7CCP4000 - Project in Physics: Numerical Relativity in 1-Dimension

King's College London

Author - Snehan S Sighat
Supervisor - Dr Eugene Lim

October 20, 2019

Abstract

General Relativity models physical theories with the complication of gravity in tow. Naturally, one very rarely encounters problems with analytic solutions in GR due to the abundance of multidimensional, non-linear differential equations. In such a field, analytic solutions can only easily be found for idealized cases, and so they are solved numerically. In this project, the goal was to essentially be able to solve equations in GR with spherical symmetry, with the final goal of being able to simulate the collapse of a scalar field into a black hole in such a space-time. Intermediate steps involved implementing the RK4 algorithm to integrate simple first and second-order ODEs, before using MOL-RK4 to solve the Klein-Gordon equation in Cartesian coordinates and then in spherical coordinates where spherical symmetry was assumed. All of these preceding tasks were completed; however many difficulties were encountered in simulating the 'collapse of the lapse' during the final task. Eventually only somewhat desireable results were obtained and possible reasons for this will be discussed.

Contents

Chapter 1

Introduction

1.1 Numerical Methods

The underlying concept behind numerical methods is the need to solve complex mathematical problems using computer programs. A large subset of these problems concern themselves with integration, most of which become far simpler through approximation using numerical methods. These approximate solutions however, can be obtained to great accuracy by simply increasing the number of iterations in a given time-frame; more on this later. More specifically, these methods allow us to integrate equations in physics where for example, singularities can be avoided, or irregular boundaries need to be outlined and taken into account.

1.1.1 Finite difference methods

Some motivation for the idea behind finite-difference methods can be explained using the idea of a field. A field can be described as a quantity point-wise defined everywhere in space and time. As most of the problems in this project deal with basic field theory, we used finite-differencing approximation heavily to discretise space and time, therefore creating a mesh of a finite number of points over which our simulations would run. For the purposes of this project, the spatial and temporal steps were kept consistent, i.e. a uniform grid was assumed. One must assume that the solution to our desired differential equation is valid everywhere within the specified domain. The main idea behind using finite difference approximations in numerical relativity is to be able to express the derivative of a function at any given point in terms of its value at neighbouring points. Consequently, we first introduce a temporal mesh containing N_t points, distributed uniformly between $t_i, ..., t_f$, with the temporal grid spacing, δ_t, thus giving us

$$t_{i+1} = t_i + \delta_t \tag{1.1}$$

for a function of time. Where a function is that of time as well as space, $f(x, y)$, we also introduce a spatial mesh containing N_x points, distributed uniformly between $x_i, ..., x_f$, with grid spacing, δx, using which we then have

$$x_{i+1} = x_i + \delta_x \tag{1.2}$$

We are now able to justify the solutions to our equations on all points within this space-time grid. Now, to be able to solve differential equations, we need to be able to approximate any derivatives we encounter in the function in question. To do this, with the assumption that our

function is differentiable to sufficiently high-orders, we are able to expand it as a Taylor series, about a small perturbation, δx, which is our grid spacing.

$$f_{i+1} = f(x_i + \delta x) = f(x_i) + \delta x f^{'}(x) + \frac{\delta x^2}{2} f^{''}(x) + \frac{h^3}{6} f^{'''}(x) + \mathcal{O}(\delta x^4) \tag{1.3}$$

$$f_{i-1} = f(x_i - \delta x) = f(x_i) - \delta x f^{'}(x) + \frac{\delta x^2}{2} f^{''}(x) - \frac{\delta x^3}{6} f^{'''}(x) + \mathcal{O}(\delta x^4) \tag{1.4}$$

The above can be manipulated to obtain approximations for the first and second-order derivatives, $f^{'}(x)$ and $f^{''}(x)$ to give (1.5) and (1.6)

$$(f^{'}(x))_i \approx \frac{f(x + \delta x) - f(x - \delta x)}{2\delta x} + \mathcal{O}(\delta x^2) \approx \frac{f_{i+1} - f_{i-1}}{2\delta x} \tag{1.5}$$

$$(f^{''}(x))_i \approx \frac{f_{i+1} - 2f_i + f_{i-1}}{(\delta x)^2} \tag{1.6}$$

With the above manipulation, we are also able to completely eradicate the error incurred due to truncation of the series [3]. On inspection, we see that in order to compute the derivatives of any function using this method, we require the values of the function on either side of the iteration being considered; consequently, such approximations are called are more specifically called centred difference schemes. So we find that for the above implementation to be viable, we require initial and boundary conditions. The nature of these conditions will be discussed later, but we see that in order to implement any boundary conditions, we require our differential equation to be valid on the internal mesh points $x_1, ..., x_{N_x - 1}$. We see visual representation of this process in Figure 1.1.

While it is entirely possible to evolve a system in time using finite difference methods, in this project the focus was on using centred difference approximations to express spatial derivatives in terms of their neighbouring points allowing for the time dimension to remain continuous. Such an approach is known as method of lines and requires a method that would evolve the discretised system in time. We shall explore one such method next.

1.1.2 Fourth Order Runge-Kutta method

We will now look at the fourth order Runge-Kutta method, which when combined with method of lines, is known as MOL-RK4. The theory behind the Runge-Kutta methods is outlined well in [8] and will be summarised in parts below. Euler's method is used to approximate solutions to differential equations by use of the truncated Taylor expansion of x about t. (1.7) describes the first step in this method.

$$x_{n+1} = x_n + hf(t_n, x_n) + \mathcal{O}(\delta t^2) \tag{1.7}$$

Given an initial value (x_0, y_0), the curve describing the function is essentially approximated by line segments, where the concept of local linearity is utilised. As can be seen from Figure 1.2, the derivative of the function is calculated at the beginning of any interval and the solution carries across a chosen step-size, δt. It should be noted that the value of h should of course

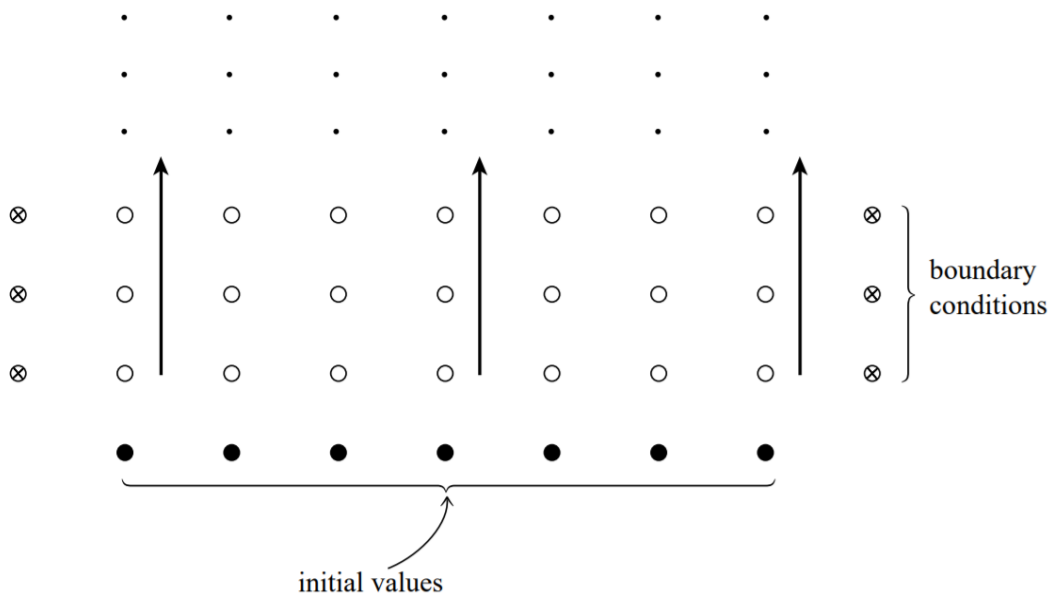


Figure 1.1: This diagram shows us how the value of our solution is evolved through a grid, with space discretised along the x-axis and time along the y-axis. We see how through the use of initial conditions, one is able to advance the system through all time slices. We also see how boundary conditions are applied in space at every time-step. Taken directly from [8]

be kept small to maximize accuracy, as a small time-step results in a larger number of iterations overall. Euler's method is relatively inaccurate; this can be seen from Figure 1.2, where the approximate solution increasingly underestimates the curvature of the exact solution with each step. To increase accuracy, one would intuitively include more terms in the Taylor expansion. This would however require computation of expressions for higher-order derivatives. The second-order Runge-Kutta method is a viable alternative as it does not require this computation. (1.7) is used to compute the value of the desired function halfway across the interval being used, which is used to find the derivative across the entire interval before being discarded, thus giving rise to second-order accuracy in δt, as shown in (1.8). A visual of the extrapolation of the solution is shown in Figure 1.3.

$$x_{n+1} = x_n + \delta t f(t_n + \delta t/2, x_n + \delta t f(t_n, x_n)/2) + \mathcal{O}(\delta t^3) \tag{1.8}$$

where

$$k_1 = \delta t f(t_n, x_n) \tag{1.9}$$

$$k_2 = \delta t f(t_n + \frac{\delta t}{2}, x_n + \frac{k_1}{2}) \tag{1.10}$$

And so 1.8 can more implicitly be written as

$$x_{n+1} = x_n + \delta t k_2 + \mathcal{O}(\delta t^3) \tag{1.11}$$

As a result of its second order accuracy, the method is also known as the second order Runge-Kutta method. By inference, fourth order Runge-Kutta methods are merely an extension of the above to fourth-order accuracy in δt. More terms of the Taylor expansion are included and given specific parameters in the derivation, the most commonly used fourth-order Runge-Kutta method, known simply as the Runge-Kutta method is obtained.

$$x_{n+1} = x_n + \frac{1}{6}(k_1 + 2(k_2 + k_3) + k_4) \tag{1.12}$$

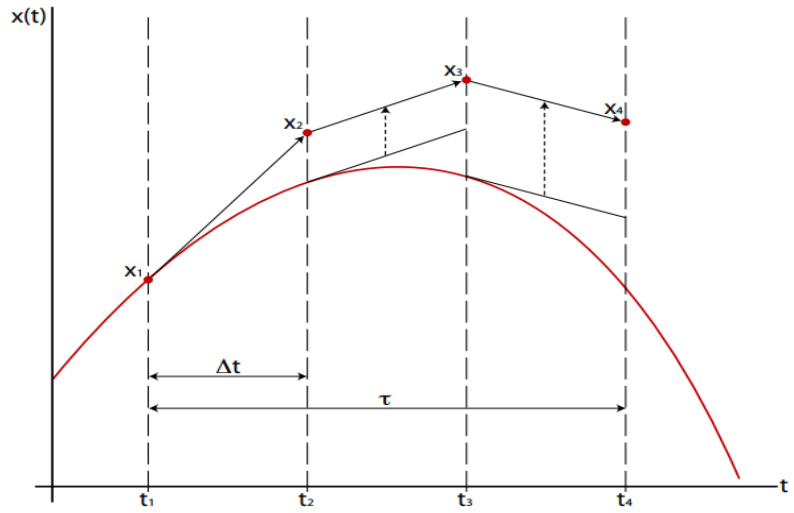


Figure 1.2: A pictorial representation of how the solution advances when an ODE is integrated using Euler's method. Each iteration is computed using the slope at the preceding point on the curve. Taken directly from [2]

where k_1 and k_2 are found using (1.9) and (1.10). The remaining two evaluations of the derivative per time-step h are given below in (1.15) and (1.16)

$$k_1 = \delta t f(t_n, x_n) \tag{1.13}$$

$$k_2 = \delta t f(t_n + \frac{\delta t}{2}, x_n + \frac{k_1}{2}) \tag{1.14}$$

$$k_3 = \delta t f(t_n + \frac{\delta t}{2}, x_n + \frac{k_2}{2}) \tag{1.15}$$

$$k_4 = \delta t f(t_n + \delta t, x_n + k_3) \tag{1.16}$$

Approximating solutions through Taylor expansions requires derivatives of the function in question to be computed for each new term one wants to find. The Runge-Kutta method eradicates this problem thus allowing for complicated differential equations to be solved. Let us elaborate on Equations (1.12)-(1.16). As before, the values of t are only increased by the chosen step-size, δt. The values of x however, undergo more involved steps. (1.13) allows us to define a slope along the first numerical solution, which when multiplied by δt gives us Δx for that first iteration. (1.14) then allows us to compute Δx for a numerical solution found half-along along the time-step. This happens another two times in (1.15)-(1.16); the weighted average for these values is then found, with the extrapolations from the mid-points being given double the weighting.

In this method however, we see an abundant issue - the step-size δt must essentially be assumed. Adaptive methods bypass this problem through the computation of the error due to truncation of the Taylor series, which is then used to adjust the step-size accordingly [6]. Although this is outside the scope of this project, these techniques could have been implemented to improve our code and allow for more accurate results.

The duration of these simulations is important to be aware of. The longer a numerical simulation runs, the larger the errors due to rounding-off and truncation become. The errors can,

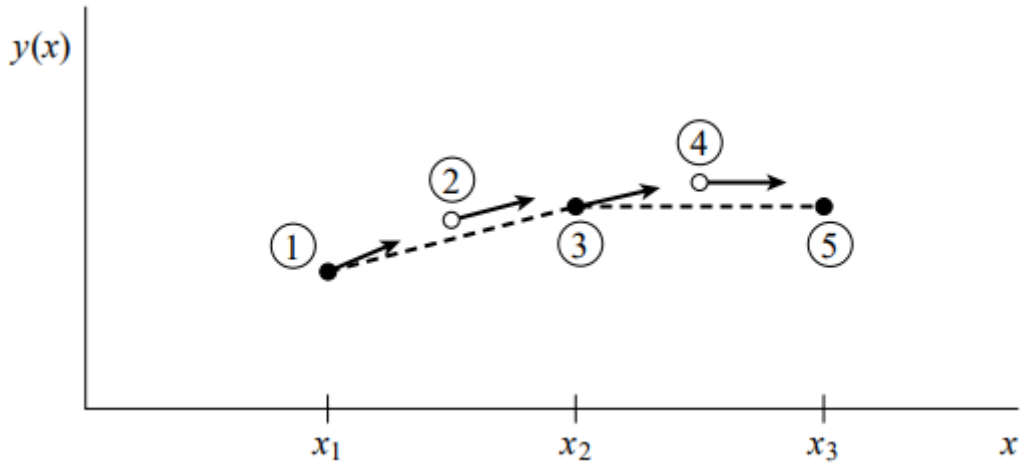


Figure 1.3: A pictorial representation of how the solution advances when an ODE is integrated using the second-order Runge-Kutta method. The hollow circles refer to solutions discarded after computation. Taken directly from [8]

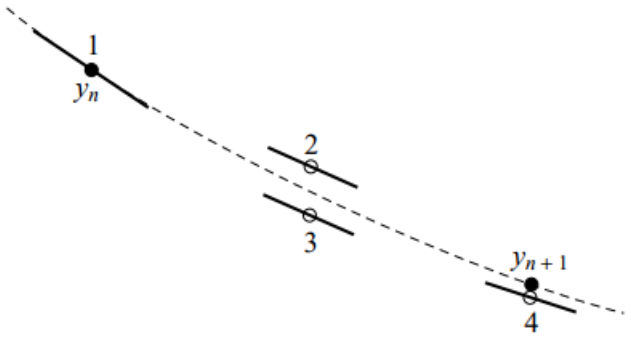


Figure 1.4: A typical fourth-order Runge-Kutta calculation. For each time step δx, the derivative is calculated four times. As before, the hollow circles represent trial solutions that are discarded after use and those filled-in represent solutions that we seek. Taken directly from [8]

given enough time, drastically change the solution and lead to inaccurate results. This is why it is important for the total time duration to be kept relatively low, for the number of temporal iterations to be high and as a result, for the time-step to be kept low, to maximise accuracy.

1.1.3 Initial and Boundary condition problems

Figure 1.1 describes what is known in numerical analysis as the initial-value problem. In this project, we also deal with boundary-value problems, where a solution is identified to be valid within a specific region only and the system must be integrated from the boundary; whereas for the former, the system is integrated forward in time. Finite differences and RK4 correspond to the problems respectively.
It is evident that for any such problem, we require boundary conditions. For a large portion of this project, we dealt with wave propagation, so we allude to he most common choices for a system governing the behaviour of a wave, listed below.

- Dirichlet condition - This condition allows us to mirror wave behaviour at boundary it is implemented on. The way one applies this condition is by defining the value of the

function at the boundary, e.g. for a system with stationary boundary points, we have (1.17) at $x = 0, L$.

$$f = 0 \tag{1.17}$$

- Neumann condition - This allows us to reflect the wave at the boundary, by defining the value of the derivative of the solution at this boundary. E.g. at $x = 0, L$

$$f^{'} = 0 \tag{1.18}$$

When the wave approaches this point, the reflected wave interacts with the first wave, which results in the amplitude doubling in magnitude, before the entire pulse is completed reflected back into the domain.

- Open boundary condition or radiation conditions, allow the wave to pass through the domain ($x = 0, x = L$) uninterrupted, while periodic boundary condition allow the wave to propagate from one side of the domain only to be fed back into the system from the opposite side. These conditions were not implemented in this project, but are described in (1.19)-(1.21) for completeness.

$$\dot{f} = f^{'} \text{at x = 0} \tag{1.19}$$

$$\dot{f} = -f^{'} \text{at x = L} \tag{1.20}$$

$$f(t, x = 0) = f(t, x = L) \tag{1.21}$$

Now that we have introduced the ingredients required to implement MOL-RK4 in numerical integration, we will move on to how these were applied throughout this project.

Chapter 2

Background and Methodology

In this section, we will look at how the aforementioned techniques were used to solve actual problems in this project. To be able to tackle the partial differential equations in numerical relativity, learning how to solve ordinary differential equations was a stepping stone; we begin with ODEs of the first order.

2.1 Ordinary Differential Equations

2.1.1 Solving First Order Differential Equations

The first ODE integrated in this project was a simple cosine function.

$$f^{'}(t,x) = \cos t \tag{2.1}$$

Naturally, one can infer that the solution to (2.1) is a sinusoidal function. This is useful as there is now an actual solution for our numerical solution to be compared with. The solution to this problem was found between $0 \leq t \leq 20$ with $\delta t = 1000$ and the initial conditions implemented were: $t_0 = 0$, $x_0 = 0$

Implementation

To begin with, the computational grid was constructed. The variables t_i, t_f and the number of iterations, N were defined and given the values stated above. The time-step, δt was then calculated by the program by dividing the range of spatial points by the number of iterations. The temporal grid was constructed using the *linspace* function to create an array containing $N + 1$ evenly distributed points. Next, the *zeros* function was used to create an empty 1-d array containing $N + 1$ points. The initial conditions discussed previously were expressed as an empty array and then the two empty slots were assigned values t_0, x_0 respectively. The main RK4 function was defined next, containing a *for*-loop for iteration i in the range $1 - N$. Equations (1.13) - (1.16) were defined next; for generality in the first program, the arguments of f contained and evolved the i^{th} element of t as well as x, eventhough this is unnecessary as the actual function in question is independent of x. The weighted average according to Simpson's Rule was next stated with the argument of t and x describing the the $(i + 1)^{th}$ iteration, and the right-hand-side containing the i^{th} iterations (previous values of t and x). Within the loop, another variable, x_{true} was also stated and evolved separately with its $(i + 1)^{th}$ value being iterated along with the i^{th} value of our actual solution, $\sin t$. This was done to be able to obtain numerical solutions for both functions for comparison. For this part of the code to work

however, an empty array had to be created that stored these values for x_{true}. After the function is called, the values for t, x and x_{true} were saved into a filed using the *savetext* function. Finally, another function was defined that allowed for the results from the RK4-function to be plotted using *Matplotlib*. The *loadtext* function was used to load the variables stored inside the text file containing t, x and x_{true}; two different plots were then created, axes were labelled appropriately, a title was created and boundaries for the plot were specified. The function was called last. Once desireable results were obtained and retained for use, the next problem was considered.

2.1.2 Solving Second Order Differential Equations

The next stepping stone in being able to solve problems in Numerical Relativity is the ability to solve second-order ODEs. When dealing with an ODE of the second order, it is very convenient to separate into a system of ODEs of the first order. This practice also applies to PDEs, which will be discussed later.

Simple Oscillating system (1+1 dim)

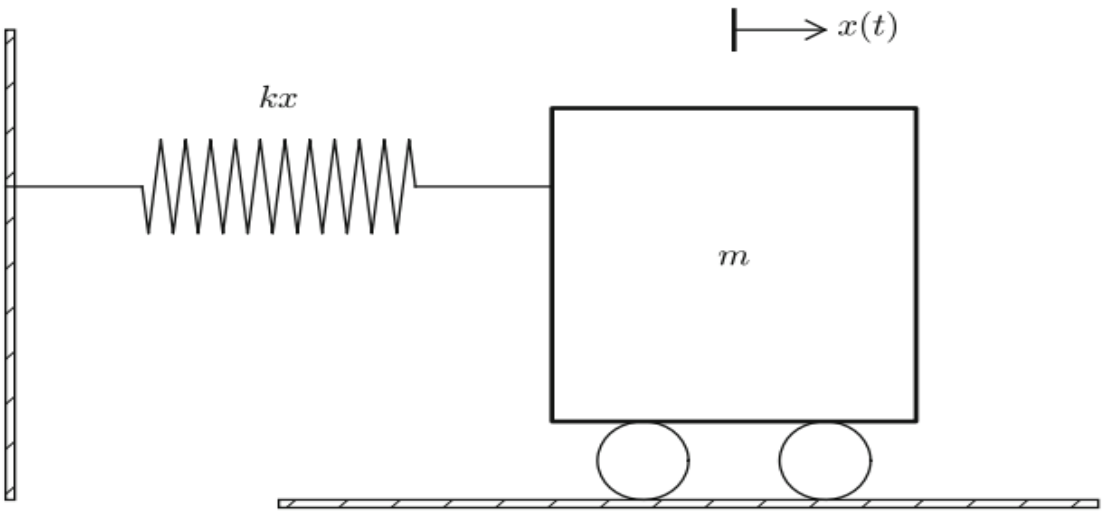


Figure 2.1: A figure describing an object of mass m attached to a spring with spring-constant k, which is allowed to move along a smooth surface in one direction only, x. Taken directly from [7].

In this part of the project, the simplest model of a one-dimensional oscillating system without damping was considered. See Figure 2.1. Using Newton's Second Law of motion, we begin with (2.2) where F is the force the spring exerts on the mass.

$$F = -kx = m\ddot{x} \tag{2.2}$$

The above equation can be rearranged and written more appropriately as (2.3) where we use that the angular frequency, $\omega = \sqrt{\frac{k}{m}}$

$$\ddot{x} = -\frac{k}{m}x = -\omega^2 x \tag{2.3}$$

We first decompose the above equation into first-order equations as follows

$$\dot{x} = u \tag{2.4}$$

$$\dot{u} = -\omega^2 x \tag{2.5}$$

Equations (2.4) and (2.5) were considered and evolved separately. The initial conditions chosen for this system are as follows:

- $t_0 = 0$
- $x_0 = 1$
- $u_0 = 0$
- $x_{true} = 1$

Implementation

For simplification, we let $\omega = 1$ at the beginning of the code for this task. Just as before, preliminary variables t_i, t_f, N were defined but unlike before, the *input* function was used for assigning values to them. Upon running the code, the user would be asked to manually assign values to each of the three variables, thus making it easier to manipulate the accuracy of the numerical solution by altering the step, N.
The computational grid was set up as before, along with the array for x_{true}, containing $N + 1$ points. These variables were defined globally, therefore the RK4 function needed no parameters. The initial conditions outlined above were implemented next.
Two separate functions which return (2.4) and (2.5) were defined next. Following this, the main RK4 function was defined, as before. This time however, there is a key difference; in the case for a system of differential equations (two in this case), one needs to define sets of equations for the variables $j_1 - j_4$ as well as $k_1 - k_4$, corresponding to (2.5) and (2.4) respectively. j_1 and k_1 described the x-jump over the time-step δt and were written out with the arguments x_i and u_i respectively. For the j_2 term, in the argument for $f2$, the x_i term was dependant on k_1. This followed on for j_3, k_3 to j_4, k_4. The main weighted average equations for u_{i+1} and x_{i+1} are dependant on the j and k - terms. The actual expected solution - a cosine function - was also evolved in this *for*-loop. It should be noted that, as with all differential equations, this expected solution is only unique given specific initial and and boundary conditions [3]. The plotting function mentioned in the previous task was implemented once again, to allow for comparisons to be drawn between the actual and numerical solutions - this allowed for viability of the code and methodology to be gauged; once approved, the next task for considered.

2.2 Partial Differential Equations

Classification of PDEs

Linear partial differential equations can generally be classified into three types of equations - elliptical equations, parabolic equations and hyperbolic equations. Elliptical equations have a negative discriminant and are encountered in steady-state problems describing quantities such as electrostatic potential and temperature distribution within a medium - do not contain time-derivatives. A prime example of such an equation is Laplace's equation (2.6). Such equations also usually are boundary-valued problems and change drastically in response to change in type

or value of these conditions. Parabolic equations and hyperbolic equations are similar to each other in that both are examples of initial-value problems.

$$\nabla^2 f = f_{xx} + f_{yy} + f_{zz} = 0 \tag{2.6}$$

Klein-Gordon equation (1+1 dim)

In this part of the project, the focus was on being able to solve the one-dimensional Klein-Gordon equation in Cartesian coordinates. Generally however, for a 3-dimensional case we have the following equation of motion (2.7):

$$\partial_\mu \partial^\mu \phi + m^2 \phi = 0 \tag{2.7}$$

As discussed before, a field, ϕ can be described as a quantity point-wise defined everywhere in space and time. We shall consider a real scalar field $\phi(\mathbf{x}, t)$, where $\mathbf{x}$ is a three-vector with components $\mathbf{x} = x^i = (x, y, z)$ and $t = x^0$. The Klein-Gordon equation (2.7), is a relativistic wave-equation that is derived by obtaining the Euler-Lagrange equations of motion (2.9) for the specific Lagrangian density (2.8) describing the dynamics of a real scalar field $\phi(x)$, which in itself interacts with scalar bosons. Bosons are particles with integer spin and the term 'scalar' assigns these bosons 0-spin.

$$L_{scalar} = (\partial_\mu \phi)(\partial^\mu \phi) - \frac{1}{2} m^2 \phi^2 \tag{2.8}$$

$$\partial_\mu \frac{\partial L}{\partial(\partial_\mu \phi_i)} - \frac{\partial L}{\partial_\mu \phi_i} \tag{2.9}$$

2.2.1 Massless case without gravity

To avoid unnecessary complications, only massless particles were considered, therefore the Klein-Gordon equation reduces to (2.10) when the time and spatial components are expanded out. Under this limit, the KG equation takes the form of the wave equation in 1+1 dimensions. In such a case, when ϕ is a wave function which describes massless particles such as photons, the wave nature of these particles is eluded to [5].

$$-\partial_t^2 \phi + \partial_x^2 \phi = 0 \tag{2.10}$$

In the interest of modelling a relatively simple process to begin with, the behaviour of a standing wave was selected for simulation; therefore a string of fixed length, L was considered. As we know, standing waves do not propagate and are produced as a result of interference between two waves. The system was designed such that only three nodes and two anti-nodes were expected in the simulation.
We refer to the general solution for such a system (2.11), where if the values of the wave-number, k were restricted to $k = \frac{n\pi}{L}$, we obtain the re-scaled initial profile of our solution where we have two sinusoidal waves travelling in opposite directions (2.12).

$$\phi(t, x) = \sin(kx - wt) + \sin(kx + wt) = 2\sin(kx)\cos(wt) \tag{2.11}$$

$$\phi(0, x) = \sin(\frac{2\pi x}{L}) \tag{2.12}$$

The domain for the simulation was set to be $x = [0, L = 1], t = [0, T = 1]$ and the number of time and spatial steps were set to be $N_t = 100, N_x = 50$. Both ends of the string were required to be fixed so that the formation of a standing-wave could be predicted; for this to be applied, the Dirichlet condition (1.17) was used. Thus we have $\phi(t, x) = 0$ at $x = 0, L$. The initial (2.13) and boundary conditions (2.14) chosen for this problem were as follows:

$$t_0 = 0, x_0 = 0, u_0 = 0 \tag{2.13}$$

$$\frac{\partial^2}{\partial x^2}\phi(t, 0) = \frac{\partial^2}{\partial x^2}\phi(t, L) = 0 \tag{2.14}$$

Implementation

This task differs form the previous tasks in that RK4 alone is no longer sufficient in integrating the whole equation as our problem is now dependant on two variables. This is where centred finite difference approximations were utilised alongside RK4. As before, the second-order PDE was separated into two first-order PDEs:

$$f_1 = \frac{\partial}{\partial t}\phi(t, x) = u \tag{2.15}$$

$$f_2 = \frac{\partial}{\partial t}u(t, x) = \frac{\partial^2}{\partial x^2}\phi(t, x) \tag{2.16}$$

To begin with, 1-dimensional arrays for x, t and the initial conditions, `IC` were defined using the *linspace* function, with each containing N_x points with the latter containing N_t points respectively. Following this, two empty 2-dimensional arrays for u and ϕ were also defined using the *zeros* function, with N_t rows and N_x columns each. Next, the initial conditions, (2.13) were implemented by allocating 0 to the 0^{th} element of the t, x, u, IC arrays. Here, the 0^{th} element for the u, IC arrays represents the value of the associated variable at t_0, x_0; this makes sense as from our initial profile, it can be easily seen that at its value at $x_0 = 0$. The initial condition for ϕ was applied by defining a function *phiprof* without arguments that returned the initial profile, (2.12) as all variables within were globally defined. Here, *numpy* was used to call $\sin$ instead of *math* as it was found that the former may be called on an array and is obviously capable of parallel computation whereas *math* only deals with scalars; this was a problem as the variable x in the initial profile was an array. All the remaining components of the IC-array were assigned the values of *phiprof* for $r \geq 1$ via slicing.
The functions representing our PDEs (2.15) - (2.16) were defined next, with the arguments u and ϕ respectively; while the function f_1 only returned the variable *u*, function f_2 was were the centred finite difference approximations were applied.

Within f_2, an empty 1-dimensional array of the same name was created. This allowed the creation of a *for*-loop in the range $1 - N_x$ within which the i^{th} iteration of the f_2-array was equated to (1.6). Preceding this loop, the our boundary conditions (2.14) were implemented by allocating zeros to the 0^{th} and N_x^{th} values in the f_2-array; this is possible as the *for*-loop only runs until the value preceding the right-extremum in the *range* function, thus allowing for the Dirichlet condition to be used.
Next, a function containing the RK4-loop was defined without arguments and once again, two sets of equations were coded in - for $j_1 - j_4$ as well as $k_1 - k_4$; this main solver function remained the exact same as that used in the previous task, save for the fact that ϕ was iterated

over in the loop instead of the x used previously - the function, f_2 was a function of ϕ only. Finally, to be able to see the evolution of ϕ with time, an animation was required; [3] was very illuminating in this and was referred to. Before introducing code, the *time*-module was imported alongside *Numpy*, *Math* and *Matplotlib*. Next, the spatial components of the ϕ-array were extracted and allocated to a new variable y by once again using slicing. Following this, the 'wall-time' measured by the computer is called upon using *time.clock* and equated to new variables t_0, t_1; this allows for information regarding elapsed time between these variables. Interactive mode was switched on using *pyplotion* to allow for each *pyplot* command to be implemented automatically without being explicitly called. The plot was then given a title and had its axes labelled using *pyplot*. We next declare a new variable *lines* which describes the behaviour of x against y; this will be modified during the animation later. We also define a variable *counter* which is assigned the value 0. The last thing to do was to create a *for*-loop in the range between 0 and the temporal variables in y. Within this loop, we require the current iteration of time, t_i to be printed, for the *lines*-object to be updated in every iteration using *set_ydata* and for the actual figure to be drawn and image to be saved in a dedicated folder. When assigned to a *counter*, += adds a value to the variable and then replaces the variable's current value - this allowed for images to be saved after every iteration. Finally, the images produced were combined together using *ffmpeg* to create a movie. We now move on to a more suitable problem to solve using MOL-RK4. In order to model the behaviour of fields in numerical relativity, it is more convenient to work in spherical coordinates. This intuitively makes sense as most stellar bodies can be assumed to be nearly spherically symmetric. Consequently, the penultimate task in this project was to simulate the behaviour of a spherically symmetric scalar field in the absence of gravity.

2.3 Massless Scalar Fields in Spherical Symmetry

We of course consider the Klein-Gordon equation in the spherical coordinate system (t, r, θ, ϕ) [9] next, as opposed to the Cartesian coordinate system (t, x, y, z) described above. When the motion of ϕ is restricted to radial propagation only, $d\theta = d\psi = 0$ is assumed as the values for θ and ψ remain constant; consequently, the $3+1$-dimensional system reduces once again to a $1+1$-dimensional system - must easier to simulate. We first state the equation of motion for our field in a spherically symmetric spacetime:

$$\frac{\partial^2}{\partial t^2}\phi(t,r) = \frac{2}{r}\frac{\partial}{\partial r}\phi(t,r) + \frac{\partial^2}{\partial r^2}\phi(t,r) \tag{2.17}$$

The chosen domain for this simulation was $r = [0, r_f = 15]$ and $t = [0, T = 10]$ and $N_t = 500, N_x = 200$ to achieve stable simulations. Before any programming, the first step was to give the wave an initial profile. A Gaussian packet Figure 2.2 was chosen for the profile; the general form for a Gaussian pulse is given in (2.18).

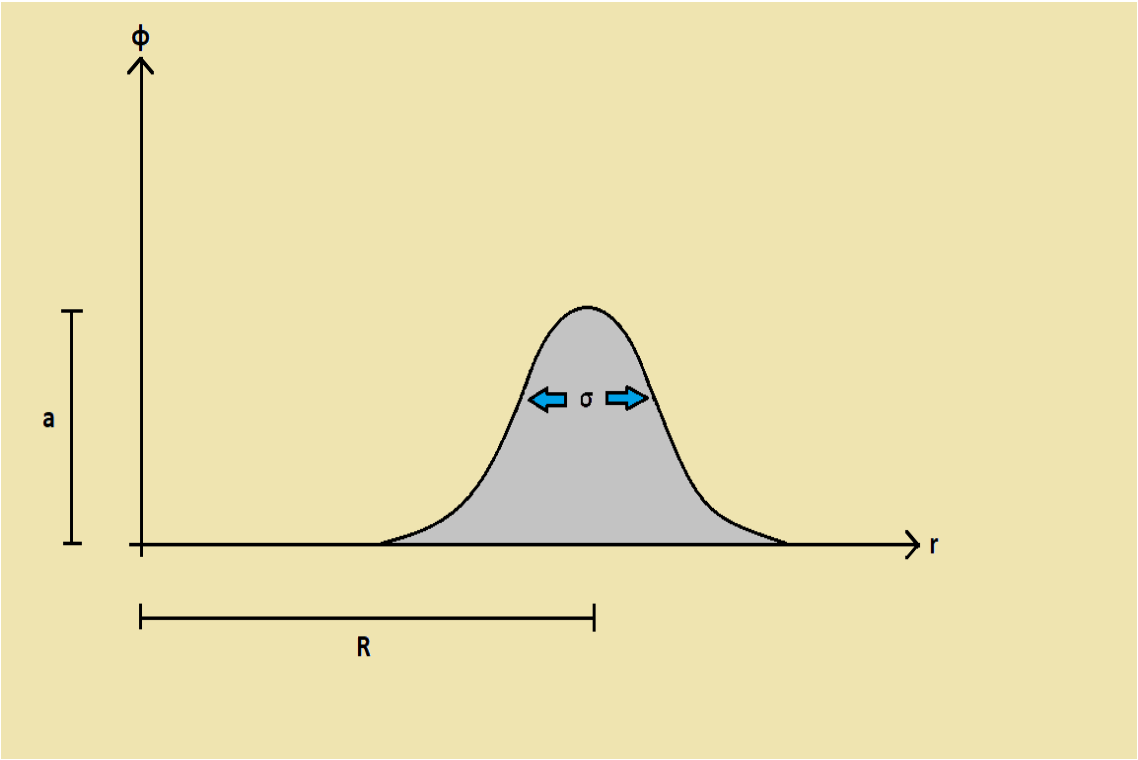


Figure 2.2: A typical Gaussian distribution is shown here. a denotes its normalised amplitude, σ refers to its standard distribution and R is the variance. In this case however, as a pulse of the same shape is being simulated, a, R and σ are simply constants that are assigned values

$$\phi(0, r) = a \exp^{-\frac{(r-R)^2}{\sigma^2}} \tag{2.18}$$

where the amplitude, a is given by

$$a = \frac{1}{(2\pi R)^{\frac{1}{2}}} \tag{2.19}$$

The initial conditions for this simulation remained the exact same as those from the previous task, however there were key differences in the boundary conditions. We know from ([7]) that a typical solution takes the form of two waves travelling in opposite directions; we impose conditions on each of these. The pulse was predicted to separate into two different spherical waves travelling away from each other, with the wave travelling to the left expected to grow larger in amplitude as it approached $r = 0$ and that travelling to the right expected to lose amplitude. This was expected as on inspection, one can immediately see from (2.17) how as $r \to \infty$, the RHS becomes smaller due to the inversely proportional relationship with r and how the opposite is true for $r \to 0$. On contact with the origin, the wave was required to be mirrored back into the system, thus the Neumann condition (1.18) had to be applied for the left-pulse. The behaviour of the right-pulse was considered relatively unimportant and so the Dirichlet condition was applied at $r = 15$.

Implementation

The bulk of this code remained the same as the Cartesian case. Key differences showed themselves in the initial profile and the centred finite difference approximations for all spatial derivatives. The former has already been discussed, so only the latter will be explained. As before,

the equation of motion was re-expressed as a system of first-order PDEs

$$f_1 = \frac{\partial}{\partial t}\phi(t,r) = u \tag{2.20}$$

$$f_2 = \frac{\partial}{\partial t}u(t,r) = \frac{2}{r}\frac{\partial}{\partial r}\phi(t,r) + \frac{\partial^2}{(\partial r)^2}\phi(t,r) \tag{2.21}$$

The finite difference approximations for the spatial derivatives are shown below (2.22).

$$(f_2)_i = \frac{2}{r_i}\frac{\phi_{i+1} - \phi_{i-1}}{2\delta r} + \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{(\delta r)^2} \tag{2.22}$$

When this approximation is applied at the boundary $i = 0$,

$$\frac{\partial}{\partial r}\phi(t,0) = \frac{\phi_1 - \phi_{-1}}{2\delta r} = 0 \tag{2.23}$$

and so we are left with

$$(f_2)_0 = \frac{\phi_1 - 2\phi_0 + \phi_{-1}}{(\delta r)^2} \tag{2.24}$$

which clearly contains the point $r = -1$ which lies outside the domain. To overcome this numerical obstacle, we simply use (2.23) to let $\phi_1 = \phi_{-1}$. We now obtain a condition applicable at the boundary

$$(f_2)_0 = \frac{2}{\delta r^2}(\phi_{+1} - \phi_0) \tag{2.25}$$

and we reintroduce the Dirichlet condition (2.26.

$$(f_2)_{N_r} = 0 \tag{2.26}$$

For completeness, the exact initial profile that was used is explicitly stated below.

$$\phi(0,r) = \frac{1}{(\pi)^{\frac{1}{2}}}\exp^{-2(r-5)^2} \tag{2.27}$$

2.3.1 3+1 decomposition of Einstein's equations

To be able to simulate the behaviour of a scalar field in the presence of gravity, one would naturally look towards Einstein"s equations (2.28), which describe the dynamics and the matter content of the universe, and how each affects the other; more specifically, how the Einstein tensor, $G\mu\nu$ describes the curvature produced in space-time as a result of the matter content present in the vicinity, which the stress-energy tensor, $T_{\mu\nu}$ corresponds to.

$$G_{\mu\nu} = 8\pi T_{\mu\nu} = R_{\mu\nu} - \frac{1}{2}g_{\mu\nu}R \tag{2.28}$$

When attempting to solve problems in General Relativity using numerical methods however, we encounter a blaring problem. General Relativity treats space-time as a 4-dimensional construct, without any significant distinction. We have seen how numerical methods can be extremely effective in solving evolution equations; but we have been able to treat time and space separately, thus allowing for the implementation of MOL-RK4 for example. Intuitively, one must reproduce these equations in a system where numerical integration can take place. In this section, the explanations in[3] to briefly summmarise how space and time are split up under the '3 + 1'-formulation.

Foliation of 4-D Space-time

Under classical dynamics, unique solutions to initial-value problems are usually solved by using the initial positions and velocities for a given system. For evolution of a gravitational field, there exist analogues to these initial conditions - components of the metric tensor, $g_{\mu\nu}$ and their first time-derivatives $\dot{g_{\mu\nu}}$. These quantities are required to be defined everywhere on a spacelike hypersurface Figure 2.4

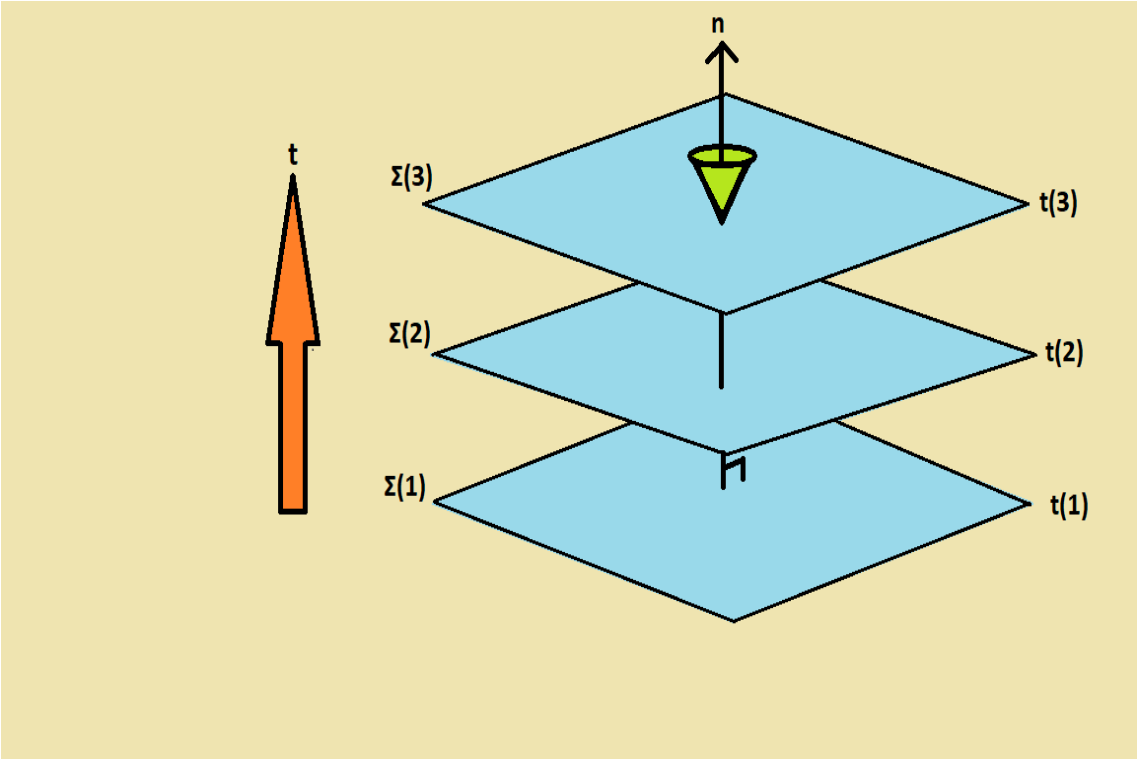


Figure 2.3: This figure is a depiction of something called *synchronization*, where 4-dimensional space-time is foliated such that the result is curved, spacelike hypersurfaces separated in time by δt

The idea is to obtain $\partial_t^2 g_{\mu\nu}$ from Einstein's equations (2.28) at every point x_i on a hypersurface, Σ_1 then integrate forward in time to find $\partial_t g_{\mu\nu}$ and $g_{\mu\nu}$ on the next hypersurface Σ_2 at $t + \delta t$, and then repeat. Now, these 3-dimensional hypersurfaces carry an *interior metric*, $\gamma_{\mu\nu}$ due to their internal geometry and these can then be embedded into standard 4-dimensional space-time which carries an *exterior curvature*. This quantity is simply a measure of how the normal vector, n changes under parallel transport to different points and is denoted by $K_{\mu\nu}$. Interestingly, it turns out that the extrinsic curvature is related to the first time-derivative of the the spatial metric $\gamma_{\mu\nu}$ and so can be thought of as analogues of position and velocities [3].

Gauge functions and ADM equations

In this formalism, the sixteen degrees of freedom in (2.28) must be reduced. Luckily four of these d.o.f can be dealt with through the choice of gauge functions - the lapse function, α and the shift vector, β. These can be stated arbitrarily.
$\gamma_{\mu\nu}$ however cannot be stated arbitrarily and must satisfy certain constraint equations - the *Hamiltonian constraint* (2.29) and the *Momentum constraint* (2.30). Using the purely timelike

and spacelike quantities n and γ, one is able to create projection operators that can decompose all 4-dimensional tensors into their distinct temporal and spatial components. After decomposing Einstein's equations (2.28), one obtains the four constraint equations stated below (2.29), (2.30) which are independent of time derivatives and are applicable everywhere on the spacelike hypersurfaces; we also obtain the equations which deal with the evolution of the field between the hypersurfaces - evolution of the extrinsic curvature (2.31) and spatial metric (2.32).

$$R + K^2 - K_{\mu\nu}K^{\mu\nu} = 16\pi\rho \tag{2.29}$$

$$D_\mu K^\mu_\nu - D_\nu K = 8\pi S_\nu \tag{2.30}$$

$$\partial_t K = -D^2\alpha + \alpha(K_{\mu\nu}K^{\mu\nu} + 4\pi(\rho + S)) + \beta^\mu D_\mu K \tag{2.31}$$

$$\partial_t \gamma_{\mu\nu} = -2\alpha K_{\mu\nu} + D_\mu \beta^\mu \tag{2.32}$$

Where R is the Ricci scalar, K is the trace of the extrinsic curvature tensor, D_ν is the 3-dimensional covariant derivative, ρ, S_ν are source terms corresponding to matter that depends on the field being considered and K and S are traces of their corresponding tensors. (2.29) and (2.30) must be satisfied for these 3-dimensional hypersurfaces to be embedded into the required 4-dimensional spacetime. Equations (2.29) - 2.32) are a result of the ADM (Arnowitt, Deser, Misner) formulation and are equivalent to Einstein's equations in 4-dimensional spacetime.

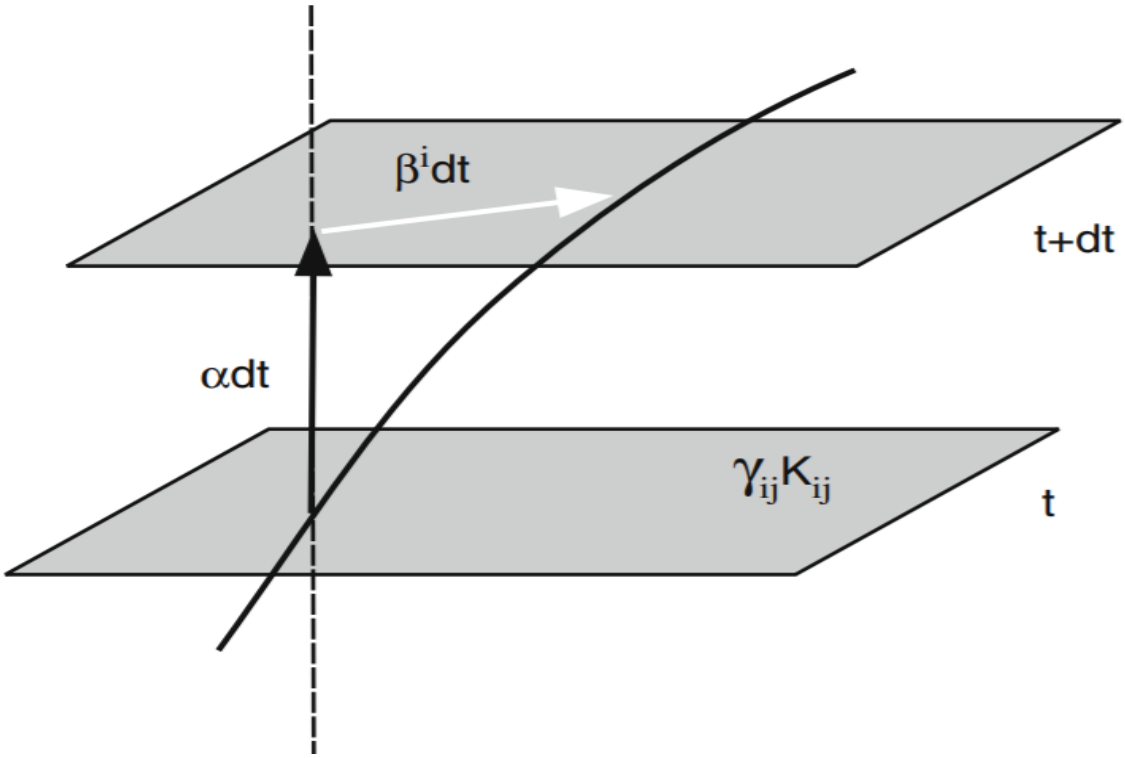


Figure 2.4: This diagram shows how the lapse function, α is a measure of the proper time that elapses between the time slices along the vector normal to the hypersurfaces. It also shows the shift vector, which is a measure of the amount of deviation between the proper time lines and the actual time lines. Taken directly from [4]

Gauge freedom

As discussed in the previous section, gauge functions can have arbitrary values, but some choices are superior to others. A choice for the lapse function is called a slicing condition, and the most basic conditions one can come up with would be to let $\alpha = 1$ and $\beta = 0$ - we fine Geodesic slicing. This name derives itself from the fact that at $\beta = 0$, there is no displacement in coinciding spatial points across the slices; if $alpha = 1$, we have that the coordinate time coincides with the proper time (proper time is directly proportional to coordinate time with α being the constant of proportionality). Due to the vanishing proper acceleration of the observers, they can be said to now be following timelike geodesics. In any case, this condition naturally takes advantage of any non-uniform gravitational fields, producing coordinate singularities in the process. For us to be able to model the behaviour of a scalar field in spherically symmetric spacetime, we desire a singularity-avoiding slicing condition, to avoid crashes during simulations.
We shall let $\beta = 0$ for our final task for simplification. We now look at another choice of slicing - the Maximal slicing condition. To avoid the drawbacks as the Geodesic slicing condition, we require the volume elements between observers to be constant [1]. It turns out, this can be implemented by setting the trace of the exterior curvature, $K = 0$ initially as well as during evolution; so we also have that $\partial_t K = 0$. Applying these statements to (2.31) gives us our equation for evolving α 2.33.

$$D^2\alpha = \alpha(K_{\mu\nu}K^{\mu\nu} + 4\pi(\rho + S)) \tag{2.33}$$

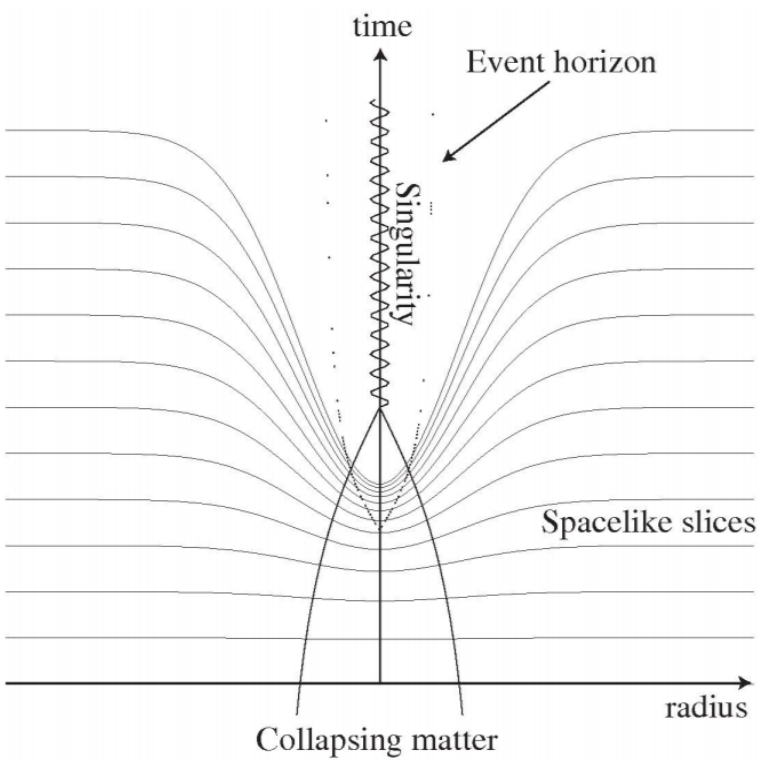


Figure 2.5: From this figure, one can see how as singularities are approached, the lapse 'collapses' and drops drastically. Proper time essentially freezes in this region but runs normally outside, thus allowing for long, relatively stable simulations. Figure taken directly from ([1]).

2.3.2 Scalar fields

In order to obtain the equation of motion that dictates the dynamics of a real, massive scalar field, we first state the metric [1]:

$$ds^2 = -\alpha^2(t,r)dt^2 + A(t,r)dr^2 + r^2B(t,r)d\theta^2 + r^2Sin^2\theta B(t,r)d\psi^2 \tag{2.34}$$

The equation of motion can be found using (2.35):

$$-\partial_t(g^{tt}\sqrt{-g}\partial_t\phi) = \partial_r(g^{rr}\sqrt{-g}\partial_r\phi) \tag{2.35}$$

$$\partial_t(\frac{A^{1/2}B}{\alpha}\partial_t\phi) = \frac{1}{r^2}\partial_r(\frac{\alpha Br^2}{A^{1/2}}\partial_r\phi) \tag{2.36}$$

$$\partial_t\Pi = \frac{1}{r^2}\partial_r(\frac{\alpha Br^2}{A^{1/2}}\Psi) \tag{2.37}$$

Where we have set $\Pi = \frac{A^{1/2}B}{\alpha}\partial_t\phi$ and $\Psi = \partial_r\phi$. After the ADM evolution and constraint equations are recast for the case of spherical symmetry, they can be satisfied as follows:

- To satisfy momentum constraint equation, let $K^r_r = K^\theta_\theta = K^\psi_\psi = 0$
- To satisfy Hamiltonian constraint equation, we first let $B = 1$ and the solve equation (2.38)

$$\partial_r A = A(\frac{1}{r}(1-A) + 8\pi r\rho) \tag{2.38}$$

Where $\rho = \frac{1}{2A}(\Pi^2 + \Psi^2)$. Next, we need to implement the Maximal slicing condition. Applying the statement above for the trace of the exterior curvature tensor to simplify the condition for Maximal slicing (2.33) reduces it to a second-order PDE (2.39) to be integrated alongside (2.38).

$$\frac{\partial^2\alpha}{\partial r^2} = \frac{8\pi\alpha\Pi^2}{A^2} \tag{2.39}$$

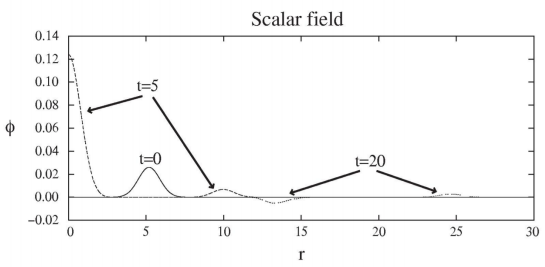
This second-order PDE should be split into first-order PDEs, just as with all others before it Penultimately, initial profiles for the functions ϕ and Ψ are declared in (2.40) and (2.41) respectively.

$$\phi(0,r) = ar^2\exp^{-(r-5)^2} \tag{2.40}$$

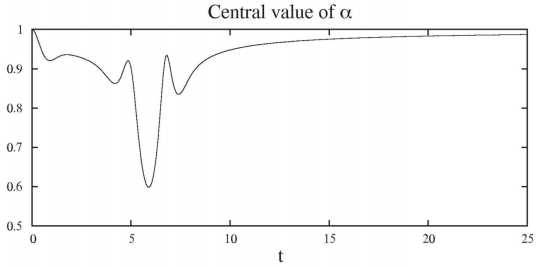
$$\Psi(0,r) = \exp^{-(r-5)^2}(2ar - 2ar^2(r-5)) \tag{2.41}$$

Implementation

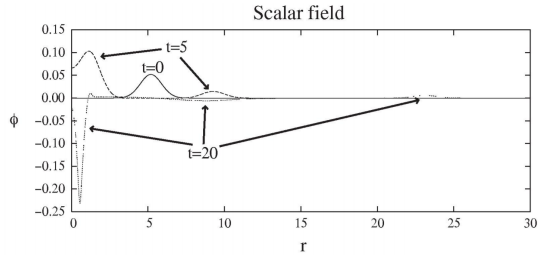
The programming for this part of the project was a lot more difficult than all preceding tasks due to the sheer instability of the system. According to Miguel Alcubierre in [1], we were to produce the results shown below.



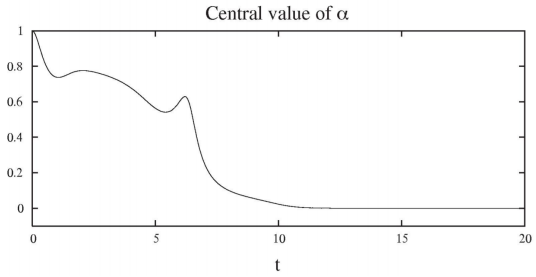
(a) Data produced for amplitude, $a = 0.001$



(b) Collapse of lapse function for $a = 0.001$



(c) Data produced for amplitude, $a = 0.002$



(d) Collapse of lapse function for $a = 0.001$

Unfortunately, the results obtained were only produced in very special situations. For example, the slicing condition (2.39) turned out to be the cause of many errors and had to be replaced by the Polar-slicing condition [3]. To deal with the instabilities, Kreiss-Oliger dissipation[1] was implemented on all the variables that centred finite difference approximations were applied to. In addition to this, an altered version of the finite difference approximations was applied to $\partial_t \Pi$ on recommendation in [1] due to the existence of the $\frac{1}{r^2}$-term. Even the application of the three-step iterative Crank-Nicholson scheme was attempted, albeit unsuccessfully.

Choptuik's Critical Collapse

Eventhough collapse of the lapse function was not achieved, we see how theoretically there exists a critical value for the amplitude a, which if exceeded would lead to black-hole behaviour. This was an example of Choptuik's research that showed how critical phenomenon underpins many physical processes.

Chapter 3

Results

3.1 ODE Solvers

3.1.1 First Order ODE

Figure 3.1 is the result of the very first simulation carried out in this project. The actual solution - a sine wave is shown here in the red-dashed line. It is clear that for a sufficiently large time-step of 0.2 for example, even the RK4 method tends to produce inaccurate numerical results. In the next run of this simulation, the time-step reduced to 0.02; immediately we can see much finer results. This is expected as a lower time-step corresponds to a higher number of numerical iterations. One can see from Figure 1.3 that a higher number of iterations lead to a much more accurate model for the actual curvature of a function.

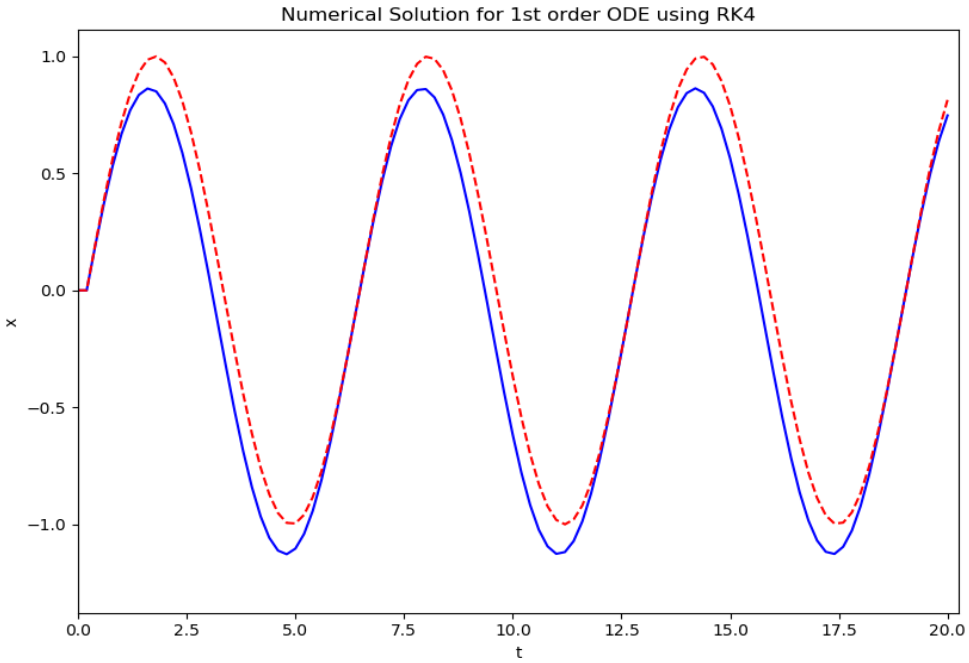


Figure 3.1: The first numerical solution produced when a simple cosine function was integrated at time-step, $\delta t = 0.2$. Due to the inaccuracy of this run, the simulation was repeated

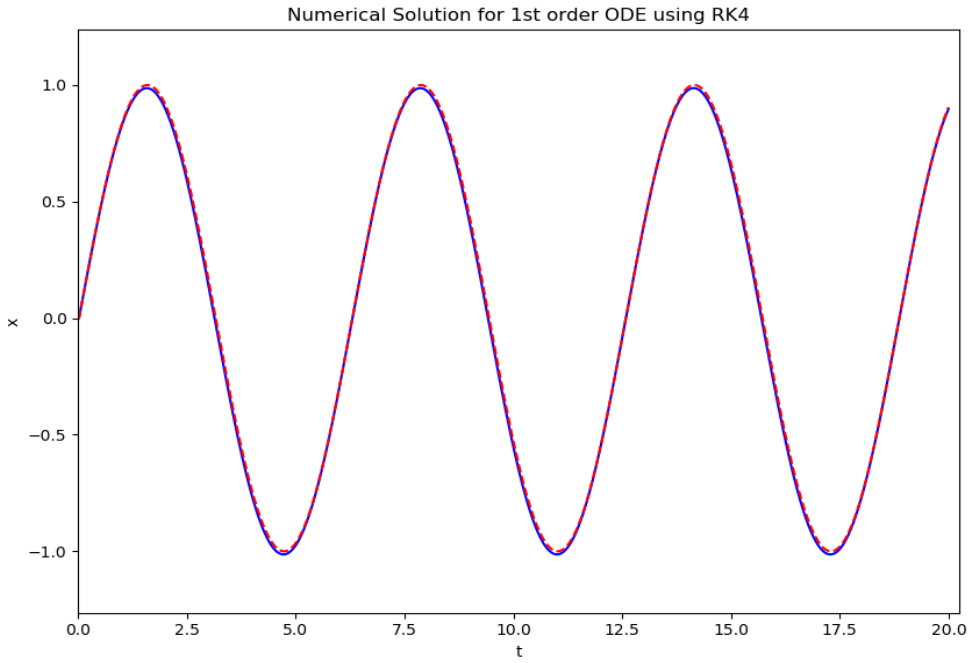


Figure 3.2: The second and final simulation produced when a cosine function was integrated at time-step, $\delta t = 0.02$. This result was sufficiently accurate and was approved.

In retrospect, more simulations could have been carried out to prove the affect of the duration of the simulation on the accuracy of the numerical solutions.

3.1.2 Second Order ODE

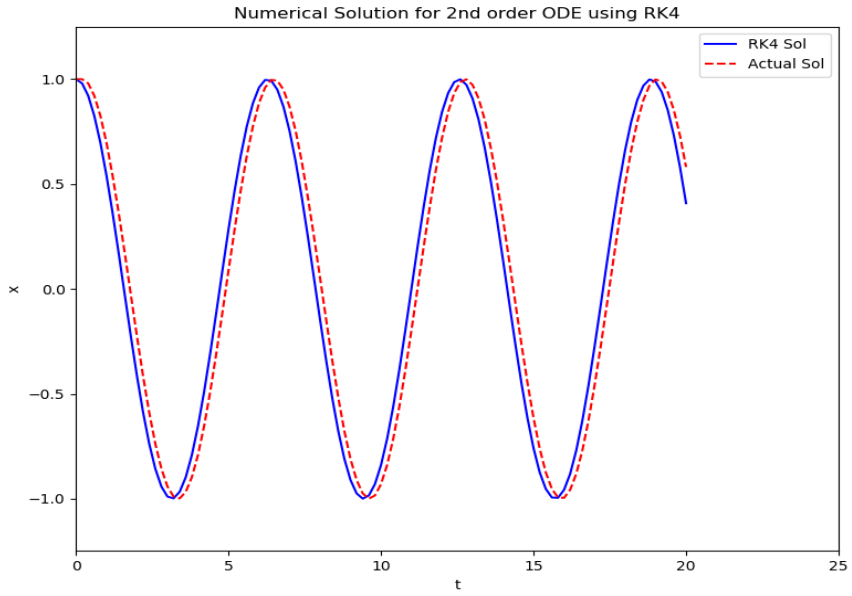


Figure 3.3: The first simulation in this task produced this cosine wave. It was carried out at time-step, $\delta t = 0.2$ and while there is some inaccuracy, the numerical solution seems to be sufficiently close to the actual solution

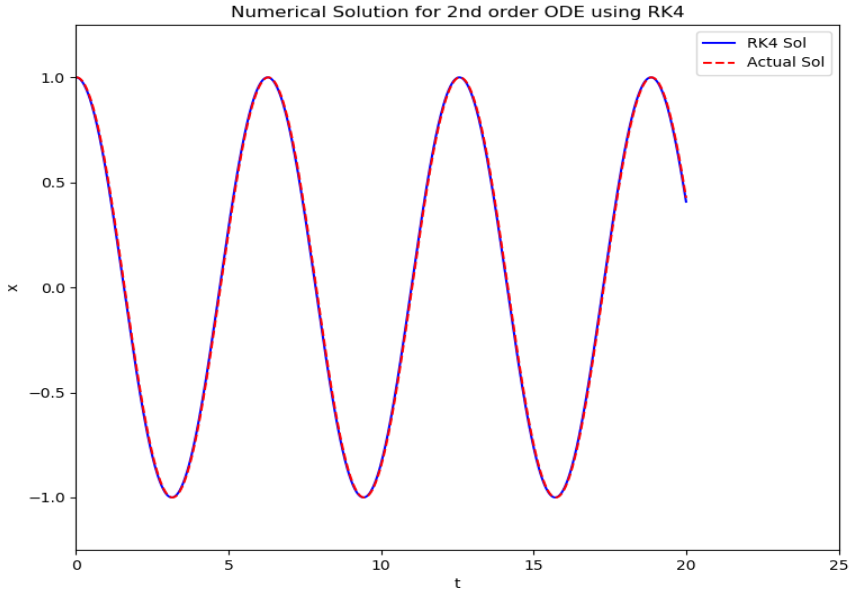
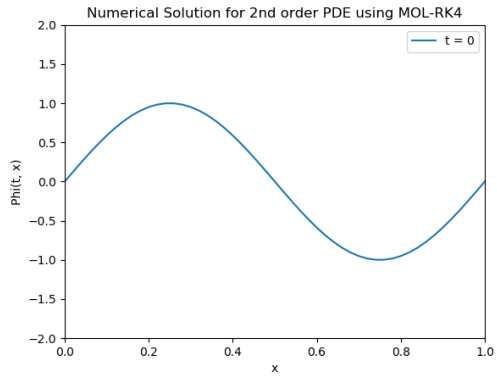
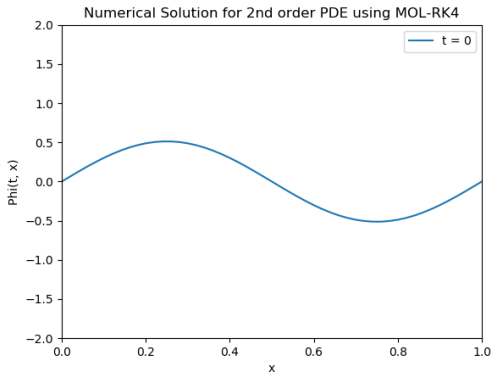


Figure 3.4: The final simulation in solving second order ODEs was carried out at time-step, $\delta t = 0.02$, clearly displaying a much more accurate solution given the smaller time-step

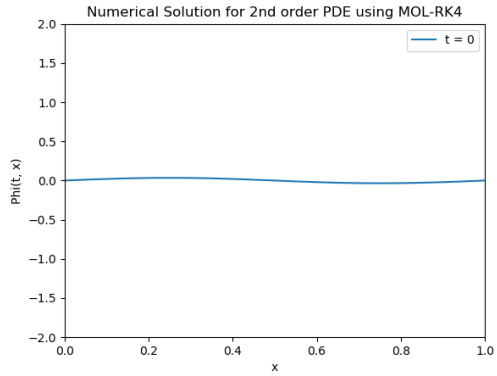
3.2 Simple Harmonic Motion in a fixed string



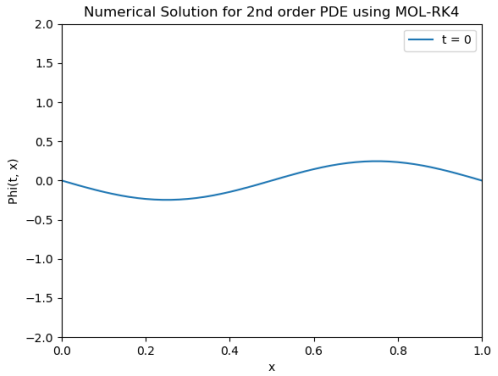
(a) Iteration 1/100



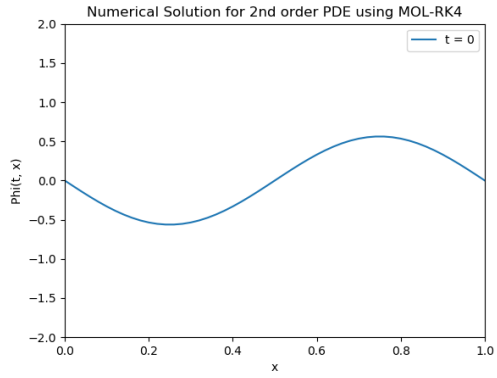
(b) Iteration 33/100



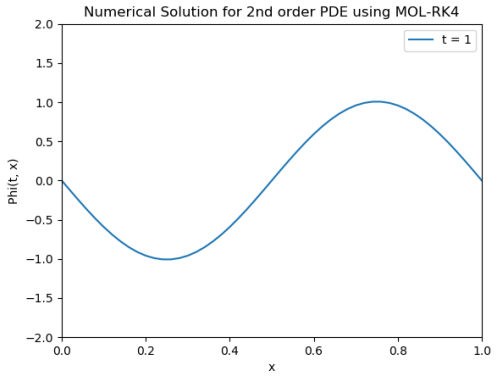
(c) Iteration 49/100



(d) Iteration 58/100



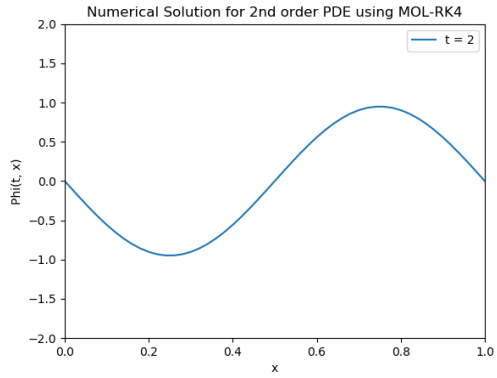
(e) Iteration 69/100



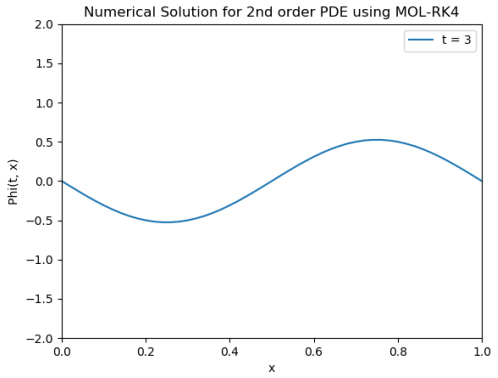
(f) Iteration 100/100

Figure 3.5: This figure shows the evolution of our solution over time through snapshots of the animation that was created. As expected, a standing wave was produced with two nodes and three antinodes

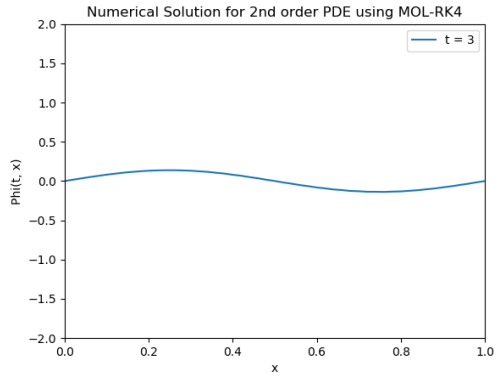
During this simulation, the sensitivity of these algorithms was highlighted. In the attempt to observe the effects of duration on stability, the time period was increased from $1 \rightarrow 10$, while the time-step was kept constant. The outcome is shown below.



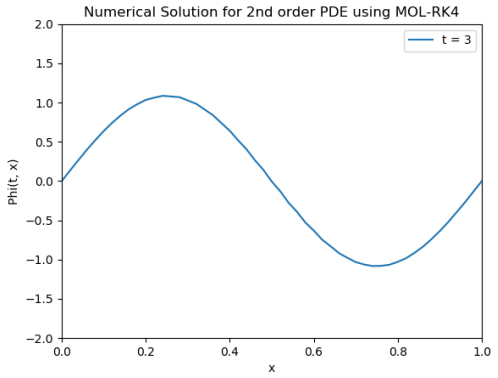
(a) Iteration 277/1000



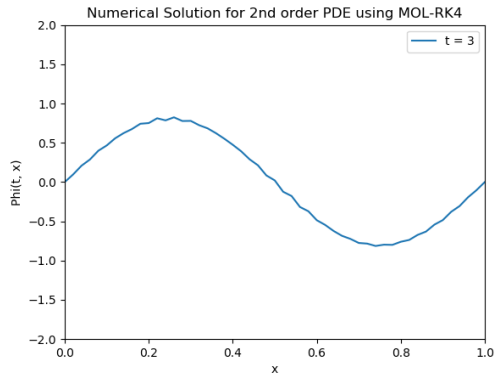
(b) Iteration 297/1000



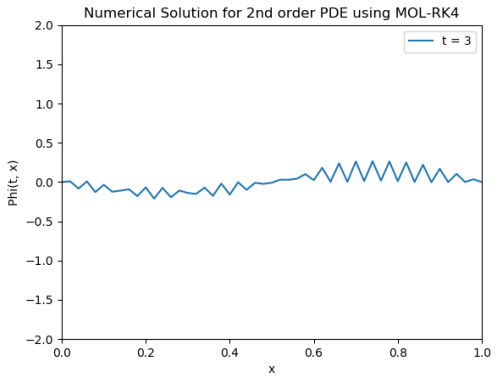
(c) Iteration 277/1000



(d) Iteration 297/1000



(e) Iteration 312/1000

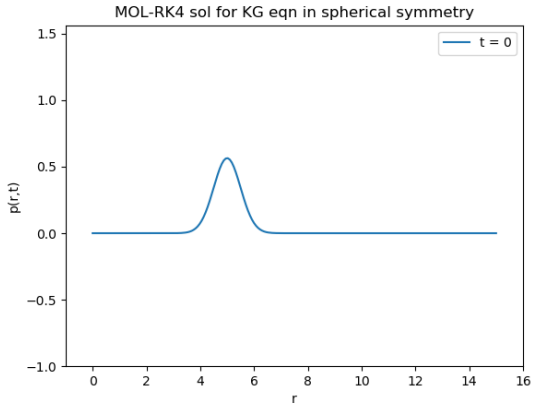


(f) Iteration 327/1000

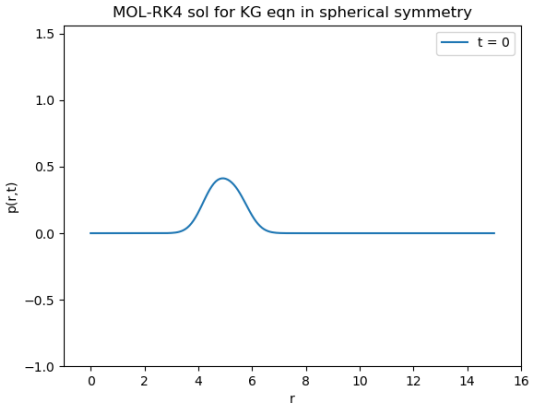
Figure 3.6: This figure shows the evolution of our solution over time through snapshots of the animation that was created. Unexpectedly, system crashed rapidly around the 312^{th}-iteration

It seems to be wise to minimise the duration of the simulation as a longer duration corresponds with errors due to values rounding-up and the error due to truncation of the infinite series growing over time.

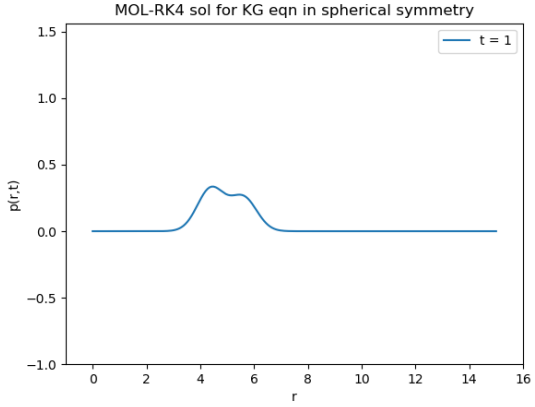
3.3 Scalar field in Spherical Symmetry



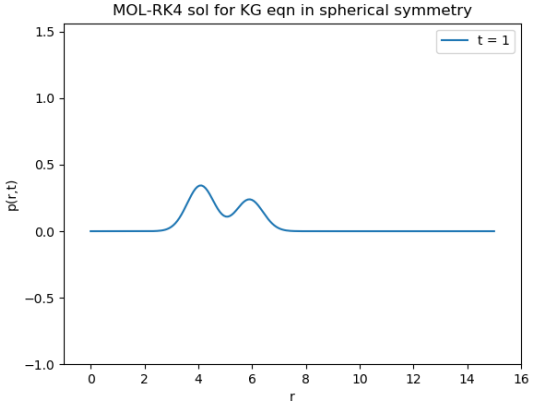
(a) Iteration 1/500



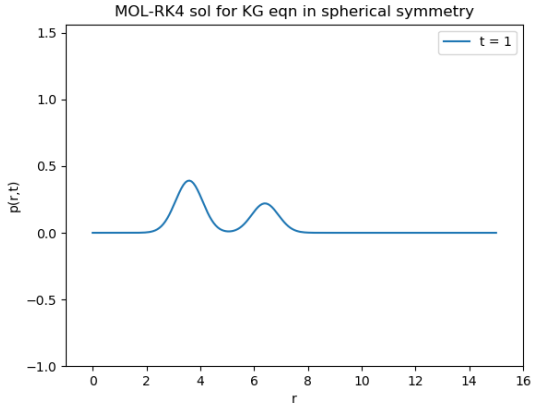
(b) Iteration 20/500



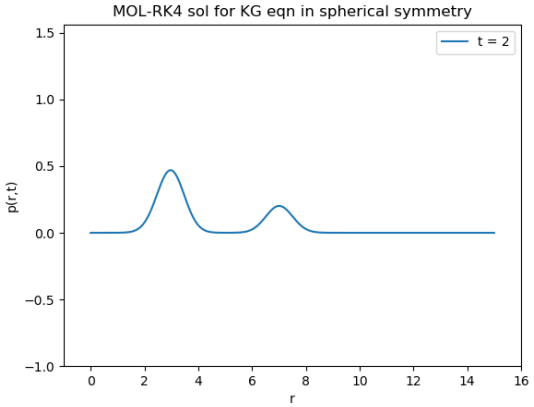
(c) Iteration 30/500



(d) Iteration 45/500

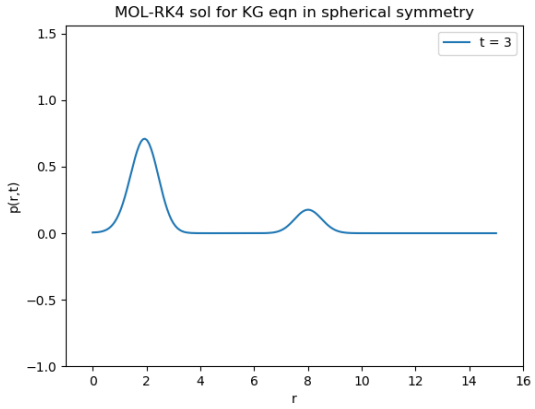


(e) Iteration 70/500

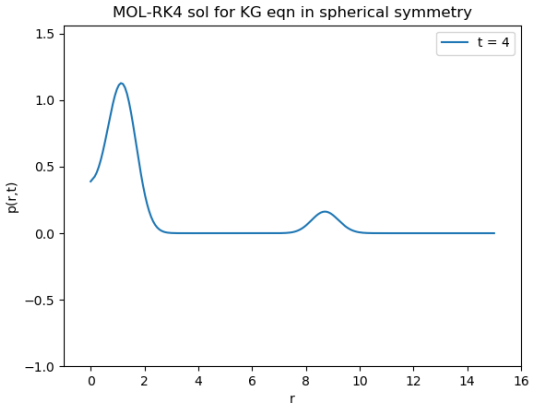


(f) Iteration 100/500

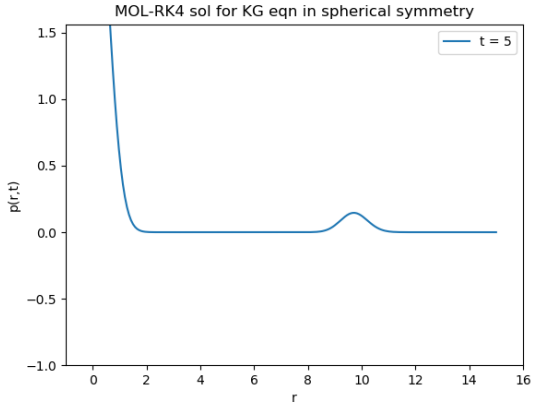
Figure 3.7: This figure shows the evolution of our solution over time. As expected, the initial Gaussian pulse splits into two separate waves that travel in opposite directions



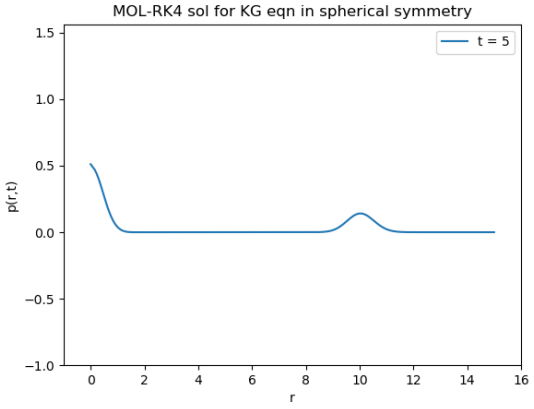
(a) Iteration 150/500



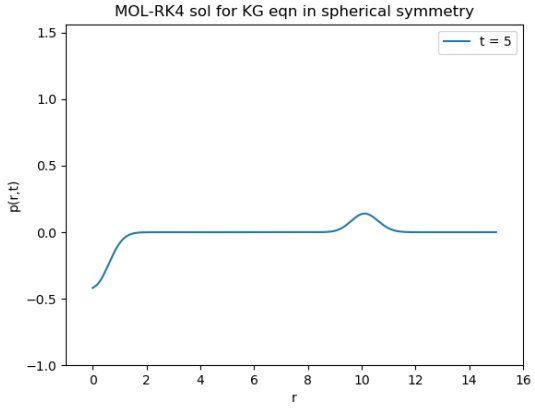
(b) Iteration 185/500



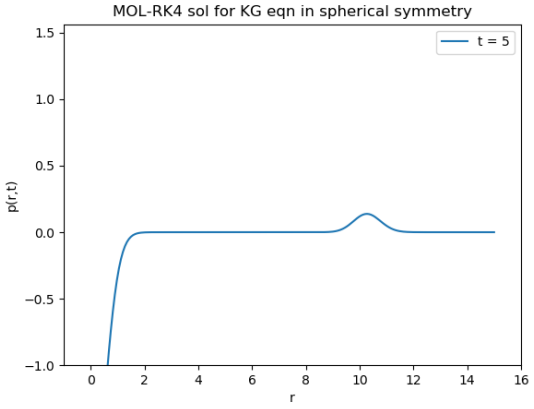
(c) Iteration 235/500



(d) Iteration 251/500

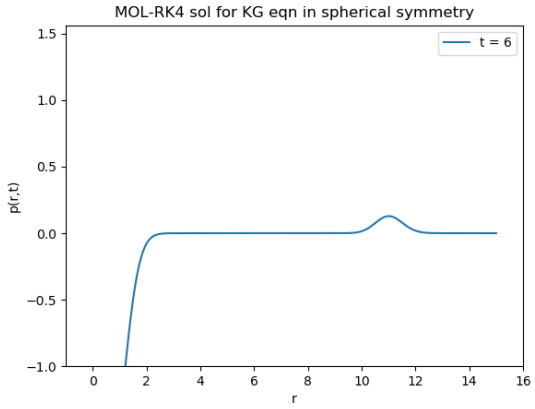


(e) Iteration 255/500

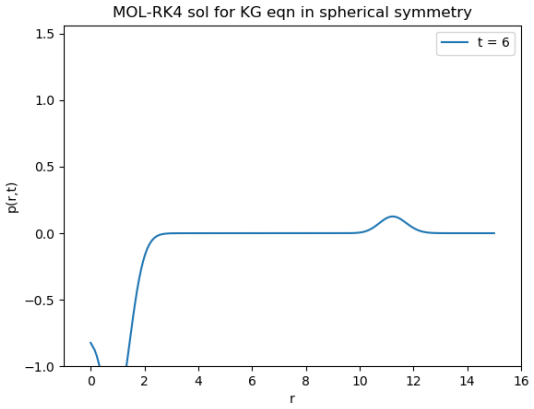


(f) Iteration 263/500

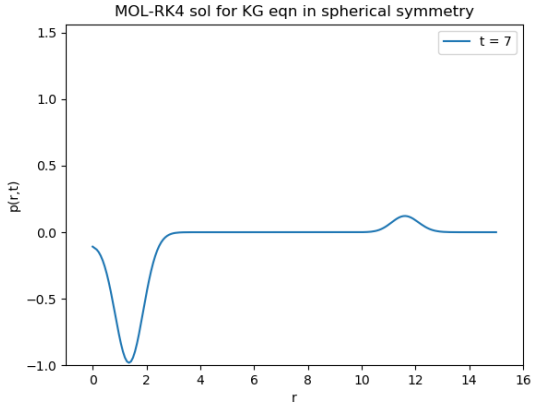
Figure 3.8: This figure shows the evolution of our solution over time. As expected, the initial Gaussian pulse splits into two separate waves that travel in opposite directions



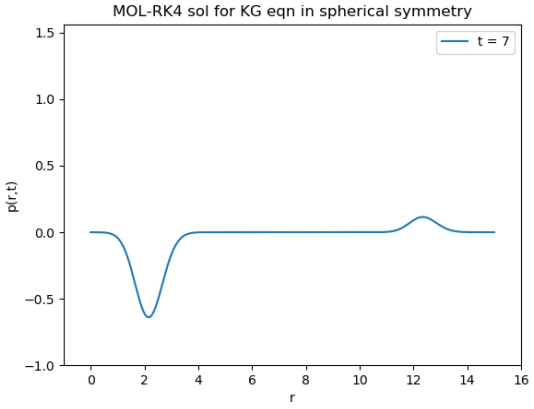
(a) Iteration 300/500



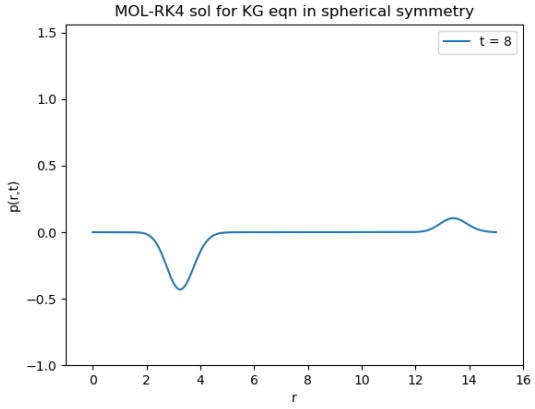
(b) Iteration 311/500



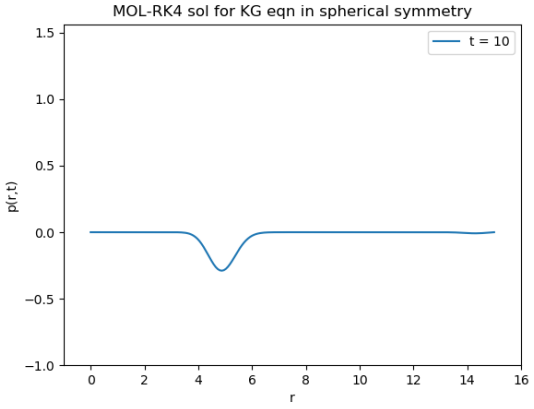
(c) Iteration 330/500



(d) Iteration 367/500



(e) Iteration 420/500

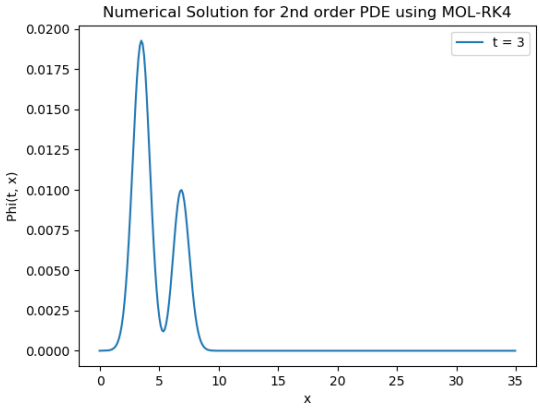


(f) Iteration 500/500

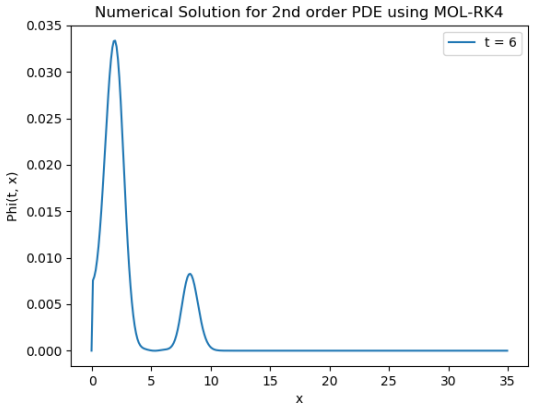
Figure 3.9: This figure shows the evolution of our solution over time. As expected, the initial Gaussian pulse splits into two separate waves that travel in opposite directions

The behaviour of this solution was just as expected. At $t = 0$, the Gaussian pulse begins to split apart as per (2.11), with two separate waves travelling in opposite directions. As the wave on the left approaches $r = 0$, the contribution to its magnitude increases due to the inverse proportionality. The wave travelling on the left however is allowed to diminish until it reaches the end of the domain where the Dirichlet condition mirrors its behaviour within the domain, as can be seen from iterations 420-500.

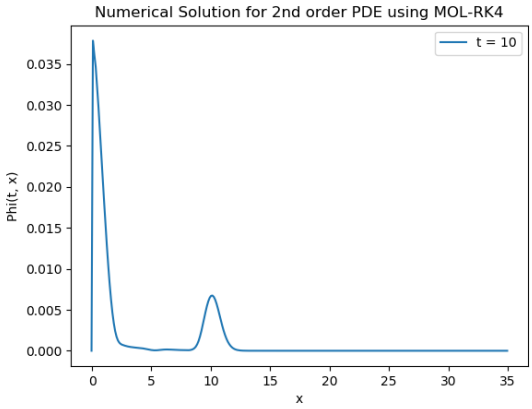
3.4 Scalar field collapse (with gravity)



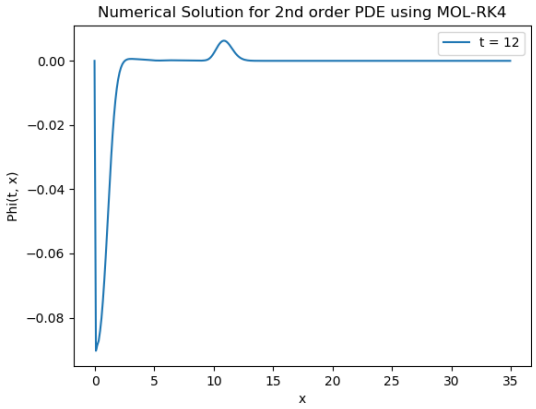
(a) Iteration 300/500



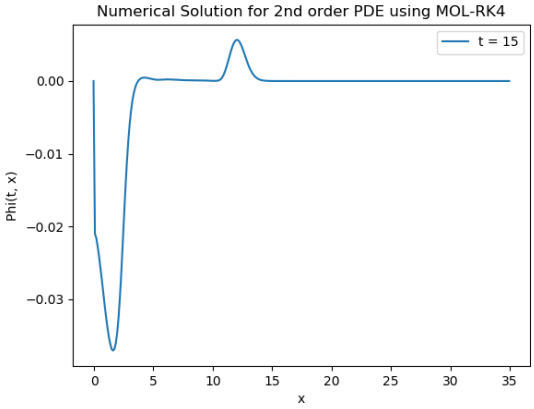
(b)



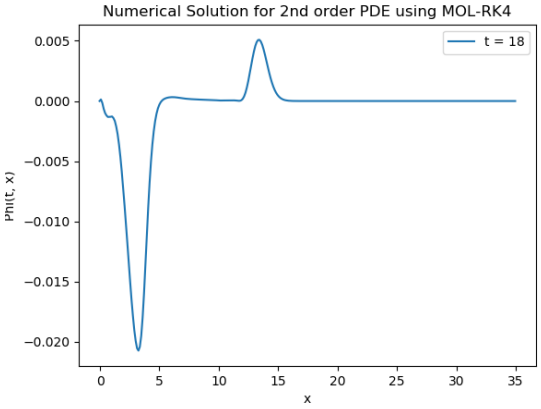
(c)



(d)



(e)



(f)

Figure 3.10: This figure shows the evolution of our solution over time. As expected, the initial Gaussian pulse splits into two separate waves that travel in opposite directions. The solution is very similar to that in flat space, except for the behaviour close to the origin

Chapter 4

Conclusion

In conclusion, the application of numerical methods to complicated physical phenomenon was thoroughly realised. Every simulation was repeated until stable solutions found and the link this has to the CFL condition was realised. In future endeavours, I hope to become well versed with vectorization which would allow me to achieve much faster computational speeds - very important for systems where a lot of computational time and power is used, for example, in the simulation of the collapse of a scalar field in spherical symmetry. I would also like to learn more about adaptive step-size methods as picking an appropriate value for the time-step is always very wasteful. I also want to better understand why the system in the final task was so sensitive and would want to learn about more techniques that could bypass that.

List of Figures

Bibliography

[1] Miguel Alcubierre. *Introduction to 3+1 Numerical Relativity*. Oxford, 2012.

[2] Dr Eric Ayars. *Computational Physics with Python*. 2013.

[3] Thomas W. Baumgarte and Stuart L. Shapiro. *Numerical Relativity: Solving Einstein's Equations on the Computer*. New York, NY, USA: Cambridge University Press, 2010. ISBN: 052151407X, 9780521514071.

[4] Carles Bona-Casas Carles Bona Carlos Palenzuela-Luque. *Elements of Numerical Relativity and Relativistic Hydrodynamics*. Springer, 2009.

[5] E. M. De Jager. "The Lorentz-Invariant Solutions of the Klein-Gordon Equation". In: *SIAM Journal on Applied Mathematics* 15.4 (1967), pp. 944–963. ISSN: 00361399. URL: http://www.jstor.org/stable/2099798.

[6] Jaan Kiusalaas. *Numerical Methods in Engineering with Python 3*. Cambridge University Press, 2013.

[7] Svein Linge and Hans Petter Langtangen. *Programming for Computations - Python. A Gentle Introduction to Numerical Simulations with Python*. Springer, 2010.

[8] William Press et al. *Numerical Recipes in C*. Cambridge University Press, 1992.

[9] Charles G. Torre. "13 Spherical Coordinates". In: *Foundations of Wave Phenomena, Book 10* (2014).