Assignments for Scientific Computing

General guidelines

You are requested to write three concise reports of all the practical work you do for this course. The reports should contain a clear introduction stating the problem at hand, a theory section that introduces necessary theory, a methods section that describes the methods used, e.g. short pseudo code of algorithms, a results section that presents the main results that you obtained, figures of the simulated results, and finally a short discussion that critically goes through the results in view of the original problem, e.g. did you solve the problem in a satisfactory and complete way. It is important to also include information about the software that you create. That is, you also provide text that explains how you implemented the simulations. *Any reader of your report should be able to reproduce your results.* Furthermore, when multiple options seem possible to reach a solution, try to motivate your choice. Use this as a guideline to decide the amount of detail that will go into your report.

Please note that a lab assistants are available during the lab sessions. If you have any questions regarding the practical work you should ask them for advice. We strongly advise you, if you have any questions, to attend the lab sessions. Beyond this, you are invited to make full use of the Canvas 'Discussions' page. When you have a question it is highly likely several others have the same or similar questions in mind, and discussing it openly on Canvas will increase efficiency for everyone.

You are asked to write the report in *groups of three* people. Use Canvas groups (even if you work alone) as discussed during the first 'Course Overview' lecture. Your report should be 6 pages maximum including figures, references, and appendices if applicable. Half a point per extra page is subtracted from the assignment mark. *Note:* You can use the assignment document as a valid reference, and refer to figures and equations in it to avoid text duplication.

For the programming language we strongly suggest Python or C/C++. Other languages are possible at your own responsibility, the TAs might not be able to provide help with them.

Every set is graded and the combined result counts for 50% towards your final grade (≥ 5.0 is required). The other half of the grade is determined by the result of the exam. The lab reports are to be submitted through Canvas, and should be in the PDF format. Together with your lab report you must send (in a zip archive) all your source files (both a URL to the repository and a copy of the repository) and possibly graphs and or movies not presented in the report. We should be able to run your programs (and this will be tested).

You will be graded on the structure and content of your report. There is also the possibility to obtain bonus points through some optional exercises.

Assignment sets

Note: the deadlines will be at 8pm! For more details see the first introductory slides.

Set 1: Vibrating string, Time dependent diffusion, Jacobi iteration, Gauss-Seidel iteration, Successive over relaxation.
Deadline: End of 3rd week.

Set 2: Diffusion-limited aggregation, Random walk, Reaction-Diffusion System.
Deadline: End of 5th week.

Set 3: Challenge assignment! 1. Solve the Karman vortex street with three methods: FD, LBM, FEM methods (Navier-Stokes equation). 2. Optimize the position of the router to get a good WiFi signal (Helmholtz equation).
Deadline: End of 7th week.

1 Assignment Set 1

1.1 Vibrating string

The problem at hand is the numerical solution of a one-dimensional wave equation, which might describe the vibrations of a uniform string, the transport of voltage and current along a lossless transmission line, the pressure and flow rate of a compressible liquid or gas in a pipe, sound waves in gases or liquids, and optical waves. In this example, the domain of the wave equation is a string.

The one-dimensional wave equation is

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

The solution $\Psi(x,t)$ is the vibration amplitude expressed as a function of position and time. The problem becomes fully posed with the addition of boundary conditions on the spatial domain, and initial position and velocity distributions. We attempt a numerical solution method by introducing a uniform discretization of the spatial and temporal domains, and approximating the partial differential equation by finite difference expressions. The grid points of this problem consist of discrete nodal points at which Ψ is to be evaluated.

A. *(0.5 point)* Discretize the wave equation, and write it in a form suitable for implementing in a computer program. Assume that the boundaries are fixed, $\Psi(x = 0, t) = 0$, $\Psi(x = L, t) = 0$. L is the length of the string. Take $L = 1$ for simplicity. Divide the string in N intervals, so that the interval length is $\Delta x = L/N$. Also consider the boundary cases.

If you use Euler's method, you need to use both $\Psi(x,t)$ and $\Psi_t(x,t)$ as variables. Or use the stepping method from the lectures, which uses Ψ at the two most recent time points to calculate it at the next one.

B. *(1 point)* Implement the time stepping. Determine the time development of the string, with the following initial conditions. The string is at rest at $t = 0$, i.e. $\Psi_t(x, t = 0) = 0$.

i. $\Psi(x, t = 0) = \sin(2\pi x)$.
ii. $\Psi(x, t = 0) = \sin(5\pi x)$.
iii. $\Psi(x, t = 0) = \sin(5\pi x)$ if $1/5 < x < 2/5$, else $\Psi = 0$.

Take $c = 1$ and use the time step $\Delta t = 0.001$. Plot the result at several times in the same figure, e.g. varying the color of the curve.

C. *(1 point)* Make an animated plot of the time development. This can be done from within matplotlib, see the following reference:
`https://matplotlib.org/stable/users/explain/animations/animations.html`
With this technique, you can show the animation from within the Python program, or save it to a file in various video formats to use later, e.g. in presentations.

You can also use matplotlib to save individual images, e.g. in the `.png` format, and then pack the images into an animation using `ffmpeg` or `avconv`.

Optional. *(1 point)* Use the Leapfrog method (also introduced later in the course) as time integrator! How does it change the accuracy and stability as the simulation progresses? *Context*: Symplectic integrators are time-integration methods designed to preserve the underlying phase-space (symplectic) structure (for Hamiltonian systems). As a result, they do not systematically gain or lose energy, but instead keep the energy error bounded over long times, making them ideal for oscillatory and conservative systems. The leapfrog method is a simple, explicit, second-order symplectic integrator that staggers positions and velocities. *Hint*: One time-step should consist of the following: 1. Update Velocity (at half-step); 2. Update Position (at full-step); 3. Update Velocity (at the final half-step).

1.2 The Time Dependent Diffusion Equation

Consider the two-dimensional time dependent diffusion equation

$$\frac{\partial c}{\partial t} = D\nabla^2 c \tag{1}$$

here $c(x, y; t)$ is the concentration as a function of the coordinates x and y and time t, and D is the diffusion constant. In all the assignments we will consider a square domain, and without loss of generality we assume that $0 \leq x, y \leq 1$. Furthermore, we always assume the following boundary conditions:

$$c(x, y = 1; t) = 1 \text{ and } c(x, y = 0; t) = 0 \tag{2}$$

So, on the top of the domain the concentration is always equal to one, and on the bottom of the domain the concentration is always equal to zero. Furthermore, in the x-direction we will always assume periodic boundary conditions:

$$c(x = 0, y; t) = c(x = 1, y; t) \tag{3}$$

These boundary conditions are once more drawn in fig. 1.

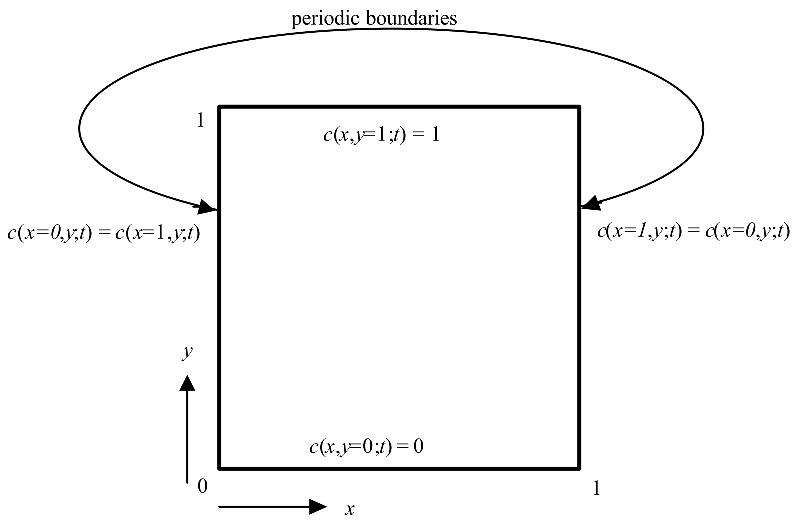

Figure 1: The computational domain and boundary conditions.

As an initial condition we take

$$c(x, y; t = 0) = 0 \text{ for } 0 \leq x \leq 1, 0 \leq y < 1. \tag{4}$$

Note that for this specific set of initial and boundary conditions, the solution depends only on the y-coordinate (because of symmetry) and on time. This is a very useful feature to test the correctness of your code. Furthermore, for this specific set of initial and boundary conditions, the diffusion equation can also be solved analytically. The analytical solutions, assuming $D = 1$, as a function of the y-coordinate are shown in fig. 2 at a number of time points.

Note that for $t \to \infty$ the concentration profile is simply a straight line,

$$\lim_{t\to\infty} c(y, t) = y. \tag{5}$$

You can easily derive this yourself from the time-independent diffusion equation (try it). This is also a powerful check of the correctness of your simulation. For finite times you can also compare the simulation results with the exact solution.

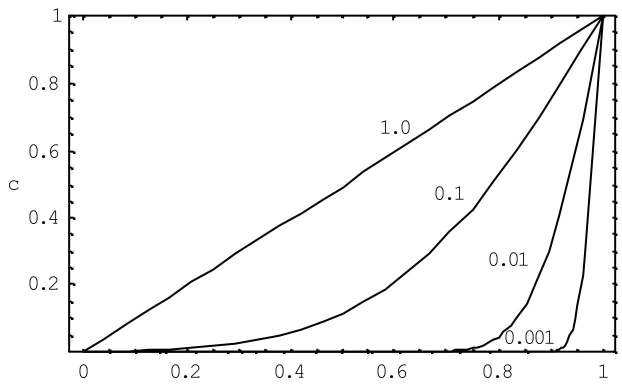

Figure 2: The analytical solution of the concentration, as a function of the y-coordinate, for t equal to 0.001, 0.01, 0.1, and 1 respectively. Here, $D = 1$ and the initial and boundary conditions are as in the text.

Next the spatial and temporal domain are discretized. The x- and y-axes are divided in intervals with length δx. Assuming that we have N interval in the x- and y-directions, we have $\delta x = 1/N$, and $x = i\delta x$, $y = j\delta x$ where $i, j \in (0, 1, 2, \ldots N)$. Furthermore, we assume a small time interval δt, so that $t = k\delta t$ where $k \in \mathbb{N}$.

We now want to find solution to the concentration at these discretized space and time points. With the definition

$$c(i\delta x, j\delta x; k\delta t) \equiv c^k_{i,j} \tag{6}$$

and discretizing the diffusion equation using standard finite difference methods the following explicit scheme is easily derived.

$$c^{k+1}_{i,j} = c^k_{i,j} + \frac{\delta t D}{\delta x^2}\left(c^k_{i+1,j} + c^k_{i-1,j} + c^k_{i,j+1} + c^k_{i,j-1} - 4c^k_{i,j}\right) \tag{7}$$

This scheme is stable if

$$\frac{4\delta t D}{\delta x^2} \leqslant 1 \tag{8}$$

This determines the maximum time step δt that you can take. In this assignment you implement a finite difference simulation to solve the time-dependent diffusion equation, using the finite difference scheme in eq. (7) and subject to the boundary and initial conditions in eqs. (2) to (4).

Due to the five-point *stencil* every grid point only needs local information to update for the next time step. Therefore, this calculation scheme is also suitable for parallel computations.

D. *(0.5 point)* Determine the equation to use at the boundaries of the domain. Clearly show the ranges of the indices of the grid. A figure is extremely helpful for figuring this out.

Write a program for the simulation of the two-dimensional time dependent diffusion equation discretized using the explicit finite difference formulation from eq. (7). You may want to write your data to a file (e.g. after every iteration, or maybe after every 100 iterations) so that you can analyze the data later on or plot them immediately.

E. *(1 point)* Test the correctness of your simulation. Compare to the analytic solutions, plot $c(y)$ for different times. The analytic solution is

$$c(x,t) = \sum_{i=0}^{\infty} \operatorname{erfc}\left(\frac{1 - x + 2i}{2\sqrt{Dt}}\right) - \operatorname{erfc}\left(\frac{1 + x + 2i}{2\sqrt{Dt}}\right). \tag{9}$$

F. *(1 point)* Plot the results, show the 2D domain, with a color representing the concentration at each point. Make a plot of the state of the system at several times: t = {0, 0.001, 0.01, 0.1, and 1}.

G. *(1 point)* Make an animated plot of the time dependent diffusion equation until equilibrium.

1.3 The Time Independent Diffusion Equation

Now assume that we are not very much interested in the time development of the concentration profile (i.e. the transient behavior), but only in the steady state. In that case it is possibly more effective to directly solve the time independent diffusion equation:

$$\nabla^2 c = 0. \tag{10}$$

This is the famous Laplace equation. We again assume the same boundary conditions as in the previous section. Taking the same spatial discretization as before, and again applying the same 5-points stencil for the second order derivatives, eq. (7) transforms to the following set of finite difference equations:

$$\frac{1}{4}\left(c_{i+1,j} + c_{i-1,j} + c_{i,j+1} + c_{i,j-1}\right) = c_{i,j} \tag{11}$$

for all values of (i, j). Note that the superscript k is no longer present, because the time behavior is now suppressed. Many methods exist to solve the equations. We will here concentrate on iterative methods, because they are potentially efficient, and demand less memory than direct methods (and allow for relative easy parallelism). A direct method is tried in exercise set 3.

1.4 The Jacobi Iteration

Using now the superscript k to denote the k-th iteration, the Jacobi iteration is immediately suggested from eq. (11):

$$c_{i,j}^{k+1} = \frac{1}{4}\left(c_{i+1,j}^{k} + c_{i-1,j}^{k} + c_{i,j+1}^{k} + c_{i,j-1}^{k}\right) \tag{12}$$

Note that eq. (12) is easily derived from eq. (7) by putting

$$\frac{\delta t D}{\delta x^2} = \frac{1}{4} \tag{13}$$

i.e. the Jacobi iteration is nothing but the solution of the time-dependent equation using the scheme from eq. (7) with the maximum allowed time step. This suggests that more efficient iterative methods are needed.

There is one noticeable difference between the Jacobi iteration and the solution of the time dependent equation. In the time dependent case one defines the time step and the total time that should be simulated. In an iterative method one needs a stopping condition. This stopping condition typically is some global measure. Assume that the stopping condition is such that the solution is assumed to be converged if for all values of (i, j)

$$\delta \equiv \max_{i,j} \left|c_{i,j}^{k+1} - c_{i,j}^{k}\right| < \epsilon, \tag{14}$$

where ϵ is some small number, say 10^{-5}.

For implementing the Jacobi iteration, note that separate matrices are needed for $c_{i,j}^{k}$ and $c_{i,j}^{k+1}$, one cannot simply use one matrix and update it, as one would then overwrite values that are needed later.

1.5 The Gauss-Seidel Iteration

An improvement over the Jacobi iteration is the Gauss-Seidel iteration, where during the iteration a new value is used as soon as it has been calculated. Assuming that the iteration proceeds along the rows (i.e. incrementing i for fixed j), the Gauss-Seidel iteration reads

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

The Gauss-Seidel iteration is not a big improvement over the Jacobi iteration (in terms of the amount of iterations needed for convergence) and is only a first step in introducing the Successive Over Relaxation method (next section). However, the update can be performed *in place*.

1.6 Successive Over Relaxation

Now that you have the parallel Gauss-Seidel iteration in place, it is easy to take the next and final step. The Gauss-Seidel iteration did not provide a huge improvement over the Jacobi iteration. A next improvement comes from the Successive Over Relaxation (SOR). SOR is obtained from Gauss-Seidel by an over-correction of the new iterate, in formula

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

This method converges only for $0 < \omega < 2$. For $\omega < 1$ the method is called under-relaxation. The new value is then the weighted average of the Gauss-Seidel method and the previous value. For $\omega = 1$ we recover the Gauss-Seidel iteration.

It turns out that for our diffusion problem the optimal ω (that minimizes the number of iterations) lies somewhere in the interval $1.7 < \omega < 2$. The exact value depends on the grid size N.

H. *(1 point)* Implement the Jacobi iteration, the Gauss-Seidel method and SOR. Try $N = 50$. Test the methods by comparing the result to the analytical result in eq. (5), i.e. the linear dependence of the concentration on y.

I. *(1 point)* Show how the convergence measure δ in eq. (14) depends on the number of iterations k for each of the methods. A log-lin plot may be suitable. For SOR, choose a few representative values for ω.

J. *(1 point)* In the SOR method, find the optimal ω. How does it depend on N?.

So far we have only looked at diffusion in a rather dull domain. Now that we have an efficient iterative solver available, it's time to start including some object into the domain. So, now we assume that within our computational domain we include say a square object. We assume that the object is a sink for the diffusion concentration, that is, the concentration is zero everywhere on the object.

K. *(2 points)* Implement the possibility to include objects into the computational domain. The objects should be sinks. Experiment a little bit with some objects in the computational domain (e.g. a rectangle or a few rectangles, ...). What is the influence on the number of iterations. What about the optimal ω , is it influenced by the presence of objects? Look at the resulting concentration fields, and try to interpret what happens. The implementation in this exercise will also be used for diffusion-limited aggregation in Set 2.

Hint: For the iterations, the presence of the objects is not complicated. If a point (i, j) is part of an object, the concentration is just 0, and an iteration is not necessary (i.e., the new value is also 0). Therefore, you must implement some easy encoding of the object in the computational grid, and during the iterations simply test if the grid point that you are updating is part of the object or not. If not, you apply the

SOR rule, if yes, just put the new value to zero. The easiest encoding is just an extra array of integers, where e.g. a one-value would code for the presence of an object, and a zero value for the absence of an object.

L. *(1 point)* Think of a way to incorporate objects with insulating material in your domain. What changes in the time evolution of the system? And in the final state?

2 Assignment Set 2

2.1 Diffusion Limited Aggregation

The diffusion-limited aggregation (DLA) is a growth model based on diffusing particles. The growth is started with a single seed (just a small square object in a single lattice point, at the bottom of the computational domain). As described in detail in the lecture notes, DLA can be modeled by solving the time independent diffusion equation (i.e. the Laplace equation), locating growth candidates around the cluster, and assigning a growth probability p_g for each growth candidate, as a function of the concentration of diffusing nutrients at that growth location. Next, a single growth candidate is added to the cluster with probability p_g. After this growth step, the diffusion equation is again solved, and the process is iterated for a large number of growth steps. You currently have almost all the tools available for a simulation of the DLA growth model. Let us take a closer look at the growth model itself, allowing you to insert this into your programs. Figure 3 shows a possible configuration of an object (the filled dots) and the growth candidates (the open circles). A growth candidate is basically a lattice site that is not part of the object, but whose north, east, south, or west neighbor is part of the object.

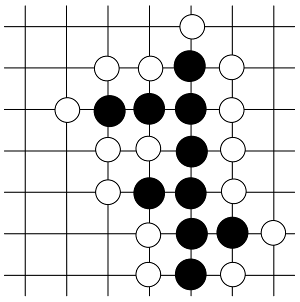

Figure 3: DLA cluster (filled circles) with growth candidate sites (open circles)

The probability for growth at each of the growth candidates is calculated by

$$p_g(i,j) = \frac{c_{i,j}^{\eta}}{\sum_{\text{growth candidates}} c_{i,j}^{\eta}}$$

The parameter η determines the shape of the object. For $\eta = 1$ we get the normal DLA cluster. For $\eta < 1$ the object becomes more compact (with $\eta = 0$ resulting in the Eden cluster), and for $\eta > 1$ the cluster becomes more open (and finally resembles say a lightning flash).

Modelling the growth is now a simple procedure. A set is created of all growth candidates with their associated weights, and a single candidate is chosen. *Hint: use numpy.choice with the p parameter for this*

A. *(4 points)* Implement the growth model, paying special attention to the calculation of the growth probabilities. Run a number of growth simulations. Try to do this for a domain of size 100×100 and investigate the influence of the η parameter. Can you still optimise by setting your ω parameter in the SOR iteration to a specific value?

Hint: the SOR iteration is run over and over again on a slowly growing object. As the growth step is constructed in such a way that on average only one lattice site is grown to the object, the concentration fields will hardly change. Therefore, it is advantageous to start a new SOR iteration with the solution of the previous growth step. Also, you can start the simulation with the analytical result for the empty system, the linear concentration gradient of eq. (5).

B. *(1 point)* Think of a way to reduce the time required to solve the diffusion equation. Compare your results for the 100×100 grid in question A and try larger grid sizes. Some suggestions: parallelize one of the iteration schemes in the previous exercise set, and (*optional*) use a GPU to do the calculations.

2.2 Monte Carlo simulation of DLA

DLA can also be simulated by releasing random walkers on a grid, and letting them walk until they hit the cluster. When they hit, the walkers are stopped and become part of the cluster.

One random walker at a time is released in the system. It moves in steps, which are randomly chosen to be one lattice point up, down, left, or right. If the walker reaches a cell neighboring the cluster, the walker is stopped there, so that the cell with the walker becomes part of the cluster.

To simulate with the same boundary conditions as above, start the walkers on a randomly chosen point on the top boundary. If a walker walks out of the system on the top or bottom boundary it is removed and a new one created instead. If it walks across the left or right boundary, it should enter the system from the other side, as periodic boundary conditions were assumed in the horizontal direction.

C. *(2 points)* Implement the Monte Carlo version of DLA. Compare the resulting cluster to those obtained with the diffusion equation.

D. *(1 point)* In this model, the η parameter is no longer easily variable, it is fixed to 1. However, another parameter can be introduced, namely a sticking probability p_s. The sticking rue can then be stated in the following way: if the walker enters a cell which is a neighbor of the cluster, it stops there with probability p_s. If it does not stick, the walk continues as normal. The walker is however not allowed to move into a site belonging to the cluster. Run the simulation for different values of p_s, and plot the results. How does the cluster shape depend on p_s?

2.3 The Gray-Scott model - A reaction-diffusion system

The Gray-Scott model (J. E. Pearson, *Science*, Vol 261, 5118, 189-192 (1993)) describes a system of chemical reactions, involving the two chemicals U and V. Both chemicals diffuse in the system, and also react with each other. The reaction rate at any point in space, is determined by the *local* concentrations of U and V. The reactions are:

$$U + 2V \rightarrow 3V \tag{15}$$

$$V \rightarrow P \tag{16}$$

U is continuously fed into the system. It then reacts with V to produce more V. V spontaneously decays into P, a reaction product which is not interacting with U and V. The first reaction is said to be autocatalytic, since the reaction product V enhances the production of itself.

If we let u and v denote the concentrations of U and V, the following equations can be formulated.

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u - uv^2 + f(1-u), \tag{17}$$

$$\frac{\partial v}{\partial t} = D_v \nabla^2 v + uv^2 - (f+k)v. \tag{18}$$

Here, f controls the rate at which U is supplied, and $f + k$ controls the rate at which V decays. For different values of f and k a large variety of behaviors can be observed. Some result in stable patterns, while others remain time-dependent.

E. *(3 points)* Implement the Gray-Scott model in two dimensions. Explain the discretization and how the equations and boundary conditions are implemented in the program (you may choose which boundary conditions to use). Plot the resulting concentrations of U and/or V for several choices of the parameters. The time-dependent diffusion program from Set 1 can be used as a base. In this case, there are two variables to keep track of. For parameters, start with $\delta t = 1, \delta x = 1, D_u = 0.16, D_v = 0.08, f = 0.035, k = 0.060$. For the initial conditions, you can take $u = 0.5$ everywhere in the system, and $v = 0.25$ in a small square in the center of the system, and 0 outside. Try adding a small amount of noise too.

3 Assignment Set 3

Let Op! This third assignment is under development, it might still change during the first half of the course!

This final assignment presents you with two open challenges. What does that mean? You will go beyond the strict course material in application. Use all the techniques you have learned, and go further when needed.

For this assignment you can use everything at your disposal, apart from asking external human help! You are welcome to consult books, articles, AI-tools (including LLMs).

Most important: always motivate and argue your choices well, for this assignment the train of thought, the techniques you apply, and the reason why you are applying them is much more important than the numerical value of the results. Results without proper motivation will not be accepted.

At the end of the course I will publish the ranking among the teams:

- Highest achieved numerically stable *Re*.
- Highest WiFi signal strength.

Good luck!

3.1 Solving the Navier-Stokes equation

At the heart of fluid dynamics lies the incompressible Navier-Stokes equations, a set of PDEs that describe the motion of fluid-like substances. You have seen this at the second guided self-study. For a fluid with constant density ρ and viscosity ν, the equations are expressed as:

$$\begin{aligned} &\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u}, \\ &\nabla \cdot \mathbf{u} = 0, \end{aligned} \tag{19}$$

where $\mathbf{u}$ is the velocity vector and p is the pressure.

In both industry and research, these equations are used to model everything from the aerodynamic lift of an aircraft wing and the cooling of microelectronics to complex blood flow through human arteries. Because analytical (exact) solutions to these equations exist only for the simplest geometries, we rely entirely on numerical solutions. The accuracy and stability of these solutions depend heavily on the chosen discretization method, the resolution of the mesh, and the handling of the non-linear convective term $(\mathbf{u} \cdot \nabla)\mathbf{u}$.

It is very important to understand that several different solution options might exist for the same mathematical model. Understanding the benefits, weaknesses, and limits of these options allows you to identify the best solution in a given problem context. In this challenge, you will implement the same flow problem using three different numerical methods: 1. finite difference, 2. finite element (using ngsolve), 3. lattice Boltzmann.

To validate our numerical solvers, we frequently look to the Kármán vortex street. This phenomenon occurs when a fluid flows past a bluff body (such as a cylinder) at specific Reynolds numbers, resulting in a repeating pattern of swirling vortices caused by the unsteady separation of flow. It serves as a classic benchmark in computational fluid dynamics (CFD) because it tests a solver's ability to capture time-dependent oscillations and wake dynamics accurately.

Beyond the lab, Kármán vortex streets occur naturally on a massive scale; they are often visible in satellite imagery as clouds drift past high-altitude islands, and they are the physical cause behind the "singing" of power lines in the wind or the structural vibrations of underwater cables.

Challenge A. *(5 points)* Implement the Kármán vortex street following the setup below, test for correctness, then make it break! Test the limits of each of the three methods: how high can you go with *Re* and stay stable?

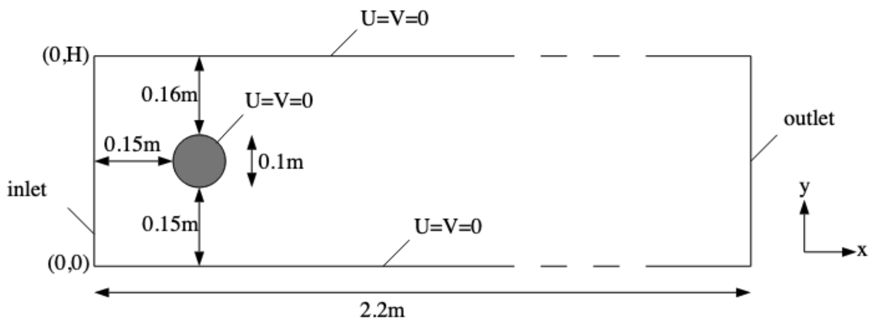

Figure 4: Geometry setup for the Kármán vortex street with SI units.

While trying to reach the highest *Re* you can, build and present your own opinion on the three methods! Here are some ideas on what you can compare to get you started (and you are invited to bring your own ideas too): How does the computational cost compare? How does the accuracy and/or numerical stability scale with the mesh resolution? How does accuracy differ very close to curved objects? How easy is it to produce an implementation? Is it suitable for parallel execution? Would it fit GPU execution?

3.2 Optimizing the position of the WiFi router

Here you will apply the Helmholtz equation to a practical, modern challenge: approximating WiFi signal strength within a complex indoor environment. WiFi signals are electromagnetic waves that oscillate at high frequencies (typically 2.4 GHz or 5 GHz). While full Maxwell equations are required for high-fidelity physics, the scalar Helmholtz equation provides an excellent approximation for mapping signal coverage in a 2D floor plan:

$$\Delta u + k^2 u = f, \tag{20}$$

where u is the complex wave field, $k = \omega/c$ is the wavenumber (scaled in this document for visualization), and f is a Gaussian pulse source at the router position

The source term f in this challenge can be modeled as a Gaussian pulse centered at the router position (x_r, y_r):

$$f(x,y) = A \cdot \exp\left(-\frac{(x-x_r)^2 + (y-y_r)^2}{2\sigma^2}\right),$$

where the amplitude $A = 10^4$ controls the overall signal strength, and the width $\sigma = 0.2$ m determines the spatial extent of the source (40 cm diameter at half-maximum)

Side-note: This represents a localized, omnidirectional point source. The Gaussian shape provides smooth derivatives for numerical stability while approximating a point emitter.

Challenge B. *(5 points)* Find the best position for the router that yields the highest signal strength summed over the given measurement points!

Use the following floor plan. The wall thickness is uniformly 0.15 m.

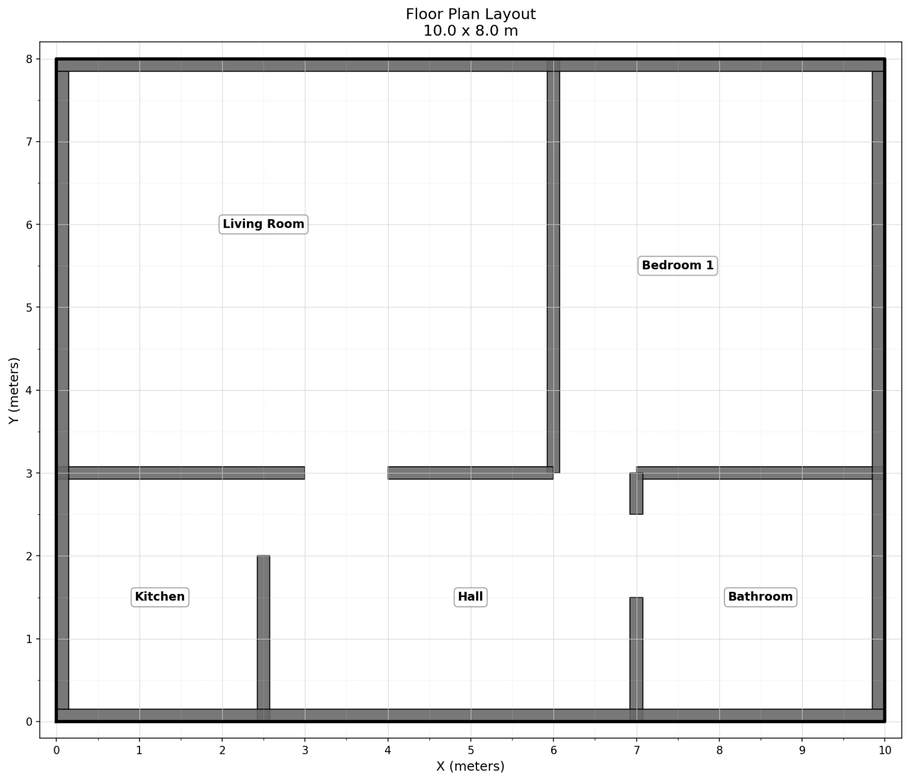

Figure 5: Floor plan of the house.

The material properties are as follows:

Region	Refractive Index	Effect
Air	1.0	Standard propagation
Walls	$2.5 + 0.5j$	Slower propagation + absorption

The boundary conditions:

- Outer walls: Impedance (absorbing) BC: $\partial u/\partial n - iku = 0$
- Internal walls: Continuity with the k values from the above table (Walls).

The wavenumber k is calculated using the standard formula:

$$k = \frac{2\pi f}{c}, \tag{21}$$

where: f is the frequency (e.g. 2.4 GHz or 5 GHz) and c is the speed of light ($3 \times 10^8 m/s$).

This leads to the following:

Frequency	Wavelength	Wavenumber (k)
2.4GHz	$\sim$ 12.5 cm	$\sim$ 50.3rad/m
5.0GHz	$\sim$ 6.0 cm	$\sim$ 104.7rad/m

Using the given parameters and floor plan, an approximating solution for the signal strength would look like this:

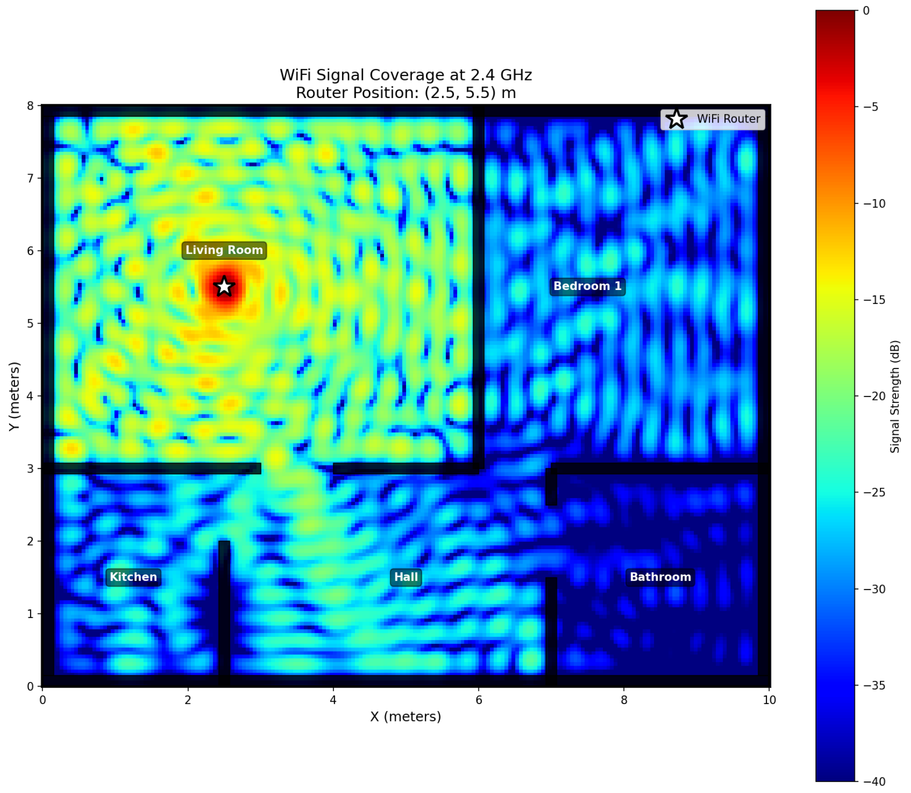

Figure 6: Example of WiFi signal strength. Note: approximation for better visibility, the wavenumber is scaled: $1/3 * 2.4 = 0.8$ GHz

In order to get the sum of signal strength, use the measurement locations below. Evaluate the signal strength by averaging in a small circular region ($r = 5$ cm) around these points, then add sum up these 4 averages to get the total measured signal strength.

Room	**X [m]**	**Y [m]**
Living room	1	5
Kitchen	2	1
Bathroom	9	1
Bedroom 1	9	7

Table 1: Signal strength measurement locations.

At real WiFi frequencies, the wavelength ($\approx 6 - 12$ cm) is very small compared to the floor plan (10×8 m). Resolving these waves accurately would require an extremely fine mesh (element size << wavelength, typically $\lambda/10$) leading to millions of elements (≈ 1 cm mesh for 2.4 GHz) and a massive computational cost. Not to mention that it is an optimization problem, therefore you will need a sampling strategy, and run the simulation multiple times to generate and find the best positioning. How will you solve this? Will you scale the wavenumber and come up with an approximation? Will you use HPC to brute-force it? Or will you do something else? Given this is an open challenge, there are many potential solutions, but whichever route you take, **it is very important to motivate your choices**! Convince the reader of the report that you have thought this through and the chosen method yields a good solution.

4 Appendix. Tool recommendations

Python
Always use Python $\geq$ 3.10.

The suggested distribution is https://anaconda.org/.
Hint: don't overwrite your system's python (which might be older), install a new distribution next to it.

Python Modules - required
matplotlib
numpy
scipy
json

Python Modules - optional
numba
taichi
cupy

You can also take a look at Google Colab https://colab.research.google.com/, for instance for GPU programming.

Scientific code development practices Aim to follow good practices: write structured, well-documented code, apply unit test (e.g., look at https://docs.python.org/3/library/doctest.html), even if you use a notebook develop modular code in multiple files that are imported in the notebook.

For more details on good scientific coding practices please see the following workshop materials: https://github.com/JuliaDynamics/GoodScientificCodeWorkshop.

Code editor
There is a wealth of options, the default suggestion is https://code.visualstudio.com/.

AI tools
At all times the official guidelines of the university are in effect, unless explicitly instructed by the lecturer. Also see discussion during the intro lecture.

Version control
Use git, preferably https://github.com/. Your UvA ID will give you a 'pro' account.

Drawing figure
Beyond matplotlib you might want to use TikZ (latex) or Inkscape.

Other useful references
An introduction to Numpy and Scipy:
https://sites.engineering.ucsb.edu/~shell/che210d/numpy.pdf

Python Scientific Lecture Notes
https://lectures.scientific-python.org/

Scipy getting started
https://projects.scipy.org/getting-started.html

Matplotlib documents
https://matplotlib.org/stable/

5 Appendix. Matrices

Example of a matrix encoding the connections of a 3 by 3 grid. The boundary conditions are “reflective” or “no-flow”, meaning that there is no diffusion over the edges of the system. See Heath (optional book), Chapter 11.

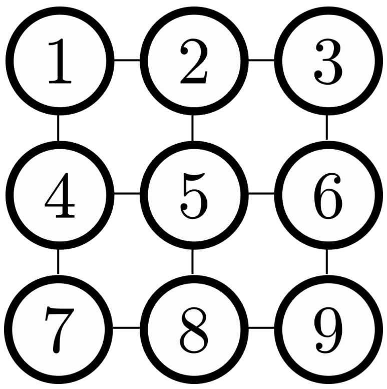

Figure 7: A 3 by 3 grid.

$$\begin{pmatrix} -2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -3 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -3 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -2 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \end{pmatrix} \tag{22}$$