Scientific Computing I

Exploring the SciCoPyth's mind

Jaro Camphuijsen (6042473) and Rahiel Kasim (10447539)

April 13, 2016

Contents

1 Tools

Rahiel: I've been using Linux for many years now. My favourite programming tool (and favourite software all-round) is GNU Emacs. It is a very advanced text editor. You can better understand Emacs if you think of it as a Lisp virtual machine with many functions for text editing. This means that any editing operation is actually a lisp function, so you can use lisp itself to extend the editor in any way you want. You however don't have to do everything yourself because Emacs has a built-in package manager, with more than 3000 packages available. For newcomers I recommend using `www.spacemacs.org`, a community maintained Emacs configuration with a focus on Vim emulation.

Another tool that is more easily adopted in someone's existing workflow is z (`www.github.com/rupa/z/`). It tracks which directories you use most often so you can more easily jump to them. For example, if I type `z bio` it cd's my shell to `~/Dropbox/univ/Computational Biology`, and `z chrom` brings me to `~/Code/archiveror/chromium`.

I am going to become a better programmer simply by programming more and by studying more ways how people program to solve problems.

Jaro: I have used Linux in the past on some other machines however after two months in my Masters Computational Science of using Windows on my own laptop, Rahiel convinced me to switch to Linux. It has made my life easier from the very beginning, and with the text editor Emacs almost anything is

possible. It has a steeper learning curve then other text editors, but once you know the basic key combination the more complex operations become very easy and less time consuming.
For sharing our code Rahiel and I like to use GitHub [1], it keeps track of changes and you can find back older versions at any time. Another nice function of GitHub is adding so called "issues" to your repository, this you can keep track of bugs and it also gives you a to do list towards the end of your project.
For reporting we like to use LaTeX because of its easy and clean layout generation, it automatically places figures and tables at good locations and keeps track of the reference numbers. LaTeX files can also be shared on GitHub, however better tools exist like `www.sharelatex.com` and `www.overleaf.com`, where you can edit your online LaTeX files interactively.

2 Vibrating String

The time evolution of a vibrating string is given by the partial differential equation 1

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

To get a unique solution for $\Psi(x,t)$ we have to declare boundary and initial conditions.

$$\Psi(x,0) = f(x) \qquad \Psi(0,t) = 0$$
$$\Psi'(x,0) = 0 \qquad \Psi(L,t) = 0$$

For the initial position of the string we use three different cases:

$$f(x) = \sin(2\pi x) \tag{2a}$$
$$f(x) = \sin(5\pi x) \tag{2b}$$
$$f(x) = \sin(5\pi x) \qquad \text{if } \frac{1}{5} \leq x \leq \frac{2}{5}, \text{ else } \Psi = 0 \tag{2c}$$

Only in a few cases can this PDE be solved analytically, hence we would like to construct a numerical model. To do this we discretize the continuous problem. For time step k and spatial step i we can write for the partial second derivatives of Ψ:

$$\frac{\partial^2 \Psi}{\partial x^2} = \frac{u_{i+1}^k - 2u_i^k + u_{i-1}^k}{\Delta x^2} \tag{3a}$$
$$\frac{\partial^2 \Psi}{\partial t^2} = \frac{u_i^{k+1} - 2u_i^k + u_i^{k-1}}{\Delta t^2} \tag{3b}$$

Now we can solve the discrete (1D) wave equation for u_i^{k+1} so we can compute the new value of each spatial bin i from the current and previous time steps and its surrounding bins [2]:

$$u_i^{k+1} = 2u_i^k - u_i^{k-1} + c\left(\frac{\Delta t}{\Delta x}\right)^2 \left(u_{i+1}^k - 2u_i^k + u_{i-1}^k\right), \qquad i = 1, ..., n \tag{4}$$

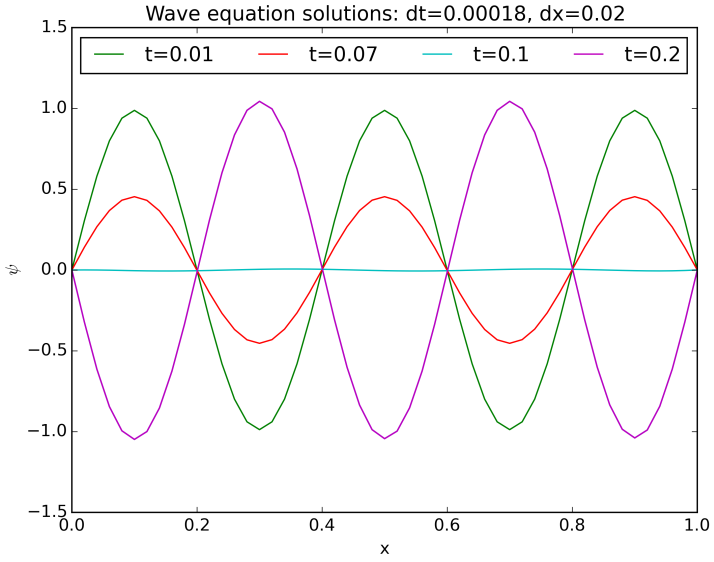

Figure 1: Solutions to the wave equation calculated with the finite difference method described in the text.

We've implemented this scheme and in figure 1 we see solutions to the wave equation with the initial conditions of eq. 2b. We also have animations that better elucidate the time evolution and the inevitable numerical instability.

3 The Time Dependent Diffusion Equation

$$c_{i,j}^{k+1} = c_{i,j}^{k} + \frac{\Delta t}{\Delta x^2}\left(u_{i+1,j}^{k} + u_{i-1,j}^{k} + u_{i,j+1}^{k} + u_{i,j+1}^{k}\right), \qquad i = 1, ..., n \tag{5}$$

At the boundaries of the x domain from we use the following function values to make periodic boundary conditions:

$$u_{N+1,j}^{k} = u_{0,j}^{k} \tag{6a}$$
$$u_{-1,j}^{k} = u_{N,j}^{k} \tag{6b}$$

for the boundaries of the y domain we lock the values of $u_{i,0}^{k}$ and $u_{i,N}^{k}$ to respectively 1 and 0. We then let the simulation only generate values in the y domain {1, N-1 }.

The simulation was done with N=30 spatial bins in both the x and y direction, the value of the time step was determined by the stability constraint:

$$\Delta t = 0.9\frac{\Delta x^2}{4D} \tag{7}$$

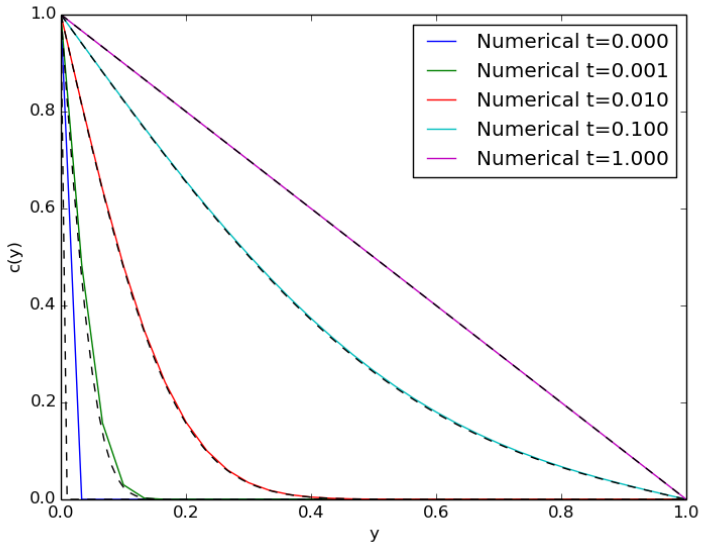

Figure 2: Comparison of the analytic solution (dashed line) and the numerical model at several points in time for the time dependent diffusion equation

For this specific configuration we can compute the analytic solution:

$$c(x,t) = 1 - \sum_{i=1}^{\inf} \left[\text{erfc}\left(\frac{1-x+2i}{2\sqrt{Dt}}\right) - \text{erfc}\left(\frac{1+x+2i}{2\sqrt{Dt}}\right) \right] \tag{8}$$

And so we can compare the analytic solution with our model. The result for at different points in time can be seen in figure 2 where we dispose of the x dimension. This can be done without loss of information since u is constant along the x dimension, due to the configuration of this simulation. The full two dimensional outcome can be found in figure 3 where heatmaps are drawn for the same points in time as in figure 2. In addition an animated plot was made which can be viewed by running the code.

4 The Time Independent Diffusion Equation

We've implemented the Jacobi iteration, Gauss-Seidel method and SOR with $N = 50$ and a tolerance of $\epsilon = 10^{-5}$. In figure 4 we compare the results of the three methods with the analytic solution. We see that they all come very close to the analytic solution: the result of SOR is superimposed on the analytic solution ($\lim_{t\to\infty} c(y,t) = -y$), while the Gauss-Seidel is further down and the Jacobi iteration is still below that. The nice thing is that this is also the same order for efficiency of the methods, SOR is both the most efficient (least amount of iterations) and the most accurate (closest to the analytic solution).

We can further investigate the efficiency of the methods we've implemented by looking at the convergence measure δ versus the number of iterations k. In

figure 5 we see this relation for the three methods, we again see that the SOR method is the most efficient as it converges at the lowest number of iterations.

We can find the optimal ω for the SOR method by doing the SOR computation with different values for ω between the specified range while looking for the ω that needs the least amount of iterations. One can find ω naively by trying all values, or by doing a (binary) search over the range. We found as optimal $\omega = 1.891$. We can also look at the optimal ω for different values of N, using the same method we found that smaller values of N have a smaller optimal ω (for $N = 40$ we found $\omega = 1.87$) and for larger N we also find larger ω (for $N = 60$ we found $\omega = 1.91$).

We put a 10 x 10 square in the middle of the domain and find that the convergence is faster, it took 204 iterations and without an object it takes 250 iterations. We find a slower convergence of 224 iterations if we put two 5 x 10 rectangles in the domain. Using the same method to find the optimal ω we find with the 10 x 10 square the optimum $\omega = 1.87$ with 159 iterations, so adding an object slightly lowered the optimal ω. For the two 5 x 10 rectangles we find an optimum of $\omega = 1.89$ with 222 iterations, so here the object very slightly lowered the optimal.

For these two examples we found that adding objects lowered the iterations and may also lower the optimal ω value.

References

[1] Jaro Camphuijsen, Rahiel Kasim. Sunsistemo on GitHub. [`https://github.com/sunsistemo/sunsistemo`; accessed 12-April-2016].

[2] Micheal T. Heath. *Scientific Computing: An Introductory Survey*. Elizabeth A. Jones, 2nd edition, 2002.

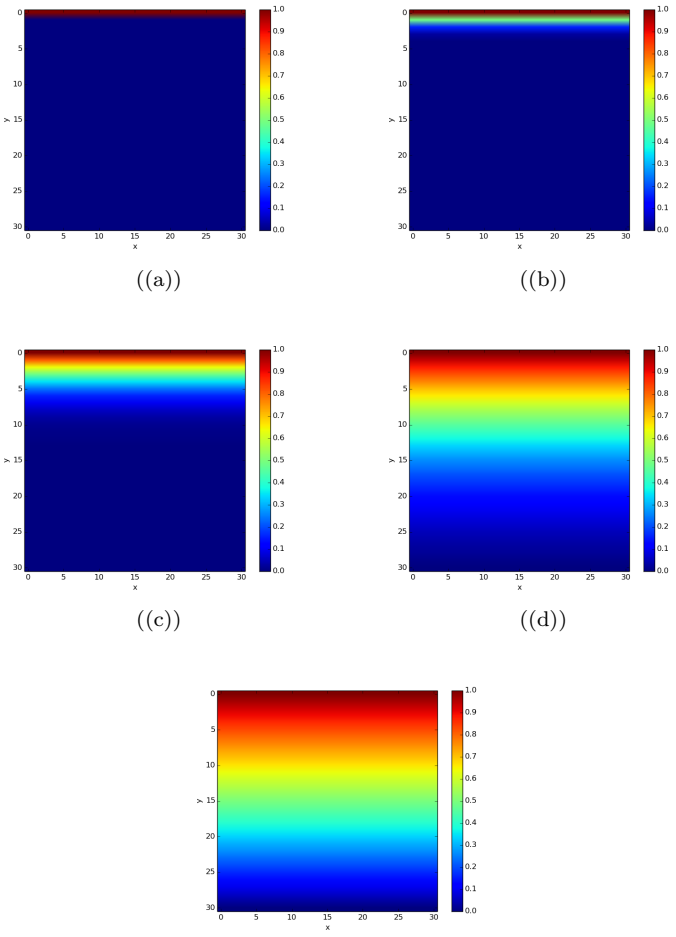

Figure 3: Several stages in the time dependent diffusion simulation in a 2 dimensional heatmap, the plots correspond to respectively t = [0, 0.001, 0.01, 0.1, 1]

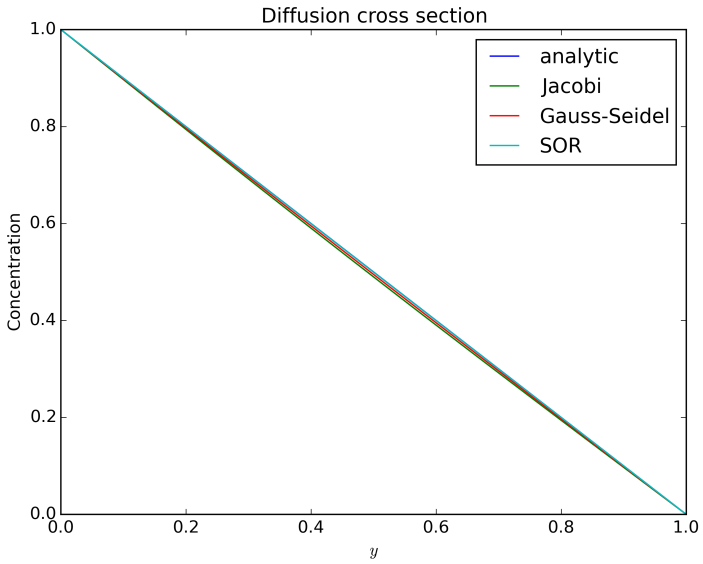

Figure 4: A cross section of the solution of the time independent diffusion equation for different methods..

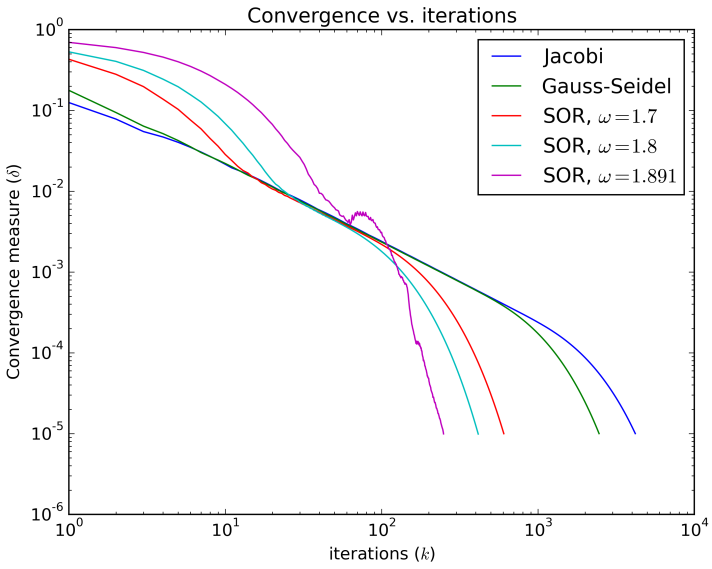

Figure 5: The convergence measure vs. the number of iterations for the methods solving the time independent diffusion equation. We see that the SOR method converges with the least amount of iterations.