MACM 416 PROJECT: WAVE EQUATION

CLAIRE CURRY, THAHN DO

1. The Wave Equation. We study the wave equation, a partial differential equation which models the propagation of waves. The specifics of what a solution may represent varies considerably depending on context but includes surface water waves, pressure waves in fluids and vibrations in string. Our particular context is an idealized wave which loses no energy to outside sources.

Define u as a scalar valued function in $\mathbb{R}^d$. We write the wave equation as

$$u_{tt} = c^2 \Delta u$$

where u_t is the partial derivative of u with respect to the variable t and Δu is the *laplacian* operator applied to u. In d dimensions the laplacian is defined as

$$\Delta u = \sum_{n=1}^{d} \frac{\partial^2 u}{\partial x_i^2}$$

where x_i are the spatial variables of u (not including t). The constant c represents the *wave speed* and determines how quickly a wave propagates in the solution.

2. Problem Statement. We are concerned with the wave equation solved on a bounded domain Ω, and further more we wish to enforce certain conditions on the boundary $\partial\Omega$. For this project we wish to solve the two-dimensional wave equation

$$u_{tt} = u_{xx} + u_{yy}$$

with initial conditions

$$u(x,0) = \exp(-[x-a]^2 - [y-b]^2)$$
$$u_t(x,0) = 0$$

and boundary conditions

$$\frac{\partial u}{\partial \hat{n}} |_{\partial\Omega} = 0$$

where $\frac{\partial u}{\partial \hat{n}} = \nabla u \cdot \hat{n}$ with ∇u representing the gradient of u and $\hat{n}$ outward oriented unit normal of Ω.

We take our domain Ω to be a four pointed star defined by the linear interpolation of the points

$$\left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right) \to (0,2) \to \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right) \to (2,0) \to \left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right) \to$$

$$(0,-2) \to \left(-\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right) \to (0,-2) \to \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right).$$

3. Approximation Methods. For some simple or advantageous domains we can produce exact solutions to the bounded wave equation by use of Fourier series. This is accomplished by utilizing the predictable roots of $\sin(x), \cos(x)$ in order to satisfy boundary conditions, and Fourier coefficients to allow a sum of such sinusoids to accurately represent the solution.

TABLE 1
Maximum error over sampled points in test problem at $t = 1$, error factor α, and experimental order of spatial accuracy p

	Maximum Error over Sampled Points at $t = 1$	α	p
Resolution 15	7.3860e-04		
Resolution 30	1.1704e-04	0.1585	2.6574
Resolution 60	5.4842e-05	0.4686	1.0936
Resolution 120	7.5027e-06	0.1368	2.8699

In the case of a complicated domain in multiple dimensions these methods will not be sufficient. In order to solve the wave equations on arbitrary domains we rely on approximation, and in this case the method of finite elements.

The method of finite elements is chosen primarily for its ability to handle arbitrary domain geometry and boundary conditions. For the wave equation in particular, a great deal of its most interesting behavior relies in its interaction with boundaries which allows the opportunity to model constructive and destructive interference created by waves as they reflect off surfaces.

For time stepping with choose an explicit 3-point centered difference approximating a second derivative. We write our time discretization as

$$\frac{\partial^2 u}{\partial t^2}\,|_{t=t_n} = \frac{u(t_{n+1}) - 2u(t_n) + u(t_{n-1})}{\Delta t^2} + O(\Delta t^2).$$

This gives $O(\Delta t^2)$ accuracy without too much computation time. Since our equation has no steady state, we wish to be able to model the system over a large time range. As such, we seek to achieve accuracy with an eye for time stepping efficiency.

Stability is a concern as it is difficult to predict using finite elements, as such we have determined our window of stability through trial and error. We find that a time step of $\Delta t = 10^{-3}$ allows for stability under a fine mesh resolution. As part of the implementation we define a global *resolution* variable R which determines the number of grid points on each boundary. The library FreeFem++ extrapolates this grid spacing to the rest of the domain. With our chosen time step, we find stability for all resolutions tested, specifically $R \in [1, 400]$. For a larger time step $\Delta t = 10^{-2}$, we find stability only for $R \in [1, 49]$.

The abstraction involved with using finite element libraries such as FreeFem++ makes it difficult to ascertain an analytical formula for the order of accuracy in our spatial discretization, however we can use experimental results from Figure 2 to approximate our spatial order of accuracy. In particular, we take the maximum error over sampled grid points at $t = 1$. From this data we can determine a trend in error.

Let E_R be the maximum error over the grid points for a resolution R, then define an *error factor* α by $\alpha E_R = E_{2R}$. Then we calculate our order of spatial accuracy, we'll call it p, as p such that $\frac{1}{2^p} = \alpha$ (Table 1). From these calculations we conservatively take the order of spatial accuracy to be approximately $O(h^2)$ where h is the maximum diameter of triangles in our finite elements mesh. This gives an overall consistency for our method as $O(\Delta t^2 + h^2)$.

4. Test Problem: Forced Wave. Since we possess no exact solutions for our PDE we instead must infer the accuracy of our method from other sources. To the end we will use our numerical method to approximate a problem for which an exact solution is calculable. We should expect an effective approximation to be able to reproduce our test solutions to a satisfactory accuracy.

Suppose a string pinched between two fingers, simultaneously raise the string from both sides, then lower it and continue this in oscillation. The string experiences a uniform oscillating force along its length with both boundary points unable to move. This is our forced wave equation in one dimension which we write as

$$u_{tt} = u_{xx} + \sin(t),$$

where

$$u(x,0) = 0$$
$$u_t(x,0) = 0$$

and at the boundary

$$u(0,t) = u(1,t) = 0.$$

By the use of Fourier methods [3, Chapter 7.2] we can calculate the exact solution

$$u(x,t) = 2\left(\frac{1-(-1)^n}{\pi n(\pi^2 n^2 - 1)}\right)\left(\sin(t) - \frac{1}{\pi n}\sin(n\pi t)\right)\sin(n\pi x).$$

This is exact when written on the page, however there are complications to consider. We cannot compare an approximation to an infinite sum. Therefore, for the purpose of these error calculations we take a truncated sum of all $n \leq 10^6$. This is computationally equivalent to an exact solution as we can judge by the Fourier coefficients

$$2\left(\frac{1-(-1)^n}{\pi n(\pi^2 n^2 - 1)}\right),$$

that any further terms would be proportional to $\frac{1}{10^{18}}$ which is beyond machine error.

To solve the wave equation with the use of finite elements we must first define the variational problem. Define U^n as an approximation of $u(x,y,t_n)$, then we write the problem in its *weak* form as finding U^{n+1} such that for all $v \in \mathcal{V}$,

$$\int_\Omega \frac{1}{\Delta t^2} vU^{n+1}dx + \int_\Omega -\frac{2}{\Delta t^2} vU^n dx + \frac{1}{\Delta t^2} vU^{n-1}dx - \nabla v \cdot \nabla U^n dx - vf dx = 0.$$

We take the test function space $\mathcal{V} := \left\{v \mid \int_\Omega |v|^2 dx < \infty, \int_\Omega |\nabla v|^2 < \infty\right\}$. Then we define a finite dimensional basis of a derivative space $\mathcal{V}_h$ of linear hat functions to approximate the variational problem. Section 7 is the implementation of this finite elements time stepping scheme in the FreeFem++ library [1] with the assistance of a MatLab compatibility library [2].

We can test approximations on the test problem to demonstrate the algorithm's effectiveness. For the smooth test problem 4 we can compare our approximation to the exact solution at $t = 1$ to gauge accuracy.

From Figure 1 we can determine the increasing effectiveness of the algorithm for a higher resolution. Additionally, we can plot the maximum error over a sampled time grid (Figure 2), which shows that the maximum disagreement to our exact answer increases slowly over time, allowing for accuracy on the order of 10^{-4} over a range $t = [0,5]$ for the highest tested resolution. We can also see the continued improvement for a finer resolution of mesh.

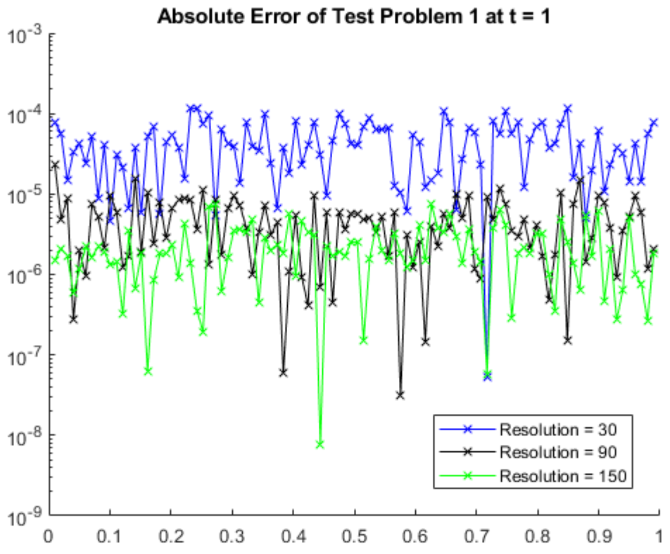

FIG. 1. *Absolute error at $t = 1$ for forced wave test problem at sampled equi-spaced grid points.*

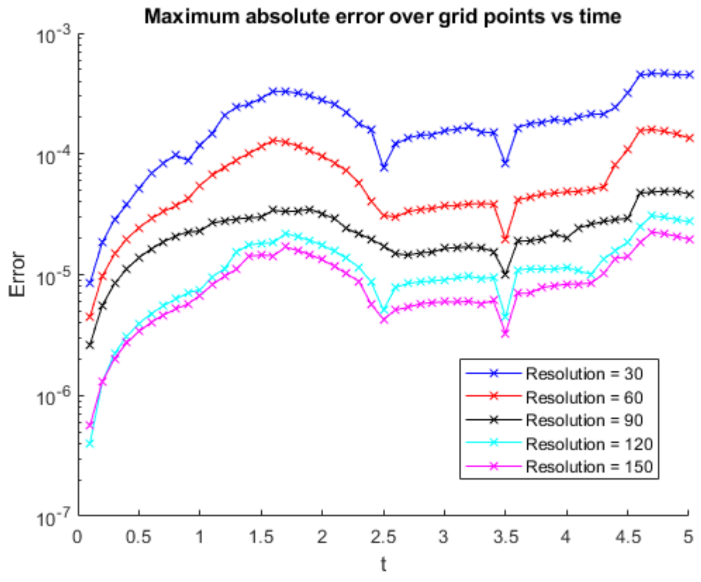

FIG. 2. *Maximum absolute error at grid points for forced wave test problem plotted against time*

5. Approximate Solution the to Wave Equation on a Complex Domain. We can now apply our algorithm to our original problem on a complex domain [2]. We will approximate two variations of initial conditions. The symmetrical version with $a, b = 0$ (no translation) and one with asymmetrical translation $a = -0.4$ and $b = 0.6$. Animation of both approximations can be found on GitHub (https://github.com/ClaireRCurry/MACM-416-Final-Project-Wave-Equation).

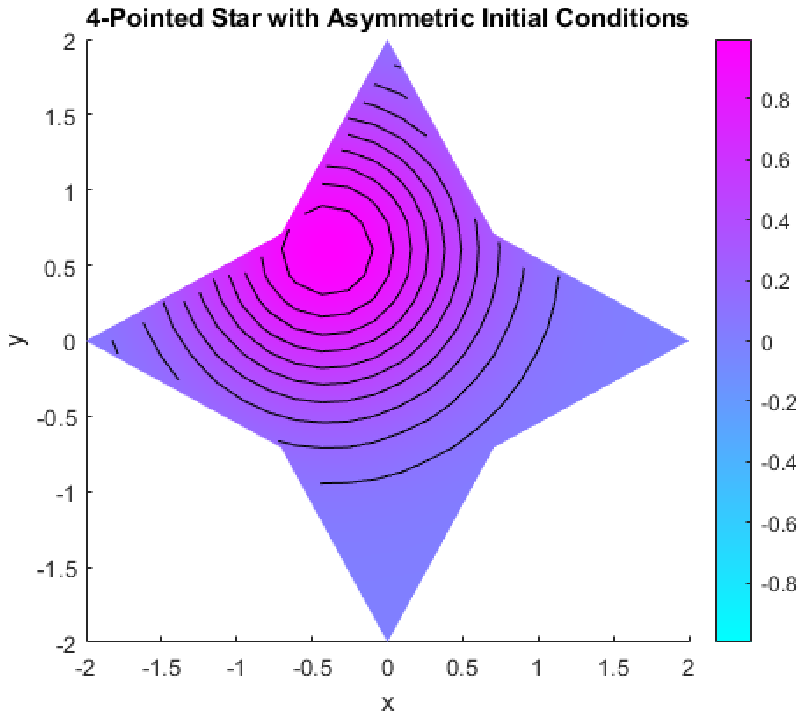

FIG. 3. *Asymmetric initial conditions for the wave equation on a 4-point star domain [2]*

TABLE 2
Difference of minimum and maximum energy over $t \in [0, 5]$ for the approximation of problem 2 with symmetric initial condition

	Total Energy Variation (Symmetric Initial Conditions)
Resolution 30	1.0000e-05
Resolution 60	3.0000e-05
Resolution 90	4.0000e-05

As a measure to gauge accuracy we can include an additional *energy* metric to our numerical analysis, computed as

$$E = \int_\Omega u_{tt}^2 + |\nabla u|^2 dx$$

which remains constant for the non-forced wave equation [6, Chapter 9.1]. We can calculate the energy present at each time step in our approximation as part of our inductive argument for its accuracy. We take our time step to be $\Delta t = 10^{-3}$, then over a time range $t \in [0, 5]$ calculate the absolute difference between our minimum and maximum energy present. We call this our *total energy variation*.

We find from this analysis that our approximation does not appear to be violating the this energy conserving principle beyond what we would expect from the error observed on our test problem. This gives some confidence in our approximations accuracy for this problem.

TABLE 3
Difference of minimum and maximum energy over $t \in [0, 5]$ for the approximation of problem 2 with asymmetric initial condition

	Total Energy Variation (Asymmetric Initial Conditions)
Resolution 30	2.0000e-05
Resolution 60	3.0000e-05
Resolution 90	4.0000e-05

TABLE 4
Maximum difference and average difference between approximations with asymmetric initial conditions at varying resolutions measure at $t = 1$

Resolution	Maximum Difference	Average Difference
$R_1 = 15$, $R_2 = 30$	0.0306	0.0025
$R_1 = 30$, $R_2 = 60$	0.0197	0.0012
$R_1 = 60$, $R_2 = 90$	0.0088	4.6632e-04
$R_1 = 90$, $R_2 = 120$	0.0051	2.6388e-04

Additionally we wish to see that our approximation is converging to some underlying solution as we increase the resolution of our mesh. To this end we compute five approximations with resolutions 15, 30, 60, 90, and 120 at grid points (x_i, y_j), and we denote the value of the approximation at a grid point for a given resolution $U_R(x_i, y_j)$. We wish to compare these approximations in two ways, first a maximum difference between two resolutions R_1 and R_2 defined as

$$D^{max}_{R_1,R_2} := \max_{(x_i,y_j)} |U_{R_1}(x_i, y_j) - U_{R_2}(x_i, y_j)|,$$

and an average difference defined as

$$D^{ave}_{R_1,R_2} := \frac{1}{N} \sum_{(x_i,y_j)} |U_{R_1}(x_i, y_j) - U_{R_2}(x_i, y_j)|,$$

where N is the total number of grid points (Table 4).

This analysis supplies evidence that our approximation is converging to some underlying function given the speed at which our difference metrics are decreasing. From this and our energy calculations we believe that our approximation is genuinely modeling the true solution and more accurately so for finer resolution meshes.

Finally we will discuss the efficiency of this algorithm and we find some positives and some negatives. The benefits of this method is its accuracy for coarse meshes. Referring to our analysis of the test problem in 2 we find that even for the largest meshes tested, we achieve an error less than 10^{-3} which is quite sufficient for applications such as generating animations. However, in order to access the greater accuracy at finer meshes there are computation times to consider. Table 5 gives the compute time of our approximation for the case of problem 2 with asymmetric initial conditions for $t \in [0, 1]$.

As a qualitative summary of this analysis, we believe this is an effective approximation algorithm for the problem 2 posed at the beginning of this paper. The strengths of this algorithm lies in its relatively high baseline accuracy for coarse meshes, and some of its downsides lie in the increasing computation costs for finer meshes.

TABLE 5
Computation time taken for problem 2 with asymmetric initial conditions for $t = [0, 1]$

Resolution	Compute Time (seconds)
15	61.84
30	56.94
60	156.22
90	393.22
120	702.47
150	1148.51

We encourage the reader to modify our code for their own approximations on complex or absurdly shaped domains–perhaps one in the shape of a cat.

6. Neural Network Approach. In this section we will discuss about a comparison between our method (Finite Element Method - FEM) and a Physic Informed Neural Network (PINN) on the test problem implemented in MatLab [5][4][7]. In this section, we will compare: accuracy, efficiency, and CPU-time taken.

The overview is that Physics-Informed Neural Networks (PINNs) are a class of neural networks designed to incorporate physical laws; therefore, it is formless and untrained. We first need to define the NN architecture then train the data based on a combination of intial condition and boundary condition.

As previously defined, we are trying to solve for the **Wave Equation**

$$u_{tt} = c^2 \Delta u$$

For unknown function $u(x, y, t)$ and c to be the speed of the wave. We using the Gaussian Distribution function as **Initial condition**:

$$u(x, y, 0) = \phi(x, y) = A \cdot e^{-x^2 - y^2}$$

With the Neumann **Boundary condition**:

$$\frac{\partial u}{\partial n} = 0$$

The **analytical solution** for the equation is:

$$u(x, y, t) = exp\left[-(\sqrt{x^2 + y^2} - t)^2\right]$$

After the training process, the product will be a NN can predict the same PDE, starting condition, and boundary condition as $t \to \infty$ from a sample of $t : 0 \to 5$.

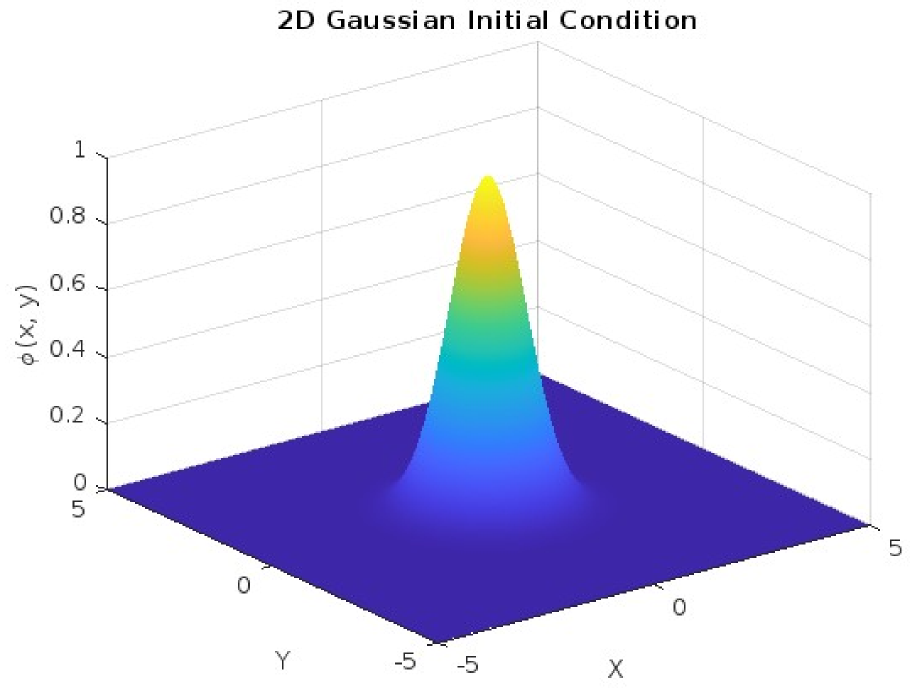

FIG. 4. *Initial condition.*

PINNs use a neural network to approximate the solution to PDEs by minimizing a composite loss function that enforces PDE residuals, initial conditions, and boundary conditions. We implemented neural network with 3 hidden layers and 10 neurons per layer, trained using the Levenberg-Marquardt algorithm. By using Levenberg-Marquardt algorithm, we do not need to manually define the Loss function. This approach is Mesh-free; however, it can only predict data from the trained domain, anything beyond that, the method will return outright incorrect result.

Metric	FEM	PINN
Accuracy but Error often to be $1e^{-3}$ it still have low accuracy	Depends on the case With enough data and training,	
Efficiency a problem can be solve efficiently iterative training process.	Due to sparse matrix, Less efficient due to	
CPU time after that, the prediction cost less than 2 seconds	roughly 10 - 15s	solid 2m30-ish seconds to train

6.1. Accuracy. FEM achieved higher accuracy for this test problem due to its structured discretization and convergence guarantees when using finer meshes. The PINN solution had bad accuracy, with slight deviations near steep gradients or regions requiring fine detail. This is likely due to limited network capacity or insufficient training iterations. On some region, the error peak as high as 0.25 from the original solution.

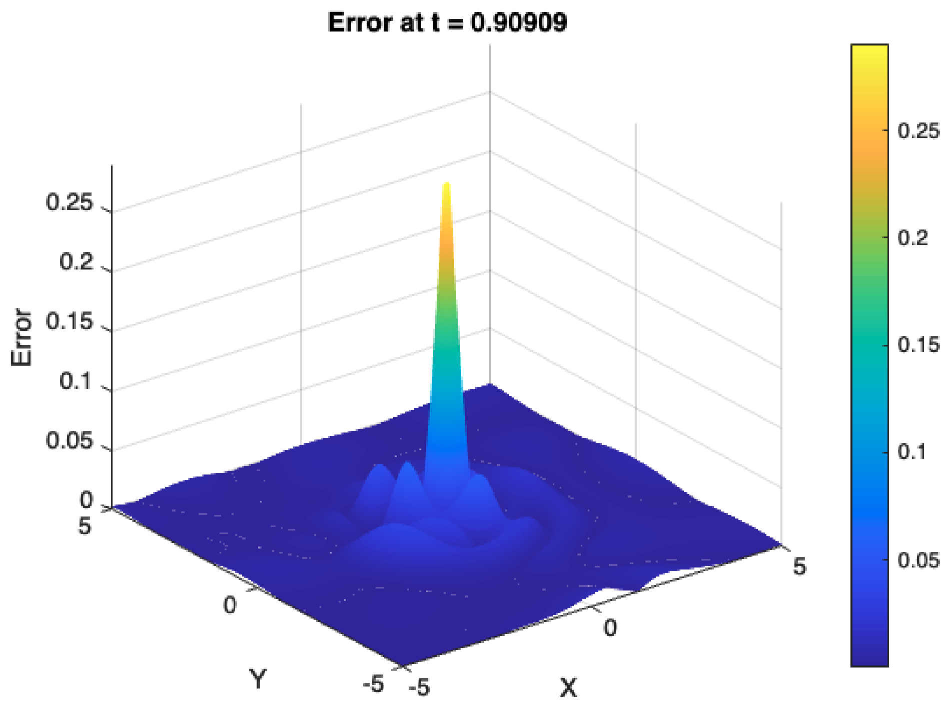

Fig. 5. *Error at $t \approx 1$*

FEM requires a fine mesh for high accuracy, which can be potentially expensive for complex domains. PINNs avoid meshing but rely on adequate training data and network capacity to capture sharp solution features.

6.2. Efficiency. FEM was more efficient in terms of computation for structured problems, as the sparsity of the system matrices enabled faster solutions. PINNs required significantly more computational time due to the iterative optimization process. The computational cost scales with the number of training iterations and the size of the neural network.

FEM is well-suited for structured problems in lower dimensions due to its optimized sparse matrix solvers. PINNs show promise for high-dimensional problems, where meshing and matrix assembly become prohibitive for FEM.

6.3. CPU Time. FEM completed the solution in is generally shorter on regulated domain, primarily spent on matrix assembly and sparse matrix solving. PINNs required really long time for training, as each training epoch involved backpropagation and gradient computations for all training samples. Afterward, the prediction for various time t can be assembled relatively quick.

7. Implementation in FreeFem++.

```

include "ffmatlib.idp"

// Initial Conditions
func ic = (exp(-(x)^2 - (y)^2));

// Timestep
real dt = 0.001;

// End time
real T = 10;

// Precomputing time step scaling constant for efficiency
real idt2 = 1/(dt^2);

// Number of grid points on each boundary
// Mesh is extrapolated based on the density
// of gridpoints on the boundary
int resolution = -10;

// Defining the boundary as a single object with one ID
int C0 = 100;

// Defining the domain boundary with piecewise parametric functions
border C01(t=0., 1.){x=(1-t)*(-1/sqrt(2.))           ; y=(1-t)*(1/sqrt(2.)) + 2.*t;     label=C0;}
border C02(t=0., 1.){x=1/sqrt(2.)*t                  ; y=(1-t)*(2.) + (1/sqrt(2.))*t;   label=C0;}
border C03(t=0., 1.){x=(1-t)*(1/sqrt(2.)) + 2.*t     ; y=(1-t)*(1/sqrt(2.));            label=C0;}
border C04(t=0., 1.){x=(1-t)*2. + (1/sqrt(2.))*t     ; y=(-1/sqrt(2.))*t;               label=C0;}
border C05(t=0., 1.){x=(1-t)*(1/sqrt(2.))            ; y=(1-t)*(-1/sqrt(2.)) + -2.*t;   label=C0;}
border C06(t=0., 1.){x=-1/sqrt(2.)*t                 ; y=(1-t)*(-2.) + (-1/sqrt(2.))*t; label=C0;}
border C07(t=0., 1.){x=(1-t)*(-1/sqrt(2.)) + -2.*t   ; y=(1-t)*(-1/sqrt(2.));           label=C0;}
border C08(t=0., 1.){x=(1-t)*(-2.) + (-1/sqrt(2.))*t ; y=(1/sqrt(2.))*t;                label=C0;}

// Show boundary
plot(    C01(resolution) +
    C02(resolution) +
    C03(resolution) +
    C04(resolution) +
    C05(resolution) +
    C06(resolution) +
    C07(resolution) +
    C08(resolution), wait=true);

// Construct the mesh
mesh Th = buildmesh(
    C01(resolution) +
    C02(resolution) +
    C03(resolution) +
    C04(resolution) +
    C05(resolution) +
    C06(resolution) +
    C07(resolution) +
    C08(resolution)
    );

// Show mesh
plot(Th, wait=true);

// Defining linear finite element space
fespace Vh(Th,P1);

// Both mid point and previous point in time defined as the initial conditions
// This ensures our first time derivative is equal to zero in our time step
Vh u, v, umid=ic, uold=ic;

// Plot initial conditions
plot(uold, value=true, wait=true);

// Defining the time stepped wave equation via a variational formula.
// Next time step u is calculated based on two previous time steps
// using a centered three point approximation. O(dt^2) accuracy
problem wave(u,v)
    = int2d(Th)(u*v*idt2) + int2d(Th)(-2*umid*v*idt2 + dx(v)*dx(umid) + dy(v)*dy(umid) + uold*v*idt2);



// Saving approximation data to external files
ofstream ff("HPMovie.dat");
savemesh(Th,"HPMovie.msh");
ffSaveVh(Th,Vh,"HPMovieVh.txt");

// Initializing first step of the wave equation
wave;

// Save a snap shot every "saveEvery" time steps.
int count = 0;
int saveEvery = 100;

// Save first timestep
ffSaveData(umid,"HPMovie"+count/saveEvery+".txt");
for(real t = 0; t < T; t += dt){

    // Updating our time steps
    uold = umid;
    umid = u;

    // Solve the equation for u(t_n+1)
    wave;

    // Plot to observe the evolution
    plot(u,value=true,fill=true);

    // Save snapshots at specified timestep
    if(count % saveEvery == 0){
        ffSaveData(umid,"HPMovie"+count/saveEvery+".txt");
    };
    count += 1;
}
```

REFERENCES

[1] https://doc.freefem.org/introduction/index.html.
[2] https://github.com/samplemaker/freefem_matlab_octave_plot.
[3] M. S. Gockenbach, *Partial Differential Equations: Analytical and Numerical Methods*, siam, 2002.
[4] https://github.com/matlab-deep-learning/Physics-Informed-Neural-Networks-for-Heat-Transfer.
[5] https://www.mathworks.com/help/deeplearning/ug/solve-partial-differential-equations-with-lbfgs-method-and-deep-learning.html.
[6] W. A. Strauss, *Partial Differential Equations, An Introduction*, John Wiley Sons, 2008.
[7] https://www.mathworks.com/matlabcentral/fileexchange/172049-pinn-loss-function-generation-with-symbolic-math.