Numerical Techniques for the Wave Equation

Math 437 Capstone Project, Fall 2023

Author: Christian Lentz

The goal of this project will be to investigate numerical techniques for approximating solutions to PDEs. We will use finite differences to build a simple model for the wave equation with randomized initial conditions and periodic boundary conditions. We include a total of four implementations of finite differences. This will motivate discussions and comparisons of other numerical techniques and thier implementations, particularly the Fourier spectral method. We include one (partially complete) implementation of the Fourier spectral method using the Fast Fourier Transform through SciPy's FFT package.

1 The Wave Equation

The wave equation is a second order partial differential equation which describes both traveling and standing waves as they occur in classical physics. The equation is particularly useful in modeling mechanical waves, such as sound waves, or electromagnetic waves, such as light waves. Note that the wave equation is not the same as the wave function, which is used in quantum mechanics to describe the quantum state of an isolated quantum system.

Let x be some n-dimensional vector $x = (x_1, x_2, \dots x_n)$, and suppose that $u(x, t)$ describes the displacement of a system from an initial rest position. The wave equation will describe the acceleration of this displacement over time. Specifically, the scalar form of the wave equation is

$$\frac{\partial^2 u}{\partial t^2} = c^2 (\frac{\partial^2 u}{\partial x_1^2} + \cdots + \frac{\partial^2 u}{\partial x_n^2})$$

,

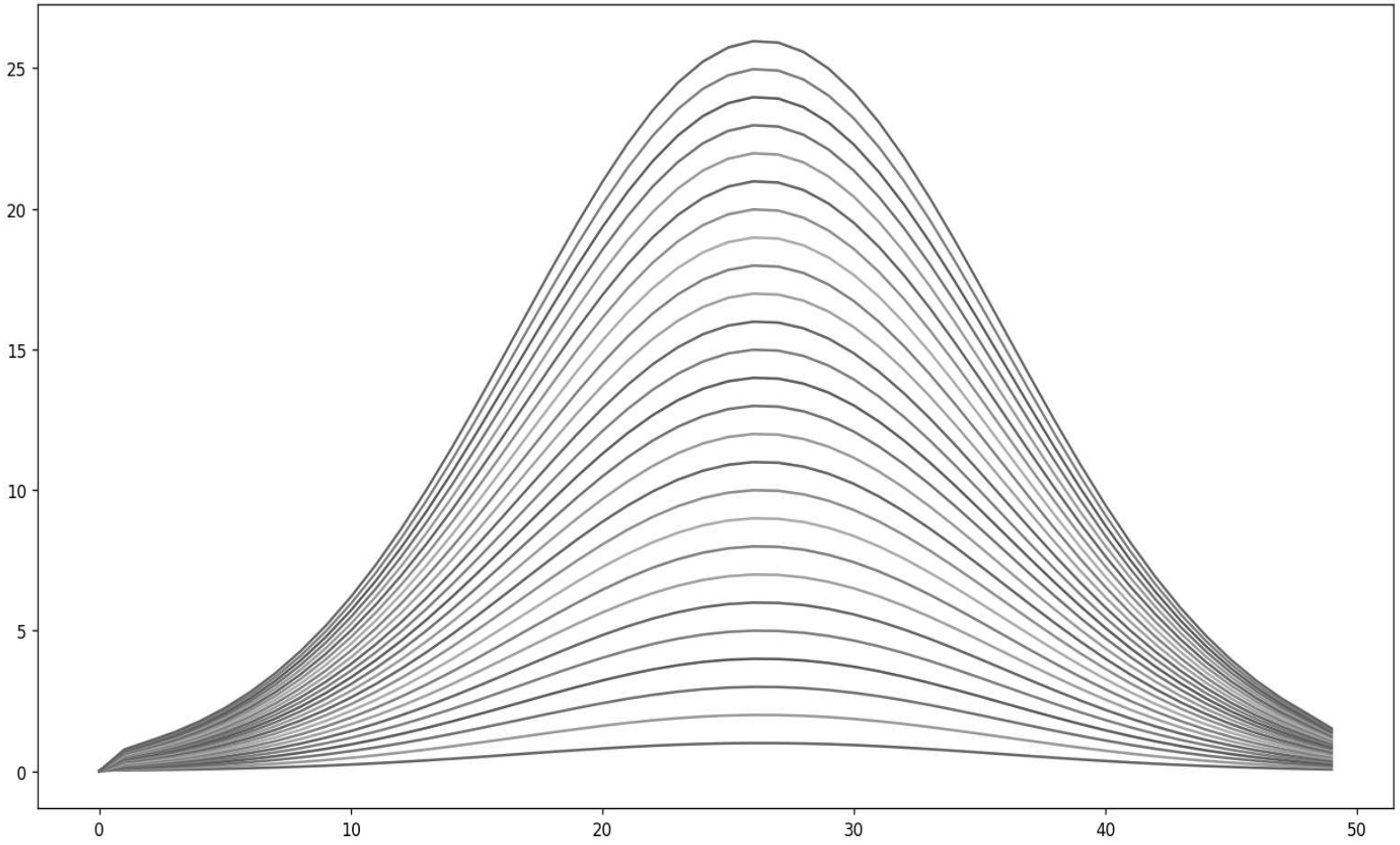
where $c \in \mathbb{R}^{>0}$ is some scalar. The vectorized form can be written as

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u$$

This form of the wave equation which we use assumes no friction. For this project, we will use finite difference and spectral methods to approximate solutions to the wave equation with periodic boundary conditons in order to investigate and compre the underlying numerical techniques.

2 Finite Differences

The finite differences method makes use of the definition of the derivative, or the difference quotient. However, rather than taking the limit, we will approximate solutions to the PDE at finitely many fixed points in time, hence why we call this finite differences.



2.1 A bit of Analysis

First, we recall the definition of the derivative. Suppose some function $f : E \to \mathbb{R}$ where $a \in E$ is a limit point of $E \subseteq \mathbb{R}$. We say that f is differentiable at a if the following limit exists:

$$lim_{h \to 0} = \frac{f(a+h) - f(a)}{h}$$

Equivalently, we may also write the difference quotient as:

$$lim_{h \to 0} = \frac{f(a) - f(a-h)}{h}$$

Writing the difference quotient in either of these forms will be useful in proving the difference quotient for the second derivative of a function. To make our definition above more concise, it is also useful to recall that if $a \in E$ is a limit point, then:

$$\forall \epsilon > 0 : \exists x \in E : 0 < |x - a| < \epsilon$$

In words, E contains points which are arbitrarily close but not equal to a. This definition is the foundation of the finite differences approach. By manipulating the difference quotient, we can build a technique to approximte solutions to any ODE or PDE of any order. It is possible to do this since we can generalize the definition of the difference quotient to higher order derivatives as well. For our purpose, we will need to know the difference quotient for the second derivative of a function.

Claim: Let $f : \mathbb{R} \to \mathbb{R}$ be differentiable and let $a \in \mathbb{R}$. Further, suppose that $f''(a)$ exists. Then:

$$f''(a) = lim_{h \to 0} \frac{f(a+h) + f(a-h) - 2f(a)}{h^2}$$

We can provide two proofs of this fact. The first is quite clear but is also quite an abuse of notation. The second makes use of Cauchy's generalized Mean Value Theorem.

Proof 1: Since $f''(a)$ exists, it follows that:

$$f''(a) = lim_{h \to 0} \frac{f'(a+h) - f'(a)}{h}$$

Further, since f is a differentiable function, then we have:

$$f''(a) = lim_{h \to 0} \frac{\frac{f(a+h_1)-f(a)}{h_1} - \frac{f(a)-f(a-h_2)}{h_2}}{h}$$

Letting $h = h_1 = h_2$, then we may simplify to get the form stated in the claim, and this concludes the proof. Although this is concise, we cannot interchange the limits, nor is it appropriate to assume that $h = h_1 = h_2$.

To provide our second proof, we first state the Cauchy MVT, without proof.

Cauchy MVT: Let $a < b$ and $f, g : [a, b] \to \mathbb{R}$. Assume that f, g are continuous on $[a, b]$ and differentiable on (a, b). Then, there exists $\xi \in (a, b)$ such that

$$f'(\xi)(g(b) - b(a)) = g'(\xi)(f(b) - f(a)).$$

Furthermore, under the assumtion that $g'(x) \neq 0$ for all $x \in (a, b)$, then we can write this result as

$$\frac{f'(\xi)}{g'(\xi)} = \frac{f(b) - f(a)}{g(b) - g(a)}.$$

Using this, we can provide a stronger proof of the claim above.

Proof 2: Consider the compact interval $[0, h]$ and define $F(x) = f(a + x) + f(a - x) - 2f(x)$ and $G(x) = x^2$ for $h > 0$. Consider $lim_{h \to 0^+}$. We may apply Cauchy, as $F(x), G(x)$ are continuous on $[0, h]$ and differentiable on $(0, h)$ with $G(0) \neq G(h)$. It follows that

$$\begin{aligned} \frac{F(h) - F(0)}{G(h) - G(0)} &= \frac{f(a+h) - f(a-h) - 2f(a)}{h^2} \\ &= \frac{f'(a + \xi_h) + f'(a - \xi_h)}{2\xi_h} \\ &= \frac{F'(\xi_h)}{G'(\xi_h)} \end{aligned}$$

where $\xi_h \in (0, h)$ with $lim_{h \to 0^+} \xi_h = 0$. Evaluating this limit, we see that

$$\begin{aligned}\frac{f'(a+\xi_h)+f'(a-\xi_h)}{2\xi_h} &= \frac{(f'(a+\xi_h)-f(a))+(f(a)-f'(a-\xi_h))}{2\xi_h} \\ &= \frac{f''_+(a)+f''_+(a)}{2} \\ &= f''_+(a)\end{aligned}$$

Note that we must consider $lim_{h\to 0^-}$ separately. Here we only consider the right derivative, and note that the case for the left derivative follows similarly. This concludes the proof. With this, we now have a stong understanding of the definition of the first and second derivative. Now we can use these facts to construct a method for approximating solutions to PDEs.

2.2 Finite Differences for the Wave Equation

Here, we outline our finite diffeences technique for the wave equation. Consider the wave equation in two spatial dimensions x and y:

$$\frac{\partial^2 u}{\partial t^2} = c^2(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2})$$

The first step of this process will be to write the PDE using the difference quotient. To make things a bit more intuitive, we will write the limits as $lim_{\Delta t\to 0}$, $lim_{\Delta x\to 0}$ and $lim_{\Delta y\to 0}$. Suppose that we let $x=a$, $y=b$ and $t=T$. Then we have:

$$\begin{aligned} lim_{\Delta t\to 0}\frac{u(a,b,T+\Delta t)+u(a,b,T-\Delta t)-2u(a,b,T)}{(\Delta t)^2} &= \dots \\ \dots = c^2(lim_{\Delta x\to 0}\frac{u(a+\Delta x,b,T)+u(a-\Delta x,b,T)-2u(a,b,T)}{(\Delta x)^2} &+ \dots \\ \dots + lim_{\Delta y\to 0}\frac{u(a,b+\Delta y,T)+u(a,b-\Delta y,T)-2u(a,b,T)}{(\Delta y)^2}&) \end{aligned}$$

By inspection, we see that this is quite easy to generalize to the n-dimensional case. Although this form is quite cumbersome, we can leverage it to numerically approximate solutions to the wave equation. The key is to ignore the limit, and let $u(a,b,T+\Delta t)$ be unknown. By doing this, we will be able to approximate solutions for fixed t beyond our initial time T. Thus, we get something that looks like this:

$$
\begin{aligned}
u(a,b,T+\Delta t) &\approx \dots \\
\dots &\approx 2u(a,b,T) - u(a,b,T-\Delta t) + \dots \\
\dots &+ \frac{c^2(\Delta t)^2}{(\Delta x)^2}(u(a+\Delta x,b,T) + u(a-\Delta x,b,T) - 2u(a,b,T)) + \dots \\
\dots &+ \frac{c^2(\Delta t)^2}{(\Delta y)^2}(u(a,b+\Delta y,T) + u(a,b-\Delta y,T) - 2u(a,b,T))
\end{aligned}
$$

Using this equation, we can approximate solutions for our PDE.

3 Choosing Δx, Δy, and Δt

We must take care when chooisng parameter values for Δx, Δy, and Δt to assure the stability of our program. In certain cases, a poor choice for these parameters will result in our program producing incorrect solutions.

Suppose our wave PDE with solutions of the form $u(x,y,t)$ where $x, y \in R$ and $t \in \mathbb{R}^{>0}$. Such a PDE can be thought of as an infinite dimensional function space, where at each $t \in \mathbb{R}^{>0}$ we have a unique function describing a solution to the PDE. Numerical techniques for approximating solutions to such a PDE, including finite differences, will rely on a **discretization** of the function space to an finite sequence of probelms.

Suppose that we wish to model our PDE over the time interval T = $[t_0, t_{max}]$. A discretization of the time variable will split this iterval into finitely many points by choosing some Δt. The cardinality of this disretized set is:

$$N = \frac{t_{max} - t_0}{\Delta t}$$

And the set itself is

$$dT = \{t_0 + \Delta t * i\} \text{ for } i \in \{1, \dots, N\}$$

By doing this, we have transformed the uncountably infinite interval T into a finite sequence of N probelms, where we approximate a solution to the PDE for each $t \in dT$. We can easily generalize this to the case where we wish to model the PDE for all $t \in \mathbb{R}^{>0}$ by simply letting $t_0 = 0$ and $t_{max} = \infty$. In this case, dT is a countably infinite set. However, we must also place a discretization on the spatial variables as well. Thus, we must also choose some Δx and Δy in order to transform the space into a discretized grid.

At a high level, we must choose these parameters such that Δt is not too large as too allow the wave to travel to another coordinate in the grid within that interval of time. We can ensure proper selection of these parameters by adhering to the

CFL condition, which is commonly used with finite differences to model advection like processes. The condition states that for a PDE with spatial variables $x = (x_1, \ldots, x_n)$ and temporal variable t, a discretization of these parameters must satisfy that

$$\Delta t(\sum_{i=1}^{n} \frac{u_{x_i}}{\Delta x_i}) \leq C$$

where C is a dimensionless constant and u_{x_i} is the velocity with respect to x_i. We typically let $C = 1$. Essentially, the CFL conditions tells us that if we choose a small value for Δx and Δy, we must also choose a sufficiently small value for Δt as well.

4 Finite Differences Implementation

In this project, we provide a total of four implementations of finite difference for the wave equation, including two implementations for both one and two spatial dimensions. For the one dimensional case, we include a slow implementation which loops over all spatial coordinates at each time step, and a vectorized implementation which makes use of NumPy arrays, thus allowing us to loop over only the temporal variable.

For the two dimensional case, we include a vectorized implementation as well as an implementation which splits the probelms into two one dimensional probelms and "stitches" one dimensional solutions together at each time step. This implementation uses a vectorized approach to solve each one dimensional problem.

We do not include pseudocode or a full discussion of these implementations here. Instead, we direct the reader interested in the finer details of these implementations to the file **FiniteDiff.py** in the github repository for this project.

5 Other Numerical Methods

It is worth noting other common classes of numerical techniques, including Finite Element Methods (FEMs) and Finite Volume Methods. FEMs and Finite Elements both provide continuous or discontinuous piecewise approximations over a tesselation of the domain. While they are more similar to Finite Differences than Spectral Methods, they are considerably more difficult mathematically when compared to finite differences, and are also more difficult to implement. However, $\mathbf{Spectral\ Methods}$ can deliver highly accurate, $\mathbf{smooth}$ approximations for a solution to a PDE.

Spectral methods approximate solutions to PDEs in the form of fixed degree polynomials, which are $\mathbf{smooth}$ by definiton. Recall that for some open set U, any function $f : U \to \mathbb{R}$ is said to belong to the $\mathbf{differentiability\ class}$ C^n iff each higher-order derivative of f in the set $\{f^{(i)}\}_{i=1}^{n}$ exists and is continuous over U. A smooth function belongs to the class C^∞, or is

infinitely differentiable. If the underlying function that we are applying the PDE to is a smooth function, then a spectral method can provide a highly accurate, smooth approximation.

More specifically, Spectral Methods will seek an approximation in the form of a linear combination of a set of mutually **orthogonal polynomials**. Suppose the following linear combination of polynomials over the reals:

$$\sum_{i=1}^{n} f_i(x)$$

For any two polynomials $f_j(x)$ and $f_k(x)$ for $j, k \in \{1, \ldots, n\}$, we may define the inner product

$$< f_j, f_k > = \int f_j(x) f_k(x) \, d\alpha(x)$$

where α is a non-decreasing function over the reals. If $< f_j, f_k >$ = 0, then the polynomials are said to be orthogonal. Thus, a spectral method will seek to construct a space of mutually orthogonal functions which we use to approximate the PDE. We will not provide a full discussion of spectral methods in the general sense, but instead will discuss within the context of the Fourier method for the wave equation in one spatial dimension.

6 The Fourier Transform

Fourier analysis is an extension of the study of Fourier series, in which we wish to represent some function as a linear combinations of trigonomentric functions. The key insight is that we can express any general function in this way. Fourier analysis has applications across pure and applied mathemaitcs, including numerical techniques for PDEs. First, we examine the wave equation using these techniques.

Consider the wave equation in one spatial dimension:

$$u_{tt} = c^2 u_{xx} \tag{1}$$

Suppose that we wish to to solve with separation of variables, and that we use the following ansatz:

$$u_1(x,t) = A(t) cos kx$$

By plugging in and solving in the usual way (which we do not explicitly show here), we can show that u_1 is in fact a solution, but only under the assumption that $A(t)$ is a **harmonic oscillator**. Equivalently, using Newton's Second Law, we can say that A

satisfies that

$$A_{tt} = -k^2 v^2 A(t)$$

Following the same logic, we can also say that

$$u_2(x,t) = B(t) sin kx$$

is a general solution, under the assumption that $B(t)$ is also a harmonic oscillator. In fact, it is also quite easy to show that if u_1 and u_2 are solutions to (1), then so too is $u_1 + u_2$. Therefore, we may actually write our solution to the wave equation using arbitrary linear combinations of u_1 and u_2 with varying frequencies k.

$$u(x,t) = A_1(t) cos k_1 x + B_1(t) sin k_1 x + A_2(t) cos k_2 x + B_2(t) sin k_2 x + \dots$$

Such a solution works in a discrete setting, where the underlying function that we are approximating solutions for is periodic, and we have a sequence of equally spaced values k_n. To bring this into a more general setting, we must work with a continuum of values k, such that $\Delta k \to 0$ as $k \to \infty$. Thus, rather than using a sum, we must integrate:

$$u(x,t) = \int_{-\infty}^{\infty} A(k,t) cos kx + B(k,t) sin kx \, dk$$

Recall that since $A(t)$ and $B(t)$ are harmonic oscillators, and thus rely on k, we must now make explicit that both A and B are actually functions of k, since k now ranges over a continuum of values. Thanks to Euler, we may write this in the more concise form

$$\mathcal{F}^{-1}\{u(k,t)\} = \int_{-\infty}^{\infty} u(k,t) e^{ikx} \, dk$$

where, resepctively, A and B correspond to the real and imaginary part of u. What we have just derived is the $\mathbf{Inverse\ Fourier\ Transform}$, which describes how we may write $u(x,t)$ as a sum over a continuum of its frequencies. The $\mathbf{Fourier\ Transform}$ is the integral transform which actually sends $u(x,t)$ from its spatial (or temporal) domain into a representation over a domain of its freqeuncies:

$$\mathcal{F}\{u(x,t)\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} u(x,t) e^{-ikx} \, dx$$

It is common to include the term $1/\sqrt{2\pi}$ to normalize the integral, making the Fourier transform **unitary**, and therefore **symmetric**. The Fourier Transform is useful not only for solving PDEs by hand, but for spectral methods as well.

7 Fourier Spectral Methods

Recall that we previously noted that a spectral method for approximating solutions to PDEs will seek solutions in the form of linear combinations of mutually orthogonal polynomials. In fact, it is quite easy to show that sine and cosine functions of arbitrary frequency are indeed orthogonal over correctly chosen bounds. Let $N \in \mathbb{R}^{>0}$ and $m, n \in \mathbb{R}$. Using the inner product for polynomials, it follows that

$$\int_{-N}^{N} cos(mx)sin(nx)\ dx = 0.$$

Although this integral does not converge in the limit as $N \to \infty$, we may still use this to construct a sepctral method since numerical techniqes must place a discretization on the function space. Therefore, we cannot use the Fourier transform in the continuous form that we have written above. What this means is that we must use the **Discrete Fourier Transform**, which is a finite sum ranging over a discretization of the spatial variable x.

7.1 The Discrete Fourier Transform (DFT)

Suppose a sequence of points $x_0, x_1, \dots x_{n-1}$ where any point $x_j \in \mathbb{R}$. Equivalently, we may write this as a vector $X \in \mathbb{R}^n$. The DFT will map such a vector in a spatial (or temporal) dimension to a vector $Y \in \mathbb{C}^n$ given by $(y_0, y_1, \dots y_{n-1})^T$ using the following definition:

$$y_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j e^{-i2\pi jk/n}$$

We can augment this process to produce the vector Y by constructing the **Fourier Matrix**, $\mathcal{F}$. Each row of the matrix corresponds to a different value of k, and thus a mapping $x_j \mapsto y_k$:

$$Y = \frac{1}{\sqrt{n}} \mathcal{F} X$$

Letting $\omega = e^{-i2\pi/n}$, then $\mathcal{F}$ is defined as the unitary matrix

$$\begin{pmatrix} \omega^0 & \omega^0 & \omega^0 & \dots & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \dots & \omega^{n-1} \\ \omega^0 & \omega^2 & \omega^4 & \dots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \omega^0 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)^2} \end{pmatrix}$$

.

Applying the inverse Fourier transform is as simple as applying the matrix $\mathcal{F}^{-1}$ to some vector. Since $\mathcal{F}$ is unitary, then $\mathcal{F} = \overline{\mathcal{F}}^T$. This property is rather useful to exemplify why it is common to normalize both the continuous and discrete Fourier transforms. This matrix-vector product is the foundation of a class of algorithms dedicated to computing a discrete Fourier transform, most notably Fast Fourier Transfrom (FFT) algorithms. For our purpose, the FFT will be useful in implementing a Fourier spectral method.

7.2 Fast Fourier Trasnform

There is a rich history of Fast Fourier Transform algorithms. The most commonly used variation is that of Cooley and Tukey, first described in their 1965 paper. This is a recursive, divide and conquer algorithm, and thus works best for input sizes which are powers of two. However, other variations exist, such as the Good's prime factor FFT, which is efficient for input data of relatively prime size, and makes use of the Chinese Remainder Theorem. In fact, Good's work was a first step to inspiring Cooley and Tukey's algorithm. Before either of these implementations, the algorithm was first described, albeit in less detial, by Gauss. Today, most imlementations use the Cooley and Tukey algorithm, or some variation.

Since the DFT is a matrix-vector product, a straightforward algorithm to compute would typically be quadratic in the size of the vector which the transform is applied to. However, the Cooley and Tukey algorithm provides an alternative which runs in loglinear time (at worst) by taking advantage of the recursive structure of the DFT. Here, we motivate how and why the FFT works, without explicitly proving it's time complexity.

The key insight is in fact a simple consequence of the properties of exponential functions as well as properties of complex conjugates, which is stated with the following proposition.

Proposition: Suppose $Y = \{y_k\}$ is the Fourier Transform of $X = \{x_j\}$ where $X \in \mathbb{R}^n$. It follows that $y_0 \in \mathbb{R}$ and $y_{n-k} = \overline{y_k}$ for each $k \in \{1, \dots, n-1\}$.

Proof: Part a follows immediately from inspection of the definition of the DFT, and is a simple conseqeunce of the fact that $e^0 = 1$. Therefore, for $k = 0$, we have:

$$y_0 = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j$$

Part b is slightly less trivial. First, consider y_{n-k}, where once agian we take $\omega = e^{-i2\pi/n}$

$$y_{n-k} = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j (\omega^{n-k})^j$$

Using Euler's formula, closer inspection of ω^{n-k} implies that

$$\begin{aligned}
\omega^{n-k} &= e^{-i2\pi(n-k)/n} \\
&= e^{(-i2\pi n)/n} e^{(i2\pi k)/n} \\
&= e^{-i2\pi} e^{(i2\pi k)/n} \\
&= e^{(i2\pi k)/n} \\
&= cos(\frac{2\pi k}{n}) + isin(\frac{2\pi k}{n})
\end{aligned}$$

Furthermore,

$$\begin{aligned}
\omega^k &= e^{-i2\pi k/n} \\
&= cos(\frac{2\pi k}{n}) - isin(\frac{2\pi k}{n}).
\end{aligned}$$

Therefore $\omega^{n-k} = \overline{\omega^k}$. Returning to y_{n-k}, we have

$$
\begin{aligned}
y_{n-k} &= \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j (\omega^{n-k})^j \\
&= \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \overline{(\omega^k)}^j \\
&= \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} \overline{x_j (\omega^k)^j} \\
&= \overline{y_k}
\end{aligned}
$$

This concludes the proof. Above, we use two key facts to arrive at the final result. First, since $X \in \mathbb{R}^n$, then $\overline{x_j} = x_j$. Second, the product of complex conjugates is equal to the conjugate of their product. The implications of this proposition are rather important. What this means is that y_k is in the form

$$
(y_0, \ldots, y_{\frac{n}{2}-1}, y_{\frac{n}{2}}, \overline{y_{\frac{n}{2}-1}}, \ldots, \overline{y_1})^T,
$$

and therefore, the DFT of a real vector will result in a complex vector where half of the information contained is just the complex conjugate of another entry in the vector. Thus, we can break our probelm down recursively, and each computation of a DFT of size n is equivalent to two DFTs of size $n/2$. The FFT will recursively break the problem into DFTs of smaller size until hitting the base case of size 2. From there, we reconstruct larger solutions using multiplication and addition operations, and finally a division by $\sqrt{n}$. We do not explicitly include pseudocode for the FFT in this note.

8 Implementation

To start, let's make the assumption that for the one-dimensional wave equation $u_{tt} = c^2 u_{xx}$, we may write $u(x,t)$ as a product of functions with respect to x and t. More specifically, we have

$$
u(x,t) = \phi(x)\psi(t).
$$

With this assumption, it follows that:

$$
\phi\psi_{tt} = c^2 \phi_{xx}\psi \Rightarrow \frac{\psi_{tt}(t)}{\psi(t)} = c^2 \frac{\phi_{xx}(x)}{\phi(x)} \tag{2}
$$

However, this implies that:

$$\frac{d}{dt}(\frac{\psi_{tt}(t)}{\psi(t)}) = \frac{\partial}{\partial t}(c^2\frac{\phi_{xx}(x)}{\phi(x)}) = 0$$

Therefore, by a simple application of the constancy theorem, it follows that the RHS and LHS of (2) are constant. Here, we write this as follows:

$$-\lambda^2 = \frac{\psi_{tt}(t)}{\psi(t)} = c^2\frac{\phi_{xx}(x)}{\phi(x)}$$

The assumption that this constant is of the form $-\lambda^2$ is rather important. With this, it is quite simple to show that the following are general solutions to ϕ and ψ:

$$\psi(t) = \alpha_1 e^{\lambda t} + \alpha_2 e^{-\lambda t}, \qquad \phi(x) = \gamma_1 e^{\frac{\lambda t}{c}} + \gamma_2 e^{\frac{-\lambda t}{c}}$$

Using these general solutions, and letting $\lambda = ij$ and $\frac{\lambda}{c} = ik$, we now have

$$u(x,t) = (\sum \alpha_j e^{ijx})(\sum \gamma_k e^{ikt}),$$

where $\{\alpha_j\}$ and $\{\gamma_k\}$ are complex vectors computed with the DFT. As an aside, note that with this assumption, we now have

$$\lambda = ij \text{ and } \frac{\lambda}{c} \Rightarrow c = \frac{i}{k},$$

where i is associated with some Δx and k is associated with some Δt. As c corresponds to the speed coefficient of the wave equation, this assumption makes sense. As a simplifying assumption, we can alternatively assume that both $\{\alpha_j\}, \{\gamma_k\} \in \mathbb{C}^n$, thus allowing us to write $u(x,t)$ in the more concise form

$$u(x,t) = \sum_{i=0}^{n-1} \alpha_j e^{ijx} \gamma_j e^{ijt}.$$

At an implementation level, finding solutions to the wave equation now decomposes into solving the following set of IVPs:

$$u(x,0) = \sum \sigma_j e^{ijx}, u_t(x,0) = 0$$
$$u(x,0) = 0, u_t(x,0) = \sum \tau_j e^{ijx},$$

where the first set corresponds to a standing wave at $t = 0$, and the second corresponds to the velocity of that wave at $t = 0$. This project includes an incomplete implementation of the Fourier spectral method, which uses SciPy's FFT package and a hardcoded version of the inverse DFT to interoplate randomly generated data which is evenly spaced and periodic.

9 Conclusion

We have described in detail how to construct both a finite differences and Fourier spectral approach to approixmate solutions to PDEs. Specifically, we use the wave equation to motivate discussion and implementation of these techniques. Throughout, we have appealed to well known results of Analysis and Fourier Analysis and used tools of linear algebra to provide a solid mathematical foundation to this work. Finally, we have briefly noted implementation level techniques to increase efficincy for finite differences as well as the advantge of spectral methods in providing smooth, polynomial approximations to PDEs. Natural extensions of this work would be a complete implementation of the Fourier Spectral method, as well as discussion and implementations of other techniques, such as FEMs and Finite Volumes.

References

[1] Lentz, C. (2023). "Christian Lentz: Math 437 Capstone." GitHub, github.com/ChristianLentz/Math437Capstone/tree/main.

[2] Cooley, J. W., & Tukey, J. W. (1965). *An algorithm for the machine calculation of complex Fourier series*. Mathematics of computation, 19(90), 297-301.

[3] Good I. J. (1958). *The Interaction Algorithm and Practical Fourier Analysis*, Journal of the Royal Statistical Society: Series B (Methodological), Volume 20, Issue 2, Pages 361–372.

[4] "Scipy.Fft.Fft#." Scipy.Fft.Fft - SciPy v1.11.4 Manual, docs.scipy.org/doc/scipy/reference/generated/scipy.fft.fft.html.

[5] Süli, E. *A Brief Introduction to the Numerical Analysis of PDEs*.

[6] S H, L. (2012). *Numerical analysis of partial differential equations*. John Wiley & Sons.

[7] Wörner, S. (2008). *Fast fourier transform*. Swis Federal Institute of Technology Zurich, 10.