithm to update time dilation variables

Input
    Player data of player i
Compute γ_i by Eq. 5
$r_i \leftarrow r_i + \frac{1}{\gamma_i}$
if $r_i \geq 1$ **then**
    $r_i \leftarrow r_i - 1$
    $b_{ri} \leftarrow$ true
else
    $b_{ri} \leftarrow$ false
end if

3.3 Process mechanisms

Mechanism is one of the major components of the universe model. Given the 3D view V_i of a player i, mechanisms describe how the data of player i should be updated, and which commands should be generated from the process.

Mechanisms are classified into regular mechanism and dilated mechanism, where dilated mechanism is affected by time dilation and regular mechanism is not. For example, if a player should synchronize an internal state based on the state of other player, then this should be a regular mechanism. If a player needs to produce an output where the amount is proportional to time, then this should be a dilated mechanism.

Since commands generated by mechanism should be executed immediately if the time delay between the target and the sender is zero, it is more convenient to consider all player data in a cube together in the mechanism processing algorithm, as demonstrated in algorithm. 4. The output commands from Algorithm 4 are then grouped by id of the target player, and added to M_c after randomizing the order of senders

Algorithm 4 Algorithm to update players in a cube by mechanisms

Input
    Coordinates (X', Y', Z')
Output
    A list of commands
Compute the shared 3D view by Algorithm 1
for all Group in (X', Y', Z') **do**
    Compute the 3D view for this group by Algorithm 2
    for all player i in group **do**
        if b_{ri} is true **then**
            Update player data by f_{rm} and f_{dm},
        else
            Update player data by f_{rm}
        end if
    end for
    for all generated command **do**
        if target player is in this group **then**
            Update target player by f_{target}
        end if
    end for
end for
return the rest of the generated commands

3.4 Process command map

The command map M_c is a map from player id to a list of commands that is sending to that player. At each turn, the distance between the target player and the sent positions of all commands in the list are calculated, the command is executed on the player if the spacetime interval is larger than zero. Algorithm 5 illustrates the process.

Algorithm 5 Algorithm to process command map

Input
    M_c the command map
for all key i in M_c **do**
    Get list of command $M_c[i]$
    for all command C in the list **do**
        if $\|\mathbf{s}_{\text{from}} - \mathbf{s}_i\| \geq 0$ **then**
            Execute the command
        end if
    end for
end for
Remove all executed commands

3.5 AI input

The AI function f_{AI} determines the commands to be sent given the 3D view of a player. Unlike mechanisms, a command needs to be self-executed on the sender first, then the command will be sent if the self-execution succeeded. Similar to Sec. 3.3, commands should be executed immediately if the time delay between the target and the sender is zero, it is more convenient to consider all player data in a cube in the algorithm, as demonstrated in Algorithm 6. The output commands from Algorithm 6 are then grouped by id of the target player, and added to M_c after randomizing the order of senders.

Algorithm 6 Algorithm to compute and process AI input

Input
    Coordinates (X', Y', Z')
Output
    A list of commands
Compute the shared 3D view by Algorithm 1
for all group in (X', Y', Z') **do**
    Compute the 3D view for this group by Algorithm 2
    for all player i in group **do**
        Compute commands by f_{AI}
        for all command from player i **do**
            Self-execute the command by f_{self}
            Keep the command if succeeded
            if the target of the command is self **then**
                Execute the command to self by f_{target}
            end if
        end for
    end for
    for all Command **do**
        if Target player is in this group **then**
            Update target player by f_{target}
        end if
    end for
end for
return the rest of the generated commands

3.6 Move players, add afterimages, and update time

Moving players and storing their data require additional considerations in this simulation framework, and here is the reason:

1. assume player i and player j are located in the same cube,
2. player j moves to the other cube,
3. the new information takes time to travel to player i, so player i cannot see the new position of player j,

4. player i cannot see the old information of player j either, because player j is no longer there,
5. player j disappears from the sight of player i.

This "disappearance" is caused by the problem of the integer-based coordinates used in the computation of player's 3D view.

Consider a more generic situation, supposed player i is located at $\overrightarrow{U_i}$, and player j moves from $\overrightarrow{U_j}$ to $\overrightarrow{U_k}$. Ignoring the possibility of zero time delay, the maximum time player i has to wait to see player j is bounded by the Eq. 7,

$$\begin{aligned}
\Delta T &= td(\overrightarrow{U_i}, \overrightarrow{U_j}) - td(\overrightarrow{U_i}, \overrightarrow{U_k}), \\
&= \left\lceil \frac{|\overrightarrow{U_i} - \overrightarrow{U_j}|}{c} \right\rceil - \left\lceil \frac{|\overrightarrow{U_i} - \overrightarrow{U_k}|}{c} \right\rceil, \\
&\leq \left\lceil \frac{|\overrightarrow{U_i} - \overrightarrow{U_j}|}{c} - \frac{|\overrightarrow{U_i} - \overrightarrow{U_k}|}{c} \right\rceil, \\
&\leq \left\lceil \frac{|\overrightarrow{U_j} - \overrightarrow{U_k}|}{c} \right\rceil,
\end{aligned}$$

$$\Delta T \leq td(\overrightarrow{U_j}, \overrightarrow{U_k}). \tag{7}$$

Therefore, if we include back the possibility where the time delay between player i and player j can be zero, the maximum duration of the disappearance produced by the movement is bounded by $\Delta T_{max} = td((0,0,0),(1,1,1))$. To prevent the unrealistic disappearance from happening, the old player data has to stay at the original position for at least ΔT_{max} turn, we call this the "afterimage" of the player. Note that afterimages only participate in the 3D view of players, they should not be updated by commands or mechanisms.

Algorithm 7 does multiple things: update the universe time, move players by their velocities, synchronize time of players, store old coordinates to the history of player, clean the history if the stored coordinates is too old, and add the afterimages to the latest data in the 4D data array. Since the universe time has been updated, this simulation step has finished. The universe should go to the next step and loop over all algorithms in Sec. 3 again.

Algorithm 7 Algorithm to move player, add afterimage, and update time

$T_{\text{current}} \leftarrow T_{\text{current}} + 1$
for all player i **do**
  $t_i \leftarrow T_{\text{current}}$
  $x_i \leftarrow x_i + v_{ix}$
  $y_i \leftarrow y_i + v_{iy}$
  $z_i \leftarrow z_i + v_{iz}$
  $T_i \leftarrow T_{\text{current}}$
  $X_i \leftarrow \lfloor x_i \rfloor$
  $Y_i \leftarrow \lfloor y_i \rfloor$
  $Z_i \leftarrow \lfloor z_i \rfloor$
  $g_i \leftarrow g(\overrightarrow{u_i}, \overrightarrow{U_i})$ by Eq. 4
  if coordinates or group is new **then**
    Save old coordinates to history H_i
  end if
  for all (T_i', X_i', Y_i', Z_i') in H_i **do**
    Remove if $T_{\text{current}} - T' > \Delta T_{max}$
  end for
  for all (T_i', X_i', Y_i', Z_i') in H_i **do**
    Add the old data to $L_{T_i X_i' Y_i' Z_i'}$
  end for
end for

4 Discussion

4.1 New player and dead player

Because creating new player and defining living status of players are not crucial to all ABM, they are not introduced in Sec. 3. In fact, Relativitization supports both creating new player and turning alive player to dead player.

To avoid violation of the speed of light limit, new player cannot be created freely anywhere. Instead, a command has to be sent by a player, and the command executes on self or another player to indicate in the player data that a new player should be created. This information is visited by an algorithm every turn and a new player will be created at the location of the player with this information.

To turn an alive player dead, there needs to be a command to execute on that to change the living status. The dead player will not be processed by any algorithms in Sec. 3, but it will still remain to be in the history stored in the 4D data array. Until no other player can see the player in their 3D view, the dead player will disappear from the 4D data array.

4.2 Time complexity, optimized 3D view

The time complexity of a model depends on the actual design of mechanisms and AI computation. If we assume that the extra computation costs are $O(1)$, then most of algorithms presented initial this paper are $O(n)$, where n is the number of players. The only exception is Algorithm 1, it has the time complexity $O(nm)$, where m is the spatial volume of the universe.

In practice, if we want to avoid $O(n)$ mechanisms and AI computation, the model often need to get a set of neighbouring cubes around a player to determine how the player should be updated. The map data structure of the 3D view introduced in Sec. 2.6 does not have a fast method to get the neighbouring cubes. Therefore, in the actual implementation of the 3D view of players in Relativitization, the original map is supplemented with a 3D player id array to allow fast spatial lookup, at the price of having a slower computation speed of 3D view.

4.3 Parallelization

The data flow in simulation framework is pretty straightforward, most of the computations only depend on the historical data in the 4D array. Therefore, as long as immutability of data is properly handled, many of the algorithms can be easily parallelized.

In Relativitization, the mechanisms algorithm in Sec. 3.3 and the AI input algorithm in Sec. 3.5 are computed in parallel for each cube, and the command map process algorithm in Sec. 3.4 is computed in parallel for each player.

4.4 Rest mass and photon rocket

There is an important aspect of physics we have yet to talk about in this paper - how should the velocity of player be changed? Of course, an ABM does not necessary be physically realistic, and the velocity can change to whatever value the modeler wants. Nevertheless, the major goal of this simulation framework is to assure that the model obeys relativistic physics, it makes sense to also be physically correct regarding the velocity.

A natural choice of a physically reasonable propulsion system is photon rocket [10]. In spite of the debates on various factors affecting the speed limit of a photon rocket [11, 12]. ideal photon rocket is a good starting point to introduce realism in how a player should move.

In order to formulate how a player is propelled by a photon rocket, we have to define a "rest mass" property in the extra player data D_i. Denote the initial rest mass of the player as m_i and the initial velocity as $\overrightarrow{v_i}$. After the photon rocket propulsion, denote the final rest mass of the player as m_f and the final velocity as $\overrightarrow{v_f}$. Sometimes it is convenient to express $\overrightarrow{v_f}$ as $v_f\hat{v}_f$ to separate the magnitude and the unit vector.

In special relativity, 4-momentum is used to represent the kinematic state of an object. There are 3 momenta involved in a photon rocket propulsion process:

- photon 4-momentum: P_{ph},
- player initial 4-momentum $P_i = (\gamma_i m_i c, \gamma_i m_i \overrightarrow{v_i})$, where γ_i is the initial Lorentz factor,
- player final 4-momentum $P_f = (\gamma_f m_f c, \gamma_f m_f \overrightarrow{v_f})$, where γ_f is the final Lorentz factor.

One of the useful scenarios is that the player turn some rest mass into photon, i.e., $m_i > m_f$, and given that the player will face towards a specific direction after the propulsion, we want to know the final velocity v_f. From conservation of

4-momentum:

$$
\begin{aligned}
&P_{ph} = P_i - P_f, \\
&P_{ph}^2 = P_i^2 + P_f^2 - 2P_i \cdot P_f, \\
&0 = (m_i c)^2 + (m_f c)^2 - 2\gamma_i \gamma_f m_i m_f (c^2 - v_f \overrightarrow{v_i} \cdot \hat{v}_f), \\
&(m_i c)^2 + (m_f c)^2 = \frac{2\gamma_i m_i m_f}{\sqrt{1 - v_f^2/c^2}} (c^2 - v_f \overrightarrow{v_i} \cdot \hat{v}_f), \\
&\left(\frac{(m_i c)^2 + (m_f c)^2}{2\gamma_i m_i m_f} \right)^2 = \frac{(c^2 - v_f \overrightarrow{v_i} \cdot \hat{v}_f)^2}{1 - v_f^2/c^2}, \\
&\left(\frac{(m_i c)^2 + (m_f c)^2}{2\gamma_i m_i m_f} \right)^2 (1 - v_f^2/c^2) = c^4 - 2c^2 v_f \overrightarrow{v_i} \cdot \hat{v}_f + v_f^2 (\overrightarrow{v_i} \cdot \hat{v}_f)^2.
\end{aligned}
$$

Eq. 8 is a quadratic equation, solving the equation gives us the solution of v_f:

$$
\begin{aligned}
&((\overrightarrow{v_i} \cdot \hat{v}_f)^2 + \left(\frac{(m_i)^2 + (m_f)^2}{2\gamma_i m_i m_f} \right)^2 c^2) v_f^2 + \\
&(-2c^2 \overrightarrow{v_i} \cdot \hat{v}_f) v_f + c^4 \left(1 - \left(\frac{(m_i)^2 + (m_f)^2}{2\gamma_i m_i m_f} \right)^2 \right) = 0.
\end{aligned} \tag{8}
$$

Another useful scenario is that given the final velocity $\overrightarrow{v_f}$, we want to know the rest mass needed to be converted into photon.

$$
\begin{aligned}
P_{ph} &= P_i - P_f \\
P_{ph}^2 &= P_i^2 + P_f^2 - 2P_i \cdot P_f \\
0 &= (m_i c)^2 + (m_f c)^2 - 2\gamma_i \gamma_f m_i m_f (c^2 - \overrightarrow{v_i} \cdot \overrightarrow{v_f})
\end{aligned}
$$

Eq. 9 is also a quadratic equation, solving this equation gives us the solution of m_f:

$$
c^2 m_f^2 + (-2\gamma_i \gamma_f m_i (c^2 - \overrightarrow{v_i} \cdot \overrightarrow{v_f})) m_f + (m_i c)^2 = 0. \tag{9}
$$

There are in-built utility functions in Relativitization to compute these solutions. Nevertheless, modelers are free to implementation other more realistic propulsion equations in their model.

5 Summary

In this paper, we have presented a set of algorithms to implement ABM simulation in a 4D, relativistic spacetime. Based on these algorithms, we have developed a simulation framework we call "Relativitization" [1]. Our framework will lower the barrier of entry and encourage social scientists to apply their expertise to explore the interstellar future of human civilization. We hope our framework can be used to initiate meaningful and academically interesting discussions about our future.

References

[1] Kwun Hang Lai. *Relativitization*. https://github.com/Adriankhl/relativitization. 2022.

[2] Bruce Edmonds and Ruth Meyer. *Simulating social complexity*. Springer, 2015.

[3] Robert H Gray. "The Fermi paradox is neither Fermi's nor a paradox". In: *Astrobiology* 15.3 (2015), pp. 195–199.

[4] Jason T Wright. "Dyson spheres". In: *arXiv preprint arXiv:2006.16734* (2020).

[5] Robert H Gray. "The Extended Kardashev Scale". In: *The Astronomical Journal* 159.5 (2020), p. 228.

[6] U. Wilensky. *NetLogo*. http://ccl.northwestern.edu/netlogo/. 1999.

[7] Jackie Kazil, David Masad, and Andrew Crooks. "Utilizing Python for Agent-Based Modeling: The Mesa Framework". In: *Social, Cultural, and Behavioral Modeling*. Ed. by Robert Thomson et al. Cham: Springer International Publishing, 2020, pp. 308–317. ISBN: 978-3-030-61255-9.

[8] George Datseris, Ali R. Vahdati, and Timothy C. DuBois. "Agents.jl: a performant and feature-full agent-based modeling software of minimal code complexity". In: *SIMULATION* 0.0 (2021), p. 00375497211068820. DOI: 10.1177/00375497211068820. eprint: https://doi.org/10.1177/00375497211068820. URL: https://doi.org/10.1177/00375497211068820.

[9] Constantin-Valentin Pal et al. "A review of platforms for the development of agent systems". In: *arXiv preprint arXiv:2007.08961* (2020).

[10] JR Pierce. "Relativity and space travel". In: *Proceedings of the IRE* 47.6 (1959), pp. 1053–1061.

[11] Espen Gaarder Haug. "The ultimate limits of the relativistic rocket equation. The Planck photon rocket". In: *Acta Astronautica* 136 (2017), pp. 144–147.

[12] Daniele Tommasini, Angel Paredes, and Humberto Michinel. "Comment on "the ultimate limits of the relativistic rocket equation. The Planck photon rocket"". In: *Acta Astronautica* 161 (2019), pp. 373–374.