Exercises for Scientific Computing 2017/2018

Version 1 – March 20, 2018
Teacher: Jaap Kaandorp, J.A.Kaandorp@uva.nl
Assistant: Britt van Rooij, B.J.M.vanRooij@uva.nl

You are requested to write three concise reports of all the practical work you do for this course. The the report 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, etc, 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, like did you solve the problem satisfactory. It is important that you also write a report 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 advice you, if you have any questions, to attend the lab sessions.

You are allowed to write the report in groups of maximum two people. If you wish to collaborate with a fellow student, you should report this to the assistant when you start this practical work (by sending an e-mail).

You will get a single mark for your lab work. Together with the mark you get for the examination, your final mark is will be calculated as the average. The lab reports are to be submitted through blackboard, and should be in the PDF format. If working in pairs, please submit one report for the pair. Together with your lab report you must send (in a zip archive) all your source files and possibly graphs and or movies. We should be able to run your programs (and this will be tested).

Deadlines

Set 1: Vibrating string, Time dependent diffusion, Jacobi iteration, Gauss-Seidel iteration, Successive overrelaxation.
Deadline Wednesday 25.4 by midnight.

Set 2: Diffusion-limited aggregation, Random walk, Reaction-Diffusion System.
Deadline Friday 11.5 by midnight.

Set 3: Eigenmodes of a circular drum, Direct methods, Schrödinger's equation.
Deadline Wednesday 25.5 by midnight.

1 Set 1

1.1 Tools

A. *(Optional, 0.5 point)* Install and try out a Linux distribution on your own computer, with the aim to use it for the exercises during this course. Document your experience briefly. If you already use Linux, great! Briefly document your favorite (preferably free) programming tools for the same bonus point. How are you going to become a better programmer?

1.2 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. Numerical solution results in a uniform discretization of the spatial and temporal domains, and approximates 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.

B. *(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.

C. *(1.5 points)* Implement the time stepping. Simulate 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.

Optionally, make an animated plot. 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.

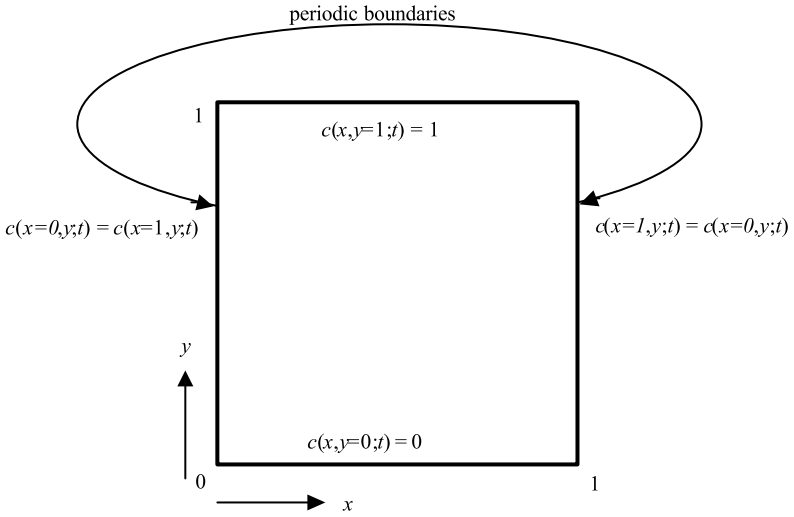


Figure 1: The computational domain and boundary conditions.

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`.

1.3 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.

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,

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

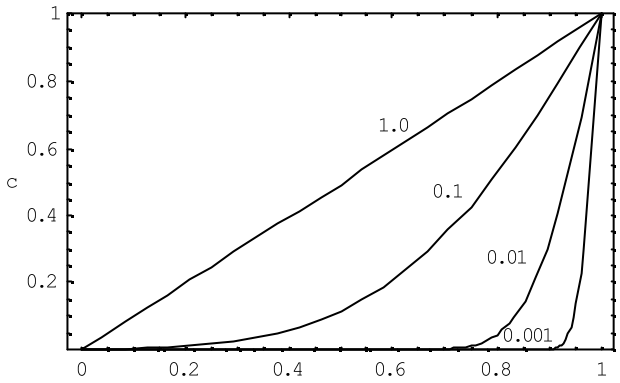


Figure 2: The analytical solution of the concentration, as a function of the y-coordinate, for t equal to 0.001, 0.01, 0.1, and 1 respectively. Here, $D = 1$ and the initial and boundary conditions are as in the text.

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 $k \in (0, 1, 2, ...)$.

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

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

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 form of 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

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

F. *(1.5 points)* Plot the results using matplotlib, show the 2D domain, with a color representing the concentration at each point. Hint: matplotlib's `imshow` function can be used. Make a plot of the state of the system at several times: t = {0, 0.001, 0.01, 0.1, and 1}. Optionally, make an animated plot.

1.4 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.

1.5 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.

1.6 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*.

1.7 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.

G. *(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.

H. *(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.

I. *(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.

J. *(2 points)* Implement the possibility to include objects into the computational domain. The objects should be sinks. 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. Optionally, Read in the objects from an image file, use scipy.misc.imread to easily create a numpy array from the image.

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.

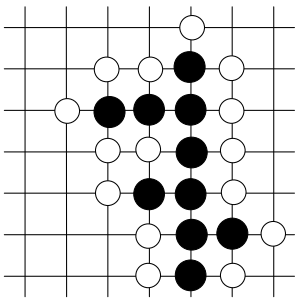


Figure 3: DLA cluster (filled circles) with growth candidate sites (open circles)

2 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, the growth candidates are 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.

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).

Modeling the growth is now a simple procedure. For each growth candidate a random number between zero and one is drawn and if the random number is smaller than the growth probability, this specific site is successful and is added to object. In this way, on average just one single site is added to the object.

K. *(4 points)* Implement the growth model, paying special attention to the calculation of the growth probabilities. Hint: You need random numbers, you can use the function `numpy.random.random()`. Now, with the growth model in place you are ready to perform growth simulations.

Run a number of growth simulations. Take large domains (say 512^2) and investigate the influence of the η parameter. Can you still optimize 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).

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.

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

M. *(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.

N. *(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 Set 3

3.1 Eigenmodes of drums or membranes of different shapes

In this exercise we solve the wave equation on a two-dimensional elastic material. We use boundary conditions that fix the membrane along its edge, and solve for different shapes of the membrane. This time, we will not explicitly solve the time development as in exercise 1, instead we are interested in the *eigenmodes* and the *eigenfrequencies* or resonance frequencies of the system.

A famous question related to this problem is: *Can you hear the shape of the drum*?, or more specifically, whether the membrane shape can be identified uniquely from its eigenvalue spectrum. The answer recently turned out to be no in general, but we'll still try to hear the difference between some simple membranes.

Consider the wave equation in 2 dimensions.

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u \tag{19}$$

We look for a solution in the form

$$u(x, y, t) = v(x, y)T(t) \tag{20}$$

where the time and space dependencies can be separated in two independent functions. Not every $u(x, y, t)$ can be separated this way, rather we state that we are specifically looking for a u of this form due to the nice properties. Inserting this in the wave equation and moving all t-dependencies to the left and all x, y dependencies to the right, we get

$$\frac{1}{T(t)} \frac{\partial^2 T(t)}{\partial t^2} = c^2 \frac{1}{v(x, y)} \nabla^2 v(x, y) \tag{21}$$

The left hand side depends only on t, the right hand side only on x and y. Thus both sides must equal some constant K independent of x, y, and t. Now the left hand side and the right hand side can be treated independently. We'll start with the left hand side:

$$\frac{\partial^2 T(t)}{\partial t^2} = Kc^2 T(t). \tag{22}$$

If $K < 0$, it has an oscillating solution

$$T(t) = A \cos(c\lambda t) + B \sin(c\lambda t), \tag{23}$$

where $\lambda^2 = -K$. $\lambda > 0$ can be assumed without loss of generality. (If $K > 0$, the solution either grows or decays exponentially, and this case is not interesting at the moment. If $K = 0$, the solution is a constant or a linear function of x and y, and has to be $v = 0$ due to the boundary conditions.)

The right-hand side is

$$\nabla^2 v(x, y) = Kv(x, y), \tag{24}$$

an eigenvalue problem. The solutions to this problem give the eigenmodes v and eigenfrequencies λ. A way to find v numerically is to discretize Eq. (19), to obtain a set of linear equations, one for each discretization point. If this system of equations is written in the form $\boldsymbol{M}\boldsymbol{v} = K\boldsymbol{v}$, where $\boldsymbol{M}$ is a matrix, $\boldsymbol{v}$ a vector of $v(x, y)$ at the grid points and K is a scalar constant. This matrix eigenvalue problem can be solved efficiently with library methods e.g. in Python. Boundary conditions: $v(x, y) = 0$ on the boundary, i.e. the boundary is fixed. Take $c = 1$.

O. *(1 point)* Discretize eq. (24). Formulate a matrix version of the eigenvalue problem, taking the boundary conditions into account. Hint: draw a small example,

number the elements and think about which elements are connected and which are not. Draw a figure showing the discretization points and their positions. Show and explain the shape of the matrix for a very small problem, maybe a 4x4 system.

P. *(3 points)* Consider the following three shapes.
i. a square with side length L
ii. a rectangle with sides L and $2L$
iii. a circle with diameter L

Solve the eigenvalue problem. Try `scipy.linalg.eig()`, `eigh()` or `eigs()`, (or something else). Which did you use, and why? Plot the eigenvectors v for some of the smallest eigenvalues, for $L = 1$. Label the plots with their frequencies.

P'. *(Optional, 0.5 points)* For higher performance and 0.5 bonus points, you can try *sparse* matrices: `scipy.sparse.linalg.eig()` etc. Do you see a difference in speed and/or memory use?

Q. *(1 point)* How does the spectrum of eigenfrequencies depend on the size L? Plot the eigenfrequencies for each shape as a function of L. Do the frequencies depend on the number of discretization steps?

R. *(1 point)* Use eq. 20 and the solutions to eq. 24 to construct time-dependent solutions. Show how the first few eigenmodes behave in time. Try to make an animated plot of some eigenmodes for one of the three systems.

Hints for the circular domain: Grid points within the distance $R = L/2$ from the center belong to the domain. Points further away do not, and for all those $v_{ij} = 0$. There are two options for implementing these irregular boundary conditions:

1) The vector $\boldsymbol{v}$ contains only points inside the domain. You need to construct an indexing scheme mapping index in $\boldsymbol{v}$ to a point on the grid, and construct the matrix so that all connections and boundary conditions are satisfied.

2) The vector $\boldsymbol{v}$ contains points in a rectangular grid. Some of them do not belong to the domain, and for those, $v_k = 0$. This can be written in the matrix form by filling the k-th row of the matrix with 0:s. In the matrix eigenvalue equation $\boldsymbol{M}\boldsymbol{v} = K\boldsymbol{v}$, the k-th row will then be $0 = Kv_k$, forcing $v_k = 0$ (or $K = 0$, and these eigenvalues we can ignore).

3.2 Schrödinger's equation

In quantum mechanics, eigenvalue problems are important. The time-independent Schrödinger equation is actually an eigenvalue problem. For a single particle in a potential $V(x)$:

$$-\hbar^2 \frac{\nabla^2}{2m}\Psi(x) + V(x)\Psi(x) = E\Psi(x) \tag{25}$$

A solution $\Psi(x)$ to the time-independent Schrödinger equation is an allowed state of the system, and E gives the corresponding energy. The wave function is in general a complex quantity, but the eigenfunctions can be assumed to be real, which simplifies the calculations here a little.

To avoid tiny numbers and units, you can express E in units of $m/\hbar^2$ or set $m = 1$, $\hbar = 1$..

S. *(Optional, 2 points)* Consider the following simple one-dimensional potentials, and find a few of the lowest eigenstates and their energies, by discretizing the equation and finding the eigenvalues and -vectors of the discretized version.

i. an infinite potential well – analytically solvable

$$V(x) = 0 \text{ if } -a \leq x \leq a, \text{ infinite otherwise} \tag{26}$$

Hint: make your system extend from $-a$ to a, with $\Psi = 0$ on the boundary. The infinite potential is difficult to handle numerically otherwise.

ii. a finite potential well – almost analytically solvable

$$V(x) = 0 \text{ if } -a \leq x \leq a, \text{ otherwise a constant } V_0 \tag{27}$$

iii. a parabolic potential well – analytically solvable

$$V(x) = bx^2 \tag{28}$$

Assume $\Psi(x) = 0$ at the boundaries of the system, this will make the system closed, as if there were infinite potential barriers there, preventing the particle from escaping the system. Plot $|\Psi(x)^2|$, which gives the probability distribution of the particle, together with the potential. The results for potentials 1 and 3 can most easily be compared with the analytically known results.

3.3 Direct methods for solving steady state problems

In this exercise we will again find the steady state of the diffusion equation. In the previous exercises it was done using *iterative* methods. Now direct methods will be used, where a matrix for the ∇^2 is constructed, a matrix equation of the type $\boldsymbol{Mc} = \boldsymbol{b}$ is formed, and solved with standard library methods.

The domain is a circular disk with radius 2, centered on the origin. On and beyond the boundary of the disk, the concentration $c = 0$. At the point $(0.6, 1.2)$ a source is present, so that the concentration there is 1.

T. *(3 points)* Find the steady state concentration $c(x, y)$ by discretizing the diffusion equation as mentioned above. Plot the solution.

U. *(1 point)* Explain carefully how the matrix $\boldsymbol{M}$ and the vector $\boldsymbol{b}$ are constructed, and how the boundary conditions are taken into account.

Note that the hints about the circular domain from the preceding exercise still apply, and that the matrix M there might be quite similar to the one needed here.

For solving the matrix equation, you can use `scipy.linalg.solve()`. For larger systems and very little additional effort, sparse matrix operations can be used instead, `scipy.sparse.linalg.spsolve()`. For even larger systems, iterative sparse matrix solvers may be needed.

See Heath for a discussion about direct vs iterative methods.

4 Appendix. Tool recommendations

Python
Python 2 or Python 3? If it's possible to choose, take version 3, since this is the future. In practice the differences are usually small. But watch out for integer division in Python 2, where $2/3 = 0$

If you use Python 2, but want the new behavior, use this line in the beginning:

```
from __future__ import division
from __future__ import print_function
```

Python Modules
matplotlib
numpy
scipy
json

Linux
Easy to install many programming tools, due to powerful package management. Learn to use the terminal and the package manager.

OSX
OSX is also built on a variant of unix, and it has a powerful terminal. The package manager `brew` can be used to install many of the free programs and programming tools available for Linux. The system Python is old, but don't change it, install a separate one instead. You can do this with `brew` or Anaconda.

A decent code editor
Features to demand: syntax highlighting, automatic indentation, possibly code completion.
emacs, vim – very powerful, somewhat steep learning curve.
sublime
notepad++ (Windows)
gedit / kate

IDEs
An alternative to a text editor is a full development environment.
eclipse
code::blocks
XCode (OSX)

git for version control

Inkscape for drawing figures

References

An introduction to Numpy and Scipy:
http://www.engr.ucsb.edu/~shell/che210d/numpy.pdf

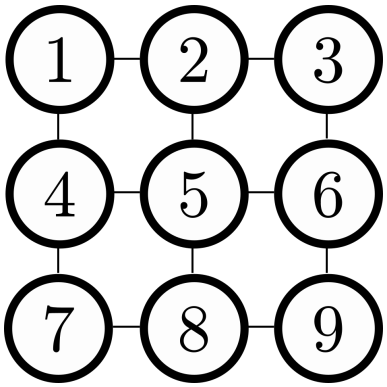
Python Scientific Lecture Notes
http://scipy-lectures.github.io/index.html

Scipy getting started
http://www.scipy.org/getting-started.html

Matplotlib FAQ
http://matplotlib.org/faq/usage_faq.html

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, chapter 11.



$$\begin{pmatrix} -2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -3 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -3 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -2 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \end{pmatrix} \tag{29}$$