Master thesis

Numerical Approximation of the replicator equations for the Nash bargaining game

Author:
Panagiotis Sarridis

Supervisors:
Dr. Paul Zegeling
Dr. Thijs Ruijgrok

Utrecht University

Mathematical Sciences

February 12, 2011

Abstract

There is a variety of phenomena that take place around us. A good mathematical tool to analyse and understand the behaviour of these phenomena, is to approximate them numerically. That is, first we model the phenomenon itself, then we computationally reproduce it, so that we can imitate the phenomenon as many times and in any way we need. That gives us the opportunity, to check the behaviour of the model for different parameters in order to figure out in which way, the phenomenon is affected by them.

The phenomenon that is numerically approximated in this work, comes from the field of game theory and it is known as the Nash bargaining game. Its mathematical model, consists of two time dependent partial integrodifferential equations that interact with each other, also known as the replicator equations for the Nash bargaining game.

Both fixed grid method and adaptive moving grid method, have been used to numerically approximate the model. The central difference scheme, has been used to numerically approximate the $2nd$ order space derivatives, while the approximation of all the integrals, has been based on the trapezoidal rule for both fixed and adaptive moving grid methods. For the approximation of the $1st$ order time derivative, Euler forward numerical scheme has been used for the fixed grid method and Euler backward scheme for adaptive moving grid.

The results that have been achieved during this work, show that the behaviour of the model, depends mostly on the parameters ϵ_1, ϵ_2. For the case where $\epsilon_1 = \epsilon_2$, the initial curves are moving towards to each other until they have the same shape and position, where they become stable and as ϵ_1, ϵ_2 get closer to zero, although the curves preserve their volumes, they become steeper and steeper. While for $\epsilon_1 \neq \epsilon_2$, we see that the curves converge to stationary solutions, that are no longer concentrated at the same position. In this case, the ratio $r = \frac{\epsilon_1}{\epsilon_2}$, seems to affect the behaviour of the model. Finally, it seems that, the initial values of the model, do not affect the result.

The algorithms that have been used are built on Matlab for the fixed grid method, while for the adaptive moving method fortran 77 was used. The computer that was used for the runs has processor Intel Core 2 Duo T7250 (2.0 GHz).

Contents

1 Introduction

1.1 The bargaining game:

The bargaining game was introduced by John Nash in 1950, see [5]. In this game there are two players who can divide a unit of a certain good, possibly money. Each player submits a *demand*, which is the fraction of the good they desire. If the sum of the demands equals one or less, each player gets the demanded fraction, otherwise both players get nothing.

The players derive a certain utility from the good they receive. The corresponding utility function is taken to be a increasing, concave function of the amount of the good. Such a function need not be linear. Imagine, for instance, that one of the parties is a manufacturer who needs 100 widgets from the other party for its own production process. Getting 200 widgets would be nice, but not twice as nice as getting 100 widgets, since the extra 100 widgets are not essential for the production process. Such a firm would be very reluctant to overask, since the gain of obtaining a lot of widgets is not balanced by the risk of obtaining none. A common interpretation of utility functions is therefore that they indicate the amount of *risk averseness* of a player. In bargaining theory, the division of the good is not the quantity of interest, but the resulting utilities.

The bargaining game can be seen from different view points.

- Economical aspect: Since [5] was published, the bargaining game immediately grabbed the attention of economists and a huge amount of literature has been published on the subject, see [4]. Consider of a bargaining situation between a labor union and the management of a firm, where the good to be divided is the yearly profit of the firm. We can imagine that both parties immediately put down their 'final offers'. If these are compatible, everybody is happy. If not, the negotiations have broken down and both players lose, for instance the unions decide to strike since the negotiations were sunk.
  Also, many economic transactions have a bargaining aspect. For instance, if I want to sell my house for at least 200K euro and a buyer wants to buy it for at most 300K, then in a sense we have 100K euro to divide.

- Philosophical aspect: The game also plays a role in philosophy, in particular in questions of ethics and morality. An impressive list of thinkers, from Plato, Aristotle, Kant, Hume to Rawls have discussed the question of what is good or bad in society. Is a just society one where everybody receives the same (egalitarianism) or one where the sum of the happiness of the members of the society is maximized (utilitarianism) or some other principle? Recently, Skyrms (1996) and Binmore (2005), in [7] and [2] have taken the bargaining game as a starting point to investigate these questions.

In his paper, Nash found a truly ingenious solution to the bargaining game. He assumed the existence of a function which, given the utilities of the players, would produce the division of the good. To find this solution-function, he formulated a number of properties that the function should have. These properties, or axioms, were as follows:

1. The solution should divide the whole unit.

2. The solution should be symmetric in the sense that if both players have the same utility function, then they are indistinguishable and so the division of the good should be the equal split (50-50)

3. If the scale of a utility function were changed, this would not affect the division of the good. For instance, assume both players express their utilities in euro's and the solution of a given problem is 10 euro's for player 1 and 20 for player 2. Should player 1 choose to express his utilities in cents (in effect multiplying his utility function by 100), then the solution to this transformed problem should be that player 1 received a utility of 1000 cents from the division.

4. The final axiom, known as Independence of Irrelevant Alternatives looks slightly more complicated. Suppose that a certain bargaining problem, which is completely determined by the set of possible outcomes in terms of utilities, yields as solution a certain pair of utilities called P. Now consider a different bargaining problem, where the set of possible utility outcomes is contained in those of the first problem, but it also contains P. Since P was the preferred solution in the 'larger' problem, it should also be the solution in this 'smaller' problem. As an analogy: imagine you are in a restaurant

and you have the choice between gyros, moussaka and souvlaki. Assume that your choice is gyros this is P for 'larger' problem. If the waiter comes back and informs you that the souvlaki is not available today, then you you are in a 'smaller' problem, but gyros (P) is still a possible outcome. Then this change, would not affect your choice for gyros.

Nash was able to show that from the properties listed above, which seem quite reasonable, there is only one solution function that satisfies them. This solution is known as the Nash bargaining solution and can be computed as follows. Let $u_i(x)$ be the utility function of player $i = 1, 2$. Find the value of x that maximizes the product $u_1(x)u_2(1-x)$. Then the division of the good is $(x, 1-x)$.

The research that followed the publication of Nash's paper focused on three aspects.

- A number of papers investigated what happens if certain axioms are changed or omitted. This line developed, because some researchers disagreed with the naturalness or applicabilty of these axioms.
- Secondly, there are many papers on applications of Nash's theory to actual problems, usually economic.
- Finally, the problem was taken from the cooperative context and reformulated as a strategic game, or as an evolutionary game.

This work is focused on the latter aspect. But what is exactly a strategic game or an evolutionary game?

1.2 Evolutionary game theory

In strategic game theory, of which John Nash is also one of the founding fathers, we define the strategic games as the games where the players usually have opposing interests. Each player has a number of strategies he can use and the question now becomes which strategy to use to maximize his payoff for the game, knowing that his adversary is considering the same question. An example of a (very complicated) strategic game is chess. An easier example is the *Prisoners Dilemma*. In this game two prisoners, who are suspected of a crime they perpetrated together, each have two strategies: testify against the other or stay silent. If both keep their mouth shut, they both receive a sentence of 1 year. If one testifies and the other doesn't, the talker goes free and the other receives a 7 year sentence. If both testify, both receive a 5 year sentence.

For that kind of games the main solution concept is that of an eqilibrium (also named after Nash). For a two-player game, we call equilibrium a pair of strategies such that neither player has an incentive to change his strategy. For instance, in the Prisoners Dilemma it is easy to see that the pair of strategies where both prisoners testify, is a Nash equilibrium since neither player has an incentive to change his strategy. The reason this game is called a Dilemma, is because there exists an option in which both gain more, namely when both stay silent, yet this is not a Nash equlibrium, since player 1 has an incentive to change his strategy, given that player 2 stays silent.

Considered as a strategic game, the bargaining game has a large set of Nash equlibria. It is clear that for every $x \in [0, 1]$ the division $(x, 1-x)$ is a Nash equilibrium, and there is no reason to prefer one over the other.

The field of Game Theory enjoyed a rapid progress from the 50's onward, both theoretical and applied, mainly to economics. Starting from the early 70's, biologists became interested in the subject. At first sight, this may seem puzzling. On one hand, the actors in the economic models, that were standard at the time, were assumed to have all information about their situation, be completely rational and have the reasoning powers of a trained mathematician. As such, they were ideally suited to play strategic games in the way that the theory predicted, that is the concept of *homo economicus*. On the other hand, fish or other animals for that matter, can hardly be said to have any of these properties. Yet these animals often face situations that can be described using a strategic game. A famous example is a game that exemplifies the fight-or-flight dilemma many living creatures often encounter. When two individuals meet in a conflict situation, they both have two strategies: fight or flight. If both choose fight, both lose, if both choose flight the outcome is a draw and if one fights and the other flies, the fighter wins and the flyer loses a bit. This game can be analysed by standard methods, and we can find the Nash equilibria and so on. Of course, animals don't do this, they simply act.

The idea of *evolutionary game theory*, in particular *replicator dynamics* is that the animals reproduce and that their tactics (for instance fight or flight) are passed on to their offspring. Those animals with a tactic that gives them a large payoff will have relatively more offspring, as evolutionary theory predicts.

Putting these ideas in to a mathematical model results in the *replicator equations*. In the case that there are only a finite number of strategies, these take the form:

$$\frac{dx_i}{dt} = (\pi_i(\mathbf{x}) - \bar{\pi}(\mathbf{x}))x_i \quad , \quad i = 1, \ldots, n.$$

Here, x_i is the fraction of the population who play strategy i. $\pi_i(\mathbf{x})$ is the payoff, or fitness, of strategy i against the current distribution of strategies $\mathbf{x}$ and $\bar{\pi}(\mathbf{x})$ is the average payoff of the total population, $\bar{\pi}(\mathbf{x}) = \sum_{i=1}^{n} x_i \pi_i(\mathbf{x})$. It is easy to check that $x_1(t) + \ldots + x_n(t) = 1$, for all t.

The point of the equation is that strategies whose payoff is larger than average will grow in the population, whereas the fraction of strategies that are doing less than average will diminish.

Soon after its introduction, economists started to become interested in evolutionary game theory. This is because many found the concept of the *homo economicus* as sketched above, to be far from reality. In practice, people making economic decisions do not have all necessary informations, nor do they always act rationally in the textbook sense. An alternative theory, that of *bounded rationality* became popular. This still doesn't explain what evolution has to do with it. However, it turns out that the replicator equations can be also be seen as a type of *learning dynamics*. i.e Let's assume that we have a game with a finite number of strategies and a large group of players playing against another large group, in the sense that at each (short) time interval two players are picked from the respective groups who then play the game against each other. At certain times, one player is picked at random. He observes the strategy of a random other agent from his group. If the payoff of this alternative strategy is higher than his own, he will switch to this strategy with a probability proportional to the difference in payoffs. If the payoff of the other strategy is lower than that of his own, he will stick to his original strategy.
Taking various limits (population of the groups to infinity, time between events to zero) we, miraculously, recover the replicator equations.

Through this learning, we see that succesfull strategies will increase, at the expense of less succesfull ones. Note that the players do not use any of their reasoning powers, they simply use the strategy that they have at the moment. Their rationality has become so bounded that it is actually zero. They might as well be amoebe.

Of course, assuming that the average economic actor has the intelligence of an amoebe is also a caricature, but perhaps this concept is closer to reality than that of the *homo economicus*.

1.3 The replicator equation for the bargaining game

The bargaining game has been studied in an evolutionary setting before, but the replicator equation for this game has never been explored. In [6] of Ruijgrok this is done. Although the equations look quite formidable, it is possible to make some analytic predictions of the outcomes. To confirm these predictions and to solve the equations for situations where analysis does not give an answer, we need numerical simulations.

The equations are basically in the same form as described in the previous section, but with some adaptions.

- First, players have an infinite set of strategies to choose from, namely the interval $[0, 1]$. Rather than having a finite set of variables describing the distribution of strategies in the population, we now have a density $\rho_i(x, t)$, $i = 1, 2$. Here, $\rho_i(x, t)dx$ measures the fraction of players in the population who play a strategy in the interval $[x, x + dx]$. We want $\int_0^1 \rho_i(x, t)dx = 1$, since the measure of the population is equal to 1.

- Secondly, we introduce the concept of *mutation*. Apart from occasionally learning, the agents sometimes simply change their strategy by a small amount. In the learning interpretation of the equations, this may be thought of as experimentation or innovation. In [6], it is shown that this mutation term may be modelled as a diffusion: $\varepsilon_i \frac{\partial^2 \rho_i(x,t)}{\partial x^2}$, with ε_i small.

- Finally, we need to provide boundary conditions. These are chosen to be reflecting, so that the total measure of the population is preserved and equal to one.

The replicator equations of the previous section then become:

$$\begin{aligned}\frac{\partial \rho_1(x,t)}{\partial t} &= \rho_1(x,t)(u_1(x)\int_0^{1-x}\rho_2(y,t)\,dy - \lambda_1(t)) + \epsilon_1\frac{\partial^2\rho_1(x,t)}{\partial s^2}\\ \frac{\partial \rho_2(x,t)}{\partial t} &= \rho_2(x,t)(u_2(x)\int_0^{1-x}\rho_1(y,t)\,dy - \lambda_2(t)) + \epsilon_2\frac{\partial^2\rho_2(x,t)}{\partial s^2},\end{aligned}$$

where

$$\begin{aligned}\lambda_1(t) &= \int_0^1\int_0^{1-x} u_1(x)\rho_1(x,t)\rho_2(y,t)\,dy\,dx\\ \lambda_2(t) &= \int_0^1\int_0^{1-x} u_2(x)\rho_1(y,t)\rho_2(x,t)\,dy\,dx\end{aligned}$$

are the average payoffs to members of the two groups and $u_1(x), u_2(x)$, are the utility functions of each group. The value $r = \frac{\epsilon_1}{\epsilon_2}$, designates the ratio of the mutations.
To complete the description of the problem, boundary conditions and an initial value are imposed:

$$\begin{aligned}&\frac{\partial\rho_i(0,t)}{\partial x} = \frac{\partial\rho_i(1,t)}{\partial x} = 0 \quad , \quad i = 1,2.\\ &\rho_i(x,0) = \rho_{0i}(x) \quad , \quad i = 1,2.\end{aligned}$$

2 The model

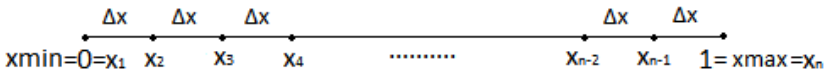
As we have shown in the introduction, the mathematical interpretation of the model is given by a system of two time dependent partial integro-differerential equations, named the replicator equations for the Nash bargaining game.

$$\begin{cases}\frac{\partial\rho_1(x,t)}{\partial t} = \epsilon_1\frac{\partial^2\rho_1(x,t)}{\partial x^2} + \rho_1(x,t)\left[u_1(x)\int_0^{1-x}\rho_2(y,t)\mathrm{d}y - \int_0^1\rho_1(x,t)u_1(x)\left(\int_0^{1-x}\rho_2(y,t)\mathrm{d}y\right)\mathrm{d}x\right],\\ \frac{\partial\rho_2(x,t)}{\partial t} = \epsilon_2\frac{\partial^2\rho_2(x,t)}{\partial x^2} + \rho_2(x,t)\left[u_2(x)\int_0^{1-x}\rho_1(y,t)\mathrm{d}y - \int_0^1\rho_2(x,t)u_2(x)\left(\int_0^{1-x}\rho_1(y,t)\mathrm{d}y\right)\mathrm{d}x\right]\end{cases},$$

where the utility functions that we will use, are the increasing concave functions $u_1(x) = x^{a_1}$ and $u_2(x) = x^{a_2}$, for $0 \leqslant a_1, a_2 \leqslant 1$ and $\epsilon_1, \epsilon_2 \in \mathbb{R}$. We want to approximate the density functions $\rho_1(x,t)$ and $\rho_2(x,t)$, as $\epsilon_1, \epsilon_2 \to 0$.

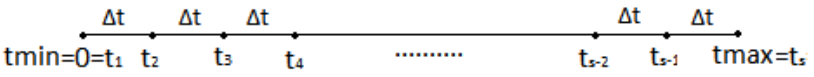
3 Fixed grid method

In fixed grid method, we place the spatial and time gridpoints, in such a way that the distance between any two successive gridpoints is equal. So, suppose we have n spatial gridpoints and s time gridpoints. We equidistribute these n spatial gridpoints into the spatial domain $[0,1]$. That yields to a subdivision of the spatial interval into N equal subintervals, as in the following picture



so we get N equal subintervals of length $\Delta x = \frac{xmax-xmin}{N} = \frac{1}{N}$ and the spatial gridpoints are given by $x_i = (i-1)\Delta x$ for $i = 1, 2, \ldots, n$ (where $n = N+1$).

Similarly we place s time gridpoints into the time domain $[0, tmax]$. In other words, we subdivide the time domain into S equal subintervals.



so, we get S equal subintervals of length $\Delta t = \frac{tmax-tmin}{S} = \frac{tmax}{S}$ and the time gridpoints are given by $t_j = (j-1)\Delta t$ for $j = 1, 2, \ldots, s$ (where $s = S+1$).

For every of these fixed time gridpoints, we calculate the values of both ρ_1 and ρ_2 curves, for every fixed spatial gridpoint. So, for the $1st$ time gridpoint t_1, which corresponds to time $t = 0$, we calculate the values of ρ_1 and ρ_2, on every fixed gridpoint x_i, $i = 1, 2, \ldots, n$ on space. Afterwards, we make a time step of length Δt and we reach the $2nd$ time gridpoint t_2, which coresponds to time $t = \Delta t$ and we calculate the values of ρ_1 and ρ_2, on every fixed gridpoint x_i, $i = 1, 2, \ldots, n$ on space. We continue doing that, until we reach the last time gridpoint t_s, which corresponds to time $t = tmax$.

3.1 Numerical approximation

In this section, we will derive the formulas that will allow as calculate the values of $\rho_1(x_i, t_j)$ and $\rho_2(x_i, t_j)$ on every fixed spatial gridpoint x_i, $i = 1, 2, \ldots, n$, for any given time gridpoint t_j, $j = 1, 2, \ldots, s$. General information for the theory of numerical approximation of partial differential equations, can be found in references [1] and [3].

We start by deriving the numerical approximation formulas of the second order space derivatives $\frac{\partial^2 \rho_1}{\partial x^2}$, $\frac{\partial^2 \rho_2}{\partial x^2}$. Central Difference scheme is used here, the scheme can be easily derived by Taylor expansion on ρ_1 and ρ_2 at some point x_i.
Central difference scheme:

$$\begin{aligned} \frac{\partial^2 \rho_1}{\partial x^2}\Big|_{(x_i)=((i-1)\Delta x)} &\approx \frac{\rho_1(x_{i+1}) - 2\rho_1(x_i) + \rho_1(x_{i-1})}{(\Delta x)^2}, \\ \frac{\partial^2 \rho_2}{\partial x^2}\Big|_{(x_i)=((i-1)\Delta x)} &\approx \frac{\rho_2(x_{i+1}) - 2\rho_2(x_i) + \rho_2(x_{i-1})}{(\Delta x)^2}, \end{aligned}$$

for every $i = 1, 2, \ldots, n$.

We continue by deriving the numerical approximation formulas of the integrals $F_1(x) = \int_0^{1-x} \rho_2(y)\mathrm{d}y$, $F_2(x) = \int_0^{1-x} \rho_1(y)\mathrm{d}y$. These integrals are functions of x, that explains the usage of $F_i(x)$ instead of F_i. A variety of numerical methods for the approximation of integrals can be found in §2.12.4 of reference [1]. At this part, we will use the trapezoidal rule defined as follows, to calculate the definite integrals on some specified spatial domain.
Trapezoidal rule:
For integrating area $[a, b] \subset [0, \infty)$, we have:

$$\int_a^b f(y)\mathrm{d}y \approx (b-a)\frac{f(a)+f(b)}{2}.$$

Hence, for any $[x_1, x_n] \subset [0, \infty)$, we have:

$$\begin{aligned} \int_{x_1}^{x_n} f(y)\mathrm{d}y &\approx \frac{\Delta x}{2}\left[f(x_1) + 2f(x_2) + \ldots, +2f(x_{n-1}) + f(x_n)\right] \\ &= \frac{\Delta x}{2}\left[f(x_1) + f(x_n) + 2\sum_{q=2}^{n-1} f(x_q)\right]. \end{aligned}$$

The rest formulas that will be derived from now on, are based on the latter definition.
For integrating area $[a, x_i]$, where x_i can be any spatial gridpoint, we have:

$$\begin{aligned} \int_a^{x_i} f(y)\mathrm{d}y &= \int_{x_1}^{x_i} f(y)\mathrm{d}y \\ &\approx \frac{\Delta x}{2}\left[f(x_1) + 2f(x_2) + \ldots, +2f(x_{i-1}) + f(x_i)\right] \\ &= \frac{\Delta x}{2}\left[f(x_1) + f(x_i) + 2\sum_{q=2}^{i-1} f(x_q)\right]. \end{aligned}$$

For integrating area $[a, b - x_i]$, where x_i can be any spatial gridpoint, we have:

$$\int_a^{b-x_i} f(y)\mathrm{d}y = \int_{x_1}^{b-x_i} f(y)\mathrm{d}y =$$

Note that we have $b - x_i = (n-1)\Delta x - (i-1)\Delta x \Rightarrow b - x_i = (n-1-i+1)\Delta x = (n-i)\Delta x = ((n-i+1)-1)\Delta x = x_{n-i+1}$, where x_{n-i+1} is also some spatial gridpoint. So,

$$\begin{aligned}\int_{x_1}^{x_{n-i+1}} f(y)\mathrm{d}y &\approx \frac{\Delta x}{2}\left[f(x_1) + 2f(x_2) + \ldots, +2f(x_{n-i}) + f(x_{n-i+1})\right] \\ &= \frac{\Delta x}{2}\left[f(x_1) + f(x_{n-i+1}) + 2\sum_{q=2}^{n-i} f(x_q)\right].\end{aligned}$$

Hence, the numerical approximation formula of $F_1(x) = \int_0^{1-x} \rho_2(y)\mathrm{d}y$, in every spatial gridpoint x_i, $i = 1, 2, \ldots, n$ is:

$$\begin{aligned}F_1(x_i) &= \int_0^{1-x_i} \rho_2(y)\mathrm{d}y = \int_0^{x_{n-i+1}} \rho_2(y)\mathrm{d}y \\ &\approx \frac{\Delta x}{2}\left[\rho_2(x_1) + \rho_2(x_{n-i+1}) + 2\sum_{q=2}^{n-i} \rho_2(x_q)\right].\end{aligned}$$

Similarly, for $F_2(x) = \int_0^{1-x} \rho_1(y)\mathrm{d}y$, we have:

$$\begin{aligned}F_2(x_i) &= \int_0^{1-x_i} \rho_1(y)\mathrm{d}y = \int_0^{x_{n-i+1}} \rho_1(y)\mathrm{d}y \\ &\approx \frac{\Delta x}{2}\left[\rho_1(x_1) + \rho_1(x_{n-i+1}) + 2\sum_{q=2}^{n-i} \rho_1(x_q)\right],\end{aligned}$$

for every x_i, $i = 1, 2, \ldots, n$.

Other integrals that are part of the model and need to be approximated, are the definite integrals $P_1 = \int_0^1 \rho_1(x)u_1(x)\left(\int_0^{1-x} \rho_2(y)\mathrm{d}y\right)\mathrm{d}x$, $P_2 = \int_0^1 \rho_2(x)u_2(x)\left(\int_0^{1-x} \rho_1(y)\mathrm{d}y\right)\mathrm{d}x$.

The values $F_1(x) = \int_0^{1-x} \rho_2(y)\mathrm{d}y$ and $F_2(x) = \int_0^{1-x} \rho_1(y)\mathrm{d}y$, are part of P_1 and P_2 respectivelly and have been already approximated by some formulas, so we have:

$$\begin{aligned}P_1 &= \int_0^1 \rho_1(x)u_1(x)\left(\int_0^{1-x} \rho_2(y)\mathrm{d}y\right)\mathrm{d}x = \int_0^1 \rho_1(x)u_1(x)F_1(x)\mathrm{d}x, \\ P_2 &= \int_0^1 \rho_2(x)u_2(x)\left(\int_0^{1-x} \rho_1(y)\mathrm{d}y\right)\mathrm{d}x = \int_0^1 \rho_2(x)u_2(x)F_2(x)\mathrm{d}x.\end{aligned}$$

Since these are definite integrals, the results will be just numbers independent of x. That explains the usage of P_i instead of $P_i(x)$. Let $f_1(x) = \rho_1(x)u_1(x)F_1(x)$, then we have that

$$P_1 = \int_0^1 \rho_1(x)u_1(x)F_1(x)\mathrm{d}x = \int_0^1 f_1(x)\mathrm{d}x = \int_{x_1}^{x_n} f_1(x)\mathrm{d}x$$

and by the definition of trapezoidal rule we get the formula

$$\begin{aligned}&\approx \frac{\Delta x}{2}\left[f_1(x_1) + 2f_1(x_2) + \ldots + 2f_1(x_{n-1}) + f_1(x_n)\right] \\ &= \frac{\Delta x}{2}\left[f_1(x_1) + f_1(x_n) + 2\sum_{q=2}^{n-1} f_1(x_q)\right] \\ &= \frac{\Delta x}{2}\left[\rho_1(x_1)u_1(x_1)F_1(x_1) + \rho_1(x_n)u_1(x_n)F_1(x_n) + 2\sum_{q=2}^{n-1} \rho_1(x_q)u_1(x_q)F_1(x_q)\right].\end{aligned}$$

This is the approximation formula of the value P_1. Note that in order to calculate the value P_1, we must first have calculated all the values $F_1(x_1), F_1(x_2), \ldots, F_1(x_n)$. Algorithically speaking, the value P_1, will have to be calculated only once at the beginning of each time iteration.

Similarly, we derive the approximation formula of the value P_2.

$$\begin{aligned} P_2 &= \int_0^1 \rho_2(x)u_2(x)F_2(x)\mathrm{d}x = \int_{x_1}^{x_n} \rho_2(x)u_2(x)F_2(x)\mathrm{d}x \\ &\approx \frac{\Delta x}{2}\left[\rho_2(x_1)u_2(x_1)F_2(x_1) + \rho_2(x_n)u_2(x_n)F_2(x_n) + 2\sum_{q=2}^{n-1}\rho_2(x_q)u_2(x_q)F_2(x_q)\right]. \end{aligned}$$

As for P_1, first we must have calculated all the values $F_2(x_1), F_2(x_2), \ldots, F_2(x_n)$, in order to be able to calculate P_2. The value P_2, will have to be calculated only once at the beginning of each time iteration.

Consider the initial model

$$\begin{cases} \frac{\partial \rho_1}{\partial t} = \epsilon_1 \frac{\partial^2 \rho_1}{\partial x^2} + \rho_1(x)\left[u_1(x)\int_0^{1-x}\rho_2(y)\mathrm{d}y - \int_0^1 \rho_1(x)u_1(x)\left(\int_0^{1-x}\rho_2(y)\mathrm{d}y\right)\mathrm{d}x\right], \\ \frac{\partial \rho_2}{\partial t} = \epsilon_2 \frac{\partial^2 \rho_2}{\partial x^2} + \rho_2(x)\left[u_2(x)\int_0^{1-x}\rho_1(y)\mathrm{d}y - \int_0^1 \rho_2(x)u_2(x)\left(\int_0^{1-x}\rho_1(y)\mathrm{d}y\right)\mathrm{d}x\right] \end{cases},$$

$$\Leftrightarrow \begin{cases} \frac{\partial \rho_1}{\partial t} = \epsilon_1 \frac{\partial^2 \rho_1}{\partial x^2} + \rho_1(x)\left[u_1(x)F_1(x) - P_1\right], \\ \frac{\partial \rho_2}{\partial t} = \epsilon_2 \frac{\partial^2 \rho_2}{\partial x^2} + \rho_2(x)\left[u_2(x)F_2(x) - P_2\right] \end{cases}.$$

The right side of both equations of the model, have been fully approximated by numerical formulas.

Consider the first equation of the latter model.

$$\frac{\partial \rho_1}{\partial t} = \epsilon_1 \frac{\partial^2 \rho_1}{\partial x^2} + \rho_1(x)\left[u_1(x)F_1(x) - P_1\right],$$

for some $x = x_i$, we have:

$$\begin{aligned} \frac{\partial \rho_1}{\partial t}\Big|_{(x_i)} &= \epsilon_1 \frac{\partial^2 \rho_1}{\partial x^2}\Big|_{(x_i)} + \rho_1(x_i)\left[u_1(x_i)F_1(x_i) - P_1\right] \\ &\approx \epsilon_1 \frac{\rho_1(x_{i+1}) - 2\rho_1(x_i) + \rho_1(x_{i-1})}{(\Delta x)^2} + \rho_1(x_i)\left[u_1(x_i)F_1(x_i) - P_1\right], \end{aligned}$$

where everything have been numerically approximated.

Now consider the second part of the latter model.

$$\frac{\partial \rho_2}{\partial t} = \epsilon_2 \frac{\partial^2 \rho_2}{\partial x^2} + \rho_2(x)\left[u_2(x)F_2(x) - P_2\right],$$

for some $x = x_i$, we have:

$$\begin{aligned} \frac{\partial \rho_2}{\partial t}\Big|_{(x_i)} &= \epsilon_2 \frac{\partial^2 \rho_2}{\partial x^2}\Big|_{(x_i)} + \rho_2(x_i)\left[u_2(x_i)F_2(x_i) - P_2\right] \\ &\approx \epsilon_2 \frac{\rho_2(x_{i+1}) - 2\rho_2(x_i) + \rho_2(x_{i-1})}{(\Delta x)^2} + \rho_2(x_i)\left[u_2(x_i)F_2(x_i) - P_2\right], \end{aligned}$$

where everything have been numerically approximated.

Hence, the last part that remains to be approximated, is the left side of the equations in the model, which is consisted by $1st$ order time derivatives $\frac{\partial \rho_1}{\partial t}$, $\frac{\partial \rho_2}{\partial t}$. Up until now, time was not considered at all. The time gridpoints, were not mentioned in any of the previous formulas. That is because until now, we were considering time to be constant since it does not effect the right side of the model. But now time is essential for the numerical approximation of the time derivatives.

For this approximations, we will use another classic method known as Euler forward method. It is derived as follows.

Euler forward method:

Given the definition of the derivative

$$\frac{\partial f}{\partial t}(x,t) = \lim_{\Delta t \to 0}\frac{f(x,t+\Delta t) - f(x,t)}{\Delta t},$$

for some spatial gridpoint $x = x_i$ and some time gridpoint $t = t_j$ we have

$$\frac{\partial f}{\partial t}\Big|_{(x_i,t_j)} \approx \frac{f(x_i, t_j + \Delta t) - f(x_i, t_j)}{\Delta t},$$

where the value $t_j + \Delta t = (j-1)\Delta t + \Delta t = j\Delta t = ((j+1)-1)\Delta t = t_{j+1}$, corresponds to the next time gridpoint. So,

$$\frac{\partial f}{\partial t}\Big|_{(x_i,t_j)} \approx \frac{f(x_i, t_{j+1}) - f(x_i, t_j)}{\Delta t}.$$

By applying this method to the time derivatives of our model, we get:

$$\begin{aligned}
\frac{\partial \rho_1}{\partial t}\Big|_{(x_i,t_j)} &\approx \frac{\rho_1(x_i, t_{j+1}) - \rho_1(x_i, t_j)}{\Delta t}, \\
\frac{\partial \rho_2}{\partial t}\Big|_{(x_i,t_j)} &\approx \frac{\rho_2(x_i, t_{j+1}) - \rho_2(x_i, t_j)}{\Delta t}.
\end{aligned}$$

Now we are ready to derive the final formulas that are able to numeriacally approximate the values of the curves ρ_1 and ρ_2. By pluging in all the previous approximations into the initial model, the first equation of the model takes the form:

$$\frac{\partial \rho_1}{\partial t}(x,t) = \epsilon_1 \frac{\partial^2 \rho_1}{\partial x^2}(x,t) + \rho_1(x,t)\left[u_1(x)F_1(x,t) - P_1\right],$$

and for some time gridpoint t_j, $j = 1, 2, \ldots, s$ and any spatial gridpoint $x = x_i$, $i = 1, 2, \ldots, n$, we have:

$$\begin{aligned}
\frac{\partial \rho_1}{\partial t}\Big|_{(x_i,t_j)} &= \epsilon_1 \frac{\partial^2 \rho_1}{\partial x^2}\Big|_{(x_i,t_j)} + +\rho_1(x_i,t_j)\left[u_1(x_i)F_1(x_i,t_j) - P_1\right] \\
\Rightarrow \frac{\rho_1(x_i,t_{j+1}) - \rho_1(x_i,t_j)}{\Delta t} &= \epsilon_1 \frac{\rho_1(x_{i+1},t_j) - 2\rho_1(x_i,t_j) + \rho_1(x_{i-1},t_j)}{(\Delta x)^2} \\
&\quad +\rho_1(x_i,t_j)\left[u_1(x_i)F_1(x_i,t_j) - P_1\right] \\
\Rightarrow \rho_1(x_i,t_{j+1}) &= \Delta t \epsilon_1 \frac{\rho_1(x_{i+1},t_j) - 2\rho_1(x_i,t_j) + \rho_1(x_{i-1},t_j)}{(\Delta x)^2} \\
&\quad +\Delta t \rho_1(x_i,t_j)\left[u_1(x_i)F_1(x_i,t_j) - P_1\right] + \rho_1(x_i,t_j),
\end{aligned}$$

where

$$\begin{aligned}
F_1(x_i,t_j) &= \frac{\Delta x}{2}\left[\rho_2(x_1,t_j) + \rho_2(x_{n-i+1},t_j) + 2\sum_{q=2}^{n-i}\rho_2(x_q,t_j)\right], \\
P_1 &= \frac{\Delta x}{2}\left[\rho_1(x_1,t_j)u_1(x_1)F_1(x_1,t_j) + \rho_1(x_n,t_j)u_1(x_n)F_1(x_n,t_j)\right. \\
&\quad \left. + 2\sum_{q=2}^{n-1}\rho_1(x_q,t_j)u_1(x_q)F_1(x_q,t_j)\right].
\end{aligned}$$

Computationally, for each and every $j = 1, 2, \ldots, s$, we first need to compute $F_1(x_i, t_j)$ $\forall i = 1, \ldots, n$ and save in a vector $F_1 = [F_1(x_1,t_j), F_1(x_2,t_j), \ldots, F_1(x_n,t_j)]$ with dimension $1 \times n$, and then, we will be able to calculate P_1, which will be a single value.

Similarly, the second equation of the model takes the form:

$$\frac{\partial \rho_2}{\partial t}(x,t) = \epsilon_2 \frac{\partial^2 \rho_2}{\partial x^2}(x,t) + \rho_2(x,t)\left[u_2(x)F_2(x,t) - P_2\right],$$

and for some time gridpoint $t = t_j$, $j = 1, 2, \ldots, s$ and any spatial gridpoint $x = x_i$, $i = 1, 2, \ldots, n$, we have:

$$\begin{aligned}
\frac{\partial \rho_2}{\partial t}\Big|_{(x_i,t_j)} &= \epsilon_2 \frac{\partial^2 \rho_2}{\partial x^2}\Big|_{(x_i,t_j)} + \rho_2(x_i,t_j)\left[u_2(x_i)F_2(x_i,t_j) - P_2\right] \\
\Rightarrow \frac{\rho_2(x_i,t_{j+1}) - \rho_2(x_i,t_j)}{\Delta t} &= \epsilon_2 \frac{\rho_2(x_{i+1},t_j) - 2\rho_2(x_i,t_j) + \rho_2(x_{i-1},t_j)}{(\Delta x)^2} \\
&\quad +\rho_2(x_i,t_j)\left[u_2(x_i)F_2(x_i,t_j) - P_2\right] \\
\Rightarrow \rho_2(x_i,t_{j+1}) &= \Delta t \epsilon_2 \frac{\rho_2(x_{i+1},t_j) - 2\rho_2(x_i,t_j) + \rho_2(x_{i-1},t_j)}{(\Delta x)^2} \\
&\quad +\Delta t \rho_2(x_i,t_j)\left[u_2(x_i)F_2(x_i,t_j) - P_2\right] + \rho_2(x_i,t_j),
\end{aligned}$$

where

$$
\begin{aligned}
F_2(x_i,t_j) &= \frac{\Delta x}{2}\left[\rho_1(x_1,t_j)+\rho_1(x_{n-i+1},t_j)+2\sum_{q=2}^{n-i}\rho_1(x_q,t_j)\right],\\
P_2 &= \frac{\Delta x}{2}\left[\rho_2(x_1,t_j)u_2(x_1)F_2(x_1,t_j)+\rho_2(x_n,t_j)u_2(x_n)F_2(x_n,t_j)\right.\\
&\quad \left.+2\sum_{q=2}^{n-1}\rho_2(x_q,t_j)u_2(x_q)F_2(x_q,t_j)\right].
\end{aligned}
$$

Computationally, also here, we first need to compute $F_2(x_i,t_j)\ \forall i=1,\ldots,n$ and save in a vector $F_2=[F_2(x_1,t_j),F_2(x_2,t_j),\ldots,F_2(x_n,t_j)]$ with dimension $1\times n$, then we are able to calculate P_2, which will be a single value.

By the latter two numerical formulas, we can see that we can approximate the values of both $\rho_1(x_i,t_{j+1})$ and $\rho_2(x_i,t_{j+1})$, for every spatial gridpoint x_i, $i=1,2,\ldots,n$ at a given time gridpoint t_{j+1}, by just using the values of $\rho_1(x_i,t_j)$ and $\rho_2(x_i,t_j)$ of the previous time gridpoint t_j. Finally, the above numerical formulas give a fully discrete solution for all the gridpoints $(x_i,t_j)\ \forall i=1,\ldots,n$ for any given time gridpoint t_j, $j=1,2,\ldots,s$.

3.2 Stability

In order to use the numerical scheme, that we derived in the previous section, we first need to investigate its stability. That is to find some restriction, for the values that the scheme's parameters can take, such that, when the restriction is fulfilled, the scheme will be stable.

Since the equations of the scheme, are difficult to be analyzed, a theoretical approach is not an option. So, we used a more practical way to do that. The parameters were quite a few, so we had to choose some specific cases and focus on them. Several numerical experiments have been done, in order to acquire some relationship among the parameters of the scheme, for which the scheme is stable. In order to check for what values the above scheme is stable, we followed the following procedure.

We fixed the parameters a_1,a_2, ϵ_1,ϵ_2 and $tmax$. Once the scheme is free of these parameters, we give a number of spatial gridpoints n, and find the minimum possible number of time gridpoints s, for which the scheme is stable. This procedure was repeated several times, each time for a different number of spatial gridpoints.

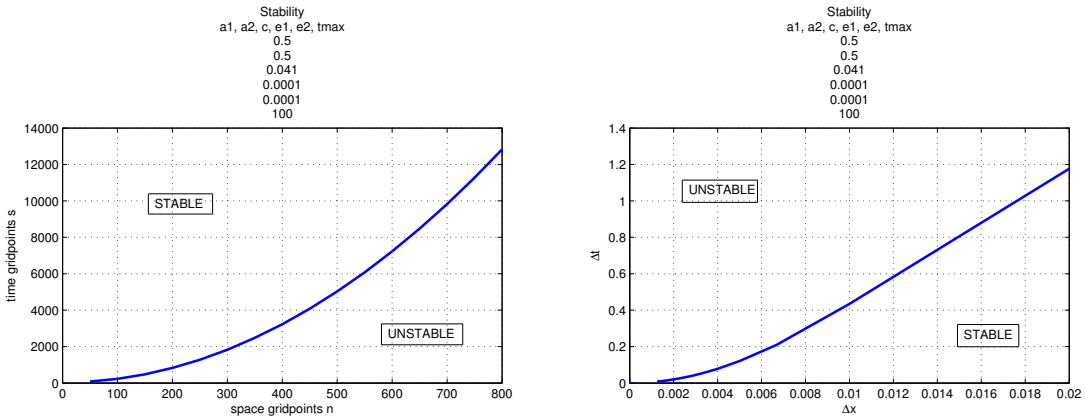
The figures below, show the results for several cases. Leften figures, correspond to the relation between the number of spatial gridpoints n and the number of time gridpoints s and right figures, correspond to the relation between Δx and Δt.

In order to see what happens as $tmax$ is changing, we doubled the value of $tmax$ and followed the same procedure.

Hence figures below correspond to the following cases:

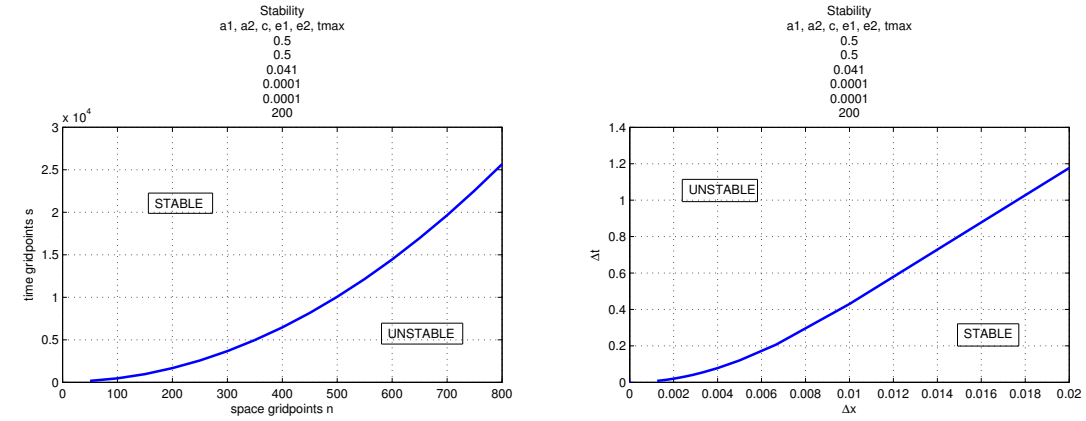
Case	$a_1=a_2$	$\epsilon_1=\epsilon_2$	$tmax$
1	$0,5$	10^{-4}	100
2	$0,5$	10^{-4}	200
3	$0,5$	10^{-5}	100
4	$0,5$	10^{-5}	200
5	$0,5$	10^{-7}	100
6	$0,5$	10^{-7}	200

For $\epsilon_1 = \epsilon_2 = 10^{-4}$ and $tmax = 100$



We can see from the left figure that, as the number of spatial gridpoints is increasing, the number of time gridpoints must be increased even more.

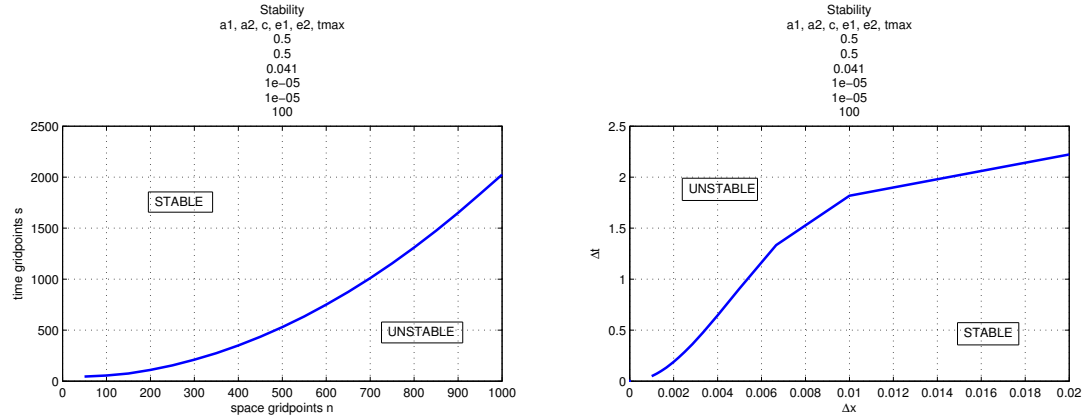
For $\epsilon_1 = \epsilon_2 = 10^{-4}$ and $tmax = 200$



By comparing the first figure, with the respective one in the previous case, we see that, when $tmax$ is doubled, the number of time gridpoints that we need to use, in order to keep the scheme stable, is almost doubled. In the previous case, for $n = 500$, we would need $s \geqslant 5000$ in order for the scheme to be stable, while in this case we need $s \geqslant 10000$.

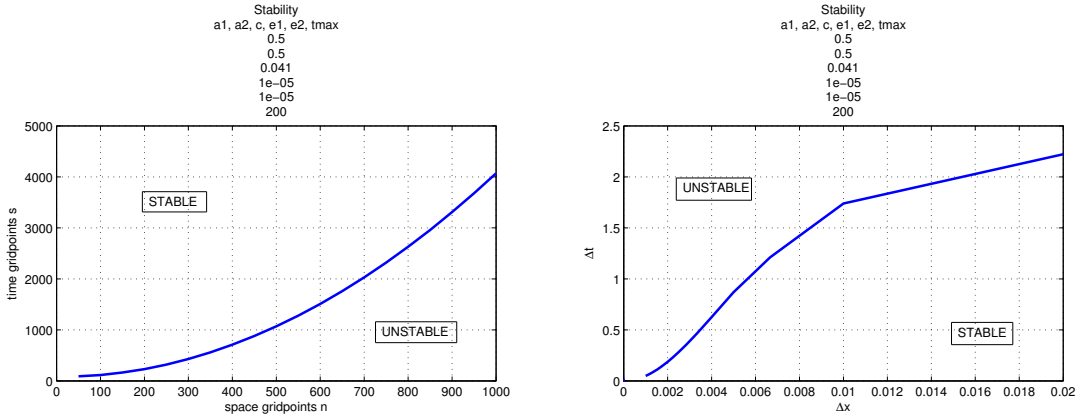
Another thing that we observe, is that in both cases, the right figures which correspond to the relation between Δx and Δt, are the same.

For $\epsilon_1 = \epsilon_2 = 10^{-5}$ and $tmax = 100$



Here we can see that, as ϵ_1, ϵ_2 are getting closer to 0, we need less time gridpoints. Even though, the pattern among n and s, has the same properties. So, also here, an increase of n, yields a larger increase of s.

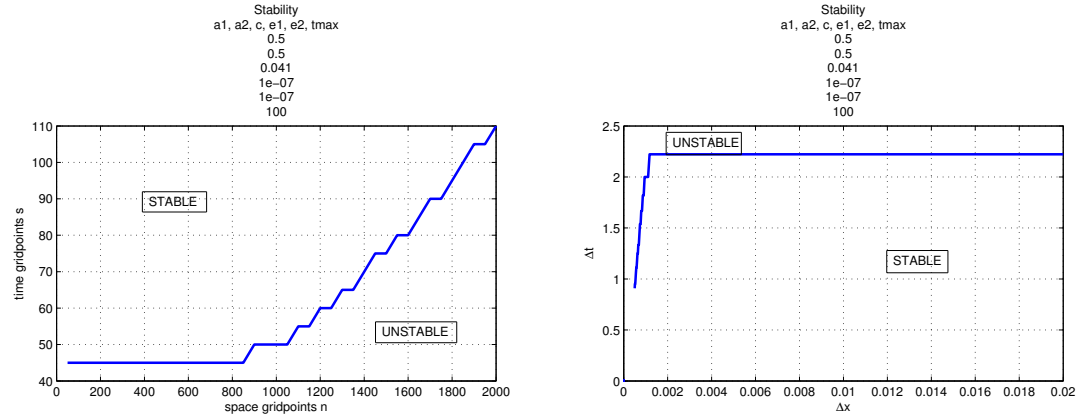
For $\epsilon_1 = \epsilon_2 = 10^{-5}$ and $tmax = 200$



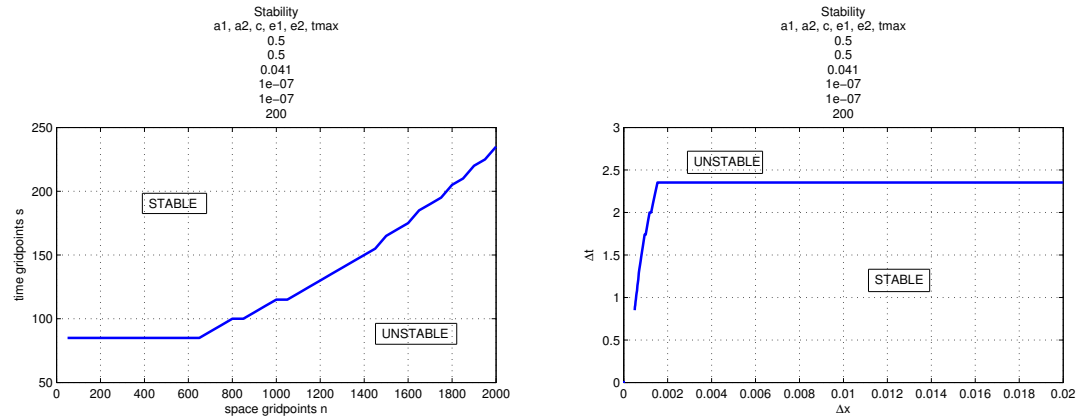
When $tmax$ is doubled the number of time gridpoints must also be doubled in order for the scheme to be stable.

Also here, the pattern of the curve in the right figure, is the same as for the one in the previous case. So, although time $tmax$ is double here, the pattern between Δx and Δt remains the same.

For $\epsilon_1 = \epsilon_2 = 10^{-7}$ and $tmax = 100$



For $\epsilon_1 = \epsilon_2 = 10^{-7}$ and $tmax = 200$



The same properties as in the previous cases, hold also for the case where $\epsilon_1, \epsilon_2 = 10^{-7}$.

In this section, we found where the scheme is stable and where not. In general, the fact that for any value $\epsilon = \epsilon_1 = \epsilon_2$, the pattern of the relation between Δx and Δt is the same for both $tmax = 100$ and $tmax = 200$, gives a general feeling of stability of the numerical scheme.

Also, the left figures, for all cases, lead to the fact that, in order for the scheme to be stable, by increasing the number of spatial gridpoints n, we need to increase the number of time gridpoints s even more. That conclution, becomes even stronger, when we already have some large number of spatial gridpoints.

At last, another thing that we need to mention, is that, by the figures above, we can understand that for some number of spatial gridpoints n, by doubling the time $tmax$, we need to at least double the number of time gridpoints s.

Now we are ready to use the numerical scheme we derived, to approximate the curves ρ_1 and ρ_2.

3.3 Results

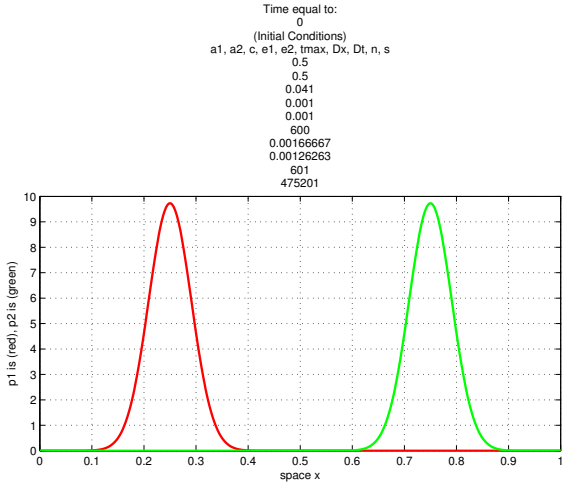
In this section, we use the numerical scheme we derived in §3.1, to approximate the values of ρ_1 and ρ_2 of the initial model. Below, we present the results.

There are two main goals. First one, is to investigate the behaviour of the model, as ϵ_1 and ϵ_2, are getting closer to zero. Then, we want to see what happens in the case where $\epsilon_1 \neq \epsilon_2$.

The results that are presented here, correspond to the following cases:

Case	$a_1 = a_2$	ϵ_1	ϵ_2	$tmax$
1	$0,5$	10^{-3}	10^{-3}	600
2	$0,5$	10^{-4}	10^{-4}	1000
3	$0,5$	10^{-5}	10^{-5}	10000
4	$0,5$	$1,2 * 10^{-5}$	10^{-5}	10000
5	$0,5$	$1,4 * 10^{-5}$	10^{-5}	10000
6	$0,5$	$1,6 * 10^{-5}$	10^{-5}	10000
7	$0,5$	$1,8 * 10^{-5}$	10^{-5}	10000
8	$0,5$	$2 * 10^{-5}$	10^{-5}	10000

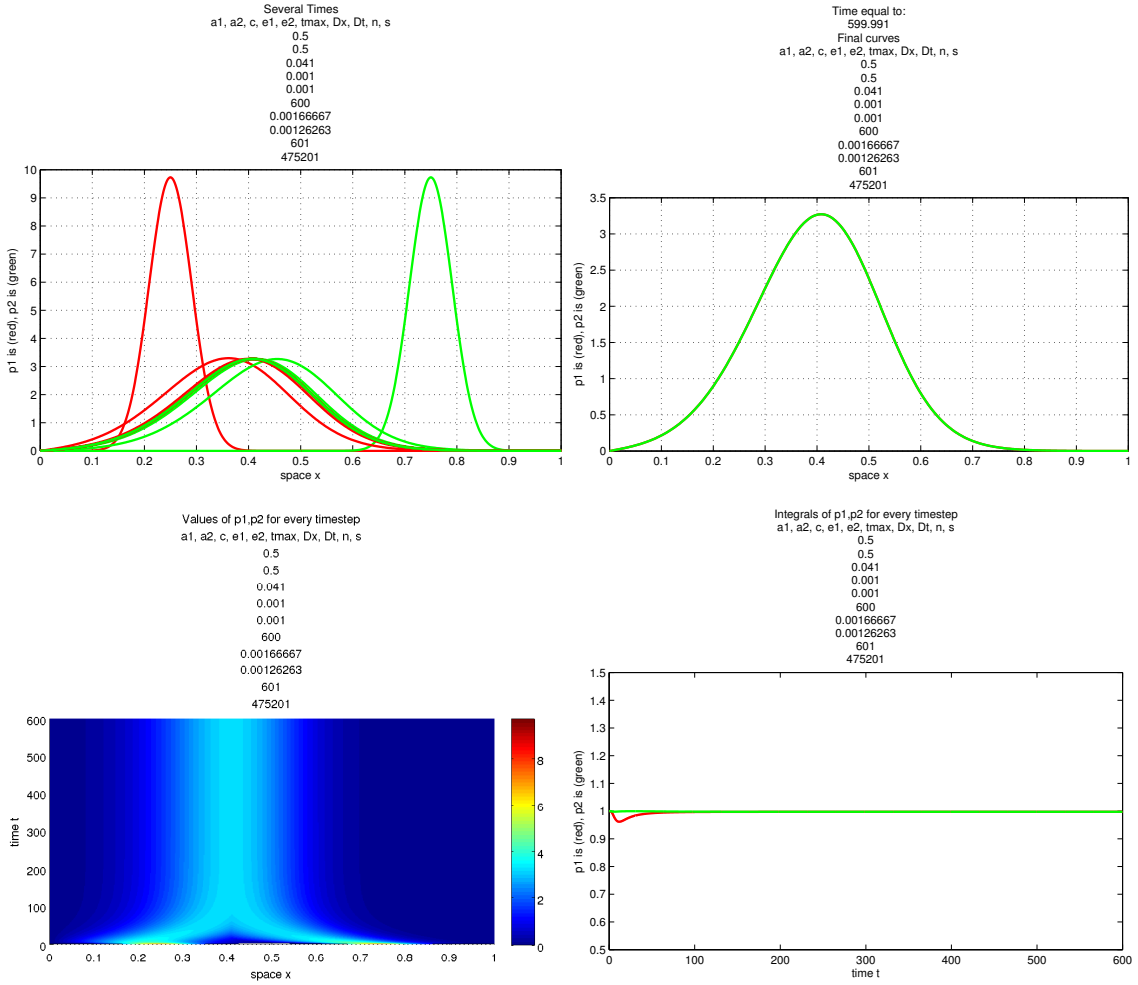
In all cases we present here, we have used Gaussian functions as initial conditions. The volume of the Gaussians, is taken to be equal to 1. So, at time $t = 0$, the curves ρ_1 and ρ_2 are given in the following figure



The curve ρ_1 (red), has its peak on $x = 1/4$, while ρ_2 (green), has its peak on $x = 3/4$.

In every case, the first figure, gives the approximation of the model for some intermediate times, while the second figure, gives the approximation of the model at time $t = tmax$. Furthermore, the third figure gives a 3D representation of the model for $0 \leq x \leq 1$ and $0 \leq t \leq tmax$. At last, the fourth figure, gives the volume of both ρ_1 and ρ_2, for every time $0 \leq t \leq tmax$. That is done by integrating ρ_1 and ρ_2 on the interval $[0, 1]$. Trapezoidal rule was used for the integration.

For $\epsilon_1 = \epsilon_2 = 10^{-3}$ and $tmax = 600$



From the first figure, we can see that both curves are moving towards the same position.

Initially, for time $t = 0$, we have two Gaussian functions, with height almost equal to 10. From the first time steps, we see that both curves are morphologically changing. Both ρ_1 and ρ_2, are becoming shorter while both are gaining width. The morphological change, is taking place until both curves reach height equal to something more than 3. Afterwards, the curves do not change their shape any further, but they continue moving until they reach the same position.

The fusion is clearer in the second figure, where we can easily see that at time $t = tmax = 600$, ρ_1 and ρ_2, have the same position and shape.

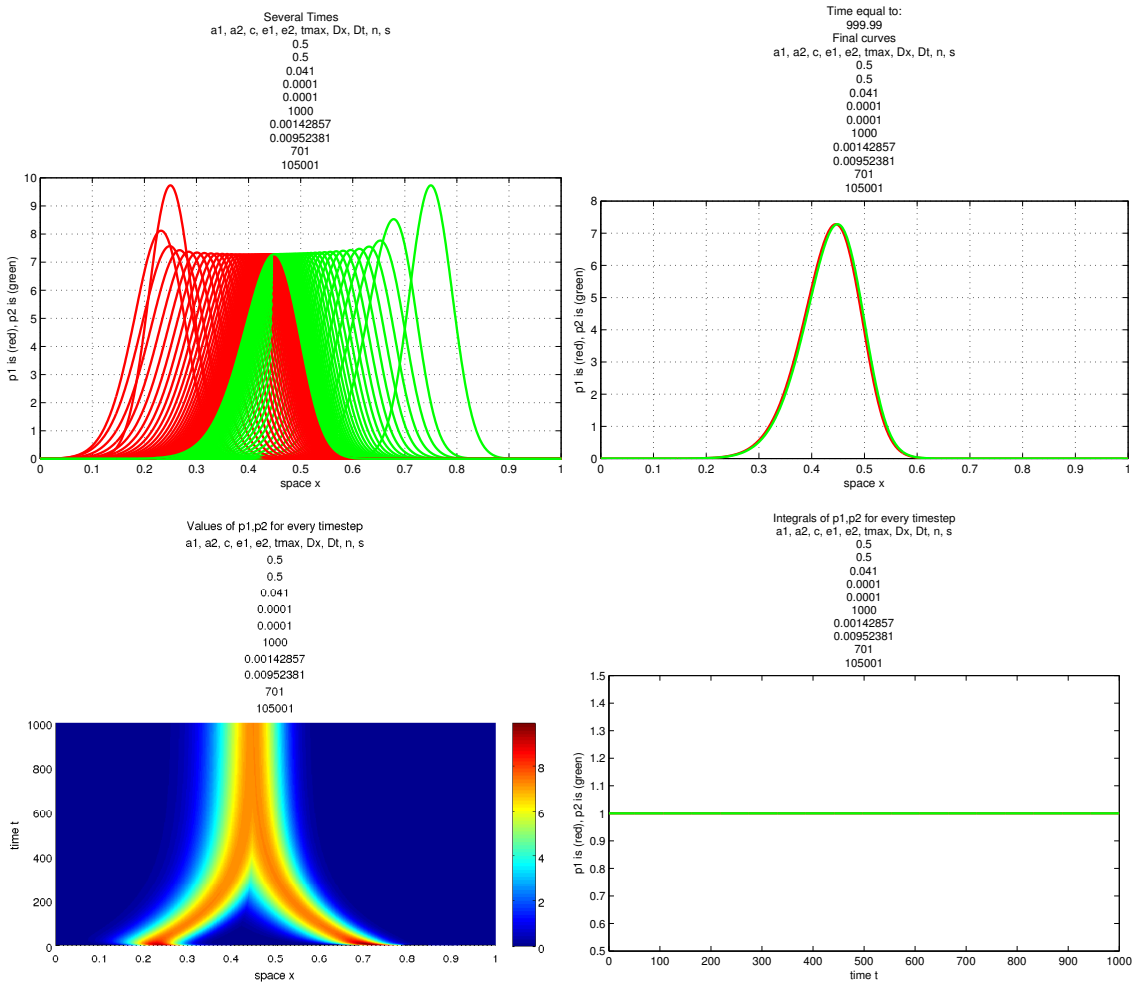
The next figure shows a 3D representation of the curves. We can see that, they converge to the equal curves at around time $t = 300$. Ever since, the curves remain stable.

One thing that we have to mention, is that, in the third figure, we can see that the curve ρ_1, initially is moving away from ρ_2. That happens for a very small time interval, at the very beggining. But soon, it changes its direction.

In the last figure, we can see that, although the curves are changing their shape and moving on space, their volume remains equal to 1 at all times.

For $\epsilon_1 = \epsilon_2 = 10^{-3}$, we used $n = 601$ spatial gridpoints and $s = 475201$ time gridpoints.

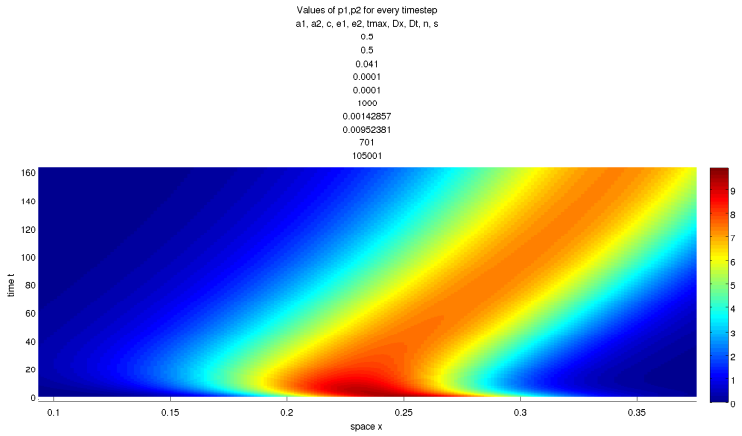
For $\epsilon_1 = \epsilon_2 = 10^{-4}$ **and** $tmax = 1000$



From the first figure here, we can see that, both ρ_1 and ρ_2, are morphologically changing. As in the previous case, both the curves are losing height while gain width. We observe that, in the current case, the curves do not lose that much height and do not gain that much width, as they did in the previous case. Here the curves reach height close to 7,2, until they stop changing their shape.

In the second figure, we can see that ρ_1 and ρ_2, share the same position and shape. This is taking place at around time $t = 1000$.

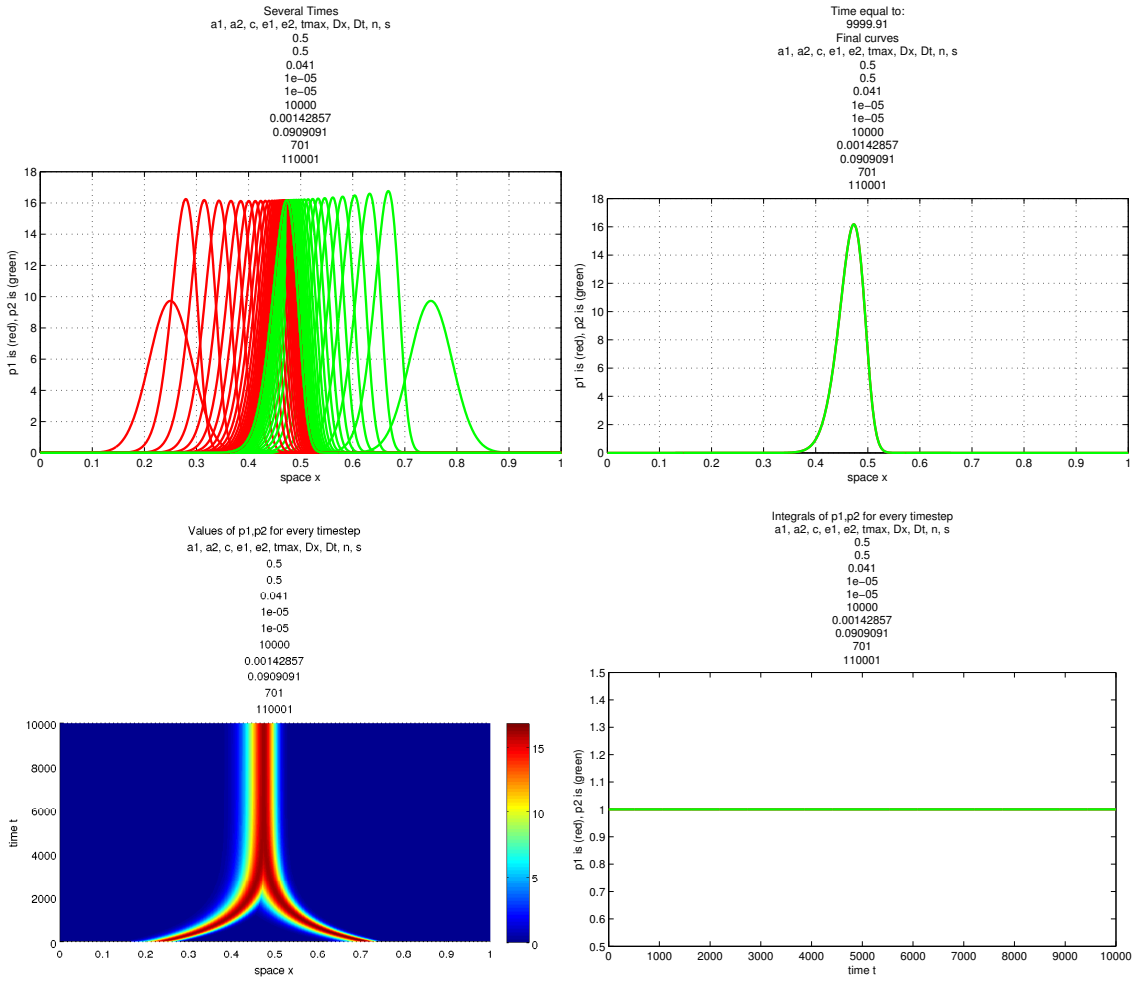
In the 3D figure, the phenomenon where ρ_1 is initially moving backwards until it change direction, is more obvious. The following figure, shows a zoom in the part that this phenomenon occurs.



The same phenomenon can be observed in the first figure also.

From the fourth figure, the volumes of both ρ_1, ρ_2, are always equal to 1.
In the current case $n = 701$ spatial gridpoints were used and $s = 105001$ time gridpoints.

For $\epsilon_1 = \epsilon_2 = 10^{-5}$ and $tmax = 10000$



The same properties hold also in this case, where $\epsilon_1 = \epsilon_2 = 10^{-5}$ are closer to 0. The curves are moving towards the same position, until as it is obvious from the second figure, they converge to equal curves. This is taking place at around time $t = 7000$.

We see that this time, the curves are gaining height and loosing width. Initially the height is close to 10 while a little time later, it reaches 16.

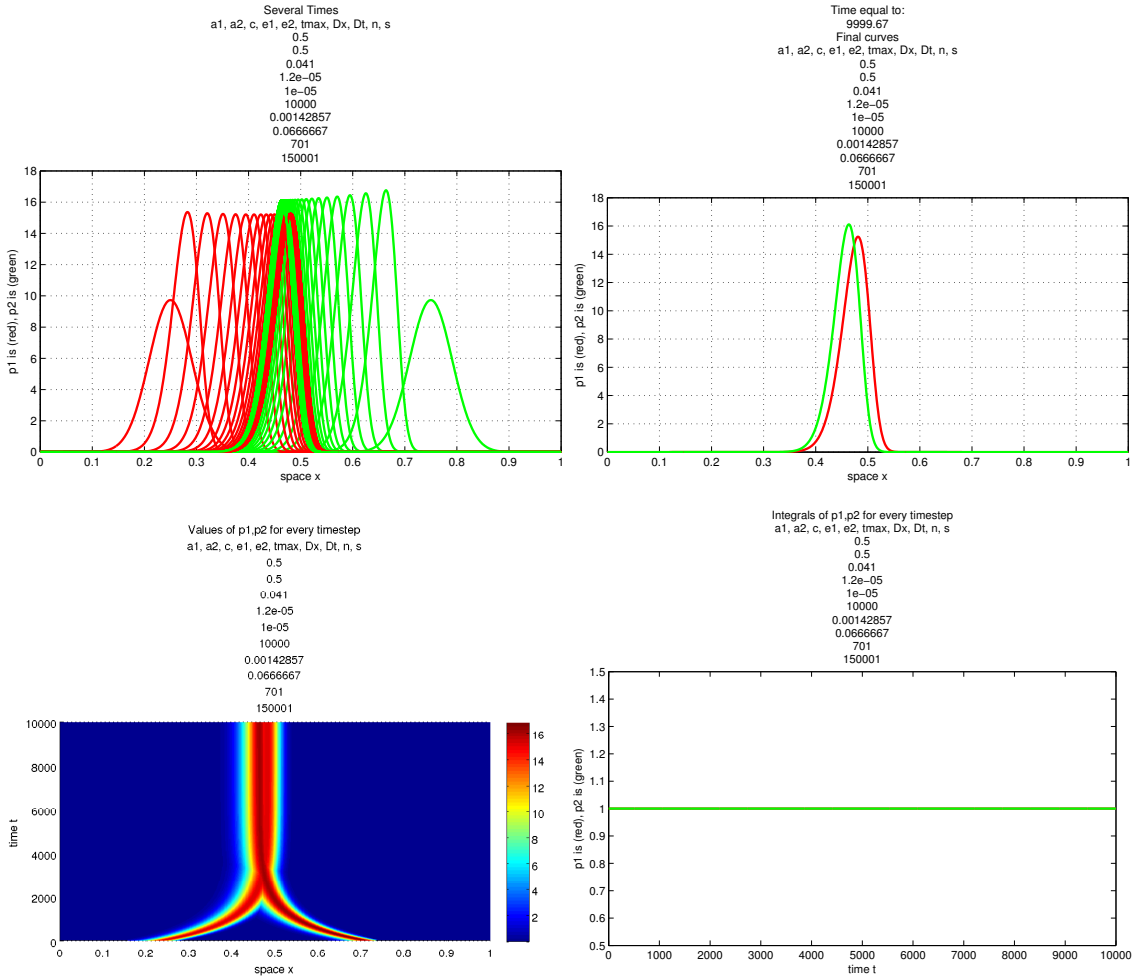
For the approximation of the model here, we used $n = 701$ spatial gridpoints and $s = 110001$ time gridpoints.

Some conclutions could be drawn from the results up until now. Firstly, for $\epsilon_1 = \epsilon_2$, the curves ρ_1 and ρ_2, are moving towards the same position, until some time point where they reach that position. Then, they become stable. As ϵ_1, ϵ_2 become smaller, the curves are moving slower, hence they reach the same position at a later time.

The curves are changing not only their position in space, but also their shape. We observed that, as ϵ_1, ϵ_2 become smaller, the curves become steeper. Here, we should also mention that, for $\epsilon_1 = \epsilon_2$, the morphological changes are the same for both the curves, even the height becomes the same for both ρ_1 and ρ_2 as time grows. Even though the curves are changing, their volume remains always equal to 1.

Now we continue with the cases where $\epsilon_1 \neq \epsilon_2$, in order to reach the second goal. So, we keep the rest of the parameters more or less fixed.

For $\epsilon_1 = 1,2 * 10^{-5}$, $\epsilon_2 = 10^{-5}$ and $tmax = 10000$



From the first figure here, we can see that, the general behaviour of the model is the same also for $\epsilon_1 \neq \epsilon_2$. That is, both curves are moving close to each other. Even more, apart from the moving behaviour, we also have the morphological change of the curves. As in the case where $\epsilon_1 = \epsilon_2 = 10^{-5}$, ρ_1 and ρ_2 become steeper as time grows.

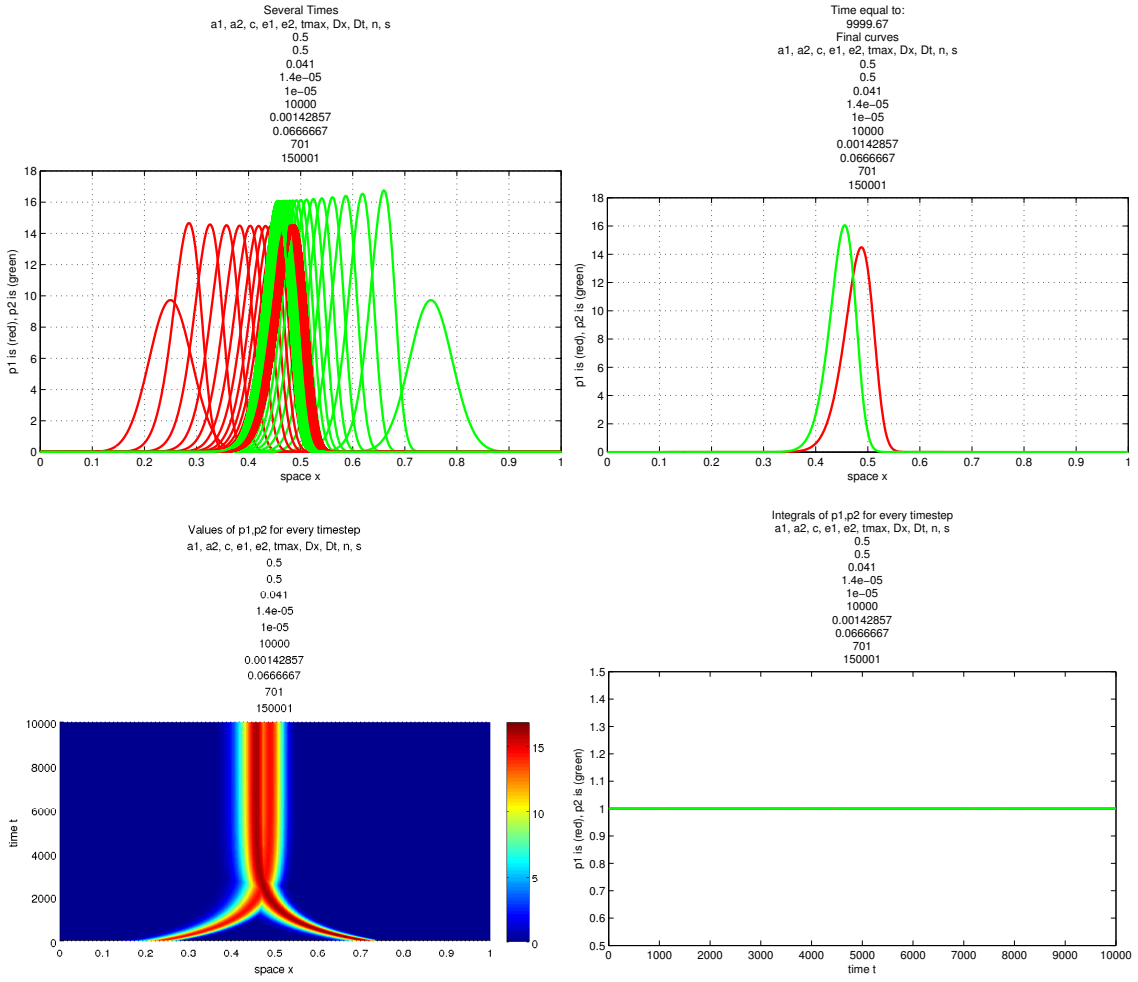
A main difference is that, here, ρ_1 does not gain the same height as ρ_2. We observe that, since ϵ_2 is closer to 0 than ϵ_1, ρ_2 is getting steeper than ρ_1. As in the previous cases the morphological change, is happening only at some early time interval. Afterwards, the curves are only moving in space.

By the 3D representation, we can see another main difference. Here, the X phenomenon is taking place. The curves are moving towards the same position, but when they have converged to a stationary solution, we see that they do not share the same position or shape. . The X phenomenon is taking place at around time $t = 3000$.

From the second figure, we can see the shape and the position of both ρ_1 and ρ_2 after they have become stable. Another thing that is more obvious here, is the skeweness of ρ_1.

Finally, even though the curves are changing in a different way, the volumes are always equal to 1.

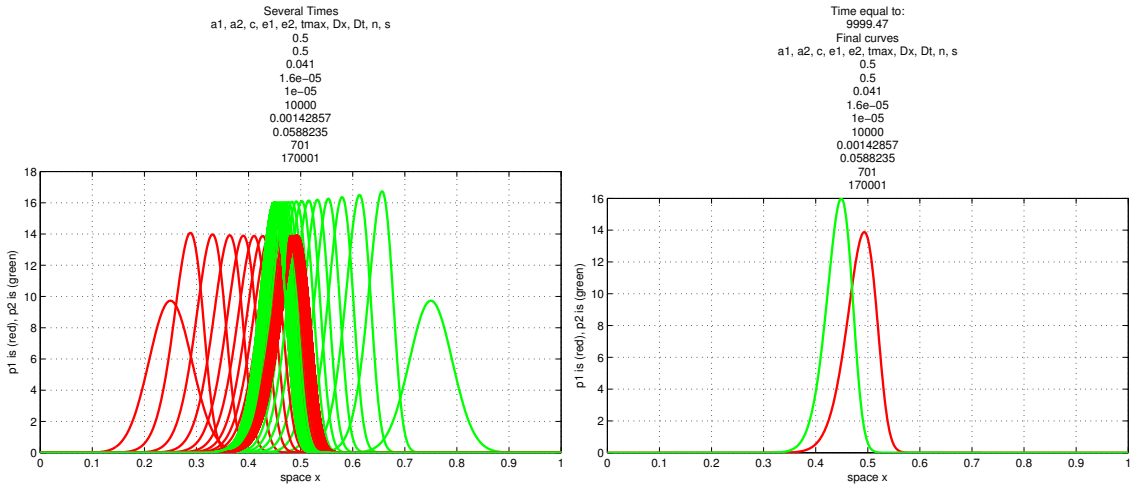
For $\epsilon_1 = 1,4 * 10^{-5}$**,** $\epsilon_2 = 10^{-5}$ **and** $tmax = 10000$

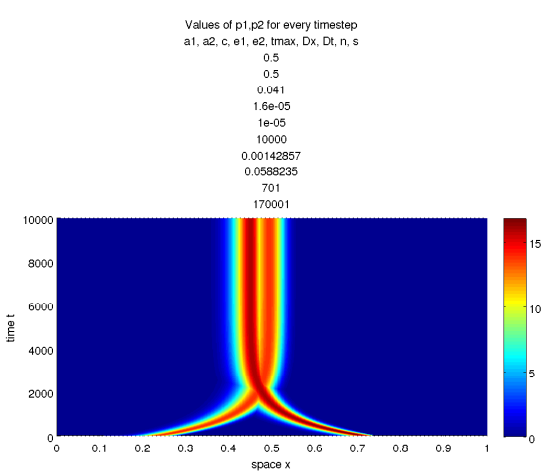


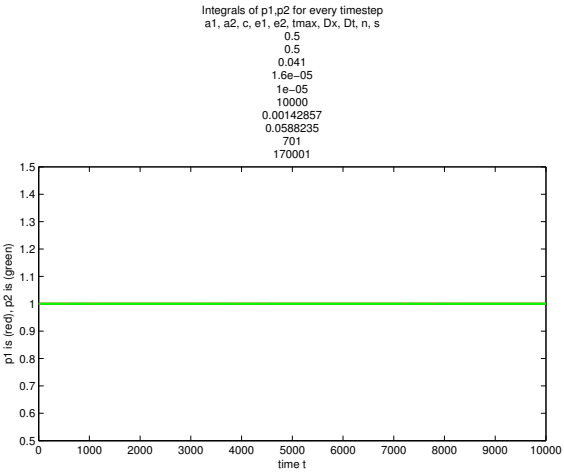
As we expected, since ϵ_1 is even larger than the previous case, ρ_1 is even shorter. On the other hand, since ϵ_2 has the same value as before, we see that ρ_2 is becoming as high as in the previous case.

By looking at the third figure, we see that for $\epsilon_1 = 1,4 * 10^{-5}$ and $\epsilon_2 = 10^{-5}$, the X phenomenon is taking place at around time $t = 2500$. Furthermore, after they have converged to a stationary solution, we see that the distance between their positions is larger compared to the previous case. The distance between the curves, is more obvious in the second figure.

For $\epsilon_1 = 1,6 * 10^{-5}$**,** $\epsilon_2 = 10^{-5}$ **and** $tmax = 10000$





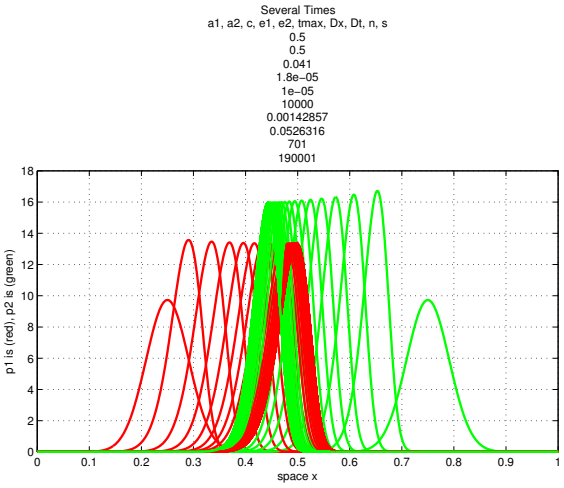


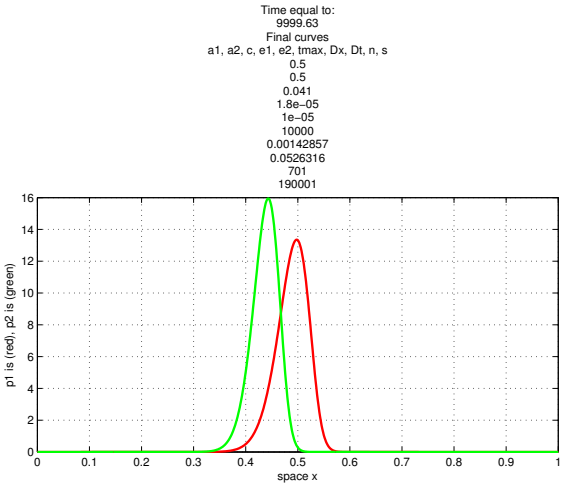
Here ρ_1, is shorter since ϵ_1 is greater than the previous cases, at the same time, ρ_2 is the same since ϵ_2 is not changed.

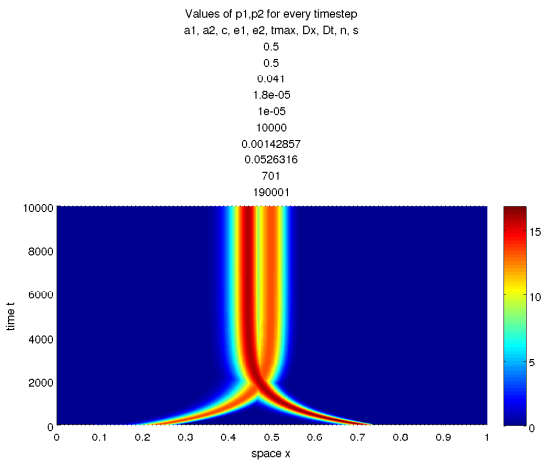
The X-phenomenon, happens even sooner than in any of the previous cases, at around time $t = 2000$. Furthermore, the distance between ρ_1 and ρ_2, is also greater than in all the previous cases.

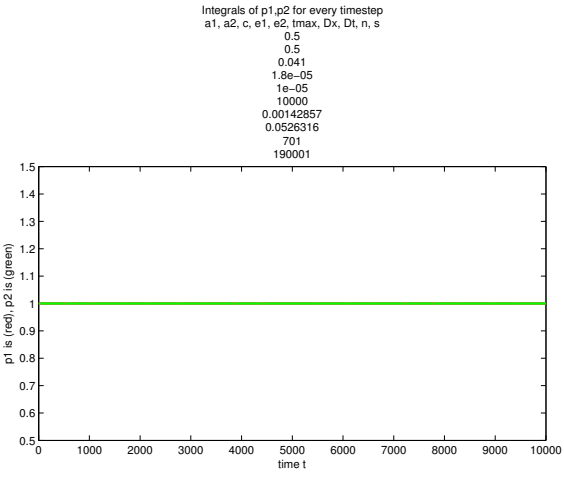
In the second figure, the skeweness of ρ_1 is observed.

For $\epsilon_1 = 1,8 * 10^{-5}$**,** $\epsilon_2 = 10^{-5}$ **and** $tmax = 10000$



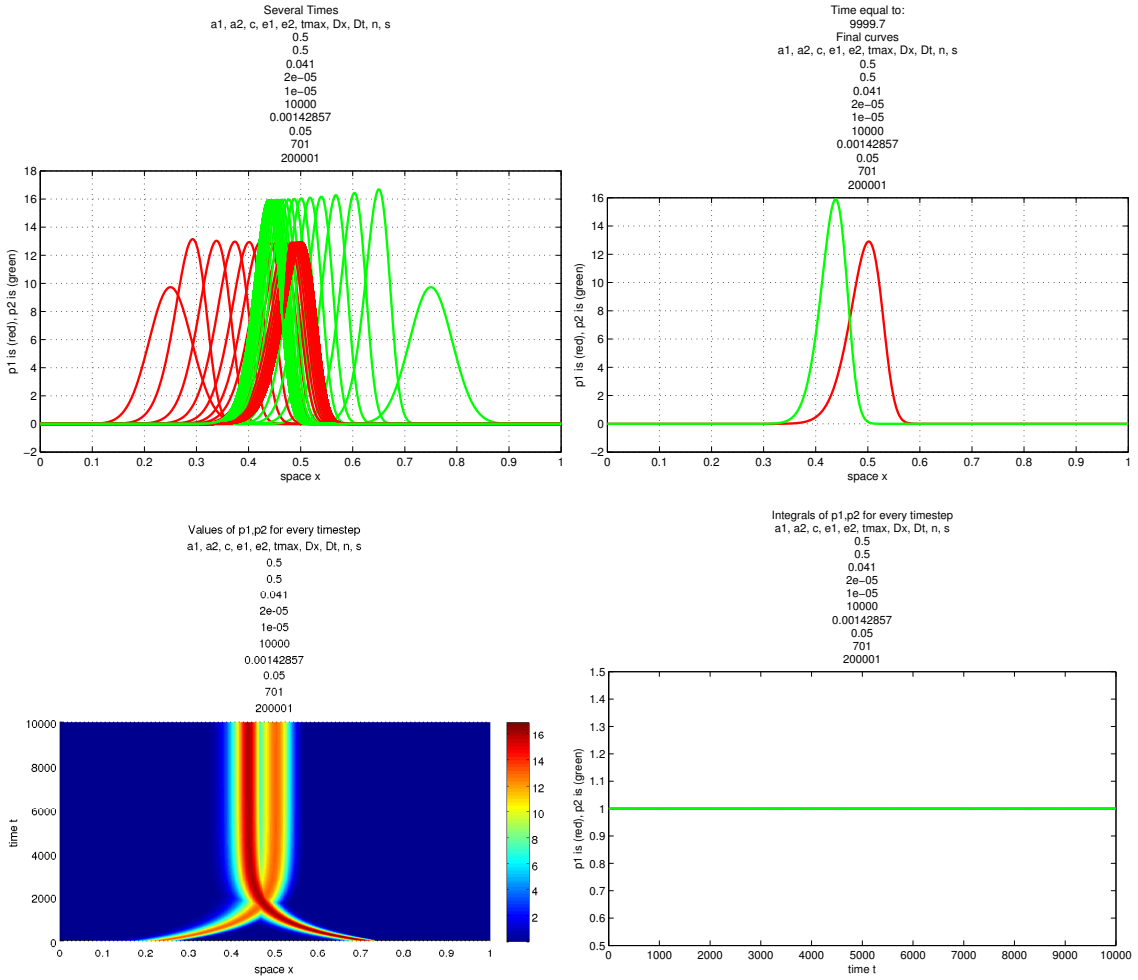






Same behaviour can be observed here as well.

For $\epsilon_1 = 2 * 10^{-5}$, $\epsilon_2 = 10^{-5}$ and $tmax = 10000$



For $\epsilon_1 = 2 * 10^{-5}$ and $\epsilon_2 = 10^{-5}$, we can see that, ρ_2 is gaining much more height compared to ρ_1. The X-phenomenon, is happening in a much earlier time and the distance between the curves, after they have become stable, is larger than all the previous cases.

By looking at the second figure, we can observe, the skewness of ρ_1.

To conclude, we gather all the results we can draw up until now for the case where $\epsilon_1 \neq \epsilon_2$. We have observed that, the height that each curve ρ_i is gaining as time grows, depends on the value of the corresponding ϵ_i. More precise, for a value ϵ_i closer to zero, we will have a higher curve ρ_i. At the same time, since the volume of the curves is equal to 1 for all times, a higher curve results also to a steeper one.

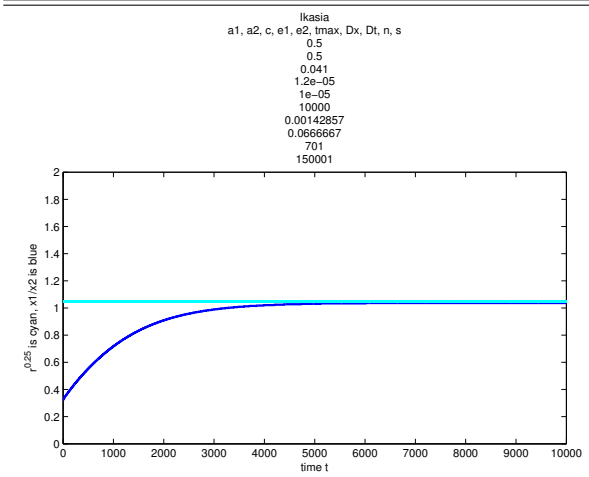
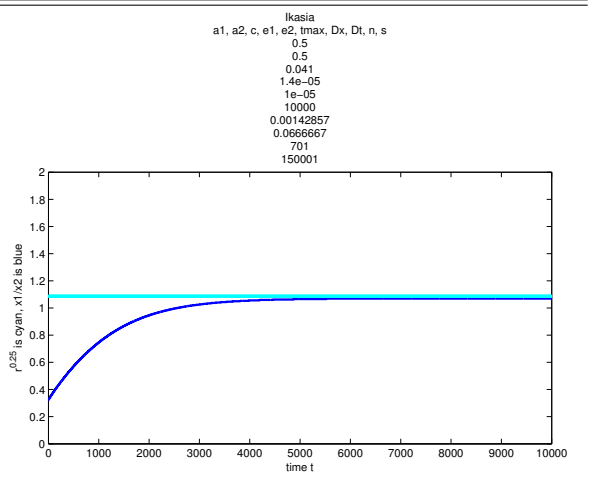
Another conclusion we can draw, has to do with the time that the X phenomenon is taking place. We have observed that, as the difference between ϵ_1 and ϵ_2 is growing, the X phenomenon is taking place at a sooner time level.

At last, the distance between the peaks of ρ_1 and ρ_2, after the curves have become stable, is getting larger as the difference between ϵ_1 and ϵ_2 is growing.

In order to study a little bit more the latter phenomenon, where at the last cases the distance between the curves ρ_1 and ρ_2 is growing with respect to the values of ϵ_1 and ϵ_2, we tried to answer the following question.

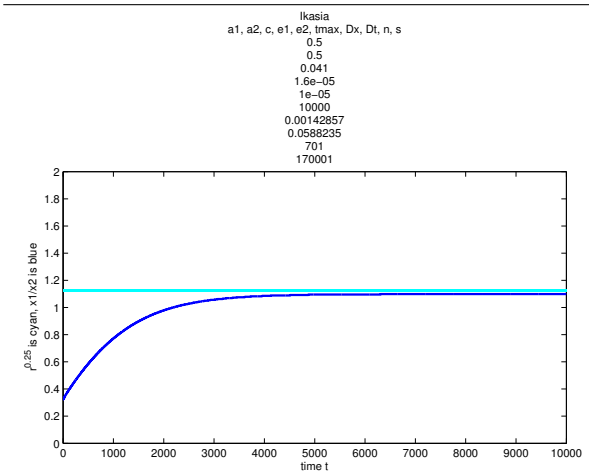
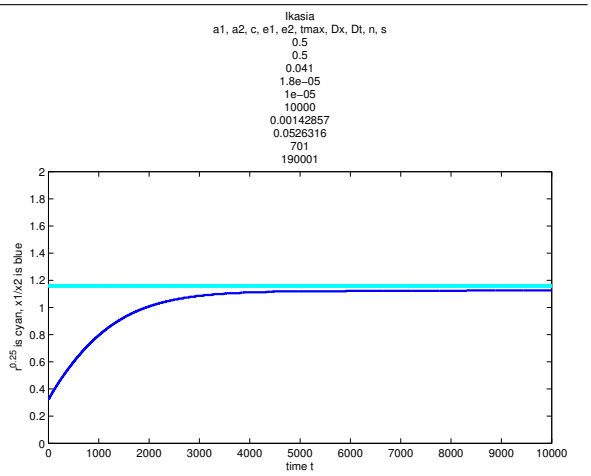
Let $x1$ be the location of the peak of ρ_1 and $x2$ the location of the peak of ρ_2, for every value of time t. We wanted to see if the sequence $\left(\frac{x1}{x2}\right)_t$, converges to $r^{0.25}$, where $r = \frac{\epsilon_1}{\epsilon_2}$, as time t increases.

In the following figures, the value $\frac{x1}{x2}$ corresponds to the blue line and the value $r^{0.25}$ corresponds to the cyan line.

For $\epsilon_1 = 1,2 * 10^{-5}$, $\epsilon_2 = 10^{-5}$	For $\epsilon_1 = 1,4 * 10^{-5}$, $\epsilon_2 = 10^{-5}$
	

For $\epsilon_1 = 1,2 * 10^{-5}$, $\epsilon_2 = 10^{-5}$, we have $r = \frac{\epsilon_1}{\epsilon_2} = 1,2$. The location of the peak of ρ_1, converges to $x1 = 0,48142857142871$ and of ρ_2, to $x2 = 0,464285714285714$.

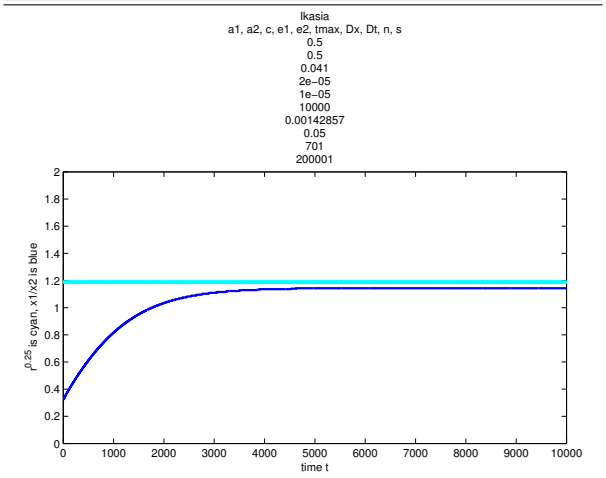
For $\epsilon_1 = 1,4 * 10^{-5}$, $\epsilon_2 = 10^{-5}$, we have $r = \frac{\epsilon_1}{\epsilon_2} = 1,4$. The location of the peak of ρ_1, converges to $x1 = 0,487142857142857$ and of ρ_2, to $x2 = 0,455714285714286$.

For $\epsilon_1 = 1,6 * 10^{-5}$, $\epsilon_2 = 10^{-5}$	For $\epsilon_1 = 1,8 * 10^{-5}$, $\epsilon_2 = 10^{-5}$
	

For $\epsilon_1 = 1,6 * 10^{-5}$, $\epsilon_2 = 10^{-5}$, we have $r = \frac{\epsilon_1}{\epsilon_2} = 1,6$. The location of the peak of ρ_1, converges to $x1 = 0,492857142857143$ and of ρ_2, to $x2 = 0,448571428571429$.

For $\epsilon_1 = 1,8 * 10^{-5}$, $\epsilon_2 = 10^{-5}$, we have $r = \frac{\epsilon_1}{\epsilon_2} = 1,8$. The location of the peak of ρ_1, converges to $x1 = 0,498571428571429$ and of ρ_2, to $x2 = 0,442857142857143$.

For $\epsilon_1 = 2 * 10^{-5}$, $\epsilon_2 = 10^{-5}$



For $\epsilon_1 = 2 * 10^{-5}$, $\epsilon_2 = 10^{-5}$, we have $r = \frac{\epsilon_1}{\epsilon_2} = 2$. The location of the peak of ρ_1, converges to $x1 = 0,501428571428571$ and of ρ_2, to $x2 = 0,438571428571429$.

We gather all the values in the following table.

r	$x1$	$x2$	$x1 - x2$	$x1/x2$	m
1,2	0,48142857142871	0,464285714285714	0,0171	1,0369	0,1989
1,4	0,487142857142857	0,455714285714286	0,0314	1,0690	0,1982
1,6	0,492857142857143	0,448571428571429	0,0443	1,0987	0.2003
1,8	0,498571428571429	0,442857142857143	0,0557	1,1258	0.2016
2	0,501428571428571	0,438571428571429	0,0629	1,1433	0,1932

where $r = \frac{\epsilon_1}{\epsilon_2}$, $x1$ is the peak of ρ_1 at the stable state of the solution, similarly $x2$ for ρ_2, the value $x1 - x2$ gives the distance of the peaks in each case and m comes from $\frac{x1}{x2} = r^m$.

The locations of the peaks of ρ_1 and ρ_2, are converging to some value since both the curves become stable after some time level. The location of the peaks, for each value of r, are given in the second and third column of the table.

From the fourth column, we can see something we have already noticed, the distance between the peaks of ρ_1 and ρ_2 is growing for larger values of r.

In the figures above, we can see the behaviour of the sequence $\left(\frac{x1}{x2}\right)_t$, as time grows. The values that the sequence converges for all r, is given in the fifth column of the table.

At last we wanted to see, if the sequence $\left(\frac{x1}{x2}\right)_t$, converges to $r^{0.25}$ as time grows. In the last column of the table, we give the value m, that is $\frac{x1}{x2} = r^m$, for all values of r. Unfortunatelly, m, does not seem to be equal to 0,25 as we were hoping, but is close to 0,2.

It would be important to see, if the sequence $(\frac{x1}{x2})_t$ converges to $r^{0.25}$, for ϵ_1, ϵ_2 closer to zero. But we observed that as ϵ_1, ϵ_2 are getting smaller, the curves are becoming steeper. That has the following consequence, in order for the result to have some accuracy, we will need to increase the number of spatial gridpoints. As we have already seen from the stability analysis, if we increase the number of spatial gridpoints by some factor, then we will also have to increase the number of time gridpoints several times more, in order for the scheme to be stable. Unfortunatelly, giving large values of gridpoints makes the scheme time consuming.

So, if we want to approximate the model for smaller ϵ_1, ϵ_2, we need to find a more efficient method.

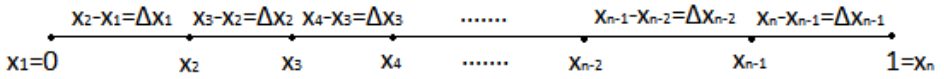
4 Adaptive moving grid method

We need a method that will give the best possible accuracy, by using only a small number of gridpoints. There are intervals in space or time, in which the curves have peaks, or even more, the curves are steep. In order to have a good approximation in these intervals, we need more gridpoints. At the same time there are intervals, where the curves have no peaks or their slope is not large. Hence, we can approximate the curves in these intervals by using less gridpoints.

So, we need a numerical approximation method, that can distribute both spatial and time gridpoints, in such a way that, a large amount of gridpoints is concentrated only to intervals that the curve is steep, so we can have the best possible accuracy, while less gridpoints are distributed, to approximate the curves, in intervals where the curves are less steep or their slope is almost zero.

Adaptive moving grid method, is the method that can do exactly that. The method, adapts the gridpoints in space and time, by distributing them in the way we explained. In every time step, the gridpoints are reallocated, under the same philosophy, by moving wherever they are necessary. The crusial decision, for the reallocation of the spatial gridpoints, is done by the so called monitor function.

In the adaptive moving grid method, the subdivision of the spatial and time domain, are not fixed anymore. The gridpoints are distributed in space, according to the monitor function. Suppose we have n spatial gridpoints. That yields, to a subdivision of the spatial interval $[0, 1]$, into N non equal subintervals, as in the following picture.



so, we get N subintervals of length $\Delta x_i = x_{i+1} - x_i$ for $i = 1, 2, \ldots, n-1$ and the spatial gridpoints are given by $x_i = \Delta x_1 + \Delta x_2 + \ldots + \Delta x_{i-1}$ for all $i = 1, 2, \ldots, n$ (where $n = N + 1$).

Suppose we also have s time gridpoints. The time domain is subdivided into S nonequal subintervals, in the same way as the spatial domain.

For every of these time gridpoints, we let the monitor function allocate the spatial gridpoints in the interval $[0, 1]$, then we calculate the value of both ρ_1 and ρ_2 for every spatial gridpoint.

4.1 Numerical approximation

From now on, we set our focus on the dimension of space.

In this section, we will derive the numerical approximation scheme, that will allow us calculate the values of $\rho_1(x_i)$ and $\rho_2(x_i)$, for any spatial gridpoint $x_i, i = 1, 2, \ldots, n$.

We follow the same format as in section §3.1. So, we start by deriving the numerical approximation formulas for the second order space derivatives $\frac{\partial^2 \rho_1}{\partial x^2}$, $\frac{\partial^2 \rho_2}{\partial x^2}$. Central difference scheme is also used here.

Central Difference Scheme:

Consider of a real function f. The Taylor expansion of f on some point x_{i+1} is

$$\begin{aligned} f(x_{i+1}) &\approx f(x_i) + \frac{1}{1!}\frac{\partial f}{\partial x}(x_i)(x_{i+1} - x_i) + \frac{1}{2!}\frac{\partial^2 f}{\partial x^2}(x_i)(x_{i+1} - x_i)^2 + \ldots \\ &= f(x_i) + \frac{\partial f}{\partial x}(x_i)\Delta x_i + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(x_i)(\Delta x_i)^2 + \ldots \end{aligned}$$

Similarly, the Taylor expansion of f on some point x_{i-1} is

$$\begin{aligned} f(x_{i-1}) &\approx f(x_i) + \frac{1}{1!}\frac{\partial f}{\partial x}(x_i)(x_{i-1} - x_i) + \frac{1}{2!}\frac{\partial^2 f}{\partial x^2}(x_i)(x_{i-1} - x_i)^2 + \ldots \\ &= f(x_i) + \frac{\partial f}{\partial x}(x_i)(-(x_i - x_{i-1})) + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(x_i)(-(x_i - x_{i-1}))^2 + \ldots \\ &= f(x_i) + \frac{\partial f}{\partial x}(x_i)(-\Delta x_{i-1}) + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(x_i)(-\Delta x_{i-1})^2 + \ldots \end{aligned}$$

According to the central difference scheme, we must have:

$$\begin{aligned}
\frac{\partial^2 f}{\partial x^2}(x_i) &\approx Af(x_{i+1}) + Bf(x_i) + Cf(x_{i-1}) \\
&= Af(x_i) + A\frac{\partial f}{\partial x}(x_i)\Delta x_i + A\frac{1}{2}\frac{\partial^2 f}{\partial x^2}(x_i)(\Delta x_i)^2 \\
&\quad + Bf(x_i) \\
&\quad + Cf(x_i) + C\frac{\partial f}{\partial x}(x_i)(-\Delta x_{i-1}) + C\frac{1}{2}\frac{\partial^2 f}{\partial x^2}(x_i)(-\Delta x_{i-1})^2 \\
&= (A+B+C)f(x_i) + (A\Delta x_i + 0 - C\Delta x_{i-1})\frac{\partial f}{\partial x}(x_i) \\
&\quad + (\frac{A}{2}(\Delta x_i)^2 + 0 + \frac{C}{2}(\Delta x_{i-1})^2)\frac{\partial^2 f}{\partial x^2}(x_i).
\end{aligned}$$

Since $\frac{\partial^2 f}{\partial x^2}(x_i) = 0 \cdot f(x_i) + 0 \cdot \frac{\partial f}{\partial x}(x_i) + 1 \cdot \frac{\partial^2 f}{\partial x^2}(x_i)$, we end up with the following system.

$$\begin{aligned}
A + B + C &= 0, \\
A\Delta x_i - C\Delta x_{i-1} &= 0, \\
\frac{A(\Delta x_i)^2}{2} + \frac{C(\Delta x_{i-1})^2}{2} &= 1.
\end{aligned}$$

Its solution, determines the values A, B and C.

$$\begin{aligned}
A &= \frac{2}{\Delta x_i(\Delta x_i + \Delta x_{i-1})}, \\
B &= -\frac{2}{\Delta x_i \Delta x_{i-1}}, \\
C &= \frac{2}{\Delta x_{i-1}(\Delta x_i + \Delta x_{i-1})}.
\end{aligned}$$

We plug in the values A, B, C, into the following equation, to get the numerical approximation formula, of the second spatial derivative of function f.

$$\begin{aligned}
\frac{\partial^2 f}{\partial x^2}\big|_{(x_i)} &\approx Af(x_{i+1}) + Bf(x_i) + Cf(x_{i-1}) \\
&= 2\frac{\Delta x_{i-1} f(x_{i+1}) - (\Delta x_i + \Delta x_{i-1})f(x_i) + \Delta x_i f(x_{i-1})}{\Delta x_i \Delta x_{i-1}(\Delta x_i + \Delta x_{i-1})}.
\end{aligned}$$

So, for $f = \rho_1$, we have:

$$\frac{\partial^2 \rho_1}{\partial x^2}\big|_{(x_i)} \approx 2\frac{\Delta x_{i-1}\rho_1(x_{i+1}) - (\Delta x_i + \Delta x_{i-1})\rho_1(x_i) + \Delta x_i \rho_1(x_{i-1})}{\Delta x_i \Delta x_{i-1}(\Delta x_i + \Delta x_{i-1})},$$

for every $i = 1, 2, \ldots, n$. Similarly, for $f = \rho_2$, we have:

$$\frac{\partial^2 \rho_2}{\partial x^2}\big|_{(x_i)} \approx 2\frac{\Delta x_{i-1}\rho_2(x_{i+1}) - (\Delta x_i + \Delta x_{i-1})\rho_2(x_i) + \Delta x_i \rho_2(x_{i-1})}{\Delta x_i \Delta x_{i-1}(\Delta x_i + \Delta x_{i-1})},$$

for every $i = 1, 2, \ldots, n$.

Next, we focus on the numerical approximation of the integrals $F_1(x) = \int_0^{1-x} \rho_2(y)\mathrm{d}y$, $F_2(x) = \int_0^{1-x} \rho_1(y)\mathrm{d}y$. We will use the trapezoidal rule for the approximation, as is defined below.

Trapezoidal Rule:

For integrating area $[a,b]=[x_1,x_n]$, we have:

$$\begin{aligned}
\int_a^b f(y)\mathrm{d}y &= \int_{x_1}^{x_n} f(y)\mathrm{d}y \\
&= \int_{x_1}^{x_2} f(y)\mathrm{d}y + \int_{x_2}^{x_3} f(y)\mathrm{d}y + \ldots + \int_{x_{n-2}}^{x_{n-1}} f(y)\mathrm{d}y + \int_{x_{n-1}}^{x_n} f(y)\mathrm{d}y \\
&\approx (x_2-x_1)\frac{f(x_1)+f(x_2)}{2} + (x_3-x_2)\frac{f(x_2)+f(x_3)}{2} + \ldots \\
&\quad \ldots + (x_{n-1}-x_{n-2})\frac{f(x_{n-2})+f(x_{n-1})}{2} + (x_n-x_{n-1})\frac{f(x_{n-1})+f(x_n)}{2} \\
&= \frac{\Delta x_1}{2}(f(x_1)+f(x_2)) + \frac{\Delta x_2}{2}(f(x_2)+f(x_3)) + \ldots \\
&\quad \ldots + \frac{\Delta x_{n-2}}{2}(f(x_{n-2})+f(x_{n-1})) + \frac{\Delta x_{n-1}}{2}(f(x_{n-1})+f(x_n)) \\
&= \frac{\Delta x_1}{2}f(x_1) + \frac{\Delta x_1+\Delta x_2}{2}f(x_2) + \frac{\Delta x_2+\Delta x_3}{2}f(x_3)\ldots \\
&\quad \ldots + \frac{\Delta x_{n-3}+\Delta x_{n-2}}{2}f(x_{n-2}) + \frac{\Delta x_{n-2}+\Delta x_{n-1}}{2}f(x_{n-1}) + \frac{\Delta x_{n-1}}{2}f(x_n) \\
&= \frac{\Delta x_1}{2}f(x_1) + \frac{\Delta x_{n-1}}{2}f(x_n) + \frac{1}{2}\sum_{q=2}^{n-1}(\Delta x_{q-1}+\Delta x_q)f(x_q) \\
&= \frac{1}{2}\left[\Delta x_1 f(x_1) + \Delta x_{n-1}f(x_n) + \sum_{q=2}^{n-1}(\Delta x_{q-1}+\Delta x_q)f(x_q)\right].
\end{aligned}$$

This is the definition of the trapezoidal rule, for nonequal space subintervals. The formulas that will be derived later on, are based on the latter definition. For integrating area $[a,x_i]$, we have:

$$\begin{aligned}
\int_a^{x_i} f(y)\mathrm{d}y &= \int_{x_1}^{x_i} f(y)\mathrm{d}y \\
&\approx \frac{1}{2}\left[\Delta x_1 f(x_1) + \Delta x_{i-1}f(x_i) + \sum_{q=2}^{i-1}(\Delta x_{q-1}+\Delta x_q)f(x_q)\right].
\end{aligned}$$

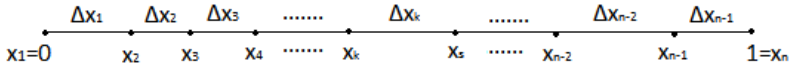
For integrating area $[a,b-x_i]$, we have:

$$\int_a^{b-x_i} f(y)\mathrm{d}y = \int_{x_1}^{b-x_i} f(y)\mathrm{d}y$$

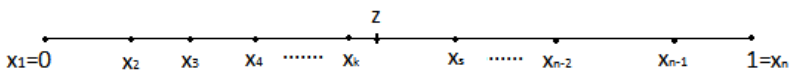
Note that we have $b-x_i = x_n - x_i = (\Delta x_1+\Delta x_2+\ldots+\Delta x_{n-1}) - (\Delta x_1+\Delta x_2+\ldots+\Delta x_{i-1})$ $= \Delta x_i + \Delta x_{i+1} + \ldots + \Delta x_{n-2} + \Delta x_{n-1}$.

In §3.1, we saw that, for the fixed grid method, $\forall i = 1,2,\ldots,n$, $b-x_i$ is always equal to some gridpoint x_j, $j=1,2,\ldots,n$. Unfortunately, this not the case in the adaptive moving grid method, since the spatial intervals are not of the same size.

Consider that the spatial gridpoints are distributed in the interval $[0,1]$, as in the following figure.



Let $z = b-x_i = \Delta x_i + \Delta x_{i+1} + \ldots + \Delta x_{n-2} + \Delta x_{n-1}$. In general, as we said, we will have $z \neq x_j$, $\forall j = 1,2,\ldots,n$, since Δx_i's are not equal. So, suppose that $z \in [x_k, x_s]$, where x_k, x_s are the closest gridpoints such that $x_k \leqslant z \leqslant x_s$.



The integral that we want to approximate, can be written as follows:

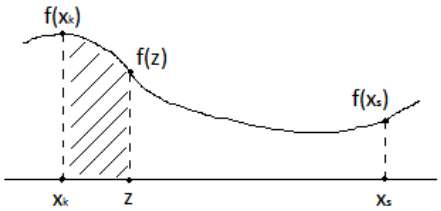
$$\int_{x_1}^{b-x_i} f(y)\mathrm{d}y \quad = \quad \int_{x_1}^{z} f(y)\mathrm{d}y = \int_{x_1}^{x_k} f(y)\mathrm{d}y + \int_{x_k}^{z} f(y)\mathrm{d}y,$$

the summation of two integrals.

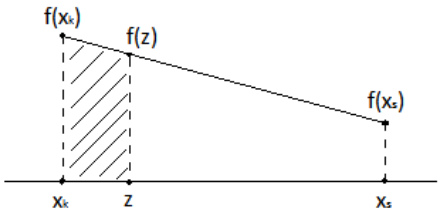
In order to calculate the first intergral, we can simply use the definition of the trapezoidal rule, which yields:

$$\int_{x_1}^{x_k} f(y)\mathrm{d}y = \frac{1}{2}\left[\Delta x_1 f(x_1) + \Delta x_{k-1} f(x_k) + \sum_{q=2}^{k-1}(\Delta x_{q-1} + \Delta x_q) f(x_q)\right].$$

On the other hand, in order to calculate the second integral, we need to calculate the value of f on point z. Then we will be able to use the trapezoidal rule. Suppose the next figure, shows the function f on the subinterval $[x_k, x_s]$,



since f is numerically approximated, then the approximation in the given interval, will look like in the figure below.



Generally, the numerical approximation of f, in the domain $[x_k, x_s]$, where x_k, x_s can be any successive spatial gridpoints, is either strictly increasing, or strictly decreasing or stable, with either some positive, or negative or equal to zero slope. So, the slope of f in the interval $[x_k, x_s]$, is the same for every point $z \in [x_k, x_s]$, since in that domain, f is strictly monotonic. Hence, the slope is

$$\frac{f(x_s) - f(x_k)}{x_s - x_k} = \frac{f(z) - f(x_k)}{z - x_k}, \forall z \in [x_k, x_s]$$

$$\Rightarrow f(z) = (z - x_k)\frac{f(x_s) - f(x_k)}{x_s - x_k} + f(x_k), \forall z \in [x_k, x_s],$$

where everything is known. The latter equation, gives the value of f on any point $z \in [x_k, x_s]$.

So, now we can approximate the second integral, by combining the trapezoidal rule and the latter

equation for $f(z)$. That is:

$$\begin{aligned}
\int_{x_k}^{z} f(y)\mathrm{d}y &= (z-x_k)\frac{f(z)+f(x_k)}{2} \\
&= \frac{(z-x_k)}{2}f(x_k)+\frac{z-x_k}{2}f(z) \\
&= \frac{(z-x_k)}{2}f(x_k)+\frac{z-x_k}{2}\left[(z-x_k)\frac{f(x_s)-f(x_k)}{x_s-x_k}+f(x_k)\right] \\
&= \frac{(z-x_k)}{2}f(x_k)+\frac{(z-x_k)^2}{2}\frac{f(x_s)-f(x_k)}{x_s-x_k}+\frac{(z-x_k)}{2}f(x_k) \\
&= (z-x_k)f(x_k)+\frac{(z-x_k)^2}{2(x_s-x_k)}(f(x_s)-f(x_k)) \\
&= (z-x_k)f(x_k)+\frac{(z-x_k)^2}{2\Delta x_k}(f(x_s)-f(x_k)).
\end{aligned}$$

To conclude, in order to calculate the integral $\int_a^{b-x_i} f(y)\mathrm{d}y$, we first have to compute the value $z=b-x_i$, then we find the closest successive gridpoints x_k, x_s, such that $x_k \leqslant z \leqslant x_s$. And finally, we have

$$\begin{aligned}
\int_a^{b-x_i} f(y)\mathrm{d}y &= \int_{x_1}^{z} f(y)\mathrm{d}y = \int_{x_1}^{x_k} f(y)\mathrm{d}y + \int_{x_k}^{z} f(y)\mathrm{d}y \\
&\approx \frac{1}{2}\left[\Delta x_1 f(x_1)+\Delta x_{k-1}f(x_k)+\sum_{q=2}^{k-1}(\Delta x_{q-1}+\Delta x_q)f(x_q)\right] \\
&\quad +(z-x_k)f(x_k)+\frac{(z-x_k)^2}{2\Delta x_k}(f(x_s)-f(x_k)).
\end{aligned}$$

Hence, the integral $F_1(x)=\int_0^{1-x}\rho_2(y)\mathrm{d}y$, can be numerically approximated, by the following procedure.

For a spatial gridpoint $x_i, i=1,2,\ldots,n$, we have to calculate

$$F_1(x_i)=\int_0^{1-x_i}\rho_2(y)\mathrm{d}y.$$

To do that, first, we calculate $z=b-x_i=1-x_i$, then we find the closest successive gridpoints x_k, x_s, such that $z\in[x_k,x_s]$. Finally, we use the following numerical formula.

$$\begin{aligned}
F_1(x_i)=\int_0^{1-x_i}\rho_2(y)\mathrm{d}y &\approx \frac{1}{2}\left[\Delta x_1\rho_2(x_1)+\Delta x_{k-1}\rho_2(x_k)+\sum_{q=2}^{k-1}(\Delta x_{q-1}+\Delta x_q)\rho_2(x_q)\right] \\
&\quad +(z-x_k)\rho_2(x_k)+\frac{(z-x_k)^2}{2\Delta x_k}(\rho_2(x_s)-\rho_2(x_k)).
\end{aligned}$$

Similarly, for $F_2(x)=\int_0^{1-x}\rho_1(y)\mathrm{d}y$, we have

$$F_2(x_i)=\int_0^{1-x_i}\rho_1(y)\mathrm{d}y.$$

First, we calculate $z=b-x_i=1-x_i$, then we find the closest successive gridpoints x_k, x_s, such that $z\in[x_k,x_s]$. Finally, we use the formula below.

$$\begin{aligned}
F_2(x_i)=\int_0^{1-x_i}\rho_1(y)\mathrm{d}y &\approx \frac{1}{2}\left[\Delta x_1\rho_1(x_1)+\Delta x_{k-1}\rho_1(x_k)+\sum_{q=2}^{k-1}(\Delta x_{q-1}+\Delta x_q)\rho_1(x_q)\right] \\
&\quad +(z-x_k)\rho_1(x_k)+\frac{(z-x_k)^2}{2\Delta x_k}(\rho_1(x_s)-\rho_1(x_k)).
\end{aligned}$$

The last part, that we need to approximate, are the defined integrals
$P_1 = \int_0^1 \rho_1(x)u_1(x)\left(\int_0^{1-x}\rho_2(y)\mathrm{d}y\right)\mathrm{d}x = \int_0^1 \rho_1(x)u_1(x)F_1(x)\mathrm{d}x$,
$P_2 = \int_0^1 \rho_2(x)u_2(x)\left(\int_0^{1-x}\rho_1(y)\mathrm{d}y\right)\mathrm{d}x = \int_0^1 \rho_2(x)u_2(x)F_2(x)\mathrm{d}x$.

Let $f_1(x) = \rho_1(x)u_1(x)F_1(x)$, then we have

$$P_1 = \int_0^1 \rho_1(x)u_1(x)F_1(x)\mathrm{d}x = \int_0^1 f_1(x)\mathrm{d}x = \int_{x_1}^{x_n} f_1(x)\mathrm{d}x$$

and by definition of the trapezoidal rule, we derive the following numerical formula

$$\begin{aligned}
&\approx \frac{1}{2}\left[\Delta x_1 f_1(x_1) + \Delta x_{n-1} f_1(x_n) + \sum_{q=2}^{n-1}(\Delta x_{q-1} + \Delta x_q) f_1(x_q)\right] \\
&= \frac{1}{2}\left[\Delta x_1 \rho_1(x_1)u_1(x_1)F_1(x_1) + \Delta x_{n-1}\rho_1(x_n)u_1(x_n)F_1(x_n)\right. \\
&\quad \left. + \sum_{q=2}^{n-1}(\Delta x_{q-1} + \Delta x_q)\rho_1(x_q)u_1(x_q)F_1(x_q)\right].
\end{aligned}$$

We can see that, in order to calculate the value P_1, we must first have calculated all the values $F_1(x_1), F_1(x_2), \ldots, F_1(x_n)$. The value P_1, will have to be calculated only once at the beginning of each time iteration.

Similarly, we derive the numerical formula for P_2.

$$\begin{aligned}
P_2 &= \int_0^1 \rho_2(x)u_2(x)F_2(x)\mathrm{d}x = \int_{x_1}^{x_n} \rho_2(x)u_2(x)F_2(x)\mathrm{d}x \\
&\approx \frac{1}{2}\left[\Delta x_1 \rho_2(x_1)u_2(x_1)F_2(x_1) + \Delta x_{n-1}\rho_2(x_n)u_2(x_n)F_2(x_n)\right. \\
&\quad \left. + \sum_{q=2}^{n-1}(\Delta x_{q-1} + \Delta x_q)\rho_2(x_q)u_2(x_q)F_2(x_q)\right]
\end{aligned}$$

As for P_1, we must first calculate all the values $F_2(x_1), F_2(x_2), \ldots, F_2(x_n)$, in order to be able to calculate P_2. The value P_2, will have to be calculated only once at the beginning of each time iteration.

Finally, we apply all formulas to the initial model. The initial model can be written as:

$$\begin{cases} \frac{\partial \rho_1}{\partial t} = \epsilon_1 \frac{\partial^2 \rho_1}{\partial x^2} + \rho_1(x)\left[u_1(x)F_1(x) - P_1\right], \\ \frac{\partial \rho_2}{\partial t} = \epsilon_2 \frac{\partial^2 \rho_2}{\partial x^2} + \rho_2(x)\left[u_2(x)F_2(x) - P_2\right] \end{cases}.$$

Consider the first equation of the latter model,

$$\frac{\partial \rho_1}{\partial t} = \epsilon_1 \frac{\partial^2 \rho_1}{\partial x^2} + \rho_1(x)\left[u_1(x)F_1(x) - P_1\right],$$

for some $x = x_i$, we have:

$$\begin{aligned}
\frac{\partial \rho_1}{\partial t}\Big|_{(x_i)} &= \epsilon_1 \frac{\partial^2 \rho_1}{\partial x^2}\Big|_{(x_i)} + \rho_1(x_i)\left[u_1(x_i)F_1(x_i) - P_1\right] \\
&= 2\epsilon_1 \frac{\Delta x_{i-1}\rho_1(x_{i+1}) - (\Delta x_i + \Delta x_{i-1})\rho_1(x_i) + \Delta x_i \rho_1(x_{i-1})}{\Delta x_i \Delta x_{i-1}(\Delta x_i + \Delta x_{i-1})} \\
&\quad + \rho_1(x_i)\left[u_1(x_i)F_1(x_i) - P_1\right],
\end{aligned}$$

where $z = 1 - x_i$ and x_k, x_s are the closest successive gridpoints such that $z \in [x_k, x_s]$,

$$\begin{aligned}
F_1(x_i) &= \frac{1}{2}\left[\Delta x_1 \rho_2(x_1) + \Delta x_{k-1}\rho_2(x_k) + \sum_{q=2}^{k-1}(\Delta x_{q-1} + \Delta x_q)\rho_2(x_q)\right] \\
&\quad + (z - x_k)\rho_2(x_k) + \frac{(z - x_k)^2}{2\Delta x_k}(\rho_2(x_s) - \rho_2(x_k)),
\end{aligned}$$

and

$$P_1 = \frac{1}{2}\left[\Delta x_1\rho_1(x_1)u_1(x_1)F_1(x_1) + \Delta x_{n-1}\rho_1(x_n)u_1(x_n)F_1(x_n) + \sum_{q=2}^{n-1}(\Delta x_{q-1} + \Delta x_q)\rho_1(x_q)u_1(x_q)F_1(x_q)\right].$$

Computationally, for each time gridpoint, we first need to compute $F_1(x_i)\ \forall i = 1, \ldots, n$ and save in a vector $F_1 = [F_1(x_1), F_1(x_2), \ldots, F_1(x_n)]$ with dimension $1 \times n$, and then we will be able to calculate P_1, which will be a single value.

Similarly, the second equation of the latter model,

$$\frac{\partial \rho_2}{\partial t} = \epsilon_2 \frac{\partial^2 \rho_2}{\partial x^2} + \rho_2(x)\left[u_2(x)F_2(x) - P_2\right],$$

for some $x = x_i$, takes the form:

$$\begin{aligned}\frac{\partial \rho_2}{\partial t}\Big|_{(x_i)} &= \epsilon_2 \frac{\partial^2 \rho_2}{\partial x^2}\Big|_{(x_i)} + \rho_2(x_i)\left[u_2(x_i)F_2(x_i) - P_2\right] \\ &= 2\epsilon_2 \frac{\Delta x_{i-1}\rho_2(x_{i+1}) - (\Delta x_i + \Delta x_{i-1})\rho_2(x_i) + \Delta x_i \rho_2(x_{i-1})}{\Delta x_i \Delta x_{i-1}(\Delta x_i + \Delta x_{i-1})} \\ &\quad + \rho_2(x_i)\left[u_2(x_i)F_2(x_i) - P_2\right],\end{aligned}$$

where $z = 1 - x_i$ and x_k, x_s are the closest successive gridpoints such that $z \in [x_k, x_s]$,

$$\begin{aligned}F_2(x_i) &= \frac{1}{2}\left[\Delta x_1\rho_1(x_1) + \Delta x_{k-1}\rho_1(x_k) + \sum_{q=2}^{k-1}(\Delta x_{q-1} + \Delta x_q)\rho_1(x_q)\right] \\ &\quad + (z - x_k)\rho_1(x_k) + \frac{(z - x_k)^2}{2\Delta x_k}(\rho_1(x_s) - \rho_1(x_k)),\end{aligned}$$

and

$$P_2 = \frac{1}{2}\left[\Delta x_1\rho_2(x_1)u_2(x_1)F_2(x_1) + \Delta x_{n-1}\rho_2(x_n)u_2(x_n)F_2(x_n) + \sum_{q=2}^{n-1}(\Delta x_{q-1} + \Delta x_q)\rho_2(x_q)u_2(x_q)F_2(x_q)\right].$$

Computationally, for each time gridpoint, we first need to compute $F_2(x_i)\ \forall i = 1, \ldots, n$ and save in a vector $F_2 = [F_2(x_1), F_2(x_2), \ldots, F_2(x_n)]$ with dimension $1 \times n$, and then we will be able to calculate P_2 which will be a single value.

The above scheme, is a fully discrete solution for all the gridpoints $x_i\ \forall i = 1, \ldots, n$.

Consider the dimention of time. The system is in a form that allow us to use the code *DASSL*. The code *DASSL*, is using backward difference schemes to numerically approximate the time derivatives of order from 1 to 5. In our case, the time derivatives are of order 1. Hence the code approximates them by using backward Euler scheme. That is

$$\begin{aligned}\frac{\partial \rho_1}{\partial t}(x, t_j) &= \frac{\rho_1(x, t_j) - \rho_1(x, t_{j-1})}{t_j - t_{j-1}}, \\ \frac{\partial \rho_2}{\partial t}(x, t_j) &= \frac{\rho_2(x, t_j) - \rho_2(x, t_{j-1})}{t_j - t_{j-1}}.\end{aligned}$$

Code *DASSL* is using a variation of Newton's method to solve the resulting system. For further information on code *DASSL* and the details on the methods it uses, look [13] and [14]

4.2 Gridpoint distribution

In general there are two kinds of adaptive techniques for time-dependent PDEs. The first one is the *h-refinement* technique, where new grids are added to places that are necessary, both in space and time dimentions, hence the number of gridpoints is changing with respect to our needs. The second one is the *r-refinement* technique, where the number of grids is constant and we distribute them with the best possible way. The method we used here, is an *r-refinement* technique.

Suppose we are given n spatial gridpoints. A "monitor function" M decides the way that these n gridpoints are distributed in space. The equidistribution principle is used, plus some extra concepts to make the movement of the gridpoints more smooth. More details can be found in [10]. In general in every time level, the spatial gridpoints are calculated with repect to the following formulas:

- Gridpoint x_1: $\dot{x}_1 = 0$.
- Gridpoint x_2: $\dot{x}_1 - 2\dot{x}_2 + \dot{x}_3 = 0$.
- Gridpoints x_i for any given $i = 3, 4, \ldots, n-2$:
  Let the coordinate transformation $x = x(\theta, \rho)$, $t = \theta$ with Jacobian $J = x_\rho$. In order to obtain a smooth grid distribution and smooth grid trajectories in the time direction, we let the tranformation $x = x(\theta, \rho)$ satisfy the following PDE:

  $$\tau[J_\theta M]_\rho + [S(\kappa)(J)M]_\rho = 0,$$

  where τ: temporal smoothing constant which is small enough to capture rapid solution changes in the time direction ($\approx 10^{-3}$).
  $S(\kappa)$: spatial smoothing operator and κ: spatial smoothing constant ($\kappa > 0$).
  M: the monitor function.
- Gridpoint x_{n-1}: $\dot{x}_{n-2} - 2\dot{x}_{n-1} + \dot{x}_n = 0$.
- Gridpoint x_n: $\dot{x}_n = 0$.

Note that the 1st and the last gridpoints, never change. That is $x_1 = 0$ and $x_n = 1$ for every time step. The second gridpoint x_2, is reallocated with respect to the gridpoints x_1, x_3. Similarly for the semilast gridpoint x_{n-1} which is reallocated with respect to the gridpoints x_{n-2}, x_n. This allows the gridpoints x_2 and x_{n-1} to move in more smooth manner. At last in order to reallocate the rest of the gridpoints, we evaluate the solution of the previous time level. Hence we are able to see where the approximated solution has some large slope and where not. Then the monitor function M decides how to distribute the gridpoints in a smooth way. For more details see [12] and [15].

Monitor functions and accuracy

As we have already mentioned before, the main concept of adaptive moving grid method, is the clever way to distribute the available gridpoints.

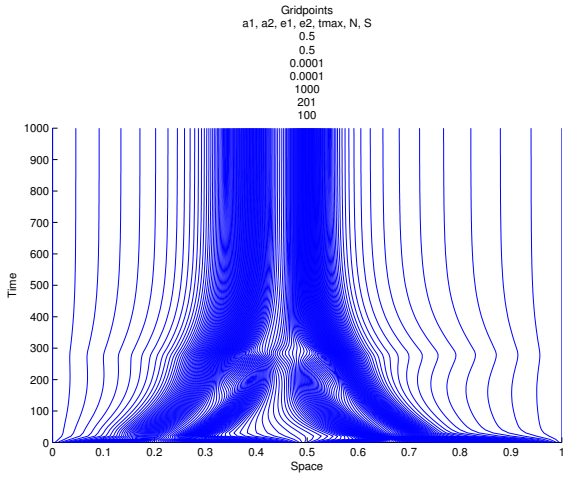
The distribution of the gridpoints, is done dynamically, in every time step, the gridpoints are distributed according to the current data. The most efficient way to do that, is to spend a larger number of gridpoints for intervals, where it is hard to accuratelly approximate the curves due to steepness. While, at the same time we need less gridpoints for places, where the slope of the curves is not large and hence, an accurate approximation, can be accomplished with a smaller number of gridpoints.

That crusial decision, is done by the monitor function. A backround theory can be found in [10] A variety of monitor functions can be found in [11]. In this work, we have used two different monitor functions.

The first one known as the arc-length monitor function, is given by:

$$M = \sqrt{1 + \left(\frac{\partial \rho_1}{\partial x} + \frac{\partial \rho_2}{\partial x}\right)^2}.$$

An example for the case where $a_1 = a_2 = 0,5$, $\epsilon_1 = \epsilon_2 = 10^{-4}$, is illustrated in the figure below.



There were used only $n = 201$ spatial gridpoints and $s = 100$ time gridpoints.

Initially, for time $t = 0$, the gridpoints are equedistributed, as in the fixed grid method. In the next time step, where time $t = \Delta t$, the monitor function is gathering most of the spatial gridpoints around the curves ρ_1 and ρ_2, leaving only a few spatial gridpoints at the intervals where $\rho_1 = \rho_2 = 0$.

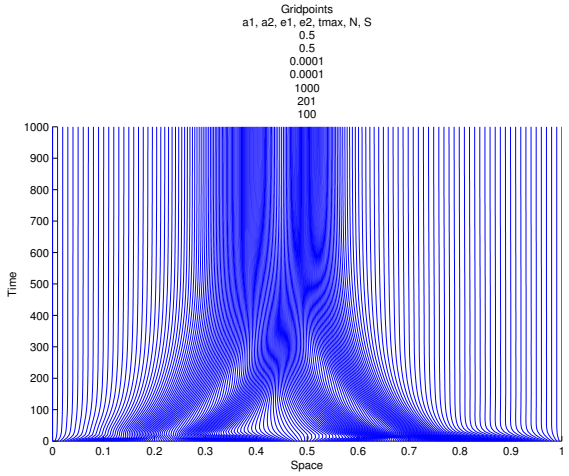
Further, as time grows, we have seen that, the curves are approaching each other, until they become equal, then they become stable. From the figure above, we can see that the gridpoints follow this behavior. So, for time $t > 600$, the position of both curves ρ_1, ρ_2, is almost the same and they are both nonzero, only at the neighborhood of $x = 0,45$. From the figure, it is obvious that almost all the spatial gripoints are gathered there.

Further informations for the monitor function can be found in references [9] and [11].

The second monitor function we used, is given by:

$$M = \int_0^1 \left(\left| \frac{\partial \rho_1}{\partial x} \right| + \left| \frac{\partial \rho_2}{\partial x} \right| \right) \mathrm{d}x + \left| \frac{\partial \rho_1}{\partial x} \right| + \left| \frac{\partial \rho_2}{\partial x} \right|.$$

We implement the latter monitor function, for the same example as for the first monitor function.



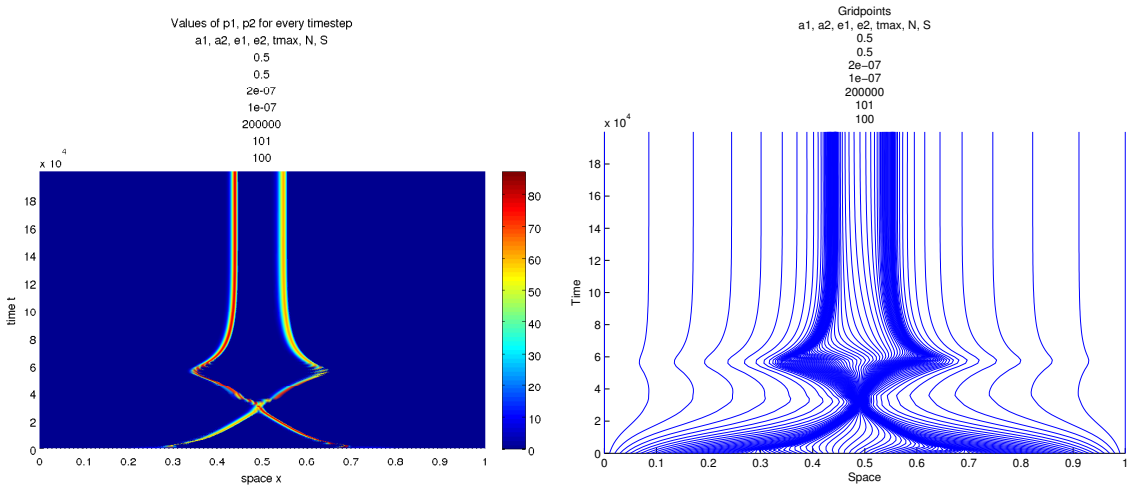
There were used $n = 201$ spatial gridpoints and $s = 100$ time gridpoints.

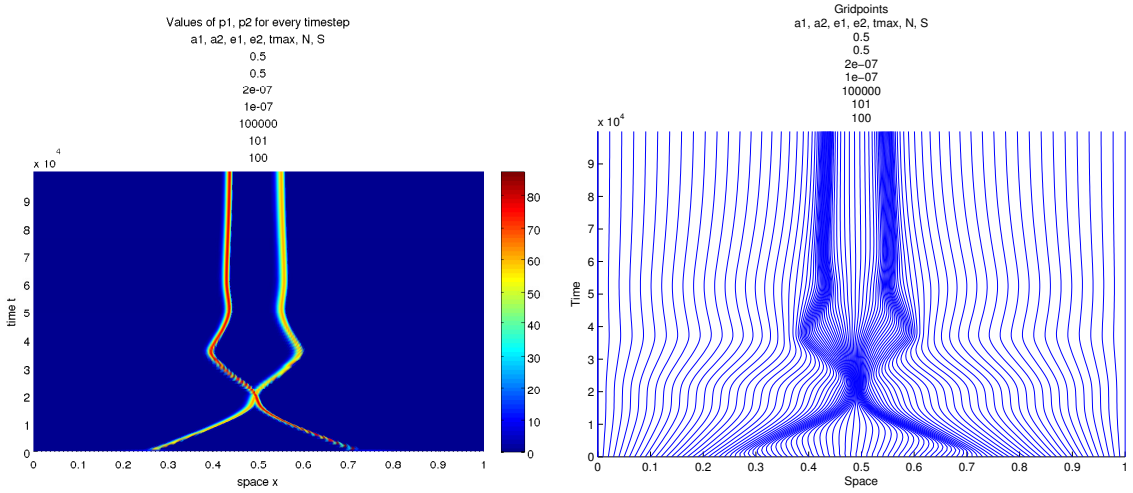
Further informations for the monitor function can be found in references [8] and [12].

We can see that the second monitor function, is also working in the same way. By comparing the two monitor functions, it is obvious that, the first one is distributing the spatial gridpoints in a much more efficient way, since it gathers almost all the spatial gridpoints, to the area that are needed. While the second monitor function, is gathering at the same area most of the gridpoints, yet leaves at the rest space also a lot gridpoints, which are not necessary since at the rest space ρ_1 and ρ_2 are equal to zero.

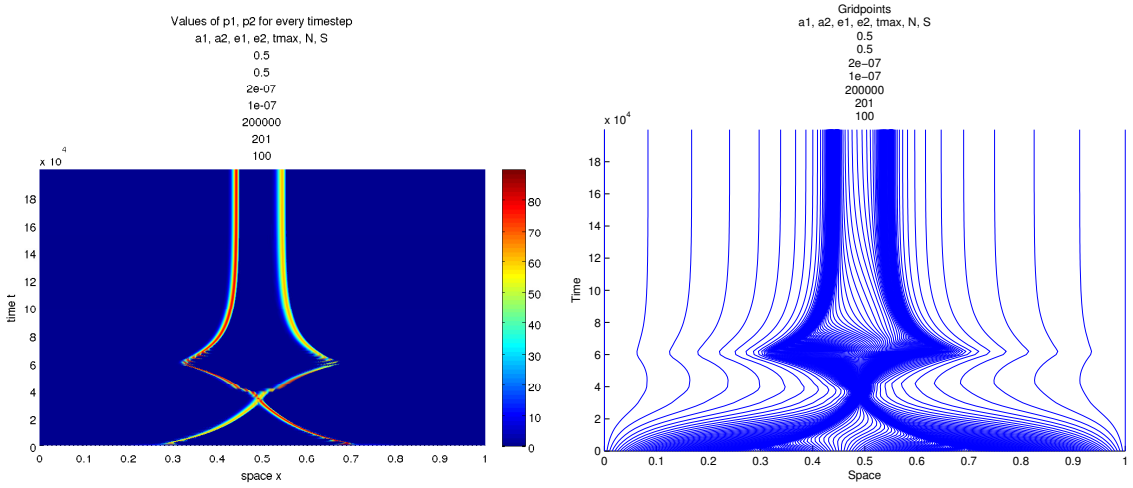
Unfortunatelly, we faced some accuracy problems, for both the monitor functions.

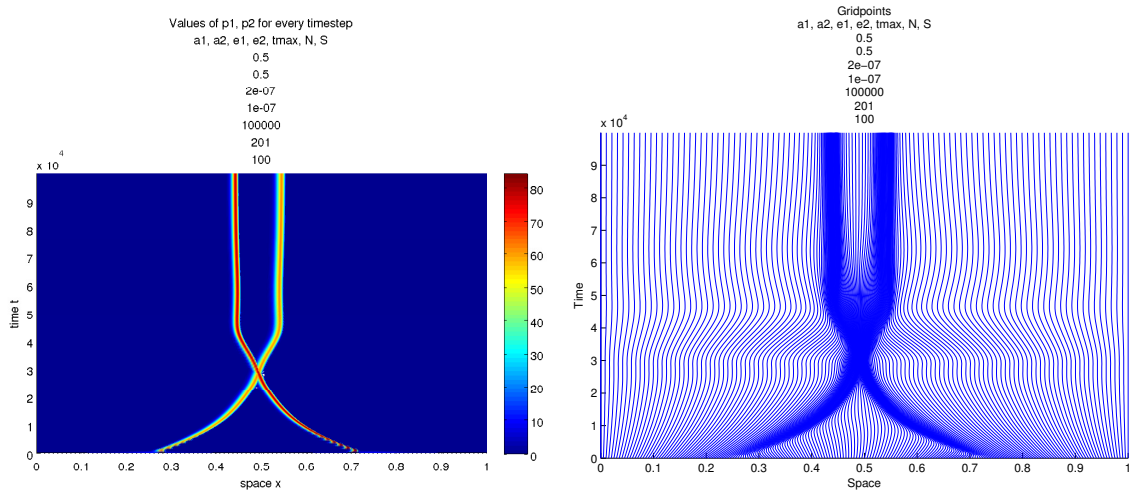
For $a_1 = a_2$, $\epsilon_1 = 2 \cdot 10^{-7}$ and $\epsilon_1 = 10^{-7}$, we used $n = 101$ spatial gridpoints and $s = 100$ time gridpoints for both monitor functions, and we got the following results respectivelly.





We can see that, both functions lead to some accuracy problems for $n = 101$. By increasing the number of spatial gridpoints to $n = 201$, we got:





From the first two figures, we can see that for $n = 201$ spatial gridpoints, the first monitor function gives the same accuracy problem. On the other hand, it seems that for $n = 201$, the second monitor function returns better results.

In the results that will be presented in the next section, we will use the first monitor function since it yields more accurate results. We will also use the second monitor function, which is also a good choise, only for the cases where the first monitor function, leads to problems similar to the one we discribed above.

For the dimension of time, suppose we have s time gridpoints. Code DASSL chooses the time step size Δt_j, $j = 1, 2, \ldots, s-1$ on every step according to the behaviour of the solution. So, suppose we have an approximation of the solution at the time level $t = t_j$ and we want to approximate the solution at the next time level $t = t_{j+1}$.

DASSL uses a "predictor polynomial" that interpolates the solution during the last time steps. By evaluating the predictor polynomial and its derivative at time level $t = t_{j+1}$, we get a guess for the solution and its derivatives at that time level.

The approximation to the solution at time level $t = t_{j+1}$ which is finally accepted by *DASSL* is the solution to the "corrector formula". The solution to the corrector formula, is the vector y_{j+1} such that the corrector polynomial and its derivative satisfy the differential algebraic equations of the model at time level $t = t_{j+1}$.

Now DASSL is able to see the rate of change of the solution between the two time levels, and decide if smaller or larger time step size is needed. For further details look [13].

4.3 Results

In this section, we use the numerical scheme we derived in section §4.1, to approximate the values of ρ_1 and ρ_2 of the initial model. Below, we present the results.

Since with the adaptive moving grid method, we can derive more accurate results than with the fixed grid method, we can see what happens as the values ϵ_1 and ϵ_2, are getting even closer to zero. So, as in section §3.3, we will investigate both the cases where $\epsilon_1 = \epsilon_2$ and $\epsilon_1 \neq \epsilon_2$, for smaller values of ϵ_1, ϵ_2. Further, we will also try to see what happens for different kind of initial values.

Hence, the results that are presented here, correspond to the following cases:

Initial values: Gaussian function for ρ_1 and Gaussian function for ρ_2.

Case	$a_1 = a_2$	ϵ_1	ϵ_2	$tmax$
1	$0,5$	10^{-5}	10^{-5}	10000
2	$0,5$	10^{-6}	10^{-6}	50000
3	$0,5$	10^{-7}	10^{-7}	150000
4	$0,5$	$1,2 * 10^{-7}$	10^{-7}	100000
5	$0,5$	$1,4 * 10^{-7}$	10^{-7}	100000
6	$0,5$	$1,6 * 10^{-7}$	10^{-7}	100000
7	$0,5$	$1,8 * 10^{-7}$	10^{-7}	100000
8	$0,5$	$2 * 10^{-7}$	10^{-7}	100000
9	$0,5$	0	0	500
10	1	10^{-6}	10^{-6}	50000

Initial values: Gaussian function for ρ_1 and sine function for ρ_2.

Case	$a_1 = a_2$	ϵ_1	ϵ_2	$tmax$
11	$0,5$	10^{-5}	10^{-5}	10000

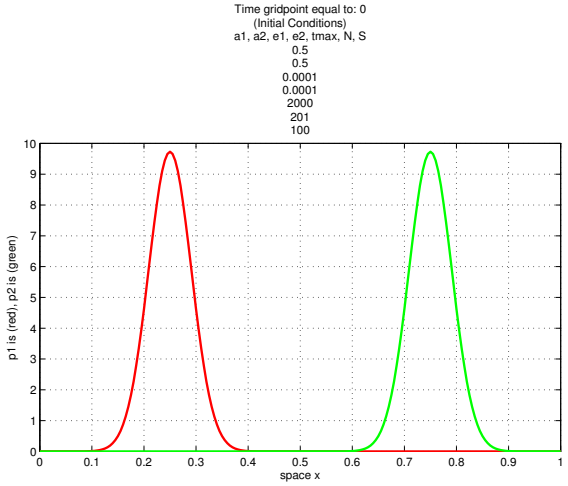
Initial values: sine function for ρ_1 and the same sine function for ρ_2.

Case	$a_1 = a_2$	ϵ_1	ϵ_2	$tmax$
12	$0,5$	10^{-5}	10^{-5}	10000

Initial values: sine function for ρ_1 and shifted sine function for ρ_2.

Case	$a_1 = a_2$	ϵ_1	ϵ_2	$tmax$
13	$0,5$	10^{-5}	10^{-5}	10000

For the first part of the results, we have Gaussian functions as initial values for both ρ_1 and ρ_2. The volume of the Gaussians is taken to be equal to 1. So, at time $t = 0$, the curves ρ_1 and ρ_2 are given in the following figure.

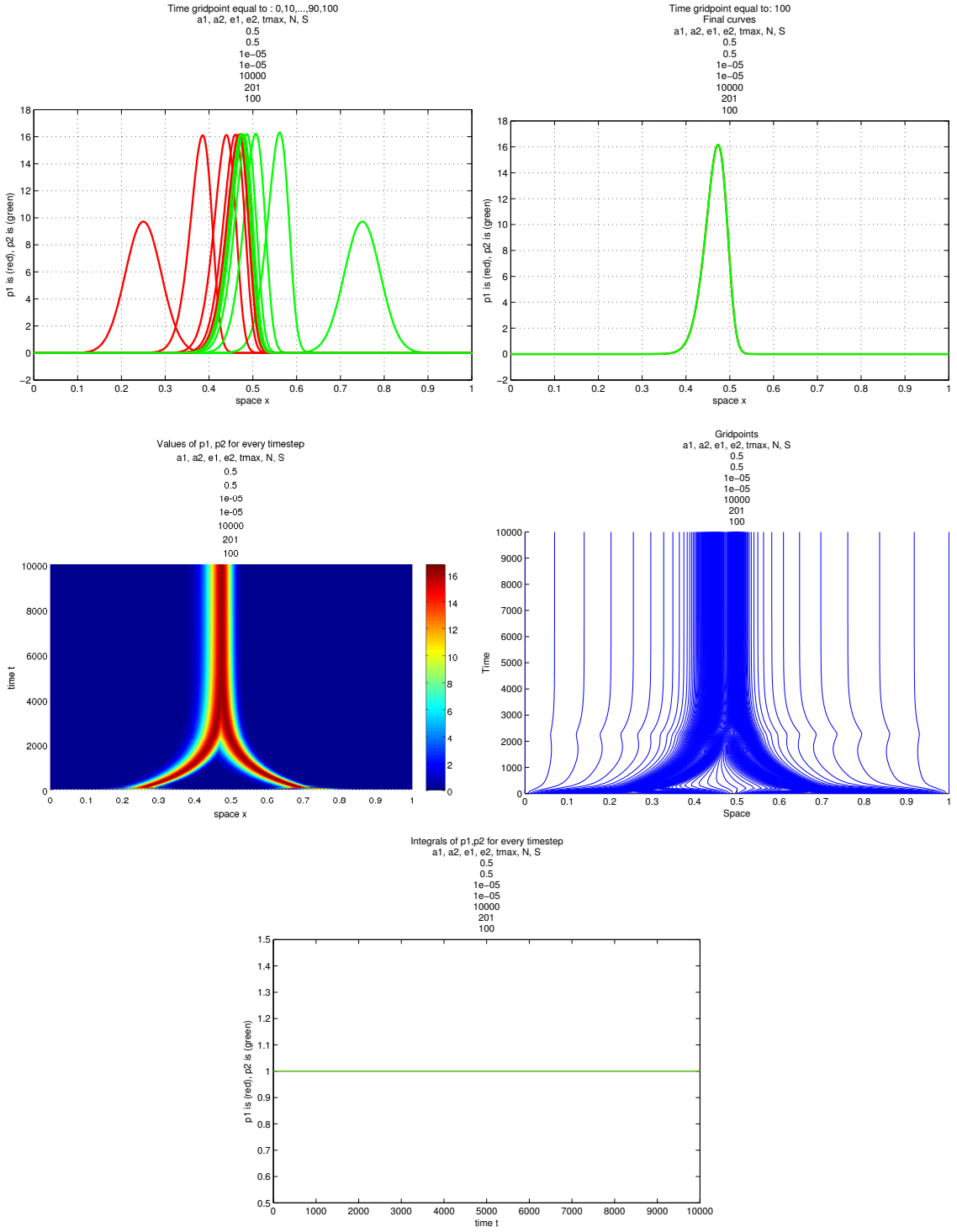


The curve ρ_1 (red), has its peak on $x = 1/4$, while ρ_2 (green), has its peak on $x = 3/4$.

In every case, the first figure, gives the approximation of the model for some intermediate times, while the second figure, gives the approximation of the model at time $t = tmax$. Furthermore, the third figure gives a 3D representation of the model for $0 \leq x \leq 1$ and $0 \leq t \leq tmax$. The fourth figure, shows how the spatial gridpoints are distributed in the interval $[0, 1]$, for every time step. At last, the fifth figure, gives the volume of both ρ_1 and ρ_2, for every time $0 \leq t \leq tmax$. That is done by integrating ρ_1 and ρ_2 on the interval $[0, 1]$. Trapezoidal rule was used for the integration.

For $a_1 = a_2 = 0,5$**,** $\epsilon_1 = \epsilon_2 = 10^{-5}$ **and** $tmax = 10000$

The same case is also given in section §3.3 for the fixed grid method. We quote this case also here, in order to compare the results of the two different methods.

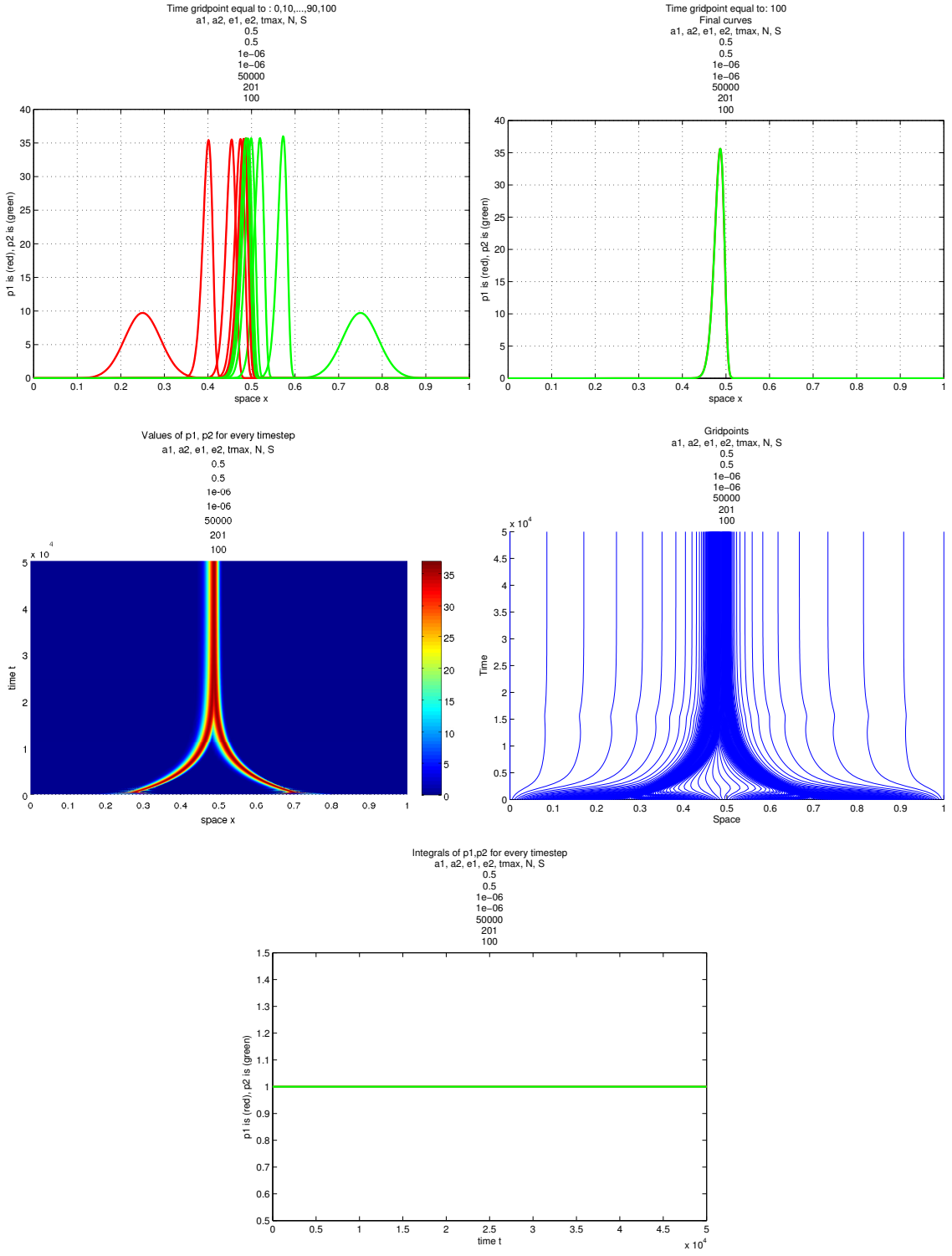


By comparing the figures, above with the corresponding ones in section §3.3, we can see that the results are exactly the same, as they should be. Hence, the model is well approximted also with the adaptive moving grid method.

We mention that for the fixed grid method, in order to get the same results, we used $n = 701$ spatial gridpoints and $s = 110001$ time gridpoints, while for the adaptive moving grid method, we used only $n = 201$ and $s = 100$.

We continue with cases where ϵ_1, ϵ_2, are smaller.

For $a_1 = a_2 = 0,5$**,** $\epsilon_1 = \epsilon_2 = 10^{-6}$ **and** $tmax = 50000$



From the first figure, it is obvious that both curves are moving towards the same position, until they converge to equal curves and become stable ever since. In the second figure, we can clearly see the position where the curves have converged to.

Morphologically, the curves are becoming very high and thin. So, initially for time $t = 0$, we have two Gaussians with height almost equal to 10, and after some timesteps, both ρ_1 and ρ_2, have reached height 35.

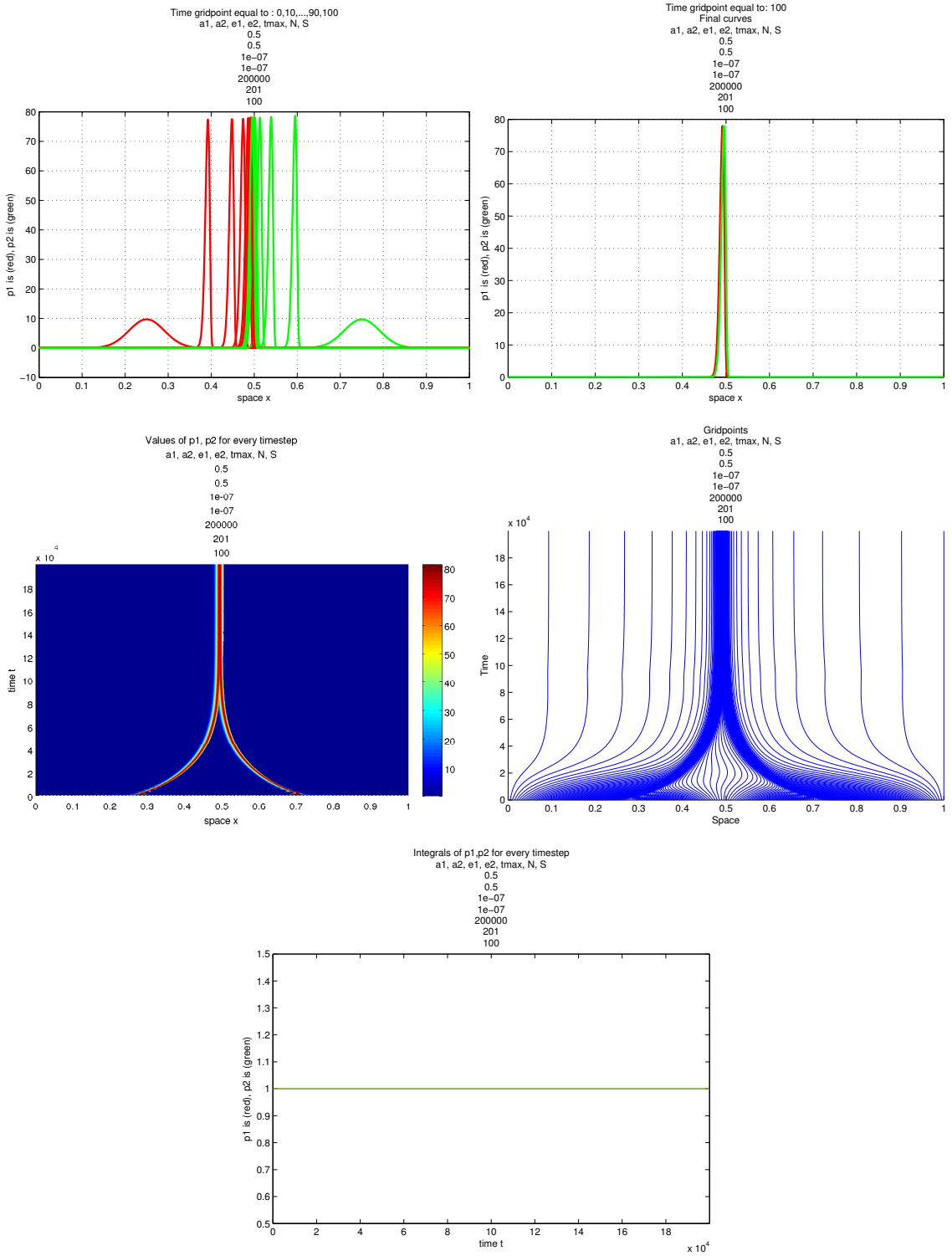
We also have to observe that in the second figure, both ρ_1 and ρ_2 are skewed.

From the 3D representation of the curves, we can see that the curves become equal at around time $t = 30000$. Eversince, the curves remain stable.

The gridpoints are moving along with the curves as time grows, as we can observe in the fourth figure, while in the last figure, we can see that the volume of both the curves remains equal to 1 at all times.

For $\epsilon_1 = \epsilon_2 = 10^{-6}$, we used $n = 201$ spatial gridpoints and $s = 100$ time gridpoints.

For $a_1 = a_2 = 0,5$, $\epsilon_1 = \epsilon_2 = 10^{-7}$ and $tmax = 200000$



As we expected, for $\epsilon_1 = \epsilon_2 = 10^{-7}$, the curves are becoming even higher. They reach height almost equal to 80. From the 3D representation, we observe that the curves are moving slower, hence they reach

the stationary position at a later time compared to the previous case, at around time $t = 130000$.

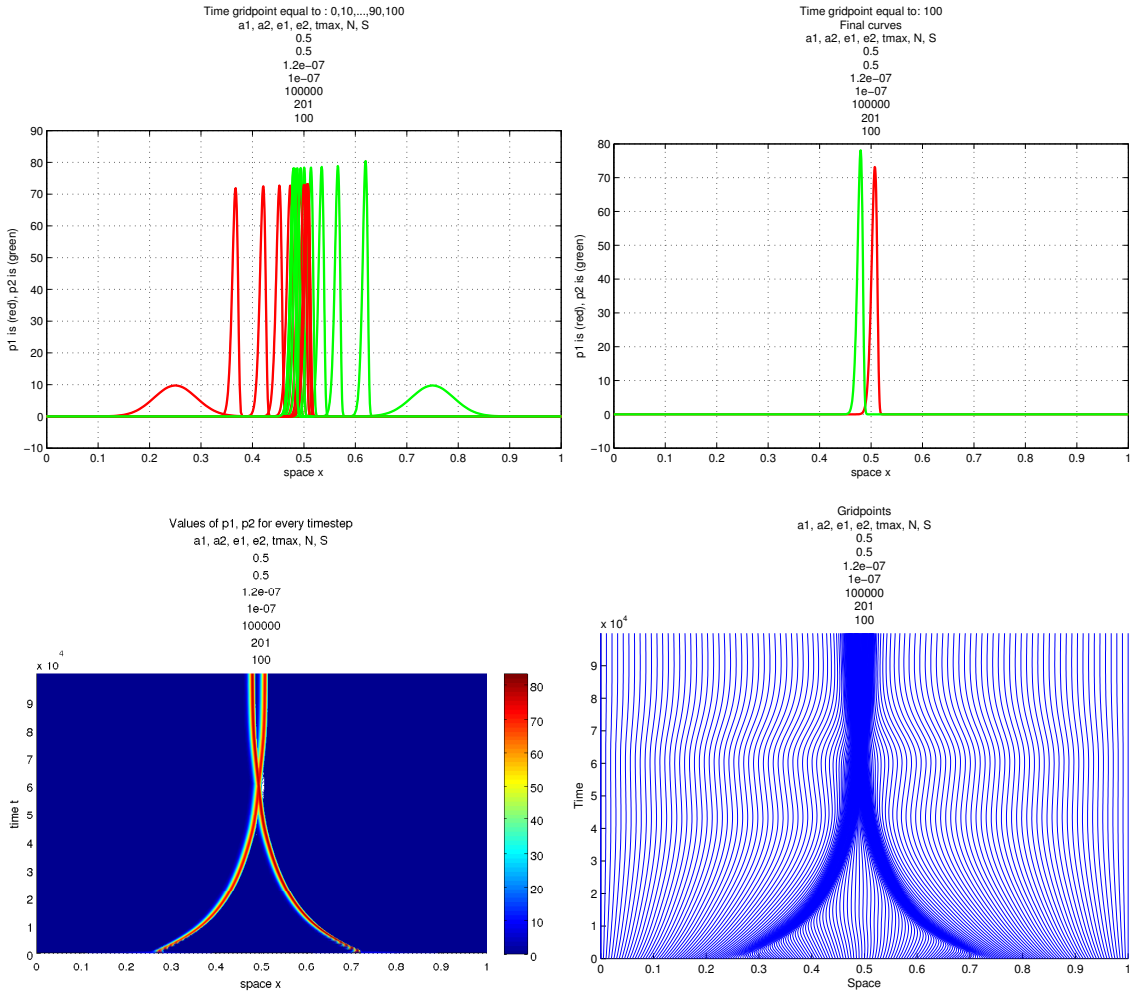
From the fourth figure, we can see that, initially the spatial gridpoints, are concentrating around the curves and move along with them. At time $t = 130000$, the curves have converged. Ever since, the curves are nonzero only in the interval $[0.45, 0.55]$. We can see that, the gridpoints are also concentrated to that interval. So, we have that out of 201 gridpoints that were used, about 183 of them are concentrated in the interval $[0.45, 0.55]$, while only about 18 spatial gridpoints are used to cover the rest of the space, where $\rho_1 = \rho_2 = 0$. Here, we have to mention that, in the fixed grid method, if we used $n = 701$ spatial gridpoints, only about 70 of them would be distributed in the interval $[0.45, 0.55]$, while about 631 gridpoints would be spend to cover the rest space where $\rho_1 = \rho_2 = 0$. In other words, in order to get the same accuracy with fixed grid method, we should use about $n = 1830$ spatial gridpoints, which by the stability analysis in section §3.2, yields that we should also use at least $s = 140000$ time gridpoints.

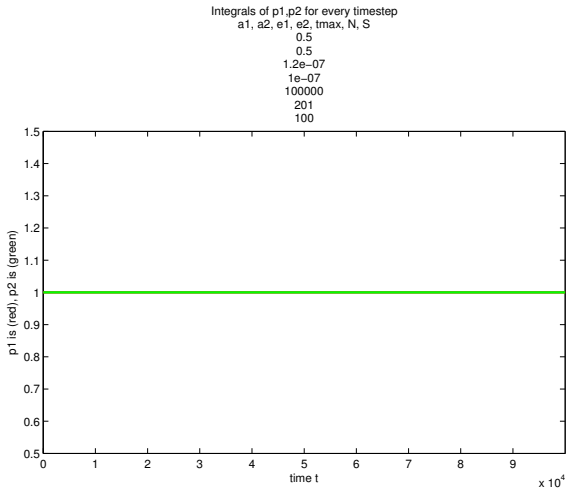
From the last figure, the volumes of both ρ_1, ρ_2 is always equal to 1.

Hence, up until now, we cement the results we had derived in section §3.3 about the behaviour of the model for $\epsilon_1 = \epsilon_2$ as ϵ_1, ϵ_2 are getting closer to zero.

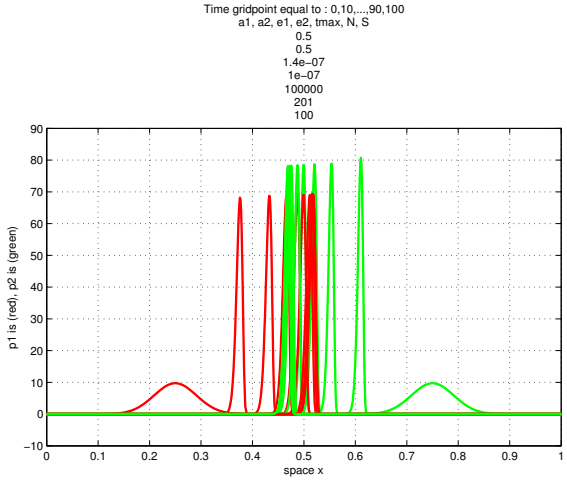
Now we continue with the cases where $\epsilon_1 \neq \epsilon_2$.

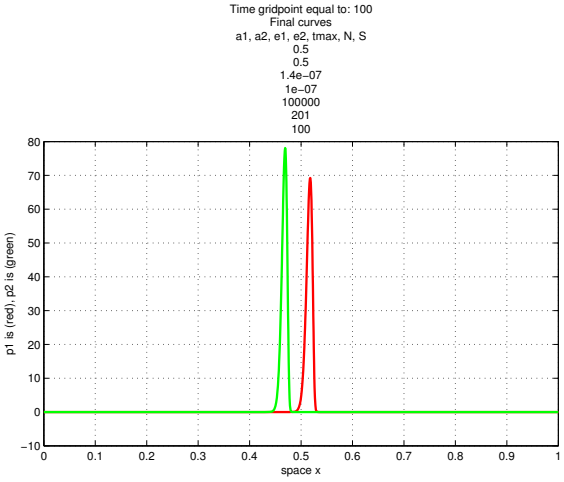
For $a_1 = a_2 = 0,5$**,** $\epsilon_1 = 1,2 * 10^{-7}$**,** $\epsilon_2 = 10^{-7}$ **and** $tmax = 100000$

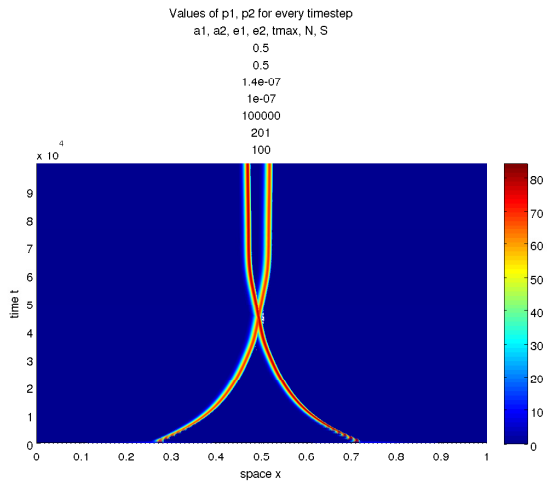


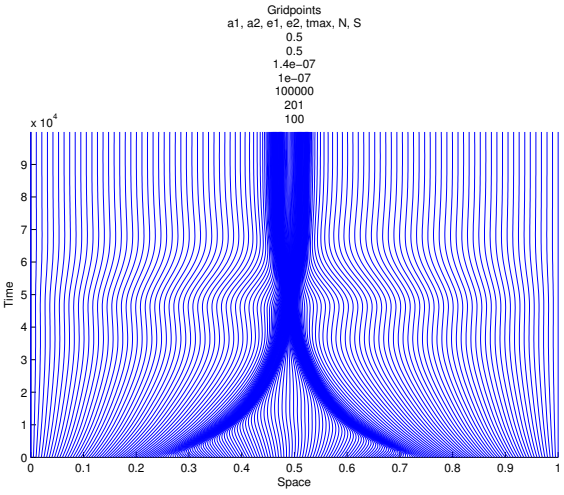


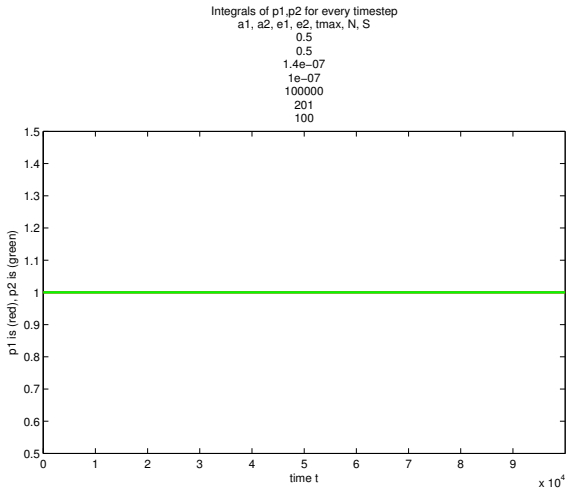
For $a_1 = a_2 = 0,5$**,** $\epsilon_1 = 1,4 * 10^{-7}$**,** $\epsilon_2 = 10^{-7}$ **and** $tmax = 100000$



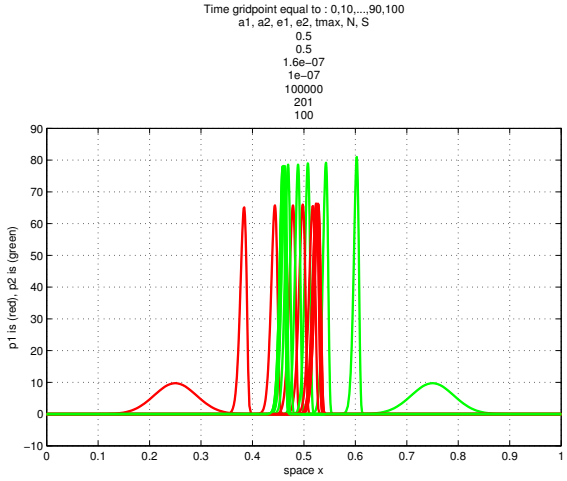


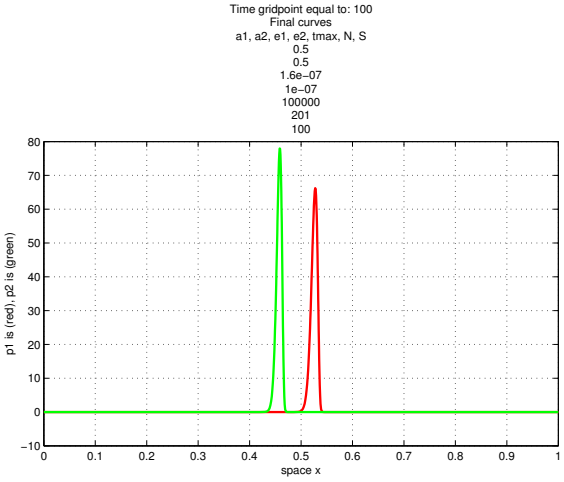


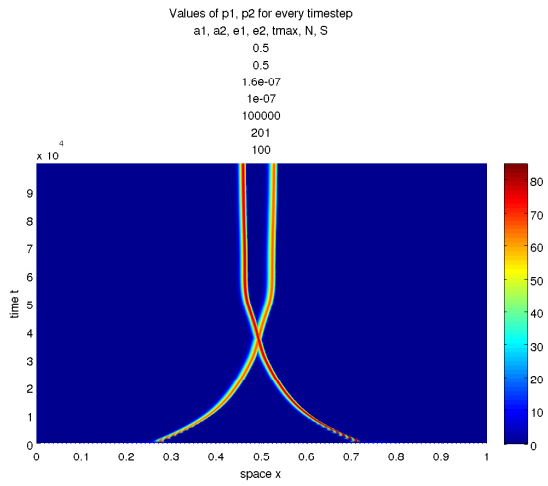


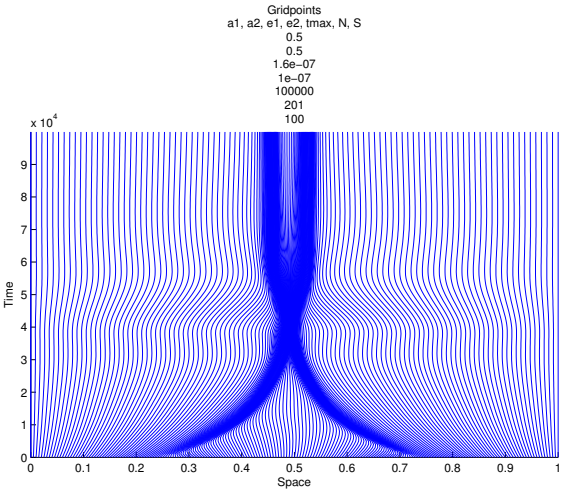


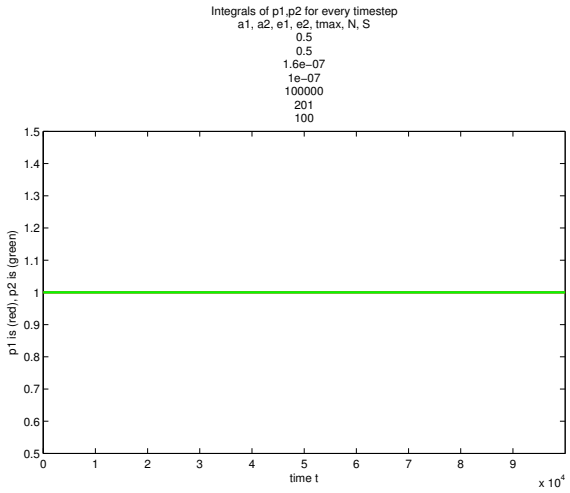
For $a_1 = a_2 = 0,5$**,** $\epsilon_1 = 1,6 * 10^{-7}$**,** $\epsilon_2 = 10^{-7}$ **and** $tmax = 100000$



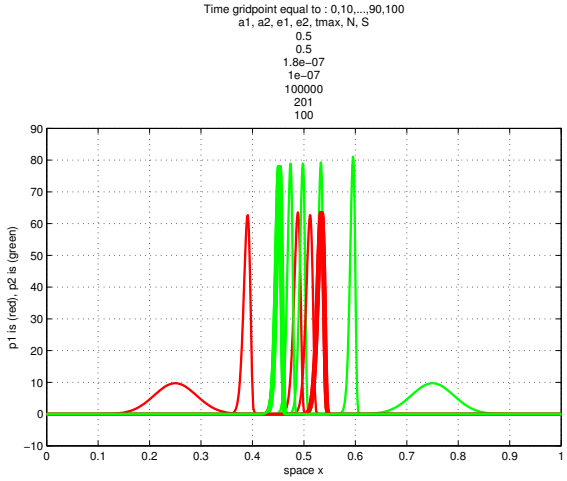


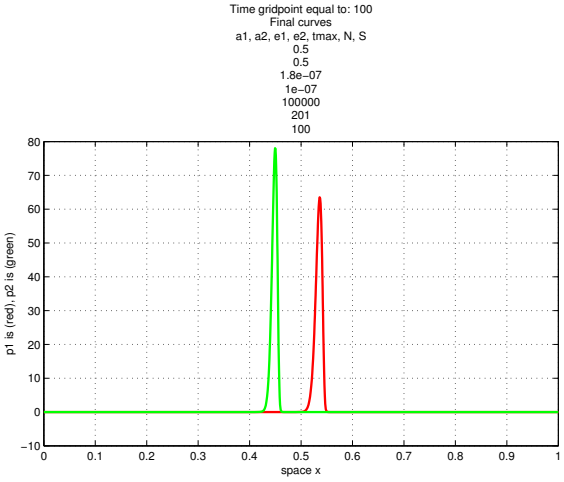


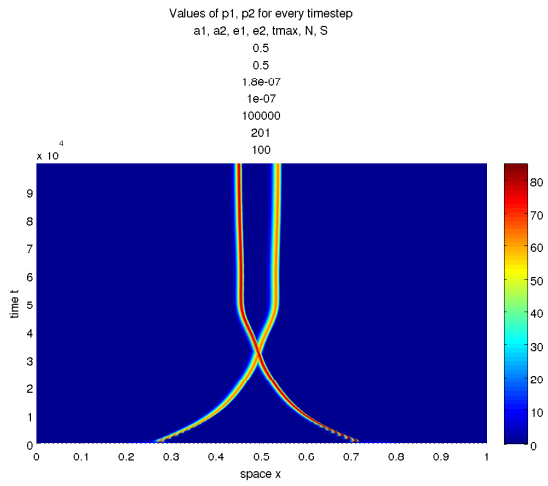


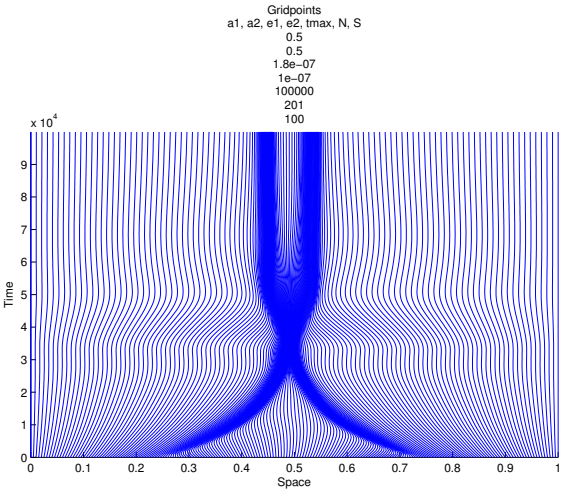


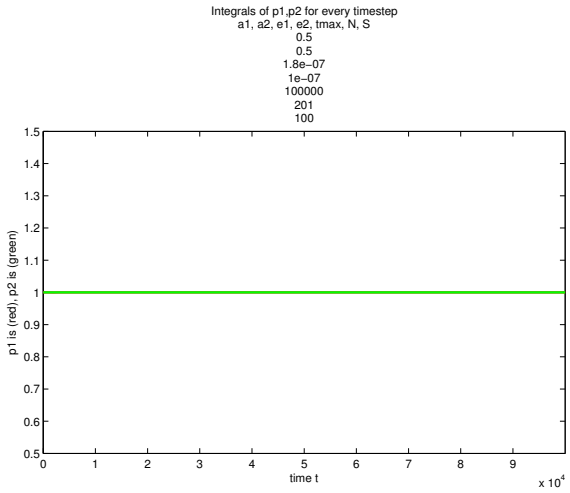
For $a_1 = a_2 = 0,5$**,** $\epsilon_1 = 1,8 * 10^{-7}$**,** $\epsilon_2 = 10^{-7}$ **and** $tmax = 100000$



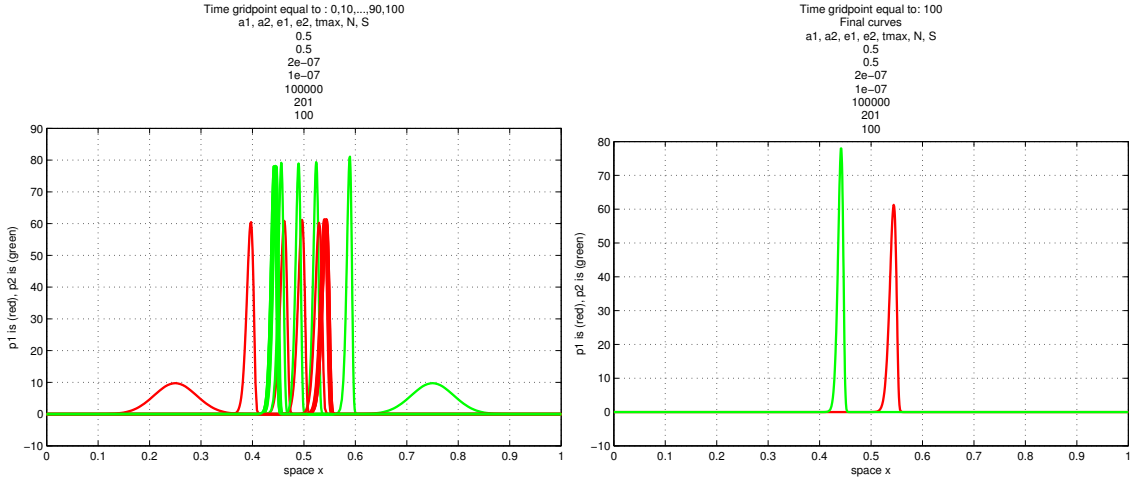


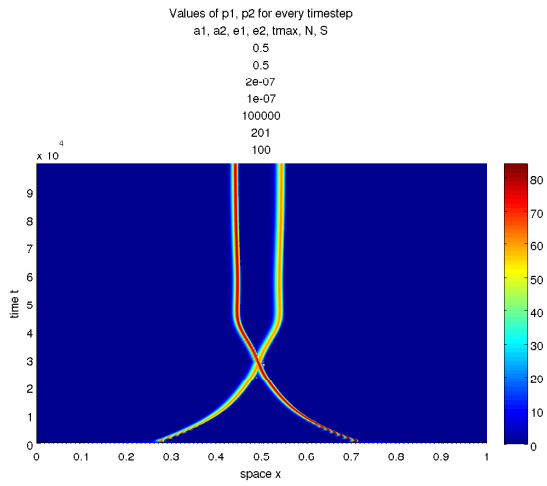


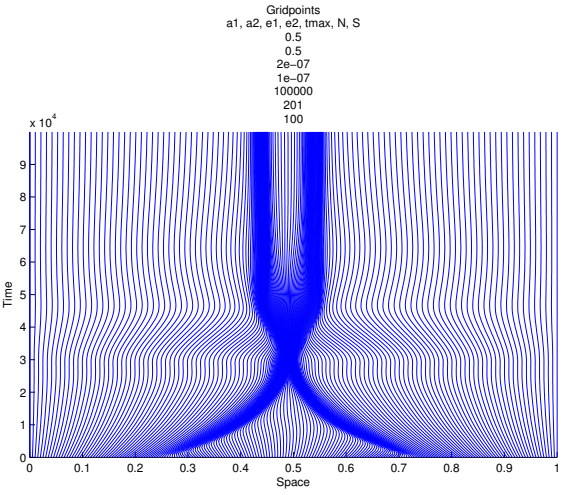


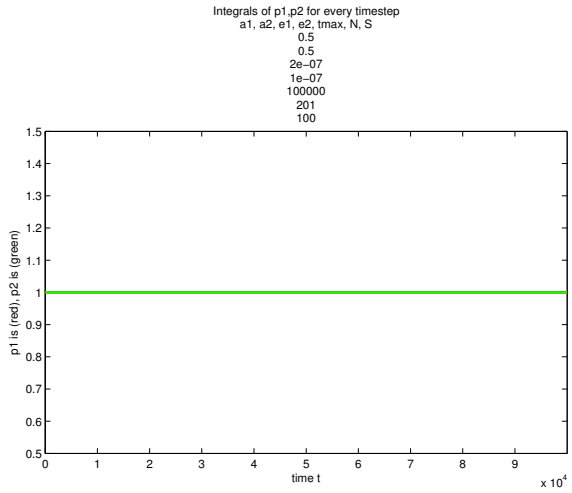


For $a_1 = a_2 = 0,5$**,** $\epsilon_1 = 2 * 10^{-7}$**,** $\epsilon_2 = 10^{-7}$ **and** $tmax = 100000$









As we expected, since ϵ_1 is getting larger from case to case, ρ_1 is getting shorter. On the other hand, since ϵ_2 has the same value for all the cases, we see that ρ_2 is becoming as high as in all the cases.

By looking at the third figure of each one of the previous cases, we see that as the difference between ϵ_1 and ϵ_2 is growing, the X phenomenon is taking place at a sooner time level. Furthermore, the distance between the peaks of ρ_1 and ρ_2, after the curves have become stable, is getting larger as the difference between ϵ_1 and ϵ_2 is growing.. The distance between the curves, is more obvious in the second figure for all cases.

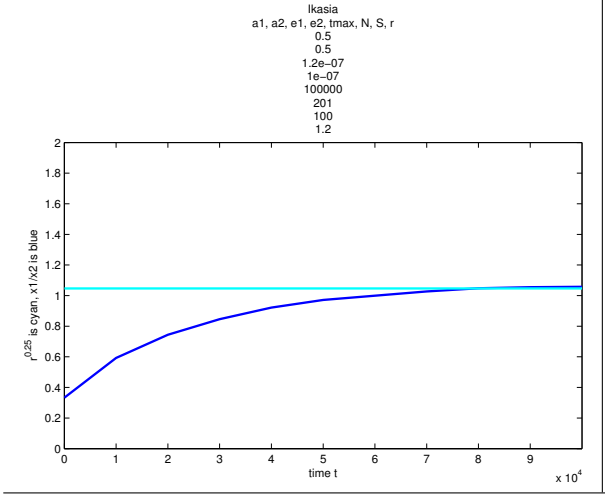
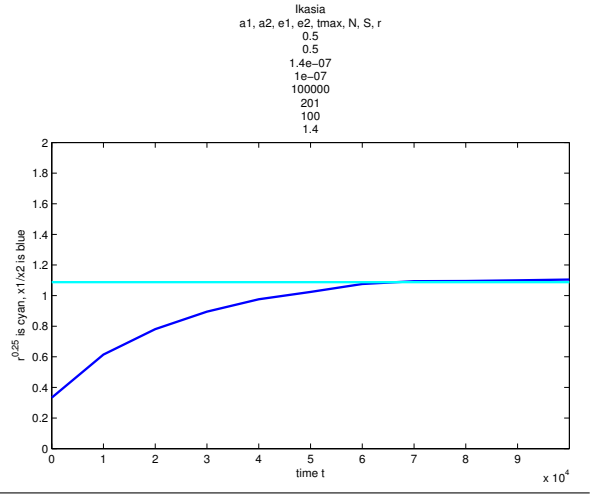
By looking at the second figure, we can observe, the skewness of both ρ_1 and ρ_2.

The last figure, shows that the volume of the curves is constantly equal to 1.

We will do the same work as in §3.3, in order to study the phenomenon, where at the last cases the curves ρ_1 and ρ_2, were getting apart from each other with respect to the values of ϵ_1 and ϵ_2.

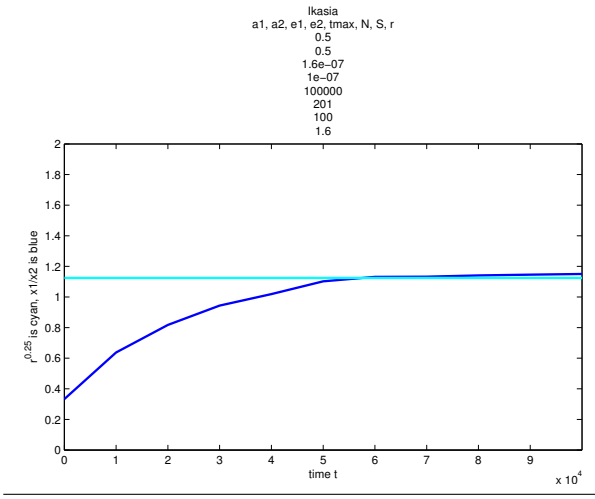
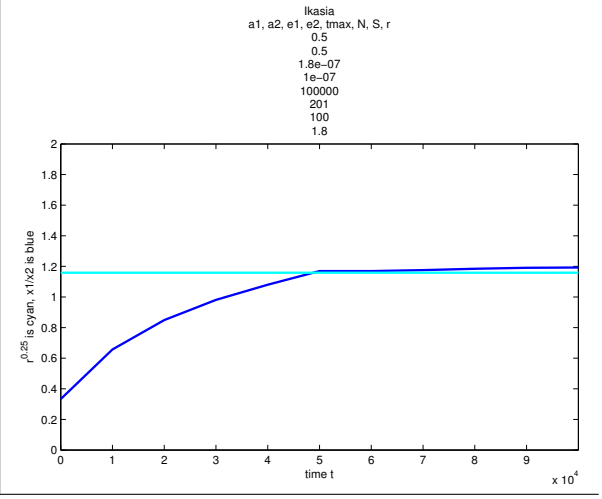
Let $x1$ be the location of the peak of ρ_1 and $x2$ the location of the peak of ρ_2 for every value of time t. We wanted to see if the sequence $\left(\frac{x1}{x2}\right)_t$, converges to $r^{0.25}$, where $r = \frac{\epsilon_1}{\epsilon_2}$, as time t increases.

In the following figures, the value $\frac{x1}{x2}$ corresponds to the blue line and the value $r^{0.25}$ corresponds to the cyan line.

For $\epsilon_1 = 1,2 * 10^{-7}$, $\epsilon_2 = 10^{-7}$	For $\epsilon_1 = 1,4 * 10^{-7}$, $\epsilon_2 = 10^{-7}$
	

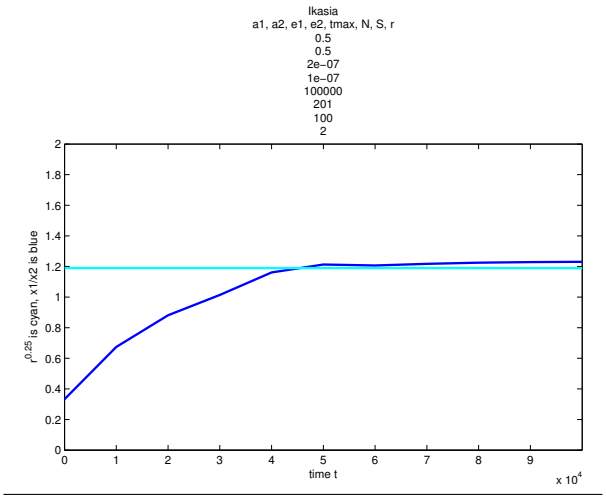
For $\epsilon_1 = 1,2 * 10^{-7}$, $\epsilon_2 = 10^{-7}$, we have $r = \frac{\epsilon_1}{\epsilon_2} = 1,2$. The location of the peak of ρ_1, converges to $x1 = 0,5072$ and of ρ_2, to $x2 = 0,4797$.

For $\epsilon_1 = 1,4 * 10^{-7}$, $\epsilon_2 = 10^{-7}$, we have $r = \frac{\epsilon_1}{\epsilon_2} = 1,4$. The location of the peak of ρ_1, converges to $x1 = 0,5181$ and of ρ_2, to $x2 = 0,4690$.

For $\epsilon_1 = 1,6 * 10^{-7}$, $\epsilon_2 = 10^{-7}$	For $\epsilon_1 = 1,8 * 10^{-7}$, $\epsilon_2 = 10^{-7}$
	

For $\epsilon_1 = 1,6 * 10^{-7}$, $\epsilon_2 = 10^{-7}$, we have $r = \frac{\epsilon_1}{\epsilon_2} = 1,6$. The location of the peak of ρ_1, converges to $x1 = 0,5275$ and of ρ_2, to $x2 = 0,4585$.

For $\epsilon_1 = 1,8 * 10^{-7}$, $\epsilon_2 = 10^{-7}$, we have $r = \frac{\epsilon_1}{\epsilon_2} = 1,8$. The location of the peak of ρ_1, converges to $x1 = 0,5362$ and of ρ_2, to $x2 = 0,4495$.

For $\epsilon_1 = 2 * 10^{-7}$, $\epsilon_2 = 10^{-7}$


For $\epsilon_1 = 2 * 10^{-7}$, $\epsilon_2 = 10^{-7}$, we have $r = \frac{\epsilon_1}{\epsilon_2} = 2$. The location of the peak of ρ_1, converges to $x1 = 0,5440$ and of ρ_2, to $x2 = 0,4422$.

We gather all the values in the following table.

r	$x1$	$x2$	$x1 - x2$	$x1/x2$	m
1,2	0,5072	0,4797	0,0276	1,0575	0,3066
1,4	0,5181	0,4690	0,0491	1,1046	0,2956
1,6	0,5275	0,4585	0,0690	1,1506	0.2985
1,8	0,5362	0,4495	0,0867	1,1928	0.3
2	0,5440	0,4422	0,1018	1,2303	0,2990

where $r = \frac{\epsilon_1}{\epsilon_2}$, $x1$ is the peak of ρ_1 after it is stabilized, similarly $x2$ for ρ_2, the value $x1 - x2$ gives the distance of the peaks in each case and m comes from $\frac{x1}{x2} = r^m$.

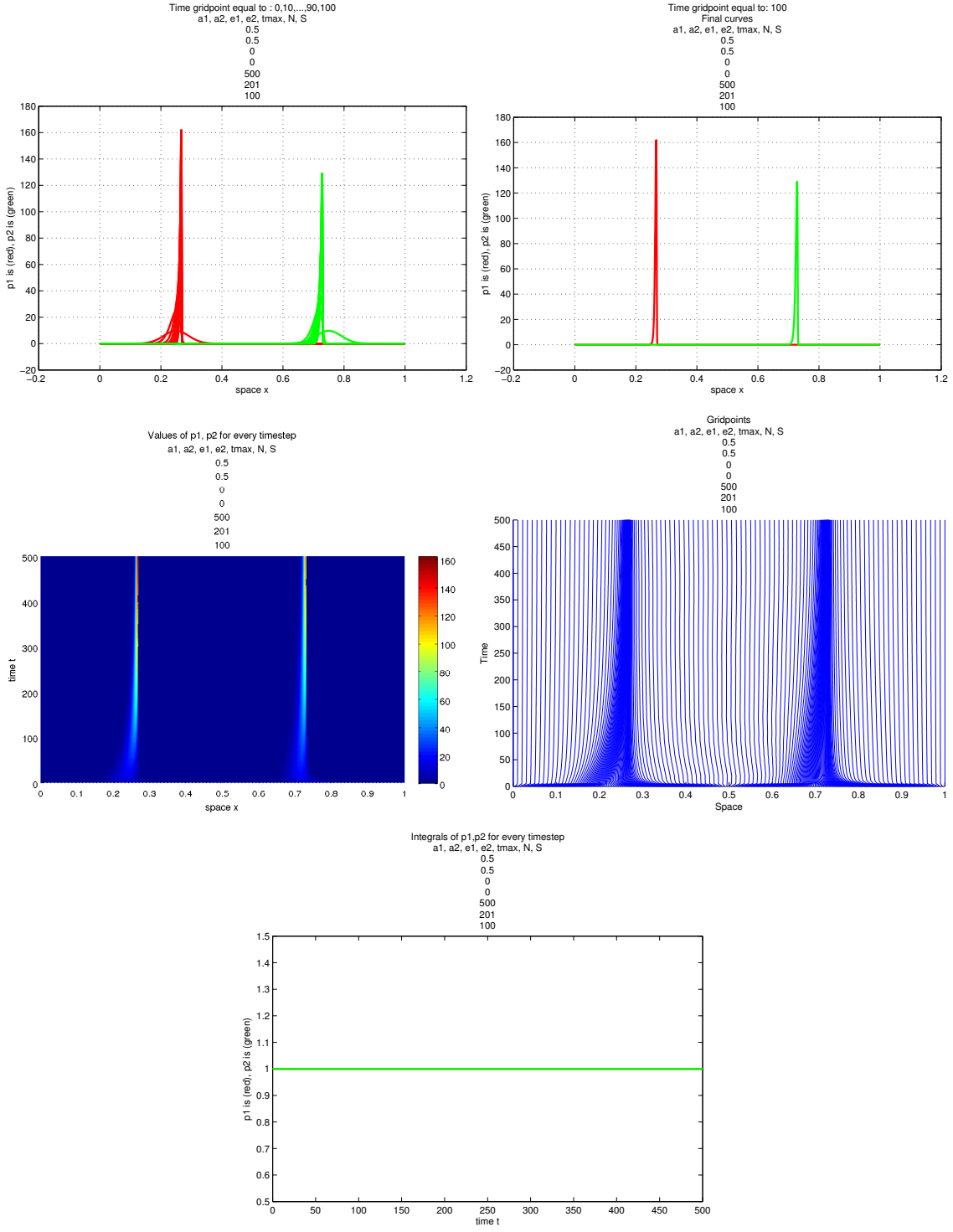
So, once again, from the fourth column, we can conclude that the distance between the peaks of ρ_1 and ρ_2 is growing for larger values of r.

In the figures above, we can see the behaviour of the ratio $\frac{x1}{x2}$, as time grows. It is obvious that, the sequence $\left(\frac{x1}{x2}\right)_t$, converges to some value for all r. These values are given in the fifth column of the table.

At last we wanted to see, if the sequence $\left(\frac{x1}{x2}\right)_t$, converges to $r^{0.25}$ as time grows. In the last column of the table, we give the value m, that is $\frac{x1}{x2} = r^m$, for all values of r. Unfortunatelly, m, does not seem to be equal to 0,25 as we were hoping, but is close to $0,3$.

For the case where $\epsilon_1 = \epsilon_2 = 0$, we have the following results.

For $a_1 = a_2 = 0,5$**,** $\epsilon_1 = \epsilon_2 = 0$ **and** $tmax = 500$



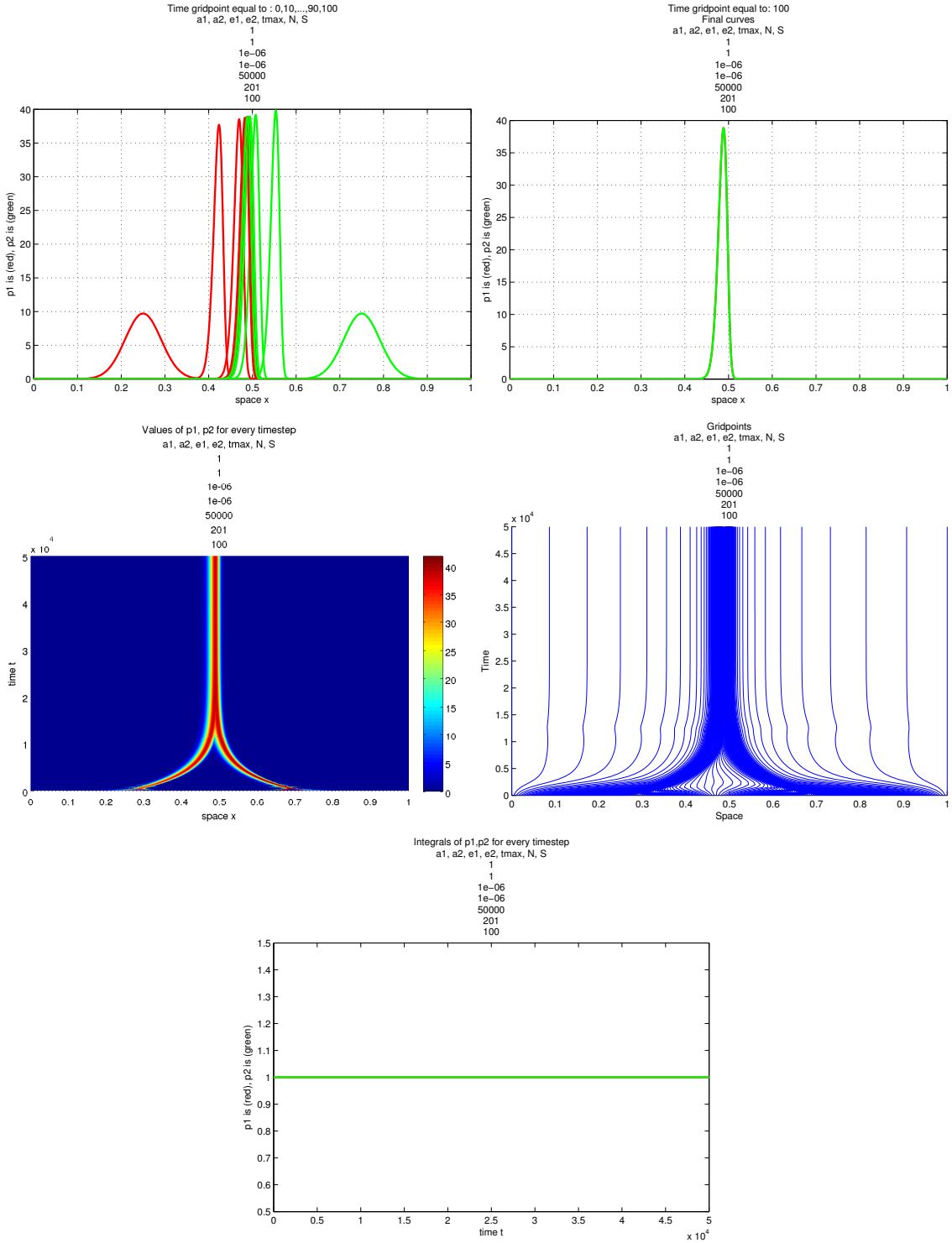
It is obvious that in this case the curves ρ_1 and ρ_2 are not moving in space, which is expected since for $\epsilon_1 = \epsilon_2 = 0$ the diffution term of the model is deleted. Even more, the curves become extremelly steep compared to any of the previous cases. In the second figure we can see that they converge to delta functions.

From the first two figures, we also observe that curve ρ_2 is shorter than ρ_1. Here skewness is also very obvious for both the curves.

At last the fifth figure shows that the volumes of both the curves are preserved at all time.

Further, we give the results that correspont to the case $a_1 = a_2 = 1$, that is $u_1(x) = x^{a_1} = x$ and $u_2(x) = x^{a_2} = x$.

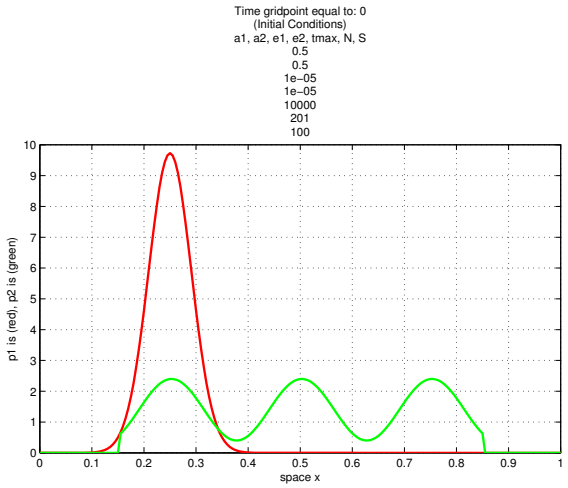
For $a_1 = a_2 = 1$**,** $\epsilon_1 = \epsilon_2 = 10^{-6}$ **and** $tmax = 50000$



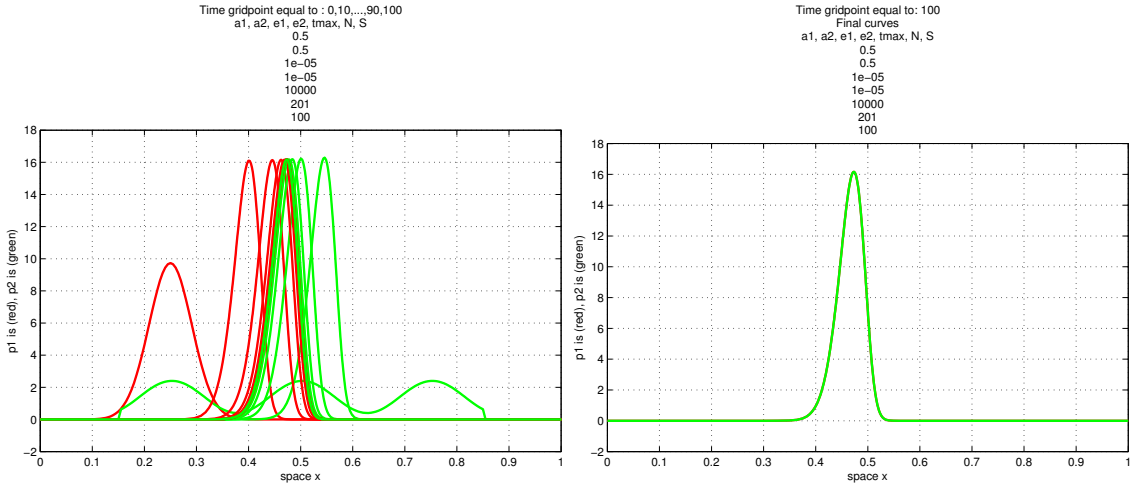
By comparing the case where $\epsilon_1 = \epsilon_2 = 10^{-6}$ and $a_1 = a_2 = 0,5$ with the latter one where $a_1 = a_2 = 1$, we observe that, for $a_1 = a_2 = 0,5$, the curves concentrate at the same position at around time $t = 30000$, while for $a_1 = a_2 = 1$, at around time $t = 25000$.

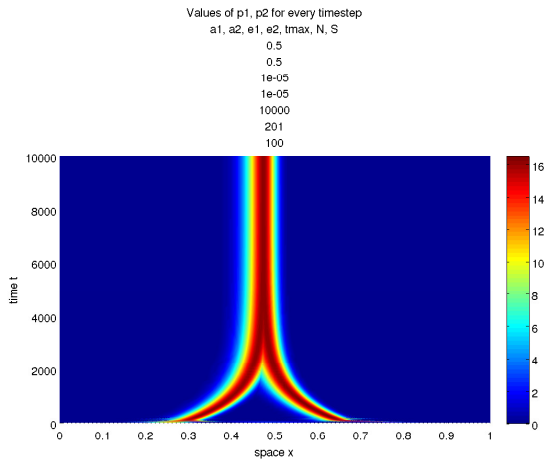
Further, for $a_1 = a_2 = 0,5$, ρ_1 and ρ_2 reach height around 35 and then they mostly change their position in space. On the other hand, for $a_1 = a_2 = 1$, we see that ρ_1 reaches height around 37 and keeps gaining more until it becomes stable. ρ_2 reaches height around 40 and until it becomes stable loses a little bit. In the end after both curves have converged, they are equal so, both share the same height around 38.

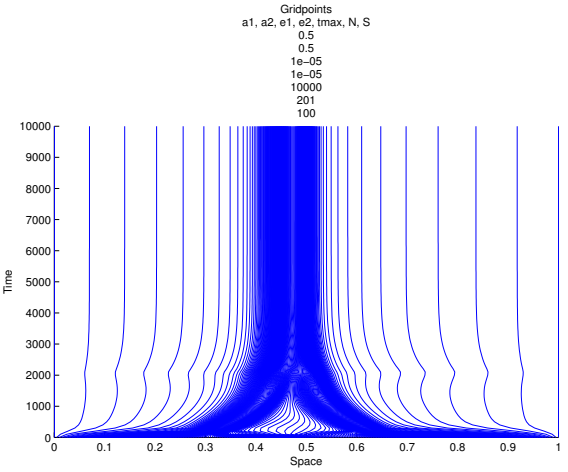
At last, we will give the results that correspond to different initial values. So, for Gaussian function as initial value of ρ_1 and sine function for ρ_2, we have that for time $t = 0$, we have:

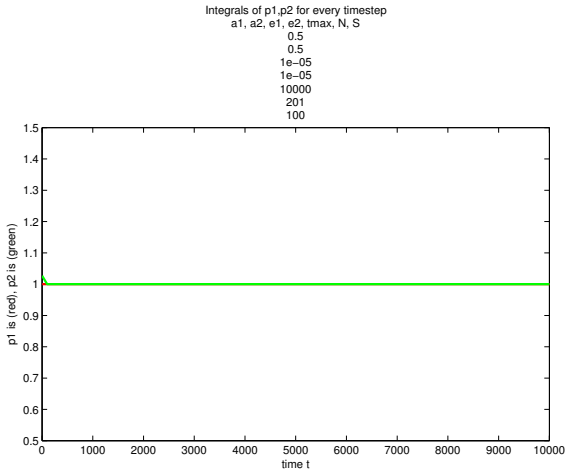


For $a_1 = a_2 = 0,5$**,** $\epsilon_1 = \epsilon_2 = 10^{-5}$ **and** $tmax = 10000$



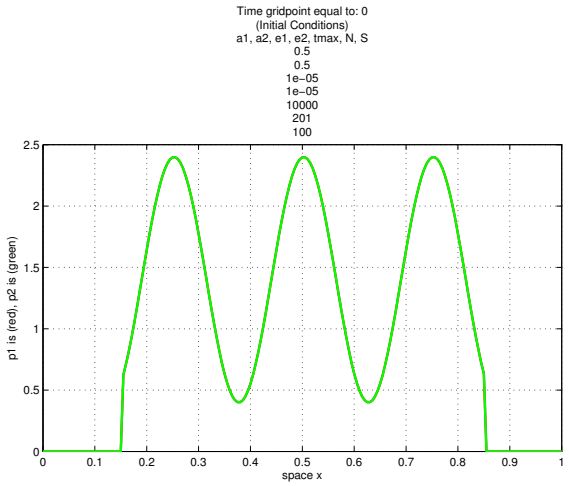




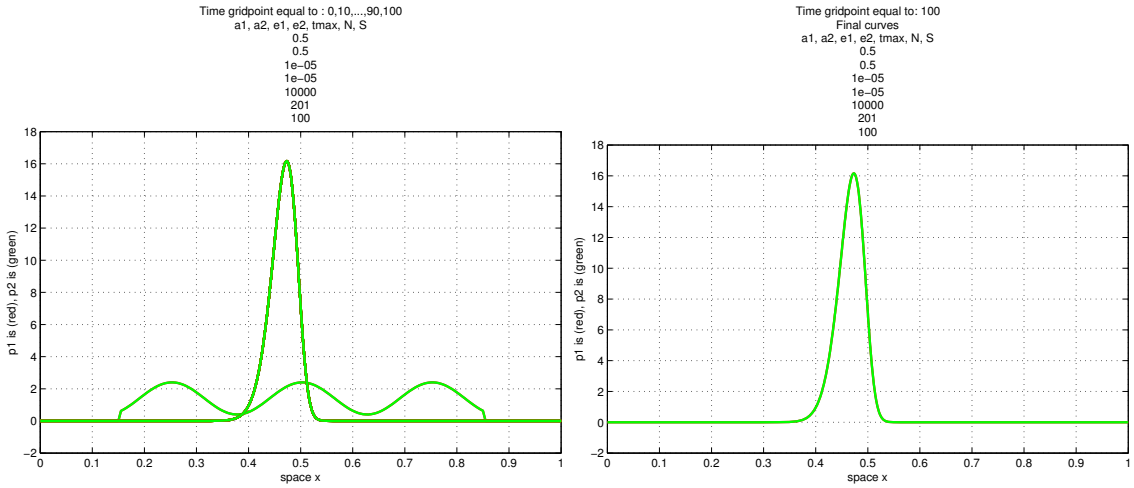


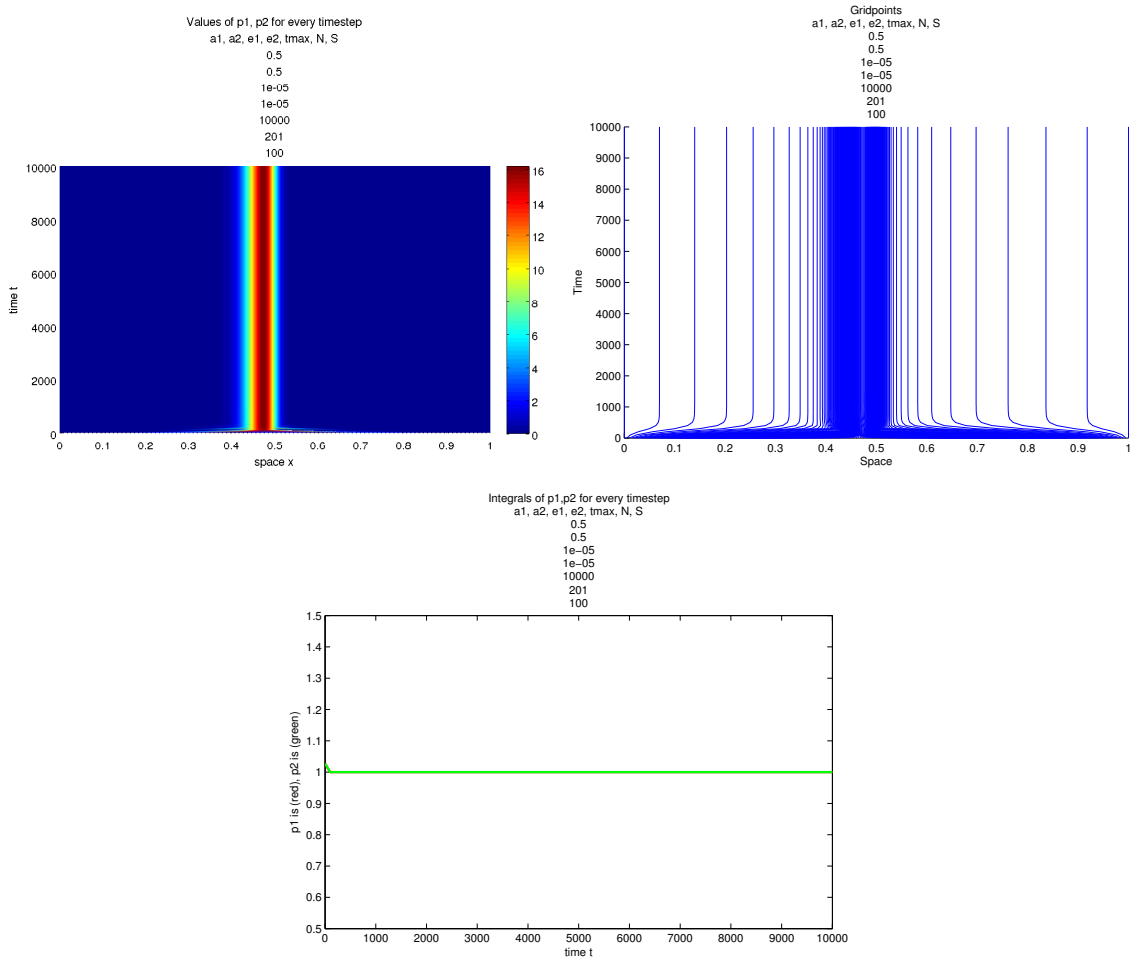
As we can see from the first figure, although ρ_2 initially is some sine function, it changes morphologically from the start. The curve becomes a Gaussian function very fast. Since it also becomes Gaussian, the behaviour of the model is similar to the previous cases. By comparing the latter case with the corresponding one with Gaussian initial functions for both ρ_1, ρ_2, we observe that the evolution is the same.

For sine function as initial value of ρ_1 and the same sine function for ρ_2, we have that for time $t = 0$, we have:

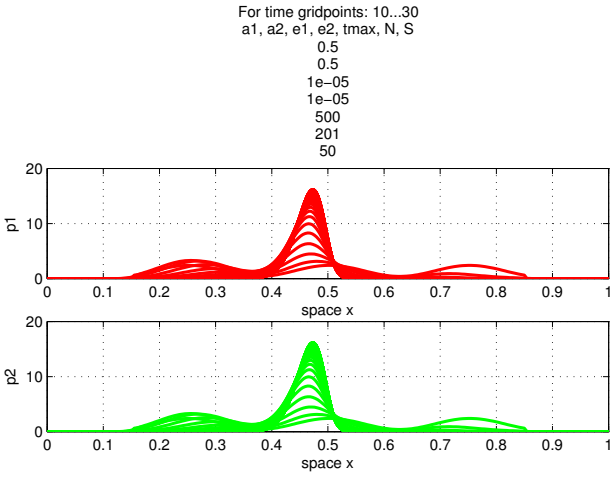


For $a_1 = a_2 = 0,5$**,** $\epsilon_1 = \epsilon_2 = 10^{-5}$ **and** $tmax = 10000$





In this case $\rho_1 = \rho_2$ for time $t = 0$. As we can see from the figures, ρ_1 is equal to ρ_2 for all times t. As in the previous case, both curves are rising and very soon they become Gaussians. Below, the figure is a zoom in time, that shows the morphological change of the curves. Initially they are sine functions and after a small time interval they are Gaussians.

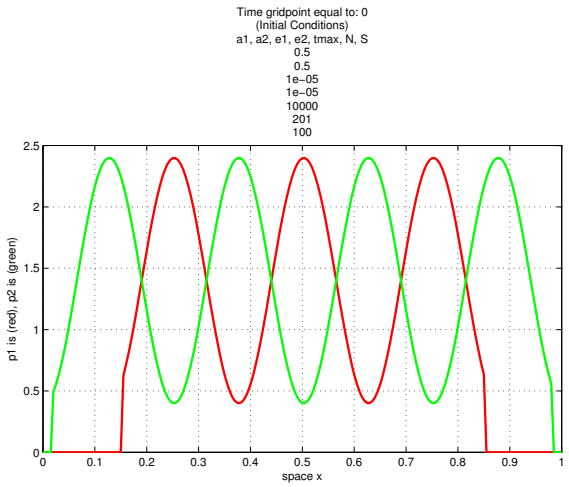


We observe that, although initially there are 3 peaks, the volume of the outer peaks is decreasing while the middle peak is gaining volume and height. At the end, we see that the outer peaks do not exist anymore, while the peak in the middle has become a Gaussian.

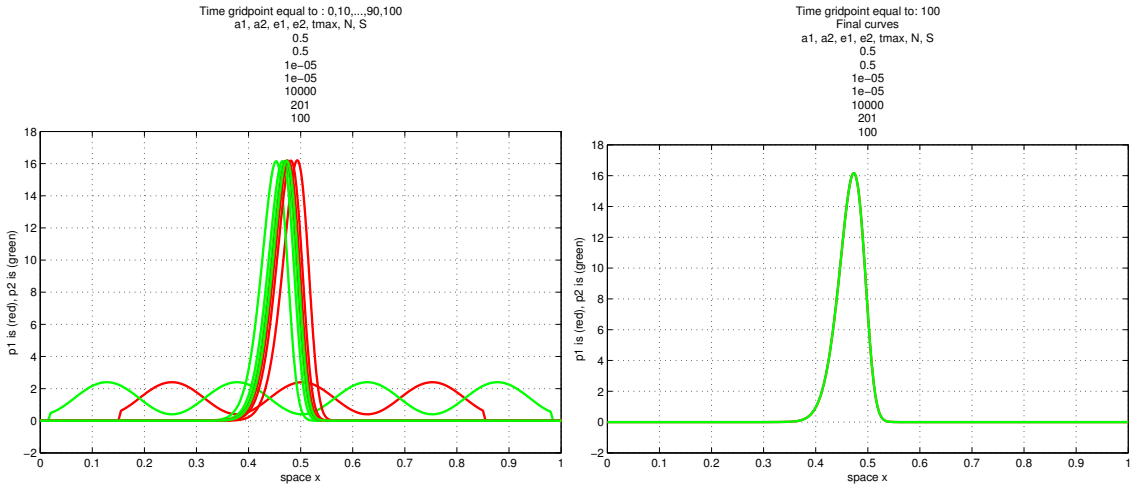
We have to mention that, by comparing the second figure of the current case with the coresponding figure of the case where $a_1 = a_2 = 0,5$, $\epsilon_1 = \epsilon_2 = 10^{-5}$ and the initial values are Gaussian functions for

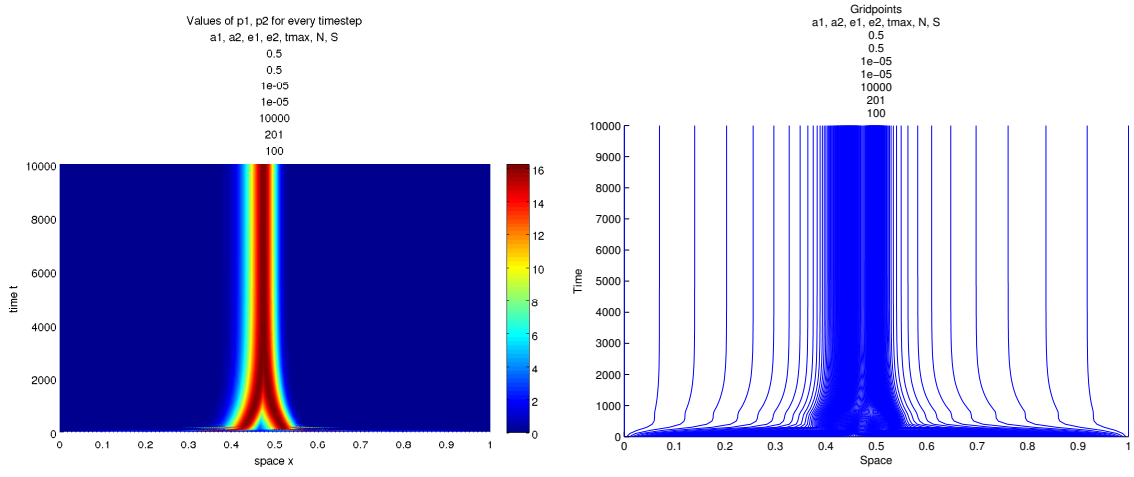
both ρ_1 and ρ_2, we see that although the curves are starting from different initial values, at the end they converge to the same stationary solution.

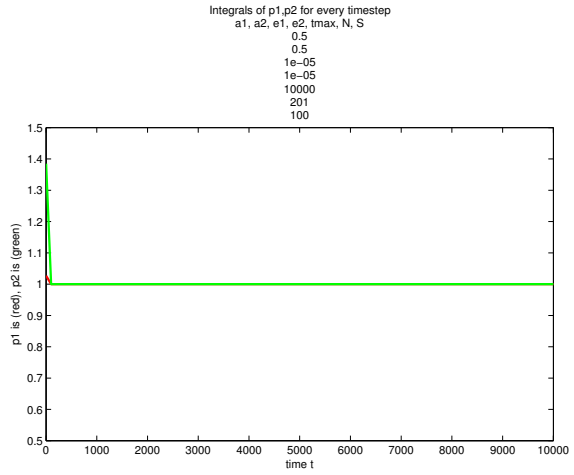
The last case we will consider, is for sine function as initial value of ρ_1 and shifted sine function for ρ_2, so for time $t = 0$, we have:



For $a_1 = a_2 = 0,5$**,** $\epsilon_1 = \epsilon_2 = 10^{-5}$ **and** $tmax = 10000$



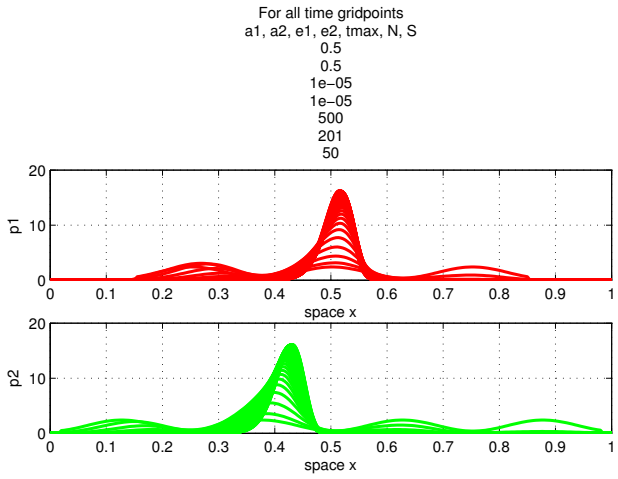




From the first figure, we can see that the curves are rising pretty fast as in the previous case. Soon enough they have become Gaussian functions. In contrast to the previous case, here we have $\rho_1 \neq \rho_2$ for time $t = 0$ and the Gaussians that are formed, have peaks with different positions. So, we see that after the curves have become Gaussians, they move towards the same position.

In the last figure, we can see that initially the volume of ρ_1 is 1, while the volume of ρ_2 is 1,38. But by the time that the curves are transformed to Gaussian functions, the volume of ρ_2 becomes also equal to 1 ever since.

The next figure, is a zoom in time, that shows the steps the curves followed until they become Gaussians.



To conclude. From the last three cases where different initial values were used, we saw that no matter what the initial values will be, the curves will become Gaussian functions very soon. Ever since the behaviour of the solution is similar to the previous cases where we had Gaussian functions as initial conditions for both ρ_1 and ρ_2. We also saw that, no matter what the initial values are, ρ_1, ρ_2, will converge to identical curves, for every case. Hence, we may assume that, taking for initial values always Gaussian functions for both ρ_1 and ρ_2, is enough for studying the model.

5 Conclusions

In this work, we numerically approximated the replicator equations of the Nash bargaining game. In sections §3.3 and §4.3, we derived some results. We gather all these results here.

1. In general, for $\epsilon_1 = \epsilon_2 \neq 0$, the curves ρ_1 and ρ_2, are moving towards the same position. After some time, both curves have converged to identical curves.

As the parameters ϵ_1, ϵ_2 are getting closer to zero, the curves are becoming steeper, they converge to a delta distribution. Evenmore, the movement is slower for smaller ϵ_1, ϵ_2, hence, the curves converge to the same position at a much later time point.

For $\epsilon_1 = \epsilon_2$, we also saw that the morphological change of both ρ_1 and ρ_2, is similar.

2. For $\epsilon_1 \neq \epsilon_2$, the curves ρ_1 and ρ_2, are moving towards to the same position. Here we observed the X-phenomenon and we saw that the curves converge to stationary solutions, that are no longer concentrated at the same position. As the parameters $\epsilon_1, \epsilon_2 \to 0$, the curves converge to delta distributions. Even more, the curve ρ_i with the smallest parameter ϵ_i is steeper than the other.

   As the difference $\epsilon_1 - \epsilon_2$, is getting larger, we observed that, the X phenomenon is taking place at a sooner time point.

   We observe that, as the ratio $r = \frac{\epsilon_1}{\epsilon_2}$ is growing, the distance between the positions of the peaks of ρ_1 and ρ_2, after the curves have become stable, is growing.

3. For $\epsilon_1 = \epsilon_2 = 0$, the curves ρ_1 and ρ_2, are not moving in space. Instead as time grows they become steeper converging to delta functions. In contrast to the case $\epsilon_1 = \epsilon_2 \neq 0$, for $\epsilon_1 = \epsilon_2 = 0$ the curves do not have the same morphological behaviour. Curve ρ_1 is gaining height faster than ρ_2.

4. For every case, we observed that, although the curves ρ_1 and ρ_2 are moving in space and at the same time they change morphologically, their volumes are always equal to 1, for all times, as we wanted them to be.

5. The statinary solution of the equations is not affected by the initial values.

6. At last, we need to mention that the curves become skewed.

The basic results above, do agree with the theoretical analysis of the replicator equations for the Nash bargaining game that has been done in [6].

The difference between the fixed grid and adaptive moving grid methods, is the accuracy that each one can offer. We saw that, as ϵ_1, ϵ_2 are getting closer to zero, the curves ρ_1 and ρ_2 are becoming steeper. That makes our job harder. If we insist on using the fixed grid method, then we have to use a large number of spatial gridpoints and hence due to stability, we also have to use a lot more time gridpoints. Of course that is possible and indeed will result to more accurate approximations, but makes the fixed grid method time consuming. On the other hand, with adaptive moving grid method, it is easier to increase the accuracy level by using less spatial and time gridpoints. Of course, adaptive moving grid method is a heavier method and hence it takes quite some time to finish its calculations. Even though, if the curves ρ_1 and ρ_2 become steep enough, then adaptive moving grid method, needs less run time in order to give some accurate results, compared to the corresponding fixed grid method that would provide the same accuracy. The difference between the two methods, can be seen also in the introduction of [10].

To conclude, for large ϵ_1 and ϵ_2, the curves that are to be approximated, are not very steep and hence fixed grid method, would be the best method to use. On the other hand, for ϵ_1 and ϵ_2 close to zero it would be better to switch to adaptive moving grid method, since fixed grid method would need longer run times.

6 References

[1] Uri M. Ascher. Numerical Methods for Evolutionary Differential Equations. *SIAM.*

[2] Binmore. Natural Justice. *Oxford University Press*, 2005.

[3] W.Hundsdorfer and J.G. Verwer. Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations. *Springer, 1st edition*, 2003.

[4] A. Muthoo. Bargaining Theory with Applications. *Cambridge University Press*, 1999.

[5] J.F, Nash. The bargaining problem. *Econometrica*, Vol. 18 (no. 2): pp. 155-162, 1950.

[6] M. Ruijgrok. Replicator dynamics for the Nash bargaining game. To appear 2011.

[7] B. Skyrms. Evolution of the Social Contract. *Cambridge University Press*, 1996.

[8] A. van Dam and P.A. Zegeling. A robust moving mesh finite volume method applied to 1d hyperbolic conservation laws from magnetohydrodynamics. *J. Comput. Phys., (216):526-546*, 2006.

[9] P.A. Zegeling. Moving-grid methods for time-dependent partial differential equations. *CWI-track No.94, Centre for Mathematics and Computer Science, Amsterdam*,1993.

[10] P.A. Zegeling. Theory and Application of Adaptive Moving Grid Methods. *Chapter 7 in Adaptive Computations: Theory and Algorithms, T. Tang, J. Xu*, 2007.

[11] P.A. Zegeling. Moving grid techniques. *Chapter 37 in Handbook of Grid Generation, Eds. Joe F. Thompson, Bharat K. Soni, Nigel P. Weatherill, CRC Press LLC*, 1999.

[12] P.A. Zegeling, I. Lagzi, and F. Izsak. Transition of Liesegang precipitation systems: simulations with an adaptive grid PDE method. to appear in *Communications in Computational Physics*, 2011.

[13] K.E. Brenan, S.L. Campbell, L.R. Petzold. Numerical solution of initial-value problems in differential-algebraic equations. Chapter 5 *SIAM.* 1996.

[14] L. Petzold. DASSL code. *http://pitagora.dm.uniba.it/ testset/solvers/dassl.php.*

[15] F. Doster, P.A. Zegeling, R. Hilfer. Numerical Solutions of a Generalized Theory for Macroscopic Capillarity. *Physical Review E81, 1*, 2010.

7 Appendices

7.1 Fixed grid method codes

The code that is given in this senction, corresponds to the fixed grid method. The code is builted on Matlab.

```
clear all

% Definitions
tmax = input('Tmax= : ');
N = input('Number of spatial subintervals: ');
S = input('Number of time subintervals: ');
time1 = cputime;
xmin = 0;
xmax = 1;
tmin = 0;
Dx = (xmax - xmin)/N;
Dt = (tmax - tmin)/S;
n = N+1;
s = S+1;
x = zeros(1,n);  % (1xn)-space gridpoints
t = zeros(1,s);  % (1xs)-time gridpoints
```

```
p1 = zeros(1,n); % values of p1(i)=p1(x(i))
p2 = zeros(1,n); % values of p2(i)=p2(x(i))
p1new = zeros(1,n);
p2new = zeros(1,n);
F1 = zeros(1,n);
F2 = zeros(1,n);
e1 = 10^(-7); % parameter e1=epsilon1 from the model
e2 = 10^(-7); % parameter e2=epsilon2 from the model
a1 = 0.5; % this is used in function u1(x,a1)
a2 = 0.5; % this is used in function u2(x,a2)
% Function u1 is defined in the path call the function u1(x,a1)
% Function u2 is defined in the path call the function u2(x,a2)
figs = zeros(1,4);

if n*s>10000000 % maximum matrix size matlab can take
    maxs = floor(10000000/n);
    bhma = floor(s/maxs)+1;
    ss = floor(s/bhma)+1;
else
    bhma = 1;
    ss = s;
end

p11 = zeros(ss,n); % all data for figures
p22 = zeros(ss,n); % all data for figures
Integralsp1 = zeros(1,ss);
Integralsp2 = zeros(1,ss);
t1 = zeros(1,ss); % time axis for the figures



% Create the gridpoints for space and time

for i=1:n
    x(i)=(i-1)*Dx;
end
for j=1:s
    t(j)=(j-1)*Dt;
end


% Initial conditions (t=0 -> j=1)

b1=1/4; % gives the center of the bell
b2=3/4;
c=0.041; %0.0064;      % gives the width of the bell
alpha=1/(c*sqrt(2*pi));   % gives the height of
                          % the bell w.r.t. c
                          % such that the integral of
                          % the gaussian from (-inf,inf)
                          % is equal to one 1.
for i=1:n
    p1(i)=alpha*gaussmf(x(i),[c b1]); % p1 for t=0
    p2(i)=alpha*gaussmf(x(i),[c b2]); % p2 for t=0
end
j=1;
jj=1;
```

```
p11(jj,:)=p1;
p22(jj,:)=p2;
t1(jj)=t(j);
% Give the first 2D figure1 for t=0=tmin (initial conditions)
figure(1)
plot(x,p1,'r',x,p2,'g')
xlabel('space x')
ylabel('p1 is (red), p2 is (green), gauss function')
grid on
title({'Time equal to:' (j-1)*Dt '(Initial Conditions)' 'a1, a2,
>>>>c, e1, e2, tmax, Dx, Dt, N, S' a1 a2 c e1 e2 tmax Dx Dt N S})
figs(1)=figure(1);
set(gcf, 'Visible', 'off')


% Calculate the volume of p1, p2 in [0,1] to see if its close to one
Summa1=(Dx/2)*(2*(sum(p1)-p1(1)-p1(n))+p1(1)+p1(n));
fprintf('For time t = 0, the integral of the function p1(x) from 0 to
>>>>1 is equal to : %f . \n',Summa1 );
Integralsp1(jj) = Summa1;
Summa2=(Dx/2)*(2*(sum(p2)-p2(1)-p2(n))+p2(1)+p2(n));
fprintf('For time t = 0, the integral of the function p2(x) from 0 to
>>>>1 is equal to : %f . \n',Summa2 );
Integralsp2(jj) = Summa2;
% The integral in [0,1] of the initial condition must be equal to 1


% Main loop

for j=2:s

    % Calculate F1,F2 with previous time level data
for i=1:(n-1)
% until (n-1) since F1(n)=int(0,1-x_n)f(x)=int(0,1-1)=int(0,0)=0 always
    F1(i) = (Dx/2)*(p2(1)+p2(n-i+1)+2*SUMF1(p2,n,i));
    F2(i) = (Dx/2)*(p1(1)+p1(n-i+1)+2*SUMF2(p1,n,i));
end
% Calculate P1,P2 with previous time level data
P1=(Dx/2)*(p1(1)*u1(x(1),a1)*F1(1)+p1(n)*u1(x(n),a1)*F1(n)
>>>>  +2*SUMP1(p1,x,F1,n,a1));
P2=(Dx/2)*(p2(1)*u2(x(1),a2)*F2(1)+p2(n)*u2(x(n),a2)*F2(n)
>>>>  +2*SUMP2(p2,x,F2,n,a2));

% Boundary conditions of p1,p2 for x=0, x=1
p1new(1)=0;
p1new(n)=0;
p2new(1)=0;
p2new(n)=0;
% the new values of p1, p2
for i=2:(n-1)
    p1new(i)=((Dt*e1)/(Dx)^2)*(p1(i+1)-2*p1(i)+p1(i-1))
>>>>   +Dt*p1(i)*(u1(x(i),a1)*F1(i)-P1)+p1(i);
    p2new(i)=((Dt*e2)/(Dx)^2)*(p2(i+1)-2*p2(i)+p2(i-1))
>>>>   +Dt*p2(i)*(u2(x(i),a2)*F2(i)-P2)+p2(i);
end
p1=p1new;
p2=p2new;
```

```
if j/bhma == floor(j/bhma)    % mod(j/bhma,1) == 0
    jj=jj+1;
    p11(jj,:)=p1;
    p22(jj,:)=p2;
    t1(jj)=t(j);
end

% Calculate the volume of p1, p2 in [0,1] to see if its close to one
% Trapezoidal Rule:
Summa1=(Dx/2)*(2*(sum(p1)-p1(1)-p1(n))+p1(1)+p1(n));
fprintf('For time t = %f, the integral of the function p1(x)
>>>>from 0 to 1 is equal to : %f . \n', (j-1)*Dt , Summa1 );
if j/bhma == floor(j/bhma)    % mod(j/bhma,1) == 0
    Integralsp1(jj) = Summa1;
end
Summa2=(Dx/2)*(2*(sum(p2)-p2(1)-p2(n))+p2(1)+p2(n));
fprintf('For time t = %f, the integral of the function p2(x)
>>>>from 0 to 1 is equal to : %f . \n', (j-1)*Dt , Summa2 );
if j/bhma == floor(j/bhma)    % mod(j/bhma,1) == 0
    Integralsp2(jj) = Summa2;
end
% The integral in [0,1] of the initial condition must be equal to 1

end
time2 = cputime;
time=time2-time1;
fprintf('The CPU time elapsed is: %f . \n', time);

if n*s>10000000 % maximum matrix size matlab can take
    fprintf('Too many data, so the number of the data will be changed
>>>>to fit the figures. \n');
end


% Give the last 2D figure2 for (t=n*Dt=tmax)
figure(2)
plot(x,p1,'r',x,p2,'g')
xlabel('space x')
ylabel('p1 is (red), p2 is (green)')
title({'Time equal to:' (s-1)*Dt  'a1, a2, c, e1, e2, tmax,
>>>>Dx, Dt, N, S' a1 a2 c e1 e2 tmax Dx Dt N S})
grid on
figs(2)=figure(2);
% set(gcf, 'Visible', 'off')


save('/home/students/3402266/Results_Ikasia/c=0,041_e1=r*e2_e2=1e-7_r=0_
>>>>Tmax=300000_N=2000_data.mat','r', 'Dt','Dx','Integralsp1',
>>>>'Integralsp2',
>>>>'N','S','a1','a2','alpha','b1','b2','bhma','c','e1','e2','n','p11',
>>>>'p22','s','ss','t','t1','time','tmax','tmin','x','xmax','xmin')
```

where the functions that are used in the code are given below:

```
function [FF1]=SUMF1(p2,n,i)
FF1=0;
for l=2:(n-i)
    FF1=FF1+p2(l);
```

```
end

function [FF2]=SUMF2(p1,n,i)
FF2=0;
for q=2:(n-i)
    FF2=FF2+p1(q);
end

function [PP1]=SUMP1(p1,x,F1,n,a1)
PP1=0;
for ll=2:(n-1)
    PP1=PP1+p1(ll)*u1(x(ll),a1)*F1(ll);
end

function [PP2]=SUMP2(p2,x,F2,n,a2)
PP2=0;
for qq=2:(n-1)
    PP2=PP2+p2(qq)*u2(x(qq),a2)*F2(qq);
end

function [output] = u1(x,a1)
output = x.^a1;
end

function [output] = u2(x,a2)
output = x.^a2;
end
```

7.2 Adaptive moving grid method codes

The code that is given in this senction, corresponds to the adaptive moving grid method. The code is builted on Fortran77 and is a basic subroutine of a larger code *lib.f* that was used in this work. *lib.f*, is using the code *DASSL*, look [13, 14]. The code that is presented here, gives the part where the spatial gridpoints are reallocated in each time level and the formulas that were derived in this work for the numerical approximation of the functions ρ_1 and ρ_2. The time gridpoints, are reallocated in *lib.f* file.

```
      program my
      parameter(npts=201,neq=3*npts)
c       solving my system u1t=u1x+f(u2) u2t=u2x+f(u1) on interval [xl,xr]
c       implicit using DDASSL

      dimension y(neq),ydot(neq),info(15),iwork(100000)
      dimension x(npts)
      dimension tprint(9)

      double precision u,x,xdot,y,ydot
      double precision length,alpha,dt,t,tout,v,d
      double precision xleft,xright,uleft,uright
      double precision em,xl,xr,rpi,dd,rk,uexact
      double precision rtol,atol,rpar,rwork(100000)
CCC my variables
      double precision b,c,aa,pi,b1,b2
      double precision Integral1,Integral2,Integ1,Integ2

      character*10 xuname,uplnm
      character*12 outname
```

```
      common /needed/ alpha,dt,beta,k,gamma
      common /werk/ rwork

      external resid,ddassl

c-----------------------------------------------------------------------
c       input of variables

**       write(*,'(''Give name of output file: '',$)')
**       read(*,*) outname
      outname='UPLOTn.dat'
      write(*,'(''Give Tend: '',$)')
      read(*,*) tend
      dt = tend/100.0d0
*       write(*,'(''Give number of time steps: '',$)')
*       read(*,*) nts
      nts=100

c-----------------------------------------------------------------------


c       initialisation

c       initialisation for ddassl
*       write(*,'(''Give tol: '',$)')
*       read(*,*) tol
      tol = 0.0001d0
      atol=tol
      rtol=tol
      liw=20+neq
*****       lrw=40+(maxord+4)*neq+neq**2

      do 10 i=1,neq
        ydot(i)=0.0d0
10      continue
      do 20 i=1,15
        info(i)=0
20      continue
***** bandstructure:
      info(6)=1
*let op: 6de afgeleiden in dit model!
      iwork(1)=12
      mu=iwork(1)
      iwork(2)=12
      ml=iwork(2)
*       info(3) = 1
      info(11)=0
C maxord is niet standaard=5:
      info(9)=1
C maxord is wel=iwork(3)=...:
      iwork(3)=5
      maxord=iwork(3)
* bandmatrix:
      lrw=40+(maxord+4)*neq+(2*ml+mu+1)*neq+2*(neq/(ml+mu+1)+1)

      length=1
      open(unit=18,file='te.dat')
```

```
      write(18,*) tend
      close(unit=18)
      open(unit=19,file='npt.dat')
      write(19,*) npts
      open(unit=20,file='nts.dat')
      write(20,*) nts

      rpi=4.0d0*atan(1.)
      xl = 0.0d0
      xr = 1.0d0
      open(unit=28,file='xr.dat')
      write(28,*) xr
      open(unit=29,file='xl.dat')
      write(29,*) xl

      do 49 i=1,npts
       x(i)=xl+real(i-1.)*(xr-xl)/(npts-1.)
49    continue

C      PRINT *,'space gridpoints for time-gridpoint=0 are calculated'
C      PRINT *,x

c     INITIAL CONDITIONS

      rpi=4.*atan(1.)
      b1=0.250d0
      b2=0.750d0
      do i=1,npts
      y(3*i)=x(i)
      enddo

      do 50 i=1,npts
C       Gaussian initial condition for u1:
C        y(3*i-2) = uexact(x(i),b1,0.0d0)
C       Sine-function initial condition for u1:
        y(3*i-2) = uexact1(x,y,npts,i)
C       Gaussian initial condition for u2:
C        y(3*i-1) = uexact(x(i),b2,0.0d0)
C       Sine-function initial condition for u2:
        y(3*i-1) = uexact2(x,y,npts,i)
        y(3*i)=x(i)
50      continue

C      PRINT *,'initial values u1, u2 for time-gridpoint=0 are calculated'
C      PRINT *,y
      PRINT *,'Time Gridpont is =   0'

      xuname='UPLOT0.dat'
      open(unit=37,file=xuname)
      do 601 i=1,npts
        write(37,*) sngl(x(i)),sngl(y(3*i-2)),sngl(y(3*i-1))
601     continue
      close(unit=37)

      open(unit=16,file='Grid.dat')

c        write grid points to file
```

```
       do 88 i=1,npts
*** als niet dt in plaatje.m wil aanpassen, dan k->k*dt en in
*** plaatje.m n*dt->n.........
          write(16,*) i,k,sngl(y(3*i)),sngl(y(3*i-2)),sngl(y(3*i-1))
88        continue

C        Calculation of Integrals
       open(unit=54,file='Integrals.dat')

         Integ1=0.50d0*(y(6)*y(1)+(1.0d0-y(3*npts-3))*y(3*npts-2)
     >     +Integral1(y,npts))
         Integ2=0.50d0*(y(6)*y(2)+(1.0d0-y(3*npts-3))*y(3*npts-1)
     >     +Integral2(y,npts))
          write(54,*) k,sngl(Integ1),sngl(Integ2)
c-----------------------------------------------------------------------
c      MAIN LOOP

c      calculating new x- and u-values
       nprint = 9
       do 499 k=1,nprint,1
 499    tprint(k) = k*tend/(nprint+1)
       kk = 1
       do 500 k=1,nts
         t=(k-1)*dt
         tout=k*dt

       PRINT *,'Time Gridpont is =  ',k

          call ddassl(resid,neq,t,y,ydot,tout,info,rtol,atol,
     >               idid,rwork,lrw,iwork,liw,rpar,ipar,jac)

       if ( kk .gt. 9 ) goto 501
       if ( (abs(tprint(kk)-tout))/tout .le. 1.e-3) then
          write(uplnm,'(a,i1,a)') 'UPLOT', kk, '.dat'
       OPEN (UNIT= 13, FILE=uplnm)

c     plot x,u-values:
          do 165 i=1,npts
          write(13,*) sngl(y(3*i)),sngl(y(3*i-2)),sngl(y(3*i-1))
 165       continue
          CLOSE( 13 )
          kk = kk + 1
       endif

c       write grid points to file
501    do 450 i=1,npts
          write(16,*) i,k,sngl(y(3*i)),sngl(y(3*i-2)),sngl(y(3*i-1))
450       continue

C       write Integrals to file
         Integ1=0.50d0*(y(6)*y(1)+(1.0d0-y(3*npts-3))*y(3*npts-2)
     >     +Integral1(y,npts))
         Integ2=0.50d0*(y(6)*y(2)+(1.0d0-y(3*npts-3))*y(3*npts-1)
     >     +Integral2(y,npts))
          write(54,*) k,sngl(Integ1),sngl(Integ2)

c       next time step
```

```
500       continue
        close(unit=54)
        close(unit=16)

c-------------------------------------------------------------------------

c         write output to file
        open(unit=17,file=outname)
        do 600 i=1,npts
          write(17,*) sngl(y(3*i)),sngl(y(3*i-2)),sngl(y(3*i-1))
600       continue
        close(unit=17)


C         write grid points to file
         do 452 i=1,npts
*          print *,tout,y(3*i),y(3*i-2),y(3*i-1)
           write(16,*) i,k,sngl(y(3*i)),sngl(y(3*i-2)),sngl(y(3*i-1))
452        continue

C         write Integrals to file
          Integ1=0.50d0*(y(6)*y(1)+(1.0d0-y(3*npts-3))*y(3*npts-2)
     >      +Integral1(y,npts))
          Integ2=0.50d0*(y(6)*y(2)+(1.0d0-y(3*npts-3))*y(3*npts-1)
     >      +Integral2(y,npts))
          write(54,*) k,sngl(Integ1),sngl(Integ2)
        close(unit=54)
C
Cc   **** next time step
C500      continue
C
        close(unit=16)

        end

C       Extra functions

        double precision function Integral1(y,npts)
        double precision y(3*npts),In1
        In1=0
        do v=2,npts-1
        In1=In1+(y(3*v+3)-y(3*v-3))*y(3*v-2)
        enddo
        Integral1=In1
        return
        end

        double precision function Integral2(y,npts)
        double precision y(3*npts),In2
        In2=0
        do vv=2,npts-1
        In2=In2+(y(3*vv+3)-y(3*vv-3))*y(3*vv-1)
        enddo
        Integral2=In2
        return
        end
```

```
      double precision function alfaa1(y,npts)
      double precision y(3*npts),alf1
      alf1=0
      do f=2,npts-1
      alf1=alf1+(y(3*f+3)-y(3*f-3))*abs(y(3*(f+1)-2)-y(3*f-2))
      enddo
      alfaa1=alf1
      return
      end

      double precision function alfaa2(y,npts)
      double precision y(3*npts),alf2
      alf2=0
      do ff=2,npts-1
      alf2=alf2+(y(3*ff+3)-y(3*ff-3))*abs(y(3*(ff+1)-1)-y(3*ff-1))
      enddo
      alfaa2=alf2
      return
      end

      double precision function uexact(x,b,t)
      double precision x,t,rpi,b,c,pi,aa
      rpi = 4.0d0*atan(1.)
      pi=3.14159265358979323846264300
      c=0.0410d0
      aa=1.0d0/(c*sqrt(2.0d0*pi))
      uexact = aa*exp((-(x-b)*(x-b))/(2*c*c))
      return
      end

      double precision function uexact1(x,y,npts,i)
      double precision pi
      double precision y(3*npts), x(npts), Integral1
      integer i
      pi=3.14159265358979323846264300
      do iii=1,npts
      if ((x(iii) .le. 0.15) .OR. (x(iii) .ge. 0.852)) then
         y(3*iii-2)=0.0d0
      else
         y(3*iii-2)=sin(8*pi*x(iii)+1.50d0)+1.40d0
C          y(3*iii-2)=sin(8*pi*x(iii)+1.50d0 +0.30d0)+1.40d0
C       PRINT *,'exact1'
      endif
      enddo
      Integ1=0.50d0*(y(6)*y(1)+(1.0d0-y(3*npts-3))*y(3*npts-2)
     >     +Integral1(y,npts))
      do iii=1,npts
      y(3*iii-2)=y(3*iii-2)/Integ1
      enddo
      uexact1=y(3*i-2)
      return
      end

      double precision function uexact2(x,y,npts,i)
      double precision pi
      double precision y(3*npts), x(npts), Integral2
      integer i
```

```
      pi=3.14159265358979323846264430d0
      do ii=1,npts
      if ((x(ii) .le. 0.02) .OR. (x(ii) .ge. 0.98)) then
         y(3*ii-1)=0.0d0
      else
C          y(3*ii-1)=sin(8*pi*x(ii)+1.50d0)+1.40d0
        y(3*ii-1)=sin(8*pi*x(ii)+1.50d0+pi)+1.40d0
C       PRINT *,'exact2'
      endif
      enddo
      Integ2=0.50d0*(y(6)*y(2)+(1.0d0-y(3*npts-3))*y(3*npts-1)
     >     +Integral2(y,npts))
      do ii=1,npts
      y(3*ii-1)=y(3*ii-1)/Integ2
      enddo
      uexact2=y(3*i-1)
      return
      end

      double precision function SUMF1(y,zz,dx,npts)
      double precision y(3*npts),FF1
      real dx(npts-1)
      integer zz
      FF1=0
      do 998, l=2,zz-1
      FF1=FF1+(dx(l-1)+dx(l))*y(3*l-1)
998   continue
      SUMF1=FF1
      return
      end

      double precision function SUMF2(y,zzz,dx,npts)
      double precision y(3*npts),FF2
      real dx(npts-1)
      integer zzz
      FF2=0
      do 997, ll=2,zzz-1
      FF2=FF2+(dx(ll-1)+dx(ll))*y(3*ll-2)
997   continue
      SUMF2=FF2
      return
      end

      double precision function SUMP1(y,dx,F1,npts,a1)
      double precision y(3*npts),a1,F1(npts),PP1
      real dx(npts-1)
      PP1=0
      do 996, q=2,npts-1
      PP1=PP1+(dx(q-1)+dx(q))*y(3*q-2)*v(y(3*q),a1)*F1(q)
996   continue
      SUMP1=PP1
      return
      end

      double precision function SUMP2(y,dx,F2,npts,a2)
      double precision y(3*npts),a2,F2(npts),PP2
      real dx(npts-1)
```

```
      PP2=0
      do 995, qq=2,npts-1
      PP2=PP2+(dx(qq-1)+dx(qq))*y(3*qq-1)*v(y(3*qq),a2)*F2(qq)
995   continue
      SUMP2=PP2
      return
      end

      double precision function v(xx,aa)
      double precision xx,aa
      if (xx .lt. 0.0d0) then
      v=abs(xx)**aa
      else
      v=xx**aa
      endif
      return
      end


      subroutine resid(t,y,ydot,r,ires,rpar,ipar)
      parameter(npts=201)
      integer n
      dimension y(3*npts),ydot(3*npts),r(3*npts)
      dimension dx(npts-1),up(npts),u2p(npts)
      dimension upr(npts,5),wee(npts,3)
      double precision t,y,ydot,r,rpar,upr,wee
      double precision x,u,ux,uxx,uxxxx,ux6
      double precision alfax,taux,rkappax,rmux,rohx,sigx,betax
      double precision emm,ef,es,uexact
      double precision rwork(100000),factor,ffactor
      double precision alf,bet,gam,delt,dd,rk,xscale,rpi
      double precision hi,him1,hip1,hip2,delta1,epsil1
      double precision alfa2,beta2,gamma2,delta2,epsil2
CCC my variables
      real dx
      double precision alfa11,beta11,gamma11,alfa12,beta12,gamma12
      double precision ux1,ux2,utwo1,utwo2
      double precision duim11x,duim12x,duix1,duix2
      double precision e1,e2,e,a1,a2,z
      integer mm,kkk,s
      dimension F1(npts),F2(npts)
      double precision F1,F2,P1,P2
      double precision aF,aaF1,aaF2
      double precision v,SUMF1,SUMF2,SUMP1,SUMP2

      common /needed/ alpha,dt,beta,k,gamma
      common /werk/ rwork

      rpi=4.0d0*atan(1.)


      r(3) = ydot(3) - 0.0d0
cc special treatment for second x:
      r(6) = 0.0d0+ydot(3)-2.0d0*ydot(6)+ydot(9)

C   REALLOCATION OF SPATIAL GRIDPOINTS + MONITOR FUNCTION
C   Use alfax as monitor function:
```

```
C    ( for fixed grid method set alfax=0.0d0 )
       alfax = 1.0d0
C    Use alfax2 as monitor function:
C       alfax = 0.50d0*(y(6)*abs(y(4)-y(1))
C      >    +(1.0d0-y(3*npts-3))*abs(y(3*npts-2))+alfaa1(y,npts)
C      >    +y(6)*abs(y(5)-y(2))
C      >    +(1.0d0-y(3*npts-3))*abs(y(3*npts-1))+alfaa2(y,npts))
       taux = 0.001d0
       rkappax = 2.0d0
       rmux = rkappax*(rkappax+1.0d0)
       rohx = taux*rmux
       sigx = taux*(1.0d0+2.0d0*rmux)
cc general grid-equations ("xdot"!):
        do 312 i=9,3*npts-6,3
         dxim2x=y(i-3)-y(i-6)
         dxim1x=y(i)-y(i-3)
         dxix=y(i+3)-y(i)
         dxip1x=y(i+6)-y(i+3)
         duim11x=y(i-2)-y(i-5)
         duix1=y(i+1)-y(i-2)
         duim12x=y(i-1)-y(i-4)
         duix2=y(i+2)-y(i-1)
C    if use alfax as monitor function then also this is needed:
         emim1x=sqrt(1.0d0+alfax*((((duim11x+duim12x)/2.0d0)/dxim1x)**2))
         emix=sqrt(1.0d0+alfax*((((duix1+duix2)/2.0d0)/dxix)**2))
C    if use alfax2 as monitor function then also this is needed:
C         emim1x=alfax+abs(y(i-2)-y(i-5))+abs(y(i-1)-y(i-4))
C         emix=alfax+abs(y(i+1)-y(i-2))+abs(y(i+2)-y(i-1))
         a1x=rohx/(emim1x*dxim2x**2)
         a2x=rohx/(emix*dxim1x**2)
         a3x=rohx/(emim1x*dxix**2)
c NB a4=a2
         a5x=rohx/(emix*dxip1x**2)
         b1x=sigx/(emim1x*dxim1x**2)
         b2x=sigx/(emix*dxix**2)
         enslim1x=-rmux/dxix+(1.0d0+2.0d0*rmux)/dxim1x-rmux/dxim2x
         enslix=-rmux/dxip1x+(1.0d0+2.0d0*rmux)/dxix-rmux/dxim1x
 312     r(i)=enslix/emix-enslim1x/emim1x+a1x*ydot(i-6)-
     >        (a2x+b1x+a1x)*ydot(i-3)+(a3x+b1x+b2x+a2x)*ydot(i)-
     >        (a3x+b2x+a5x)*ydot(i+3)+a5x*ydot(i+6)
* 312    r(i)=ydot(i)-0.0d0

cc special treatment for lastminusone x:
       r(3*npts-3) = 0.0d0+ydot(3*npts-6)-2.0d0*ydot(3*npts-3)+
     >          ydot(3*npts)
       r(3*npts) = ydot(3*npts) - 0.0d0

C       PRINT *,'x(i) for new time gridpoint are calculated'
C       PRINT *,y

       do i=2,npts
       if (y(3*i) .lt. 0.0d0) then
       PRINT *,'x(',i,')= ',y(3*i)
C       Pause
       endif
       enddo
       do i=1,npts-1
```

```
      if (y(3*i) .gt. 1.0d0) then
      PRINT *,'x(',i,')= ',y(3*i)
C      Pause
      endif
      enddo

      dx(1)=y(6)
      do i=2,npts-2
      dx(i)=y(3*i+3)-y(3*i)
      enddo
      dx(npts-1)=1.0d0-y(3*npts-3)

C      PRINT *,'dx(i)s for new time gridpoint are calculated'
C      PRINT *,dx

      a1=0.50d0
      a2=0.50d0
      e1=1.0d0/(10.0d0**5.0d0)
      e2=1.0d0/(10.0d0**5.0d0)
      e=min(e1,e2)
      open(unit=50,file='alfa1.dat')
      write(50,*) a1
      close(unit=50)
      open(unit=51,file='alfa2.dat')
      write(51,*) a2
      close(unit=51)
      open(unit=52,file='epsilon1.dat')
      write(52,*) e1
      close(unit=52)
      open(unit=53,file='epsilon2.dat')
      write(53,*) e2
      close(unit=53)

      do j=1,npts
      F1(j)=0.0d0
      F2(j)=0.0d0
      enddo
      F1(1)=1.0d0
      F2(1)=1.0d0
      F1(npts)=0.0d0
      F2(npts)=0.0d0

      do 994, j=2,npts-1
CCC Calculation of gridpoints z in [xk,xs]
      z=1.0d0-y(3*j)
      mm=1
999   if (z .gt. y(3*mm)) then
      mm=mm+1
      goto 999
      endif
      s=mm
      kkk=mm-1

C      PRINT *,'z=',z,'   s=',s,'   kkk=',kkk,'   y(3*kkk)=',
C     >              y(3*kkk),'   y(3*s)=',y(3*s)

CCC Calculation of values F1 and F2:
```

```
      if (z .eq. y(3*s)) then
C      PRINT *,'z=y(3*s)'
      F1(j)=0.50d0*(dx(1)*y(2)+dx(s-1)*y(3*s-1)
     >                +SUMF1(y,s,dx,npts))
      F2(j)=0.50d0*(dx(1)*y(1)+dx(s-1)*y(3*s-2)
     >                +SUMF2(y,s,dx,npts))
      else
      if (kkk .eq. 1) then
C      PRINT *,'z is in (y(3*kkk),y(3*s))'
      aF=((z-0.0d0)*(z-0.0d0))/(2.0d0*dx(kkk))
      aaF1=(z-0.0d0)*y(3*kkk-1)+aF*(y(3*s-1)-y(3*kkk-1))
      aaF2=(z-0.0d0)*y(3*kkk-2)+aF*(y(3*s-2)-y(3*kkk-2))
      F1(j)=aaF1
      F2(j)=aaF2
      else
C      PRINT *,'z is in (y(3*kkk),y(3*s))'
      aF=((z-y(3*kkk))*(z-y(3*kkk)))/(2.0d0*dx(kkk))
      aaF1=(z-y(3*kkk))*y(3*kkk-1)+aF*(y(3*s-1)-y(3*kkk-1))
      aaF2=(z-y(3*kkk))*y(3*kkk-2)+aF*(y(3*s-2)-y(3*kkk-2))
      F1(j)=0.50d0*(dx(1)*y(2)+dx(kkk-1)*y(3*kkk-1)
     >                +SUMF1(y,kkk,dx,npts))+aaF1
      F2(j)=0.50d0*(dx(1)*y(1)+dx(kkk-1)*y(3*kkk-2)
     >                +SUMF2(y,kkk,dx,npts))+aaF2
      endif
      endif
994   continue

C      PRINT *,'F1 and F2 for new time gridpoint are calculated'
C      PRINT *,'F1', F1
C      PRINT *,'F2', F2

CCC Calculation of values P1 and P2:
      P1=0.50d0*(dx(1)*y(1)*v(y(3),a1)*F1(1)
     >    +dx(npts-1)*y(3*npts-2)*v(y(3*npts),a1)*F1(npts)
     >    +SUMP1(y,dx,F1,npts,a1))
      P2=0.50d0*(dx(1)*y(2)*v(y(3),a2)*F2(1)
     >    +dx(npts-1)*y(3*npts-1)*v(y(3*npts),a2)*F2(npts)
     >    +SUMP2(y,dx,F2,npts,a2))

C      PRINT *,'P1 and P2 for new time gridpoint are calculated',P1,P2

      do 300, i=2,npts-1
CCC Calculation of values u1xx and u2xx and more:
      ux1=(y(3*i+1)-y(3*i-5))/(y(3*i+3)-y(3*i-3))
      alfa11=dx(i)+dx(i-1)
      beta11=dx(i-1)*y(3*i+1)-alfa11*y(3*i-2)+dx(i)*y(3*i-5)
      gamma11=dx(i)*dx(i-1)*(dx(i)+dx(i-1))
      utwo1=2*(beta11/gamma11)

      ux2=(y(3*i+2)-y(3*i-4))/(y(3*i+3)-y(3*i-3))
      alfa12=dx(i)+dx(i-1)
      beta12=dx(i-1)*y(3*i+2)-alfa12*y(3*i-1)+dx(i)*y(3*i-4)
      gamma12=dx(i)*dx(i-1)*(dx(i)+dx(i-1))
      utwo2=2*(beta12/gamma12)

      r(3*i-2)=ydot(3*i-2)-ux1*ydot(3*i)-(e1/e)*utwo1
     >    -(1.0d0/e)*y(3*i-2)*(v(y(3*i),a1)*F1(i)-P1)
```

```
      r(3*i-1)=ydot(3*i-1)-ux2*ydot(3*i)-(e2/e)*utwo2
     >     -(1.0d0/e)*y(3*i-1)*(v(y(3*i),a2)*F2(i)-P2)


300       continue


CC hom. Diri./Neum.:
      r(1) = y(1) - 0.0d0
      r(2) = y(2) - 0.0d0


CC hom. Diri./Neum.:
      r(3*npts-2) = y(3*npts-2) - 0.0d0
      r(3*npts-1) = y(3*npts-1) - 0.0d0

C      PRINT *,'r(i) for new time gridpoint are calculated'

      return
      end
```