Time Evolution PDEs and their Numerical Solution

Masters in Physics Engineering
Advanced Topics in Computational Physics

João D. Álvares[1,2]

[2]*Instituto Superior Técnico, Av. Rovisco Pais 1, 1049-001 Lisboa*

Prof. David Hilditch & Prof. Alex Vañó-Viñualez

Abstract

The impossibility of having an analytic solution for most of the yet to study problems of physics makes the numerical methods implemented to solve them very important to develop. Here we develop a class in C++ in order to implement a partial differential equation solver, using 4th order Runge-Kutta in time evolution and several methods for the spatial part, including Finite Differences for several orders and the Pseudospectral method, with or without periodic boundary conditions. An implementation of L2 norm convergence is shown as well.

1 Theoretical Introduction

1.1 Wave Equation

We here deal with partial differential equations (PDEs), which are differential equations which involve different partial derivatives as functions of other partial derivatives. We are dealing, specifically, with time evolving PDEs, i.e., something of the type,

$$\partial_t \phi(t, x^a) = F(\phi, \partial_a \phi(t, x^a))$$

where $\partial_a = \partial/\partial x^a$ and F is any function and ϕ is our time-evolving multi-variable function. In this case, we will deal use one of the best known examples of PDEs, the 1-D wave equation, with all the constants set such that the velocity is equal to 1, i.e.,

$$\partial_{tt}\phi = \partial_{xx}\phi \tag{1.1.1}$$

Note that this implies a second derivative in time. Our goal was to have something, on the left-hand side (LHS) of the equation, that was only one derivative in time. Thus, we can set a new variable, π, such that

$$\partial_t \pi = \phi$$

and thus this will change (1.1.1) into a system of equations,

$$\partial_t \pi = \partial_{xx}\phi \tag{1.1.2}$$
$$\partial_t \phi = \pi \tag{1.1.3}$$

and it's going to be with this type of systems that we are going to work with. The code implemented is able to deal with similar systems.

1.2 4th Order Runge-Kutta for Time Evolution

The discretization in space will be talked further in the report and it is the focus of the project. The time part, although it depends on the spatial grid, can be treated a priori. During the project, for future reference, a second order method was used for the spatial part, i.e., the right-hand side of the equations (RHS). Now, Runge-Kutta (RK) methods are a family of several methods, with different orders of convergence for each, which are used for equations of the type,

$$\partial_t y = f(y, t)$$

However, one could ask that this is different from, for example, (1.1.1), given the fact that our RHS dependeds, not on y, but on the second spatial derivative of y, which is indeed a problem at first. However, once one discretizes space, the derivative turns into a linear combination of the values of y in several points, i.e.,

$$\partial_{xx} u_j = \sum c_{ij} u_i$$

where u_j means the value of $u(x_j)$, i.e., a given point in the spatial grid and c_{ij} are real coefficients. Thus, once we discretize space, (1.1.1) turns into something of the form

$$\partial_t \pi = F(\phi_i) \tag{1.2.1}$$
$$\partial_t \phi = Q(\pi_i) \tag{1.2.2}$$

Thus, they are in the form that can be solved by a Runge-Kutta method. We opted by using the 4th Order RK method which means, given a standard discretization in time by steps

[1] joaodinis01@tecnico.ulisboa.pt, 96541

of h, the error of our system is of order h^4, i.e., it converges very fast. The method is calculated as follows,

$$u_{n+1} = u_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$t_{n+1} = t_n + h$$

with the constants k_i being given by,

$$k_1 = f(t_n, u_n)$$
$$k_2 = f(t_n + \frac{h}{2}, u_n + h\frac{k_1}{2})$$
$$k_3 = f(t_n + \frac{h}{2}, u_n + h\frac{k_2}{2})$$
$$k_4 = f(t_n + h, u_n + hk_3)$$

For the case of the wave equation, we do not have time as a variable on the RHS, which simplifies the work. This process is done for ϕ and π, which makes sure both variables evolve in time.

1.3 Spatial Discretization

The purpose of the project was to implement several ways of calculating derivatives in space. We start with the simplest one, finite differences.

1.3.1 Finite Differences Methods

For the first derivative, let us notice the following. Let $f(x)$ be a C^∞ function. Then,

$$f(x+h) = f(x) + hf'(x) + h^2\frac{f''(x)}{2} + O(h^3)$$

i.e.,

$$\frac{f(x+h) - f(x)}{h} - f'(x) = O(h) \qquad (1.3.1)$$

i.e., $\frac{f(x+h)-f(x)}{h}$ is a first order approximation of the derivative. This one in specific is called the forward difference approximation. The backward difference approximation is given by changing $h = -\Delta x$, which turns (1.3.1) into

$$\frac{f(x+h) - f(x)}{h} = \frac{f(x) - f(x-\Delta x)}{\Delta x}$$

We can still make the centered difference approximation of $f'(x)$, which essentially joins both forward and backward approximations. Note the following,

$$f(x+\Delta x) = f(x) + \Delta x f'(x) + \Delta x^2\frac{f''(x)}{2} + O(\Delta x^3)$$
$$f(x-\Delta x) = f(x) - \Delta x f'(x) + \Delta x^2\frac{f''(x)}{2} + O(\Delta x^3)$$

Then,

$$f(x+\Delta x) - f(x-\Delta x) = 2\Delta x f'(x) + O(\Delta x^3)$$

i.e.,

$$\frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} - f'(x) = O(\Delta x^2)$$

Thus, the centered difference approximation is actually a better approximation of forward or backward approximations. We can generalize the centered approximation for the second derivative, which is what we need for in the wave equation. This gives,

$$f''(x) = \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{\Delta x^2} + O(\Delta x^2)$$

We can also get the second order approximation of the forward and backward difference method, respectively, in this case for the second derivative,

$$f''(x) = \frac{2f(x) - 5f(x+\Delta x) + 4f(x+2\Delta x) - f(x+3\Delta x)}{\Delta x^3}$$
$$f''(x) = \frac{2f(x) - 5f(x-\Delta x) + 4f(x-2\Delta x) - f(x-3\Delta x)}{\Delta x^3}$$

All of these methods are implementend in the code, accompanined by the fourth-order finite-difference methods as well, for centered approximations [3].

1.3.2 Pseudospectral Method

This part heavilly relied on [1]. We used Chebyshev polynomials as an orthogonal basis to decompose our space. These polynomials are recurrently defined by

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

They have the further benefit,

$$T_n(\cos\theta) = \cos(n\theta)$$

Thus, in this case, if we give the initial data evaluated at the points $\cos\theta_j$, where j is an angle index, everything will be greatly simplified. This also means that our initial data has to be given in the interval $[-1, 1]$, but this is just a matter of rescaling the equation accordingly. According to [1], the differentation matrix D_N, indexed from 0 to N where N is the amount of points from the initial data, is given by,

$$(D_N)_{00} = \frac{2N^2+1}{6}$$
$$(D_N)_{NN} = -\frac{2N^2+1}{6}$$
$$(D_N)_{jj} = \frac{-x_j}{2(1-x_j^2)}, j = 1, ..., N-1$$
$$(D_N)_{ij} = \frac{c_i}{c_j}\frac{(-1)^{i+j}}{(x_i - x_j)}, i \neq j = 1, ..., N-1$$

where $c_i = 2$ if $i = 0, N$ and $c_i = 1$, otherwise. The derivative of a function $f(x_j)$ evaluated at points $x_j = cos(j\pi/N)$, is given by,

$$f'(x_i) = (D_N)_{ij} f(x_j)$$

Thus, to calculate the second derivative, we just multiply two times by this matrix, i.e., (1.1.1) turns into,

$$\partial_t \pi = (D_N)^2\phi$$
$$\partial_t \phi = \pi$$

This method goes with $O(e^{-N})$.

1.3.3 FFT using Chebyshev

One of the best known ways of calculating derivatives is through the Fast-Fourier Transform (FFT), but in this case using the Chebyshev polynomials as a basis. Again, relying on [1], we get that this is easily done by the following scheme,

$$\begin{aligned}
\tilde{V}_k &= \frac{\pi}{N}\sum_{j=1}^{2N} e^{-ik\theta_j} V_j \,, k = -N+1, ..., N \\
\tilde{W}_k &= ik\tilde{V}_k \,, \text{except } \tilde{W}_N = 0 \\
W_j &= \frac{1}{2\pi}\sum_{k=-N+1}^{N} e^{ik\theta_j}\tilde{W}_k \,, j = 1, ..., 2N \\
w_j &= -\frac{W_j}{\sqrt{1-x_j^2}} \,, j = 1, ..., N-1
\end{aligned}$$

with,

$$\begin{aligned}
w_0 &= \frac{1}{2\pi}\sum_{n=0}^{N}{}' n^2 \tilde{V}_n \\
w_N &= \frac{1}{2\pi}\sum_{n=0}^{N}{}' (-1)^{n+1} n^2 \tilde{V}_n
\end{aligned}$$

where the vector V is constructed from the vector that contains the data evaluated at the Chebyshev points, v_j. We define $V_j = v_j$ for $j = 0, ...N$ and $V_{2N-j} = v_j$, $j = 1, ..., N-1$.

1.4 Boundary Values

Because we are dealing with the wave equation, we have to give sufficient initial data. Because it is a second order differential equation, this means that we have to give two sets of initial data. The code is adapted to get the following initial data,

$$\begin{aligned}
u(x, 0) &= f(x) \\
\partial_t u(x.0) &= h(x)
\end{aligned}$$

where f and h are real-valued functions. The code can, however, be easily changed to receive any combination of initial data given. We have to further give boundary values, which can be periodic or non-periodic. The code has specific methods for each, one for establishing specific boundary values and other one for periodic boundary conditions, for each of the methods spoken above.

For periodic boundary conditions, we basically add ghost points either to the ”right” or to the ”left” of the vector of initial data, in order for the derivatives to be calculated accordingly, i.e., as if the end of the function would continue to its beginning. For the case of non-periodic boundary conditions, the way the code is working is by basically ignoring the calculations for the points we are setting the boundary conditions and fixing them as they are. These boundary values are also being taken into account as,

$$\begin{aligned}
u(0, t) &= q(t) \\
u(x_N, t) &= g(t)
\end{aligned}$$

where again q and g are two-real valued functions. They are usually taken to be constant and equal to 0, but just by a matter of simplicity of testing. This completes the amount of data we have to give to the wave equation in order for it to evolve accordingly in time and space.

1.5 L2 Norm Convergence

The l^2 norm of a vector $\vec{x}$, $|\vec{x}|_2$ is defined as,

$$|\vec{x}|_2 = \sqrt{\sum_{k=1}^{n} |x_k|^2}$$

In order to check if our implementation is working well, we should expect our code to converge faster as our step decreases. And it should decrease proportionally to the order of convergence of the method we are using. Consider a method of convergence of order p [2] used with a grid with step h. Furthermore, let ϵ be a real number s.t. $|\epsilon| <= 1$. Thus, the exact solution u_{exact} should not differ from the solution solved with step h, u_h, more than,

$$u_{\text{exact}} = u_h + \epsilon h^p$$

If we multiply the step by a factor of f, then,

$$\begin{aligned}
u_{fh} - u_h &= u_{\text{exact}} - \epsilon(fh)^p - u_{\text{exact}} + \epsilon h^p \\
u_{fh} - u_h &= \epsilon h^p(f^p - 1)
\end{aligned}$$

We also have that,

$$u_{f^2h} - u_{fh} = \epsilon(fh)^p(f^p - 1)$$

Thus,

$$\frac{u_{f^2h} - u_{fh}}{u_{fh} - u_h} = f^p \qquad (1.5.1)$$

We can do this, however, for a vector $\vec{v}$ which includes both the solution for the wave-equation solution and its derivative, i.e, for ϕ and π. Thus, let $v_j = \phi_j$ and $v_{N+j} = \pi_j$. We can then use the L2 norm for the error difference between two resolutions of the wave equation with different steps,

$$\left(\sum_j ((v_{f^2h})_j - (v_{fh})_j)^2\right)^{1/2} \Bigg/ \left(\sum_j ((v_{fh})_j - (v_h)_j)^2\right)^{1/2}$$
$$= \frac{\sqrt{N}}{\sqrt{N}} f^p = f^p$$

If we want to get the value of the order of convergence, then we must do $\log_f$, which can be rewritten, by the log rules,

$$p = \log_f(f^p) = \frac{\log_{10}(f^p)}{\log_{10}(f)}$$

and this is what is done in the code for the l^2 norm convergence.

2 Implementation

The code was done in the programming language C++ and using a very specific folder naming distribution, according to the respective Makefile. In terms of the actual implementation of the code, it is just a matter of following the previous section, it is pretty straightforward. The whole code can be retrieved from My Git.

3 Results

3.1 Finite Differences

This is done in the file **fd.cpp**. Let the initial data be given by,

$$\phi(x,0) = exp(-x^2/0.1) \tag{3.1.1}$$

$$\partial_t \phi(x,0) = 0 \tag{3.1.2}$$

$$\phi(-1,t) = \phi(1,t) \tag{3.1.3}$$

with $x \in [-1,1]$. Using second-order finite differences method for the homogeneous wave equation (1.1.1), we get the graphics in Fig. 1 for its evolution.

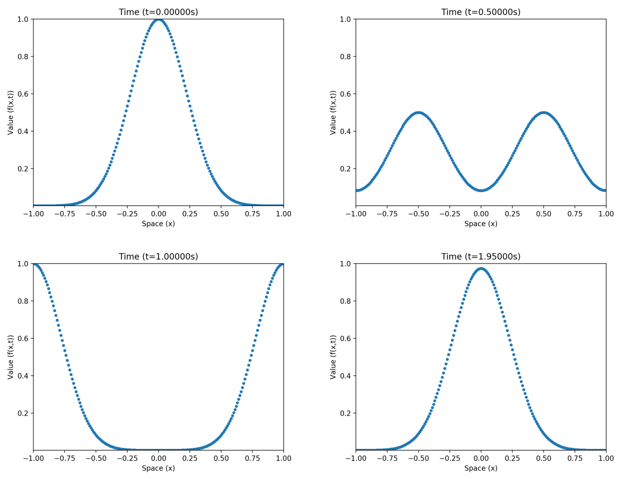

Figure 1: From top right to bottom left, it is the system in (3.1.3) using (1.1.1) evolution, through a second order finite-difference method in space and 4th order RK method in time, evaluated at times $t = 0s$, $t = 0.5s$, $t = 1s$ and $t = 1.95s$.

i.e., we get a peak that divides itself into left-going and right-going peaks, with half its height, they eventually hit the boundaries of our wall and then turn back to the same initial configuration, i.e., preserving the initial energy as was supposed given the fact that the wave equation maintains the energy of the system. Furthermore, we know that the solution for the homogeneous wave equation is a sum of two waves, one going in one direction and the other one going in the opposite direction. This is just showing the nature of the wave equation. In terms of norm convergence, we get Fig. 2. Actually, in the figure it is plotted the difference between the lowest resolution and the medium resolution, $u_{f^2h} - u_{fh}$, for $f = 2$, and the difference between the medium resolution and the high resolution, multiplied by the factor over which it is expected to diminish this difference when compared to the first one, i.e., $f^p(u_h - u_{fh})$, instead of dividing immediately, which could provide a division by a value close to 0.

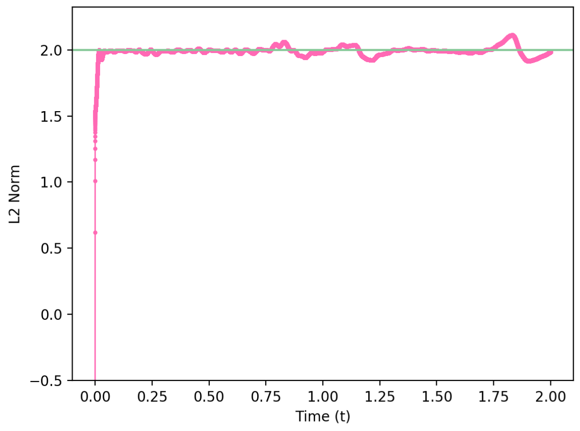

Figure 2: l^2 norm convergence of Fig.1.

Given the fact that the period of the simulation is around 1 second, Fig. 2 means when each period is reached, we have a bit of discrepancy, but it soon goes to 2 again, which is the order of convergence expected. Thus, it is converging as it should and it is an indication that the method is well implemented. In terms of point-wise convergence, i.e., for a specific time t, using (1.5.1), we get Fig. 3.

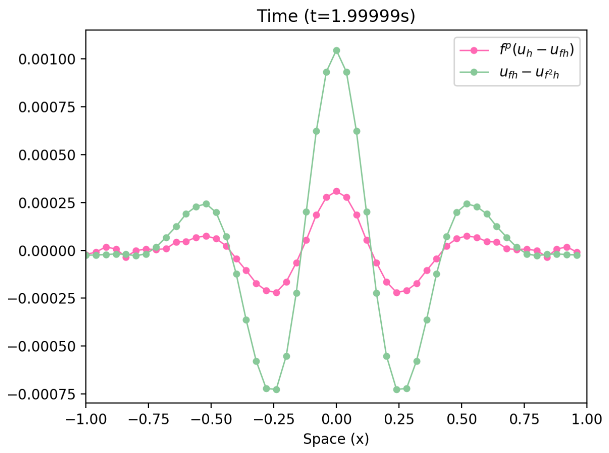

Figure 3: Point-wise convergence of Fig.1 at $t = 1.99999s$.

which behaves as we should be expecting, i.e., the highest errors are in the zones where the derivative is biggest, in the middle. However, it continues to be of 10^{-3} magnitude. If we evaluate this at another time, for example when the two peaks are closer to the boundaries, we get Fig. 4.

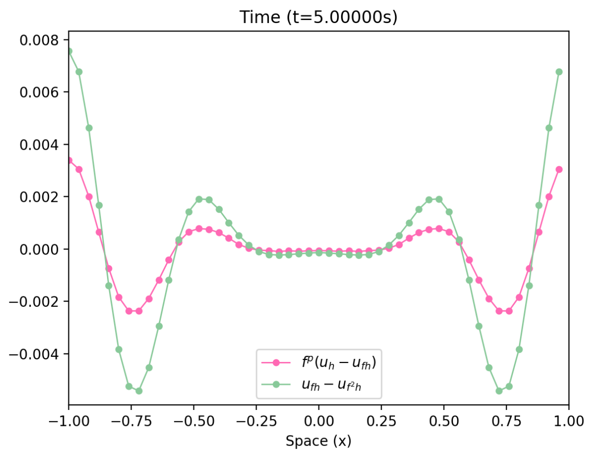

Figure 4: Point-wise convergence of Fig.1 at $t = 5.0s$.

There is yet another convergence test that was implemented in the code which shows the evolution of convergence as we make smaller and smaller spatial steps. This takes way longer in terms of computation, because one needs to readjust the time step as well given the Courant–Friedrichs–Lewy condition, that says that the time and spatial steps cannot be randomly chosen, otherwise the system blows up. This is shown in Fig. 5.

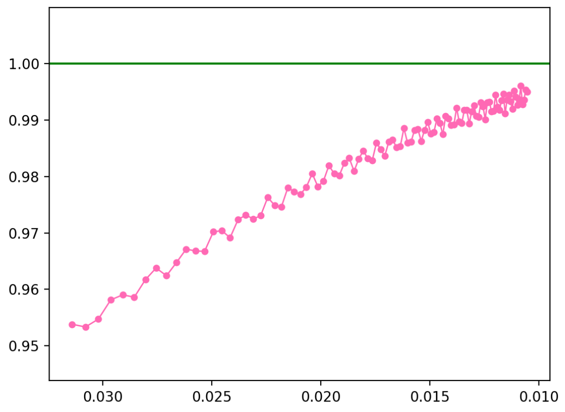

Figure 5: Norm Convergence as a function of step, at a given time t.

3.2 Pseudospectral Method

The pseudospectral method is way more precise and so, in the code, one can see that few initial points are needed to preserve accuracy to machine precision in the future iterations. The resulting images in Fig. 6 are the result of using a Newton interpolator method in order for the graphic to not just be a triangular shaped image. The few points needed for the pseudospectral method come up as well because the matrix multiplication calculations scale rapidly if the amount of points is increased as well. In Fig. 6, the boundary conditions are, at both ends, fixed at 0. Thus, it is almost as if the wave is behaving like a rope, once the both peaks reach each boundary, the wave seems to settle, but of course all the energy is gathered up on the derivative and right away it transforms into the reflection of the initial state. This goes on and on forever, as expected.

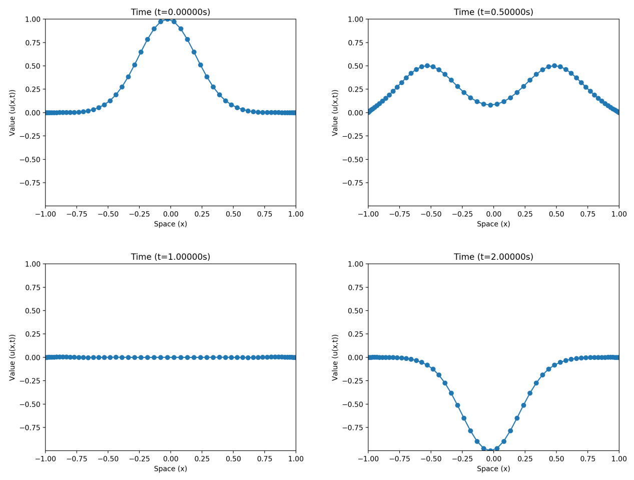

Figure 6: From top right to bottom left, it is the system in (3.1.3) using (1.1.1) evolution, through a pseudospectral method in a Gauss-Lobatto/Chebyshev grid space and 4th order RK method in time, evaluated at times $t = 0s$, $t = 0.5s$, $t = 1s$ and $t = 2s$. The boundaries are fixed, instead of periodic like in (1.1.1).

3.3 External Sources

The results shown this far are but specific cases of the homogeneous wave equation solved, with different boundary values. We now add an external source, such that,

$$\partial_{tt}\phi = \partial_{xx}\phi + f(x)$$

This is easily changed in the code as well, the source function. For the example of,

$$f(x) = 4\cos(x) \tag{3.3.1}$$

using the same initial conditions for the gaussian-function, we get the several pictures in Fig. 7.

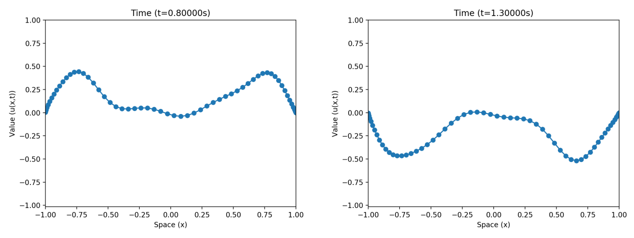

Figure 7: From right to left, it is the system in (3.1.3) using (1.1.1) evolution and external function (3.3.1), through a pseudospectral method in a Gauss-Lobatto/Chebyshev grid space and 4th order RK method in time, evaluated at times $t = 0.8s$ and $t = 1.3s$. The boundaries are fixed.

The physical intuiton of having an external function is, as far as I have searched, that this system can always be thought of as the inhomogeneous electromagnetic wave equation, which is usually given as

$$\partial_{tt}\phi - \partial_{xx}\phi = \rho \tag{3.3.2}$$

in natural units, for one dimension, where ϕ is the electric potential. Here, ρ stands for the charge density. It can be a function of space.

4 Work Almost Finished

Fast Fourier Transform using Chebyshev differentiation is basically done, apart from the boundaries, giving a little bit different values. According to [1], this is expected, but not as much as what my code is getting. The implementation with this method can be done by just typing **SetMethod(FFTPS)**, but will give some weird values.

5 Conclusion

This project allowed me to produce a good amount of code to solve the wave equation with possible external sources, but can also be used for every kind of combination of derivatives, although this will take a bit more time redoing the code for the Right-Hand Sides. This code is thus prepared to solve any kind of PDE, using RK4 in time and can use finite-differences and pseudospectral method, with a malfunctioning possible way of using pseudospectral FFT. It allows one to set as a precise step as wished. The fact that this course was given in a period does not allow time to understand mathematically how exactly the methods come about, but in terms of implementation the work is done apart from the Galerkin methods that were shown in the project description as being one of the possible goals as well.

References

[1] TREFETHEN, L. N. Spectral Methods in Matlab.

[2] VAÑÓ-VIÑUALES, A., AND HILDITCH, D. Numerical Methods in Gravitation.

[3] YEW, A. Numerical Differentiation: Finite Differences.