A Swampland Constraint on Gravitational Collapse

Thesis by
Himanshu Chaudhary

In Partial Fulfillment of the Requirements for the
degree of
Master of Science, Physics

INDIAN INSTITUTE OF SCIENCE
Bangalore, India

2020
Defended 1 July 2020

ACKNOWLEDGEMENTS

I would like to thank my advisor Professor Chethan Krishnan for his guidance and constant encouragement. His insights about the problem helped me in avoiding a lot of pitfalls and wasted effort, allowing me to focus on the most important aspects of the problem. I would also like to thank Professor Jun-Qi Guo for answering a lot of questions about the numerical parts of the problem.

I would also like to thank my family for their continuous support and providing a mental sanctuary away from all the pressure that comes with academia. Finally, I would like to thank my friends and batch-mates for all the fruitful discussion that we had about the project and other courses.

ABSTRACT

Gravitational collapse of a scalar field is a well studied phenomenon but, most of these studies are narrowly focussed on studying the field behavior close to criticality and its implications. In this thesis we take a broader perspective and study the gravitational collapse in the context of the Swampland bounds and Effective field theories. We are particularly interested in the Swampland distance bound, according to which a classical solution where scalar fields move by an $O(1)$ range (in Plank units) signals the breakdown of Effective Field Theory.

We show using numerical simulations that, while collapsing into a black hole, a scalar field minimally coupled to gravity generically moves by $O(1)$ range. This result highlights the sharp tension between the swampland distance bound and the expectation that effective field theory should hold at the horizon of a black hole. We also show that this $O(1)$ scalar field movement is quite independent of the initial data by evolving the system with different initial profiles. Because most of the research in gravitational collapse is done numerically we also give a detailed description of the numerical algorithms used and various pitfalls one should avoid.

TABLE OF CONTENTS

Acknowledgements ii
Abstract iii
Table of Contents iv
List of Illustrations v
List of Tables vi
Chapter I: Introduction 1
1.1 Gravitational collapse of scalar fields 1
1.2 Swampland conditions 1
Chapter II: Gravitational collapse of a scalar field in general relativity 4
2.1 Scalar collapse equations 5
2.2 Boundary and Initial conditions 6
Chapter III: Numerically solving the scalar field collapse 9
3.1 Finite difference methods 9
3.2 Solving scalar collapse equations 17
3.3 How do we check the accuracy of the numerical results? 25
Chapter IV: Profiles used and results 27
Chapter V: Conclusions and Future Directions 33
Bibliography 34

LIST OF ILLUSTRATIONS

Number *Page*

3.1 Error in the approximation of the first derivative of x^3 by first order finite difference methods. 13
3.2 Error in the approximation of the derivative of x^3 by second order finite difference methods. 14
3.3 This plot show error of using finite difference to approximate the derivative of $\frac{1}{x}$ vs the step size at $x = 1$. 15
3.4 This plot show error of using finite difference to approximate the derivative of $\frac{1}{x}$ vs the step size at $x = 0.1$. 16
3.5 This plot show error of using finite difference to approximate the derivative of $\frac{1}{x}$ vs the step size at $x = 0.01$. 16
3.6 This plot show error of using finite difference to approximate the derivative of $\frac{1}{x}$ vs the order of the approximation. 17
3.7 Uniform discretization of time-space into a grid with space step size δx and time step size δt. 19
3.8 Stencil used and the computational grid 23
3.9 This figure shows the grid points that can affect the grid point X at $(6\,\delta t, \delta x)$. Grid points in blue can affect the grid point X while grid points in red can not. 25
4.1 Evolution of ψ from an initial Gaussian profile 29
4.2 Evolution of ψ from an initial Maxwell profile(Modified Gaussian) profile 30
4.3 Evolution of ψ from an initial tanh profile 31
4.4 Evolution of ψ from an initial shell profile 32

LIST OF TABLES

Number *Page*

3.1 This table shows the largest number that can be represented by a particular type of float (* rounded off to two decimal places). Precision denotes the number of significant decimal digits that can be represented by a float type. 10

Chapter 1

INTRODUCTION

The main focus of this study was to look at the gravitational collapse of a scalar field in the light of swampland distance conjecture. In the following sections we will briefly touch upon these topics and give suitable references for further explorations.

1.1 Gravitational collapse of scalar fields

Critical collapse of scalar fields in gravitation is a well studied phenomenon (Gundlach and Martin-Garcia, 2007), and there are many interesting results mostly based on the numerical simulations. Gravitational collapse and the numerical methods used to study it will be described in detail in a later chapter. Here we will just mention one result that will be required to put things that we are going to discuss into perspective.

Assume that the initial configuration of the scalar field is described by a function with n adjustable parameters that we are free to choose. Then for each one of those n parameters (that are not zero modes) there will be a corresponding threshold, crossing which will lead to the formation of a black hole.

For example, if the initial scalar field configuration is a gaussian shell with fixed mean and variance, then there will a critical value of the amplitude (let's call it A_0) above which black hole formation will take place. Although, if we keep the amplitude (A) below A_0 then the scalar field will disperse to infinity. The same holds true for the mean and the variance.

An important thing that we will show via our simulations is that even if the initial value of the amplitude A and the critical value A_0 are both much smaller than 1, the scalar field will still move by $O(1)$ in supercritical cases. We will expand upon and give examples of this in later chapters. For this chapter it is enough to remember that in general relativity scalar fields can move $O(1)$ distance in the scalar field space, starting from much smaller values.

1.2 Swampland conditions

Before we can appreciate what are swampland conditions we need to understand a little bit about effective field theories and the role that they play in Physics.

Effective field theories

Physical phenomenon occurs at various scales, sometimes we are only interested in the results at a particular scale. For example, if we are interested in the collision of two slow moving bodies then Newtonian mechanics is more than enough, we do not need to consider Relativistic mechanics. The scale of interest in this case is the kinetic energy (speed) of the bodies and for speeds much smaller than the speed of light we can consider Newtonian mechanics as an "effective theory" of Relativistic mechanics. When one usually talks about scales in an effective theory that scale is energy, here we used speed as our scale because the masses are fixed. Such theories in the context of QFT are known as "effective field theories".

A prime example of an effective field theory is Fermi theory of weak interactions. It was phenomenologically constructed by Fermi as a modification to QED which accounted for the neutron decay. Now that we have the Standard model, we understand that the Fermi theory is an effective field theory valid for energy scales much smaller than the mass of W boson.

Effective theories are ubiquitous in Physics, whether it is for the ease of calculations (it is not an efficient use of time to track motion of a football using String Theory) or because they are a part of how Physics progresses (nobody could have constructed the Standard model lagrangian without the insights gained from all the effective field theories). If we have a more "complete" theory, then it is a comparatively easy matter to generate effective theories as per our needs. At this point one can ask a question, can all field theories be extended into more general and complete theories? The exact question that we are interested in is what kind of effective field theories can be completed into quantum gravity in the high energy or UV regime? This is the question that swampland conditions try to answer.

The landscape and the swampland

A lot of low energy effective field theories, each corresponding to a different vacuum, can be constructed from the string theory. The collection of such effective field theories constructed from the string theory is called the landscape. At the same time, there are a lot of other potentially complete low energy theories that when coupled to gravity lead to inconsistencies, the collection of all such theories is called the swampland.

In other words, we can complete a low energy theory from the landscape into a UV theory of quantum gravity. But, trying to UV complete a theory form the swampland

into a theory of quantum gravity leads to inconsistencies.

Swampland distance conjecture

Whether an effective field theory exists in the swampland or the landscape is the question that swampland conditions try to answer. Although, there are quite a few of these conditions most of them do not have any formal proof from microscopic physics perspective and are thus treated as conjectures. The confidence that we have in these conjectures mostly comes from the observations based on the few known vacua of the string theory and quantum gravity arguments (Brennan, Carta, and Vafa, 2017).

One of these conjectures is the swampland distance conjecture (SDC), which states that an effective field theory that has gravity and scalar fields cannot be reliable in the regimes where the scalar field moves beyond an $O(1)$ range.

As we will see in our results, $O(1)$ field movement is observed in the case of black hole formation due to the scalar field collapse. This tension between the SDC and gravitational collapse of a scalar field in general relativity is what we will be exploring in this thesis.

Chapter 2

GRAVITATIONAL COLLAPSE OF A SCALAR FIELD IN GENERAL RELATIVITY

If we look at the evolution of a minimally coupled massless scalar field in general relativity then it will either disperse to infinity or collapse into a black hole depending on the initial data. In addition to that, if we vary any one parameter p of the initial data then we can find a threshold p^* crossing which will lead to the formation of a black hole. But, if we are exactly at the threshold p^* then we get critical collapse along with a lot of interesting properties.

Firstly, there is a mass scaling law of the black hole formed due to the collapse

$$M \propto (p - p^*)^\gamma$$

Here M is the mass of the black hole, and p^* is the threshold value which depends on the function that describes in the initial data, so it will be different based on whether our initial data is described by a gaussian or a square. The important point to note here is that γ is universal or independent of the the type of the initial data.

This universality has a very important implication, that one can create a naked singularity. This can be seen in the scaling law, according to which we can create black holes with arbitrarily small masses by taking the value of p to be arbitrarily close to p^*.

Another important property of scalar field collapse close to criticality is called self-similarity. Self-similarity implies that close to the origin the scalar field follows a scaling relation,

$$\psi(t, x) = \psi(e^{At}, e^{Ax})$$

This scaling makes it very hard to study the behavior of the scalar field close to the origin, because it starts to oscillate very rapidly during the critical collapse.

First studies of scalar field collapse were done by Choptuik (Gundlach and Martin-Garcia, 2007), but the focus of his study was to understand the critical behavior.

He showed the two aforementioned properties of critical collapse for a spherically symmetric scalar field via numerical simulations. Since then a lot of research has been done in this field but most of it was focused on understanding and testing these properties for more general cases.

In our case we were not so much interested in the behavior of the field close to criticality. What we wanted to understand was how much does the scalar field moves during the black hole formation so we can relate it to the swampland distance bound.

2.1 Scalar collapse equations

To get the equations of motion we will closely follow the approach taken by (Guo and Zhang, 2019). We want to study the collapse of minimally coupled scalar field for which the action takes the form,

$$S = \frac{1}{8\pi G}\int d^4x\sqrt{-g}\left(\frac{1}{2}R - \frac{1}{2}\partial_\mu\phi\partial^\mu\phi\right) \tag{2.1}$$

Because, we want to study the scalar fields near the origin a good choice of coordinates is the double-null coordinates,

$$ds^2 = e^{-2\sigma(t,x)}\left(-dt^2 + dx^2\right) + r^2(t,x)d\Omega^2 \tag{2.2}$$

here σ and r are a function of (t, c).

We can now write our equations of motion as,

$$r\left(-r_{,tt} + r_{,xx}\right) - r_{,t}^2 + r_x^2 = e^{-2\sigma} \tag{2.3}$$

$$-\sigma_{,tt} + \sigma_{,xx} + \frac{r_{,tt} - r_{,xx}}{r} + 4\pi\left(\psi_{,t}^2 - \psi_{,x}^2\right) = 0 \tag{2.4}$$

$$-\psi_{,tt} + \psi_{,xx} + \frac{2}{r}\left(-r_{,t}\psi_{,t} + r_{,x}\psi_{,x}\right) = 0 \tag{2.5}$$

In addition to these evolution equations we also get two constraint equations,

$$r_{,tx} + r_{,t}\sigma_{,x} + r_{,x}\sigma_{,t} + 4\pi r\psi_{,t}\psi_{,x} = 0 \tag{2.6}$$

$$r_{,tt} + r_{,xx} + 2r_{,t}\sigma_{,t} + 2r_{,x}\sigma_{,x} + 4\pi r\left(\psi_{,t}^2 + \psi_{,x}^2\right) = 0 \tag{2.7}$$

We will use these equations to solve the initial conditions and also to verify that our numerical simulations are giving sensible results.

Now, we will introduce a new variable m defined by,

$$g^{\mu\nu}r_{,\mu}r_{,\nu} = e^{2\sigma}\left(-r_{,t}^2 + r_{,x}^2\right) \equiv 1 - \frac{2m}{r} \tag{2.8}$$

Defining this extra variable helps with the stability of the numerical solutions. To get the evolution equation for m we differentiate the equation 2.8 with respect to time and use the evolution and constraint equations, which gives us equation 2.12. Using the definition of m we can write the final form of the equations that we will be using to study the scalar collapse,

$$-\psi_{,tt} + \psi_{,xx} + \frac{2}{r}\left(-r_{,t}\psi_{,t} + r_{,x}\psi_{,x}\right) = 0 \tag{2.9}$$

$$-r_{,tt} + r_{,xx} - e^{-2\sigma}\cdot\frac{2m}{r^2} = 0 \tag{2.10}$$

$$-\sigma_{,tt} + \sigma_{,xx} - e^{-2\sigma}\cdot\frac{2m}{r^3} + 4\pi\left(\psi_{,t}^2 - \psi_{,x}^2\right) = 0 \tag{2.11}$$

$$m_{,t} = 4\pi r^2\cdot e^{2\sigma}\left[-\frac{1}{2}r_{,t}\left(\psi_{,t}^2 + \psi_{,x}^2\right) + r_{,x}\psi_{,t}\psi_{,x}\right] \tag{2.12}$$

To get the initial data we also need the spatial derivative of m, which we can get by differentiating the equation 2.8 with respect to x and using the evolution and constraint equations.

$$m_{,x} = 4\pi r^2\cdot e^{2\sigma}\left[\frac{1}{2}r_{,x}\left(\psi_{,t}^2 + \psi_{,x}^2\right) - r_{,t}\psi_{,t}\psi_{,x}\right] \tag{2.13}$$

2.2 Boundary and Initial conditions

There are four variables in our equations r, m, σ, ψ thus we need the initial spatial profile of all four to evolve the system in time but, at time $t = 0$ we only have the freedom to choose the spatial profile of ψ. We can not arbitrarily take the spatial

profiles for r, m, σ, because even at $t = 0$ they have to satisfy the Einstein equations to represent a physical system. The spatial profiles of all these four variables are the initial conditions of our system. To get the initial conditions we will use a simplification and set them to be time symmetric, which implies that,

$$r_{,t} = \sigma_{,t} = \phi_{,t} = \psi_{,t} = 0 \quad \text{at } t = 0 \tag{2.14}$$

Now we will use equations 2.10, 2.6 and 2.13 along with the equations 2.14 to get,

$$r_{,xx} = e^{-2\sigma} \cdot \frac{2m}{r^2} \tag{2.15}$$

$$\sigma_{,x} = -2\pi \cdot \frac{\psi_{,x}^2 \cdot r}{r_{,x}} - e^{-2\sigma} \cdot \frac{m}{r^2 r_{,x}} \tag{2.16}$$

$$m_{,x} = 4\pi r^2 \cdot e^{2\sigma} \left[\frac{1}{2} r_{,x} \cdot \psi_{,x}^2 \right] \tag{2.17}$$

The reason we used equations 2.10, 2.6 and 2.13 was partially motivated by the fact that we wanted the simplest system that can be solved to get the initial conditions. To solve these three ODEs we need the corresponding boundary conditions on r, σ and m at $(0, 0)$. Here we are just going to state the boundary conditions we will be using, to understand the motivation behind this choice refer (Guo, Wang, and Frolov, 2014).

$$r(0, 0) = 0 \tag{2.18}$$

$$r_{,x}(0, 0) = 1 \tag{2.19}$$

$$\sigma(0, 0) = 1 \tag{2.20}$$

$$m(0, 0) = 1 \tag{2.21}$$

To evolve our initial data we also need the boundary conditions at the spatial boundaries. Here we will only derive the boundary conditions at the origin, boundary conditions at the outer boundary will be set by extrapolation (to understand the motivation behind this choice refer to section 3.2).

To get the boundary conditions at the origin we will use regularity arguments. Observe that r is always set to 0 at $x = 0$, this is done to prevent formation of any

kind of cusp at the origin, which give us $r_{,t}(t,0) = 0$ and $r_{,tt}(t,0) = 0$. Now, to ensure that the term $\frac{2}{r}\left(-r_{,t}\psi_{,t} + r_{,x}\psi_{,x}\right)$ from the equation 2.9 is regular at the origin we need that $r_{,x}\psi_{,x} = 0$ because $r_{,t}(t,0)$ is already 0, therefore we have that $\psi_{,x}(t,0) = 0$. Looking at the equation 2.8 we can see that we need $m(t,0) = 0$ to keep the term $\frac{2m}{r}$ regular. Similarly from the equation 2.10 we get that $r_{,xx} = 0$. Finally, at the origin the equation 2.7 becomes $r_{,x}\sigma_{,x} = 0$ which gives us $\sigma_{,x}(t,0) = 0$.

One can see that not all of these boundary conditions are independent. To evolve the initial data we will use the following four boundary conditions at the origin,

$$r(t,0) = 0 \tag{2.22}$$

$$m(t,0) = 0 \tag{2.23}$$

$$\psi_{,x}(t,0) = 0 \tag{2.24}$$

$$\sigma_{,x}(t,0) = 0 \tag{2.25}$$

Chapter 3

NUMERICALLY SOLVING THE SCALAR FIELD COLLAPSE

While studying gravitational collapse of a scalar field we got a set of differential equations. These differential equations were complicated enough that any effort to find an analytical solution was futile. One can always make approximations and get some understanding of the solutions, but to get the complete solution we need to use numerical techniques. In this chapter we will discuss about how to solve differential equations on a computer.

3.1 Finite difference methods

Solving differential equations is required in almost every field of Science and although analytical solutions are the best thing one can get, they are hard to get for any but the simplest systems. Most of the time we have to resort to solving differential equation numerically. There a lot of methods out there developed to solve differential equations, each one of them have their own advantages and dis-advantages. In our case we will be using a class of methods called finite difference methods.

Finite difference methods are among the most intuitive and easy to implement methods of solving differential equations. They are derived using the taylor series and give a discrete approximation of the strong from of a differential equation.

To get some intuition about finite difference methods let us start with a very simple example. These simple examples will demonstrate some of the most important concepts of finite difference computing.

Suppose that we want to find the derivative of $f(x) = x^3$ at $x = 1$. We know that the answer should be 3. But, let us say that we do not know the exact answer and want to approximate the derivative. The simplest way to do this would be using the definition of differentiation,

$$\frac{dy(x_0)}{dx} = \lim_{h \to 0} \frac{y(x_0 + h) - y(x_0)}{h} \tag{3.1}$$

The definition gives us the exact answer in the limit $h \to 0$. What we can do is to take h to be a small non-zero number and see if that gives us a good enough approximation.

$$\frac{dy(x_0)}{dx} \approx \frac{y(x_0 + h) - y(x_0)}{h}, \text{for } h \ll 1 \tag{3.2}$$

We have shown the results of using the equation 3.2 to approximate the derivative of x^3 at $x = 1$ in the figure 3.1. Y axis in the figure is the log error of the approximation and X axis is the log of the step size (h). Figure 3.1 has a lot of stuff going on in it and we will break it down into parts. But, before we do that we will briefly look at how computers store decimal numbers and how does it affect our calculations.

Floating point arithmetic

Floats are how computers internally represent decimal numbers. A larger sized float can handel both a larger number and more significant digits (Table 3.1). Because computers can only stores a fixed number of digits after the decimal point we get floating point errors whenever any calculation is done on these floats. For example we know that $\frac{1}{3}$ has a recurring decimal expansion, but if we want to save $\frac{1}{3}$ in the computer, in its decimal representation, then there will be a small error because we can only store a fixed number of significant digits.

Float type	Largest Number that can be stored*	Precision	How is $\frac{1}{3}$ internally stored
Float32	3.40e+38	6	0.33333334
Float64	1.79e+308	15	0.3333333333333333
Float128	3.36e+4932	18	0.33333333333333333334

Table 3.1: This table shows the largest number that can be represented by a particular type of float (* rounded off to two decimal places). Precision denotes the number of significant decimal digits that can be represented by a float type.

Table 3.1 has the information about different kind of floats used in the graphs. Precision denotes the number of significant digits that a particular type of float can store, for example, float32 can store just 6 digits after the decimal point which is why $\frac{1}{3}$ is stored as 0.33333334. Also, note that the there is a huge difference in the largest number that can be stored between different kind of floats. If we try to store a number larger than the maximum number that a float type can hold, we will get an overflow error and computer will start giving us random garbage. Different languages and libraries deal with such errors in different ways and it is not worth going into the specifics, but the bottom line is that we can not use float64 if we want to work with numbers larger than 1.79×10^{308}.

Most of the modern computers chips use float64 internally and they have CPU circuits that are very efficient in working with float64 numbers. 64 in float64 tells us that this number will use 64bits of memory, similarly float32 numbers will use 32bits of memory; this means that we can store more float32 numbers in the same amount of memory. What this means in practical terms is that a we can store around 10^9 floats in $8GB$ of RAM, or a matrix with approximately 31000 rows and columns. If we are not clever with what we are storing in our RAM there is real danger that we may run out of memory, which will cause an error and our program will stop.

Among the parameters listed in Table 3.1 the most important parameter for any calculation is the precision. As already mentioned whenever we perform any kind of operation on a floating point number there is an error due to the finite precision of floating point numbers. We can see this phenomenon quite easily just by adding 1.0 to $\frac{1}{3}$ stored as a float32. If we add 1.0 to $0.\overline{33}$, 10000 times the expected answer is $10000.\overline{33}$ which in float32 precision should be 10000.3333334. But, we see that the actual answer that we get is 10000.334, that is we have lost 4 significant digits. This loss of accuracy is what we call floating point error and this is the reason why using float32 for numerical computations is a bad idea. If we use float64 instead of float32 we will still lose 4 significant digits but it will not matter so much because we will still have 11 significant digits in the decimal to work with. We should mention here that there is no hard and fast formula for how many digits will we lose per operation and it depends on the language and libraries used as many of them are optimized to reduce floating point errors.

We will discuss one final point we need to discuss before we wrap up this section and go back to developing finite difference algorithms. Given any two floating point numbers, when are they equal to each other?

This question is important because it is related to the precision of floating point numbers. For example, if we try to subtract 1.0000001 from 1.0000007 and they are stored as float32, then we will get some random garbage. The reason being that float32 can not handle more than 6 decimal significant digits properly. The general rule is that one should try to avoid subtracting two numbers that are close and have a lot of significant digits in their mantissa, as that usually gives a lot of error. Just for the sake of clarity, we should mention that here we are not taking about numbers with small magnitude. So, even float32 can easily handle numbers of small magnitude like subtracting 2×10^{-30} from 1×10^{-30} because their mantissa does not have a lot of significant digits. But, at the same time if we use float32

then $2 + 10^{-8}$ and 2 will both be stored as 2.0 inside the computer memory and subtracting them will give zero.

Here we have given all the examples in float32 but the concepts carry over to float64, the only difference being that float64 can handle much smaller number and more significant digits. As already mentioned modern computer CPU's are designed to deal with float64 arithmetic and it is accurate enough for our needs, so we use it for our simulations. Using float128 or larger floats can give much more accuracy but they will be very slow compared to using float64 and such high accuracy is usually not required for most of the applications.

Floating point errors in finite difference approximations

Now that we have some idea about the floating point arithmetic we can dive into the figures and try to understand them. In the figure 3.1 there are three lines, each corresponding to a different sized float used for the calculations.

To approximate the derivative of x^3 at $x = 1$ we have used the formula,

$$\frac{dy(x_0)}{dx} \approx \frac{y(x_0 + h) - y(x_0)}{h}, \text{for } h \ll 1 \tag{3.3}$$

This is called a first order finite difference approximation. To understand why is it called so, we will taylor expand the $y(x)$ around x_0 with step size h

$$y(x_0 + h) = y(x_0) + y'(x_0) \times h + \frac{y''(x_0) \times h^2}{2} + \dots \tag{3.4}$$

If we truncate the series at $O(h^2)$ and rearrange a little we get,

$$y(x_0 + h) - y(x_0) \approx y'(x_0) \times h + \frac{y''(x_0) \times h^2}{2} \tag{3.5}$$

dividing by h we get,

$$y'(x_0) \approx \frac{y(x_0 + h) - y(x_0)}{h} - \frac{y''(x_0) \times h}{2} \tag{3.6}$$

Notice that this is the exact formula that we were using for approximating the derivative except for the presence of the extra term $\frac{y''(x_0) \times h}{2}$. This extra term represents the error in our approximation (there are of course other higher order terms that we ignored, but this term is going to usually dominate them). Because the error falls

off linearly with the step size h we call this a first order approximation. Looking at the figure 3.1 this becomes clear, if the step size is decreased by a factor of two then the error also goes down by a factor of two, i.e. slope of the log(error) vs log(h) line will have a slope 1.

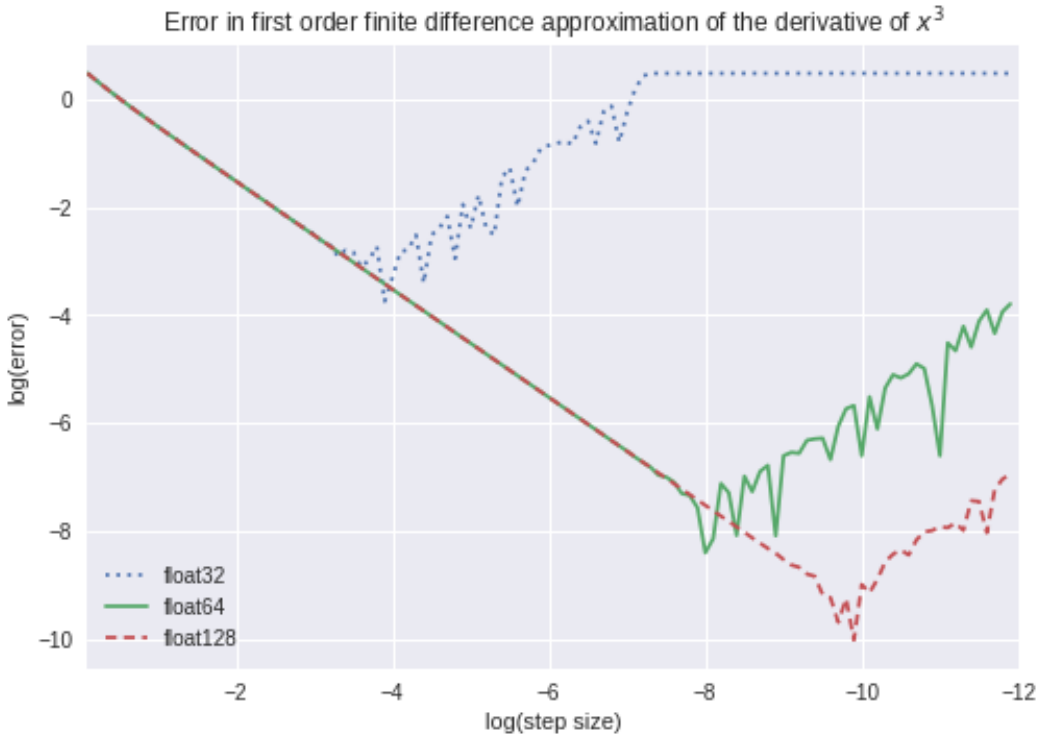


Figure 3.1: This plot show error of using first order finite difference to approximate the derivative of x^3 vs the step size. Observe that the error falls almost linearly until the floating point errors start to dominate, after which it starts to grow erratically. Also, observe that for reasonable step sizes the error falls with a slope of 1, which is why such approximations are called to be of first order.

In the beginning, irrespective of the float size, we have a straight line which turns into a wiggly mess as we keep on decreasing the step size. The reason for this sudden erratic increase in the error is floating point errors. One can clearly see that going form float32 to float64 is a big jump in accuracy. But in going to float128 from float64 the increase in the accuracy is not so dramatic, which is kind of expected as float128 has only 3 more significant digits over float64.

We can also use higher order methods, for example a second order method,

$$y'(x_0) \approx \frac{y(x_0 + h) - y(x_0 - h)}{2 \times h} \tag{3.7}$$

To show that this a second order method one can taylor expand $y(x)$ around x_0 with step size h and $-h$ and plug it back into the equation 3.7. Figure 3.2 shows the log(error) vs log(step size) graph for the second order approximation, notice that the slope of the line is now 2. Using higher order methods can give more accurate results for the same step size but they lead to other issues which we will briefly discuss later.

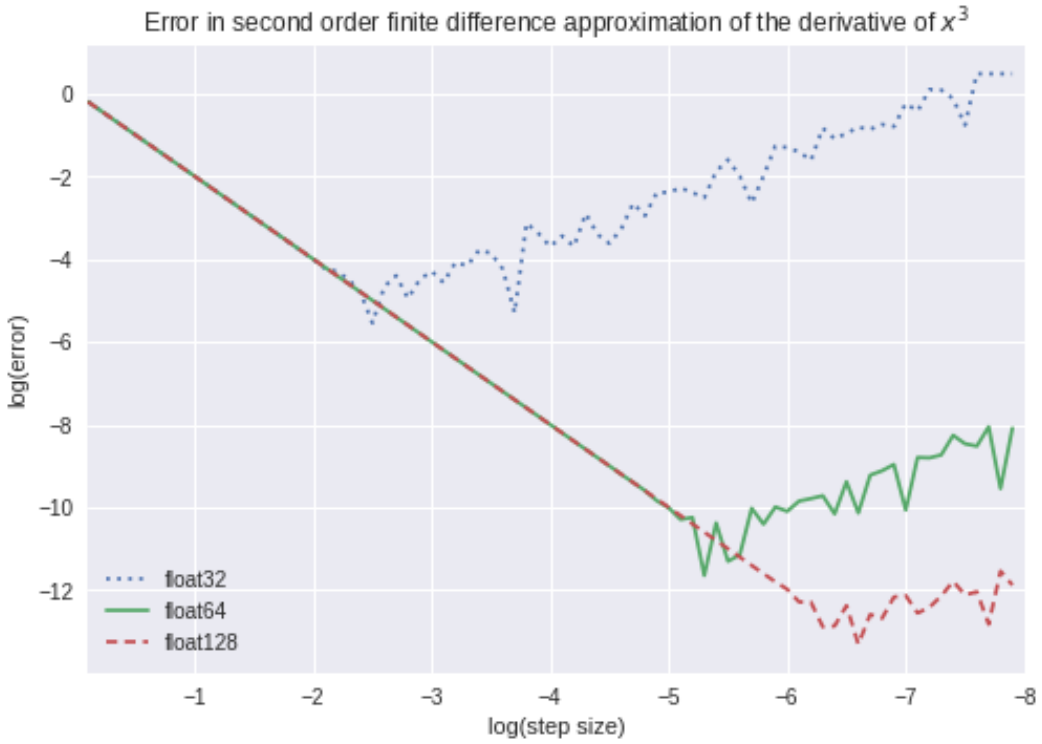


Figure 3.2: This plot show error of using second order finite difference to approximate the derivative of x^3 vs the step size. Observe that the error falls almost linearly until the floating point errors start to dominate, after which it starts to grow erratically. Also, observe that for reasonable step sizes the error falls along a line with slope 2 which is to be expected of a second order approximation.

To give some perspective, while solving the scalar collapse equations we are going to use step size of the order 10^{-5}, thus using float32 is just out of the picture. Also, our spatial domain will be [0,2] which implies that there will be at least 10^5 operations per element (we can not decrease spatial step size without decreasing the time step size at the same time, something we will explore more in the next sections). From the discussions in the section 3.1 we know that there is an error associated with each operation, thus, even though individual errors may be small given the size of calculations we are interested in they can quickly add up and destroy our simulations.

Having discussed the effect of the step size on the error of finite difference method we now turn to another major source of error. While talking about the error in the equation 3.6 we conveniently ignored the dependence of the error on the second derivative $y''(x_0)$. One can ask what will happen if the second derivative is very larger?

This is the question that figures 3.3, 3.4 and 3.5 try to answer. We know that,

$$\frac{d^n}{dx^n}\left(\frac{1}{x}\right) = (-1)^n \frac{n!}{x^{(n+1)}} \tag{3.8}$$

Because, close to the origin the derivatives are themselves very larger the error in our finite difference approximation keeps on increasing with as we get closer to the origin. This can be seen in the figures 3.3, 3.4, 3.5. Notice how we have to take smaller and smaller step size to get the same accuracy as we get closer to the origin.

The approximation at $x = 0.01$ is so bad that even taking the step size to be 10^{-10} is not giving very accurate answer (Figure 3.5).

The crux of the matter is that finite difference methods struggle when the function is changing very rapidly. Something we will see in the results of our simulations.

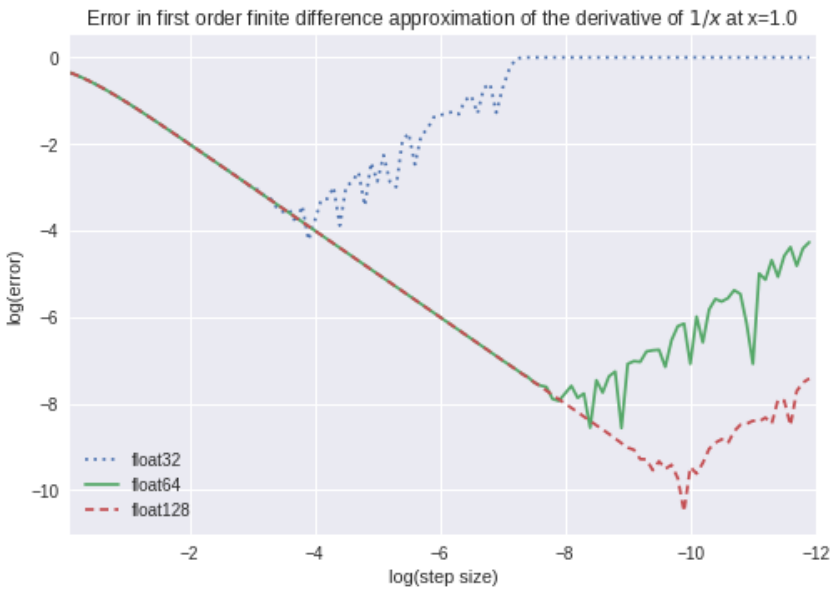


Figure 3.3: This plot show error of using finite difference to approximate the derivative of $\frac{1}{x}$ vs the step size at $x = 1$.

Figure 3.4: This plot show error of using finite difference to approximate the derivative of $\frac{1}{x}$ vs the step size at $x = 0.1$.

Figure 3.5: This plot show error of using finite difference to approximate the derivative of $\frac{1}{x}$ vs the step size at $x = 0.01$.

Although, this situation is still salvageable by using higher order methods as shown in the Figure 3.6. Notice that for the same step size the higher order methods are much more accurate, just by going to second order methods we are able to reduce the errors by a lot. Furthermore, using even higher order methods gives us a diminishing

return which is one of the reasons why we will be using second order methods.

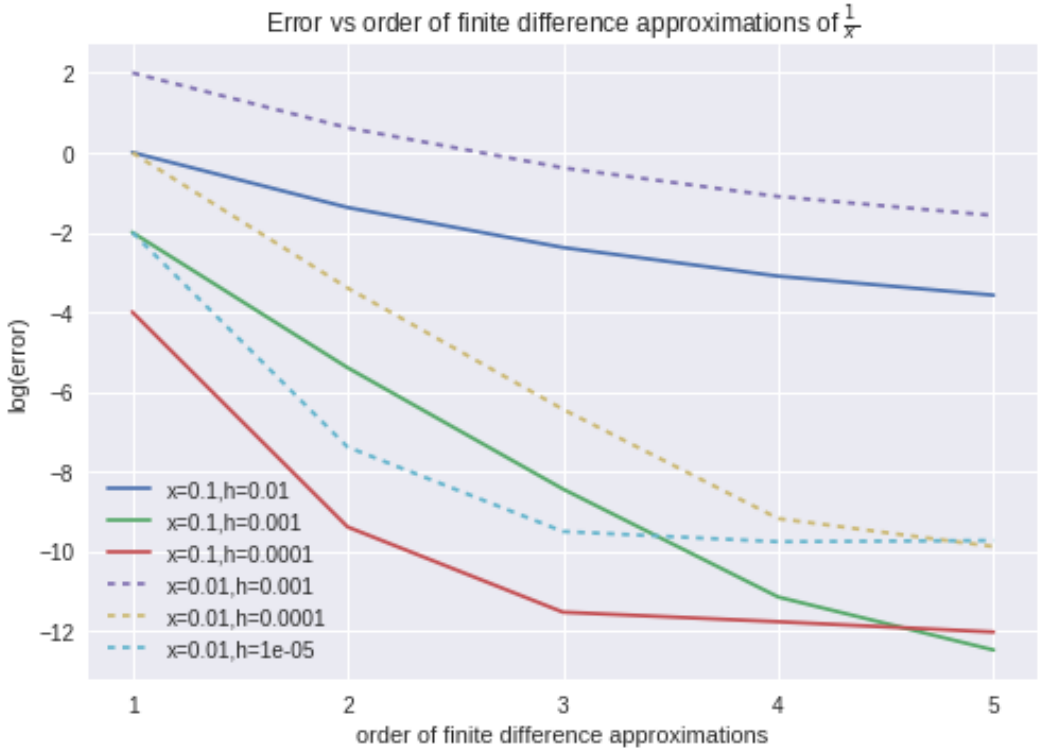


Figure 3.6: This plot show error of using finite difference to approximate the derivative of $\frac{1}{x}$ vs the order of the approximation.

3.2 Solving scalar collapse equations

Now that we have some understanding of how finite difference can be used to approximate derivatives, the next logical step is use it all to solve the differential equations that we have. In particular we want to solve,

$$-r_{,tt} + r_{,xx} - e^{-2\sigma} \cdot \frac{2m}{r^2} = 0 \tag{3.9}$$

$$-\psi_{,tt} + \psi_{,xx} + \frac{2}{r}\left(-r_{,t}\psi_{,t} + r_{,x}\psi_{,x}\right) = 0 \tag{3.10}$$

$$-\sigma_{,tt} + \sigma_{,xx} - e^{-2\sigma} \cdot \frac{2m}{r^3} + 4\pi\left(\psi_{,t}^2 - \psi_{,x}^2\right) = 0 \tag{3.11}$$

$$m_{,t} = 4\pi r^2 \cdot e^{2\sigma}\left[-\frac{1}{2}r_{,t}\left(\psi_{,t}^2 + \psi_{,x}^2\right) + r_{,x}\psi_{,t}\psi_{,x}\right] \tag{3.12}$$

There are two parts to solving the scalar collapse equations:

1. Solve the initial condition problem
2. Evolve the initial configuration in time

In general, solving initial conditions in GR is a very hard thing because we have to deal with elliptical partial differential equations. But in this case we are lucky as we are working in 1D and just need to solve a set of ordinary differential equations, which are much much easier and cheaper to solve.

We will be using finite difference methods to solve the initial conditions and then evolve the system in time, to do that we will first define some terminology and discretize the time-space domain. Discretization of the time-space domain is shown in the figure 3.7, we will use uniform discretization which means that all time (Δt) and space (Δx) step sizes will be the same throughout the grid.

Furthermore, we will denotes the value of a variable $X(t, x)$ on the grid by,

$$X^n_j = X(n\delta t, j\delta x) \tag{3.13}$$

Finite difference approximation of the derivatives $X^n_{j,x}$ and $X^n_{j,t}$ can be written as, (we will stop stop using $\approx$ sign from now on, but it should be clear from the context that we are talking about an approximation)

$$X^n_{j,t} = \frac{X^{n+1}_j - X^{n-1}_j}{2\delta t} \tag{3.14}$$

$$X^n_{j,x} = \frac{X^n_{j+1} - X^n_{j-1}}{2\delta x} \tag{3.15}$$

Similarly, second derivatives can be written as,

$$X^n_{j,tt} = \frac{X^{n+1}_j - 2X^n_j + X^{n-1}_j}{\delta t^2} \tag{3.16}$$

$$X^n_{j,xx} = \frac{X^n_{j+1} - 2X^n_j + X^n_{j-1}}{\delta x^2} \tag{3.17}$$

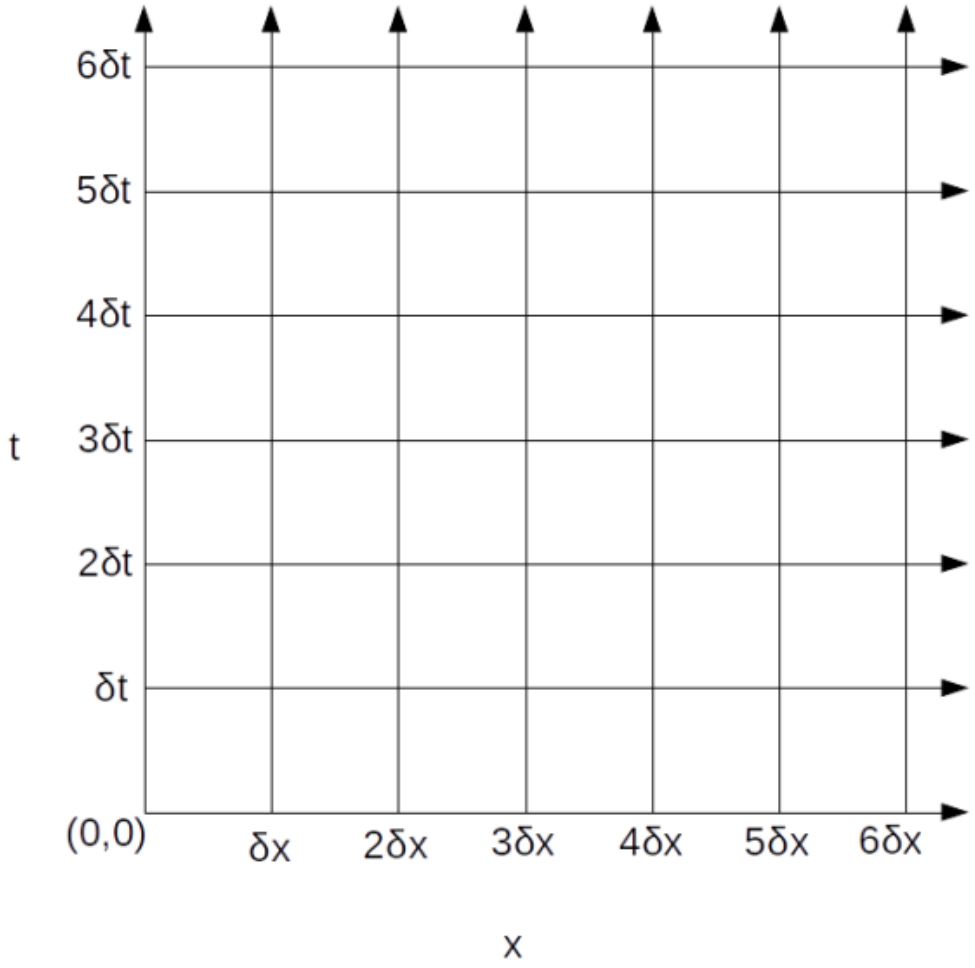


Figure 3.7: Uniform discretization of time-space into a grid with space step size δx and time step size δt.

Solving initial conditions

From the section 2.2 we know the equations that we need to solve,

$$r_{,xx} = e^{-2\sigma} \cdot \frac{2m}{r^2} \tag{3.18}$$

$$\sigma_{,x} = -2\pi \cdot \frac{\psi_{,x}^2 \cdot r}{r_{,x}} - e^{-2\sigma} \cdot \frac{m}{r^2 r_{,x}} \tag{3.19}$$

$$m_{,x} = 4\pi r^2 \cdot e^{2\sigma} \left[\frac{1}{2} r_{,x} \cdot \psi_{,x}^2 \right] \tag{3.20}$$

and their corresponding boundary conditions,

$$r(0,0) = 0 \implies r_0^0 = 0 \tag{3.21}$$

$$r_{,x}(0,0) = 1 \implies r_{0,x}^0 = 1 \tag{3.22}$$

$$\sigma(0,0) = 1 \implies \sigma_0^0 = 0 \tag{3.23}$$

$$m(0,0) = 1 \implies m_0^0 = 0 \tag{3.24}$$

We used fourth order Runge-Kutta (RK4) method to solve these equations. RK4 method is the industry standard method to solve ordinary differential equations, RK4 is a fourth order method. Like most of the other finite difference methods RK4 is derived by solving a bunch of linear equations which we get by using taylor expansions. Its derivation can be found in any numerical analysis book and we will skip it because it will not add a lot to the current discussion.

Unlike partial differential equations, solving ordinary differential equations is mostly a "solved" problem in the sense that are plethora of robust packages that can solve a system of differential equations without any significant input from the user.

The standard way to solve a system of ordinary differential equations which condition higher order derivatives is to define new variables and get a set of first order ODEs. In our case we define a new variable $\bar{r} = r_{,x}$ which gives us,

$$r_{,x} = \bar{r} \tag{3.25}$$

$$\bar{r}_{,x} = e^{-2\sigma} \cdot \frac{2m}{r^2} \tag{3.26}$$

$$\sigma_{,x} = -2\pi \cdot \frac{\psi_{,x}^2 \cdot r}{\bar{r}} - e^{-2\sigma} \cdot \frac{m}{r^2 \bar{r}} \tag{3.27}$$

$$m_{,x} = 4\pi r^2 \cdot e^{2\sigma} \left[\frac{1}{2} \bar{r} \cdot \psi_{,x}^2 \right] \tag{3.28}$$

Observe that there are no unknown derivatives in the RHS of these equations ($\psi_{,x}$ is known because we have the initial profile of ψ). At this point one can plug these equations in any differential equations solver, in our case we used DifferentialEquations.jl library for Julia language with tolerance set to 10^{-8}.

Time evolution

For time evolution we will use second order discretization of both the first and the second order derivatives. We will just show the discretization of the equation 3.29 which requires most work, rest of the equations can be discretized using the exact same procedure.

We should mention that to make things simple it is important that we solve the equation 3.9 first followed by equation 3.10 after which other two equations can be solved in any order. The reason for this choice will become clear as we discretized the system.

$$-\psi_{,tt} + \psi_{,xx} + \frac{2}{r}\left(-r_{,t}\psi_{,t} + r_{,x}\psi_{,x}\right) = 0 \tag{3.29}$$

Writing the equation 3.29 in the notation we defined earlier we get:

$$-\psi^n_{j,tt} + \psi^n_{j,xx} + \frac{2}{r^n_j}\left(-r^n_{j,t}\psi^n_{j,t} + r^n_{j,x}\psi^n_{j,x}\right) = 0 \tag{3.30}$$

Now we will use the second order finite difference approximations already discussed,

$$\psi^n_{j,tt} = \frac{\psi^{n+1}_j - 2\psi^n_j + \psi^{n-1}_j}{\delta t^2} \tag{3.31}$$

$$\psi^n_{j,t} = \frac{\psi^{n+1}_j - \psi^{n-1}_j}{2\delta t} \tag{3.32}$$

$$\psi^n_{j,xx} = \frac{\psi^n_{j+1} - 2\psi^n_j + \psi^n_{j-1}}{\delta x^2} \tag{3.33}$$

$$\psi^n_{j,x} = \frac{\psi^n_{j+1} - \psi^n_{j-1}}{2\delta x} \tag{3.34}$$

Other variables can be discretized in the same manner. Now we will make another simplifying assumption $\delta t = \delta x$, we use this assumption to make the equations look a little simpler and it does not affect the results. But, this is not something that can always be done, there are systems where making this assumption will lead to superfluous oscillations and instability. To prevent such instabilities we need to ensure that the temporal step size (δt) is bounded above by the spatial step size (δx) as dictated by the Courant–Friedrichs–Lewy or CFL conditions. Here it is enough to know that for our system we can take $\delta t = \delta x$ without causing instabilities.

Now, putting equations 3.31-3.34 and similar equations for r in the equation 3.30 and cancelling out all the δt and δx we get,

$$\psi_j^{n+1}+\psi_j^{n-1}-\psi_{j+1}^n-\psi_{j-1}^n = \frac{((\psi_{j+1}^n - \psi_{j-1}^n)(r_{j+1}^n - r_{j-1}^n) - (\psi_j^{n+1} - \psi_j^{n-1})(r_j^{n+1} - r_j^{n-1}))}{2r_j^n} \tag{3.35}$$

What we need to do now is to collect all the ψ_j^{n+1} terms on the LHS and rest of the equation on the RHS. This will give us the value of ψ at the future time $(n+1)\delta t$ using the value of r and ψ at the current time $n\delta t$ and the last time step $(n-1)\delta t$.

One can see that this will lead to a very convoluted and hard to debug equation, there is actually a lot of simplification one can do by calculating few things in advance. First, notice that we can calculate all the spatial derivatives in advance because they only depend on the current time step $(n\delta t)$. Furthermore, because we solved the equation 3.9 before the current equation of ψ we can also calculate and store the value of $r_{j,t}^n$ in advance. Using all this we can rewrite the equation 3.35 as,

$$\psi_j^{n+1} + \psi_j^{n-1} - \psi_{j+1}^n - \psi_{j-1}^n = \frac{(4\delta t^2(\psi_{j,x}^n\, r_{j,x}^n) - 2\delta t(\psi_j^{n+1} - \psi_j^{n-1})\, r_{j,t}^n)}{2r_j^n} \tag{3.36}$$

Now our equation is much more tangible and we can bring all the ψ_j^{n+1} factors to the RHS and get the time stepping equation for ψ. We should explicitly mention that this is just a cosmetic change and the final answer will be the same up to floating point errors.

For other equations, the procedure is exactly the same with the only difference being that there will be no future time step term like ψ_j^{n+1} in the LHS, which will make the corresponding time stepping equations much simpler.

There is one last technicality that we have to deal with before we can wrap up this section, we can not use the time stepping equation we derived above for the first time step $(n = 1)$. One can see this by plugging in $n = 0$ in the equation 3.36, which is supposed to give the value of ψ at time δt. But, doing so will lead to a term of type ψ_j^{-1} which is of course not something we can deal with using our current stencil (time stepping equations).

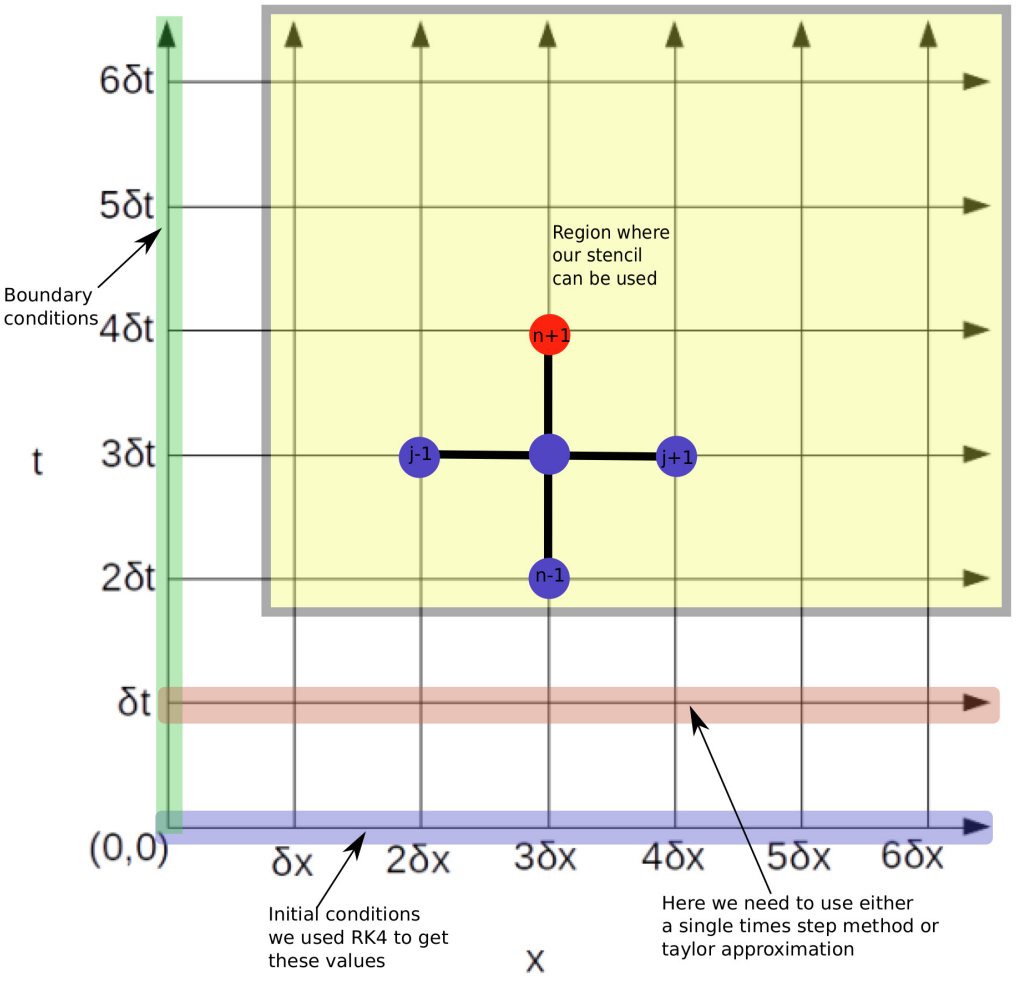


Figure 3.8: This figure shows our computational domain and the stencil used. To calculate the value at any point on the grid we need the red dot of the stencil to be at that grid point, observe that we can not put the red circle of the stencil in the regions that are green, red and blue. **Yellow**: We can use our stencil in this region. **Red**: We have to use any other methods or stencils that only depend on the last time step, we used taylor series because it was the easiest to use in our case. **Blue**: This is the initial configuration of system that we evolve in time, i.e. initial conditions of our system.

This brings us to one of the reason why higher order methods are in general harder to code and reason about. Our finite difference approximation uses three space and and last two time steps thus, we can not use it at the boundaries or for the first time step. Figure 3.8 shows various regions where we can not use out time stepping equations. This becomes a much bigger issue if one uses even higher order finite difference methods as they will use more than 3 spatial points per calculation.

Dealing with the first time step

To calculate the value of the variables at time $t = \delta t$ we used taylor series, again for brevity we will show the procedure just for ψ.

$$\psi(\delta t, x) = \psi(0, x) + \delta t\, \psi_{,t}(0, x) + \frac{\delta t^2}{2}\, \psi_{,tt}(0, x) + O(\delta t^3)$$

First notice that we are using taylor series up to $O(\delta t^3)$, this is essential because our time stepping equations are of second order. If we do not use higher order taylor series then the first order error will propagate to later time steps defeating the purpose of using second order stencil. This should be kept in mind while using single step methods as well, because they are not as accurate as our second order method for the same step size. In that case we need to divide the time step δt into smaller time steps and take multiple small time steps via a single step stencil to get to the first time step (δt).

Another convenience in using taylor series in our case is that $X_{,t}(0, x) = 0$ for $X = \psi, r, \sigma$ because of our boundary conditions. Thus we only need to solve,

$$\psi(\delta t, x) = \psi(0, x) + \frac{\delta t^2}{2}\, \psi_{,tt}(0, x) + O(\delta t^3)$$

$X_{,tt}(0, x)$ can be easily obtained using the equations 3.9-3.11.

For m we need to do some work, first we have to take time derivative of the equation 3.12 and then use it in the taylor series.

Boundary conditions

Boundary conditions (refer section 2.2) at $x = 0$ are easy to implement, for example, $r_t(t, 0) = 0$ when discretized is same as $r_0^n = r_0^{n-1}$. Similarly, $\psi_{,x}(t, 0) = 0$ when discretized implies $\psi_0^n = \psi_1^n$.

Now, we need the boundary conditions at the outer boundary, in the physical space this is same as the boundary conditions at infinity. But we are working with a finite domain so we need some kind of boundary conditions at the spatial edge of our computational domain. There are multiple ways to deal with this, one is to simply use extrapolation boundary conditions. Another is to use absorbing boundary conditions or free boundary conditions, both of these are usually more accurate but at the same time harder to implement.

In our case we are lucky because we have a system of hyperbolic equations and in addition to that we do not need to evolve the system for longer periods. As it turns out for hyperbolic systems there is a natural light cone kind of domain associated with each grid point and it can only be affected by the grid points in that cone. This can be intuitively seen in the Figure 3.9. Because of this property we can use extrapolation boundary conditions without it affecting our calculations.

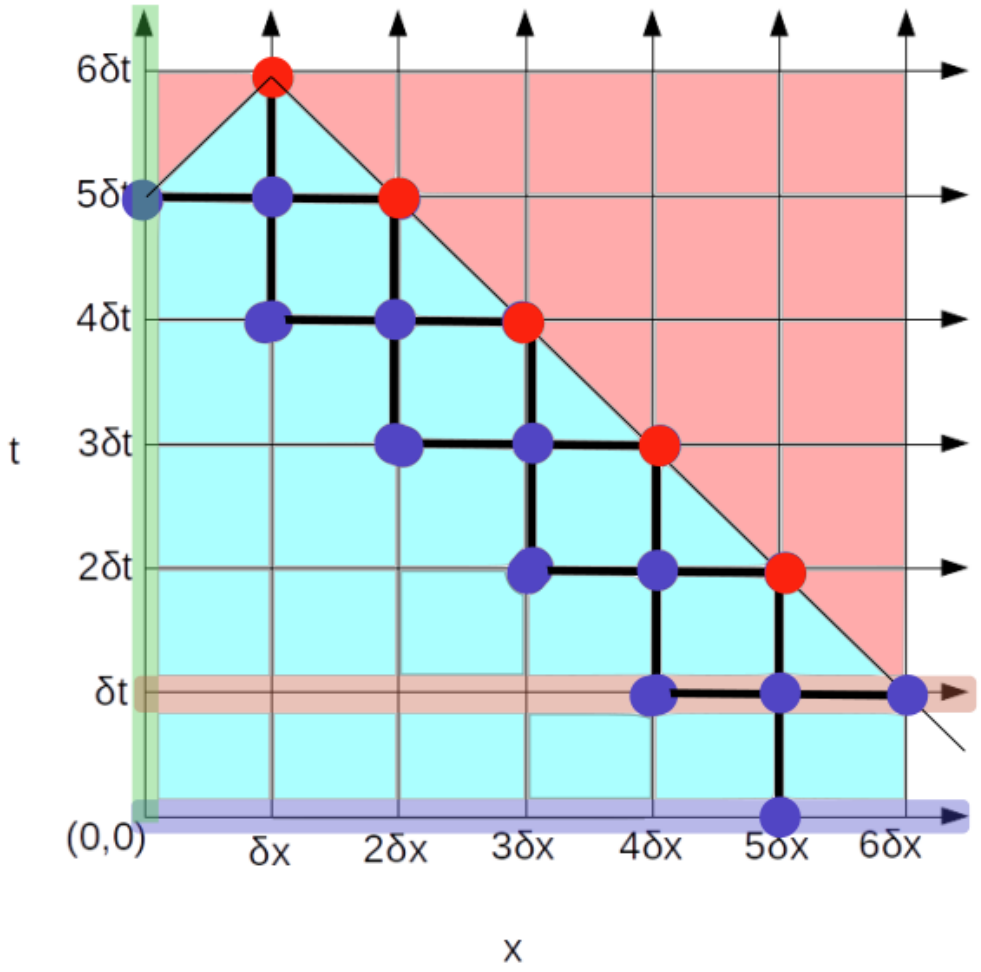


Figure 3.9: This figure shows the grid points that can affect the grid point X at $(6\,\delta t, \delta x)$. Grid points in blue can affect the grid point X while grid points in red can not.

3.3 How do we check the accuracy of the numerical results?

Because we do not have an analytical solution to compare with, we need some others ways to get confidence in our results. We used two commonly used methods to check the reliability of our numerical results,

1. Using constraint equations

2. Showing that the results are independent of the step size

Using constraint equations

As mentioned in the last chapter we get two constraint equations along with the three evolution equations. We can use these constraint equations to check whether our results are correct or not. This is the primary method we used and there will be more discussion about the values of constraints in the next chapter.

Showing that the results are independent of the step size

We have developed the whole algorithm but there is still the question of what step size should we use. If the step size is too small then our equations will not be a good approximation of the original equations. On the other hand if the step size is too small then it may take a lot of time to evolve the system. The standard way of choosing a step size is to decide on a tolerance and try out several step sizes. As we keep on decreasing the step size there will be a threshold after which the answers will effectively become independent of the step size up to the tolerance. We can then choose any step size smaller than that threshold.

This also acts as a check for the algorithm because any dependence of the answer on the step sizes smaller than the threshold is a clear indication that our algorithms are not true representation of the underlying differential equations. This method of checking the finite difference may fail for chaotic systems, but we are not dealing with those here.

Then there are obvious problem specific sanity checks like there are no superfluous oscillations. We can also check for extreme cases, for example, in our system if we take the initial configuration of the scalar field to be uniformly zero then we do not expect to see any kind of evolution.

This sums up our numerical part, in the next chapter we will discuss about the profiles of ψ used and the results.

Chapter 4

PROFILES USED AND RESULTS

Most of this chapter is taken from (Chaudhary and Krishnan, 2020).

For all these profiles the spatial domain was [0,2] and the time domain was [0,1]. We used four different profiles of ψ,

Gaussian profile:

$$\psi(t = 0, x) = A \exp\left(\frac{-(x - x_0)^2}{\delta^2}\right)$$

Maxwell (or Modified Gaussian) profile:

$$\psi(t = 0, x) = Ax^2 \exp\left(\frac{-(x - x_0)^2}{\delta^2}\right)$$

Ball (or tanh) profile:

$$\psi(t = 0, x) = A\left(\tanh\left(\frac{-(x - x_0)}{\delta^2}\right) + 1\right)$$

Shell profile:

$$\psi(t = 0, x) = A\left(\tanh\left(\frac{-(x - x_0)}{\delta^2}\right) + \tanh\left(\frac{(x - x_0 + w)}{\delta^2}\right)\right)$$

For all these profiles we first obtained a rough estimate of the critical amplitude, which was then used to evolve the system once with a subcritical amplitude and then again with a supercritical amplitude.

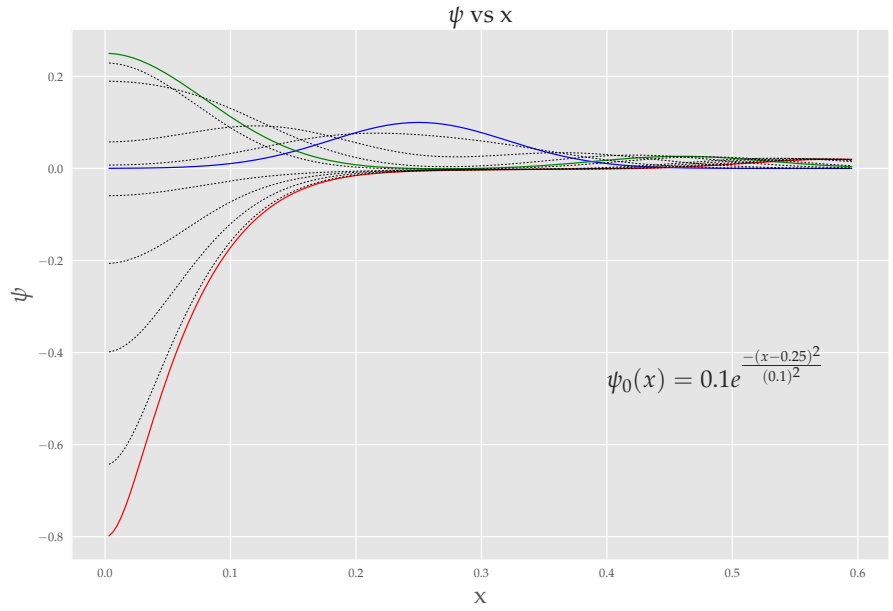
The simulation was stopped in the supercritical case when it was obvious that the scalar field has moved by $O(1)$ in plank units.

For each profile we are showing two figures, figure (a) shows the fields evolution in time. Blue line is the initial value of ψ, green is the value of the field when it reached its maxima at the origin similarly red is the value of the field when it reached its minima at the origin (We stopped the simulations when it was apparent that the field has moved by $O(1)$ distance). Figure (b) shows the value of the field at the origin vs time, blue line shows the subcritical evolution and red line shows the supercritical evolution.

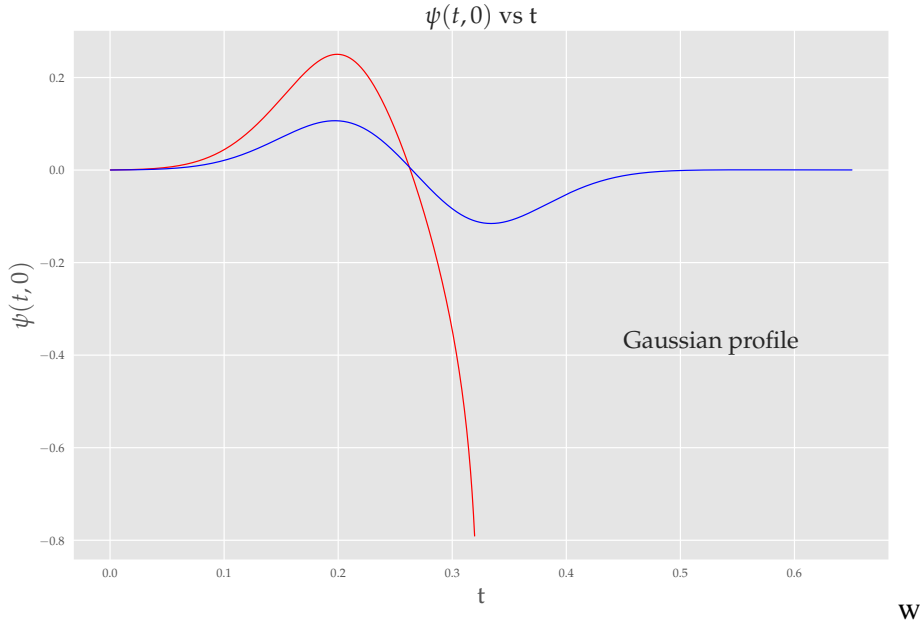
To gain confidence in the results we ran the simulations with multiple step sizes to ensure that the results are independent of the algorithm. In addition to that we also checked the two constraint equations to ensure that the system us evolving correctly.

Subcritical evolution does not involve a lot of field movement thus our algorithm gives very accurate results. But, in the supercritical cases during the collapse the fields start to move very rapidly which leads to a lot of error 3.1. For the gaussian and the modified gaussian case the constraints had values of the order $10^{-}5$ at the end. For the square and the shell profiles the value of the constraint were of the order $10^{-}3 - 10^{-}4$ at the end of the simulation. These comparatively larger values can be attributed ot the fact that during the collapse the field moves much faster in case of the square and the shell profiles (Figures 4.2b,4.2b,4.3b,4.4b). Although, even in supercritical cases when the field was not moving very fast the constraints were much smaller.

We can decrease the step size to get more accurate results but there was a limit on how small we can go due to the RAM size. One way to deal with this is to use adaptive mesh refinement(AMR), where one reduces the step size at the places where the field is moving very fast to get more accurate results. We decided not to use AMR because writing a AMR code is significantly harder and the accuracy that we were able to achieve was enough to demonstrate our point. That being said with more powerful computers and better algorithms it is possible to achieve much better accuracies if required.

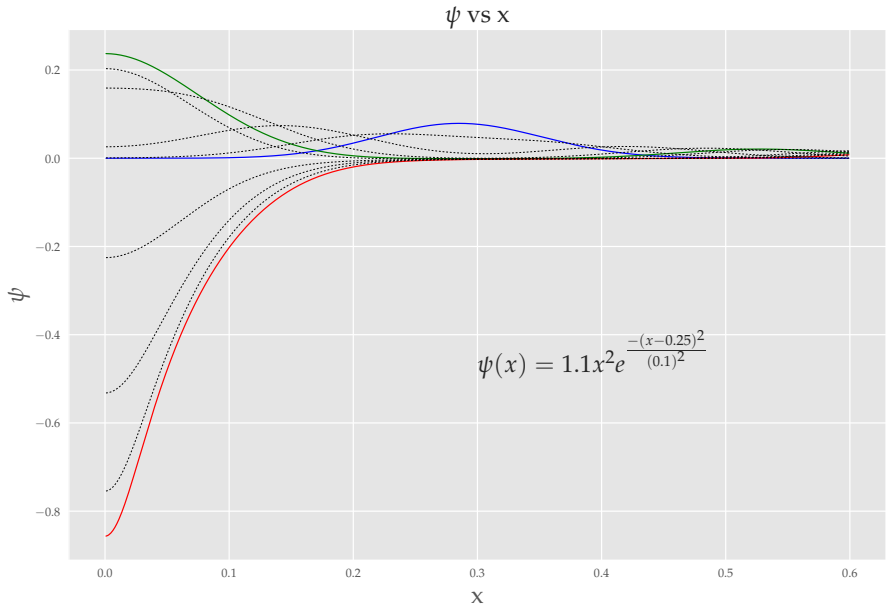


(a)

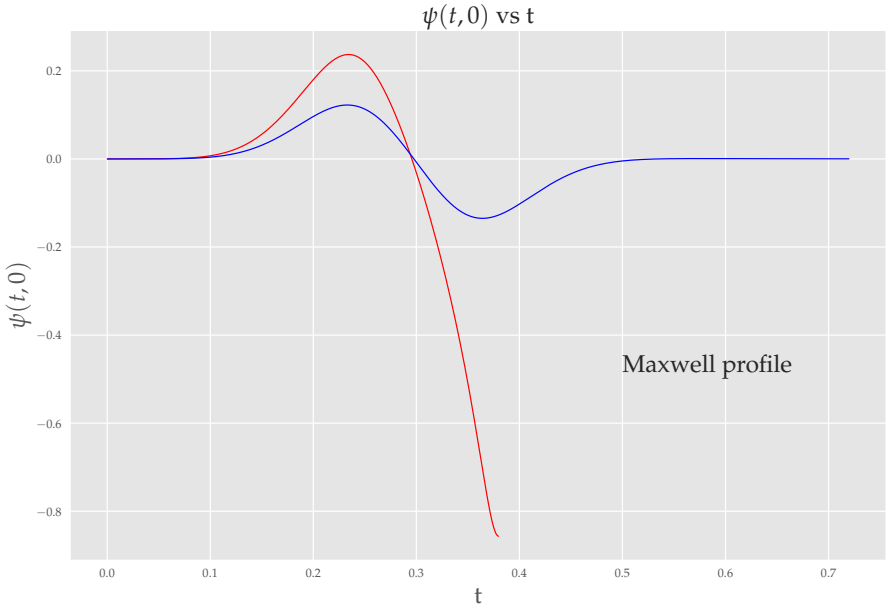


(b)

Figure 4.1: **Gaussian Profile**. Figure (a) shows supercritical evolution of the field, **Blue**: initial profile of ψ , **Green**: profile of ψ when ψ reaches its maxima at the origin, **Red**: profile of ψ when ψ reaches its minima at the origin. Figure (b) shows the supercritical (**red**) and the subcritical (**blue**) evolution of ψ at the origin with respect to time.

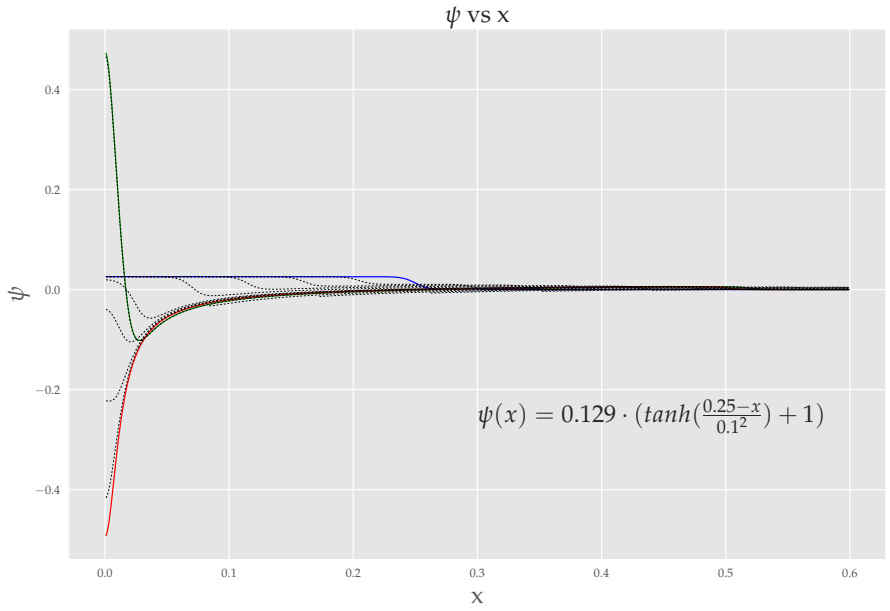


(a)

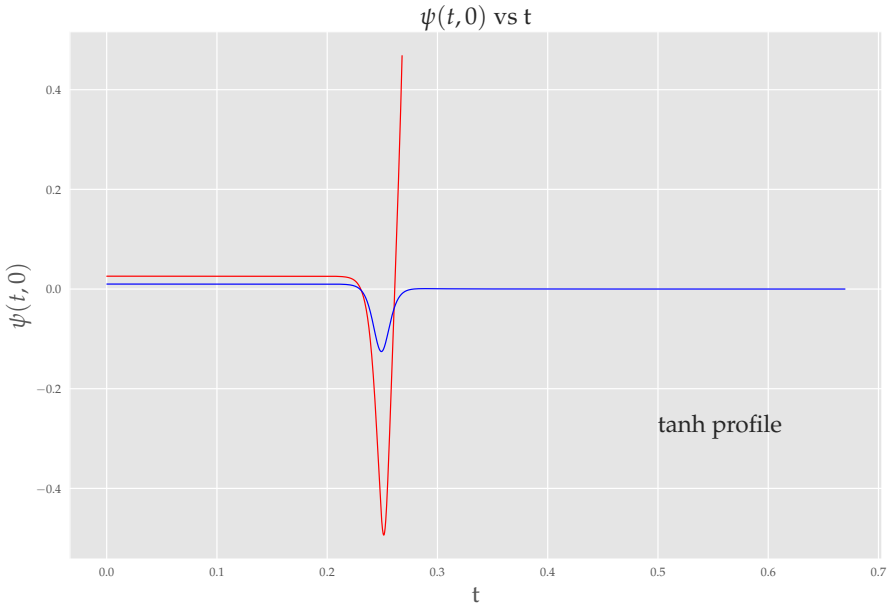


(b)

Figure 4.2: **Maxwell Profile**. Figure (a) shows supercritical evolution of the field, **Blue**: initial profile of ψ , **Green**: profile of ψ when ψ reaches its maxima at the origin, **Red**: profile of ψ when ψ reaches its minima at the origin. Figure (b) shows the supercritical (**red**) and the subcritical (**blue**) evolution of ψ at the origin with respect to time.

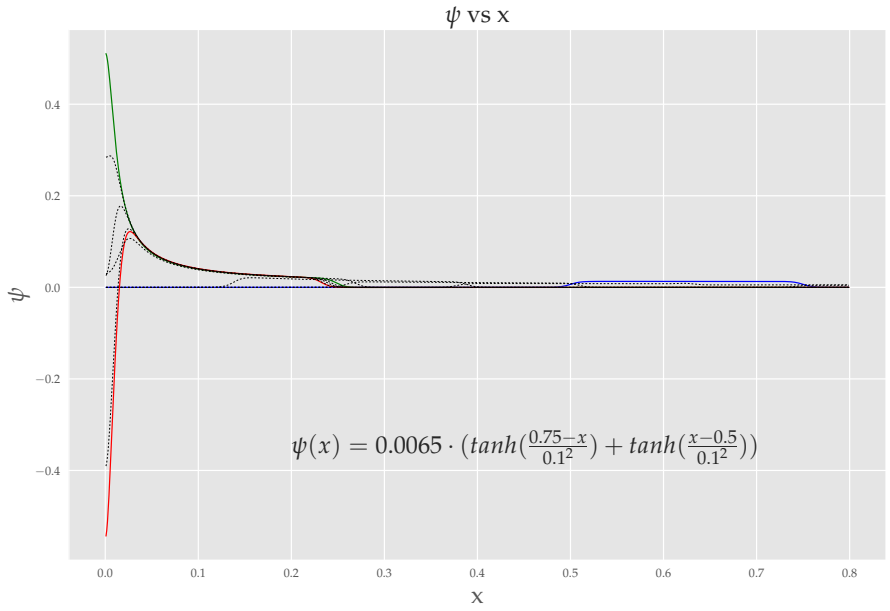


(a)

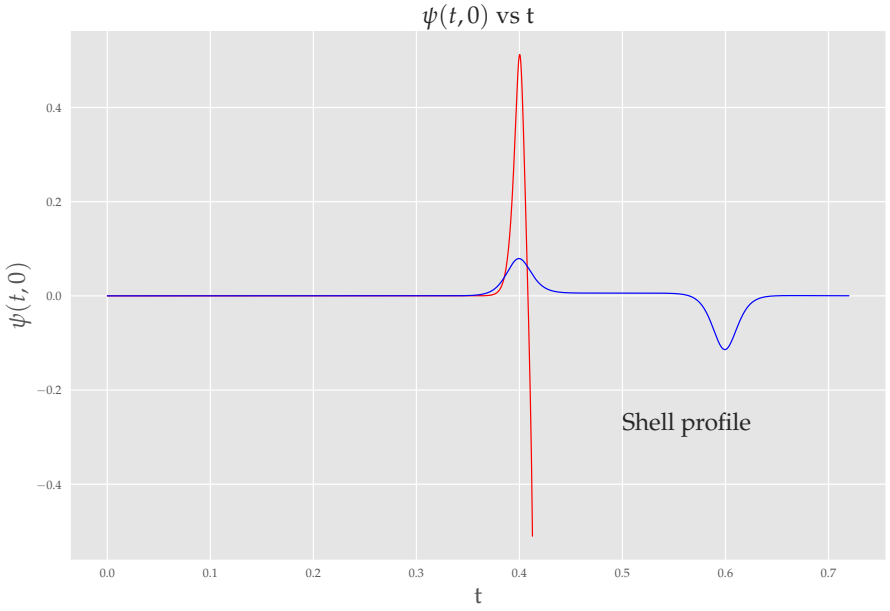


(b)

Figure 4.3: **Tanh Profile**. Figure (a) shows supercritical evolution of the field, **Blue**: initial profile of ψ , **Green**: profile of ψ when ψ reaches its maxima at the origin, **Red**: profile of ψ when ψ reaches its minima at the origin. Figure (b) shows the supercritical (**red**) and the subcritical (**blue**) evolution of ψ at the origin with respect to time.



(a)



(b)

Figure 4.4: **Shell Profile**. Figure (a) shows supercritical evolution of the field, **Blue**: initial profile of ψ , **Green**: profile of ψ when ψ reaches its maxima at the origin, **Red**: profile of ψ when ψ reaches its minima at the origin. Figure (b) shows the supercritical (**red**) and the subcritical (**blue**) evolution of ψ at the origin with respect to time.

Chapter 5

CONCLUSIONS AND FUTURE DIRECTIONS

The results that we have got show that there is a sharp tension between two ideas, first that a $O(1)$ scalar field range indicates breakdown of effective field theory and second that an effective field theory should be well-defined at the horizon. Because both these ideas are still neither experimentally verified nor have a sound theoretical "proof", we hope that this extra bit of insight will give us another way of analyzing them.

There are a lot of directions the work done in this thesis can be extended for example, one can look at what happens in the case of a massive scalar field. One can also work with non-canonical kinetic terms or multiple fields, and although we expect that the $O(1)$ field movement will still be there it will be interesting to directly verify it. This whole study was done under the assumption of spherical symmetry, another way to extend this project will be to do a full 3D evolution of a scalar field without any symmetry assumption.

BIBLIOGRAPHY

Brennan, T. Daniel, Federico Carta, and Cumrun Vafa (2017). "The String Landscape, the Swampland, and the Missing Corner". In: *PoS* TASI2017, p. 015. DOI: 10.22323/1.305.0015. arXiv: 1711.00864 [hep-th].

Chaudhary, Himanshu and Chethan Krishnan (Mar. 2020). "A Firewall Argument from the Swampland". In: arXiv: 2003.05488 [hep-th].

Gundlach, Carsten and Jose M. Martin-Garcia (2007). "Critical phenomena in gravitational collapse". In: *Living Rev. Rel.* 10, p. 5. DOI: 10.12942/lrr-2007-5. arXiv: 0711.4620 [gr-qc].

Guo, Jun-Qi, Daoyan Wang, and Andrei V. Frolov (2014). "Spherical collapse in f(R) gravity and the Belinskii-Khalatnikov-Lifshitz conjecture". In: *Phys. Rev. D* 90.2, p. 024017. DOI: 10.1103/PhysRevD.90.024017. arXiv: 1312.4625 [gr-qc].

Guo, Jun-Qi and Hongsheng Zhang (2019). "Geometric properties of critical collapse". In: *Eur. Phys. J. C* 79.7, p. 625. DOI: 10.1140/epjc/s10052-019-7144-2. arXiv: 1808.09826 [gr-qc].

INDEX

CFL, 21
Courant–Friedrichs–Lewy, 21

Discretization, 18
discretization, 21

extrapolation boundary conditions, 24

figures, 13–17, 19, 23, 25
Finite difference methods, 9
finite precision, 11
first order, 12
floating point error, 11
floating point errors, 10
Floats, 10

initial conditions, 7
instability, 21

mantissa, 11

naked singularity, 4

Precision, 10

Runge-Kutta, 20

scaling law, 4
significant digits, 10
stencil, 22
step size, 10
superfluous oscillations, 21

time stepping equation, 22
time-space domain, 18