Scientific Computing Exercise Set 1

Thomas Norton (15295273) & Balthazar Dupuy d'Angeac (15852431) & Ruben Lanjouw (12415049)

***Repository**: https://github.com/RSlanjouw/Scientific-Computing*

INTRODUCTION

AS we search to model physical and natural systems numerically, we must confront the problem of modeling continuity, whether in motion, diffusion, stress, or time [3]. Real continuity implies continuous change (not excluding stability, that is, $\delta = 0$), which itself implies differentiation. Our technology is unable to simulate continuity as it exists in the physical world but can approximate it with degrees of precision. One approach is to approximate continuous change in time by discretizing it; simulating time and its influence on a system through numerous small time steps. The simulation grows more accurate for smaller steps. Although this intuitive time-dependent approach may be more accurate at some stages of the system, at critical states - specifically stable ones - iterative methods are preferable. These do not require time simulation, they aim to approach the real mathematical/theoretical state of the system at a critical point by self-updating; for each *iteration*, getting closer to the theoretical state. This paper will introduce time-dependent discretization over the changes in a natural system: wave motions of a vibrating string. We will investigate how this approach compares to iterative methods - Jacobi, Gauss-Seidel, and Successive over Relaxation - over the stability point of heat diffusion over a finite 2-dimensional space. We will compare the precision allowed by each method in simulating a system reaching a stable status; discussing the ineffectiveness of time-dependent discretization around said status, the ability of the Jacobi and the *faster* Gauss-Seidel methods to reach high levels of precision and the notable capacity of the SOR method in reaching a result limited in precision, faster than other methods by a significant margin.

I. METHOD AND THEORY

A. Time-Dependent Discretization - Vibrating String

We first implement time-dependent discretization to simulate the continuous motion of a vibrating string. From a theoretical equation, we derive a discretized one, which we can integrate numerically.

The Wave Equation:

By abstraction of external influences like *gravity*, *drag from air*, etc, a natural system like a **string in motion** can be theoretically defined by the equation [1]:

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

Where $\Psi(x,t)$ defines the status of the string with respect to time (t) and space (x) over its length L (in this case -over a 2D plane).

The squared constant c relates to the speed of propagation of the wave, which itself doesn't concern us in the context of this research but remains relevant to the stability of our discretized formula.

Discrete Time-Dependent Equation:

The wave equation [1] relies on derivatives over both time and space, these can be approximated as:
„$\frac{df}{dx} = \frac{f(x+\Delta x)-f(x)}{\Delta x} = \frac{f(x)-f(x-\Delta x)}{\Delta x}$„
for a **very** (∞) small step Δx. For our equation [1] we may use Δt and Δx as time and spatial steps. We can then use the above approximation to eliminate derivatives from [1]; we rewrite the 2nd-order derivatives as:

- $\frac{\partial^2 \Psi}{\partial t^2}(x_n, t_j) = \frac{\frac{d\Psi}{dt}(x_n, t_{j+1}) - \frac{d\Psi}{dt}(x_n, t_j)}{\Delta t}$
  with $\Delta t = t_{j+1} - t_j = t_j - t_{j-1}$
- $\frac{\partial^2 \Psi}{\partial x^2}(x_n, t_j) = \frac{\frac{d\Psi}{dx}(x_{n+1}, t_j) - \frac{d\Psi}{dx}(x_n, t_j)}{\Delta x}$
  with $\Delta x = x_{n+1} - x_n = x_n - x_{n-1}$

Likewise, we rewrite the above 1st-order derivatives as:

- $\frac{d\Psi}{dt}(x_n, t_j) = \frac{\Psi(x_n, t_j) - \Psi(x_n, t_{j-1})}{\Delta t}$ and
  $\frac{d\Psi}{dt}(x_n, t_{j+1}) = \frac{\Psi(x_n, t_{j+1}) - \Psi(x_n, t_j)}{\Delta t}$
- $\frac{d\Psi}{dx}(x_n, t_j) = \frac{\Psi(x_n, t_j) - \Psi(x_{n-1}, t_j)}{\Delta x}$ and
  $\frac{d\Psi}{dx}(x_{n+1}, t_j) = \frac{\Psi(x_{n+1}, t_j) - \Psi(x_n, t_j)}{\Delta x}$

Replacing these **approximations** in the initial equation [1] we get the discretized equation:

$$\begin{aligned}\Psi(x_n, t_{j+1}) &= 2\Psi(x_n, t_j) - \Psi(x_n, t_{j-1}) \\ &+ \left(\frac{c\Delta t}{\Delta x}\right)^2 [\Psi(x_{n+1}, t_j) - 2\Psi(x_n, t_j) + \Psi(x_{n-1}, t_j)]\end{aligned} \tag{2}$$

Note. This equation relies on $\Psi(x_n, t_{j-1})$ i.e we must know the status of $\Psi(x_n)$ at time step t_{j-1}, we circumvent this issue at t_0 by setting the condition
$\Psi'(x, t = t_0 = 0) = 0 \implies \Psi(x, t_0) = \Psi(x, t_1)\ \forall x$.

The discretized equation [2] serves as an approximation of the wave equation [1] which can be numerically integrated. We used it to simulate wave motion over a string of arbitrary length $L = 1$ using Python. To run our simulation we must accept an initial state at $t = 0$, which will induce a subsequent wave. We used three:

- Initial String Status: $\Psi(x, t = 0) = \sin 2\pi x$
- Initial String Status: $\Psi(x, t = 0) = \sin 5\pi x$
- Initial String Status: $\Psi(x, t = 0) = \sin 5\pi x$ if $1/5 < x < 2/5$, else $\Psi = 0$

B. Time-Independent Diffusion Equation

1) The Diffusion Equation: With time-dependent discretization presented, we now introduced the case of *Heat Diffusion* over a 2-dimensional plane, which can also be theoretically defined by the equation :

$$\frac{\partial c}{\partial t} = D\nabla^2 c \tag{3}$$

Where $c(x, y; t)$ is the (*heat*) concentration of point (x, y) at time t, and the constant D (here $D = 1$) is the *diffusion coefficient*, which quantifies the speed of the diffusion (the speed itself is not relevant to our implementation outside of its numerical stability, which $D = 1$ maintains).

2) Time-Dependent Discretization of Diffusion Equation: We may discretized [3] over time:

$$c_{i,j}^{k+1} = c_{i,j}^k + \frac{dtD}{dx^2}(c_{i+1,j}^k + c_{i-1,j}^k + c_{i,j+1}^k + c_{i,j-1}^k - 4c_{i,j}^k) \tag{4}$$

where $i, j \in \{0, 1, ..., N\}$ are indices of x and y respectively, k is an index for t.

Note. This discretized equation assumes that the spatial change δx of a specific point $c(x_n)$ depends on itself and its four closest neighbors (five-point stencil) only.

We simulate the resulting diffusion over a finite domain: an $N \times N$ grid with periodic west & east boundaries $\implies c(x_0, y; t) = c(x_N, y; t)\ \forall\ y$ and t

We implement this by using alternative versions of [2] when $i = 0$ or N:

- $c_{0,j}^{k+1} = c_{0,j}^k + \frac{dtD}{dx^2}(c_{1,j}^k + c_{N-1,j}^k + c_{0,j+1}^k + c_{0,j-1}^k - 4c_{0,j}^k)$
- $c_{N,j}^{k+1} = c_{N,j}^k + \frac{dtD}{dx^2}(c_{1,j}^k + c_{N-1,j}^k + c_{N,j+1}^k + c_{N,j-1}^k - 4c_{N,j}^k)$

Note. This assures that the left- and right-most boundaries remain equal.

Finally, we use the initial status where $c(x, y_N; t) = 1$ and $c(x, y_j; t) = 0$ for $j \in \{0, 1, ..., N-1\}$, $\forall x$ and t.

3) Time-Independent Diffusion Equation: The above implementation converges toward a stable state. So, theoretically, enough time steps Δt should induce us a state where $c_{i,j}^{k+1} = c_{i,j}^k$. But there are more effective ways to simulate our heat diffusion system at this state: we can implement an iteration-based method that takes us closer to the *unavoidable* state $c_{i,j}^{k+1} = c_{i,j}^k$, using the Laplace equation, which defines the system at stability:

$$\nabla^2 c = 0 \tag{5}$$

This equation is independent of time constraints as it only assures that the diffusion is stable across time and space. From The Laplace equation [5] we derive an iterable equation in the same way we did in Section A [I-A] over the *wave equation* [2], using the same approximation of derivatives, "$\frac{df}{dx} = \frac{f(x+\Delta x)-f(x)}{\Delta x} = \frac{f(x)-f(x-\Delta x)}{\Delta x}$", we get:

$$\nabla^2 c = \frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2} = 0$$

$$\frac{\frac{dc_{i+1,j}}{dx} - \frac{dc_{i,j}}{dx}}{\Delta x} + \frac{\frac{dc_{i,j+1}}{dy} - \frac{dc_{i,j}}{dy}}{\Delta y} = 0$$

$$\frac{\frac{c_{i+1,j}-c_{i,j}}{\Delta x} - \frac{c_{i,j}-c_{i-1,j}}{\Delta x}}{\Delta x} + \frac{\frac{c_{i,j+1}-c_{i,j}}{\Delta y} - \frac{c_{i,j}-c_{i,j-1}}{\Delta y}}{\Delta y} = 0$$

Our implementation is on a square $N \times N$ grid with steps $\Delta x = \Delta y\ (= \frac{1}{N})$, thus:

$$(c_{i+1,j} - c_{i,j}) - (c_{i,j} - c_{i-1,j}) + (c_{i,j+1} - c_{i,j}) - (c_{i,j} - c_{i,j-1}) = 0 \times (\Delta x)^2$$

$$\boxed{c_{i,j} = \frac{1}{4}(c_{i+1,j} + c_{i-1,j} + c_{i,j+1} + c_{i,j-1})} \tag{6}$$

Note. This equation can be produced from [4] by ignoring index k of time and setting $D \times dt$ as one fourth of dx^2, since time steps are obsolete, thus removing $c_{i,j}$ from the right part.

[6] describes our desired *stable* status at $c_{i,j}\ \forall\ i, j$. We will explore how our model may approach this status efficiently.

C. Jacobian Iteration

We want to update all grid points $c_{i,j}$ to respect the condition [6] but since updating $c_{i,j}$ depends on its four neighbors, we cannot update all points simultaneously. Therefore, we rely on an iterative method: **Jacobi Iteration**. Essentially, from an already fully computed grid

at an iteration k, we update each point $c_{i,j}^k$ to $c_{i,j}^{k+1}$ successively, where

$$c_{i,j}^{k+1} = \frac{1}{4}(c_{i+1,j}^k + c_{i-1,j}^k + c_{i,j+1}^k + c_{i,j-1}^k) \quad (7)$$

The Jacobi iteration [7] ensures that the system approaches stability with each iteration: denote the error between $c_{i,j}^k$ and the theoretical solution $c_{i,j}$ of the time-independent equation [6] by ε, from [7] :

$$c_{i,j}^{k+1} = c_{i,j} + \varepsilon^{k+1} = \frac{1}{4}(c_{i+1,j} + \varepsilon_1^k + c_{i-1,j} + \varepsilon_2^k + c_{i,j+1} + \varepsilon_3^k + c_{i,j-1} + \varepsilon_4^k)$$

$$\implies \varepsilon^{k+1} = \frac{1}{4}(\varepsilon_1^k + \varepsilon_2^k + \varepsilon_3^k + \varepsilon_4^k)$$

$$\implies |\varepsilon^{k+1}| \leq mean_{neighbor}(|\varepsilon^k|)$$

The error at $k+1$ will not be greater than the average neighboring error at k, it follows that the error non-strictly decreases with each iteration.

Note. The true theoretical solution cannot be reached (without truncation error) with Jacobi iteration. Suppose the system is stable since iteration k, than iteration k updated at least one point $c_{i,j}$, if so than all its neighbors require an update at iteration $k+1$.

D. Gauss-Seidel Iteration

The **Gauss-Seidel Iteration** builds on the same idea as the Jacobi: approach stability [6] with each iteration reducing the error ε. It does so more efficiently though: while Jacobi updates the concentration $c_{i,j}$ for all point (i,j) subsequently within a single iteration k, based on the neighboring values at $k-1$, Gauss-Seidel updates $c_{i,j}$ based on the most recently computed neighboring values. So already-updated values will influence the update of their non-updated neighbors. Since we update from top to bottom, left to right, the Gauss-Seidel method induces the equation:

$$c_{i,j}^{k+1} = \frac{1}{4}(c_{i+1,j}^k + c_{i-1,j}^{k+!} + c_{i,j+1}^k + c_{i,j-1}^{k+1}) \quad (8)$$

As we showed in section [I-C], updating a value as the average of its neighbors insures that the error ε will not grow: $|\varepsilon^k + 1| \leq |\varepsilon^k|$, so $c_{i,j}^{k+1}$ is not further from the theoretical solution $c_{i,j}$ than $c_{i,j}^k$ for all (i,j). Therefore, we can expect the Gauss-Seidel iterative equation [8] to converge towards the theoretical solution [5] in fewer (or as many) iterations than the Jacobi equation [7] as it computes the average of 4 neighbors; two are the same as in the Jacobi method (not updated), two are 'better' (already updated) $\implies$ the error of the resulting $c_{i,j}^{k+1}$ should be smaller or equal with Gauss-Seidel method.

E. Successive over Relaxation

We can further improve on the Gauss-Seidel iteration method. Iterative equations for Jacobi [7] and Gauss-Seidel [8] do not *directly* weigh the previous value $c_{i,j}^k$ to compute $c_{i,j}^{k+1}$. We can achieve this by manipulating the time-independent equation [6] to *virtually* include some weight ω over $c_{i,j}^k$ in computing $c_{i,j}^{k+1}$:

$$\omega \times c_{i,j} = \frac{\omega}{4}(c_{i+1,j} + c_{i-1,j} + c_{i,j+1} + c_{i,j-1})$$

$$c_{i,j} - \omega c_{i,j} = c_{i,j} - \frac{\omega}{4}(c_{i+1,j} + c_{i-1,j} + c_{i,j+1} + c_{i,j-1})$$

$$c_{i,j}(1-\omega) + \frac{\omega}{4}(c_{i+1,j} + c_{i-1,j} + c_{i,j+1} + c_{i,j-1}) = c_{i,j}$$

We apply the Gauss-Seidel process to get:

$$\boxed{c_{i,j}^{k+1} = c_{i,j}^k(1-\omega) + \frac{\omega}{4}(c_{i+1,j}^k + c_{i-1,j}^{k+1} + c_{i,j+1}^k + c_{i,j-1}^{k+1})} \quad (9)$$

This updated iterative equation affords weight to the concentration of a point (i,j) at a previous step, ω is called a **relaxation factor**.

We do not want this relaxation factor to overshadow the desired weight on neighboring values, which is what allows for diffusion to happen. We want to ensure that the weight of $c_{i,j}^k$ is not inflated in calculating $c_{i,j}^{k+1}$, our equation [9] should be seen as *correcting* the former when producing the latter. We can limit $c_{i,j}^k$'s influence by assuring that $-1 < 1-\omega < 1 \implies 0 < \omega < 2$. Our research will include finding an optimal value for ω.

F. Objects and isolation material

Having implemented the above iteration methods, we will study how they perform in non-trivial initial conditions, as to properly assess each method's strengths and weaknesses.

We will simulate objects; sinkholes and insulated materials, in our initial 50×50 domain of heat diffusion, where at some locations (x,y) the concentration will be maintained throughout the entire run ($c = 0$ for sinkholes). The relative diffusion rate of the insulating material will be set at 0.95. These are both implemented inside the program we wrote as mask multipliers.

The objects simulated/studied in our research will be:

- A square from row 22 until 28 in columns 22 until 28. (S1, I1)
- A square from row 15 until 35 in columns 15 until 35. (S2, I2)
- Two squares in rows 10 until 15 and in columns 35 until 40. (S3, I3)

II. Results

We now present the results of our implementations of the time-dependent discretization in our **waving string model** as well as of our several iteration methods for approaching the stable state of our **heat diffusion model**. We will here display our models, findings, and observations as to how each method compared to one another.

A. Wave Visualization

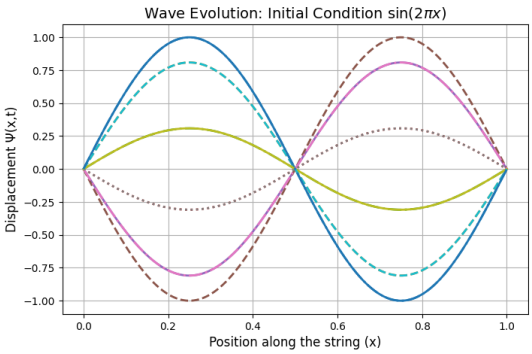

Fig. 1: Vibration of String of length $L = 1$ with Initial Condition $\Psi(x, t = 0) = \sin(2\pi x)$ over time

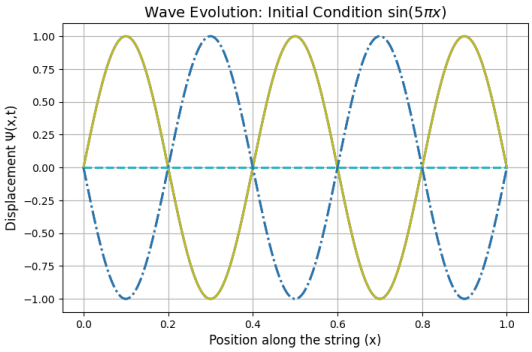

Fig. 2: Vibration of String of length $L = 1$ with Initial Condition $\Psi(x, t = 0) = \sin(5\pi x)$ over time

While we initially expected to see a wave move from left to right in a run of our models 1 and 2, we actually saw oscillations in the initial 2/5 curvatures, where parabolas remained equal (if sometimes, opposite to one-another). We observed the parabolas stretch but not *"move"* horizontally. This is because the initial axis points x_i where $Psi(x_i, t = 0) = 0$ -the null points from the initial states- remains null throughout the run. This is due to the fact that at a null point x_i, it holds that $x_{i-1} = -x_{i+1}$. Applying the discretized equation [2], we get :

$\Psi(x_i, t_{j+1}) = 2\Psi(x_i, t_j) - \Psi(x_i, t_{j-1}) + (\frac{c\Delta t}{\Delta x})^2[-2\Psi(x_i, t_j)]$

This shows that a *null* point will remain a null point in these conditions: where their neighboring values cancel each other out.

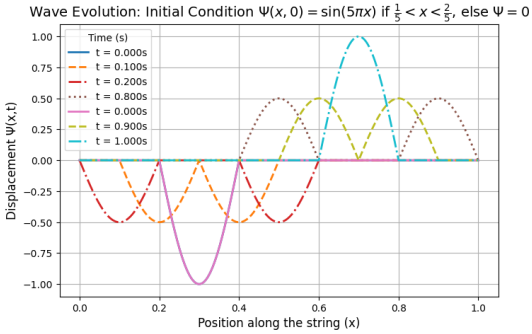

Fig. 3: Vibration of String of length $L = 1$ with Initial Condition $[\Psi(x, t = 0) = \sin(5\pi x)$ if $1/5 < x < 2/5$, else $\Psi = 0]$ over time

With this third initial condition we avoid the above case. Although there are initial *null* points ($\Psi(x_i, t = 0) = 0$) they do not remain null across time steps. This is because around a null point x_i it does not necessarily hold that $x_{i-1} = -x_{i+1}$, for instance: at $t = 0$, for $x_i = \frac{2}{5}$, $x_{i-1} < 0$ and $x_{i+1} = 0$. This allows horizontal motion of the wave as a null point does not necessarily remain one across time steps.

B. Different iterations algorithms

Now we have implemented the discretized wave equation we can have a look at the diffusion equation. In Figure 4 is shown how this is visualized with the left picture being the beginning state and the one on the right the final one. After this visualization we were able to do test on them and be sure our implemention is correct.

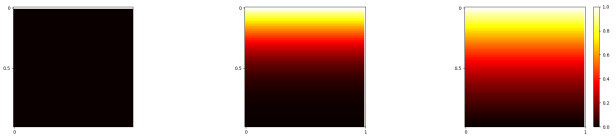

Fig. 4: Visualization of the diffusion inside the system.

To compare the different algorithms on the efficiency in convergence we first made a comparison between the time-dependent diffusion equation and the analytical equation given in the assignment to be sure our implementation was correct [2]. This showed that when comparing the total absolute error at different times the error was maximal for a t-value of 0.0477 for a 50x50 grid and as expected minimal at $t = 1$ since the system is diffused at that time. In the plot 5 this is visible to

a certain extent where at the beginning ($t = 0.01$) there seems to be a larger difference which becomes smaller over time for $t = 0.1$ and $t = 1$.

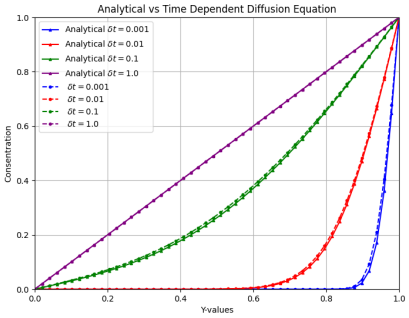

Fig. 5: Comparison of the time-dependent diffusion equation vs the analytical solution. The y-axis represents the spacial solution and each line with a certain color represents a certain time.

Next, in Figure 6, we compared the different iterative methods. It is clear from these results that the SOR with a good ω is by far the most efficient. Additionally, it can be seen that the Gauss-Seidel method is also significantly faster converging than the Jacobi function.

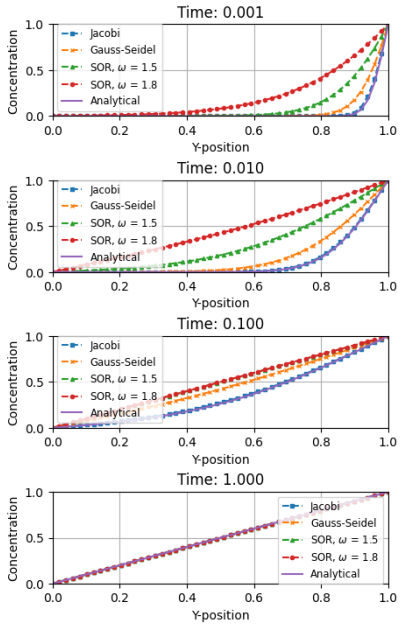

Fig. 6: Comparison of different iteration methods for the diffusion equation. The y-axis represents the concentration, while the x-axis shows the matching spatial position. With N being 50.

To further quantify these observations, Figure 7 presents the number of iterations required for each method, measured against a fixed tolerance criterion. The trends observed in Figure 6 remain consistent with this plot, confirming that SOR with an efficient ω is the most efficient. Important to note is that one iteration in the Y-axis represents one update of the matrix so the "real" amount of iterations are: $i * (n-2)^2$. Whereby n is the grid size, in this case 50, and i the amount of iterations in the y-axis.

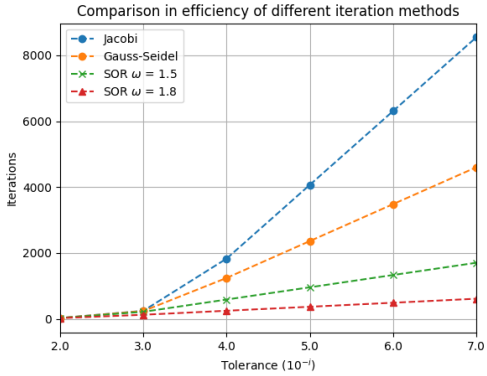

Fig. 7: Comparison of the number of iterations needed to converge to a specific tolerance. With N being 50.

C. Optimal Relaxation Factor ω for SOR

Figure 8 shows how grid size affects the most optimal ω. Initially, ω increases rapidly for small grids, after which the growth slows down and the value seems to converge more slowly. It is intresting to note that ω still increases significantly when the grid is increased from 100 to 200.

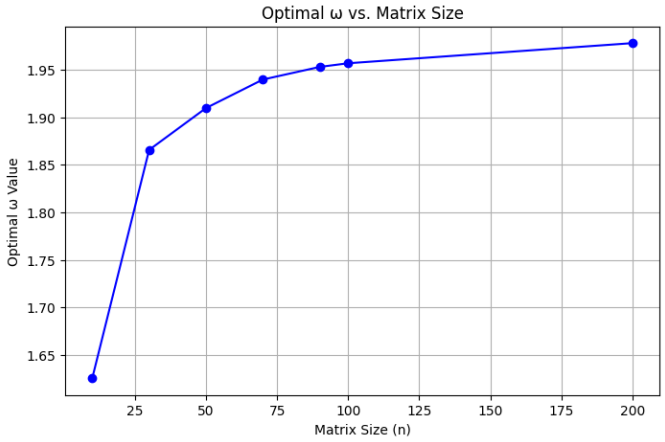

Fig. 8: Optimal SOR Parameter ω value as matrix size increases

D. Object and Insulating material

Figure 9 provides a visualization of the implementation of sinkholes within the diffusion process. The overall diffusion pattern remains largely unchanged; however,

the presence of the sinkhole introduces a significant localized perturbation. Strikingly, it extends itself upwards and downwards relative to the y-axis.

For the insulation materials, the resulting visualizations show similar characteristics. It can be noted that even with a relatively small insulation effect, modeled as a reduction of up to 95% of the normal diffusion rate, a clear region of low concentration emerges. This emphasizes the impact of local material properties on the overall diffusion dynamics.

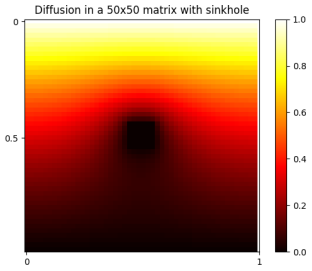

Fig. 9: Visualization of a sinkhole in the 50x50 matrix. The sinkhole is located at row 22:28 and column 22:28. The final state is shown.

In Table I the results are presented for the isolating objects and the different sinkholes as described in section I-F. It can be seen that a small sinkhole requires more iterations than a large sinkhole. Furthermore, the difference in iterations between one and two sinkholes is relatively small. It is also notable that the optimal value of ω follows a similar trend as previously observed: a lower optimal ω generally results in a reduction of the number of required iterations.

	Jacobi	**Gauss-Seidel**	**SOR (1.8)**	**Opt ω**	**Opt SOR**
N	8548	4602	610	1.91	217
S1	5016	2687	340	1.89	165
S2	2684	1430	161	1.84	109
S3	4749	2554	329	1.88	164
I1	5865	3147	437	1.91	188
I2	3171	1692	215	1.87	119
I3	5959	3205	455	1.91	192

TABLE I: Comparison of iterative methods for the different sinkholes and insulating materials. The tolerance used is 10^{-7} and the *xatol* for the optimization method is 10^{-3}. Each column represents the number of iterations needed to converge to this tolerance.

III. Discussion and Conclusion

While the time-dependent discretization of our continuous equations served to simulate wave and diffusion, this approach proved to be inefficient when reaching steady-state solutions. Explicit time-stepping captured the oscillatory behaviour in the wave equation, which shows it is well suited for dynamic simulations. However, when these methods were applied to diffusion, they required a large number of iterations to converge.

To overcome this, iterative solvers were used. Jacobi, Gauss-Seidel and Sucessive-Over-Relaxation (SOR) were compared. The Jacobi method performed the slowest, while Gauss-Seidel improved convergence by incorporating updated values in each iteration. The SOR method was able to outperform both but required optimisation of the ω relaxation factor. We found that optimal ω increases with grid size N, but the effect reduces at higher sizes.

These findings highlight the importance of choosing the correct method based on the type of problem, where explicit time-stepping is more effective for dynamic solutions, and iterative solvers are best for steady-state problems where the equilibrium should be reached efficiently.

Our paper also found that the existence of objects such as insulating materials or sinkholes can influence diffusion behaviour and affect the optimal relaxation parameter ω. We found that a lower optimal ω generally results in a reduction of the number of required iterations. These objects create local variations, which potentially require adaptive methods to remain efficient.

Future work could explore this, dynamically adjusting ω during computation, allowing it to adapt to different regions of the space. Further research could also include extending these methods to non-uniform grids and more complex shapes, which could enhance the applicability of these techniques.

We hope this research speaks to the flexibility required when modelling and simulating situations with continuous behaviour over time: the quality of a model is not solely determined by iteration count or time step size, and in critical scenarios, the specifics of the system dictate which iterative method is most suitable.

References

[1] Rudolph E Langer. “On the connection formulas and the solutions of the wave equation”. In: *Physical Review* 51.8 (1937), p. 669.

[2] UvA. *Assignments for Scientific Computing*. en. 2021.

[3] David M. Young. “A historical overview of iterative methods”. In: *Computer Physics Communications* 53.1 (1989), pp. 1–17. ISSN: 0010-4655. DOI: 10.1016/0010-4655(89)90145-8. URL: https://www.sciencedirect.com/science/article/pii/0010465589901458.