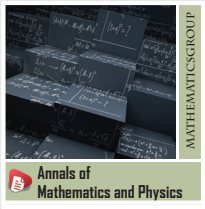
Peertechz



Annals of Mathematics and Physics OPEN ACCESS JOURNAL

ISSN: 2689-7636 Journal DOI: https://dx.doi.org/10.17352/amp

Research Article

Innovation in Mathematical Physics

Ted G Lewis*

Department of Computer Science, Naval Postgraduate School, USA

Submitted : 19 May, 2026
Accepted : 03 June, 2026
Published : 04 June, 2026

***Corresponding author:** Ted G Lewis, Department of Computer Science, Naval Postgraduate School, USA, E-mail: tedglewis@icloud.com

Keywords: Fourier Neural Operator (FNO); Mathematical physics; Machine learning; Partial Differential Equations (PDEs); Fast Fourier Transform (FFT); Operator learning; Physics-Informed Neural Networks (PINNs); Neural Operators; Scientific machine learning; Computational physics

https://www.mathematicsgroup.us

Check for updates

Abstract

Fourier Neural Operators (FNOs) have emerged as a significant breakthrough in solving mathematical physics equations, such as the heat, Navier-Stokes, and Kermack-McKendrick equations. By learning to map data in the frequency domain rather than computing over traditional mesh grids, FNOs circumvent the intensive recalculations typically required across varying grid sizes and parameter regimes. *However, FNOs often struggle with highly non-linear boundary conditions. In this work, we propose a modified FNO architecture that resolves this bottleneck.*] We demonstrate the approach maintains accuracy across modestly different parameters while accelerating computation times. This framework is particularly advantageous for iterative optimization tasks, such as aerodynamic shape design, where repeated simulations are required to converge on an optimal configuration. *Results show a100x speedup over classical solvers without loss of fidelity.*

Introduction

The following is a review of innovation in the field of mathematical physics that employs machine learning techniques to obtain fast numerical solutions of complex linear and non-linear differential equations. Traditionally, differentials are replaced by finite difference approximations, and the solution is iteratively computed for points in a grid. For example, the heat equation in 2D computes temperature at each point in a 2D grid of points by iterating over the grid for each time step – Sidebar A.

This produces an approximation based on initial conditions and parameters specified ahead of time. However, as the grid increases in size, say from 64 x 64 to 1,024 x 1,024, the number of computations increases exponentially. Machine learning overcomes this limitation, allowing for fast solutions once the solver is trained on examples. Additionally, if initial conditions and parameters such as diffusion rate or infection rate change, or the shape of the object is slightly altered, recomputing the entire solution may not be necessary due to machine learning as described below. Thus, the advantage of this approach is greater speed and modest flexibility versus traditional finite element analysis.

The idea of using machine learning to numerically solve differential equations began with foundations set by Li et al [1-3]. Li et al. train a convolutional neural network on hundreds of examples by first transforming the differential equation into a non-differential equation using the spatial Fourier transform over 1D, 2D, or 3D space. Note that this approach differs from the temporal Fourier transformation, whereby time is replaced with modes. The technique described here applies to spatial coordinates instead. This means the technique produces approximate solutions to many non-linear as well as linear equations.

At the core of the innovation is an operator FNO containing filter R that captures the physics expressed by the original differential equation. The operator is applied to the space-

Citation: Lewis TG. Innovation in Mathematical Physics. Ann Math Phys. 2026;9(3):123-128. Available from: https://dx.doi.org/10.17352/amp.000189

transformed version of the equation, and then the inverse Fourier transform is applied to the result to obtain the approximate solution on a grid. Interestingly, grid resolution (size) is somewhat invariant, so that a solution for a 64 x 64 grid, for example, can be used to obtain a solution to an arbitrarily large grid without re-computing the filter R. Additionally, the approximations are time-invariant, hence they are readily available at any time t. R results in much faster solutions once the filter R is discovered. Re-training is typically not required.

Anran Jiao and Lu Lu – along with researchers from NVIDIA and Pacific Northwest National Laboratory – were among the first to formally demonstrate that operator learning, such as FNO, can solve a broad range of biological and population-based differential equations. As the COVID-19 pandemic progressed, the focus shifted toward using FNOs to handle the "Age of Infection" (the core of the Kermack-McKendrick model of epidemic spreading). Research groups at MIT and Stanford (specifically those associated with the JuliaSim and SciML ecosystems) integrated FNO-like architectures to accelerate large-scale epidemiological simulations that were previously too slow using standard differential equation solvers [4-11].

At the center of this innovation is the FNO, which captures the inherent physics of a phenomenon modeled by its differential equation [2,3]. This significant advance combines classical Fourier transform theory with a convolutional neural network representation of a general multi-dimensional function to form an operator that, when applied in the forward direction and then reversed by operator inversion, solves equations faster than traditional finite element methods. For a brief review of Fourier transforms, see Sidebar B.

For centuries, the holy grail has been to discover general-purpose techniques that apply to differential equations regardless of their complexity. The combination of operator theory with machine learning is a new and powerful method with great promise because of its versatility and speed. This is illustrated by the heat equation and superficially by the non-linear Navier-Stokes equation. Because the FNO transforms spatial coordinates rather than temporal coordinates, the approximate solution to most non-linear equations can be found.

Overview: Finding R

Given an arbitrary differential equation $D_t\left(H\left(g,t\right)\right)=0$, we seek a solution for all and t. Here, g is a spatial coordinate (1D, 2D, or 3D) and t is time. D_t is an arbitrary differential operator or mapping from one space onto another, e. g., from differentials in terms of, $\frac{d}{dx},\frac{d^2}{dx^2}$, ... and, $\frac{\partial}{\partial x},\frac{\partial^2}{\partial x^2},\ldots$ to multiplications of terms or functions in another space. An operator such as D_t operates on a physical model expressed as an equation to simplify or reveal a solution. For example, the heat equation and the Navier-Stokes equations are expressed in terms of partial differentials – we apply one or more operators to map differentials onto terms that are multiplied insteadof differentiated (simplification) or extract a solution.

We seek a solution H for an arbitrary differential equation involving derivatives of H with respect to time, t. H may be in one, two, or three spatial dimensions and parameterized by constants such as infection rate, viscosity, diffusion rate, etc. We construct the FNO (Fourier Neural Operator) using the fast Fourier transform FFT to convert spatial dependency to the frequency domain, and the inverse Fourier transform FFT^{-1}, to convert the frequency domain back into the time-space domain. Note that the transform is applied to spatial coordinates rather than the traditional time coordinate, thus capturing the global impact of the area or volume over which the solution is approximated.

Given an equation for H, we seek a tensor[1] R such that solutions can be obtainedindependently of grid size:

$$H\left(x,y,t\right)=FFT^{-1}\left\{R\,FFT\left\{H\right\}\right\}$$

[1]For our purpose here, a tensor is a matrix.

Essentially, the right-hand side is an operator much like differentiation and FFT, itself. The purpose of machine learning is to discover tensor R from previous solutions kept in a database of training data. The tensor is a matrix containing complex numbers and multiplicatio R FFT{H} is carried out element by element instead of row by row. Think of R as a matrix of complex numbers that shapes the FFT such that the solution *H(x,y,t)* conforms to the training data. R is learned via a convolutional neural network and classical backpropagation.

The description below develops the idea underlying the FNO in gradual steps. FNO is an operator that operates on a model of the physics described by a differential equation. The inverse of { *{R FFT{H}}* yields parameterized solutions independent of grid resolution commonly used in finite element solutions. Once R is known, very fast solutions at time t can be obtained for any grid resolution without repeating the training step. Hence, the purpose of machine learning is to enable fast solvers and grid-size independence.

The solve problem

One of the great awakenings of modern science was the realization that the physical world can be *modeled* by mathematical equations – differential equations in particular. Consequently, a great deal of effort over the past several hundred years has been invested in finding methods of solving complex differential equations. Before computers were used to solve equations numerically, several clever *operator* methods were developed. For example, to solve a linear differential equation such as the following, we apply a differential operator D_t and its inverse D_t^{-1} to obtain a solution. Consider the model of heat diffusion in one dimension with constant diffusion rate, c:

Model: $\frac{dH}{dt}=-cH\left(t\right)$ given H(0)

Operator $D_t=\frac{d}{dt}+c$: therefore $D_t\{H\left(t\right)\}=H(0)$

Citation: Lewis TG. Innovation in Mathematical Physics. Ann Math Phys. 2026;9(3):123-128. Available from: https://dx.doi.org/10.17352/amp.000189

Apply inverse operator: $H(t) = D_t^{-1}\{H(0)\}$

Invert the transformation (via table lookup): $H(t) = H(0)e^{-ct}$

When equations become more complex, we apply a more powerful operator to simplify the mathematics and then invert the operator to obtain numerical results. This is typically done by first transforming the differential equation into an algebraic equation to simplify the mathematics. Transformers such as the Laplace and Fourier transforms were big steps forward for mathematical physics. For example, applying the Laplace transformer operator L to the equation for H(t converts the differentials into much simpler multiplications:

Model: $\frac{dH}{dt} = -cH(t)$ given H(0)

The Laplace transform: $H(s) = L\{H(t)\}$

Transform differentiation into multiplication:

$L\{\frac{dH}{dt}\} \underline{\underline{def}}\ sH(s) - H(0)$

Substitute, combine, and simplify: $sH(s) + cH(s) = H(0)$

Solve for H(s): $H(s) = \frac{H(0)}{s+c}$

Invert the transformation (via table lookup): $H(t) = L^{-1}\left\{\frac{H(0)}{s+c}\right\} = H(0)e^{-ct}$

We use transformers to simplify and operators to reveal underlying physics. In this case, the physical meaning is that H(t) exponentially shrinks (or grows) at a rate determined by c. Given an initial value H(0) and c, one can find H(t) for any t ≥ 0. A hot cup of coffee cools down exponentially according to the operator.

More importantly, we obtain an operator that scales or adjusts to varying parameters such as diffusion rate c and grid size in numerical solutions. Given an appropriate operator, one can apply the operator to the model (equation) and obtain solutions under varying conditions and parameters.

Things become more complicated in higher dimensions. For example, the heat equation in 2D defines the temperature at every point (x, y) in an area with an initial value at t = 0:

Model: $\frac{\partial H(x,y,t)}{\partial t} = c(\frac{\partial^2 H(x,y,t)}{\partial x^2} + \frac{\partial^2 H(x,y,t)}{\partial y^2})$;

H(x, y, 0) initially.

Traditionally, we would solve this equation numerically over a finite element grid. A grid of 64 x 64 points requires 64^2 = 4,094 calculations, but a finer grid of 1024 x 1024 = 1,048,576 requires an exponentially higher number of calculations. This exponential growth can be avoided usian FNO (Fourier Neural Operator instead. Once learned, the FNO quickly computes H(x, y, t) without sensitivity to grid size. Subsequent solutions with varying parameters can be quickly computed without re-evaluating the operator.

How do we discover the FNO? First, we simplify as before using the Fourier transform over the (x, y) space. Recall that the Fourier transform converts from spatial coordinates to frequency coordinates. Thus, (s_x, s_y) are the frequency modes rather than values of temperature – see Sidebar B. The Fourier transform $F\{H(x, y)\} = H(s_x, s_y)$ is shaped by temperatures at all points across the space. At each point (x, y) in space, we have an infinite number of frequencies (s_x, s_y) in the transformed space. *As it turns out, the magnitude of the high-frequency terms quidies out, so it is only necessary to keep a dozen or more lower-frequency terms.*

Transforming the spatial coordinates is the first step in finding a general finite operator that captures the physics as well as the mathematics of the equation. Adjusting the transformed coordinates to fit the physics (via learning) captures the combined influences of spatial shape and the physics of heat, fluids, epidemics, etc.

Frequency terms s_x and s_y are complex numbers representing the amplitude and phase of component waves. They are determined globally by all points in (x, y) space, hence their power to model the overall physics of a system. That is, the 2D Fourier transform is an operator, rather than a simple function. It maps space onto frequencies. The array of s_x and s_y terms forms a tensor that we further shape by a filter R obtained from training a neural network on hundreds of similar examples.

Let $H(s_x, s_y)$ be the 2D Fourier transform (fast Fourier transform FFT in two variables) that converts the heat equation in 2 dimensions into a simpler equation without derivatives $\frac{\partial H(x,y,t)}{\partial x}$ *and* $\frac{\partial H(x,y,t)}{\partial y}$. The transformed heat equation becomes an ordinary first-order differential equation:

$$\frac{dH}{dt} = -c(s_x^2 + s_y^2)H;\ H(x_0, y_0, 0)$$

This is easily solved in terms of time, t, and frequency parameters (s_x, s_y);

$$H(s_x, s_y, t) = H(x_0, y_0, 0)e^{-c(s_x^2 + s_y^2)t}$$

Inverting $H(s_x, s_y, t)$ yields the temperature H(x, y, t) at each point (x, y). In this case, we simply multiply each point in space by the exponential term and invert the transform. But what if the exponential term isn't known or the equation is so complex that finding it is intractable? In general, the unknown tensor R, below, is to be found or discovered so that:

$$H(x, y, t) = FFT^{-1}\{R\, FFT\{H\}\}$$

Citation: Lewis TG. Innovation in Mathematical Physics. Ann Math Phys. 2026;9(3):123-128. Available from: https://dx.doi.org/10.17352/amp.000189

This is where machine learning comes into play. If we train a convolutional neural network CNN on known solutions for various values of c, we can discard the exponential term and replace it with a filter, R. R is the result of training on thousands of examples over various parameter values, c. More generally, a Fourier Neural Operator, FNO, captures the physics of the heat equation regardless of grid size or c. Applying the learned filter R to the transformed result H and then inverting yields the solution in spatial coordinates. The solution is independent of grid size, so calculations are fast.

The elements of tensor R

The tensor R is composed of complex numbers (**a**+**b**i). Unlike a standard neural network that uses single real numbers, each element in R must store two pieces of information to correctly model waves:

- **a:** Determines how much a specific frequency is dampened or amplified. For the heat equation, R learns to "turn up/down the temperature" on high-frequency noise.
- **b:** Determines how much a wave is phase shifted. In fluid flow, this is what represents the velocity or the physical movement of a wave from one (x, y) coordinate to another.
- **Channels:** Additional values such as pressure or values of c may be taken "along for the ride" and processed in parallel with the solution.
- **CNN:** Think of the CNN as a generalized multi-parameter function in a high-dimensional space.

Here is the recipe:

1. Randomize the elements of R.
2. Input a starting heat map H(x, y) into the FNO.
3. Repeat until loss is minimum:
   - The FNO uses R to predict the future heat map, next H.
   - A *loss function* measures the difference between a prediction and a real value.
   - Backpropagation calculates the error gradient and updates the R tensor.

As training progresses, the elements of R eventually converge and settle on values that satisfy the differential equation. The loss function goes to zero or an acceptable error.

- **For the Heat Equation:** The elements of R come from the physical reality of diffusion. They stabilize into values that mimic the exponential decay $e^{-c\left(s_x^2+s_y^2\right)t}$
- **In genera, the** elements of R come from the patterns of energy transfer between big and small transfers seen in the training data.

The operator pipeline

- **Train:** Train a neural network CNN on hundreds or thousands of example input-output pairs.
- **Input:** Initial heat or flow map H(x, y)
- **Transform:** Convert H(x, y) into waves via fast Fourier Transform (FFT)
- **Filter:** Multiply waves by learned weights R.
- **Invert:** Convert waves back to H(x, y) values in space and time.

The invariance of operator learning

Once training is done, the operator is fixed. You can then plug in a completely different input (a new starting swirl or a new obstacle shape), and the operator you computed during training will automatically know how to transform that specific input into the correct physical output. It is a generalization.

Training computes a frequency-based filter that is smart enough to mimic the equation without ever having to solve the differential equations step-by-step. Training is done on smaller grids and a representative range of parameters (like viscosity). This "train small, execute big" approach is one of the main reasons FNOs are so powerful for fluid dynamics problems, which are notoriously difficult to solve.

You do not need to train on the massive, high-res grids you plan to use in production. You train on the "cheapest" grid that still captures the essential physics, making the FNO an extremely efficient method, especially when making many adjustments and re-running the solution.

Libraries for coding in PyTorch

Practitioners don't need to write everything from scratch. Libraries of these specialized physics-informed Neural Network (PINN) frameworks exist for the programming language PyTorch.

- **DeepXDE** handles the geometry, the sampling of points, and the partial differential equation constraints with very high-level functions.
- **Modulus (NVIDIA)** is built specifically for industrial-scale physics simulations using GPUs. It includes pre-built models for Navier-Stokes, etc.

Conclusion

Real-world problems often have non-linearities like chaotic flow in the Navier-Stokes equations or irregular shapes in heat conduction. These real-world cases defy analytic solutions or use enormous amounts of compute, so we must rely on training a neural network to obtain a "universal formula" that we can plug into the fast Fouriertransform equation. The

Citation: Lewis TG. Innovation in Mathematical Physics. Ann Math Phys. 2026;9(3):123-128. Available from: https://dx.doi.org/10.17352/amp.000189

training allows the FNO to learn these "hidden" physics that a standard analytical Fourier solution would miss. Once trained, calculations are very fast, regardless of grid size.

- **Data-Driven Physics:** If you have a fluid that doesn't perfectly follow Navier-Stokes (like a non-Newtonian fluid or blood), you don't need to derive new equations. You just show the FNO data, and it learns the specific spectral weights for that substance.
- **Resolution Independence:** Because the weights are defined on modes (frequencies) rather than values on an (x, y) grid, the model learns the "shape" of the physics. You can feed it a 2D flow at 128×128 resolution, and it will give you the same physical result as if you fed it 512×512. This releases the equation from a specific grid.

Adapting machine learning to solve complex equations is the latest major step in the evolution of mathematical physics. Newton and Liebnitz invented differential calculus, allowing us to build mathematical models of the real world. Pioneers such as Norbert Wiener, who laid the foundations for operator theory, and Oliver Heaviside, known for his heuristic work, developed tools and techniques illustrated here. Laplace and Fourier contributed to the field with their ingenious transforms. They recognized the duality of space-time vs frequency, which enabled the FNO to model a point space as part of a larger connected field. This idea is the structural part of FNO that allows it to model underlying physics and represents a new level of understanding and performance in the annals of mathematical physics.

The solutions obtained by FNO are approximations. The accuracy of the solution depends on the training data, which should contain variations in initial conditions and parameters. As the training data varies relative to the problem being solved, so does the accuracy. Thus, one must be careful when selecting training data. This is particularly important when approximating solutions to chaotic equations such as the Navier-Stokes equation, which is extremely sensitive to initial conditions.

A. Sidebar on finite difference method

Numerical solutions of physical models expressed as differential equations typically perform arithmetic operations on a grid of data points, see Figure 1. The technique is iterative – each sweep across all spatial points at time t is repeated to obtain the solution at time t+1, etc. Figure 1 illustrates this for the simple 2D heat equation with initial conditions of temperature at the left end H(0, y, 0) = 1, spreading out as time passes according to the diffusion rate c. As you can see, the diffusion of heat, measured in terms of temperature, declines exponentially as it spreads from left to right.

The numerical solution over a grid of points is obtained by iterating the finite difference approximation of derivatives. For simplicity, we assume all increments in time and space are equal to one, so that finite differences are simply,

$$\frac{\partial H(x,y,t)}{\partial t} \underline{\underline{def}}\ H(i,j,t+1) - H(i,j,t)$$

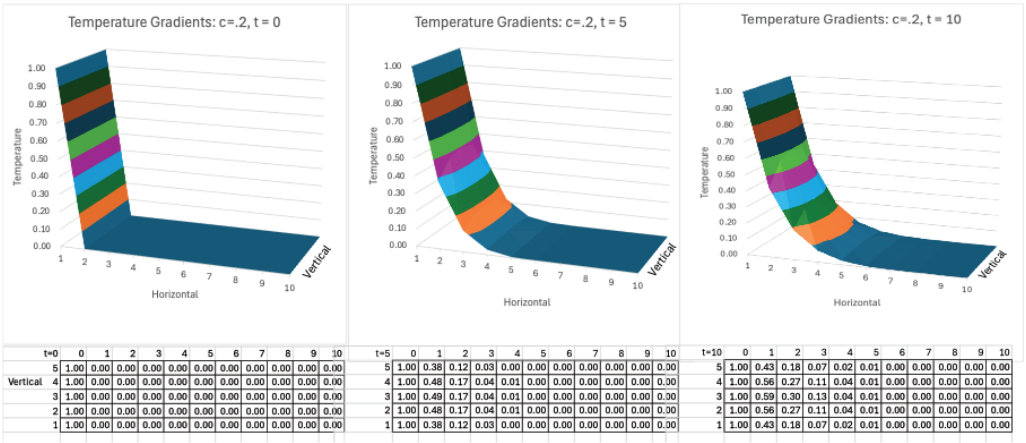
$$\frac{\partial^2 H(x,y,t)}{\partial x^2} \underline{\underline{def}}\ H(i+1,j,t) - 2 * H(i,j,t) + H(i-1,j,t)$$

$$\frac{\partial^2 H(x,y,t)}{\partial y^2} \underline{\underline{def}}\ H(i,j+1,t) - 2 * H(i,j,t) + H(i,j-1,t)$$

Substituting and simplifying yields a finite difference approximation to the continuous heat equation.

$$H(i,j,t+1) = H(i,j,t) + c[H(i+1,j,t) + H(i-1,j,t) + H(i,j+1,t) + H(i,j-1,t) - 4H(i,j,t)]$$

Results for t = 0, 5, 10 are shown with a simple unit initial condition in Figure 1. Temperature decreases exponentially along the horizontal axis in a medium assumed to be uniform with constant conductivity, c = 0.2. Note that a change in the grid means the calculations must be repeated. Increasing the grid size to 1,024 x 1,24, say from 64 x 64, increases the number of calculations 256-fold. The problem with this simple grid solution is scalability.



t=0	0	1	2	3	4	5	6	7	8	9	10
5	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Vertical 4	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

t=5	0	1	2	3	4	5	6	7	8	9	10
5	1.00	0.38	0.12	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	1.00	0.48	0.17	0.04	0.01	0.00	0.00	0.00	0.00	0.00	0.00
3	1.00	0.49	0.17	0.04	0.01	0.00	0.00	0.00	0.00	0.00	0.00
2	1.00	0.48	0.17	0.04	0.01	0.00	0.00	0.00	0.00	0.00	0.00
1	1.00	0.38	0.12	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00

t=10	0	1	2	3	4	5	6	7	8	9	10
5	1.00	0.43	0.18	0.07	0.02	0.01	0.00	0.00	0.00	0.00	0.00
4	1.00	0.56	0.27	0.11	0.04	0.01	0.00	0.00	0.00	0.00	0.00
3	1.00	0.59	0.30	0.13	0.04	0.01	0.00	0.00	0.00	0.00	0.00
2	1.00	0.56	0.27	0.11	0.04	0.01	0.00	0.00	0.00	0.00	0.00
1	1.00	0.43	0.18	0.07	0.02	0.01	0.00	0.00	0.00	0.00	0.00

Figure 1: Numerical finite element (FE) solution to the 2D heat equation using the traditional method.

Citation: Lewis TG. Innovation in Mathematical Physics. Ann Math Phys. 2026;9(3):123-128. Available from: https://dx.doi.org/10.17352/amp.000189

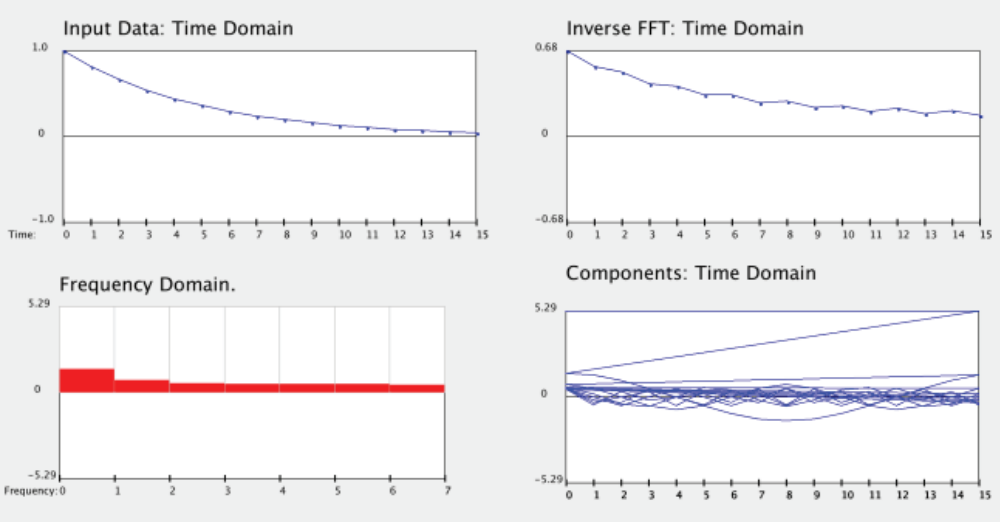


Figure 2: Illustration of 1D FFT and its inverse for points lying on an exponential function. Note the global impact of waves compared with the single-point impact of points in the time domain.

B. Sidebar on FFT and Inverse FFT-1

$$H(s): F\{H(t)\} \underline{\underline{def}} \sum_{m=0}^{n-1} H(t)e^{-i2\pi sm/n}, s = 0,1,\ldots,n-1$$

$$\text{Inverse } H(s): H(t) \underline{\underline{def}} \frac{1}{n}\sum_{s=0}^{n-1} H(s)e^{i2\pi t/n}, t = 0,1,\ldots,n-1$$

The fast Fourier transform, FFT, is the discrete version of the continuous variable Fourier transform. To illustrate its behavior, consider n points lying along the 1D exponential function of time shown in Figure 2. FFT is called "fast" because its execution time complexity is O(n log(n)) for n points. Transforming this time domain set of points to the frequency domain produces (real-valued) frequency modes as shown in Figure 2. The resulting set of frequencies shown below the time domain plot is expanded out into oscillating cosine waves as shown in the Components plot. For each point in the bar chart of the frequency domain plot, a cosine wave appears in the Component time domain plot.

Inverting the FFT reproduces an approximation of the original function. The Inverse FFT of Figure 2 shows the effects of considering only the 8 lowest frequencies of the 16 frequency modes in the FFT. Note that filtering out high-frequency "noise" is a common technique used to simplify FNO.

References

1. Li Z, Kovachki NB, Azizzadenesheli K, Liu B, Bhattacharya K, Stuart AM, et al. A. Fourier neural operator for parametric partial differential equations. In: Proceedings of the International Conference on Learning Representations (ICLR); 2021. Available from: https://doi.org/10.48550/arXiv.2010.08895

2. Li Z, Kovachki NB, Azizzadenesheli K, Liu B, Bhattacharya K, Stuart AM, et al. A. Physics-informed neural operator for learning partial differential equations. ACM/IMS J Data Sci. 2022. Available from: https://doi.org/10.48550/arXiv.2111.03794

3. Kovachki N, Li Z, Liu B, Azizzadenesheli K, Bhattacharya K, Stuart A, et al. A. Neural operator: learning maps between function spaces. J Mach Learn Res. 2023;24(89):1-97. Available from: https://www.jmlr.org/papers/volume24/21-1524/21-1524.pdf

4. Jiao A, He H, Ranade R, Pathak J, Lu L. One-shot learning for solution operators of partial differential equations [Internet]. arXiv; 2021 [cited 2026 Jun 4]. Available from: https://doi.org/10.48550/arXiv.2104.05512

5. Li Z, Kovachki N, Azizzadenesheli K, Liu B, Bhattacharya K, Stuart A, et al. Fourier neural operator for parametric partial differential equations [Internet]. arXiv; 2020 [cited 2026 Jun 4]. Available from: https://doi.org/10.48550/arXiv.2010.08895

6. Quarteroni A, Gervasio P, Regazzoni F. Combining physics-based and data-driven models: advancing the frontiers of research with scientific machine learning [Internet]. arXiv; 2025 [cited 2026 Jun 4]. Available from: https://doi.org/10.1142/S0218202525500125

7. Lu L, Jin P, Pang G, Zhang Z, Karniadakis GE. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nat Mach Intell. 2021;3(3):218-229. Available from: https://doi.org/10.1038/s42256-021-00302-5?urlappend=%3Futm_source%3Dresearchgate.net%26utm_medium%3Darticle

8. Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys. 2019;378:686-707. Available from: https://faculty.sites.iastate.edu/hliu/files/inline-files/PINN_RPK_2019_1.pdf

9. Zhang D, Lu L, Guo L, Karniadakis GE. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. J Comput Phys. 2019;397:108850. Available from: https://doi.org/10.1016/j.jcp.2019.07.048

10. Chen Y, Lu L, Karniadakis GE, Dal Negro L. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. Opt Express. 2020;28(8):11618-11633. Available from: https://doi.org/10.1364/OE.384875

11. Karniadakis GE, Kevrekidis IG, Lu L, Perdikaris P, Wang S, Yang L. Physics-informed machine learning. Nat Rev Phys. 2021;3(6):422-440. Available from: https://doi.org/10.1038/s42254-021-00314-5

Citation: Lewis TG. Innovation in Mathematical Physics. Ann Math Phys. 2026;9(3):123-128. Available from: https://dx.doi.org/10.17352/amp.000189