Exercises for Scientific Computing (Set 1)

Deadline: February 23, 2024

Teacher: Jaap Kaandorp (J.A.Kaandorp@uva.nl)
Assistants: Niksa Mohammadi bagheri (n.mohammadibagheri@uva.nl)
and Konstantinos Kevopoulos (kevopoulos.kon@gmail.com)

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. pseudo code of algorithms, a results section that presents the main results that you obtained, pictures 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 satisfactory. It is important that you 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. This more or less dictates the amount of detail that will go into your report.

Please note that a lab assistant is available during the lab sessions. If you have any questions regarding the practical work you should ask him for advice. We strongly advise you, if you have any questions, to attend the lab sessions.

You are asked to write the report in groups of two people. Use Canvas to set up groups, even if you work by yourself. Your report should be 8 pages maximum including appendices, figures and references if applicable. Half a point per extra page is subtracted from the assignment mark. Please use Python 3.

Every set is graded and the combined result counts for 50% towards your final grade. 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 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. Please look at the grading scheme on Canvas for more details.

Deadlines

Set 1: Vibrating string, Time dependent diffusion, Jacobi iteration, Gauss-Seidel iteration, Successive overrelaxation.

Set 2: Diffusion-limited aggregation, Random walk, Reaction-Diffusion System.

Set 3: Eigenmodes of a circular drum, Direct methods.

1 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 $\Psi'(x,t)$ 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. $\Psi'(x, t = 0) = 0$.

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 references:
https://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/
http://matplotlib.org/examples/animation/simple_anim.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`.

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.

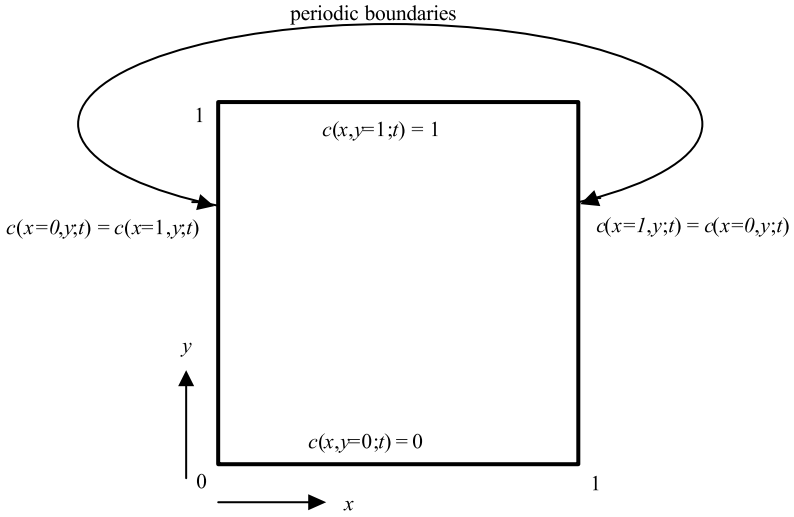

Figure 1: The computational domain and boundary conditions.

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 that the solution only depends 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.

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

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

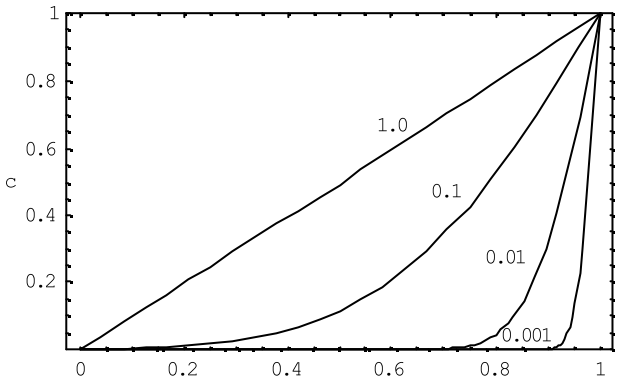

Figure 2: The analytical solution of the concentration, as a function of the y-coordinate, for t equal to 0.001, 0.01, 0.1, and 1 respectively. Here, $D = 1$ and the initial and boundary conditions are as in the text.

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

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

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 a 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 file (e.g. after every iteration, of maybe after every 100 iterations) so that you can analyze the data later on, or plot it 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

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

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.

Optional. *(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?