Finite Element Method for 2D Heat Equation

1st Tianyuan Wu
ShanghaiTech University

2nd Mengying Wu
ShanghaiTech University

***Abstract*—In this paper, we implement a fast, parallel finite element method solver for 2D heat equation, a mesh generator based on delaunay triangulation and a GUI which is able to show the dynamic changes of temperature ($T(x, y)$) by time (t) of a given region with initial conditions. Then, we test our solver on differentregions and different boundary conditions. Also, we analysis the error between our numeric solution and the anylatical solution.**

***Index Terms*—Finite element method, Heat equation, Delaunay triangulation, Error analysis**

I. Introduction

Finite element method (FEM) is the most commonly used method for solving differential equations of engineering and mathematical models, such as structural analysis, heat transfer and fluid flow [1]. Our work focus on the application of FEM in solving 2D heat equation, error analysis of FEM in special designed problems and visualization of heat transfer. It basically includes 3 parts: (1) The FEM solver for 2D heat equation; (2) The mesh construction algorithm; (3) The heat transfer visualization and animation part. All 3 parts are implemented in Python, with third-party libraries `numpy` [8] (for matrix computations), `scipy` [10] (for basic linear algebra support, like solving linear equations), and `matplotlib` [9] (for animation and visualization)

II. 2D Heat Equation

Heat equation is a certain partial differential equation which describes the heat transfers in a specific system. The 2-dimensional form of the equation is known as:

$$\frac{\partial T}{\partial t} = \lambda(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}) + f(x, y, t)$$

Where $T = T(x, y, t)$ is the heat (or temperature) of given position (x, y) at time point t, $f(x, y, t)$ describes the external heat given to the system, which is similar to the "external force" in mechanics and λ is a given constant describes the property of the system.

To solve a specific 2D heat transfer problem, the following conditions must be specified first:

1) The region Ω and the initial condition in Ω:

$$T_{\Omega,initial} = T_\Omega(x, y, 0)$$

2) Neumann boundary conditions

$$\frac{\partial T}{\partial \vec{n}}\Big|_{\partial\Omega_n} = T_n$$

3) Dirichlet boundary conditions

$$T\Big|_{\partial\Omega_d} = T_d$$

Once the initial condition, Dirichlet and Neumann boundary conditions are specified, we can simulate the state of the system at any time.

III. Methods

A. Finite Difference Method

This equation is actually a 3-dimensional partial differential equation, which is computational expensive if we use a pure FEM solver. Hence, we use finite difference method to discretize the time dimension:

$$\frac{T(x, y, t_{n+1}) - T(x, y, t_n)}{dt} = \lambda(\frac{\partial^2 T(x, y, n+1)}{\partial x^2} + \frac{\partial^2 T(x, y, n+1)}{\partial y^2}) + f(x, y, t_{n+1})$$

To simplify the problem, we take $\lambda = 1$, and denote $\frac{\partial^2 T_n}{\partial x^2} + \frac{\partial^2 T_n}{\partial y^2}$ as ΔT_n, so the equation can be written as:

$$\frac{T_{n+1} - T_n}{dt} = \Delta T_{n+1} + f_{n+1}$$

To maintain the stability of the solution, we use the implicit Euler form of this equation:

$$T_{n+1}(1 - \Delta T_{n+1} dt) = f_{n+1} dt + T_n$$

Consequently, once the heat distribution at $t = n$ is computed by FEM, we can calculate the solution at $t = n + 1$ by finite difference method with limited error rate.

B. Finite Element Method

By finite difference method, the 3-dimensional problem is simplified into two dimensions, then we will apply finite element method to solve it. Suppose we discretize the region into k nodes $\{n_1, n_2, \dots, n_k\}$, which make up m triangle mesh $\{m_1, m_2, \dots, m_p\}$, Then the discreted solution is:

$$T_{sol} = T_{sol,d} + T_{rest}$$

where $T_{sol,d}$ is to fit the Dirichlet boundaries.

Then, we use Galerkin's method to solve T_{rest}. Denote $\{\phi_1, \phi_2, \dots, \phi_k\}$ as the basis of k nodes. For each base function ϕ, we have

$$\phi_j(N_i) = \begin{cases} 1, & i = j \\ 0, & \text{otherwise} \end{cases}$$

The visualization of this base function is shown in Fig.1. [2]

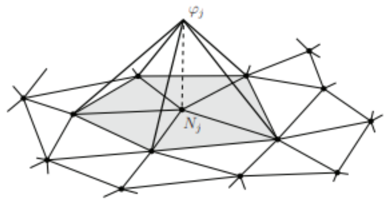

Fig. 1. Visualization of a 2D base function

Notice that, the following equation holds for any base function ϕ_j:

$$\int_\Omega T_{rest,n}\phi_j dx + (\int_\Omega \nabla T_{rest,n} \cdot \nabla \phi_j dx) \cdot dt =$$
$$\int_\Omega T_{rest,n-1}\phi_j dx + (\int_{\partial\Omega_n} T^{neum}_{rest,n}\phi_j ds) \cdot dt + (\int_\Omega f_n \phi_j dx) \cdot dt$$

And T_{rest} and time t can be represented as a linear combination of all basis.

$$T_{rest,t} = \sum_{i=1}^{k} \alpha_i \phi_t^i$$

To convert the integral equation into a linear system which is convienent of solving by computers, we construct the mass matrix M:

$$M_{ij} = \sum_{A \in Tri} \int_A \phi_i \phi_j dx$$

and the stiffness (coefficient) matrix A:

$$A_{ij} = \int_\Omega \nabla \phi_i \cdot \nabla \phi_j dx$$

Finally, we can construct the following linear system:

$$(Adt + M)T_{rest,n} = MT_{rest,n-1}$$
$$+ (\int_{\partial\Omega_n} T^{neum}_{rest,n}\phi_j ds) \cdot dt + (\int_\Omega f_n \phi_j dx) \cdot dt$$

C. Mesh Construction - Delaunay Triangulation

Another key difference between 1D and 2D problems is, we need a proper method to generate triangle mesh for the FEM solver. For this problem, we refer to the Delaunay triangulation [4] [5] [7], which is a triangulation algorithm for a given set of discrete points P such that no point in P is inside the circumcircle of any triangle in $DT(P)$. Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation [4].
There are several algorithms available for Delaunay triangulation, here we use the Bowyer-Watson's algorithm [6]. The pseudo-code of this algorithm is shown in Algorithm 1.
After one iteration of triangulation, we traverse all edges, and find which ones are longer than the given threshold. Then we add additional points pm these edges to split them into shorter edges. Then run Delaunay triangulation recursively, until edges of all triangles are smaller enough. The triangulation result

Algorithm 1 Bowyer-Watson's Algorithm

```
Input Points := $\{p_1, p_2, \ldots, p_n\}$
 1: Triangulation := []
 2: Add a super triangle including all points in Triangulation
 3: for each point in Points do
 4:     badTriangles := {}
 5:     for each triangle in Triangulation do
 6:         if point is inside triangle's circumcircle then
 7:             badTriangles = badTriangles $\bigcup$ triangle
 8:         end if
 9:     end for
10:     Polygon := []
11:     for each triangle in badTriangles do
12:         for each edge in triangle do
13:             if edge is not shared by badTriangles then
14:                 Add edge to Polygon
15:             end if
16:         end for
17:     end for
18:     for each triangle in badTriangles do
19:         Remove triangle from Triangulation
20:     end for
21:     for each edge in Polygon do
22:         NewTriangle := Triangle(edge, point)
23:         Add NewTriangle to Triangulation
24:     end for
25:     for each triangle in Triangulation do
26:         if triangle contains a vertex in super triangle then
27:             Remove triangle from Triangulation
28:         end if
29:     end for
30: end for
31: return Triangulation
```

of the 1st run is shown in Fig.2, and the final result is shown in Fig.3. Also, we show the triangulation result of `scipy.spatial.Delaunay` [10] in Fig.4 as a benchmark for comparison. It's shown that our performance is similar to the benchmark.

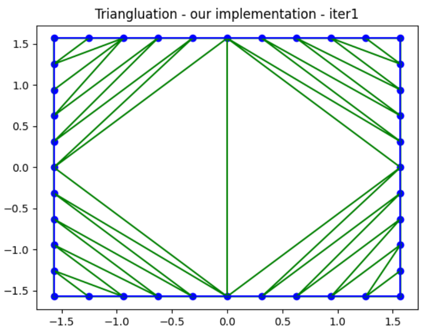

Fig. 2. Iteration 1 of delaunay triangulation

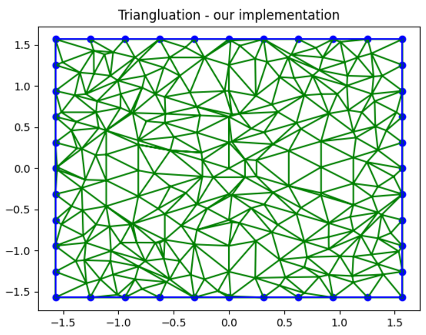

Fig. 3. Triangulation result of our algorithm

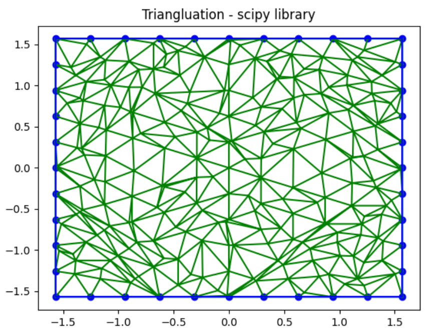

Fig. 4. Benchmark (scipy)

IV. Results

A. Performance Evaluation

Our FEM solver implemented all in Python, with linear system solver `scipy.linalg.lsqr`, which is fast and parallel. The test platform of our FEM solver is `Intel(R) i7-8700k` (6 core, 12 threads), 16GB memory, with `Ubuntu 20.04`. The performance is shown in Table 1.

TABLE I
Performance of FEM solver

# of Nodes	Solving Time
253	3.58s
282	4.58s
717	9.42s

B. Solution Visualization

We test our FEM solver in the region and boundary conditions shown in Fig.5. The red lines represents Neumann boundary conditions, the blue lines represents the Dirichlet boundary conditions, and the green lines shows the triangle mesh.

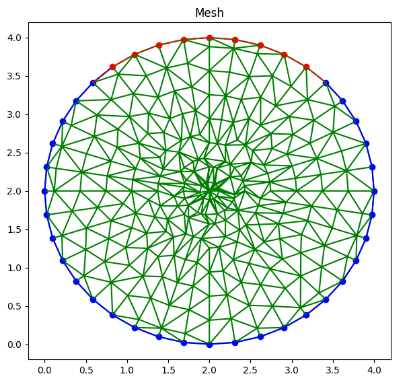

Fig. 5. Problem visualization

The configuration of first case is, set $f(x, y, t) = 10$, Neumann boundary $\frac{\partial T}{\partial \vec{n}}\big|_{\partial\Omega_n} = 0$, Dirichlet boundary $T\big|_{\partial\Omega_d} = 0$ and the initial condition $T_{initial} = 0$. Run 100 time steps, with each step 0.01s. The result is shown in Fig.6

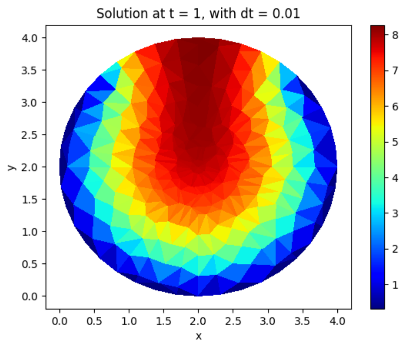

Fig. 6. Solution of case 1

The other conditions of the second case is the same as the first one, except the Neumann boundary $\frac{\partial T}{\partial \vec{n}}\big|_{\partial\Omega_n} = -10$. The result is shown in Fig.7

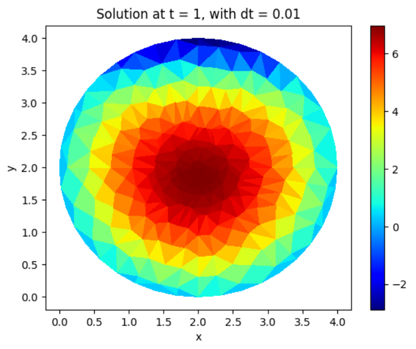

Fig. 7. Solution of case 2

The thrid case is, set all other conditions to 0, except the Neumann boundary $\frac{\partial T}{\partial \vec{n}}\big|_{\partial\Omega_n} = 10$, and the result is shown in

Fig.8

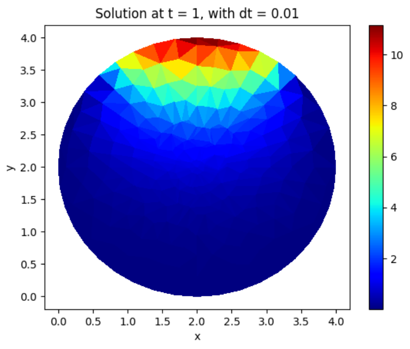

Fig. 8. Solution of case 3

Limited to the PDF format, the dynamic visualization cannot be shown in this paper. But the animation can be done with `FEM_Solver.show_animate` in our implementation.

V. Error Analysis

A. Problem Construction

To test the accuracy of our solver, we specially designed a problem with prior anylatical solution. The problem is shown in Fig.3, which is defined in $[-\frac{\pi}{2}, \frac{\pi}{2}] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$, with all boundary are Dirichlet boundary, and $T\big|_{\partial\Omega_d} = 0$. The initial state is

$$T(x, y, 0) = cos(x)cos(y)$$

and we can get the anylatical solution is:

$$T(x, y, t) = cos(x)cos(y)e^{-2t}$$

B. Comparison & Analysis

We simulate the following problem 100 steps with $dt = 0.01$, and the solution of our solver is shown in Fig.9, while the anylatical solution is shown in Fig.10. It's no significant difference between the 2 solutions.

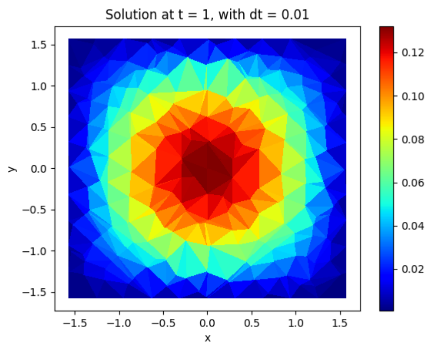

Fig. 9. Our solution

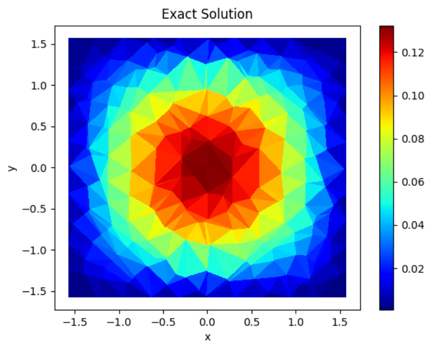

Fig. 10. Anylatical solution

Then, we analysis the error rate between the two solutions by time, and plot the error rate in Fig.11. We can see that the error rate is less than 1%, and grows linearly by time.

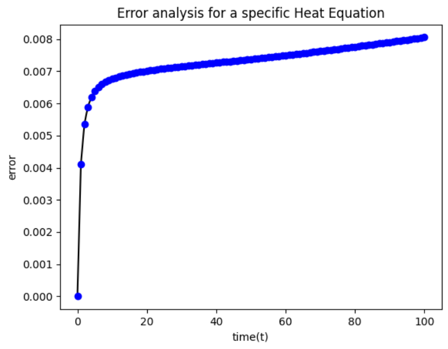

Fig. 11. Relationship between error and time

VI. Conclusion

In this work, we implement a FEM solver for 2D heat equation, and test its performance and accuracy in different conditions. Also, we implemented a mesh generation algorithm to construct proper mesh for our solver. Then, we use `matplotlib` [9] to visualize the solution, and creates animation to show the heat transfer. Finally, we analysis the error rate of our solver. In conclusion, our implementation is fast, easy for visualization and have a relative good accuracy.

References

[1] Finite Element Method, wikipedia (http://en.wikipedia.org/wiki/Finite_element_method)

[2] M. G. Larson and F. Bengzon, The Finite Element Method: Theory, Implementation, and Applications, vol. 10. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[3] P. Ciarlet, The Finite Element Method for Elliptic Problems, NorthHolland, New York, 1980.

[4] Delaunay triangulation, wikipedia (http://en.wikipedia.org/wiki/Delaunay_triangulation)

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Cheong, Computational Geometry: Algorithms and Applications. Springer-Verlag, 2008.

[6] Bowyer-Watson's algorithm, wikipedia (http://en.wikipedia.org/wiki/Bowyer%E2%80%93Watson_algorithm)

[7] B. Delaunay: Sur la sphere vide, Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk, 7:793-800, 1934

[8] Numpy, https://numpy.org/

[9] Matplotlib, https://matplotlib.org/

[10] Scipy, https://www.scipy.org/