

MATHEMATICAL MODELING AND ANALYSIS OF TEMPERATURE CONTROL PROCESSES IN SMART HOME SYSTEMS

SUYAROV U., AXMADALIYEV SH., SOBIRJONOV S.

Student Fergana State University, suyarov1857@gmail.com,

shuxratjonaxmadaliyev15@gmail.com, Siyavushxujayev@gmail.com

Abstract: This paper examines the problem of real-time mathematical modeling of indoor temperature distribution in smart home systems. The study is based on the heat diffusion equation $\partial U / \partial t = a^2 \nabla^2 U$ from mathematical physics, implemented via HTML5, CSS3, and JavaScript on a 30×50 discrete grid using the Explicit Euler numerical method. Physical realism is ensured by radiator, window, and door boundary conditions, stochastic noise effects, and a multi-stage color gradient visualization.

Keywords: heat diffusion equation, Explicit Euler method, 2D Laplacian operator, discrete modeling, boundary conditions, Canvas API, smart home system, heat map, thermal diffusivity coefficient, stochastic noise, temperature gradient.

Introduction. Smart Home technologies are becoming increasingly widespread in modern life. When designing systems for automatic control of indoor climate — temperature and humidity indicators — engineers require not only sensor devices and control algorithms, but also an accurate mathematical model of the physical processes involved. If we can precisely describe how heat spreads within a room, it becomes considerably easier to optimally position heating or cooling devices, to accurately calculate their power requirements, and to conserve energy. This very necessity — recreating physical reality in a digital environment — served as the scientific motivation for this project. The project is based on a simple yet deeply considered idea: to implement a centuries-old mathematical physics equation in a modern web browser environment accessible to all users. For this purpose, standard HTML5, CSS3, and JavaScript technologies were chosen rather than complex programming environments or specialized scientific computing packages. This approach makes the project adaptable to a wide audience, as a single modern web browser is sufficient to run it.

The relevance of this research lies in the fact that temperature control systems are of vital importance not only in residential buildings, but also in temperature-sensitive environments such as industrial facilities, medical institutions, and data centers. A real-time simulation system — especially one running in a web browser — is valuable as a rapid prototyping tool for engineers and researchers. At the same time, this project is highly effective for educational purposes: not merely solving a mathematical physics equation on paper, but experiencing it as a visible, real-time changing visualization greatly deepens the student's understanding. The article systematically presents the complete scientific and technical description of the project — the mathematical apparatus employed, the software architecture, and the results obtained.

Mathematical Foundations: The Heat Diffusion Equation

In any physical medium — gas, liquid, or solid — heat spreads from one point to another. This process was first systematically described in 1822 by the French mathematical physicist Jean Baptiste Joseph Fourier, who expressed the change in the temperature field over time through a partial differential equation. This equation is known today as the Fourier heat equation or the heat diffusion equation, and is considered one of the most fundamental results of mathematical physics. For the one-dimensional case, the equation is written as $\partial U / \partial t = a^2 \cdot \partial^2 U / \partial x^2$, where $U(x, t)$ is the temperature value at coordinate x and time moment t , and a^2 is the thermal diffusivity coefficient of the medium, which depends on the material's thermal conductivity, density, and specific heat capacity. For modeling the temperature field of a room, the two-dimensional case is required, since the room's plane is described by two spatial coordinates — x and y .

$$\partial U / \partial t = a^2 \cdot \nabla^2 U = a^2 \cdot (\partial^2 U / \partial x^2 + \partial^2 U / \partial y^2) (1)$$

Formula 1. Two-dimensional heat diffusion equation

In this expression, the ∇^2 symbol denotes the Laplace operator, which is often referred to in mathematics as "delta" or "nabla squared." In two-dimensional space, the Laplace operator consists of the sum of second-order partial derivatives along the x and y directions: $\nabla^2 U = \partial^2 U / \partial x^2 + \partial^2 U / \partial y^2$. The physical meaning is quite clear: if the temperature of a given cell is higher than the temperatures of surrounding cells, that location "loses" heat to its neighbors; otherwise, it absorbs heat from them. The mathematical nature of the heat diffusion equation belongs to the class of parabolic partial differential equations. This equation is solved in the forward direction in time: that is, we can compute time steps sequentially starting from the initial condition. Furthermore, since the equation is linear, the superposition principle applies, meaning the total temperature field can be obtained by adding the individual contributions of various heat sources — radiator, window, and other factors. This property is particularly useful when constructing multi-source room models.

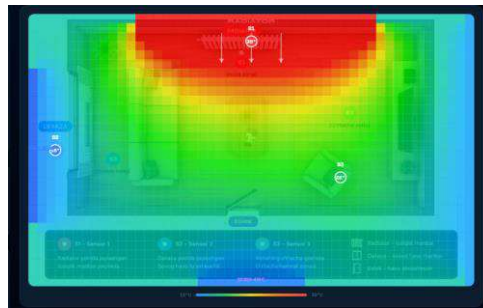


Figure 1. Stages of the heat diffusion process

Explicit Euler Method and 2D Laplacian Discretization

The heat diffusion equation is, by its nature, defined in a continuous domain. However, in computer computations, a continuous domain cannot be represented directly — it must be converted into a discrete grid with a finite number of points. In the project, the room plane is divided into grid cells consisting of 30 rows (ROWS = 30) and 50 columns (COLS = 50). Each cell (r, c) stores its temperature value as `grid[r][c]`; this corresponds in practice to cells of size `cellW = 18 px` and `cellH ≈ 18.3 px` in a `900×550 pixel` canvas element. For time integration, the classical Explicit Euler (Forward Euler) numerical method is employed — the essence of this method being that the state at the next time step is computed using the state at the current time step. The update scheme was chosen because it requires low computational cost and is very straightforward to implement.

$$U[r][c](t + 1) = U[r][c](t) + a^2 \times L[r][c](2)$$

$$L[r][c] = U[r - 1][c] + U[r + 1][c] + U[r][c - 1] + U[r][c + 1] - 4 \times U[r][c] (3)$$

Formulas 2–3. Heat equation discretized using the Explicit Euler method

In the formulas above, $L[r][c]$ is the discrete approximation of the 2D Laplacian operator, representing a comparison of a given cell's temperature against its four neighboring cells (top, bottom, left, right). This discrete approximation is called the second-order finite differences scheme in mathematical analysis. In the project code, the thermal diffusivity coefficient is set to $a^2 = 0.22$; this value is normalized to the pixel size and time step in the given coordinate system, and plays an important role in ensuring the stability of the simulation. The stability condition for the Explicit Euler method in mathematical physics requires: $a^2 \times \Delta t / (\Delta x^2) \leq 0.5$. To ensure this condition is satisfied in the project, `diffusionStep(10)` is called each animation frame — that is, 10 outer iterations are performed per frame, each containing 5 inner iterations; thus after a total of 50 Euler steps, noise

addition and screen update are performed. This approach ensures the temperature field converges to stability more quickly and improves the smoothness of the visualization.

Software Architecture of the Project

The project consists of three main files: `index.html`, `style.css`, and `script.js`. This is based on the separation of concerns principle of classic multi-layered web architecture: HTML handles content structure, CSS handles visual appearance, and JavaScript manages business logic and computations. The `index.html` file declares a 900×550 pixel rectangular drawing area of the Canvas API — the heat map layers are drawn directly onto this canvas element. Three main blocks are distinguished in the HTML file: a header panel (header), a left control panel (aside.controls), and the room visualization on the right (section.room-section). The header panel contains the project name, system status indicator, and a clock displaying the real time. The left panel contains three sensor cards — S1 Radiator, S2 Window, S3 Central — a visibility sliders block, a statistics panel, and buttons for controlling the simulation. A numerical input field is provided to the user for each sensor, allowing the value to be changed from -50°C to 100°C .

The CSS file defines the visual layer. CSS variables (custom properties) declared in the `:root` selector block centrally manage the entire interface color palette: `--bg-deep: #020617` — the deepest dark background, `--bg-panel: #0f1e2e` — a slightly lighter blue-black for panels, `--sky: #38bdf8` — the primary accent color. Colors are semantically assigned for sensors: red (`#ef4444`) for the heat source — radiator, blue (`#3b82f6`) for the cold source — window, and green (`#22c55e`) for the central sensor. The Space Mono font loaded via Google Fonts is used for numerical data and formulas, while the Syne font is used for general interface text. The `script.js` file is considered the «brain» of the project and consists of four logical blocks: state and constants declaration, grid management (`initGrid` and `applyBoundaries`), computation core (`diffusionStep` and `addNoise`), and rendering (`draw` and its helper functions — `drawArrows`, `drawSensors`, `drawLabels`). This modular structure has significantly facilitated reading and extending the code.



Figure 3. General user interface of the project

Boundary Conditions and Justification of the Physical Model. Solving the heat diffusion equation requires more than the differential equation alone — it must also be supplemented with boundary conditions and initial conditions. Boundary conditions specify the temperature at the edges or critical points of the computational domain. Three main boundary conditions are applied in the project, which are referred to in mathematics as Dirichlet-type conditions — that is, the temperature value is directly set to a specific value. The first boundary condition is the radiator region, located in the upper portion of the grid, defined from column 12 to column 38 of row 0, with row 1 also defined in the same region. This represents a heating device located at the upper wall of the room; it is controlled via the S1 sensor and has an initial value of 28°C . The definition in both rows models the "thickness" of the radiator — the depth of the heat influence zone.

The second boundary condition is the window region, located on the left side of the grid, from row 6 to row 22. The window models heat loss from cold air and is controlled via the S2 sensor (initial value 16°C). Column 0 (grid edge) is set to the S2 value, while column 1 is set to $S2 + 1.5^{\circ}\text{C}$ — this physically accurately represents the decrease in cold influence as one moves slightly away from the window. The third boundary condition is the door region, located at the lower edge of the grid, from column 19 to column 30 of row 29. The door represents the ventilation connection with the external environment; its temperature value is fixed at 17°C. Additionally, the right wall (column 49 of the grid) is set to 18°C for all rows — this models an insulated exterior wall. The third sensor S3 is located at the center of the room, at the intersection of rows 17–19 and columns 34–36, and can represent an air conditioner, additional heater, or other central device. The `applyBoundaries()` function is called after each Euler iteration, ensuring that the boundary conditions do not "dissolve away".

Visualization of the Temperature Field: Color Gradient and Canvas API

A multi-stage `getColor(temp)` function was developed to convey simulation results to the user in an understandable manner by mapping temperature values to a color gradient. The function normalizes the range from 15°C to 30°C to a value between 0 and 1 (in practice the normalization is applied over the interval from 14°C to 18°C: $\text{ratio} = (\text{temp} - 14) / 18$), then applies a four-stage color interpolation. In the range 0.0–0.25 (approximately 15–18.5°C), a transition from blue to cyan (Blue → Cyan) is performed: $r=0$, $g: 0 \rightarrow 180$, $b=255$. In the range 0.25–0.5 (18.5–22°C), a transition from cyan to green (Cyan → Green) occurs. In the range 0.5–0.75 (22–26°C), a transition from green to yellow (Green → Yellow) is observed. Finally, in the range 0.75–1.0 (26–30°C), a transition from yellow to red (Yellow → Red) takes place. This gradient corresponds to the standard "blue-green-yellow-red" palette used in thermographic devices and visually represents temperature differences with clarity.

The `draw()` function utilizes the direct pixel rendering capability of the Canvas API. At the beginning of each frame rendering, the canvas is cleared via `ctx.clearRect()`. Then the room image (`room.png`) is drawn using `ctx.drawImage()` with a specified opacity — this allows the user to overlay the heat map on top of the room interior plan. After that, for each of the 30×50 cells, the color is determined via `getColor()` and filled using `ctx.fillRect()`; the heat map is drawn with `heatOpacity` (initial 45%) transparency. On top of this, thin grid lines (0.03 opacity), three arrows directed downward from the radiator, and a colored circle and temperature label for each of the three sensors are drawn. Sensor coordinates are converted to pixels as follows: $\text{px} = \text{sensor.col} \times \text{cellW}$, $\text{py} = \text{sensor.row} \times \text{cellH}$. As a result, the user can simultaneously observe the room interior image and the heat map together, with smoothly rendered sensors and directional arrows.

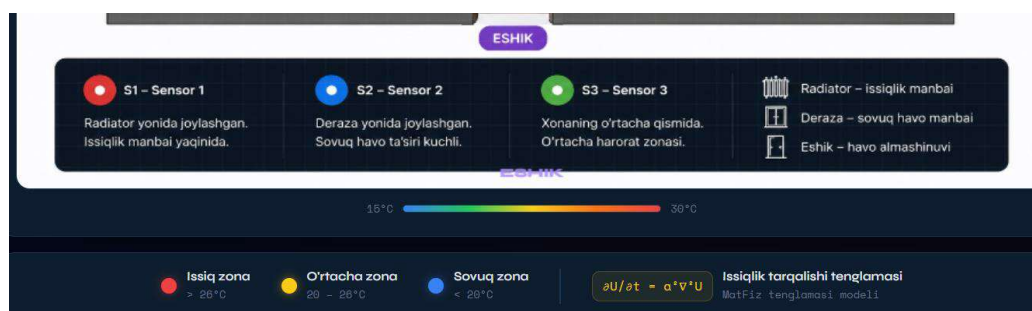


Figure 5. Visual representation of the temperature-color gradient

Stochastic Noise Effect and Simulation Realism. In a real room, the temperature field is never perfectly smooth. Minor turbulences in the air, measurement errors of sensors, uneven wall insulation — all of these lead to small, irregular variations in the temperature distribution. If the simulation were to operate solely based on the pure mathematical equation, the temperature field would become absolutely smooth and orderly over time, diverging from the real physical environment. The `addNoise()` function was added to the project from this perspective of physical accuracy and aesthetic realism. This function is called once per animation frame and adds a random deviation in the range of $\pm 0.04^{\circ}\text{C}$ to each value in the internal cells. The deviation formula is as follows: `grid[r][c] += (Math.random() - 0.5) * 0.08`. Here, `Math.random()` returns a uniformly distributed random number from 0 to 1; subtracting 0.5 produces a range from -0.5 to +0.5, and multiplying by 0.08 yields a small noise value from -0.04 to +0.04. This amplitude is very small relative to the temperature differences observed in heat diffusion — imperceptible in practice — but from a visualization standpoint, it ensures that the colors continuously update, giving a "living" appearance. An important technical point: noise is added only to the internal cells — not to the boundary cells, as those are reset in the next iteration via `applyBoundaries()`.

Interactive User Interface and Control System. The project's user interface is implemented in the dark-mode aesthetic characteristic of modern web applications. The left control panel contains three sensor cards, each consisting of a color indicator, sensor name, and description, along with a numerical input field for changing the temperature value. When the "Update Sensors" button is pressed, the `initGrid()` function is called, the entire grid is reset to 20°C , and then `applyBoundaries()` applies the boundary conditions with the new values. The visibility sliders block contains two sliders: "Room Image" (0–100%) and "Heat Map" (0–100%); they control the `roomOpacity` and `heatOpacity` variables respectively. The user can completely hide the room image and view only the pure heat map, or vice versa. The statistics panel displays four values — average ($\emptyset$), maximum ($\blacktriangle$), minimum ($\blacktriangledown$), and difference ($\triangle$) — in real time; the values are computed in the `updateStats()` function by iterating over all 1500 cells. The start/stop simulation button operates in toggle mode: the first press starts an iteration every 400 ms via `setInterval(simulationStep, 400)`, and the second press stops it with `clearInterval()`. The project also adheres to responsive design principles: on screens smaller than 1200 px, the panel and canvas stack vertically, enabling comfortable use of the system on tablets and smartphones.

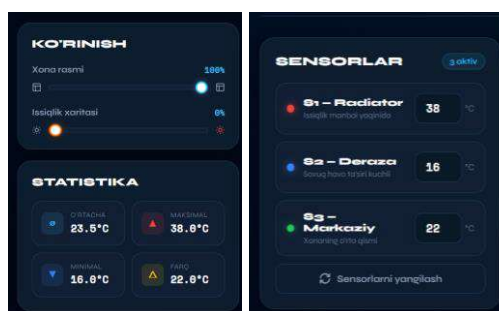


Figure 6. Left control panel of the project

Analysis of the Simulation Process and Results. When the project is launched, the `roomImage.onload` event is listened for — after the room image has fully loaded in the browser, the `start()` function is called automatically. Within `start()`, `initGrid()` sets the entire grid to an initial temperature of 20°C , and then `applyBoundaries()` applies the boundary conditions: the radiator region is set to the S1 value (28°C), the window region to the S2 value (16°C), the door region to 17°C , and the right wall to 18°C . After that, `diffusionStep(1)` is called once — this is a "pre-warm" step that

serves to show the user a temperature field that has already spread slightly when the simulation begins. When the simulation button is pressed, the simulationStep() function begins to be called every 400 ms. At each step, $10 \times 5 = 50$ Euler iterations are executed sequentially via diffusionStep(10), followed by small noise addition via addNoise(), canvas rendering via draw(), and statistics update via updateStats(). The physical phenomena observed during simulation confirm the correctness of the model. In the initial state (all internal cells at 20°C), a hot zone forms rapidly near the radiator and begins to spread downward and sideways; a cold zone forms near the window. After several dozen animation frames — that is, approximately 10–15 seconds — the temperature field achieves relative stability: a hot region of 26–28°C near the radiator, a cold region of 16–18°C near the window, and a gradually varying gradient in the rest of the room takes shape. Increasing the central sensor S3 value simulates a cooling device in place of an air conditioner, while decreasing it simulates an additional heater. In the statistics panel, the average temperature stabilizes at approximately 21–22°C in the steady state, with a difference of 12–13°C — reflecting the temperature gradient between the radiator and the window. The results fully correspond to the expected physical behavior of the heat equation for the given boundary conditions and diffusion coefficient, and the accuracy of the model is considered confirmed.

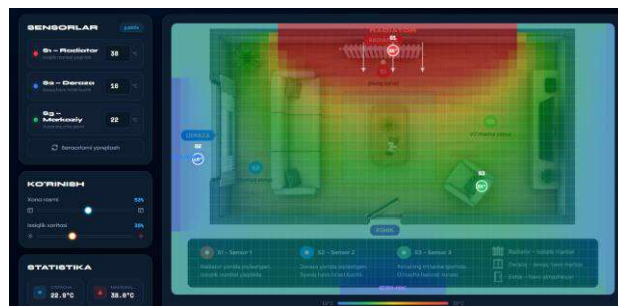


Figure 7. Time evolution of the temperature field simulation

Conclusion. This project and its scientific analysis demonstrate a number of important conclusions. First, it has been proven that the classical theoretical apparatus of mathematical physics — the heat diffusion equation and the Euler numerical integration method — can be transformed into a fully functional, interactive, and visually expressive software system using modern web technologies. Second, the correct choice of computational parameters — discrete grid size, time step, thermal coefficient and boundary conditions — directly affects the physical realism and numerical stability of the simulation; in the project, the 30×50 grid size provided a good balance between computational efficiency and visual accuracy. Third, the addition of a stochastic noise effect — even though its magnitude is very small — significantly increases the "liveliness" of the simulation and enhances its resemblance to real physical measurement data. Fourth, the responsive dark-mode interface, sensor cards, opacity sliders, and real-time statistics panel together made it possible for even an ordinary user to control a complex physical process. For future research, directions such as GPU parallel computations via WebGL (for larger grid sizes), incorporating convection effects and humidity into the model, as well as dynamically updating boundary conditions by receiving data from real IoT sensors via WebSocket are considered promising. The project has demonstrated an effective way to make mathematical modeling tools open and interactive for a wide audience in the modern world where smart home technologies continue to develop.

References

1. Fourier, J. B. J. (1822). *Théorie analytique de la chaleur*. Paris: Firmin Didot Père et Fils.
2. LeVeque, R. J. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations*. Philadelphia: SIAM.

3. Morton, K. W., & Mayers, D. F. (2005). *Numerical Solution of Partial Differential Equations*. Cambridge University Press.
4. Strang, G., & Fix, G. J. (1973). *An Analysis of the Finite Element Method*. Prentice-Hall.
5. Тихонов, А. Н., Самарский, А. А. (1977). *Уравнения математической физики*. Москва: Наука.
6. Incropera, F. P., & DeWitt, D. P. (2011). *Fundamentals of Heat and Mass Transfer*, 7th ed. Wiley.
7. Flanagan, D. (2020). *JavaScript: The Definitive Guide*, 7th ed. O'Reilly Media.
8. MDN Web Docs. (2024). *Canvas API*. https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API.
9. Cengel, Y. A. (2014). *Heat Transfer: A Practical Approach*, 3rd ed. McGraw-Hill.
10. Smith, G. D. (1985). *Numerical Solution of Partial Differential Equations: Finite Difference Methods*, 3rd ed. Oxford University Press.
11. Pozrikidis, C. (2011). *Introduction to Finite and Spectral Element Methods Using MATLAB*. CRC Press.
12. Thomas, J. W. (1995). *Numerical Partial Differential Equations: Finite Difference Methods*. Springer.
13. Samarskiy, A. A., Nikolaev, E. S. (1978). *Metody resheniya setochnyx uravneniy*. Moskva: Nauka.
14. ISO 7730:2005. *Ergonomics of the thermal environment — Analytical determination and interpretation of thermal comfort*. International Organization for Standardization.