Scientific Computing Exercise Set 1

Athish Prakalath V P (16162021) Pranav Bali (16076591) Tarreau Boone (13981412)

***Repository**: https://github.com/XthPB/SCI_COMP_set1*

I. Introduction

THIS report studies the use of numerical methods to solve two fundamental physical models: the vibrating string, governed by the wave equation (in a 1D domain) and diffusion processes, governed by the diffusion equation (in a 2D domain). These phenomena arise naturally across both science and engineering. Wave propagation appears in electrical transmission lines, compressible fluid flow in pipes, acoustics in gases and liquids, and optical wave propagation. Diffusion models the transport of heat, particles, and chemical concentration in physical and biological systems.

We present the governing models and the assumptions under which they hold. The continuous equations are discretised in space and time so they can be solved using computer programs. Finite difference schemes approximate derivatives on structured grids. Iterative solvers are then used for efficient computation. In particular, the Jacobi method, the Gauss–Seidel method, and Successive Over Relaxation (SOR) are implemented and compared, including parallelisation strategies for iteration loops.

II. Theory

A. Wave equation

1) Governing equation: The motion of a vibrating string is described by the one dimensional wave equation

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \tag{1}$$

where $u(x,t)$ denotes the vertical displacement of the string at position x and time t, and c represents the wave propagation speed. The spatial domain is $x \in [0, L]$ and the time domain is $t \geq 0$.

2) Finite difference approximation: The domain is discretised using a uniform grid,

$$x_i = i\Delta x, \quad t^k = k\Delta t, \tag{2}$$

where Δx and Δt are the spatial and temporal step sizes. Using Taylor series and central differences, substitution into the wave equation yields the explicit update rule

$$u_i^{k+1} = 2u_i^k - u_i^{k-1} + \left(\frac{c\Delta t}{\Delta x}\right)^2 \left(u_{i+1}^k - 2u_i^k + u_{i-1}^k\right). \tag{3}$$

The stability ratio is

$$r = \frac{c\Delta t}{\Delta x}. \tag{4}$$

B. Diffusion equation

1) Governing equation: Diffusion describes the transport of heat or concentration in space. The diffusion equation in 2D is

$$\frac{\partial c}{\partial t} = D\nabla^2 c, \tag{5}$$

where $c(x,y,t)$ denotes concentration, D is the diffusion coefficient, and ∇^2 is the Laplacian. The computational domain is $(x,y) \in [0,1] \times [0,1]$.

2) Finite difference approximation: Using a uniform grid

$$x_i = i\Delta x, \quad y_j = j\Delta x, \quad t^k = k\Delta t, \tag{6}$$

the Laplacian is approximated by the five point stencil,

$$\nabla^2 c \approx \frac{c_{i+1,j}^k + c_{i-1,j}^k + c_{i,j+1}^k + c_{i,j-1}^k - 4c_{i,j}^k}{\Delta x^2}. \tag{7}$$

Forward difference in time gives

$$\frac{\partial c}{\partial t} \approx \frac{c_{i,j}^{k+1} - c_{i,j}^k}{\Delta t}. \tag{8}$$

Substitution yields the explicit update rule

$$\begin{aligned} c_{i,j}^{k+1} = c_{i,j}^k + \frac{D\Delta t}{\Delta x^2}\Big(&c_{i+1,j}^k + c_{i-1,j}^k \\ &+ c_{i,j+1}^k + c_{i,j-1}^k - 4c_{i,j}^k\Big). \end{aligned} \tag{9}$$

A sufficient stability condition is

$$\frac{4D\Delta t}{\Delta x^2} \leq 1. \tag{10}$$

3) Boundary and initial conditions: Fixed concentration values are imposed on the top and bottom boundaries:

$$c(x, y=1, t) = 1, \tag{11}$$

$$c(x, y=0, t) = 0. \tag{12}$$

Periodicity is enforced horizontally:

$$c(x=0, y, t) = c(x=1, y, t). \tag{13}$$

The initial condition is

$$c(x, y, 0) = 0 \quad \text{for } 0 < y < 1. \tag{14}$$

[1]

III. METHODS

A. Implementation

All simulations were implemented in **Python**. **NumPy** was used for numerical arrays and vector operations, and **Matplotlib** was used for visualisation and animation. For iterative solvers, **Numba** was used to accelerate iteration loops where parallel execution improved runtime.

B. Exercise 1.1: 1D Wave Equation

1) Numerical setup: The spatial domain $x \in [0, 1]$ is discretised using a uniform grid with spacing Δx. Time is advanced with a constant step size Δt until time T. The endpoints are held at zero for all t. Note that the first step was computed using the initial displacement and zero initial velocity. After this initial step, the standard time stepping update is applied for all subsequent time levels.

1) Pseudo code (1D wave equation):

```
Choose N, dx, dt, wave speed c, final time T
Initialise u_prev[i] = u(x_i, 0) for i = 0..N
Compute u_curr[i] at t = dt using zero initial
    velocity

for n = 1 .. n_steps-1:
    for i = 1 .. N-1:
        u_next[i] = 2*u_curr[i] - u_prev[i]
                  + (c*dt/dx)^2 * (u_curr[i+1]
    - 2*u_curr[i] + u_curr[i-1])

    u_next[0] = 0
    u_next[N] = 0

    u_prev = u_curr
    u_curr = u_next
```

C. Exercise 1.2: Concentration Diffusion

1) Numerical setup: The diffusion equation was solved on the unit square using a structured uniform grid. Forward difference was used for time updates and central differences for spatial updates. Spatial derivatives use the five point stencil at each interior grid node. Fixed values are applied on the top and bottom boundaries, while periodicity is enforced horizontally through index wrapping.

1) Pseudo code (explicit diffusion update):

```
Initialise concentration field c[i,j]
Apply fixed top and bottom boundary values

for each time step:
    for i = 0 .. N:
        ip = (i+1) mod N
        im = (i-1) mod N

        for j = 1 .. N-1:
            c_new[i,j] = c[i,j] + D*dt/dx^2 * (
                c[ip,j] + c[im,j] +
                c[i,j+1] + c[i,j-1] -
                4*c[i,j]
            )

    reapply fixed top and bottom boundary values
    copy c_new -> c
```

D. Exercise 1.3: Steady diffusion and relaxation methods

1) Parallelisation strategy: The grid is decomposed into subdomains distributed across multiple cores. After each sweep, neighbouring subdomains exchange one layer of boundary nodes so that stencil updates near subdomain boundaries use consistent neighbour values.

2) Iterative solvers: Gauss–Seidel uses in place updates so newly computed values are reused within the same sweep. Convergence is monitored using the maximum change between successive iterations.

1) Pseudo code (Jacobi):

```
Initialise c and boundary values

repeat until convergence:
    for each interior (i,j):
        c_new[i,j] = 0.25 * (
            c[i+1,j] + c[i-1,j] +
            c[i,j+1] + c[i,j-1]
        )
    copy c_new -> c
```

2) Pseudo code (Gauss–Seidel):

```
Initialise c and boundary values

repeat until convergence:
    for each interior (i,j):
        c[i,j] = 0.25 * (
            c[i+1,j] + c[i-1,j] +
            c[i,j+1] + c[i,j-1]
        )
```

3) Pseudo code (SOR):

```
Choose relaxation factor w

repeat until convergence:
    for each interior (i,j):
        gs = 0.25 * (
            c[i+1,j] + c[i-1,j] +
            c[i,j+1] + c[i,j-1]
        )
        c[i,j] = (1-w)*c[i,j] + w*gs
```

4) Pseudo code (SOR with sink objects):

Grid points inside a sink object have their concentration fixed to zero; the SOR update is skipped at these points.

```
Build mask[i,j]: 0 = fluid, 1 = sink

repeat until convergence:
    for each interior (i,j):
        if mask[i,j] == 1:
            c[i,j] = 0
            continue

        gs =
    0.25*(c[i+1,j]+c[i-1,j]+c[i,j+1]+c[i,j-1])
        c[i,j] = (1-w)*c[i,j] + w*gs
```

5) Pseudo code (SOR with insulating objects):

Insulating points are skipped. If a neighbour lies inside the object, it is replaced by the current point value, enforcing zero normal flux.

```
Build mask[i,j]: 0 = fluid, 2 = insulating

repeat until convergence:
    for each interior (i,j):
        if mask[i,j] == 2:
            continue

        for each neighbour nb of (i,j):
            if mask[nb] == 2:
                use c[i,j] instead of c[nb]

        gs = 0.25*(c_e + c_w + c_n + c_s)
        c[i,j] = (1-w)*c[i,j] + w*gs
```

IV. Results

A. Wave equation

The simulation was performed for three different initial conditions with the parameters shown in Table I.

Parameter	Value
L	1
N	200
Δx	0.005
Δt	0.001
c	1
T	2

TABLE I: Wave equation parameters.

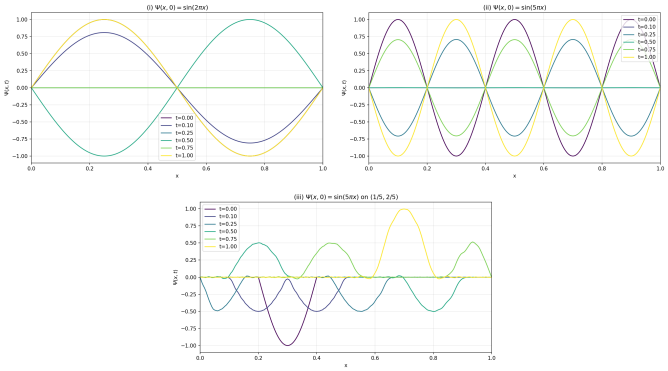

Fig. 1: Wave equation solutions for: 1) $u(x,0) = \sin(2\pi x)$; 2) $u(x,0) = \sin(5\pi x)$; 3) $u(x,0) = \sin(5\pi x)$ for $\frac{1}{5} < x < \frac{2}{5}$ (and 0 elsewhere).

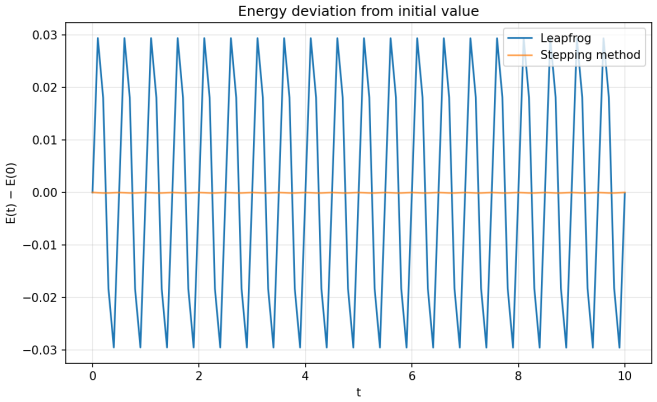

Fig. 2: Energy deviation from the initial value for the wave-equation solvers

The ratio: $r = \dfrac{c\Delta t}{\Delta x} = \dfrac{1 \times 0.001}{0.005} = 0.2 \leq 1.$ so the problem is stable for the setup.

B. Diffusion

The simulation was executed on the 2D unit square with the parameters in Table II.

Parameter	Value
Domain	$[0,1] \times [0,1]$
N	50
D	1
ε	10^{-5}
ω	1.8

TABLE II: Diffusion parameters.

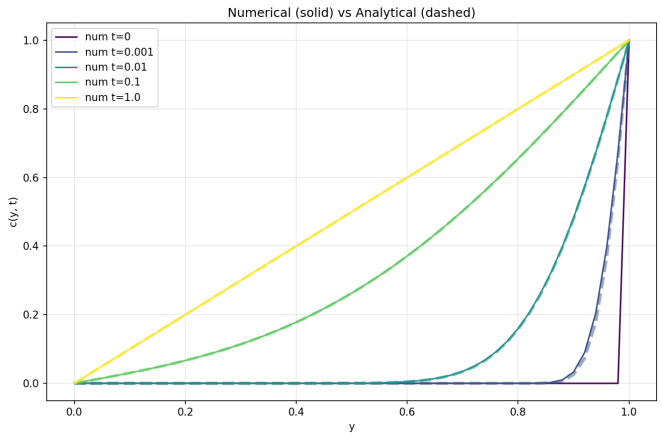

Fig. 3: Analytical vs numerical solution

Error compared to the transient analytical solution in [2] is small and decays with time in Fig. 3: the RMSE decreases from 6.54×10^{-3} at $t = 10^{-3}$ to 5.19×10^{-8} at $t = 1$.

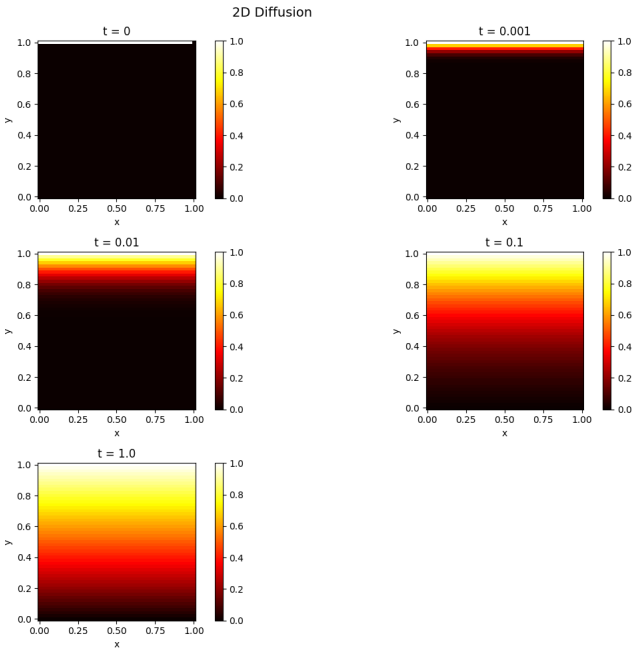

Fig. 4: Time evolution of diffusion field

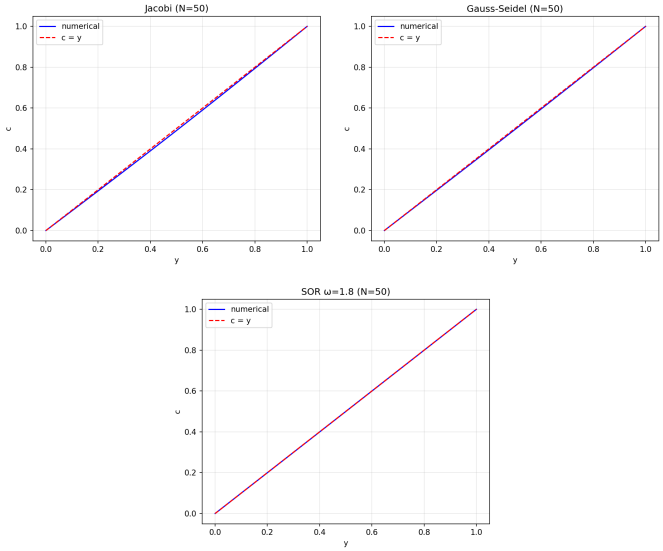

Fig. 5: Steady state comparison of iterative solvers

At steady state, the max error is lowest for SOR in Fig. 5 (5.33×10^{-4}), followed by Gauss–Seidel (5.06×10^{-3}) and Jacobi (1.01×10^{-2}).

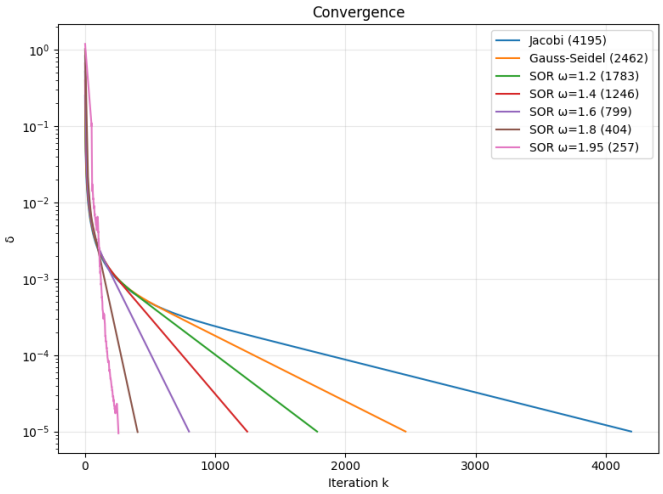

Fig. 6: Convergence of iterative solvers

The convergence-rate ordering matches the error ordering in Fig. 6: SOR converges fastest (404 iterations), Gauss–Seidel is intermediate (2462), and Jacobi is slowest (4195).

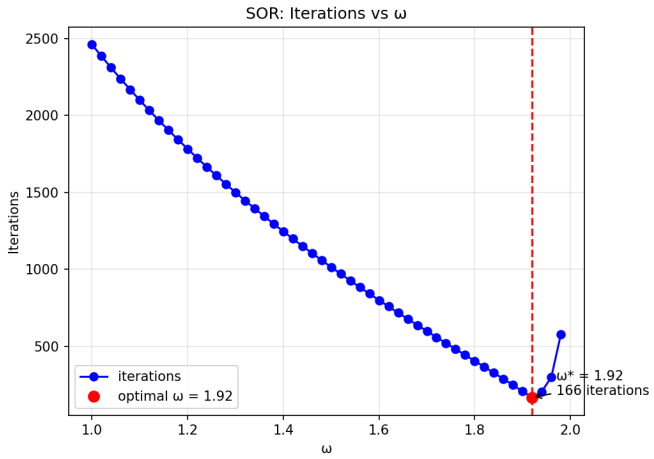

Fig. 7: Optimal relaxation parameter

For $N = 50$, the optimal relaxation is $\omega^\star \approx 1.92$ (Fig. 7), which gives the fastest SOR convergence. Across $N = 10$ to 100 (Fig. 8), $\omega^\star$ increases from about 1.66 to 1.96, showing an asymptotic trend toward 2 as grid resolution increases.

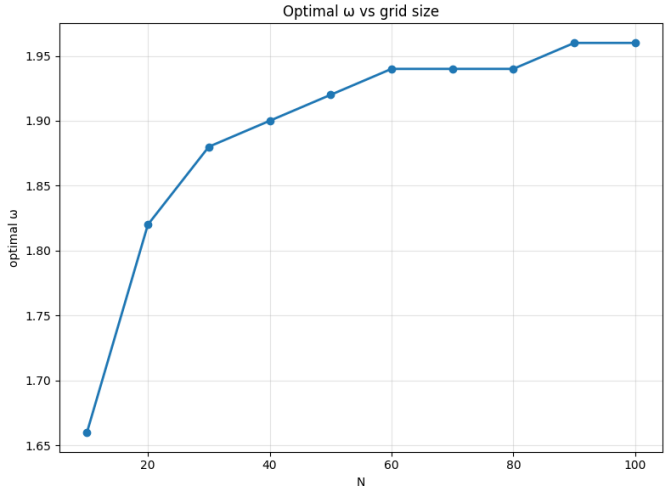

Fig. 8: Optimal relaxation vs grid size

C. Diffusion with objects

We now analyze the effects of placing sink and insulating-type objects in the flow field. We compare how these two object types modify the concentration field, convergence behavior and the optimal relaxation parameter.

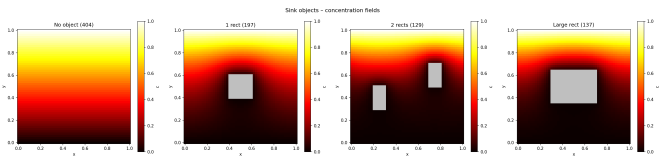

Fig. 9: Diffusion with sink objects

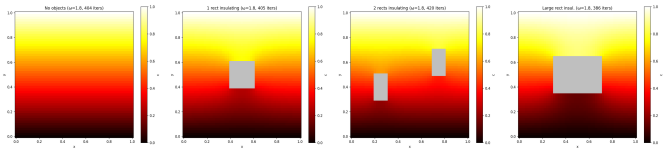

Fig. 10: Diffusion with insulating objects

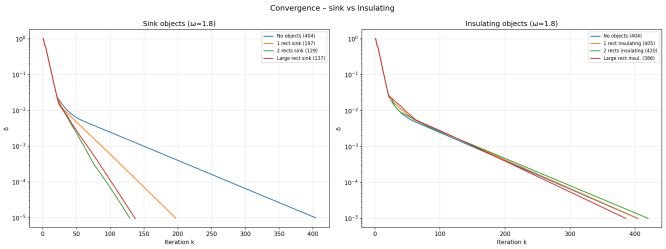

Fig. 11: Convergence with sink and Insulating objects

Sink objects accelerate convergence relative to the no-object case in Fig. 11: iterations drop from 404 (no objects) to 197 (1 sink), 129 (2 sinks), and 137 (large sink), because fixed-zero regions reduce the effective fluid domain.

Insulating objects change convergence much less in Fig. 11: 404 (no objects), 405 (1 insulator), 420 (2 insulators), and 386 (large insulator). Compared with sinks, insulators generally keep convergence close to the baseline and can even slow it slightly for fragmented geometries.

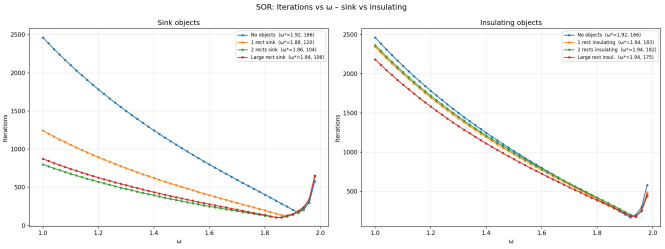

Fig. 12: Optimal ω with sink and insulating objects

Relative to the baseline no-object case in Fig. 12 ($\omega^\star = 1.92$, 166 iterations), insulating objects shift the optimum slightly upward to $\omega^\star \approx 1.94$ and increase iteration counts to 175–183.

For sink objects, the trend is opposite in Fig. 12: compared to the same baseline ($\omega^\star = 1.92$, 166), the optimum shifts downward to $\omega^\star \approx 1.84$–$1.88$ and the best iteration counts decrease to 104–120.

V. Discussion

A. Wave Equation (Fig. 1)

The three initial conditions produce qualitatively different wave dynamics. When sinusoidal initial conditions are used, the solutions stay smooth and oscillatory, consistent with a non-dissipative wave equation with fixed endpoints (Fig. 1). These profiles resemble the eigenmodes of the system. On the other

hand, local initial displacement excites a broader range of spatial modes, leading to more complex waves (Fig. 1). Simulations are stable for all cases because the Courant ratio $r = c\,\Delta t/\Delta x$ satisfies $r < 1$. If $r > 1$, this explicit scheme becomes unstable. Even when stable, higher-frequency content can exhibit greater phase error on a finite grid.

1) Leapfrog method (Fig. 2): The Leapfrog method was implemented as a symplectic time integrator for the 1D wave equation. It staggers velocity and displacement in time, updating velocity at half-steps and displacement at full steps. The energy of the system was monitored and compared with the standard time-stepping method. The Leapfrog method shows bounded oscillations in the total energy error over long times, indicating that it does not introduce systematic energy gain or loss. This behavior is characteristic of symplectic integrators, which preserve the phase-space structure of Hamiltonian systems. In contrast, non-symplectic schemes generally exhibit a slow drift in energy as the simulation progresses. Therefore, the Leapfrog method provides improved long-term stability and accuracy for oscillatory systems such as the vibrating string.

B. Time-dependent Diffusion (Figs. 3, 4)

For the time-dependent diffusion, the RMSE decreases strongly over time (Fig. 3). The field approaches a near linear vertical profile. For periodic boundaries in x and Dirichlet conditions $c(x, 0, t) = 0$, $c(x, 1, t) = 1$ the steady state is $c(y) = y$. The diffusion progressively smooths out original gradients and removes x-dependence. The explicit scheme satisfies the diffusion stability condition so the solutions remains stable and the error drops as the field becomes smoother and approaches a steady state.

C. Steady Diffusion Solvers (Figs. 5, 6, 7, 8)

SOR converges with less iterations than Gauss-Seidel and Jacobi in the steady diffusion problems. The optimal ω increases with grid size N, approaching $\to 2$. This aligns with the idea that over-relaxation increases the convergence speed by damping error modes more effectively. The measured ω values lie in the expected $\approx 1.7 - 2.0$ range and are dependent on N.

D. Objects: sink vs insulating (Figs. 9, 10, 11, 12)

For the object experiments (Fig. 11), adding sink object decreases the number of SOR iterations substantially, while insulating objects cause only a small change in the convergence rate. The difference in convergence rate is expected: sinks 'fix' the solution by imposing fixed values on a subset of grid points. This reduces the degrees of freedom and alters the dominant error modes, which makes the system easier for SOR to solve. On the other hand insulating objects enforce a zero-flux condition without pinning the concentration itself. This leaves the solution freer to adjust and convergence properties change less.
Consistent with this, the optimal relaxation parameter ω shifts downwards for sinks and shifts slightly upward for insulation objects (Fig. 12). The concentration fields also show these mechanisms, sinks locally lower the the concentration fields and increase gradients around object boundaries, while insulators redirect diffusion paths around the objects without significantly lowering the concentration in the domain (Figs. 9, 10).

E. Future works

Objects placed in the concentration field are effectively locked to the grid points. This introduces inaccurate simulation at object boundaries, especially when the object is small or is placed slightly off-grid. To overcome these challenges, we propose using sub-grid techniques like adaptive mesh refinement and avoid refining the grid excessively.

VI. Conclusion

In conclusion, we solved the 1D wave equation and the 2D diffusion equation using finite-difference schemes and iterative relaxation methods. The explicit update rule for the wave equation stayed stable for $r = 0.2$, and different initial conditions produced the expected dynamics. For diffusion, the transient solution relaxed toward the steady profile $c(y) = y$ and the RMSE decayed strongly over time. For the steady diffusion problem, SOR converged faster than Gauss-Seidel and Jacobi and the optimal relaxation parameter increased with grid size. Finally, introducing objects showed that sinks reduce the iteration counts and shift ω downwards, while insulating objects mainly redirect flux and have a significantly smaller effect on convergence.

References

[1] K. A. Novak, *Numerical Methods for Scientific Computing: The Definitive Manual for Math Geeks*, 2nd ed. Equal Share Press, 2022.
[2] G. Závodszky, "Scientific Computing — Assignments (Set 1)," Course handout (PDF), 2025, provided via the course platform.