NUMERICAL METHODS FOR PARTIAL DIFFERENTIAL EQUATIONS

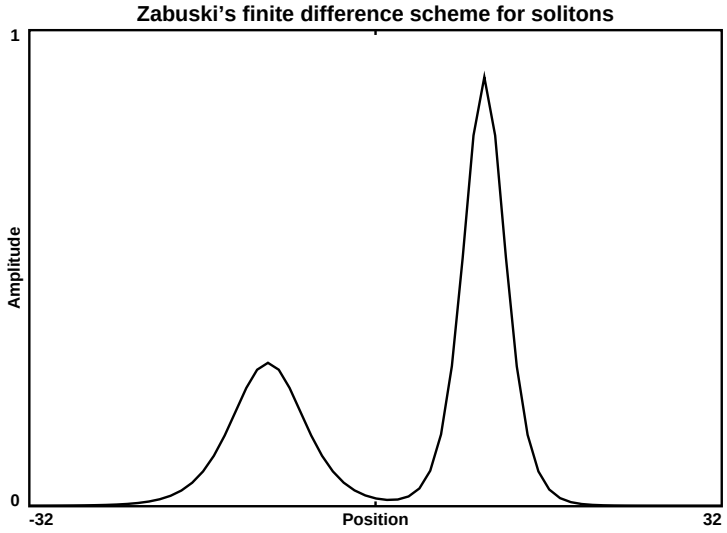
http://pde.fusion.kth.se

André Jaun <jaun@fusion.kth.se>

September 10, 2001

with the participation of

Thomas Johnson <thomasj@fusion.kth.se>
Thomas Hürtig <hurtig@fusion.kth.se>
Thomas Rylander <rylander@elmagn.chalmers.se>
Mikael Persson <elfmp@elmagn.chalmers.se>
Laurent Villard <Laurent.Villard@epfl.ch>



2001 course of the Summer University of Southern Stockholm, held simultaneously at the
Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
Chalmers Institute of Technology, Göteborg, Sweden
Royal Institute of Technology, Stockholm, Sweden

This document is typeset with LaTeX 2ε[1] using the macros in the `latex2html`[2] package on Sun[3] and Linux[4] platforms.

[1] http://www.tug.org/
[2] http://cdc-server.cdc.informatik.tu-darmstadt.de/%7Elatex2html
[3] http://www.sun.com
[4] http://www.linux.org

Contents

PREFACE

This is the syllabus of the course taught at the Royal Institute of Technology (KTH, Stockholm) to graduate students in physics, engineering and quantitative social sciences. With the development of collaborative teaching and distance learning, the material is now also shared with the Chalmers (CTH, Göteborg) and the Swiss Federal Institutes of Technology (EPFL, Lausanne), the University of Perugia (Perugia, Italy) and independent learners from the Internet.

Recognizing the value of an introductory level text describing a whole range of numerical methods for partial differential equations (PDEs) with practical examples, a *virtual learning laboratory* has been developed around this highly interactive document [5] . Every problem is exposed all the way from the formulation of the master equation, the discretization resulting in a computational scheme, to the actual implementation with hyper-links into the JAVA source code. The JBONE applet executes every scheme with edit-able parameters and initial conditions that can be modified by the user. Comparisons of different methods show advantages and drawbacks that are generally exposed separately in advanced and specialized books. The complete source of the program can be obtained free of charge for personal use after registration.

For this second web edition accessible to everyone on the internet, I would like to thank Kurt Appert (EPFL, Lausanne) for the inspiration he provided when I was a student of his own course *Expérimentation Numérique*. Johan Carlsson, Johan Hedin and Thomas Johnson (KTH, Stockholm) have one after the other been responsible for the Monte-Carlo method, Johan Hedin providing valuable advice for the development of the educational technology. Ambrogio Fasoli (MIT, Cambridge), with whom I have the pleasure to collaborate in fusion energy research (by comparing numerical solutions of PDEs with actual experiments), provided the measurement of an Alfvén instability, showing the importance of aliasing in the digital acquisition of data.

I hope that this learning environement will be useful to you; I will consider my task more than satisfactorily accomplished, if, by the end of the course, you will not be tempted to paraphrase Oscar Wilde's famous review: "Good in parts, and original in parts; unfortunately the good part were not original, and the original parts were not good".

André JAUN, Stockholm, 1997–2001

[5] accessible with a java-powered web browser from `http://pde.fusion.kth.se`

1 INTRODUCTION

1.1 How to study this course

Studying is fundamentally an individual process, where every student has to find out himself what is the most efficient method to *understand* and *assimilate* new concepts. Our experience however shows that major steps are taken when a theory exposed by the teacher (in a regular classroom- or a video-lecture or a syllabus) is first reviewed, questionned, discussed with peers (best accross the table during a coffee break, alternatively during video-conferences or in user forums) and later applied to solve practical problems.

The educational tools that have been developed for this course reflect this pedagogical understanding: they are meant to be combined in the order / manner that is most appropriate for each individual learner. Along the same line, the assessment of the progress is carried out with a system of bonus points. They reward original contribution in the user forum as well as the assignments that have been sucessfully carried out.

An example showing how you could study the material during a typical day of the intensive summer course involves three distinct phases.

Passive learning (1h). This is when new concepts are first brought to you and you have to follow the teacher's line of thought. You may combine

- **Video-Lecture.** From the course main page[6], select *COURSE: video-lecture*. Download the video file once for all to your local disk (press SHIFT + select link) to enable you scrolling back and forth in the lecture. Use the labels on top of the monitor for synchronization with the syllabus. Open your Real player next or on the top of the web browser to use both simultaneously (Windows users: select *Always on top when playing*).
- **Syllabus.** Select the *COURSE: notes* where you can execute the applets discussed in the lectures. You may prefer reading the equivalent paper edition that can be downloaded in *PDF* or *Postsript* format and sent to your local printer.

Active learning (2h). After a first overview, you are meant to question the validity of new concepts, verify the calculations and test the parametric dependencies in the JBONE applet.

- **Syllabus.** Repeat the analytical derivations, which are on purpose scarce to force you to complete the intermediate steps by yourself. Answer the quiz questions and perform numerical experiments on the web. The original applet parameters can always be recovered and the applet restarted (click RIGHT in the white area + press SHIFT + select *Reload*).
- **Video-Conference.** Rather than a lecture, the video-conference is really meant to discuss and refine the *understanding you have previously acquired* in the passive phase. An opportunity for everyone not only to ask, but also to answer and comment the questions from peers.
- **User Forum.** Regular students select the *classroom* forum to obtain advice helping the understanding of new material. Don't hesitate to discuss related

[6] http://pde.fusion.kth.se

topics and comment the answers of your classmates: we judge this virtual classroom activity sufficiently important that your participation is rewarded with bonus points (independently on whether your arguments are correct or not — it is the teachers' duty to correct potential errors). Consult the *Forum: rules* and try make your contributions beneficial for all the learners.

Problem based learning (5h). Most important after acquiring a new skill is to test and practice it by solving practical problems. Select *USER: login* to enter into your personal web account. A whole list of problems appears under *assignments*; each can be edited in your browser simply by clicking on the *identification number*. Below the handout, three input windows allow you to submit your solution to the compilers:

- **TeX.** The first window takes regular text (ASCII) and LaTeXinput, allowing you to write text and formulas (*symbols* inserted between two dollar signs, such as `$\sin^2\alpha + \cos^2\alpha = 1$`, will appear as regular algebra in your web browser). Explain how you derive your numerical scheme, how you implement it and analyse the numerical output. Users who are not familiar with LaTeXgenerally find it easy to first only modify the templates and use the *symbols* dictionary appearing on the top of every input window.
- **JAVA.** The content of the second window is compiled into the JBONE applet, allowing you to execute your own schemes locally in your browser. Careful: be sure to correct all compiler errors before you switch to a new exercise — or your applet will stop working everywhere! Also, your web browser may store old data in your cash; you may have to force it to reload the new compiled version (again, click RIGHT in the white area + press SHIFT + select *Reload*). If you don't get immediate programming advice from the *User Forum*, you can temporarily deactivate a problematic scheme using the `/* java comment delimiters */`.
- **Parameters.** Choose defaults values that exploit the strength of your scheme, but remain compatible with the numerical limitations.

Be sure to *submit only one input window* at a time and please do always compile your work before you navigate further to the syllabus or the user Forum. Sometimes, the *Back* button of your browser restores lost data... but don't count on it! When your solution is ready and documented, click on the *CheckMe* button (appears only after the first compilation) and select *Submit Check* (bottom of the list) to send your assignment for correction to the teachers. Take into account the modifications that are suggested until your solution is accepted and the exercise is signalled as passed.

The amount of work in each chapter is sufficiently large that it is not possible to complete all the requirements directly within two intensive weeks; rather than proceeding sequentially chapter by chapter, it is important that you *start at least one assignment* before every topic is discussed in the video-conference. Remember, this is not a lecture and is rather useless if you did not study the material before!

Individual work (1 week). One week is reserved to complete the assignments that could not be finished in time during the first two weeks. Combine the learning tools until you meet the *course requirements*. Please submit your evaluation of the course as we keep improving it in the future.

Project (1 week). Regular students are given an opportunity to apply their newly acquired skills in a topic that could be of interest for their own research. Part of the intention is to reward taking a risk (stricktly limited to one week), to assess whether one of the tools could potentially result in an useful development in the frame of a PhD thesis. A small report with no more than four A4 pages will be published under the main course web page.

1.2 Differential Equations

1.2.1 Ordinary differential equations

Ordinary differential equations (ODE) are often encountered when dealing with initial value problems. In Newtonian mechanics, for example, the trajectory of a particle is described by a set of *first order* equations in time:

$$\frac{d^2\mathbf{X}}{dt^2} = \frac{F}{m} \quad \Longleftrightarrow \quad \frac{d}{dt}\begin{pmatrix} \mathbf{X} \\ \mathbf{V} \end{pmatrix} = \begin{pmatrix} \mathbf{V} \\ \mathbf{F}/m \end{pmatrix}$$

This example shows explicitly how higher order equations can be recast into a system of first order equations, with components of the form

$$y' = f(t, y), \qquad y(0) = y_0 \tag{1.2.1\#eq.1}$$

Under very general assumptions, an initial condition $y(0) = y_0$ yields exactly one solution to Eq. (1.2.1#eq.1), which can be approximated with a digital computer. Rather than developing the details of elementary numerical analysis, this section is meant only to identify the problem and review the elementary solutions that will be used later in the Monte-Carlo method for partial differential equations.

Introducing a discretization $y_n = y(t_n)$ with a finite number n of time steps $t_n = n\Delta t + t_0$, a straight forward manner to solve an ODE is to approximate the derivative with a finite difference quotient $(y_{n+1} - y_n)/h$, which leads to the *Euler recursion formula*

$$y_{n+1} = y_n + hf(t_n, y_n), \qquad y_0 = y(t_0) \tag{1.2.1\#eq.2}$$

Because all the quantities are known at time t_n, the scheme is called *explicit*. An *implicit* evaluation of the function $f(t_{n+1}, y_{n+1})$ is sometimes desirable in order to stabilize the propagation of numerical errors in $\mathcal{O}(h)$; this is however computationally expensive when the function cannot be inverted analytically.

More precision is granted with a symmetric method to evaluate the derivative, using a Taylor expansion that involves only even powers of h. This achieved in the so-called *midpoint formula*

$$y_{n+1} = y_{n-1} + 2hf(t_n, y_n), \qquad y_{-1} = y_0 - hf(t_0, y_0) \tag{1.2.1\#eq.3}$$

It is accurate to $\mathcal{O}(h^2)$, but requires a special initialization to generate the additional values that are needed from the past. Writing the mid-point rule as

$$y_{n+1} = y_n + hf\left(t_n + \frac{h}{2}, y(t_n + \frac{h}{2})\right)$$

this initialization problem is cured in the *second order Runge-Kutta* method, by using an Euler extrapolation for the intermediate step

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + \frac{h}{2}, y_n + \frac{1}{2}k_1) \\ y_{n+1} &= y_n + k_2 \end{aligned} \qquad (1.2.1\#\text{eq.4})$$

Such elementary and more sophisticated methods are commonly available in software packages such as MATLAB. They are usually rather robust, but become extremely inefficient when the problem is *stiff* (exercise 1.1) — i.e. involves two very different spatial scales, that limit the step size to the shorter one even if it is not a priori relevant.

1.2.2 Partial differential equations

Partial differential equations (PDE) involve at least 2 variables in space (*boundary value problems*) or / and time (*initial value problems*):

$$A\frac{\partial^2 f}{\partial t^2} + 2B\frac{\partial^2 f}{\partial t \partial x} + C\frac{\partial^2 f}{\partial x^2} + D(t, x, \frac{\partial f}{\partial t}, \frac{\partial f}{\partial x}) = 0 \qquad (1.2.2\#\text{eq.1})$$

It is of *second order* in the derivatives acting on the unknown f, it is *linear* if D is linear in f, and is *homogenous* if all the terms in D depend on f.

Initial value problems with several spatial dimensions and / or involving a combination of linear and non-linear operators

$$\frac{\partial f}{\partial t} = \mathcal{L}f \qquad (1.2.2\#\text{eq.2})$$

can sometimes be solved with the so-called *operator splitting*. The idea, based on the standard separation of variable, is to decompose the operator in a sum of m pieces $\mathcal{L}f = \mathcal{L}_1 f + \mathcal{L}_2 f + \ldots + \mathcal{L}_m f$ and to solve the whole problem in sub-steps, where each part of the operator is evolved separately while keeping all the others fixed. One particularly popular splitting for is the alternating-direction implicit (ADI) method, where only one spatial dimension is evolved at any time using an implicit scheme. Non-linearities can equally be split from the linear operators and treated with an appropriate technique.

1.2.3 Boundary / initial conditions

Depending on the problem, initial conditions (IC)

$$f(x, t = 0) = f_0(x)\ ,\ \forall x \in \Omega \qquad (1.2.3\#\text{eq.1})$$

and / or boundary conditions (BC) need to be imposed. The latter are often of the form

$$af + b\frac{\partial f}{\partial x} = c\ ,\ \forall x \in \partial\Omega\ \forall t \qquad (1.2.3\#\text{eq.2})$$

and are called *Dirichlet (b=0)*, *Neumann (a=0)*, or *Robin (c=0)* conditions. Other forms include the *periodic* condition $f(x_L) = f(x_R)$, where $\{x_L, x_R\} \in \partial\Omega\ \forall t$, and the *outgoing-wave* conditions if the domain is open. Note that to prevent reflections from the computational boundary of an open domain, it can be useful to introduce *absorbing boundary conditions*: they consist in a small layer of an artificial material distributed over a few mesh cells that absorb outgoing waves. The *perfectly matched layers* [1] are often used in problems involving electromagnetic waves.

1.2.4 Characteristics and dispersion relations

The characteristics of a PDE can loosly be defined as the trajectories $x(t)$ along which discontinuities and the initial conditions propagate: think of the path a heat pulse takes in an inhomogeneous material. The chain rule can be used more formally to classify second order equations ((1.2.2#eq.1) with D=0) with the ansatz $f(x,t) = f(x(t),t)$

$$\frac{\partial^2 f}{\partial x^2}\left[A\left(\frac{\partial x}{\partial t}\right)^2 + 2B\frac{\partial x}{\partial t} + C\right] = 0 \implies \frac{\partial x}{\partial t} = A^{-1}\left(-B \pm \sqrt{B^2 - AC}\right) \quad (1.2.4\#\text{eq.1})$$

into three categories depending on the sign of the discriminant: if

- $B^2 - AC < 0$ the equation has no characteristic and is called *elliptic* (Laplace eq.)
- $B^2 - AC = 0$ the equation has one characteristic and is called *parabolic* (heat eq.)
- $B^2 - AC > 0$ the equation has two characteristics and is called *hyperbolic* (wave eq.)

The characteristics play an important role by themself and will be exploited numerically in the so-called *Lagrangian methods* in sect.6.

The local properties of a linear equation can be analyzed with a harmonic ansatz $f(t,x) = f_0 \exp(-i\omega t + ikx)$, transforming the differential operators in (t,x) into algebraic expressions in (ω,k) called the *dispersion or stability relation*:

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial t} = 0 \implies -\omega + uk = 0 \quad (1.2.4\#\text{eq.2})$$

Dispersion relations relate the phase velocity $\Re e(\omega)/k$ or the growth rate $\Im m(\omega)/k$ to the scale of the solution k. Assuming moreover a homogeneous grid in space $x_j = j\Delta x$ and time $t_n = n\Delta t$, it is possible to assess the quality of a numerical approximation as a function of the spatial $k\Delta x$ or temporal resolutions ωt. Take for example the wavenumber $k_{\text{eff}} = -if'/f$ that is obtained when the mid-point rule (1.2.1#eq.3) is used to approximate the first derivative of $f(x) = \exp(ikx)$:

$$k_{\text{eff}} = -i\frac{\exp(ik\Delta x) - \exp(-ik\Delta x)}{2\Delta x} = k\frac{\sin(k\Delta x)}{k\Delta x} \quad (1.2.4\#\text{eq.3})$$

The wave number is under-estimated for poor resolution $k\Delta x \to \pi$ and even changes sign with less than two mesh points per wavelenth. The forward difference (1.2.1#eq.2)

$$k_{\text{eff}} = -i\frac{\exp(ik\Delta x) - 1}{\Delta x} = k\frac{\sin(\frac{k\Delta x}{2})}{\frac{k\Delta x}{2}}\left[\cos\left(\frac{k\Delta x}{2}\right) + i\sin\left(\frac{k\Delta x}{2}\right)\right] \quad (1.2.4\#\text{eq.4})$$

has an additional imaginary part responsible for a damping $f \propto \exp[-\Im m(k_{\text{eff}})\Delta x]$.

1.2.5 Moments & conservation laws

Differential calculus is at the heart of science and engineering because it describes the interactions *locally*, relating infinitesimal changes at the microscopic scale to the macroscopic scale of a system. At the macroscopic scale, *global* quantities can be found that remain constant despite these microscopic changes: the total density, momentum and the energy in a closed box isolated from the outside world do not change. Such conservation

laws can in general be constructed by taking *moments* in phase space x, where the moment of order K a function $f(x)$ is defined by

$$\mathcal{M}_K = \int_\Omega dV x^K f \qquad (1.2.5\#\text{eq.1})$$

Usually, $\mathcal{M}_0$ is the total density, $\mathcal{M}_1$ the total momentum and $\mathcal{M}_2$ the total energy in the system. Conservation laws provide useful *self-consistency checks* when PDEs are solved in an approximate manner with the computer: deviations from the initial value can be used to monitor the quality of a numerical solution as it evolves in time.

1.3 Prototype problems

Here are some of the most important initial value problems /equations, illustrated in a 1D periodic slab $x \in [0; L[$.

1.3.1 Advection

Also called *convection*, advection models the streaming of infinitesimal element in a fluid. It generally appears when a transport process is modeled in a Eulerian representation through the convective derivative

$$\frac{d}{dt}f \equiv \frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} = 0 \qquad (1.3.1\#\text{eq.1})$$

For a constant advection velocity u, this can be solved analytically as $f(x,t) = f_0(x-ut)$ $\forall f_0 \in \mathcal{C}(\Omega)$, showing explicitly the characteristic $x = ut$. Use the applet [7] to compute the evolution of a Gaussian pulse with an advection computed using the Lagrangian CIP/FEM method from sect.6. For a constant advection speed, the wave equation can be written in flux-conservative form

$$\frac{\partial^2 h}{\partial t^2} - u^2\frac{\partial^2 h}{\partial x^2} = 0 \iff \frac{\partial}{\partial t}\begin{pmatrix} f \\ g \end{pmatrix} - \frac{\partial}{\partial x}\left[\begin{pmatrix} 0 & u \\ u & 0 \end{pmatrix}\cdot\begin{pmatrix} f \\ g \end{pmatrix}\right] = 0 \qquad (1.3.1\#\text{eq.2})$$

suggesting how the advection schemes may be generalized for wave problems.

1.3.2 Diffusion

At the microscopic scale, *diffusion* is related to a *random walk* and leads to the prototype equation (exercise 1.4)

$$\frac{\partial f}{\partial t} - D\frac{\partial^2 f}{\partial x^2} = 0 \qquad (1.3.2\#\text{eq.1})$$

where $D \geq 0$ is the so-called diffusion coefficient. For a homogeneous medium, the combined advection-diffusion equation

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} - D\frac{\partial^2 f}{\partial x^2} = 0 \qquad (1.3.2\#\text{eq.2})$$

[7] in the electronic version of the lecture notes `http://pde.fusion.kth.se`

can be solved analytically in terms of a *Green's function* (exercise 1.3)

$$\begin{aligned} f(x,t) &= \int_{-\infty}^{+\infty} f_0(x_0) G(x - x_0 - ut, t) dx_0 \\ G(x - x_0 - ut, t) &= \frac{1}{\sqrt{\pi 4Dt}} \exp\left(-\frac{(x - x_0 - ut)^2}{4Dt}\right) \end{aligned} \qquad (1.3.2\#\text{eq.3})$$

A numerical solution is generally required to solve the equation in an inhomogeneous medium, where $u(x)$, $D(x)$:

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} - \frac{\partial}{\partial x}\left(D\frac{\partial f}{\partial x}\right) = 0$$

Check the document on-line for an example of a numerical solution describing the advection-diffusion of a box computed with the finite element method from sect.3.

A harmonic ansatz in space and time $f \sim \exp i(kx - \omega t)$ yields the dispersion relation

$$\omega = -iDk^2 \qquad (1.3.2\#\text{eq.4})$$

and shows that short wavelengths (large k) are more strongly damped than long wavelengths. Change the initial conditions to `Cosine` and model this numerically in the applet by altering the wavelength.

1.3.3 Dispersion

Dispersion occurs when different wavelengths propagate with different phase velocities. Take for example the third order dispersion equation

$$\frac{\partial f}{\partial t} - \frac{\partial^3 f}{\partial x^3} = 0 \qquad (1.3.3\#\text{eq.1})$$

A harmonic ansatz $f \sim \exp i(kx - \omega t)$ yields the phase velocity

$$-if(\omega - k^3) = 0 \quad \Longrightarrow \quad \frac{\omega}{k} = k^2 \qquad (1.3.3\#\text{eq.2})$$

showing that short wavelengths (large k) propagate faster than long wavelengths (small k). In the Korteweg-DeVries equation below, this explains why large amplitude *solitons* with shorter wavelengths propagate more rapidly than low amplitudes solitons.

1.3.4 Wave-breaking

Wave-breaking is a non-linearity that can be particularly nicely understood when surfing at the see shore... where the shallow water steepens the waves until they break. It can also be understood theoretically from the advection equation with a velocity proportional to the amplitude $u = f$:

$$\frac{\partial f}{\partial t} + f\frac{\partial f}{\partial x} = 0 \qquad (1.3.4\#\text{eq.1})$$

As a local maximum (large f) propagates faster than a local minimum (small f) both will eventually meet and and the function becomes multi-valued, causing the wave (and our

numerical schemes) to break. Sometimes, this wave-breaking is balanced by a competing mechanism. This is the case in the *Burger* equation for *shock-waves*

$$\frac{\partial f}{\partial t} + f\frac{\partial f}{\partial x} - D\frac{\partial^2 f}{\partial x^2} = 0 \qquad (1.3.4\#\text{eq.2})$$

where the creation of a shock front (with short wavelengths) is physically limited by a diffusion — damping the short wavelengths (1.3.2#eq.4). Check the document on-line to see a shock formation computed using a 2-levels explicit finite difference scheme from sect.2.

Another example, where the wave-breaking is balanced by dispersion leads to the *Korteweg-DeVries* equation for *solitons*

$$\frac{\partial f}{\partial t} + f\frac{\partial f}{\partial x} + b\frac{\partial^3 f}{\partial x^3} = 0 \qquad (1.3.4\#\text{eq.3})$$

The on-line document shows how large amplitudes solitons (short wavelengths) propagate faster than lower amplitudes (long wavelength).

1.3.5 Schrödinger

Choosing units where Planck's constant $\hbar = 1$ and the mass $m = 1/2$, its appears clearly that the time-dependent *Schrödinger* equation is special type of wave / diffusion equation:

$$i\frac{\partial \psi}{\partial t} = H\psi \qquad \text{with} \qquad H(x) = -\frac{\partial^2}{\partial x^2} + V(x) \qquad (1.3.5\#\text{eq.1})$$

In quantum mechanics, such a Schrödinger equation evolves the complex wave-function $|\psi >= \psi(x,t)$ that describes the probability $< \psi|\psi >= \int_\Omega |\psi|^2 dx$ of finding a particle in a given interval $\Omega = [x_L; x_R]$. Take the simplest example of a free particle modeled with a wave-packet in a periodic domain with a constant potential $V(x) = 0$. The JBONE applet on-line shows the evolution of a low enery $E = - < \psi|\partial_x^2|\psi >$ (long wavelength) particle initially known with a rather good accuracy in space (narrow Gaussian envelope): the wave-function $\Re e(\psi)$ (blue line) oscillates and the probability density $|\psi|^2$ quickly spreads out (black line) — reproducing Heisenberg's famous uncertainty principle.

1.4 Discretization

Differential equations are solved numerically by evolving a discrete set of values $\{f_j\}$, $j = 1, N$ and by taking small steps in time Δt to *approximate* what really should be a continuous function of space and time $f(x,t)$, $x \in [a;b]$, $t \geq t_0$. Unfortunately, there is *no universal method.* Rather than adopting the favorite of a "local guru", your choice for a discretization should really depend on

- the structure of the solution (continuity, regularity, precision), the post-processing (filters) and the diagnostics (Fourier spectrum) that are expensive but might be anyway required
- the boundary conditions, which can be difficult to implement depending on the method
- the structure of the differential operator (the formulation, the computational cost in memory×time, the numerical stability) and the computer architecture (vectorization, parallelization).

This course is a lot about advantages and limitations of different methods; it aims at giving you a broad knowledge, so that you to make the optimal choices and that allows you to pursue with relatively advanced literature when this becomes necessary.

1.4.1 Convergence

The most important aspect of a numerical approximation of a continuous function with a finite number of discrete values is that this approximation *converges* to the exact value as more information is gathered (exercise 1.5). This convergence can be defined *locally* for any arbitrary point in space / time (more restrictive) or by monitoring a *global* quantity (more permissive).

Take for example the function $f(x) = x^\alpha$ in $[0;1]$ discretized using a homogeneous mesh with N values measured in the middle of each interval at $x_n = (n+1/2)/N$. The lowest order approximation $f(0) \approx f^{(N)}(x_0) = (2N)^{-\alpha}$ converges at the origin provided $\alpha \geq 0$. The *convergence rate* can be estimated from a geometric sequence of approximations $\{f^{(i)}\}$, obtained by successively doubling the numerical resolution $i = N, 2N, 4N$

$$r = \frac{\ln\left[(f^{(N)} - f^{(2N)})/(f^{(2N)} - f^{(4N)})\right]}{\ln[2]} \qquad (1.4.1\#\text{eq.1})$$

For the example above, this gives a local convergence rate $r = \alpha$ when $\alpha > 0$. Using the mid-point rule (sect.3.3) $\int_0^1 f(x)dx \approx \sum_{n=0}^{N} x_n^\alpha \Delta x$ with $\Delta x = 1/N$, global convergence is achieved when $\alpha > -1$; because of the weak singularity, the rate however drops from the $\mathcal{O}[(\Delta x)^2]$ expected for "smooth" functions to $r \simeq 1.5, 0.5, 0.2$ when $\alpha = 0.5, -0.5, -0.8$.

1.4.2 Sampling on a grid

The main advantage of a sampling data on a grid is the *simplicity*: starting from a continuous function $f(x)$ a finite number of values $\{(x_j; f_j)\}$ are measured on what is generally a homogeneous mesh $x_j = j\Delta x$, $j = 1, N$. The values at the domain boundary appear explicitly as x_1 and x_N. Figure 1 shows that the sampled function is *unknown*

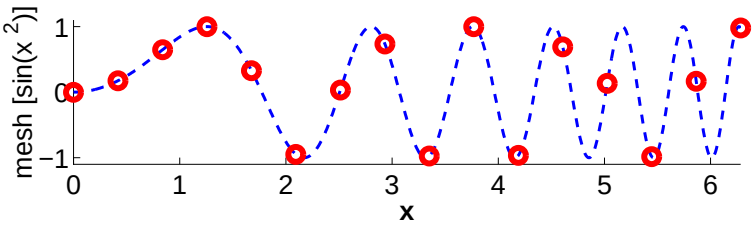


Figure 1: Approximation of $\sin(x^2)$ on a homogeneous mesh.

almost everywhere except on the grid points where it is exact. If the sampling is dense enough and the function sufficiently smooth, the intermediate values can be interpolated from neighbouring data using a Taylor expansion

$$f(x_0 + \Delta x) = f(x_0) + \Delta x f'(x_0) + \frac{\Delta x^2}{2} f''(x_0) + \mathcal{O}(\Delta x^3) \qquad (1.4.2\#\text{eq.1})$$

Derivatives are obtained by finite differencing from neighboring locations

$$\begin{aligned} f_{j+1} \equiv f(x_j + \Delta x) &= f_j + \Delta x f'_j + \frac{\Delta x^2}{2} f''_j + \dots \\ f_{j-1} \equiv f(x_j - \Delta x) &= f_j - \Delta x f'_j + \frac{\Delta x^2}{2} f''_j + \dots \end{aligned}$$

leading to the formulas for the k-th derivative $f^{(k)}$ from Abramowitz [2] §25.1.2

$$f_j^{(2n)} = \sum_{k=0}^{2n} (-1)^k \left(\begin{array}{c} 2n \\ k \end{array} \right) f_{j+n-k} \qquad (1.4.2\#\text{eq.2})$$

$$f_{j+1/2}^{(2n+1)} = \sum_{k=0}^{2n+1} (-1)^k \left(\begin{array}{c} 2n+1 \\ k \end{array} \right) f_{j+n+1-k} \qquad (1.4.2\#\text{eq.3})$$

$$\left(\begin{array}{c} n \\ k \end{array} \right) = \frac{n(n-1)\cdots(n-k+1)}{k!}$$

Note that a discretization on a grid is non-compact: the convergence depends not only on the initial discretisation, but also on the interpolation used *a posteriori* to reconstruct the data between the mesh points.

1.4.3 Finite elements

Following the spirit of Hilbert space methods, a function $f(x)$ is decomposed on a *complete* set of *nearly orthogonal* basis functions $e_j \in \mathcal{B}$

$$f(x) = \sum_{j=1}^{N} f_j e_j(x) \qquad (1.4.3\#\text{eq.1})$$

These finite-elements (FEM) basis functions span only as far as the neighboring mesh points; most common are the normalized "roof-tops"

$$e_j(x) = \left\{ \begin{array}{ll} (x - x_{j-1})/(x_j - x_{j-1}) & x \in [x_{j-1}; x_j] \\ (x_{j+1} - x)/(x_{j+1} - x_j) & x \in [x_j; x_{j+1}] \end{array} \right. \qquad (1.4.3\#\text{eq.2})$$

which yield a piecewise linear approximation for $f(x)$ and a piecewise constant derivative $f'(x)$ *defined almost everywhere* in the interval $[x_1; x_N]$.

Boundary conditions are incorporated by modifying the functional space $\mathcal{B}$ (e.g. taking "unilateral roofs" at the boundaries). Generalization with "piecewise constant" or higher order "quadratic" and "cubic" FEM is also possible. Important is the capability of *densifying the mesh* for a better resolution of short spatial scales. Figure 1 doesn't exploit this, but illustrates instead what happens when the numerical resolution becomes insufficient: around 20 linear (and 2 cubic) FEM per wavelength are usually required to achieve a precision of 1% and a minimum of 2 is of course necessary only to resolve the oscillation.

1.4.4 Splines

Given an approximation on an inhomogeneous mesh, the idea of splines is to provide a global *interpolation* which is continuous up to a certain derivative. Using a cubic polynomial with tabulated values for the function and second derivative, this is achieved with

$$f(x) = A f_j + B f_{j+1} + C f''_j + D f''_{j+1} \qquad (1.4.4\#\text{eq.1})$$

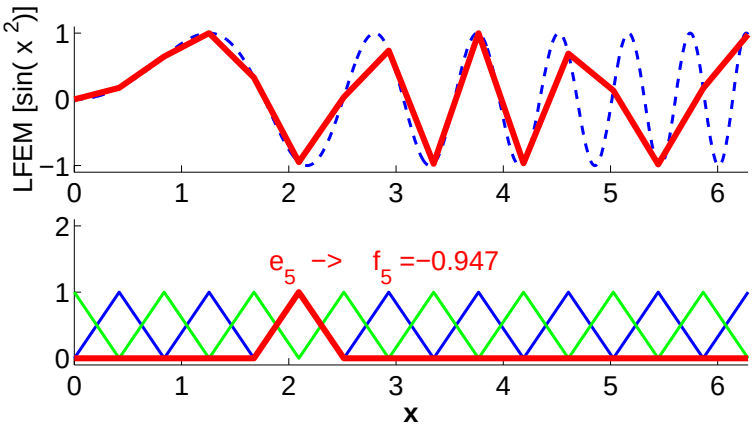


Figure 1: Approximation of $\sin(x^2)$ with roof-top linear finite elements.

$$\begin{aligned} A(x) &= (x_{j+1}-x)/(x_{j+1}-x_j) & B(x) &= 1-A \\ C(x) &= (x_{j+1}-x_j)^2(A^3-A)/6 & D(x) &= (x_{j+1}-x_j)^2(B^3-B)/6 \end{aligned}$$

from which it is straight forward to derive

$$f'(x) = \frac{f_{j+1}-f_j}{x_{j+1}-x_j} - \frac{3A^2-1}{6}(x_{j+1}-x_j)f''_j + \frac{3B^2-1}{6}(x_{j+1}-x_j)f''_{j+1} \qquad (1.4.4\#\text{eq.2})$$

$$f''(x) = Af''_j + Bf''_{j+1} \qquad (1.4.4\#\text{eq.3})$$

Usually f''_i, $i = 1, N$ is calculated rather than measured: requiring that $f'(x)$ is continuous from one interval to another (1.4.4#eq.2) leads to a tridiagonal system for $j = 2, N-1$

$$\frac{x_j-x_{j-1}}{6}f''_{j-1} + \frac{x_{j+1}-x_{j-1}}{3}f''_j + \frac{x_{j+1}-x_j}{6}f''_{j+1} = \frac{f_{j+1}-f_j}{x_{j+1}-x_j} - \frac{f_j-f_{j-1}}{x_j-x_{j-1}} \qquad (1.4.4\#\text{eq.4})$$

leaving only two conditions to be determined at the boundaries for f'_1 and f'_N (1.4.4#eq.2). Figure 1 illustrates the procedure and shows the excellent quality of a cubic approximation

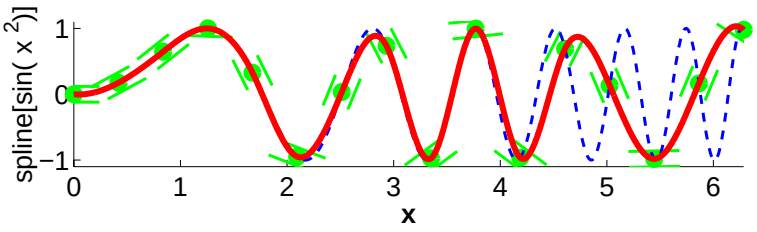


Figure 1: Approximation of $\sin(x^2)$ with cubic splines.

until it breaks down at the limit of 2 mesh points per wavelength.

1.4.5 Harmonic functions

A harmonic decomposition is obtained from discrete Fourier transform (DFT) assuming a *regular mesh* and a *periodicity* L. Using the notations $x_m = m\Delta x = mL/M$ and $k_m = 2\pi m/L$, the forward and backward transformations are given by:

$$\widehat{f}(k_m) = \frac{1}{M}\sum_{j=0}^{M-1} f(x_j)W^{-k_m x_j} \quad , \qquad W = exp(2\pi i/M) \quad (1.4.5\#\text{eq.1})$$

$$f(x_j) = \sum_{m=0}^{M-1} \widehat{f}(k_m)W^{+k_m x_j} \quad (1.4.5\#\text{eq.2})$$

If M is a power of 2, the number of operation can be dramatically reduced from $8M^2$ to $M\log_2 M$ with the *fast Fourier transform (FFT)*, applying recursively the decomposition

$$\begin{aligned}\widehat{f}(k_m) &= \frac{1}{M}\sum_{j=0}^{2^M-1} f(x_j)W^{-k_m x_j} = \sum_{j\text{ even}} + \sum_{j\text{ odd}} = \\ &= \frac{1}{M}\sum_{j=0}^{2^{M-1}-1} f(x_{2j})(W^2)^{-k_m x_j} + \frac{W^{-k_m}}{M}\sum_{j=0}^{2^{M-1}-1} f(x_{2j+1})(W^2)^{-k_m x_j}\end{aligned} \quad (1.4.5\#\text{eq.3})$$

with $W^2 = \exp(2i\pi/2^{M-1})$ until a sum of DFT of length $M = 2$ is obtained. Figure 1 illustrates how the approximation of a periodic square wave converges with an increasing resolution. Note how the function overshoots close to sharp edges: this is the *Gibbs phenomenon* and it will always be there for a finite number of terms in the sum.

It should be no surprise to anyone to hear that harmonic decompositions are well suited for smooth global functions having long wavelengths $\lambda \sim L$ and result in a rather narrow spectrum $|k| \leq 2\pi/\lambda \ll \pi/\Delta x$. Finally note that the convergence is not polynomial and that the implementation of non-periodic boundary conditions can be problematic.

1.4.6 Wavelets

Starting with a coarse (global) approximation, the first idea behind wavelets is to *successively refine* the representation and store the difference from one scale to the next

$$V_J = V_{J-1} \oplus W_{J-1} = \ldots = V_0 \oplus W_0 \oplus \ldots W_{J-1} \quad (1.4.6\#\text{eq.1})$$

This is illustrated with *Haar wavelets* in 1, showing that the piecewise constant approximation at the level V4 can be brought to the higher level V5 by adding a correction W4. Appropriate for integral equations and best suited for the understanding, *Haar* wavelets are however not practical for the evaluation of derivatives in PDEs. In the spirit of the

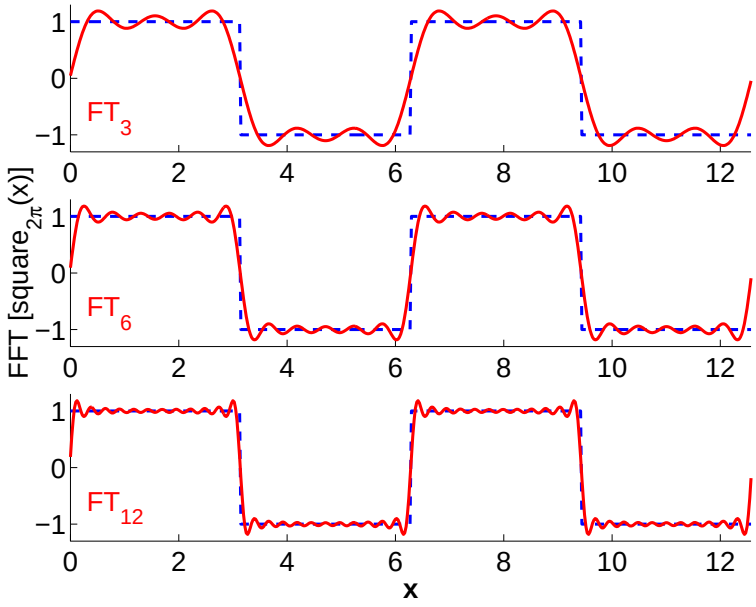


Figure 1: Square wave $\sum_{m=0}^{M} \frac{2}{\pi(2m+1)} \sin\left(\frac{(2m+1)\pi x}{L}\right)$ with $M = 3, 6, 12$.

FFT, Daubechies proposed a fast $\mathcal{O}(N)$ linear discrete wavelet transformation (DWT)

$$\begin{bmatrix} c_0 & c_1 & c_2 & c_3 & & & & & & & & \\ c_3 & -c_2 & c_1 & -c_0 & & & & & & & & \\ & & c_0 & c_1 & c_2 & c_3 & & & & & & \\ & & c_3 & -c_2 & c_1 & -c_0 & & & & & & \\ \vdots & \vdots & & & & & \ddots & & & & & \\ & & & & & & & c_0 & c_1 & c_2 & c_3 & \\ & & & & & & & c_3 & -c_2 & c_1 & -c_0 & \\ c_2 & c_3 & & & & & & & & c_0 & c_1 & \\ c_1 & -c_0 & & & & & & & & c_3 & -c_2 & \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_5 \\ f_6 \\ f_7 \\ f_8 \end{bmatrix} = \begin{bmatrix} \tilde{f}_1 \\ \tilde{g}_1 \\ \tilde{f}_2 \\ \tilde{g}_2 \\ \vdots \\ \tilde{f}_3 \\ \tilde{g}_3 \\ \tilde{f}_4 \\ \tilde{g}_4 \end{bmatrix} \rightarrow \begin{bmatrix} \tilde{f}_1 \\ \tilde{f}_2 \\ \tilde{f}_3 \\ \tilde{f}_4 \\ \vdots \\ \tilde{g}_1 \\ \tilde{g}_2 \\ \tilde{g}_3 \\ \tilde{g}_4 \end{bmatrix}$$

$$c_0 = (1+\sqrt{3})/4\sqrt{4} \qquad c_1 = (3+\sqrt{3})/4\sqrt{4} \qquad c_2 = (3-\sqrt{3})/4\sqrt{4} \qquad c_3 = (1-\sqrt{3})/4\sqrt{4}$$

(1.4.6#eq.2)

called DAUB4, which is applied successively with the permutation to the function $\tilde{f}$ until only the first two components remain. The inverse simply is obtained by reversing the

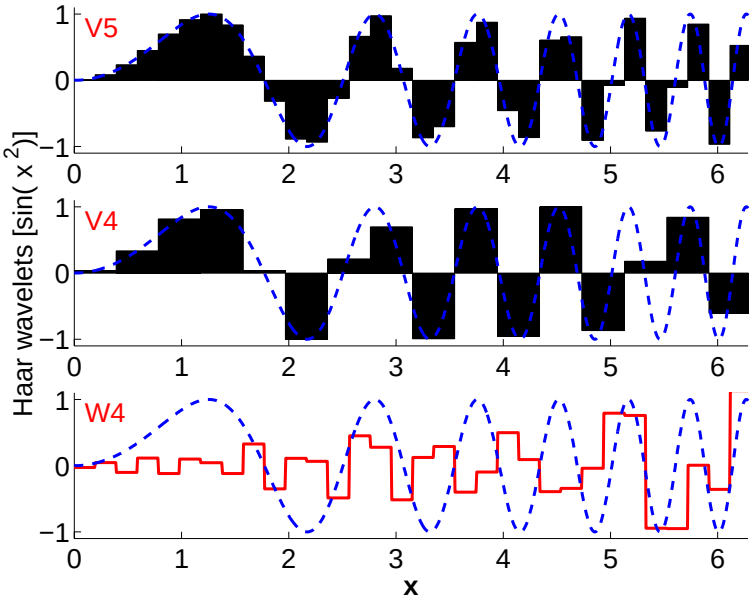


Figure 1: Successive approximation of $\sin(x^2)$ using *Haar* wavelets.

procedure and using the inverse matrix:

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_5 \\ f_6 \\ f_7 \\ f_8 \end{bmatrix} = \begin{bmatrix} c_0 & c_3 & & & \cdots & & & & & c_2 & c_1 \\ c_1 & -c_2 & & & \cdots & & & & & c_3 & -c_0 \\ c_2 & c_1 & c_0 & c_3 & & & & & & & \\ c_3 & -c_0 & c_1 & -c_2 & & & & & & & \\ & & & & \ddots & & & & & & \\ & & & & & c_2 & c_1 & c_0 & c_3 & & \\ & & & & & c_3 & -c_0 & c_1 & -c_2 & & \\ & & & & & & & c_2 & c_1 & c_0 & c_3 \\ & & & & & & & c_3 & -c_0 & c_1 & -c_2 \end{bmatrix} \begin{bmatrix} \tilde{f}_1 \\ \tilde{g}_1 \\ \tilde{f}_2 \\ \tilde{g}_2 \\ \vdots \\ \tilde{f}_3 \\ \tilde{g}_3 \\ \tilde{f}_4 \\ \tilde{g}_4 \end{bmatrix} \leftarrow \begin{bmatrix} \tilde{f}_1 \\ \tilde{f}_2 \\ \tilde{f}_3 \\ \tilde{f}_4 \\ \vdots \\ \tilde{g}_1 \\ \tilde{g}_2 \\ \tilde{g}_3 \\ \tilde{g}_4 \end{bmatrix}$$

(1.4.6#eq.3)

Approximations with DAUB4 have the peculiar property that the derivative exists only *almost everywhere*. Even if wavelets are not smooth, they still can represent exactly piecewise polynomial functions of arbitrary slope — the cusps in the wavelets cancel out exactly! Figure 2 illustrates how the solution converges when the first 2^4 DWT components of DAUB4[$\sin(x^2)$] are taken to a higher level of refinement with 2^5 components.

Note that next to *Haar* and *Daubechies* wavelets, a whole familly can be generated with a cascading process that has very nicely been illustrated in the Wavelet Cascade Applet[8]. Wavelets are still relatively new and remain a matter of active research [3]: they have so far been proven useful in approximation theory and are *currently being developed* both for PDE and integral equations. Apparent in the figures 1 and 2 is the *problem with boundary conditions*; another issue which will become clear with the coming sections is the difficulty of calculating inexpensive inner products.

[8] http://cm.bell-labs.com/cm/ms/who/wim/cascade/

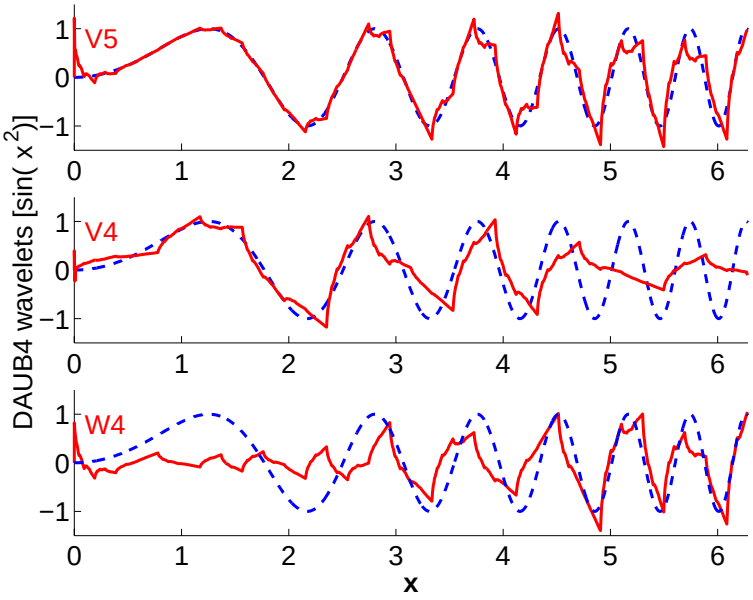


Figure 2: Successive approximation of $\sin(x^2)$ using DAUB4 wavelets.

1.4.7 Quasi-particles

Yet another way of approximating a function is to use quasi-particles

$$f(x) = \sum_{i=1}^{N} w_i S_i(x - x_i) \qquad (1.4.7\#\text{eq.1})$$

where w_i is the weight, S_i is the shape, and x_i the position of the particle. In JBONE the particle shapes are Dirac pulses $S_i(x) = \delta(x)$, so that the particles are localized in an infinitely small interval around x_i. The weight is set to unity

$$f(x) = \sum_{i=1}^{N} \delta(x - x_i) \qquad (1.4.7\#\text{eq.2})$$

This form of disctretization never converge locally, since the Dirac pulses are either zero or infinite. However, the "global properties", or moments, of a smooth and bounded function $g(x)$ discretized using Dirac pulses converge

$$\int_a^b dx \;\; x^K g(x) = \int_a^b dx \;\; x^K \sum_{i=1}^{\infty} w_i \delta(x - x_i) = \sum_{i=1}^{\infty} w_i x_i^K$$

Since there is no local convergence it is difficult to compare a quasi-particle discretization with for example a finite elements discretization. The solution is therefore often projected

onto a set of basis functions $\{\varphi_j\}_{j=1}^{N_\varphi}$

$$
\begin{aligned}
f_\varphi(x) &= \sum_{j=1}^{N_\varphi} \frac{\int f(x)\varphi_j(x))\,d\xi}{\sqrt{\int \varphi_j(\xi)^2\,d\xi}}\varphi_j(x) \\
&= \sum_{j=1}^{N_\varphi}\sum_{i=1}^{N} \frac{\int w_i\delta(\xi - x_i)\varphi_j(\xi)\,d\xi}{\sqrt{\int \varphi_j(\xi)^2\,d\xi}}\varphi_j(x) \qquad \text{(1.4.7\#eq.3)} \\
&= \sum_{j=1}^{N_\varphi}\sum_{i=1}^{N} \frac{w_i\varphi_j(x_i)}{\sqrt{\int \varphi_j(\xi)^2\,d\xi}}\varphi_j(x)
\end{aligned}
$$

In JBONE the basis functions are chosen to be piecewise linear roof-top function.

The functions $\{\varphi_j\}_{j=1}^{N_\varphi}$ used to project and plot must of course be the same. Figure 1 illustrates how a Gaussian function appears when using piecewise constant (box) and linear (roof-top) basis functions (1.4.3#eq.2). The dashed line mixes boxes for projection with roof-tops for the plotting and is wrong!

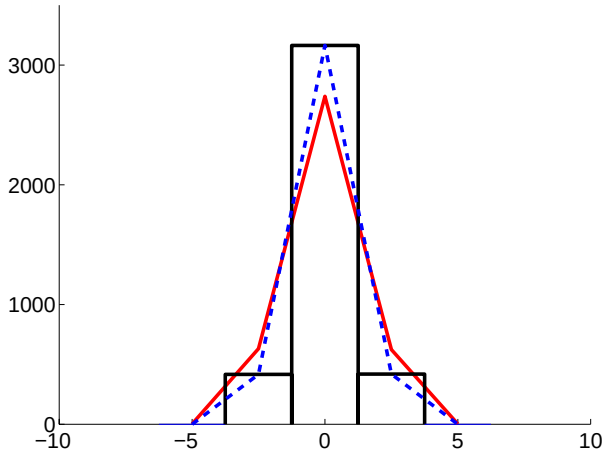


Figure 1: The solid lines are projections of Gaussian function on piecewise constant (box) and linear (roof-top) basis functions. The dashed line suggests that the result can be misleading when boxes are used for the projection and roof-tops for the plotting.

In the document on-line, try to press `INITIALIZE` a few times to get a feeling how good a quasi-particle approximation is to approximate a box with 64 grid points and a varying number of particles. The applet uses random numbers to generate a particle distribution from the initial condition $f_0(x)$ with a range $[f_{\min}, f_{\max}]$ in the interval $[a, b]$ following the procedure:

Let $i = 1$
while $i \leqslant N$

- **Let** x be a uniformly distributed random number in the interval $[a, b]$.
- **Let** y be a uniformly distributed random number in the interval $[f_{\min}, f_{\max}]$.
- **If** $y < f_i(x)$ **then let** $x_i = x$ and **advance** i by 1 **else do** nothing.

end while

1.5 Exercises

1.0 E-publishing

Familiarize yourself with the electronic submission of assignments and use the discussion forums. Follow the TEXlink to type a small text with formulas; learn how to interpret the error messages. Change the default applet parameters to compute the diffusion of a harmonic function and determine the largest diffusion coefficient that seems to give a reasonable result (this will be discussed in the next chapter). Read the rules governing the discussion forums and introduce yourself, telling a few words about your background and interests; chat with your colleagues...

1.1 Stiff ODE

Assuming boundary conditions $u(0) = 1; v(0) = 0$, use the MATLAB commands `ode23` and `ode23s` to integrate

$$\begin{aligned} u' &= 998u + 1998v \\ v' &= -999u - 1999v \end{aligned}$$

in the interval $[0; 1]$. Use the variable transformation $u = 2y - z; v = -y + z$ to compare with an analytical solution and show that the problem is stiff.

1.2 Predator-Prey model

Study the solutions of the famous Volterra Predator-Prey model

$$\frac{dy_1}{dt} = \alpha y_1 - \beta y_1 y_2; \qquad \frac{dy_2}{dt} = -\gamma y_2 + \delta y_1 y_2$$

that predicts the evolution of two populations (y_1 the number of preys and y_2 the number of predators) depending on each other for their existence. Assume $\alpha = \beta = \gamma = \delta = 1$ and use MATLAB to study periodic solutions in the interval $[0; 4] \times [0; 4]$. Is there a way to reach a natural equilibrium?

1.3 Fourier-Laplace transform

Solve the advection-diffusion analytically in an infinite 1D slab assuming both the advection speed u and the diffusion coefficient D constant. *Hint: use a Fourier-Laplace transform to first determine the evolution of the Green's function $G(x - x_0, t)$ starting from an initial condition $\delta(x - x_0)$. Superpose to describe an arbitrary initial function $f_0(x)$.*

1.4 Random-walk

Determine the diffusion constant for a random-walk process with steps of a typical duration τ and mean free path $\lambda = \sqrt{\langle v^2 \rangle}\tau$. *Hint: calculate first the RMS displacement $\langle z^2 \rangle (t)$ of the position after a large number $M = t/\tau$ of statistically independent steps took place. Take the second moment of the diffusion equation and integrate by parts to calculate the average $\overline{z^2}(t)$. Conclude by relating each other using the ergodicity theorem.*

1.5 Convergence

Calculate a discrete representation of $\sin(kx)$, $kx \in [0; 2\pi]$ for atleast four numerical approximations introduced in the first chapter. Sketch (with words) the relative local error for $kx = 1$ and show how the approximations converge to the analytical value when the numerical resolution increases. What about the first derivative?

1.6 Laplacian in 2D

Use a Taylor expansion to calculate an approximation of the Laplacian operator on a rectangular grid $\Delta x = \Delta y = h$. Repeat the calculation for an evenly-spaced, equilateral triangular mesh. *Hint: determine the coordinate transformation for a rotation of* $0, 60, 120^o$ *and apply the chain rule for partial derivatives in all the three direction.*

1.7 Hypercube

Show that when using a regular grid with n mesh points in each direction, most of the values sampled in hypercube of large dimension N land on the surface. Can you think of a discretization that is more appropriate? *Hint: estimate the relative number of mesh points within the volume with* $\left(\frac{n-2}{n}\right)^N$*, write it as an exponential and expand.*

1.6 Further reading

- **ODE, symplectic integration and stiff equations.**
  Numerical Recipes [4] §16, 16.6, Sanz-Serna [5], Dahlquist [6] §13
- **PDE properties.**
  Fletcher [7] §2
- **Interpolation and differentiation.**
  Abramowitz [2] §25.2–25.3, Dahlquist [6] §4.6, 4.7, Fletcher [7] §3.2, 3.3
- **FEM approximation.**
  Fletcher [7] §5.3, Johnson [8], Appert [9]
- **Splines.**
  Numerical Recipes [4] §12.0–12.2, Dahlquist [6] §4.8
- **FFT.**
  Numerical Recipes [4] §12.0–12.2
- **Wavelets.**
  Numerical Recipes [4] §13.10, `www.wavelet.org`[9] [3]
- **Software.**
  Guide to Available Mathematical Software[10] [10], Computer Physics Communications Library[11] [11]

[9] http://www.wavelet.org/wavelet/index.html
[10] http://gams.nist.gov
[11] http://www.cpc.cs.qub.ac.uk/cpc/

1.7 Solutions

1.3. Fourier-Laplace transform. To solve the advection-diffusion equation for $f(x,t)$ as an initial value problem, start with a Laplace transform in time

$$\int_0^\infty dt \exp(i\omega t) \left[\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} - D\frac{\partial^2 f}{\partial x^2}\right] = 0 \qquad \Im m(\omega) > 0$$

The condition $\Im m(\omega) > 0$ is here mandatory to ensure causality. Integrate the first term by parts and substitute a Dirac function $f_0(x) = \delta(x - x_0)$ for the initial condition

$$f \exp(i\omega t)|_0^\infty + \int_0^\infty dt \exp(i\omega t) \left[-i\omega f + u\frac{\partial f}{\partial x} - D\frac{\partial^2 f}{\partial x^2}\right] = 0$$
$$-\delta(x - x_0) + \left[-i\omega f(x,\omega) + u\frac{\partial f(x,\omega)}{\partial x} - D\frac{\partial^2 f(x,\omega)}{\partial x^2}\right] = 0$$

using here the notation $f(x,\omega)$ for the Laplace transform in time of $f(x,t)$. Spatial derivatives can be dealt with a simple Fourier transform

$$\int_{-\infty}^\infty dx \exp(-ikx) \left[-\delta(x - x_0) - i\omega f(x,\omega) - iukf(x,\omega) + Dk^2 f(x,\omega)\right] = 0$$
$$\exp(-ikx_0) + i\omega f(k,\omega) + iukf(k,\omega) - Dk^2 f(k,\omega) = 0$$

which yields an explicit solution for the Fourier-Laplace transformed function

$$f(k,\omega) = \frac{i \exp(-ikx_0)}{\omega + uk + iDk^2}$$

This has a pole in the complex plane for $\omega = -uk - iDk^2$ and needs to be taken into account when inverting the Laplace transform

$$\begin{aligned} f(k,t) &= \int_{-\infty+iC}^{+\infty+iC} \frac{d\omega}{2\pi} \exp(-i\omega t) \left(\frac{-i \exp(-ikx_0)}{\omega + uk + iDk^2}\right) \qquad C > 0 \\ &= 2\pi i \left(\frac{-i}{2\pi} \exp(-i[-uk - iDk^2]t) \exp(-ikx_0)\right) \\ &= \exp(i[ukt - x_0]) \exp(-Dk^2 t) \end{aligned}$$

where the residue theorem has been used to calculate the integral along the positive real frequencies and closing the contour in the positive half plane where the Laplace integral decays exponentially. Inverting the Fourier integral

$$\begin{aligned} f(x,t) &= \int_{-\infty}^\infty \frac{dk}{2\pi} \exp(ikx) \left[\exp i(ukt - x_0) \exp(-Dk^2 t)\right] \\ &= \frac{1}{2\pi} \int_{-\infty}^\infty dk \exp ik(x - x_0 - ut) \exp(-Dk^2 t) \end{aligned}$$

Using the formula (3.323.2) from Gradshteyn & I. M.Ryzhik [12]

$$\int_{-\infty}^\infty dx \exp(-p^2 x^2) \exp(\pm qx) = \frac{\sqrt{\pi}}{p} \exp\left(\frac{q^2}{4p^2}\right) \qquad p > 0$$

with $p = Dt$ and $q = i(x - x_0 - ut)$, this finally yields the explicit solution

$$f(x,t) = \frac{1}{2\sqrt{\pi Dt}} \exp\left(-\frac{(x - x_0 - ut)^2}{4Dt}\right)$$

from (eq.1.3.2#eq.3) for the Green function and shows explicitly the characteristic $x - x_0 - ut = 0$.

1.4. Random-walk. Consider a walk with a large number M of statistically independent steps ξ_i randomly distributed, so that the statistical average of the random variable yields $< \xi >= 0$. The final position $z = \sum_{i=1}^{M} \xi_i$ in average coincides with the initial position $< z >= \sum_{i=1}^{M} < \xi_i >= 0$. The root mean square (RMS) displacement, however, is finite

$$< z^2 >= \left\langle \left(\sum_{i=1}^{M} \xi_i\right) \left(\sum_{j=1}^{M} \xi_j\right) \right\rangle = \sum_{i=1}^{M} < \xi_i^2 > + \sum_{i \neq j} < \xi_i \xi_j >= M < \xi^2 >= \frac{t}{\tau} \lambda_{\text{mfp}}^2$$

where τ defines the average time elapsed between consecutive steps, $M = t/\tau$ is the number of steps taken during a time interval of duration t and λ_{mfp} is the so-called mean free path.

Now, repeat the calculation with the second moment of the diffusion equation (1.3.2#eq.1), defining the total density N as

$$N(t) = \int_{-\infty}^{+\infty} n(z,t) dz$$
$$\bar{z^2}(t) = \frac{1}{N} \int_{-\infty}^{+\infty} z^2 n(z,t) dz$$

The first term in (1.3.2#eq.1) yields

$$\int_{-\infty}^{+\infty} z^2 \frac{\partial n}{\partial t} dz = \frac{\partial}{\partial t} \int_{-\infty}^{+\infty} z^2 n dz = N \frac{\partial}{\partial t} \bar{z^2}$$

and the second after two integration by parts gives

$$\begin{aligned} \int_{-\infty}^{+\infty} z^2 D \frac{\partial^2 n}{\partial z^2} dz &= \left[z^2 D \frac{\partial n}{\partial z} \right]_{-\infty}^{+\infty} - \int_{-\infty}^{+\infty} \frac{\partial}{\partial z}(z^2 D) \frac{\partial n}{\partial z} dz = \\ &= \left[\frac{\partial}{\partial z}(z^2 D) n \right]_{-\infty}^{+\infty} + \int_{-\infty}^{+\infty} \frac{\partial^2}{\partial z^2}(z^2 D) n dz = \\ &= 2D \int_{-\infty}^{+\infty} n(z,t) dz = 2DN \end{aligned}$$

where a constant diffusion has been assumed for simplicity $D \neq D(x)$. Reassembling both terms and integrating in time leads to

$$N \frac{d\bar{z^2}}{dt} = 2DN \quad \Rightarrow \quad \bar{z^2} = 2Dt$$

The ergodicity theorem is finally used to identify the statistical average $< X >$ with the mean value of a continuous variable $\bar{X}$, leading to the well known identity

$$D = \frac{\lambda_{\mathrm{mfp}}^2}{2\tau}.$$

The theory of stochastic processes behind the Monte-Carlo Method is rather sophisticated and will be introduced later in sect.5. From the calculation above, it is however possible to conclude now already that the evolution of a large number of independent particles following a random walk can be described implemented in JBONE with the simple algorithm

```
for(int j = 0; j < numberOfParticles; j++){
  particlePosition[j] += random.nextGaussian() *
    Math.sqrt(2 * diffusCo * timeStep);
}
```

and describes in fact a diffusion.

1.8 Interactive evaluation form

Your anonymous opinion is precious. Please fill-in our anonymous form in the web edition. Thank you very much in advance for your collaboration!

2 FINITE DIFFERENCES

2.1 Explicit 2 levels

A spatial difference to the left for the advection (1.3.1#eq.1) and a centered difference for the diffusion (1.3.2#eq.1) yields an explicit scheme that involves only two time levels

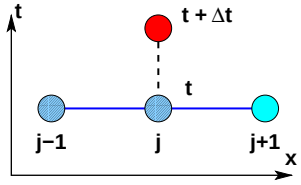


Figure 1: Explicit 2 levels.

$$\begin{aligned} \frac{f_j^{t+\Delta t} - f_j^t}{\Delta t} &+ u\frac{f_j^t - f_{j-1}^t}{\Delta x} - D\frac{f_{j+1}^t - 2f_j^t + f_{j-1}^t}{\Delta x^2} = 0 \\ f_j^{t+\Delta t} &= f_j^t - \beta\left[f_j^t - f_{j-1}^t\right] + \alpha\left[f_{j+1}^t - 2f_j^t + f_{j-1}^t\right] \end{aligned} \qquad (2.1\#\text{eq.1})$$

where the so-called *Courant-Friedrich-Lewy (CFL) number* $\beta = u\Delta t/\Delta x$ and the coefficient $\alpha = D\Delta t/\Delta x^2$ measure typical advection and diffusion velocities relatif to the characteristic speed of the mesh $\Delta x/\Delta t$. For every step in time, a new value is obtained explicitly by the linear combination of the current neighbors. This scheme has been implemented in JBONE as

```
for (int i=1; i<n; i++) {
  fp[i]=f[i] -beta *(f[i]-f[i-1])+alpha*(f[i+1]-2.*f[i]+f[i-1]); }
fp[0]=f[0]   -beta *(f[0]-f[ n ])+alpha*(f[ 1 ]-2.*f[0]+f[ n ]);
fp[n]=f[n]   -beta *(f[n]-f[n-1])+alpha*(f[ 0 ]-2.*f[n]+f[n-1]);
```

where the last two statements take care of the *periodicity*.

Sharp variations of the solution should generally be *AVOIDED* in a physically meaningful calculation. To study the properties of a numerical scheme, it is however often illuminating to initialize a box function and check how the intrinsic numerical dispersion and damping affect the superposition of short and long wavelengths. The document online shows the evolution obtained for a constant advection $u = 1$ with no physical diffusion $D = \alpha = 0$: after 128 steps of duration Δt= 0.5, the pulse propagates exactely once accross the domain with a period $L = 64$ and discretized with 64 mesh points so that $\Delta x = 1$. The lowest order moment (density) is conserved to a very good accuracy and the function remains positive everywhere as it should. The shape, however, is strongly affected by the intrinsic *numerical diffusion* of (2.1#eq.1)!
Numerical experiments:

- Change the initial condition from `Box` to `Cosine`, and vary the wavelength λ = 2–64 mesh points per wavelength to verify that it is indeed the short wavelengths associated with steep wavefronts that get damped: exactly what you expect from diffusion (1.3.2#eq.4) except that with $D = 0$, this is here a numerical artifact! Without special care, this can easily cover the physical process you want to model...

- Looking for a quick fix, you reduce the time step and the CFL number from $\beta = 0.5$ down 0.1. What happens ? Adding further to the confusion, increase the time step to exactly 1 and check what happens.

These properties can be understood from the *numerical dispersion* analysis using a local ansatz $f \sim F = \exp(i[kx - \omega t])$ with a homogeneous grid $x_j = j\Delta x$. Define the amplification factor as

$$G = \exp(-i\omega\Delta t) \qquad (2.1\#\text{eq.2})$$

and simplify by F to cast the dispersion relation into

$$\begin{aligned} G &= 1 - \beta\,[1 - \exp(-ik\Delta x)] + \alpha\,[\exp(+ik\Delta x) - 2 + \exp(-ik\Delta x)] \\ &= 1 - \beta\,[1 - \exp(-ik\Delta x)] - 4\alpha \sin^2\left(\frac{k\Delta x}{2}\right) \end{aligned} \qquad (2.1\#\text{eq.3})$$

Figure 2 illustrates with vectors in the complex plane how the first three term are combined in the presence of advection only ($D = 0$). Short wavelengths $k\Delta x \simeq \pi$ get damped $|G| < 1$

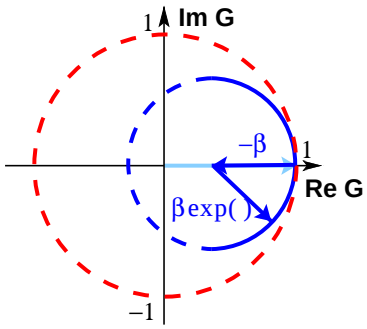


Figure 2: Numerical dispersion / stability (explicit 2 levels).

as long as the CFL number β remains smaller than unity; if this is exceeded, $|G| > 1$ and the shortest wavelengths grow into a so-called *numerical instability*. Try and model this in the JBONE applet. For a diffusive process ($\beta = 0$), the dispersion relation (2.1#eq.3) shows that the scheme is stable for all wavelengths provided that $\|G\| = \|1 - 4\alpha \sin^2(\pi/2)\| < 1$ i.e. $\alpha < 1/2$. Although the superposition of advection and diffusion is slightly more limiting, the overall conditions for numerical stability may here be conveniently summarized as

$$\begin{aligned} \alpha &= \frac{D\Delta t}{\Delta x^2} < \frac{1}{2} \\ \beta &= \frac{u\Delta t}{\Delta x} < 1 \quad \text{(CFL condition)} \end{aligned} \qquad (2.1\#\text{eq.4})$$

It is worth pointing out that a finite difference evaluated backwards for the advection is unstable for negative velocities $u < 0$. In an inhomogeneous medium, this defect of a de-centered schemes can be cured with a so-called *upwind difference*, by taking the finite difference forward or backward according to the local direction of propagation (exercise 2.1).

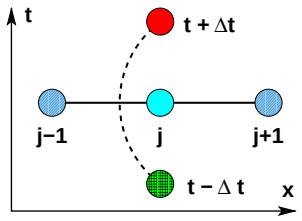


Figure 3: Explicit 3 levels.

2.2 Explicit 3 levels

A more accurate scheme in $\mathcal{O}(\Delta x^2 \Delta t^2)$ can be obtained with differences centered both in time and space

$$\frac{f_j^{t+\Delta t} - f_j^{t-\Delta t}}{2\Delta t} + u\frac{f_{j+1}^t - f_{j-1}^t}{2\Delta x} - D\frac{f_{j+1}^t - 2f_j^t + f_{j-1}^t}{\Delta x^2} = 0$$

$$f_j^{t+\Delta t} = f_j^{t-\Delta t} - \beta\left[f_{j+1}^t - f_{j-1}^t\right] + 2\alpha\left[f_{j+1}^t - 2f_j^t + f_{j-1}^t\right] \qquad (2.2\#\text{eq.5})$$

and has been implemented in JBONE as

```
for (int i=1; i<n; i++) {
  fp[i]=fm[i] -beta*(f[i+1]-f[i-1]) +2*alpha*(f[i+1]-2.*f[i]+f[i-1]);}
fp[0]=fm[0]   -beta*(f[ 1 ]-f[ n ]) +2*alpha*(f[ 1 ]-2.*f[0]+f[ n ]);
fp[n]=fm[n]   -beta*(f[ 0 ]-f[n-1]) +2*alpha*(f[ 0 ]-2.*f[n]+f[n-1]);
```

Note that a special starting procedure is required to calculate an approximation for a time $-\Delta t$, which is anterior to the initial condition $t = 0$. This can be defined by taking one explicit step backwards in time with the 2 levels scheme (2.1#eq.1). The document on-line shows the evolution with the same advection conditions as used previously: despite the (mathematically) higher accuracy in $\mathcal{O}(\Delta x^2 \Delta t^2)$ and an excellent conservation of the moment, the initial box function is here strongly distorted by the *phase errors* from *numerical dispersion* and the solution becomes locally negative — clearly a non-sense when $f(x,t)$ is a density. You should however not be mislead here by the choice of the initial condition.
Numerical experiments:

- check how well this scheme performs for the advection of harmonic functions.
- Even though we used the same spatial differencing for the diffusion term as in (2.1#eq.1), verify that a 3 levels scheme is always unstable for $D \neq 0$ (exercise 2.2)

2.3 Lax-Wendroff

Rather than counting on your intuition for the right combination of terms, is it possible to formulate a systematic recipe for solving an equation in *Eulerian coordinates* with a chosen accuracy? Yes, using the so-called *Lax-Wendroff approach*, which can easily be generalized for non-linear and vector equations.

1. Discretize the function on a regular grid $f(x) \rightarrow (x_j, f_j)$, $j = 1, N$,

2. Expand the differential operators in time using a Taylor series

$$f_{x_j}^{t+\Delta t} = f_{x_j}^t + \Delta t \frac{\partial f}{\partial t}\bigg|_{x_j} + \frac{\Delta t^2}{2}\frac{\partial^2 f}{\partial t^2}\bigg|_{x_j} + \mathcal{O}(\Delta t^3) \qquad (2.3\#\text{eq.}6)$$

3. Substitute the time derivatives from the master equation. Using only advection (1.3.1#eq.1) here for illustration, this yields

$$f_{x_j}^{t+\Delta t} = f_{x_j}^t - u\Delta t \frac{\partial f}{\partial x}\bigg|_{x_j} + \frac{(u\Delta t)^2}{2}\frac{\partial^2 f}{\partial x^2}\bigg|_{x_j} + \mathcal{O}(\Delta t^3) \qquad (2.3\#\text{eq.}7)$$

4. Take centered differences for spatial operators

$$f_j^{t+\Delta t} = f_j^t - \frac{\beta}{2}\left(f_{j+1}^t - f_{j-1}^t\right) + \frac{\beta^2}{2}\left(f_{j+1}^t - 2f_j^t + f_{j-1}^t\right) + \mathcal{O}(\Delta x^3 \Delta t^3) \quad (2.3\#\text{eq.}8)$$

This procedure results in a second order advectionscheme which is explicit, centered and stable provided that the CFL number $\beta = u\Delta t/\Delta x$ remains below unity:

```
for (int i=1; i<n; i++) {
  fp[i]=f[i] -0.5*beta *(f[i+1]-f[i-1])
             +0.5*beta*beta*(f[i+1]-2.*f[i]+f[i-1]); }
fp[0]=f[0] -0.5*beta *(f[ 1 ]-f[ n ])
           +0.5*beta*beta*(f[ 1 ]-2.*f[0]+f[ n ]);
fp[n]=f[n] -0.5*beta *(f[ 0 ]-f[n-1])
           +0.5*beta*beta*(f[ 0 ]-2.*f[n]+f[n-1]);
```

Numerical diffusion damps mainly the shorter wavelengths and distorts somewhat the box function as it propagates.
Numerical experiments:

- Test if this scheme works also for negative velocities.
- Explain what happens when you chose $\beta = 1$; why can you not rely on that?

2.4 Leapfrog, staggered grid

The leap-frog algorithm provides both for a remedy against the numerical damping and benefits from a higher accuracy $\mathcal{O}(\Delta x^2 \Delta t^2)$ allowing you to take larger time steps: it is perhaps the best finite-difference scheme for the evolution of hyperbolic wave equations. The idea, illustrated

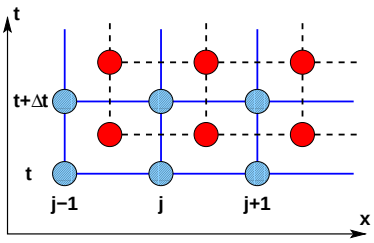


Figure 4: Staggered grids.

in figure 4, is to use so called *staggered grids* (where the mesh points are shifted with respect to each other by half an interval) and evaluate the solution using two functions of the form

$$\begin{cases} \frac{1}{\Delta t}\left[f_{i+1/2}^{t+\Delta t} - f_{i+1/2}^{t}\right] = \frac{u}{\Delta x}\left[g_{i+1}^{t+\Delta t/2} - g_{i}^{t+\Delta t/2}\right] \\ \frac{1}{\Delta t}\left[g_{i}^{t+\Delta t/2} - g_{i}^{t-\Delta t/2}\right] = \frac{u}{\Delta x}\left[f_{i+1/2}^{t} - f_{i-1/2}^{t}\right] \end{cases} \qquad (2.4\#\text{eq.9})$$

This scheme suggests with scalar fields (f, g) how Maxwell's equations can be conveniently solved using the so-called *finite differences in the time domain (FDTD)* by using different grids for the electric and the magnetic fields $(\vec{E}, \vec{B})$ [2.7]. The leapfrog algorithm in JBONE has been implemented as

```
for (int i=1; i<=n; i++) {                  //time + time_step
  fp[i]=f[i] -beta*(g[i]-g[i-1]); }
fp[0]=f[0] -beta*(g[0]-g[n]);
for (int i=0; i<=n-1; i++) {                //time + 1.5*time_step
  gp[i]=g[i] -beta*(fp[i+1]-fp[i]); }
gp[n]=g[n] -beta*(fp[0]-fp[n]);
```

Special care is required when starting the integration — here particularly since it is the initial condition which determines the direction of propagation (exercise 2.4).
Numerical experiments:

- Vary the wavelength of the harmonic oscillation in the applet and check how it performs in terms of numerical diffusion / dispersion in comparison with the previous schemes.
- Compare your conclusions with the ones you draw when propagating a box function.

Finally, note by substitution that that a leap-frog scheme is in fact equivalent to an implicit 3 levels scheme

$$\frac{1}{(\Delta t)^2}\left(f_i^{t+\Delta t} - 2f_i^t + f_i^{t-\Delta t}\right) = \frac{u^2}{(\Delta x)^2}\left(f_{i+1}^t - 2f_i^t + f_{i-1}^t\right). \qquad (2.4\#\text{eq.10})$$

where the matrix inversion is carried out explicitly in an elegant manner.

2.5 Implicit Crank-Nicholson

All the schemes that have been developed so far calculate unknowns explicitly by linear combination from quantities that are all known. *Implicit methods* allow for the coupling of unknowns and therefore require a matrix inversion. This makes the implementation considerably more complicated; the finite element approach is then generally preferable, since it offers more flexibility for the same programming effort — as will be shown in sect.3). We nevertheless introduce here two popular schemes that are based on the same combination of variables originally proposed by Crank & Nicholson: one deals with diffusive problems and the other is often used to solve the Schrödinger equation.

2.5.1 Advection-diffusion equation

Combine centered differences in space that are evaluated with equal weights from the current and the future level of time discretization:

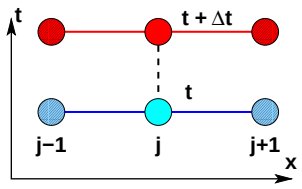


Figure 5: Implicit Crank-Nicholson.

$$\frac{f_j^{t+\Delta t}-f_j^t}{\Delta t} + \frac{u}{2}\left\{\frac{f_{j+1}^{t+\Delta t}-f_{j-1}^{t+\Delta t}}{2\Delta x}+\frac{f_{j+1}^{t}-f_{j-1}^{t}}{2\Delta x}\right\} - \frac{D}{2}\left\{\frac{f_{j+1}^{t+\Delta t}-2f_j^{t+\Delta t}+f_{j-1}^{t+\Delta t}}{\Delta x^2}+\frac{f_{j+1}^{t}-2f_j^{t}+f_{j-1}^{t}}{\Delta x^2}\right\}=0 \tag{2.5.1\#eq.11}$$

This Crank-Nicholson scheme is conveniently written as a linear system

$$\begin{pmatrix} -\alpha/2-\beta/4 \\ 1+\alpha \\ -\alpha/2+\beta/4 \end{pmatrix}^T \cdot \begin{pmatrix} f_{j-1}^{t+\Delta t} \\ f_j^{t+\Delta t} \\ f_{j+1}^{t+\Delta t} \end{pmatrix} = \begin{pmatrix} \alpha/2+\beta/4 \\ 1-\alpha \\ \alpha/2-\beta/4 \end{pmatrix}^T \cdot \begin{pmatrix} f_{j-1}^{t} \\ f_j^{t} \\ f_{j+1}^{t} \end{pmatrix} \tag{2.5.1\#eq.12}$$

showing explicitly the tri-diagonal structure of the matrix implemented in JBONE as

```
BandMatrix a = new BandMatrix(3, f.length);
BandMatrix b = new BandMatrix(3, f.length);
double[]   c = new double[f.length];
for (int i=0; i<=n; i++) {
  a.setL(i,-0.25*beta -0.5*alpha);            //Matrix elements
  a.setD(i,            1. +alpha);
  a.setR(i, 0.25*beta -0.5*alpha);
  b.setL(i, 0.25*beta +0.5*alpha);            //Right hand side
  b.setD(i,            1. -alpha);
  b.setR(i,-0.25*beta +0.5*alpha);
}
c=b.dot(f);                                   //Right hand side
c[0]=c[0]+b.getL(0)*f[n];                     // with periodicity
c[n]=c[n]+b.getR(n)*f[0];

fp=a.solve3(c);                               //Solve linear problem
```

We will show later in sect.3 how the simple BandMatrix.solve3() method solves the linear system efficiently with an LU-factorisation in $\mathcal{O}(N)$ operations [2.7] dealing explicitly with the periodicity.

Repeating the *von Neumann* stability analysis with (2.5.1#eq.11), a pure diffusive process ($u=\alpha=0$) yields the amplification factor

$$G=\frac{1-2\alpha\sin^2\left(\frac{k\Delta x}{2}\right)}{1+2\alpha\sin^2\left(\frac{k\Delta x}{2}\right)} \tag{2.5.1\#eq.13}$$

proving that this scheme is unconditionally stable $\forall\Delta x$, $\forall\Delta t$, phase errors affecting short wavelength oscillations $k\Delta x\sim 1$. The analogue is true also for the advective part. This

favorable stability property can nicely be exploited when solving diffusion dominated problems that concerned mainly with the evolution of large scale features $\lambda \gg \Delta x$.

Starting from a relatively smooth Gaussian pulse that is subject both to advection and diffusion $u = D = 1$, the document on-line shows that a reasonably accurate solution (12 % for the valley to peak ratio as the time reaches 100) can be computed using extremely large time steps with $\alpha = \beta = 5$.
Numerical experiments:

- Repeat the calculation with an initial box function; what happens?
- Add a few tiny time steps using a 2 level scheme (2.1#eq.1) at the end of an implicit evolution with very large time steps. When is this combination particularly indicated?

2.5.2 Schrödinger equation

When a complex wave function $|\psi >= \psi(x,t)$ (which represents a non-decaying particle in quantum mechanics) evolves in time, the physical problem requires that the total probability of finding that particle somewhere in space remains exactly unity at all times $< \psi|\psi >= \int |\psi|^2 dx = 1 \ \forall t$. Solving the Schrödinger equation

$$i\frac{\partial \psi}{\partial t} = H(x)\psi \qquad \text{with} \qquad H(x) = -\frac{\partial^2}{\partial x^2} + V(x) \qquad (2.5.2\#\text{eq.14})$$

here normalized so as to have Planck's constant $\hbar = 1$, the particle mass $m = 1/2$ and a static potential $V(x)$, it could therefore be important to keep the evolution operator unitary even in its discretized form. This is achieved by writing the formal solution $\psi(x,t) = \exp(-iHt)\psi(x,0)$ in the so-called *Cayley form* for the finite difference in time

$$\exp(-iH\Delta t) \ \simeq \ \frac{1 - \frac{i}{2}H\Delta t}{1 + \frac{i}{2}H\Delta t} \qquad (2.5.2\#\text{eq.15})$$

This approximation is unitary, accurate to second-order in time $\mathcal{O}(\Delta t^2)$ and suggests an evolution of the form

$$(1 + \frac{i}{2}H\Delta t)\psi^{t+\Delta t} \ = \ (1 - \frac{i}{2}H\Delta t)\psi^t \qquad (2.5.2\#\text{eq.16})$$

Replacing the second order derivative in the Hamiltonian operator $H(x)$ with finite differences centered in space $\mathcal{O}(\Delta x^2)$, one obtains a scheme that is stable, unitary and in fact again the Crank-Nicholson method in a new context:

$$\begin{aligned} \psi_j^{t+\Delta t} \ &+ \ \frac{i\Delta t}{2}\left(-\frac{\psi_{j-1}^{t+\Delta t} - 2\psi_j^{t+\Delta t} + \psi_{j+1}^{t+\Delta t}}{\Delta x^2} + V_j\psi_j^{t+\Delta t}\right) = \\ = \psi_j^t \ &+ \ \frac{i\Delta t}{2}\left(-\frac{\psi_{j-1}^{t} - 2\psi_j^{t} + \psi_{j+1}^{t}}{\Delta x^2} + V_j\psi_j^{t}\right) \end{aligned} \qquad (2.5.2\#\text{eq.17})$$

The scheme is finally cast into the linear system

$$\begin{pmatrix} -\frac{i\Delta t}{2\Delta x^2} \\ 1 + \frac{i\Delta t}{\Delta x^2} + \frac{i\Delta t}{2}V_j \\ -\frac{i\Delta t}{2\Delta x^2} \end{pmatrix}^T \cdot \begin{pmatrix} \psi_{j-1}^{t+\Delta t} \\ \psi_j^{t+\Delta t} \\ \psi_{j+1}^{t+\Delta t} \end{pmatrix} = \begin{pmatrix} \frac{i\Delta t}{2\Delta x^2} \\ 1 - \frac{i\Delta t}{\Delta x^2} - \frac{i\Delta t}{2}V_j \\ \frac{i\Delta t}{2\Delta x^2} \end{pmatrix}^T \cdot \begin{pmatrix} \psi_{j-1}^{t} \\ \psi_j^{t} \\ \psi_{j+1}^{t} \end{pmatrix} \qquad (2.5.2\#\text{eq.18})$$

exploiting the tri-diagonal structure of the matrix, and has been implemented in JBONE using complex arithmetic

```
BandMatrixC a = new BandMatrixC(3, h.length);  //Complex objects
BandMatrixC b = new BandMatrixC(3, h.length);
Complex[]   c = new Complex[h.length];
Complex     z = new Complex();

double[] V = physData.getPotential();          //Heavyside(x-L/2) times
double  scale  = 10.*velocity;                 //  an arb. scaling factor
double  dtodx2 = timeStep/(dx[0]*dx[0]);
Complex pih  = new Complex(0., 0.5*dtodx2);
Complex mih  = new Complex(0.,-0.5*dtodx2);
Complex pip1 = new Complex(1.,     dtodx2);
Complex mip1 = new Complex(1.,    -dtodx2);

for (int i=0; i<=n; i++) {
  z = new Complex(0.,0.5*scale*timeStep*V[i]);
  a.setL(i,mih);                               //Matrix elements
  a.setD(i,pip1.add(z));
  a.setR(i,mih);
  b.setL(i,pih);                               //Right hand side
  b.setD(i,mip1.sub(z));
  b.setR(i,pih);
}

c=b.dot(h);                                    //Right hand side with
c[0]=c[0].add(b.getL(0).mul(h[n]));            // with periodicity
c[n]=c[n].add(b.getR(n).mul(h[0]));

hp=a.solve3(c);                                //Solve linear problem

for (int i=0; i<=n; i++){                      //Plot norm & real part
  fp[i]=hp[i].norm();
  gp[i]=hp[i].re();
}
```

It again relys on the complex BandMatrixC.solve3() method to solve the linear system efficiently with a standard LU-factorisation with $\mathcal{O}(N)$ complex operations [2.7].

The document on-line shows the evolution of the wavefunction and the probability when a wavepacket is scattered on a (periodic) square potential barrier rizing in the right side of the simulation domain. The conservation of the moment shows that the total probability remains indeed perfectly conserved for all times.
Numerical experiment:

- Modify the energy in the wavepacket `ICWavelength` and verify that the applet recproduces the probability of reflection / transmission accross a potential barrier.

2.6 Exercises

2.1 Upwind differences, boundary conditions

Use upwind differences and modify the explicit 2 level scheme (2.1#eq.1) in JBONE to propose a new scheme that is stable both for forward and backward propagation. Implement Dirichlet conditions to maintain a constant value on the boundary up-the-wind (i.e. in the back of the pulse) and an outgoing wave in the front.

2.2 Numerical dispersion

Determine analytically how the explicit 3 levels scheme (2.2#eq.5) affects the advection of short wavelength components and calculate the growth rate of the numerical instability when $D \neq 0$. Use a harmonic initial condition to confirm your results with the JBONE applet.

2.3 Shock waves using Lax-Wendroff

Follow the Lax-Wendroff approach to solve the Burger equation (1.3.4#eq.2) with the JBONE applet. Implement a first order scheme and study the numerical convergence for different values of the physical diffusion D. If you want, you may try to go to second order and notice that it is better not to expand high order non-linear derivatives to keep $2f(\partial_x f)^2 + f^2\partial_x^2 f = (\partial_x^2 f^3)/3$. How can you compare the performance of the first and second order schemes? Hint: use the high order finite differences formulas (1.4.2#eq.2) to calculate an approximation for the derivatives.

2.4 Leapfrog resonator

Study the intitialization of the leapfrog scheme (2.4#eq.9) in the JBONE applet and examine how you can affect the direction of propagation of a Gaussian pulse (comment out one of the three definitions for `gm`). Modify the numerical scheme to incorporate perfectly reflecting boundary conditions.

2.5 European option

Use a finite difference schemes to solve the Black-Scholes equation

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r - D_0) S \frac{\partial V}{\partial S} - rV = 0 \qquad (2.6\#\text{eq.}1)$$

for the value of a *European Vanilla put option* using an underlying asset $S \in [0; 130]$ and a stike price $E = 95$, a fixed annual interest rate $r = 0.05$, a volatility $\sigma = 0.7155$, no dividend $D_0 = 0$ and $T = 0.268$ year to expiry. Start with a change of variables $t = T - t'$ to form a backward-parabolic equation in time and propose an explicit scheme for the variables (S, t').

After giving some thoughts to the numerical stability of this solution, examine the transformation into normalized variables (x, τ) defined by $t = T - 2\tau/\sigma^2$, $S = E\exp(x)$ and use the ansatz

$$\begin{aligned} V(S,t) &= E\exp\left[-\frac{1}{2}(k_2-1)x - \left(\frac{1}{4}(k_2-1)^2 x + k_1\right)\tau\right] u(x,\tau) \quad (2.6\#\text{eq.}2) \\ k_1 &= 2r/\sigma^2, \qquad k_2 = 2(r - D_0)/\sigma^2 \end{aligned}$$

to reduce the original Black-Scholes equation to a standard diffusion problem

$$\frac{\partial u}{\partial \tau} - \frac{\partial^2 u}{\partial x^2} = 0 \qquad (2.6\#\text{eq.}3)$$

Derive initial and boundary conditions and implement the improved scheme in the JBONE applet. Compare with the Monte-Carlo solution previously developed in this course now running very nicely as an applet

2.6 Particle in a periodic potential

The quantum mechanics for particles in a periodic potential is a cornerstone of Solid State Physics (see e.g. Ashcroft and Mermin [13], chapters 8 and 9).

It is shown from perturbative theory that the addidtion of a weak periodic potential modifies the energy levels of the particle as compared to the free particle case $E = h^2k^2/2m$. The most dramatic effect is felt for wavenumbers k close to a Bragg plane, for which a forbidden gap in allowed energy occurs. If L is the periodicity of the potential, $K = 2p/L$ is called a reciprocal lattice vector, and the Bragg plane is at $k = K/2$. Thus the dispersion relation for particles in a weak potential is such that the group velocity vanishes at the Bragg plane.

The aim of this exercise is to demonstrate this property. Modify the scheme for the Schroedinger equation (2.5.2#eq.18) to include a periodic potential. Choose the period length for the potential and an amplitude set by `PhysDataValue`. See how *plane wave* particles propagate when their wavelength is equal to the Bragg condition, as compared to when their wavelength differs from that. Then choose a wave packet. Document your observations and relate them to the theory.

2.7 Further Reading

- **Eulerian schemes for advection-diffusion problems.**
  Numerical Recipes [4] §19.1–19.2, Boris [14]

- **FDTD leapfrog for Maxwell's equation.**
  Jin [15], EMLIB homepage[12] [16]

- **Direct and iterative methods for solving linear systems.**
  Numerical Recipes [4] §2.4, Dahlquist [6] §6.3, §11.1–11.4, Saad [17]

- **Option pricing.**
  Wilmott [18], Björk [19], Duffie [20], Rebonato [21]

2.8 Solutions

2.5. Option pricing. It is really not possible to give here a complete introduction into the broad subject of option pricing; simple explanations can be found on-line in a tutorial by the Chicago Bord of Exchange[13]; a more thorough analysis is given in books by Wilmott [18] and Rebonato [21]. For this excercise, it is sufficient to understand that an *European Vanilla put option* is a contract giving its owner the right to sell a certain asset (called the *underlying*) for a fixed price (E the *exercise* or *strike* price) at a prescribed time T in the future (the *expiry* date).

The parameters of the exercise have been chosen for the underlying QQQ index (NASDAQ top 100) on the evening of May 24, 2000 when QQQ was traded for $S = 79.5$ USD (alternatively, you can take the value from today and repeat the exercise with updated values). Imagine you are the owner of a put option allowing you to sell a QQQ share for $E = 95.00$ USD on September 1st, 2000: you may want to know how much this right was worth on May 24, 2000, i.e. $T = 0.268$ year before expiry (alt. take today).

[12] http://emlib.jpl.nasa.gov/
[13] http://www.cboe.com

At expiry on September 1st, 2000, it is clear that the value of the put depends only on the price S of the underlying share and is given by the payoff function $P(S, t = 0) = \max(E - S, 0)$ given as initial condition to the JBONE applet: the put option is worthless if QQQ is traded above $S = 95$ USD (an option is a right you are allowed not to exercise) and you will earn the price difference if QQQ is traded below (you can buy QQQ shares for e.g. $S = 75$ USD on the market and sell them without taking any risk for the strike price of $E = 95$ USD with a net profit of 20 USD per share).

Two factors at least can change the price of this right from May to September:

1. If you give your money to a bank, you can earn an interest rate of say $r = 0.05$ on your deposit; you should therefore discount this systematic (risk free) return from the value of the option during the entire period your money is invested.
2. The market (and in particular NASDAQ) is volatile. Prices change constantly with what may be modeled as a random component in the price of a share: even if chance are slim, you may still earn money with a put option at 95 USD if QQQ is above 100 USD in August... if the market crashes, making you rich! Clearly, options keep a finite value until they expire depending on the underlying volatility[14]. For QQQ, this was $\sigma = 0.7155$ in May (alt. take the value today).

The Black & Scholes model takes these factors into account and integrates the payoff function *backwards in time* to model the price of an option before it expires.

Using an explicit 2 levels spatially centered finite difference scheme, eq.2.6#eq.1 can naively be written as

$$V^{t+\Delta t} = V^t + \Delta t \left[\sigma^2 (\frac{1}{2} j \Delta S)^2 \frac{V_{j+1} - 2V_j + V_{j-1}}{\Delta S^2} + r(j\Delta S) \frac{V_{j+1} - V_{j-1}}{2\Delta S} - rV_j \right] = 0 \quad (2.8\#\text{eq}.1)$$

and cast into

$$V^{t+\Delta t} = \frac{\Delta t}{2}(\sigma^2 j^2 + rj)V_{j+1} + (1 - \sigma^2 j^2 \Delta t + r\Delta t)V_j + \frac{\Delta t}{2}(\sigma^2 j^2 - rj)V_{j-1} \quad (2.8\#\text{eq}.2)$$

This has been implemented in JBONE as:

```
double E      = runData.getInitialShapePosition(); //Exercise price
double sigmaSq= diffusCo;                          //Volatility square
double rate   = velocity;                          //Interest rate

fp[0]=E*Math.exp(-rate*time);                      //Boundary condition
for (int i=1; i<n; i++) {
  fp[i]=f[i+1]* 0.5*timeStep*(sigmaSq*i*i +rate*i) //Explicit 2 levels
       +f[i  ]*(1.0-timeStep*(sigmaSq*i*i +rate  ))
       +f[i-1]* 0.5*timeStep*(sigmaSq*i*i -rate*i);
}
fp[n]=fp[n-1]+dx[0]*(fp[n-1]-fp[n-2]);             //Boundary condition
```

Press `Start/Stop` to start the (slow) integration backward in time. Increase the time step by a factor 2: after only about 40 steps, a numerical instability develops that can be traced down to a violation of the stability criterion $\sigma^2 \Delta t / \Delta x^2 < 1/2$ (eq.2.1#eq.4).

[14] http://www.cboe.com/tools/historical/vol0400.txt

The problem with this naive approach is that the random changes in the asset prices δS are in reality lognormally distributed, so that the natural mesh for the a random walk should in fact be equally spaced in $\log S$ rather than S. The time step is therefore limited by the largest asset price, where the relative mesh intervals ΔS is smallest.

To avoid instabilities and negative values for large asset prices, it is possible to change from financial (S, t') to lognormal variables (x, τ) and evolve the standard diffusion equation (eq.2.6#eq.3). An interpolation back to financial variable is only required for diagnostic purposes. This has been implemented in JBONE as:

```
double E      = runData.getInitialShapePosition(); //Exercise price
double sigmaSq= diffusCo;                          //Volatility square
double rate   = velocity;                          //Interest rate
double divid  = disperCo;                          //Dividend
int    j;
double x0, x1, f0, f1, xi;                         //Change variables
double tau = 0.5*sigmaSq*time;                     //  f(x,t) ->
double dtau= 0.5*sigmaSq*timeStep;                 //  fm(xx,tau)
double xx0 = Math.log(x[1]/E);
double dxx =(Math.log(x[n]/E)-Math.log(x[1]/E))/(n-1);
double k1 = 2*rate/sigmaSq;
double k2 = 2*(rate-divid)/sigmaSq;
double k2m1 = k2-1.;

//Interpolate from financial (x,t) to lognormal variables (xx,tau)
if (time <=timeStep) {
  j=1; ; x0=xx0;
  f0=f[1]/E*Math.exp(0.5*k2m1*x0+(0.25*k2m1*k2m1+k1)*tau);
  x1=x0; f1=f0; xi=x0;
  for (int i=1; i<n; i++) { //Loop over lognormal mesh index
    xi=xx0+(i-1)*dxx;
    while (xi>=x1) { j++; x0=x1; f0=f1; x1=Math.log(x[j]/E); }
    f1=f[j]/E*Math.exp(0.5*k2m1*x1+(0.25*k2m1*k2m1+k1)*tau);
    fm[i]= f0 + (xi-x0)*(f1-f0)/(x1-x0);
  }
  fm[n]= fm[n-1] + dxx*(fm[n-1]-fm[n-2]);
} else { //Retrieve fm[] from previous time step }


//Solve diffusion equation with an explicit 2 levels scheme
double D = dtau/(dxx*dxx);
for (int i=2; i<n; i++)
  f[i]= fm[i] + D*(fm[i+1]-2.*fm[i]+fm[i-1]);
f[1]= Math.exp(0.5*k2m1*xx0+0.25*k2m1*k2m1*tau); //Boundary cond.
f[n]= f[n-1] + dxx*(f[n-1]-f[n-2]);


//Interpolate back from lognormal to financial mesh variables
fp[0]=E*Math.exp(-rate*time);                    //Analytically
j=1; x0=x[0]; x1=x0; f0=fp[0];
xi=xx0; f1=f[1]*E/Math.exp(0.5*k2m1*xi+(0.25*k2m1*k2m1+k1)*tau);
for (int i=1; i<n; i++) { //Loop over financial mesh index
  while (x[i]>=x1)
    {j++;x0=x1;f0=f1;xi=xx0+(j-1)*dxx;x1=E*Math.exp(xi);}
  f1=f[j]*E/Math.exp(0.5*k2m1*xi+(0.25*k2m1*k2m1+k1)*tau);
```

```
  fp[i]= f0 + (x[i]-x0)/(x1-x0)*(f1-f0);
}
xi=Math.log(x[n]/E);
fp[n]=f[n]*E/Math.exp(0.5*k2m1*xi+(0.25*k2m1*k2m1+k1)*tau);
```

Using lognormal variables, the stability limit is the same for all prices; the maximum value can therefore be considerably larger and the problem with negative payoffs for large asset prices is cured. After the first step, you can clearly see the numerical interpolation error induced by changing to lognormal variables and back; higher precision is achieved by refining the mesh, but keep in mind that the scheme remains subject to the stability condition (eq.2.1#eq.4).

After these numerical considerations, you are ready to calculate what was the price of your put option on May 24, 2000 (alt. today).

1. Edit the value of the
   - interest rate $r \rightarrow$ `Velocity = 0.05`,
   - volatility square $\sigma^2 = 0.7155^2 = 0.512 \rightarrow$ `Diffusion = 0.512`,
   - time to expiry of 98 days = 0.268 year $\rightarrow$ `Run time = 0.268`
   - strike price $\rightarrow$ `ICPosition = 95.`
   - largest asset price $\rightarrow$ `Mesh length = 130.`
2. Press `Initialize` and then `Start/Stop` for the calculation
3. Select `Data to java console` in the applet top right selector, press `Step 1` and switch back to `Double click to edit below`
4. Open the java console (in NETSCAPE, select `Communicator->Tools->Java console`)

With an underlying QQQ traded for $S = 79.5$ USD on the evening of May 24, 2000, the nearest grid point approximation from `x[79]=80.23` shows that the price of the put option will finally reach `g[79]=14.77` on September 1st, but was still much higher on May 24 `f[79]=21.82` because of the so-called *time value* associated with the market volatility. This predicted value of 21.82 USD can finally be compared with the value set by the market, which was 20.875 on the evening of May 24, 2000 (alt. consult the current market price on the web). The agreement is certainly not bad given the crude approximations made for the input parameters... and remember: the Black-Scholes is only a model of what the market really does!

2.9 Interactive evaluation form

Your anonymous opinion is precious. Please fill-in our anonymous form in the web edition. Thank you very much in advance for your collaboration!

3 FINITE ELEMENTS METHODS

3.1 Mathematical background

To approximate a set of linear partial differential equations

$$\mathbf{L}\vec{v} = \vec{r} \quad \text{in} \quad \Omega \tag{3.1\#eq.1}$$

for an unknown $\vec{v} \in \mathcal{V} \subset \mathcal{C}^n(\Omega)$ continuously defined with n derivatives in the volume Ω and subject to the boundary conditions

$$\mathbf{B}\vec{v} = \vec{s} \quad \text{in} \quad \partial\Omega \tag{3.1\#eq.2}$$

a mathematician would probably first involve

Weighted residuals. Having defined a scalar product $(\cdot,\cdot)$ and a norm $||\cdot||$, the calculation essentially amounts to the minimization of a residual vector

$$||\vec{R}|| = ||\vec{r} - \mathbf{L}\vec{v}|| \tag{3.1\#eq.3}$$

which can be carried out using tools from the variational calculus.

Variational form. A quadratic form is constructed for that purpose by choosing a *test function* $\vec{w}$ in a sub-space $\mathcal{W}$ that is "sufficiently general" and satisfies the boundary conditions. The linear equation (3.1#eq.1) can then be written as an equivalent variational problem

$$(\vec{w}, \vec{R}) = (\vec{w}, \mathbf{L}\vec{v} - \vec{r}) = 0 \qquad \vec{v} \in \mathcal{V}, \;\; \forall \vec{w} \in \mathcal{W} \tag{3.1\#eq.4}$$

Integration by parts. If $\vec{w}$ is differentiable at least once $\mathcal{W} \subset \mathcal{C}^1(\Omega)$, the regularity required by the forth-coming discretization can often be relaxed by partial integrations. Using $\mathcal{L} = \nabla^2$ for illustration, Leibniz' rule states that

$$\nabla \cdot (v\vec{w}) = (\nabla v) \cdot \vec{w} + v\nabla \cdot \vec{w} \quad \Longrightarrow \quad \nabla v \cdot \vec{w} = -v\nabla \cdot \vec{w} + \nabla \cdot (v\vec{w}) \tag{3.1\#eq.5}$$

Integrating over the volume Ω and using Gauss' divergence theorem

$$\int_\Omega \nabla \cdot \vec{F} \, dV = \int_{\partial\Omega} \vec{F} \cdot d\vec{S}$$

yields a generalized formula for partial integration:

$$\int_\Omega \nabla v \cdot \vec{w} \, dV = -\int_\Omega v\nabla \cdot \vec{w} \, dV + \int_{\partial\Omega} v\vec{w} \cdot d\vec{S} \tag{3.1\#eq.6}$$

For the special case where $\vec{w} = \nabla u$, this is known as Green's formula

$$\int_\Omega v\nabla^2 u \, dV = -\int_\Omega \nabla v \cdot \nabla u \, dV - \int_{\partial\Omega} v\nabla u \cdot d\vec{S} \tag{3.1\#eq.7}$$

Note that the last (surface-) term can sometimes be imposed to zero (or to a finite value) when applying so-called *natural boundary conditions*.

Numerical approximation. It turns out that the formulation as a variational problem is general enough that the solution $\vec{v}$ of (3.1#eq.4) remains a converging approximation of (3.1#eq.1) even when the sub-spaces $\mathcal{V}, \mathcal{W}$ are *restricted* to finite, a priori non-orthogonal, but still complete sets $\overline{\mathcal{V}}, \overline{\mathcal{W}}$ of functions; the overlap integrals between these functions can then be handled simply as linear algebra by the computer.

In general, the discretized solution $\vec{v}$ is expanded in *basis functions* $\vec{e_j} \in \overline{\mathcal{V}} \subset \mathcal{V}$ which either reflect a property of the solution (e.g. the operator Green's function in the *Method of Moments*), or which are simple and localized enough so that they yields cheap inner products $(\vec{w}, \vec{e_j})$ and sparse linear systems (e.g. the roof-top function for the linear *Finite Element Method*). Different discretizations are possible also for the *test functions* $\vec{w}$. Among the most popular choices is the *Galerkin* method where test and basis functions are both chosen from the same sub-space $\overline{\mathcal{V}} \equiv \overline{\mathcal{W}}$; the method of *collocation* consists in taking $\vec{w} \in \overline{\mathcal{W}} = \{\delta(\vec{x} - \vec{x_j})\}, j = 1, N$ which yields a point-wise evaluation of the integrand on a discrete mesh $\{\vec{x_j}\}, j = 1, N$.

3.2 An engineer's formulation

After a section of what a physicist believes might be a mathematician's view of the subject, it is time for an example. Using the format of a "recipe" applicable to a rather broad class of practical problems, this shows how the advection-diffusion problem (1.3.2#eq.2) is formulated using Galerkin linear finite elements (FEMs) and how it is implemented in the JBONE applet.

Derive a weak variational form. "Multiply" your equation by $\forall g \in \mathcal{C}^1(\Omega), \int_\Omega dx\ g^*(x)$ where the conjugation is necessary only if your equation(s) is (are) complex:

$$\forall g \in \mathcal{C}^1(\Omega), \qquad \int_{x_L}^{x_R} dx \quad g\left[\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} - D\frac{\partial^2 f}{\partial x^2}\right] = 0 \qquad (3.2\#\text{eq.1})$$

Integrate by parts. To avoid having to use quadratic basis functions for the discretization of the second order diffusion operator, you now integrate by parts:

$$\int_{x_L}^{x_R} dx \quad \left[g\frac{\partial f}{\partial t} + ug\frac{\partial f}{\partial x} + D\frac{\partial g}{\partial x}\frac{\partial f}{\partial x}\right] - Dg\frac{\partial f}{\partial x}\bigg|_{x_L}^{x_R} = 0 \quad \forall g \qquad (3.2\#\text{eq.2})$$

Assuming a periodic domain, the surface term can here be cancelled, imposing so-called *natural boundary conditions*.

Discretize time. This can be formulated in general using a partially implicit scheme $f = (1-\theta)f^t + \theta f^{t+\Delta t}$, where $\theta \in [1/2; 1]$:

$$\int_{x_L}^{x_R} dx \left[g\left(\frac{f^{t+\Delta t} - f^t}{\Delta t}\right) + ug\frac{\partial}{\partial x}\left[(1-\theta)f^t + \theta f^{t+\Delta t}\right] + D\frac{\partial g}{\partial x}\frac{\partial}{\partial x}\left[(1-\theta)f^t + \theta f^{t+\Delta t}\right]\right] = 0$$

$$\int_{x_L}^{x_R} dx \left[\frac{g}{\Delta t} + \theta ug\frac{\partial}{\partial x} + \theta D\frac{\partial g}{\partial x}\frac{\partial}{\partial x}\right] f^{t+\Delta t} = \int_{x_L}^{x_R} dx \left[\frac{g}{\Delta t} - (1-\theta)ug\frac{\partial}{\partial x} - (1-\theta)D\frac{\partial g}{\partial x}\frac{\partial}{\partial x}\right] f^t \qquad (3.2\#\text{eq.3})$$

$\forall g$, where all the unknowns have been reassembled on the left. Re-scale by Δt, and

Discretize space using a linear FEMs expansion and a Galerkin choice for the test function:

$$f^t(x) = \sum_{j=1}^{N} f_j^t e_j(x), \qquad \forall g \in \{e_i(x)\}, \;\; i = 1, N \tag{3.2\#eq.4}$$

$$\begin{aligned} \int_{x_L}^{x_R} dx \quad & \left[e_i \sum_{j=1}^{N} f_j^{t+\Delta t} e_j + \Delta t \theta e_i u \sum_{j=1}^{N} f_j^{t+\Delta t} \frac{\partial e_j}{\partial x} + \Delta t \theta D \frac{\partial e_i}{\partial x} \sum_{j=1}^{N} f_j^{t+\Delta t} \frac{\partial e_j}{\partial x} \right] = \\ \int_{x_L}^{x_R} dx \quad & \left[e_i \sum_{j=1}^{N} f_j^{t} e_j + \Delta t (\theta - 1) e_i u \sum_{j=1}^{N} f_j^{t} \frac{\partial e_j}{\partial x} + \Delta t (\theta - 1) D \frac{\partial e_i}{\partial x} \sum_{j=1}^{N} f_j^{t} \frac{\partial e_j}{\partial x} \right] \\ & \forall i = 1, N \end{aligned} \tag{3.2\#eq.5}$$

Note how the condition $\forall g \in \mathcal{C}^1([x_L; x_R])$ is now used to create as many independent equations $i = 1, N$ as there are unknowns $\{f_j^{t+\Delta t}\}, j = 1, N$. All the *essential boundary conditions* are imposed by allowing $e_1(x)$ and $e_N(x)$ to overlap in the periodic domain. Since only the basis and test functions $e_i(x), e_j(x)$ and perhaps the problem coefficients $u(x), D(x)$ remain space dependent, the discretized equations can all be written in terms of *inner products* for example of the form $(e_i, u e_j') = \int_{x_L}^{x_R} dx \quad u(x) e_i(x) e_j'(x)$. Reassembling them in matrix notation,

Write a linear system through which the unknown values from the next time step $f_j^{t+\Delta t}$ can implicitly be calculated in terms of the current values f_j^t

$$\sum_{j=1}^{N} \mathbf{A_{ij}} f_j^{t+\Delta t} = \sum_{j=1}^{N} \mathbf{B_{ij}} f_j^t \tag{3.2\#eq.6}$$

To relate this Galerkin linear FEM scheme with the code which has been implemented in the JBONE applet, it is necessary now to evaluate the integrals from the inner product; this is usually performed with a numerical quadrature.

3.3 Numerical quadrature and solution

Except when the PDE coefficients are singular and require a special treatment, the precision required for the numerical integration really depends on the type of FEMs; in general, it is sufficient to preserve the convergence rate guaranteed by the discretization. Using a *numerical quadrature* of the form

$$\int_a^b f(y) dy = \frac{b-a}{2} \sum_{i=1}^{n} w_i f(y_i) + R_n \tag{3.3\#eq.1}$$

$$y_i = \left(\frac{b-a}{2} \right) x_i + \left(\frac{b+a}{2} \right) \tag{3.3\#eq.2}$$

with the abscissas x_i, weights w_i and rests R_n given in the table 1 below it is possible exactly integrate polynomials with a degree $(p-1)$, simply by superposing n terms for which the integrand is evaluated at the specified location $y_i(x_i) \in [a; b]$ and weighted by the factor w_i. Using the powerful *Gaussian quadrature*, the product of two cubic FEMs will therefore require

scheme	n terms	$\pm x_i$	w_i	$R_n \sim \mathcal{O}(f^{(p)})$
mid-point	2	0	1	p=2
trapezoidal	2	1	1	p=2
Gaussian	2	$\sqrt{1/3}$	1	p=4
Gaussian	3	0	$8/9$	p=6
		$\sqrt{3/5}$	$5/9$	
Gaussian	4	$\sqrt{3/7+\sqrt{120}/35}$	$1/2-5/(3\sqrt{120})$	p=8
		$\sqrt{3/7-\sqrt{120}/35}$	$1/2+5/(3\sqrt{120})$	

Table 1: Quadrature abscissas and weights for the interval $x_i \in [-1;1]$

no more than three or four evaluations of the integrand in every interval of a unidimensional mesh.

With two linear FEMs, the mid-point and the trapezoidal rules are often both precise enough with only one evaluation per interval; although slightly more expensive with one extra evaluation per interval, they can nicely be combined into a so-called *tunable integration* [22]

$$\int_a^b f(y)dy = (b-a)\left[\frac{p}{2}\left[f(a)+f(b)\right]+(1-p)f\left(\frac{a+b}{2}\right)\right]+R_2, \quad p\in[0;1] \quad (3.3\#\text{eq.3})$$

It will become clear below, how a piecewise linear FEM discretization (obtained for $p=1/3$) can then be continuously changed into either an equivalent FD discretization (for $p=1$) or a piecewise constant FEM approximation (for $p=0$). Apart from the academic interest, this feature can sometimes be used to change the slope of the numerical convergence and even to stabilize an approximation which is marginally unstable because of the numerical discretization.

Armed with new tools to complete the linear FEM discretization from sect.3.2, we can now evaluate the matrix elements in (3.2#eq.5) using a tunable integration (3.3#eq.3). Since FEMs reach only as far as to the nearest neighbors, all the matrix elements $\mathbf{A_{ij}}$ with $|i-j|>1$ vanish, except those created by the periodicity on the domain boundaries. Using a sequential numbering of the unknowns $\{f_j^{t+\Delta t}\}, j=1,N$, this results in a *tri-diagonal structure* of the matrix, plus two extra elements in the upper-right and lower left corner from the periodic boundary conditions.

To keep the FEM implementation in JBONE as simple as possible, homogeneity is assumed $u\neq u(x)$, $D\neq D(x)$ and the matrix coefficients are calculated directly in terms of the inner products

$$\begin{aligned}
\int_{x_{i-1}}^{x_i} e_i e_i dx &= \int_{x_i}^{x_{i+1}} e_i e_i dx = (1+p)\frac{\Delta x}{4} \\
\int_{x_i}^{x_{i+1}} e_i e_{i+1} dx &= \int_{x_i}^{x_{i+1}} e_{i+1} e_i dx = (1-p)\frac{\Delta x}{4} \\
\int_{x_i}^{x_{i+1}} e_i e_i' dx &= -\int_{x_i}^{x_{i+1}} e_i e_{i+1}' dx = -\frac{1}{2} \\
\int_{x_i}^{x_{i+1}} e_i' e_i' dx &= -\int_{x_i}^{x_{i+1}} e_i' e_{i+1}' dx = \frac{1}{\Delta x}
\end{aligned} \quad (3.3\#\text{eq.4})$$

In a real code, this would be replaced by a call to the function returning a local value of the integrand, and combined with the summation from one of the quadratures described above.

Substituting back into (3.2#eq.5) finally yields the FEM scheme

$$\begin{pmatrix} (1-p)\frac{\Delta x}{4} - \theta\Delta t\frac{u}{2} - \theta\Delta t\frac{D}{\Delta x} \\ (1+p)\frac{2\Delta x}{4} + \theta\Delta t\frac{2D}{\Delta x} \\ (1-p)\frac{\Delta x}{4} + \theta\Delta t\frac{u}{2} - \theta\Delta t\frac{D}{\Delta x} \end{pmatrix}^T \cdot \begin{pmatrix} f_{i-1}^{t+\Delta t} \\ f_i^{t+\Delta t} \\ f_{i+1}^{t+\Delta t} \end{pmatrix} =$$
$$\begin{pmatrix} (1-p)\frac{\Delta x}{4} - (\theta-1)\Delta t\frac{u}{2} - (\theta-1)\Delta t\frac{D}{\Delta x} \\ (1+p)\frac{2\Delta x}{4} + (\theta-1)\Delta t\frac{2D}{\Delta x} \\ (1-p)\frac{\Delta x}{4} + (\theta-1)\Delta t\frac{u}{2} - (\theta-1)\Delta t\frac{D}{\Delta x} \end{pmatrix}^T \cdot \begin{pmatrix} f_{i-1}^{t} \\ f_i^{t} \\ f_{i+1}^{t} \end{pmatrix} \qquad (3.3\#\text{eq.5})$$

After initialization where the initial condition is discretized with trivial projection on piecewise linear "roof-top" FEMs, the scheme is implemented in JBONE using two tri-diagonal matrixes $\mathbf{a}$, $\mathbf{b}$ and a vector $\mathbf{c}$:

```
BandMatrix a = new BandMatrix(3, f.length);
BandMatrix b = new BandMatrix(3, f.length);
double[]   c = new double[f.length];

double h   = dx[0];
double htm = h*(1-tune)/4;
double htp = h*(1+tune)/4;

for (int i=0; i<=n; i++) {
  a.setL(i,   htm +h*(-0.5*beta -alpha)* theta    );
  a.setD(i,2*(htp +h*(           alpha)* theta   ));
  a.setR(i,   htm +h*( 0.5*beta -alpha)* theta    );
  b.setL(i,   htm +h*(-0.5*beta -alpha)*(theta-1) );
  b.setD(i,2*(htp +h*(           alpha)*(theta-1)));
  b.setR(i,   htm +h*( 0.5*beta -alpha)*(theta-1) );
}

c=b.dot(f);                                  //Right hand side
c[0]=c[0]+b.getL(0)*f[n];                    // with periodicity
c[n]=c[n]+b.getR(n)*f[0];

fp=a.solve3(c);                              //Solve linear problem
```

The BandMatrix.solve3() method is again used to compute a direct solution in $\mathcal{O}(N)$ operations with LU-factorization. The on-line document illustrates the advection of a box, calculated with a piecewise linear "roof-top" FEMs discretization ($p = 1/3$) slightly decentered in time ($\theta = 0.55$).

After a considerable effort spent in understanding this FEM discretization, isn't it frustrating to see how similar the code is with the implicit Crank-Nicholson FD scheme from sect.2.5? This should be the main argument for the community who still uses implicit finite difference schemes! With some *thinking* but the *same computational cost*, a finite element approach offers considerably more flexibility: it is now for example easy to *densify the mesh*[15], vary the *partially implicit* time integration from centered to fully implicit $\theta \in [\frac{1}{2}; 1]$ and *tune the integration* $p \in [0; 1]$. Convince yourself that Crank-Nicholson (2.5.1#eq.12) and this FEM scheme (3.3#eq.5) are strictly equivalent for a homogeneous mesh $x_j = j\Delta x$, $j = 1, N$, a time centered integration $\theta = \frac{1}{2}$ and a trapezoidal quadrature $p = 1$. Also take a couple of minute to experiment how you can affect the numerical dispersion / diffusion of short wavelengths by varying both the parameters $\theta \in [\frac{1}{2}; 1]$ and $p \in [0; 1]$.

[15] Although this has, for pedagogical reasons, here not been exploited

Numerical experiments:

- Vary the `TimeImplicit` and `TuneIntegr` parameters and determine which combinations yields the smallest numerical damping. Do your conclusions depend on the CFL number?
- Repeat this study to minimize the phase errors

3.4 Linear solvers

Writing efficient solvers for linear systems is a complete chapter of numerical analysis — and involves much more than what can be introduced here with a couple of sentences! As a user of software libraries such as Netlib [23] or PetSc [24], it is however sufficient to have a rough idea of what type of solvers exist and what they do.

Direct LU factorization. Remember that you should a priori never calculate a matrix inverse; you can solve a linear problem directly with only a third of the operations by decomposing it first into *lower and upper triangular* parts [3.7] with $\frac{2}{3}N^3$ operations $(*,+)$

$$\mathbf{A}\cdot\mathbf{x} = (\mathbf{L}\cdot\mathbf{U})\cdot\mathbf{x} = \mathbf{L}(\mathbf{U}\cdot\mathbf{x}) = \mathbf{b} \qquad (3.4\#\text{eq.}1)$$

and then solve a *forward-backward* substitution with $2N^2$ operations

$$\begin{aligned} y_1 &= \frac{b_1}{L_{11}};\ y_i = \frac{1}{L_{ii}}\left(b_i - \sum_{j=1}^{i-1} L_{ij}y_i\right),\ i = 2,...N \\ x_N &= \frac{y_N}{U_{11}};\ x_i = \frac{1}{U_{ii}}\left(y_i - \sum_{j=i+1}^{N} U_{ij}x_j\right),\ j = N-1,...1 \end{aligned} \qquad (3.4\#\text{eq.}2)$$

A particularly simple version has been implemented in JBONE for tri-diagonal matrices, where the solve3() method computes the solution in $\mathcal{O}(N)$ operations; the first and the last equations are simply eliminated "by hand" to take care of the periodicity. Many different implementations of one and the same algorithm exist and adapt the LU-factorization to specific non-zero patterns of $\mathbf{A}$; it is likely that Netlib has a routine already tailored for your application.

For matrices with more than three diagonals, it is important to allow for a *pivoting* during the factorization process — interchanging rows and columns to avoid divisions by small values created on the diagonal. For particularly large problems, note the possibility of storing most of the matrix on disk with a *frontal implementation* of the same algorithm.

If memory consumption, calculation time or the parallelization of a solver becomes an issue for 2D, 3D or even higher dimensions, it might be useful to consider *iterative methods* as an alternative to direct solvers. The idea behind them is best understood from a *splitting* of the matrix into a sum[16] of a diagonal $\mathbf{D}$ and strict lower $-\mathbf{E}$ and upper $-\mathbf{F}$ triangular parts

$$\mathbf{A} = \mathbf{D} - \mathbf{E} - \mathbf{F} \qquad (3.4\#\text{eq.}3)$$

[16] Don't get confused here with the product previously used for the LU factorization!

An iterative solution of the problem

$$(\mathbf{b} - \mathbf{A} \cdot \mathbf{x_{k+1}})_i = 0 \ , \quad i = 1, ...N \tag{3.4\#eq.4}$$

is then sought where $\mathbf{x_{k+1}} = \left(\xi_i^{(k+1)}\right)$ is the $(k+1)$-th iterate approximating the solution, with components ranging from $i = 1, ...N$.

Iterative Jacobi. The simplest form of iteration consists in inverting only the diagonal

$$a_{ii}\xi_i^{(k+1)} = \beta_i - \sum_{i \neq j} a_{ij}\xi_j^{(k)} \tag{3.4\#eq.5}$$

which is equivalent in matrix notation to

$$\mathbf{D} \cdot \mathbf{x_{k+1}} = (\mathbf{E} + \mathbf{F}) \cdot \mathbf{x_k} + \mathbf{b} \tag{3.4\#eq.6}$$

Starting from an arbitrary initial guess $\mathbf{x_0}$, simple elliptic problems will –albeit slowly– converge to a stationary point which is the solution of the linear problem.

Gauss-Seidel. As the equations $i = 1, ...N$ are updated one after the other, it is in fact possible to immediately use the updated information available in an explicit *forward* scheme

$$a_{ii}\xi_i^{(k+1)} + \sum_{j<i} a_{ij}\xi_j^{(k+1)} = \beta_i - \sum_{j>i} a_{ij}\xi_j^{(k)} \tag{3.4\#eq.7}$$

or

$$(\mathbf{D} - \mathbf{E}) \cdot \mathbf{x_{k+1}} = \mathbf{F} \cdot \mathbf{x_k} + \mathbf{b} \tag{3.4\#eq.8}$$

where the calculation is carried out with an equation index increasing from $i = 1, ...N$. The reverse is also possible and is called *backward* Gauss-Seidel

$$(\mathbf{D} - \mathbf{F}) \cdot \mathbf{x_{k+1}} = \mathbf{E} \cdot \mathbf{x_k} + \mathbf{b} \tag{3.4\#eq.9}$$

Successive over-relaxation — SOR is obtained when either of the Gauss-Seidel iterations (3.4#eq.8 or 3.4#eq.9) is linearly combined with the value of the previous step

$$\mathbf{x_{k+1}} = \omega \mathbf{x_k^{GS}} + (1 - \omega)\mathbf{x_k} \tag{3.4\#eq.10}$$

using the parameter $\omega \in [0; 1]$ for an *under-relaxation* or $\omega \in [1; 2]$ for an *over-relaxation*. A *symmetric-SOR (SSOR)* scheme can be obtained for the combination

$$\begin{cases} (\mathbf{D} - \omega\mathbf{E}) \cdot \mathbf{x_{k+1/2}} = [\omega\mathbf{F} + (1 - \omega)\mathbf{D}] \cdot \mathbf{x_k} + \omega\mathbf{b} \\ (\mathbf{D} - \omega\mathbf{F}) \cdot \mathbf{x_{k+1}} = [\omega\mathbf{E} + (1 - \omega)\mathbf{D}] \cdot \mathbf{x_{k+1/2}} + \omega\mathbf{b} \end{cases} \tag{3.4\#eq.11}$$

Preconditionning / approximate inverse. Realizing that all the methods above are of the form $\mathbf{x_{k+1}} = \mathbf{G} \cdot \mathbf{x_k} + \mathbf{f}$ where $\mathbf{G} = \mathbf{1} - \mathbf{M^{-1}} \cdot \mathbf{A}$, the linear problem approximatively be diagonalized with a preconditionning

$$\mathbf{M^{-1}} \cdot \mathbf{A} \cdot \mathbf{x} = \mathbf{M^{-1}} \cdot \mathbf{b} \tag{3.4\#eq.12}$$

$$\begin{aligned} \mathbf{M^{Jacobi}} &= \mathbf{D} & \mathbf{M^{SOR}} &= \frac{1}{\omega}(\mathbf{D} - \omega\mathbf{E}) \\ \mathbf{M^{GS}} &= \mathbf{D} - \mathbf{E} & \mathbf{M^{SSOR}} &= \frac{1}{\omega(2-\omega)}(\mathbf{D} - \omega\mathbf{E}) \cdot \mathbf{D^{-1}} \cdot (\mathbf{D} - \omega\mathbf{F}) \end{aligned} \tag{3.4\#eq.13}$$

Note that a product $\mathbf{s} = \mathbf{M^{-1}} \cdot \mathbf{A} \cdot \mathbf{x}$ for a given $\mathbf{x}$ can be calculated without ever forming the inverse, first with a product $\mathbf{r} = \mathbf{A} \cdot \mathbf{x}$ and then solving the linear system $\mathbf{M} \cdot \mathbf{s} = \mathbf{r}$ in sparse format.

All these variants of Gauss-Seidel are simple and work fairly well for reasonable sized (e.g. 20×20) elliptic problems, making the much more complicated *multigrid* approach attractive only for larger applications which have to be solved many times. Solving hyperbolic (wave-) problems and non-Hermitian operators iteratively is however more complicated and remains a matter of research [15], [25]. Methods rely in general both on a suitable *preconditionning* and the rapid convergence of Krylov-space and minimal residual methods such as *GMRES* or *TFQMR* [3.7].

It is important finally to note that the matrix-vector multiplication to form iterates can be performed in *sparse format*, i.e. using only those matrix elements which are different from zero. This is why iterative methods can be much more economical than direct solvers which fill the matrix during the LU factorization process — if they converge!

3.5 Variational inequalities

Consider the *obstacle problem*, which arises when an elastic string held fixed at both ends is pulled over a smooth object and you seek an equilibrium without a priori knowing where are the regions of contact between the string and this object. Define a function $S(x) \in \mathcal{C}^1(\Omega)$ to measure the elevation of the string in the interval $\Omega = [x_L; x_R]$ and $O(x) \in \mathcal{C}^1(\Omega)$ to model the shape of this object. The obstacle problem amounts to finding $S(x)$ from the conditions:

1. the string always remains above the obstacle $S(x) \geq O(x)$,
2. the string satisfies the equilibrium equation. Neglecting the inertia, this says that the string has either zero curvature (straight line between the points of contact) or a negative curvature $S'' \leq 0$ (line of contact – in other words, the obstacle can only push the string up, not pull it down).

Here are two manners for solving such problems that involve inequalities:

Linear complementary formulation. Reassemble all the conditions in a form

$$\mathcal{A}\mathcal{B} = 0, \qquad \mathcal{A} \geq 0, \qquad \mathcal{B} \geq 0 \tag{3.5\#eq.1}$$

After discretization, assuming that $\mathbf{A}$ invertible and positive definite, the linear problem

$$(\mathbf{x} - \mathbf{c})(\mathbf{A}\mathbf{x} - \mathbf{b}) = 0, \qquad \mathbf{A}\mathbf{x} \geq \mathbf{b}, \qquad \mathbf{x} \geq \mathbf{c} \tag{3.5\#eq.2}$$

can be solved with the so-called *projected SOR* method, replacing (3.4#eq.10) by

$$\mathbf{x_{k+1}} = \max[\mathbf{c}, \omega\mathbf{x_k^{GS}} + (1-\omega)\mathbf{x_k}] \tag{3.5\#eq.3}$$

and starting the iteration from an initial guess $\mathbf{x_0} \geq \mathbf{c}$.

For the obstacle problem, the string follows either a straight line above the obstacle $S'' = 0$ or fits exactly the object $S - O = 0$, suggesting how the complementary problem

$$S''(S-O) = 0 \qquad S - O \geq 0 \qquad -S'' \geq 0 \tag{3.5\#eq.4}$$

can be discretized with finite differences and solved using the pojected SOR method (exercise 3.5).

Variational formulation. This second approach is paticularly well suited for a discretization with finite elements is and best illustrated directly with the example. Choose a test function $\forall w \in \mathcal{V} \in \mathcal{C}^1(\Omega)$ that satisfies the same conditions as the solution $(w - c) \geq 0$. Having already $(S - c) \geq 0$ and $-S'' \geq 0$, write two inequalities

$$\int_{x_L}^{x_R} -S''(w-c) \geq 0$$
$$\int_{x_L}^{x_R} -S''(S-c) \geq 0$$

and substract them

$$\int_{x_L}^{x_R} -S''(w-S) \geq 0 \tag{3.5\#eq.5}$$

The condition c now appears only implicitly through the fact that w and S are members of the same sub-space $\mathcal{V}$. After the usual integration by parts and a discretization with linear FEMs, the linear problem can be solved with projected SOR (3.5#eq.3) iterations (exercises 3.5, 3.7).

3.6 Exercises

3.1 Quadrature

Calculate the integral $\int_0^\pi \sin(x)dx$ for a piecewise linear FEM approximation with two intervals, comparing the analytical result both with the tunable integration scheme (3.3#eq.3) and a two point Gaussian quadrature (3.3#eq.1) using the table 1 with $m = 2$.

3.2 Diffusion in a cylinder

Use a Galerkin linear FEM approximation to compute the diffusion of heat when a homogeneous cylinder of radius r_c, a conduction capacity κ and a radially inhomogeneous initial temperature $T_0(r)$ is kept isolated from the outside world. (Hint: solve the heat equation for the evolution of the temperature $T(r,t)$ in cylindrical geometry

$$\frac{\partial T}{\partial t} - \kappa \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T}{\partial r}\right) = 0 \tag{3.6\#eq.1}$$

Formulate a variational problem, integrate by parts and impose natural boundary conditions to guarantee a regular behavior of the solution in the limit $r \to 0$. Choose either a 2-points Gaussian or a trapezoidal quadrature and form a linear system of equations which you complete with boundary conditions on the cylinder surface $r = r_c$. Implement the scheme in the JBONE applet.

3.3 Mass lumping

Examine the possibility of "lumping the mass matrix", a procedure invented by structural engineers in the 1970's where the **mass matrix** ($\mathbf{A_{ij}}$) in (3.2#eq.6) is artificially set to unity to obtain an explicit scheme in terms only of the **stiffness matrix** ($\mathbf{B_{ij}}$). Study the advection problem separately from the diffusion.

3.4 Iterative solver

Implement an iterative–SOR solver in JBONE and study the number of iterations required to achieve a 10^{-6} precision for different values of the diffusion coefficient and advection velocity.

3.5 Obstacle problem

Compute a numerical solution for the *obstacle problem* from sect.3.5. For simplicity, take an obstacle parametrized by $O(x) = \frac{1}{2} - x^2$ and an elastic string $S(x)$ attached to the edge of the solution interval $S(-1) = S(1) = 0$. Hint: you may call the `BandMatrix.ssor3()` solver documented in the JBONE tree.

3.6 Numerical dispersion

Use a local ansatz $f(x,t) \sim \exp(-i\omega t)\sum_j \exp(ikx_j)e_j(x)$ to determine how a Galerkin discretization based on linear FEMs affects the dispersion of a wave in the standard wave equation (1.3.1#eq.2). Plot the phase velocity as a function of the numerical resolution).

3.7 American option

Extending your knowledge from exercise 2.5, use a Galerkin linear FEM formulation to solve the Black-Scholes equation for an *American put option*, which differs from the *European* in that it can be exercised anytime until it expires:

$$\int_{x_-}^{x^+} \frac{\partial u}{\partial \tau}(\phi - u) + \frac{\partial u}{\partial x}\left(\frac{\partial \phi}{\partial x} - \frac{\partial u}{\partial x}\right) dx \geq 0, \qquad \forall \phi \in \mathcal{W}, \quad \forall \tau \in [0; \frac{1}{2}\sigma^2 T] \quad (3.6\#\text{eq.}1)$$

Restrict $\phi \in \mathcal{W}$ to a piecewise linear test function which remains larger than the transformed payoff functions $\phi(x,\tau) \geq g(x,\tau)$ for all x and τ

$$\begin{aligned} g_{\text{put}}(x,\tau) &= \exp\left\{\left[\frac{1}{4}(k_2-1)^2 + 4k_1\right]\tau\right\} \max\left\{0, \exp\left[\frac{1}{2}(k_2-1)x - \frac{1}{2}(k_2+1)x\right]\right\} \\ g_{\text{call}}(x,\tau) &= \exp\left\{\left[\frac{1}{4}(k_2-1)^2 + 4k_1\right]\tau\right\} \max\left\{0, \exp\left[\frac{1}{2}(k_2+1)x - \frac{1}{2}(k_2-1)x\right]\right\} \end{aligned} \quad (3.6\#\text{eq.}2)$$

and satisfying the boundary conditions $\phi(x_+,\tau) = g(x_+,\tau)$, $\phi(x_-,\tau) = g(x_-,\tau)$, $\phi(x,0) = g(x,0)$. Implement the scheme in JBONE and compare with the solution previously obtained for the *European put* in exercise 2.5.

3.7 Further reading

- **Finite elements.**
  Johnson [8], Fletcher [7], Saad [17] §2.3
- **Quadrature.**
  Abramowitz [2] §25.4.29 with table 25.4, Numerical Recipes [4] §4.5
- **Linear solvers.**
  Dahlquist [6] §6, Saad [17], Numerical Recipes [4] §2.4.
  Software from `netlib`[17], `and PetSc`[18],

3.8 Interactive evaluation form

Your anonymous opinion is precious. Please fill-in our anonymous form in the web edition. Thank you very much in advance for your collaboration!

[17] http://www.netlib.org
[18] http://www.mcs.anl.gov/petsc

4 FOURIER TRANSFORM

4.1 Fast Fourier Transform (FFT) with the computer

As mentioned earlier in sect.1.4, it is largely thanks to the possibility of computing efficiently the Fourier transformation in $\mathcal{O}(N \log N)$ operations that FFT's can be considered as a viable alternative to solve partial differential equations. And here again, it is sufficient to have only a rough idea of the underlying process to efficiently use the routines from software libraries. Particularly for FFT's, you should *privilege vendors implementations* which have in general been optimized for the specific computer you are using. Since no library is available yet in JAVA, a couple of routines from Numerical Recipes [4] have been translated for JBONE into the FFT.java class, making as little modification as possible to the original code, enabling you to learn more about the algorithm directly from the book. The Complex.java class has been imported to illustrate how rewarding it is to work with Netlib [23] libraries.

Remember that a complex function $f(x)$ of period $L = 2\pi/K$ can be represented with a *complex series*

$$f(x) = \sum_{m=-\infty}^{\infty} c_m \exp(imKx), \qquad c_m = \frac{1}{L}\int_a^{a+L} f(x)\exp(-imKx)dx \qquad (4.1\#\text{eq.1})$$

On the other hand, you can use the *cosine – sine* form

$$f(x) = \frac{a_0}{2} + \sum_{m=1}^{\infty} [a_m \cos(mKx) + b_m \sin(mKx)] \qquad (4.1\#\text{eq.2})$$

$$a_m = \frac{2}{L}\int_a^{a+L} f(x)\cos(mKx)dx \quad (m \geq 0), \qquad b_m = \frac{2}{L}\int_a^{a+L} f(x)\sin(mKx)dx \quad (m \geq 1) \qquad (4.1\#\text{eq.3})$$

with the following relations that hold between the Fourier coefficients

$$\begin{array}{lll} c_0 = \frac{1}{2}a_0, & c_m = \frac{1}{2}(a_m - ib_m), & c_{-m} = c_m^*, \quad m \geq 1 \\ a_0 = 2c_0, & a_m = c_m + c_{-m}, & b_m = i(c_m - c_{-m}), \quad m \geq 1 \end{array} \qquad (4.1\#\text{eq.4})$$

and only if $f(x)$ is real

$$a_0 = 2c_0, \quad a_m = 2\mathrm{Re}(c_m), \quad b_m = -2\mathrm{Im}(c_m), \quad c_m = c_{-m}^*, \quad m \geq 1 \qquad (4.1\#\text{eq.5})$$

This is almost how the transformation is implemented in FFT.java, except that

1. the number of modes is *assumed to be an integer power of 2*. The creation of an FFT-object from data sampled on a homogeneous mesh relies on the command

   ```
   double f = new double[64];
   FFT myFTObject = new FFT(f, FFT.inXSpace);
   ```

   where `FFT.inXSpace` chooses the original location of the variable `myFTObject` of type `FFT`.

2. To *avoid negative indices* and relying on the periodicity $f(x + L) = f(x)$, negative harmonics $(-m)$ are stored *in wrap-around order* after the positive components and are indexed by $N/2 - m$. For a real function $f(x) = [7 + 8\cos(2\pi x/64) + 4\sin(2\pi x/32) + 3\cos(2\pi x/2)]$ sampled on a mesh $x_j = 0, 1, ...63$ with a period $L = 64$, the call to

```
Complex spectrum = new Complex[64];
double periodicity=64.;
spectrum = myFTObject.getFromKSpace(FFT.firstPart,periodicity);
for (int m=0;  m<N;  m++)
   System.out.println("spectrum["+m+"] = "+spectrum[m]);
```

will print the non-zero components

```
spectrum[0] = (7. +0.0i)
spectrum[1] = (4. +0.0i)          spectrum[63] = (4. +0.0i)
spectrum[2] = (0. +2.0i)          spectrum[62] = (0. -2.0i)
spectrum[32]= (3. +0.0i)
```

It is easy to see that the shortest wavelength component [32] will always be real if $f(x)$ was real, since $\sin(2\pi x/2)$ is sampled exactly for the multiples of π.

3. Relying on the linearity of the FFT, *two real numbers* are packed into one complex number and transformed for the cost of a single complex transformation by initializing

```
double f = new double[64];
double g = new double[64];
FFT myPair = new FFT(f,g, FFT.inXSpace);
```

This is the reason for the argument `FFT.firstPart` used here above, asking for the spectrum of only the first (and in this example the only) function.

Apart from the array index which starts with [0] in JAVA, the implementation is equivalent to the routines in Numerical Recipes [4] and indeed very similar to many computer vendors.

4.2 Linear equations.

Albeit slower for a single time-step than all the numerical schemes we have seen so far, an FFT can be used to compute the Fourier representation of the advection-diffusion problem (1.3.2#eq.2) and describe the evolution of each Fourier component f_m separately

$$\frac{d\widehat{f_m}}{dt} + ik_m u\widehat{f_m} + k_m^2 D\widehat{f_m} = 0, \qquad k_m = \frac{2\pi m}{L} \tag{4.2\#eq.1}$$

This can be integrated analytically *without any restriction on the size of the time-step*; starting directly from the initial condition, this yields for every Fourier component

$$\widehat{f_m}(t) = \widehat{f_m}(0)\exp[-(ik_m u + k_m^2 D)t)] \tag{4.2\#eq.2}$$

After an initialization, where the initial condition is discretized by sampling on a regular mesh and stored for subsequent transformation, the scheme implemented in JBONE reads:

```
int N         = mesh_.size();                      //A power of 2
double boxLen = mesh_.size()*mesh_.interval(0);    //Periodicity
double   k    = 2*Math.PI/boxLen;                  //Notation
Complex ik1   = new Complex(  0.,  k  );
Complex ik2   = new Complex(-k*k,  0.  );

Complex advection = new Complex(ik1.scale(velocity));  //Variables
Complex diffusion = new Complex(ik2.scale(diffusCo));
```

```
Complex[] s0 = new Complex[f.length];              //FFT real to KSpace
s0=keepFFT.getFromKSpace(FFT.firstPart,boxLen);    //  only once

s[0] = s0[0];
for (int m=1;  m<N/2+1;  m++) {                    //Propagate directly
  total=          advection.scale((double)(m ));   //  from initial
  total=total.add(diffusion.scale((double)(m*m))); //  contition

  exp=(total.scale(timeStep*(double)(jbone.step))).exp();

  s[m  ] = s0[m].mul(exp);                         //  s0 contains the IC
  s[N-m] = s[m].conj();                            //  f is real
}
FFT ffts = new FFT(s,FFT.inKSpace);                //Initialize in Kspace
f = ffts.getFromXSpacePart(FFT.firstPart,boxLen);  //FFT real back for plot
```

If you are a careful reader, you should now wonder about the sign of the phase factor, which is exactly the opposite from (4.2#eq.2); the reason is that the phase of the spatial harmonics is exactly opposite to the one used in the FFT routine from Numerical Recipes [4]. This makes the scheme look like as if it evolves backwards in time. Also note that once the initial condition has been transformed to K-space, subsequent transformations back to X-space are only required for the plotting.

The on-line document illustrates the advection of a box calculated with the same time step $\Delta t = 0.5$ as previously. The *Gibbs phenomenon* (artifact from using harmonic functions for the discretization discussed in sect.1.4.5) is clearly visible except when $u\Delta t/\Delta x$ is an integer; don't get fooled however by the plotting, which (for simplicity – but still incorrectly) misses parts of the oscillations visible in figure 1 by linear interpolation between the grid points. The power of the method is evident when taking larger time-steps: edit the parameter to `TimeStep=64`, add some diffusion `Diffusion=0.3` and compare the solution obtained in one single step with the result computed using your favorite FD or FEM scheme from the previous sections[19]. This is very nice indeed, but remember that dealing with an inhomogeneous medium $u(x)$, $D(x)$ or complicated boundary conditions are problematic in Fourier space.
Numerical experiments:

- Switch to `Finite Differences` and `Finite Elements` to measure the quality of previous schemes for `Advection=1`, `Diffusion=0.5` by comparing the results with the exact solution obtained with the `Fourier transform`. Hint: re-select the method that has just been used to re-scale the plot window and print the maximum at the end of the evolution.

- Initialize a `Gaussian` and determine the optimal numerical parameters (`MeshPoints`, `TimeStep`, `TimeImplicit`, `TuneIntegr`) for different schemes to achieve a 2% precision in the final value of the peak.

4.3 Aliasing, filters and convolution.

One of the beauties when using Fourier transforms, is the ability to work with a spectrum of modes and act on each of the components individually with a filter. By *sampling* a function over a period L with a finite number of values $N = L/\Delta$ where Δ is the size of the sampling

[19] Don't forget to reduce again the time-step!

interval, the spectrum gets truncated at the shortest wavelength $\lambda_c = 2\Delta = 2\pi/k_c$ called *Nyquist critical wavelength*, which corresponds to exactly 2 mesh points per wavelength. This does however not mean that shorter wavelengths $|k| > k_c$ do not contribute to the Fourier coefficients (4.1#eq.1). Figure 1 illustrates how they get *aliased* back into the lower components of the spectrum. This can be important for the digital data acquisition of an experiment,

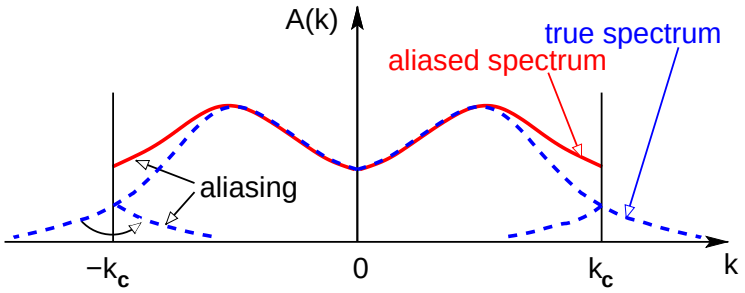


Figure 1: Aliasing from Fourier components shorter than the Nyquist critical wavelength $|k| > k_c$.

where the signal has to be *low-pass filtered before the sampling*. Figure 4.3 shows that even with the greatest precautions, such an aliasing can sometimes not be avoided, an needs then to be correctly interpreted.

It is extremely easy to design *a filter in Fourier space* simply by multiplying the spectrum by a suitable filter function $\mathcal{H}(k)$ (exercise 4.2). Simply remember that

- to keep the data real after transforming back to X-space, you must keep $\mathcal{H}(-k) = \mathcal{H}(k)^*$, for example by choosing $\mathcal{H}$ real and even in k,
- the filter has to be defined in the entire interval $k \in [-k_c; k_c]$ and should be smooth to avoid phase errors and dampings for wavelengths that appear with sharp edges.

Although they are present from the beginning when the initial condition is discretized (try to initialize and propagate an aliased cosine with a wavelength `ICWavelength=1.05` using the JBONE applet above), aliases do not actually interfere with the resolution of linear equations. The story is however different for spatial non-linearities such as the quadratic term $\partial_x f^2(x)$ that is responsible for the wave-breaking (1.3.4#eq.1). This can be understood from the *convolution theorem*, telling that the Fourier transform of a convolution $\widehat{f * g}$ is just the product of the individual Fourier transforms $\hat{f}\hat{g}$. The converse is unfortunately also true: what can be viewed as a simple product in X-space becomes a convolution in K-space

$$\widehat{f(x)g(x)} = \hat{f} * \hat{g}, \qquad \hat{f} * \hat{g} \equiv \int \hat{f}(k')\hat{g}(k-k')dk' \tag{4.3\#eq.1}$$

or in discrete form

$$(\hat{f} * \hat{g})_m \equiv \sum_{k=-N/2+1}^{N/2} \hat{f}_k \hat{g}_{m-k} \tag{4.3\#eq.2}$$

For the quadratic wave-breaking non-linearity, this shows that a short wavelength component such as $\hat{f}_{+31}$ in a sampling with 64 points, will falsely "pollute" a long wavelength channel

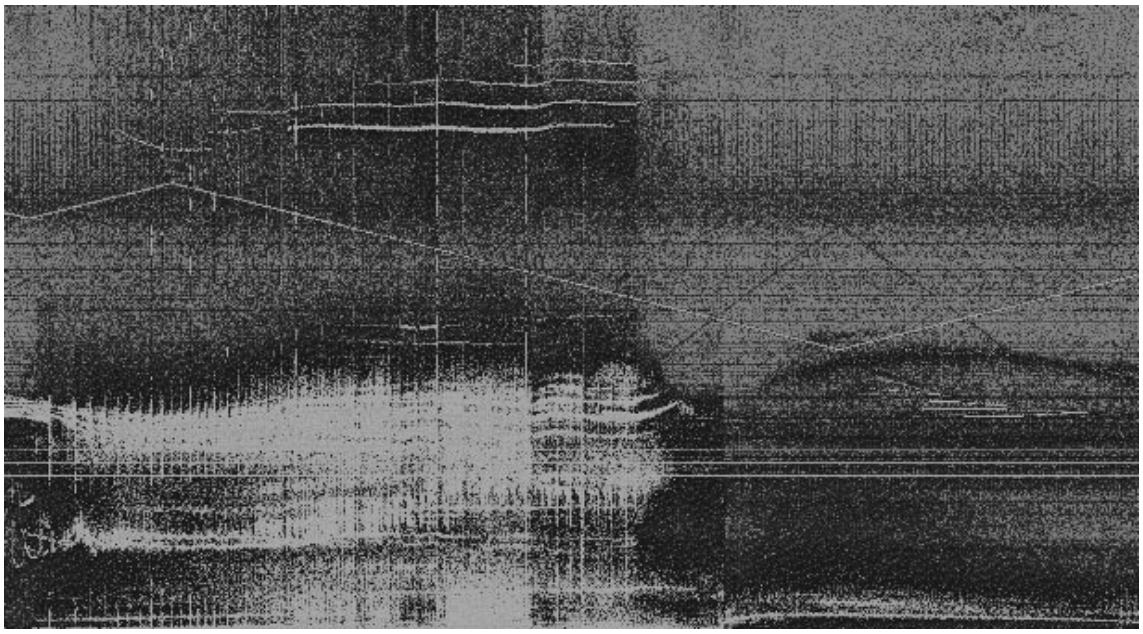


Figure 2: Experimental spectrum 0-500 kHz digitally recorded during 2 sec from the magnetic perturbations in a fusion plasma in the Joint European Torus. Apart from the Alfvén instabilities which are the subject of this research, you can see the sawteeth-like trace of an exciter antenna reaching a minimum of 200 kHz around 1.5 sec; despite heavy analogic low-pass filtering before the signal is sampled, the large dynamic range of the measurement above 80 dB is here sensitive enough to pick up (dark red line) a non-linearly induced high-frequency component which is *aliased* down into the frequency range of interest. Courtesy of Prof. A. Fasoli (MIT/USA).

through aliasing: $(\hat{f}_{+31} * \hat{f}_{+31}) = \widehat{f^2}_{+62} \longrightarrow \widehat{f^2}_{-2}$. A simple cure for this, is to expand the size of the arrays by a factor two before the convolution takes place and pad them with zeros; changing representation to calculate the multiplication of arrays twice the orgiginal size, the upper part of the spectrum is then simply discarded after the data has been transformed back. The entire procedure is illustrated in the coming section, where the non-linear Korteweg-DeVries (1.3.4#eq.3) and Burger equations (1.3.4#eq.2) are solved with a convolution in Fourier space.

4.4 Non-linear equations.

Combining the linear terms from advection (1.3.1#eq.1), diffusion (1.3.2#eq.1), dispersion (1.3.3#eq.1) and the non-linear wave-breaking term (1.3.4#eq.1) into a single non-linear equation, yields

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} - D\frac{\partial^2 f}{\partial x^2} + b\frac{\partial^3 f}{\partial x^3} + \frac{1}{2}\frac{\partial f^2}{\partial x} = 0 \qquad (4.4\#\text{eq.1})$$

where the last term has been written so as to explicitly show the quadratic non-linearity. After transformation to Fourier space, all the spatial operators become algebraic and an ordinary

evolution equation is obtained for each individual Fourier component

$$\frac{d\widehat{f_m}}{dt} + (ik_m u + k_m^2 D - ik_m^3 b)\widehat{f_m} + \frac{1}{2} ik_m \widehat{f_m^2} = 0 \qquad (4.4\#\text{eq.}2)$$

It would of course formally be possible to write the non-linear term as a convolution in K-space, but it is here much easier to write and efficient to compute the multiplication in X-space. Following the same lines as in sect.4.2, the linear terms are integrated analytically and yield a phase shift proportional to the time-step Δt. The convolution is calculated numerically by transforming back and forth from Fourier to configuration space, and after a simple Euler integration (1.2.1#eq.2) in time yields the formal solution

$$\widehat{f_m}^{t+\Delta t} = \widehat{f_m}^{t} \exp[-(ik_m u + k_m^2 D - ik_m^3 b)\Delta t] + \frac{\Delta t}{2} ik_m \widehat{f_m^2}^{t} \qquad (4.4\#\text{eq.}3)$$

This has been implemented in JBONE using the variable `keepFFT` to store the current values of spectrum $\widehat{f_m}^{t}$ and the variable `toolFFT` for the transformation to X-space required by the convolution and the plotting. As mentionned earlier in sect.4.2, the sign of time in (4.4#eq.3) has been changed to stick to the definition of the phase factor used in Numerical Recipes [4].

```
int N         = mesh_.size();                       //A power of 2
double boxLen = mesh_.size()*mesh_.interval(0);     //Periodicity
double   k    = 2*Math.PI/boxLen;                   //Notation
Complex ik1   = new Complex(  0.,  k  );
Complex ik2   = new Complex(-k*k,  0.  );
Complex ik3   = new Complex(  0.,-k*k*k);

Complex advection = new Complex(ik1.scale(velocity));  //Variables
Complex diffusion = new Complex(ik2.scale(diffusCo));
Complex dispersion= new Complex(ik3.scale(disperCo));

//----- Non-linear term: convolution
s  = keepFFT.getFromKSpace(FFT.bothParts,boxLen);  //Current Spectrum
toolFFT = new FFT(s,s,FFT.inKSpace);               //  for convolution

if (scheme.equals(jbone.ALIASED))                  //With-/out aliasing,
  sp = toolFFT.aliasedConvolution(boxLen);         //  use an FFT to
else //scheme.equals(jbone.EXPAND)                 //  calculate product,
  sp = toolFFT.expandedConvolution(boxLen);        //  FFT back to KSpace


//----- Linear terms: complex terms in spectrum s
s = keepFFT.getFromKSpace(FFT.bothParts,boxLen);   //Current Spectrum
linear= s[0];
sp[0]=linear;
for (int m=1;  m<=N/2;  m++) {
  total=          advection.scale((double)(m ));
  total=total.add(diffusion.scale((double)(m*m)));
  total=total.add(dispersion.scale((double)(m*m*m)));
  exp=(total.scale(timeStep)).exp();
  linear    = s[m].mul(exp);
  nonlin    = sp[m].mul(ik1.scale(0.5*timeStep*(double)(m)));
  sp[m  ]   = linear.add(nonlin);
  if (m<N/2) sp[N-m] = sp[m].conj();               //For a real spectrum
```

```
}

keepFFT = new FFT(sp,FFT.inKSpace);                //Spectrum is complete
toolFFT = new FFT(sp,FFT.inKSpace);                //inv FFT for plotting
f=toolFFT.getFromXSpacePart(FFT.firstPart,boxLen);
```

Depending on the `scheme` selector, the convolution is calculated either without precaution and is subject to aliasing, or by temporarily expanding the spectrum padding the upper part with zeros to cure the problem.

The on-line document shows the evolution obtained for the Korteweg-DeVries equation, when two solitons propagate and collide through the delicate balance between the non-linear wave-breaking and dispersion. Change the switch to `Aliased Convolution` and verify how the aliasing pollutes the spectrum with short wavelengths that rapidly evolve into a non-linear instability.

Replace the dispersion with a small amount of diffusion by setting `Dispersion=0.0` and `Diffusion=0.1`; evolve a Gaussian into a shock front and verify how much less aliasing seems to be an issue for the Burger equation (1.3.4#eq.2), when a diffusive process physically damps the short wavelengths... Remember however that the cascade of energy from one wavelength to another is now affected by the aliasing and is much more delicate to diagnose!
Numerical experiments:

- Go back to the KdV equation by setting `Dispersion=0.5` and try to cure the aliased scheme with a small amount of non-physical diffusion. Separate the short wavelengths (unphysical aliases) from the longuer wavelengths (physical) by reducing the `ICAmplitude`. Is such a solution attractive?

4.5 Exercises

4.1 Advection-diffusion

Propose an alternative scheme solving the linear advection-diffusion problem in Fourier space, evolving the solution with small steps in time Δt.

4.2 Equivalent filter for Zabusky's FD scheme

Study Zabusky's finite difference scheme for the Korteweg-DeVries equation

$$\frac{f_j^{t+\Delta t} - f_j^{t-\Delta t}}{2\Delta t} + \frac{1}{3}\left[f_{j+1}^t + f_j^t + f_{j-1}^t\right]\frac{f_{j+1}^t - f_{j-1}^t}{2\Delta x} + b\frac{f_{j+2}^t - 2f_{j+1}^t + 2f_{j-1}^t - f_{j-2}^t}{2\Delta x^3} = 0 \tag{4.5\#eq.1}$$

using JBONE with $b = 1/2$. Calculate the equivalent Fourier space filter that is implied for the linear terms when the calculation is performed in configuration space. Add this filter to the Fourier scheme available in JBONE and check that after filtering, both the FD and FT methods indeed are similar. What remains different? *Hint: Zabusky's finite difference scheme has already been implemented under this link and can be executed by selecting* `Finite differences`, `Explicit 3-level` *and* `KdV (solitons)` *in the applet selectors. Your task in this exercise is first to calculate and then to implement a filter function $\mathcal{H}_m$, which emulates for each Fourier component $f_m^{t+\Delta t} = \mathcal{H}_m f_m^t$ the evolution that results from the finite differencing of the linear terms.*

4.3 Prototype problems

Modify the parameters of the non-linear equation 4.4#eq.3 in Fourier space to develop a better qualitative understanding of what are advection, diffusion, dispersion and wave-breaking. Choose a regime where you trust the numerical description and propose a combination of parameters you believe is particularly interesting.

4.4 Intrinsic numerical diffusion

Use the exact solution of the advection-diffusion equation calculated with the Fourier method (4.2#eq.2) to measure the numerical damping $D_{\rm num}$ in FD / FEM schemes with different implicit-time and tunable-integration parameters. Start with $u_{\rm phys} = D_{\rm phys} = 1$ and, after evolving a narrow Gaussian, measure the final peak amplitude $\max[f](D_{\rm tot})$ for a decreasing value of the total diffusion $D_{\rm tot} = D_{\rm phys} + D_{\rm num}$. Calculate the intrinsic numerical damping $D_{\rm num}$ from the saturation you observe for small values of the physical damping $D_{\rm phys}$. *Hint: in JBONE, re-select the method that has just been used to re-scale the plot window and print the maximum of the solution.*

4.6 Further Reading

- **FFT algorithm.**
  Numerical Recipes [4] §12

4.7 Interactive evaluation form

Your anonymous opinion is precious. Please fill-in our anonymous form in the web edition. Thank you very much in advance for your collaboration!

5 MONTE-CARLO METHODS

5.1 Monte Carlo integration

The name stems from the city which is famous for its casinos and suggests a method involving random numbers. The most common use of Monte-Carlo methods (MCM) is the evaluation of multi dimensional integrals [4]. Consider first the approximation of an integral obtained with the trapezoidal rule (3.3#eq.1) by sampling on a uniform mesh

$$\int_a^b f(\xi)\,d\xi = \sum_{i=0}^{N-1} f(x_i)\frac{b-a}{N} \;+\; \mathcal{O}\left(\frac{1}{N}\right)\;, \qquad x_i = a + \frac{b-a}{N-1}i \tag{5.1\#eq.1}$$

Instead, you could use a mesh where the positions $\{x_i\}$ are random numbers uniformly distributed in the interval $[a,b]$. This suggests the Monte Carlo integration

$$\int_a^b f(\xi)\,d\xi = \sum_{i=1}^{N} f(x_i)\frac{b-a}{N} + \mathcal{O}\left(\frac{1}{\sqrt{N}}\right)\;, \qquad x_i \in \mathcal{U}(a,b) \tag{5.1\#eq.2}$$

The rate of convergence of this Monte-Carlo integration is lower than for the sampling on a uniform mesh $(5.1\#\text{eq.}1)$. The strength however appears for the evaluation of integrals in higher dimensions $d > 2$, where the MCM error scales as $N^{-1/2}$ irrespective of the number of dimensions, instead of the $\mathcal{O}\left(N^{-1/d}\right)$ obtained using a uniform mesh.

5.2 Stochastic theory

This section is intended as a very short introduction into the stochastic calculus that provides the mathematical foundation behind the Monte Carlo method. Check Kloeden [26] and Van Kampen [27] for complete courses on the subject!

Definition *The expected or mean value $\mathcal{E}$ and the variance $\mathcal{V}$ are defined by*

$$\mathcal{E}[X](x) \triangleq \int_{-\infty}^{\infty} x f_X(x)\,dx \tag{5.2\#eq.1}$$

$$\mathcal{V}[X](x) \triangleq \mathcal{E}\left[X\right]\left(\left(x - \mathcal{E}[X]\left(x\right)\right)^2\right) \tag{5.2\#eq.2}$$

$$= \mathcal{E}\left[X\right]\left(x^2\right) - \mathcal{E}^2\left[X\right]\left(x\right) \tag{5.2\#eq.3}$$

where $f_X(x)$ is the density distribution function of the stochastic variable X.

Definition *$\mathcal{N}(\mu,\sigma)$ refers to the set of Gaussian or normal distributed stochastic variable X with density distribution function*

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right] \tag{5.2\#eq.4}$$

where μ is the mean and σ^2 is the variance.

Definition *$\mathcal{U}(a,b)$ refers to the set of uniformly distributed random numbers in the interval $[a,b]$.*

Definition *A stochastic process W_t is called a Wiener process (or Brownian motion) in time t if and only if*

1. $W_{t+\Delta t} - W_t \in \mathcal{N}(0, \sqrt{\Delta t})$, *where* $\mathcal{N}$ *is the set of normal distributed random numbers.*
2. $\mathcal{E}[W_t dW_t] = \mathcal{E}[W_t]\mathcal{E}[dW_t]$, *where* $\mathcal{E}$ *denotes expected value,* $dW_t = W_{t+\Delta t} - W_t$ *and* $\Delta t > 0$, *i.e. the Wiener increment* dW_t *is independent of the past.*

The distribution function of a Wiener increment $W_{t+\Delta t} - W_t$ is essentially the same function as the Green's function of the diffusion equation (1.3.2#eq.3).

The differential calculus of stochastic processes, the so called *Itô calculus*, involves new properties that are fundamentally different from the ordinary Riemann calculus. The reason can be tracked down to the preferred direction of time t in the Itô integral:

Definition *The Itô integral is defined as the limit of an explicit (forward Euler) discretization*

$$\int_0^T b(W_t, t) \circ dW_t \equiv \sum_i \lim_{\Delta t_i \to 0} b(W_{t_i}, t_i)(W_{t_i + \Delta t_i} - W_{t_i}) \tag{5.2\#eq.5}$$

for any sequence $\{t_i : t_i \in [0, T], t_{i+1} = t_i + \Delta t_i\}$. *The circle,* $\circ$ *stands for Itô differential and states that* dW_t *is independent of* $b(W_t, t)$.

Note that an implicit discretization in (5.2#eq.5) would give a fundamentally different result. Now consider a property Y_t that is a sum of ordinary Riemann integral and an Itô integral.

$$Y_T = \int_0^T v(Y_t, t)dt + \int_0^T b(Y_t, t) \circ dW_t \tag{5.2\#eq.6}$$

Integrating over a infinitely short time interval we obtain the so called *stochastic differential equation*

$$dY_t = v(Y_t, t)dt + b(Y_t, t) \circ dW_t \tag{5.2\#eq.7}$$

which reduces to an ordinary differential equation if $b(Y_t, t) = 0$ in the absence of a stochastic component in the evolution

$$\frac{dY_t}{dt} = v(Y_t, t) \tag{5.2\#eq.8}$$

5.3 Particle orbits

When using the Monte Carlo method to solve PDEs, functions are discretized using quasi particles and differential operators need to be reformulated in terms of particle motions. In (1.4.7#eq.1) a quasi-particle is defined by a weight w_i, a position x_i and a shape function S_i. For simplicity, we assume here unit weights $w_i = 1$ and point shaped particles $S_i(x) = \delta(x)$ described by the Dirac pulse. The solution is computed from an ensemble of particle positions $\{x_i(t)\}$, that are called the *particle orbits*.

A deterministic orbit is described by an ordinary differential equation for the position $X(t)$ as a function of time t

$$\frac{dX(t)}{dt} = v(X(t), t) \tag{5.3\#eq.1}$$

or

$$dX(t) = v(X(t), t)dt \tag{5.3\#eq.2}$$

This equation can be solved numerically using the methods discussed in section 1.2.1, with an explicit or implicit discretization of time

$$X(t+\Delta t) = X(t) + v(X(t),t)\Delta t \qquad \text{“explicit”} \qquad (5.3\#\text{eq.3a})$$
$$X(t+\Delta t) = X(t) + v\left(X(t+\Delta t), t+\Delta t\right)\Delta t \qquad \text{“implicit”} \qquad (5.3\#\text{eq.3b})$$

For an ensemble of N particles with orbits $x_i(t)$, a particle density distribution function $f(x,t)$ is constructed according to section 1.4 and yields

$$f(x,t) = \sum_{i=0}^{N} \delta(x - x_i(t)) \qquad (5.3\#\text{eq.4})$$

Note that the first order moment is perfectly conserved, since no particle is lost.

Using a large number of particles, the density distribution function is then approximated by a smooth function. If the individual particle orbits $X(t)$ evolve according to

$$dX(t) = v(X(t),t)dt. \qquad (5.3\#\text{eq.5})$$

then the **Taylor's transport theorem** states that the evolution of particle density distribution function $f(x,t)$ is described by the advection equation

$$\frac{\partial f}{\partial t} + \frac{\partial}{\partial x}(vf) = 0 \qquad (5.3\#\text{eq.6})$$

This theorem makes it possible to study a PDE instead of an N-particle system; the reverse is also possible and used in particle methods involving the advection equation.

Let us now introduce the concept of **stochastic particle orbits**; that is an ensemble of possible orbits with different probabilities. As an example, picture a snowflake falling slowly from the sky: the motion is unpredictable and the evolution could be described as a stochastic particle orbit $X(t)$ following the stochastic differential equation

$$dX(t) = v(X(t),t)dt + b(X(t),t) \circ dW_t \qquad (5.3\#\text{eq.7})$$

where W_t is a Wiener processes (Brownian motion).

Starting from (5.3#eq.7), you will show in exercise 5.2 that

$$\frac{\partial}{\partial t}\mathcal{E}[X(t)](x) = v(x,t) \qquad (5.3\#\text{eq.8})$$
$$\frac{\partial}{\partial t}\mathcal{V}[X(t)](x) = b(x,t)^2. \qquad (5.3\#\text{eq.9})$$

where $\frac{\partial}{\partial t}\mathcal{E}[X(t)](x)$ is the average particle velocity, $\frac{\partial}{\partial t}\mathcal{V}[X(t)](x)$ is a measure of the broadening of the distribution of possible orbits. Eq. $(5.3\#\text{eq.7})$ can then be written as

$$dX_t = \frac{\partial}{\partial t}\mathcal{E}\left[X_t\right](x)dt + \sqrt{\frac{\partial}{\partial t}\mathcal{V}\left[X_t\right](x)} \circ dW_t \qquad (5.3\#\text{eq.10})$$

Example: Let $X(t)$ be a stochastic process with an evolution of the probability density distribution $f(x,t)$ following the advection-diffusion equation

$$\frac{\partial f(x,t)}{\partial t} + \frac{\partial}{\partial x}\left[a(x)f(x,t)\right] = \frac{\partial}{\partial x}\left(D(x)\frac{\partial f}{\partial x}\right), \quad x \in (-\infty, \infty) \qquad (5.3\#\text{eq.11})$$

From the derivations in exercise 5.1, we obtain

$$\frac{\partial}{\partial t}\mathcal{E}[X(t)](x) = a(X(t)) + \left(\frac{\partial}{\partial x}D(x)\right)_{x=X(t)} \tag{5.3\#eq.12a}$$

$$\frac{\partial}{\partial t}\mathcal{V}[X(t)](x) = 2D(X(t)) \tag{5.3\#eq.12b}$$

In the same spirit as in Taylor's transport theorem, it is possible to relate the evolution of a large number of stochastic particle orbits to a PDE. This is a form of the so-called **Feynman-Kac theorem**, tellings that a smooth density distribution function $f(x,t)$ in which the individual particles move according to

$$dX(t) = v(X(t),t)dt + b(X(t),t) \circ dW_t \tag{5.3\#eq.13}$$

evolves according to the Fokker-Planck (or Kolmogorov forward) equation

$$\frac{\partial f}{\partial t} = -\frac{\partial}{\partial x}(vf) + \frac{\partial^2}{\partial x^2}\left(\frac{b^2}{2}f\right) \tag{5.3\#eq.14}$$

which is in fact an advection-diffusion equation.

5.4 A scheme for the advection diffusion equation

Using the Feynman-Kac theorem, we solve a large number of stochastic particle orbits to approximate the advection diffusion equation. According to the Ito calculus, the time discretization has to be explicit

$$X_{t+\Delta t} = X_t + v(X_t,t)\Delta t + b(X_t,t)(W_{t+\Delta t} - W_t) \tag{5.4\#eq.1}$$

using the notation $X_t = X(t)$. It is also possible to construct implicit discretizations of a stochastic differential equation, but it not as straightforward as for ordinary differential equations [26]. Since $(W_{t+\Delta t} - W_t) \in \mathcal{N}\left(0, \sqrt{\Delta t}\right)$, the Wiener process can now be rewritten

$$W_{t+\Delta t} - W_t = \zeta\sqrt{\Delta t} \tag{5.4\#eq.2}$$

in terms of normally distributed in random numbers $\zeta \in \mathcal{N}(0,1)$. According to the central limit theorem, any sum of n equally distributed random numbers with zero mean and unit variance will eventually converge to $\mathcal{N}(0,\sqrt{n})$, for large n. Any such random number could therefore be used for ζ if the number of time steps is large; practically, $\zeta \in \mathcal{N}(0,1)$ leads to the fastest convergence.

Consider a N-particle ensemble; a numerical Monte Carlo scheme for the advection diffusion equation can now be constructed with particles evolving according to

$$X_{t+\Delta t} = X_t + v(X_t,t)\Delta t + \zeta b(X_t,t)\sqrt{\Delta t} \tag{5.4\#eq.3}$$

or

$$X_{t+\Delta t} = X_t + \frac{\partial}{\partial t}\mathcal{E}\left[X_t\right]\Delta t + \zeta\sqrt{\left(\frac{\partial}{\partial t}\mathcal{V}\left[X_t\right]\right)\Delta t} \tag{5.4\#eq.4}$$

which describes an ensemble of stochastic orbits, corresponding to the possible outcomes of the random variable ζ. With a large number of particles, it is possible to sample and approximate the entire ensemble of orbits.

In JBONE, this has been implemented as

```
for(int j = 0; j < numberOfParticles; j++){
  particlePosition[j] += velocity * timeStep +
    random.nextGaussian() *
    Math.sqrt(2 * diffusCo * timeStep);
  // Periodic boundary conditions

} // for
```

JAVA is one of the few programming languages that has a pseudo random numbers generator $\mathcal{N}(0,1)$. Most programming languages don't, but they usually have uniform pseudo random numbers $\mathcal{U}(0,1)$. Random numbers in $\mathcal{N}(0,1)$ can then be obtained from the **Box Müller method**

- Construct two uniformly distributed random numbers $U_1, U_2 \in \mathcal{U}(0,1)$.
- Then

$$N_1 = \sqrt{-2\ln(U_2)}\cos(2\pi U_1) \tag{5.4\#eq.5a}$$
$$N_2 = \sqrt{-2\ln(U_2)}\sin(2\pi U_1) \tag{5.4\#eq.5b}$$

  are two independent pseudo random numbers in $\mathcal{N}(0,1)$.

Example: 1D Diffusion equation. As an example, let us calculate a Monte Carlo approximation of the temperature in a 1D slab $u(x,t)$, which follows the equation

$$\frac{\partial u(x,t)}{\partial t} = D\frac{\partial^2}{\partial x^2}u(x,t) \qquad 0 \leqslant t \tag{5.4\#eq.6}$$

subject to the initial condition

$$u(x,0) = \begin{cases} 1 & \text{if } 0 < x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.4\#eq.7}$$

- Imagine N heat-particles, so that the local heat is given by the density of particles. According to the Feynman-Kac theorem and equation (5.4#eq.6) the position of the i'th particle evolves according to

$$dx_i(t) = \sqrt{2D} \circ dW_t, \quad i = 1, 2...N \tag{5.4\#eq.8}$$

  where W_t is a Wiener process.

- This is discretized as

$$x_i(t + \Delta t) = x_i(t) + \zeta\sqrt{2D\Delta t}. \tag{5.4\#eq.9}$$

- Randomize the initial positions $x_i(0)$ of the N particles, using a good pseudo random number generator in $\mathcal{U}(0,1)$.
- With the Box Müller method pseudo random numbers are obtained from $\zeta \in \mathcal{N}(0,1)$.
- For each particle calculate step by step the evolution $x_i(0) \to x_i(\Delta t) \to ...$
- The solution can then be visualized by projection as previously discussed in $(1.4.7\#\text{eq}.3)$.

5.5 When should you use Monte Carlo methods?

Monte Carlo methods are efficient for a large number of dimensions and complex geometries: table 1 suggests that Monte Carlo methods are more efficient than FD or FEM approximatively when $d/n > 2$ (in the limit of $N \to \infty$ and neglecting computer hardware issues).

Method	**Accuracy**
Monte Carlo	$\mathcal{O}\left(N^{-1/2}\right)$, N is the number of particles.
FEM/FD (n:th-order)	$\mathcal{O}\left(N^{-n/d}\right)$, N is number of grid points.

Table 1: Accuracy for a d-dimensional problem.

The MCM is also well suited for problems with complicated boundary conditions. Take for example a 3D cube with a ball bouncing inside. Let the cube contain a non collisional gas, so that the atoms bounce back from the surfaces of the cube and the ball. For simplicity let the cube and the ball have infinite mass during the collisions with the gas particles. The distribution of the gas atoms is fairly easy to compute with MCM, but nearly untraceable with a fluid method.

Parallelization is easy and efficient if the problem is linear and the particles are completely decoupled: just run a copy of the simulation program on several machines simultaneously. The final result is obtained simply by summation and normalization of the results from the individual machines, since the Monte Carlo time stepping is a diagonal linear operation $\mathcal{L} = \{\mathcal{L}_{ij}\}$

$$f(t+dt) = \mathcal{L}[f(t)] = \mathcal{L}[\sum_j f_j(t)] = \sum_{ij} \mathcal{L}_{ij}[f_j(t)] = \sum_i \mathcal{L}_{ii}[f_i(t)] \qquad (5.5\#\text{eq}.1)$$

where f_i is either the particle or its projection $(1.4.7\#\text{eq}.3)$. Simply be careful to seed the random numbers differently on the different machines — or the simulations will all be identical!

The MCM is however not that easy to parallelize for non-linear problems: if a and b are functions of the density distribution, the continuous density distribution function needs to be approximated after each step. This will dramatically reduce the performance on a parallel machine.

5.6 Exercises

5.1 Expectancy and variance

Calculate $\frac{\partial}{\partial t}\mathcal{E}[X(t)](x)$ and $\frac{\partial}{\partial t}\mathcal{V}[X(t)](x)$, where $X(t)$ is the position of a particle with a density distribution function $f(x,t)$, given by

$$\frac{\partial f(x,t)}{\partial t} + \frac{\partial}{\partial x}\left[a(x)f(x,t)\right] = \frac{\partial}{\partial x}\left(D(x)\frac{\partial f}{\partial x}\right), \quad x \in (-\infty, \infty) \qquad (5.6\#\text{eq}.1)$$

With initial condition $X(0) = x_i$, *ie* the density distribution is

$$f(x,0) = \delta(x - x_i) \tag{5.6\#eq.2}$$

Hint: The time derivative of $\mathcal{E}$ is obtained by

$$\frac{\partial}{\partial t}\mathcal{E}[X(t)](x) = \frac{\partial}{\partial t}\int_{-\infty}^{\infty} f(x,t)x\,dx = \int_{-\infty}^{\infty}\frac{\partial f(x,t)}{\partial t}x\,dx \tag{5.6\#eq.3}$$

Use partial integration to remove the derivatives of the Dirac distribution.

5.2 Diffusion statistics

Use the stochastic differential equation (5.3#eq.7) and the definitions

$$\begin{aligned}\frac{\partial}{\partial t}\mathcal{E}[X(t)](x) &= \lim_{dt\to\infty}\frac{1}{dt}\mathcal{E}[dX(t)]\\ \frac{\partial}{\partial t}\mathcal{V}[X(t)](x) &= \lim_{dt\to\infty}\frac{1}{dt}\mathcal{E}[dX(t)^2]\end{aligned}$$

to calculate the average particle velocity (5.3#eq.8) and the broadening of the distribution of possible orbits (5.3#eq.9) in a random walk. Combine this with the results from the previous excercise, and show that the advection-diffusion equation (5.3#eq.11) is related to the Fokker-Planck equation (5.3#eq.14).

Comment: combined with the previous exercise, you have shown that the two first moments of a density distribution function are treated in an equivalent manner by the stochastic differential equation (5.3#eq.7) and by the advection diffusion equation (5.3#eq.14). If you would extend this to include all moments, you would prove the Feynman-Kac theorem.

5.3 Periodic boundary conditions

Add periodic boundary conditions to the Monte Carlo solver in the JBONE applet.

Hint: for every particle lost on the right an identical particle should enter from the left. Remember that a kick might be larger than the length of simulation domain,

```
double[] lim = {mesh_.point(0),
                mesh_.point(mesh_.size() - 1) + dx[0]};
```

which ranges from `lim[0]` on the left to `lim[1]` on the right (`lim[1]` includes the extra mesh cell connecting consecutive domains, which is not plotted).

5.4 Steady state with velocity gradient

Simulate the equation

$$\frac{\partial f}{\partial t} = -\frac{\partial}{\partial x}\left(\frac{(x_0 - x)}{s}f\right) + D\frac{\partial^2 f}{\partial x^2} \tag{5.6\#eq.1}$$

Adjust x_0, D and s to obtain a steady state solution. Note that a steady state with random walkers can be fairly noisy.

5.5 Diffusion coefficient gradient

Simulate the equation

$$\frac{\partial f}{\partial t} = -u\frac{\partial f}{\partial x} + \frac{\partial}{\partial x}\left(D\left[\frac{1}{4} - \left(\frac{x-(x_{\mathsf{R}}+x_{\mathsf{L}})/2}{x_{\mathsf{R}}-x_{\mathsf{L}}}\right)^2\right]\frac{\partial f}{\partial x}\right) \qquad (5.6\#\text{eq.}1)$$

Why is the motion of the pulse retarded at the right boundary? Play with u and D and determine for which values the particles passes the boundary. *Hint:* Use the increments calculated in exercise 5.1.

5.6 Evolution of a crowd of people

Simulate people walking using the continuity equation

$$\frac{\partial P}{\partial t} + \frac{\partial}{\partial x}v(P)P = 0 \qquad (5.6\#\text{eq.}1)$$

where P denotes the density of people. The speed at which a person walks is strongly dependent on the density of people at that point: it is indeed very difficult to walk fast in a packed crowd! Invent and motivate your own function $v(P)$ and let your walkers go around in circle using periodic boundaries. Did you notice that people do not always know where they are going? It may therefore be appropriate to include a diffusion term, making the behaviour even more interesting. The complete equation to simulate will then look like this:

$$\frac{\partial P}{\partial t} + \frac{\partial}{\partial x}v(P)P - D\frac{\partial^2 P}{\partial x^2} = 0 \qquad (5.6\#\text{eq.}2)$$

Hint: Use the function `getValue(particlePosition[j])` as the argument for the density at the particle position.

5.7 Further readings

- **Numerical Solution of Stochastic Differential Equations**
  Kloeden [26].
- **Stochastic processes in physics and chemistry**
  Van Kampen [27].
- **Numerical Recipies**
  W. H. Press et al [4].
- **The Black-Scholes equation with an Applet**
  Carlsson[20] [28]
- **Particle Methods**
  Birdsall[21] [29] and Hockney [30]
- **Example of Monte Carlo integration**
  Ising model[22]

[20] http://fedu52.fed.ornl.gov/%7Ecarlsson/MonteCarlo
[21] http://ptsg.eecs.berkeley.edu/
[22] http://www2.truman.edu/%7Evelasco/ising.html

- **Monte Carlo Methods**
  Monte Carlo Methods[23] in WWW virtual library[24], Taygeta Scientific Incorporated[25] including C++ classes

5.8 Interactive evaluation form

Your anonymous opinion is precious. Please fill-in our anonymous form in the web edition. Thank you very much in advance for your collaboration!

[23] http://random.mat.sbg.ac.at/others/
[24] http://vlib.org/Overview.html
[25] http://www.taygeta.com/stochastics.html

6 LAGRANGIAN METHODS

6.1 Introduction

Rather than solving the convective derivative $\frac{d}{dt} = \frac{\partial}{\partial t} + u\frac{\partial}{\partial x}$ in *Eulerian coordinates*, for example by using a Taylor expansion with the Lax-Wendroff method (sect.2.2), the idea behind Lagrangian schemes is first to *split the evolution* into a sequence of alternating advection and non-advection phases

$$\frac{df}{dt} = G(f) \quad \Longrightarrow \quad \begin{cases} \dfrac{df}{dt} = 0 & \text{(advection)} \\ \dfrac{\partial f}{\partial t} = G(f) & \text{(all the rest)} \end{cases} \tag{6.1\#eq.1}$$

The advection is then evolved independently by *propagating the solution along the characteristics* (sect.1.2.4) in a suitable and if possible explicit manner with no restriction on the time step, keeping an Eulerian method such as explicit finite differences for the non-advection phase.

6.2 Cubic-Interpolated Propagation (CIP)

Introduced less than a decade ago by Yabe and Aoki [31], a whole family of schemes have been proposed along the same lines, relying on different interpolatations to propagate the solution along the characteristics.

Using a cubic-Hermite polynomial, the discretized function and its derivatives $\{x_j, f_j, f'_j\}$ can be approximated in a continuous manner with

$$\begin{aligned} F_j(x) &= \left[(a_j X - b_j)X + \Delta x f'_j\right] X + f_j \ ; \qquad a_j = \Delta x(f'_j + f'_{j+1}) - 2(f_{j+1} - f_j) \\ X &= \frac{(x - x_j)}{\Delta x} \ ; \ \Delta x = x_j - x_{j-1} \ ; \qquad b_j = \Delta x(f'_j + 2f'_{j+1}) - 3(f_{j+1} - f_j) \end{aligned} \tag{6.2\#eq.1}$$

Both satisfy the master evolution equation and its spatial derivative

$$\begin{cases} \dfrac{df}{dt} \equiv \dfrac{\partial f}{\partial t} + \dfrac{\partial}{\partial x}(uf) = g \\ \dfrac{df'}{dt} \equiv \dfrac{\partial f'}{\partial t} + \left(\dfrac{\partial u}{\partial x} f' + u\dfrac{\partial f'}{\partial x}\right) = \dfrac{\partial g}{\partial x} \end{cases} \tag{6.2\#eq.2}$$

They are split into an advection and non-advection phase

$$\begin{cases} \dfrac{df}{dt} = 0 \\ \dfrac{df'}{dt} = 0 \end{cases} \quad \text{and} \quad \begin{cases} \dfrac{\partial f}{\partial t} = g - \dfrac{\partial u}{\partial x} f \\ \dfrac{\partial f'}{\partial t} = \dfrac{\partial g}{\partial x} - \dfrac{\partial u}{\partial x} f' \end{cases} \tag{6.2\#eq.3}$$

For the advection phase, the solution is integrated analytically simply by shifting the cubic polynomials $F_j(x,t) = F_j(x - u\Delta t, t - \Delta t)$ along the characteristics

$$\begin{cases} f_{j+1}^{t+\Delta t} = F_{j+1}(x_{j+1} - u\Delta t, t - \Delta t) = f_{j+1}^t - \beta\left[\Delta x f'^t_{j+1} - \beta(b_{j+1} - \beta a_{j+1})\right] \\ f'^{t+\Delta t}_{j+1} = \frac{d}{dx}F_{j+1}(x_{j+1} - u\Delta t, t - \Delta t) = f'^t_{j+1} - \frac{\beta}{\Delta x}(2b_{j+1} - 3\beta a_{j+1}) \end{cases} \tag{6.2\#eq.4}$$

where $\beta = u\Delta t/\Delta x$ is the Courant-Friedrich-Lewy (CFL) number.

Although this is not at all mandatory (exercise 6.1), the scheme implemented in JBONE assumes for simplicity that $\beta \in [0;1]$ so that the quantities $(f_{j+1}^{t+\Delta t}, f'^{t+\Delta t}_{j+1})$ can be interpolated exclusively from the polynomial F_{j+1}, which is continuously defined in the interval $[x_j; x_{j+1}]$ — some bookkeeping is necessary to determine which interval to interpolate from when $|\beta| > 1$. After an initialization where the function is discretized with cubic-Hermite polynomials by sampling on a grid and the derivative calculated with centered finite differences, the CIP scheme is implemented in JBONE as

```
double alpha=timeStep*diffusCo/(dx[0]*dx[0]);  //These are only constant
double beta =timeStep*velocity/(dx[0]);        //  if the problem and the
int n=f.length-1;                              //  mesh are homogeneous

for (int i=0; i<n; i++) {
  a=dx[0]*(df[i]+  df[i+1])-2*(f[i+1]-f[i]);
  b=dx[0]*(df[i]+2*df[i+1])-3*(f[i+1]-f[i]);
  fp[i+1]=  f[i+1] -beta*(dx[0]*df[i+1]-beta*(b-beta*a));
 dfp[i+1]= df[i+1] -beta/dx[0]*(2*b-3*beta*a);
}
a=dx[0]*(df[n]+  df[0])-2*(f[0]-f[n]);
b=dx[0]*(df[n]+2*df[0])-3*(f[0]-f[n]);
fp[0]=  f[0] -beta*(dx[0]*df[0]-beta*(b-beta*a));
dfp[0]= df[0] -beta/dx[0]*(2*b-3*beta*a);
```

The on-line document illustrates the high quality of this approach when it is compared with the advection of a box function computed using other methods.
Numerical experiments:

- Change the initial condition to `Cosine` and reduce the spatial resolution down to 4 and 2 mesh points per wavelength in order to check how small both the numerical diffusion and dispersion are in comparison with other schemes!

6.3 Non-Linear equations with CIP

The same approach is applicable more generally for non-linear and vector equations

$$\frac{\partial \vec{f}}{\partial t} + \frac{\partial}{\partial x}(u\vec{f}) = \vec{g} \qquad (6.3\#\text{eq.1})$$

where $u = u(\vec{f})$ and $\vec{g} = \vec{g}(\vec{f})$. The problem is again decomposed in alternating phases without / with advection describing the evolution of the function

$$\begin{cases} \dfrac{\partial \vec{f}}{\partial t} = \vec{g} - \vec{f}\,\dfrac{\partial u}{\partial x} = \vec{G} & \text{(non-advection with compression term)} \\ \dfrac{\partial \vec{f}}{\partial t} + u\dfrac{\partial \vec{f}}{\partial x} = 0 & \text{(advection)} \end{cases} \qquad (6.3\#\text{eq.2})$$

and by differentiation of (eq.6.3#eq.1), the evolution of the derivative

$$\begin{cases} \dfrac{\partial \vec{f}'}{\partial t} = \vec{g}' - u\dfrac{\partial \vec{f}'}{\partial x} = \vec{G}' - \vec{f}'\dfrac{\partial u}{\partial x} & \text{(non-advection)} \\ \dfrac{\partial \vec{f}'}{\partial t} + u\dfrac{\partial \vec{f}'}{\partial x} = 0 & \text{(advection)} \end{cases} \qquad (6.3\#\text{eq.3})$$

Starting with the non-advection phase, the discretized function is first evolved according to

$$\vec{f_j^*} = \vec{f_j^t} + \vec{G_j}\Delta t \qquad (6.3\#\text{eq.}4)$$

where the super-script (*) refers to the intermediate step between the non- and advection phases. To avoid having to calculate $\vec{G'_j}$, the equation for the derivative is computed with

$$\begin{aligned} \frac{\vec{f'_j}^* - \vec{f'_j}^t}{\Delta t} &= \left[\frac{\vec{G_{j+1}} - \vec{G_{j-1}}}{2\Delta x} - \vec{f'_j}^t \frac{u_{j+1} - u_{j-1}}{2\Delta x} = \right] \\ &= \frac{\vec{f'_{j+1}}^* - \vec{f'_{j-1}}^* - \vec{f'_{j+1}}^t + \vec{f'_{j-1}}^t}{2\Delta x \Delta t} - \vec{f'_j}^t \frac{u_{j+1} - u_{j-1}}{2\Delta x} \qquad (6.3\#\text{eq.}5) \end{aligned}$$

The advection phase is then evolved in the same manner as before (eq.6.2#eq.4), by shifting the cubic-Hermite polynomials along the characteristics (exercise 6.4).

6.4 Exercises

6.1 Arbitrary CFL number

Modify the CIP scheme in the JBONE applet to allow for arbitrarily large time-steps and negative advection velocities.

6.2 Diffusion in CIP

Use your favourite method to implement the diffusion phase with CIP in the JBONE applet. Discuss the merits of this combined solution.

6.3 Lagrangian method with splines

Analyze and discuss the Lagrangian scheme proposed by Guillaume Jost (EPFL, Lausanne) using a cubic-spline interpolation of the function.

6.4 Non-linear equation

Use the formalism in sect.6.3 to solve the general non-linear problem (4.4#eq.1). Study each of the wave-breaking, diffusion and dispersion terms separatly and compare with the solutions obtained previously with another method.

6.5 Further Reading

- **Cubic-Interpolated Propagation (CIP).**
  Yabe and Aoki [31], [32], [33]

6.6 Interactive evaluation form

Your anonymous opinion is precious. Please fill-in our anonymous form in the web edition. Thank you very much in advance for your collaboration!

7 WAVELETS

7.1 Remain a matter of research

There is a growing interest in using wavelets not only for the discretization of functions (sect.1.4), but also to approximate differential and integral operators. Motivations for that are the potential gain of solving global problems with the same $\mathcal{O}(N)$ operations as there are unknowns, relying on recent advances in iterative methods (sect.3.4) to solve linear systems in sparse format. Having not had the possibility so far to implement wavelets into the JBONE applet and extract the essence of research papers in a pedagogical manner, this section is limited to a number of links to papers maintained on a web site from MathSoft[26]:

1. D. M. Bond and S. A. Vavasis, Fast Wavelet Transforms for Matrices Arising From Boundary Element Methods.[27]

2. T. Chan, W. Tang and W. Wan, Wavelet sparse approximate inverse preconditioners[28]

3. P. Charton and V. Perrier, Factorisation sur Bases d'Ondelettes du Noyeau de la Chaleur et Algorithmes Matriciels Rapides Associes[29]

4. P. Charton and V. Perrier, Towards a Wavelet Based Numerical Scheme for the Two-Dimensional Navier-Stokes Equations.[30]

5. P. Charton and V. Perrier, A Pseudo-Wavelet Scheme for the Two-Dimensional Navier-Stokes Equations.[31]

6. S. Dahlke and A. Kunoth, Biorthogonal Wavelets and Multigrid.[32]

7. S. Dahlke and I. Weinreich, Wavelet-Galerkin Methods: An Adapted Biorthogonal Wavelet Basis.[33]

8. S. Dahlke and I. Weinreich, Wavelet Bases Adapted to Pseudo-Differential Operators.[34]

9. W. Dahmen and A. Kunoth, Multilevel Preconditioning.[35]

10. R. Glowinski, T. Pan , R. O. Wells, Jr. and X. Zhou, Wavelet and Finite Element Solutions for the Neumann Problem Using Fictitious Domains[36]

11. R. Glowinski, A. Rieder, R. O. Wells, Jr. and X. Zhou, A Wavelet Multigrid Preconditioner for Dirichlet Boundary Value Problems in General Domains.[37]

[26] http://www.mathsoft.com/wavelets.html
[27] ftp://ftp.tc.cornell.edu/pub/tech.reports/tr174.ps
[28] ftp://ftp.math.ucla.edu/pub/camreport/cam96-33.ps.gz
[29] ftp://ftp.lmd.ens.fr/MFGA/pub/wavelets/produits2d.ps.Z
[30] ftp://ftp.lmd.ens.fr/MFGA/pub/wavelets/iciam95.ps.Z
[31] ftp://ftp.lmd.ens.fr/MFGA/pub/wavelets/ns.ps.Z
[32] ftp://ftp.igpm.rwth-aachen.de/pub/dahlke/dksh.ps.Z
[33] ftp://ftp.igpm.rwth-aachen.de/pub/ilona/wega.ps.Z
[34] ftp://ftp.igpm.rwth-aachen.de/pub/ilona/wega2.ps.Z
[35] ftp://ftp.igpm.rwth-aachen.de/pub/dahmen/mulpre.ps.gz
[36] ftp://cml.rice.edu/pub/reports/9201.ps.Z
[37] ftp://cml.rice.edu/pub/reports/9306.ps.Z

12. R. Glowinski, A. Rieder, R. O. Wells, Jr. and X. Zhou, A Preconditioned CG-Method for Wavelet-Galerkin Discretizations of Elliptic Problems[38]

13. F. Heurtaux, F. Planchon and M. V. Wickerhauser, Scale Decomposition in Burgers' Equation[39]

14. A. Jiang, Fast wavelet based methods for certain time dependent problems[40]

15. A. Kunoth, Multilevel Preconditioning – Appending Boundary Conditions by Lagrange Multipliers.[41]

16. J. Lewalle, Wavelet Transforms of some Equations of Fluid Mechanics[42]

17. J. Lewalle, Energy Dissipation in the Wavelet-Transformed Navier-Stokes Equations[43]

18. J. Lewalle, On the effect of boundary conditions on the multifractal statistics of incompressible turbulence[44]

19. J. Lewalle, Diffusion is Hamiltonian[45]

20. D. Lu, T. Ohyoshi and L. Zhu, Treatment of Boundary Conditions in the Application of Wavelet-Galerkin Method to a SH Wave Problem[46]

21. A. Rieder and X. Zhou, On the Robustness of the Damped V-Cycle of the Wavelet Frequency Decompositions Multigrid Method[47]

22. A. Rieder, R. O. Wells, Jr. and X. Zhou, A Wavelet Approach to Robust Multilevel Solvers for Anisotropic Elliptic Problems.[48]

23. A. Rieder, R. O. Wells, Jr. and X. Zhou, On the Wavelet Frequency Decomposition Method[49]

24. O. V. Vasilyev and S. Paolucci, A Dynamically Adaptive Multilevel Wavelet Collocation Method for Solving Partial Differential Equations in a Finite Domain.[50]

25. O. V. Vasilyev, S. Paolucci and M. Sen, A Multilevel Wavelet Collocation Method for Solving Partial Differential Equations in a Finite Domain.[51]

26. R. O. Wells, Jr. and X. Zhou, Wavelet Solutions for the Dirichlet Problem[52]

27. R. O. Wells, Jr. and X. Zhou, Wavelet Interpolation and Approximate Solution of Elliptic Partial Differential Equations[53]

[38] ftp://cml.rice.edu/pub/reports/9414.ps.Z
[39] http://wuarchive.wustl.edu/doc/techreports/wustl.edu/math/papers/burgers.ps.Z
[40] ftp://ftp.math.ucla.edu/pub/camreport/cam96-20.ps.gz
[41] ftp://ftp.igpm.rwth-aachen.de/pub/kunoth/cosh.ps.Z
[42] http://www.mame.syr.edu/faculty/lewalle/acta-94.html
[43] http://www.mame.syr.edu/faculty/lewalle/dissip-93.html
[44] http://www.mame.syr.edu/faculty/lewalle/camb-93.html
[45] http://www.mame.syr.edu/faculty/lewalle/hamdiff.html
[46] ftp://ftp.mathsoft.com/pub/wavelets/bc.ps.gz
[47] ftp://cml.rice.edu/pub/reports/9310.ps.Z
[48] ftp://cml.rice.edu/pub/reports/9307.ps.Z
[49] ftp://cml.rice.edu/pub/reports/9413.ps.Z
[50] http://landau.mae.missouri.edu/%7evasilyev/Publications/adaptive.ps.gz
[51] http://landau.mae.missouri.edu/%7evasilyev/Publications/WML.ps.gz
[52] ftp://cml.rice.edu/pub/reports/9202.ps.Z
[53] ftp://cml.rice.edu/pub/reports/9203.ps.Z

28. R. O. Wells, Jr. and X. Zhou, Representing the Geometry of Domains by Wavelets with Applications to Partial Differential Equations[54]

29. R. O. Wells, Jr., Multiscale Applications of Wavelets to Solutions f Partial Differential Equations[55]

[54] ftp://cml.rice.edu/pub/reports/9214.ps.Z
[55] ftp://cml.rice.edu/pub/reports/9409.ps.Z

8 THE JBONE USER MANUAL

The *Java Bed for ONE* dimensional evolution equations JBONE provides a flexible environement to test and compare a variety of numerical schemes using a JAVA applet. This section gives a short introduction serves as a user manual for the program. Note that **you don't need to install** the code if you open a distance-learning account on the course web server[56].

8.1 Source code & installation

In the downloadable distribution, the JBONE package consists of JAVA sources files `*.java`, installation instructions in the `README` file and a number of UNIX specific configuration `config*` and compiler rules `Makefile*`.

Download. The JBONE source code can be obtained free of charge for personal use from the course main page[57]. After registration, the server will send a password that gives you access to an automatic download service.

Install under UNIX. The installation is in principle very simple using the commands `autoconf` / `automake` that generate a `Makefile` tailored specifically for your system:

```
cd ~                              # Preferably your home directory
gunzip pde-course-*.tar.gz        # Uncompress
tar xvf pde-course-*.tar          # Unbundle
cd  jbone
configure                         # Create Makefile for your system
```

A `Makefile.default` is provided if `configure` fails on an old UNIX platform:

```
cp Makefile.default Makefile
```

Compile under UNIX. Different targets are built using the commands

```
make all                          # Everything below
make jbone                        # Compile java files
make docs                         # Generate documentation
make run.html                     # Wrapper file for appletviewer
```

It is possible to compile the JAVA sources by hand using commands of the type

```
javac jbone.java                   # Compile java files
javadoc -private -version *.java # Generate documentation
cp *.html ~/public_html/jbone     # Publish on the web
```

These are however system dependent.

Install & compile under Windows. Please refer to vendor supplied information to find out how to compile and run the JBONE code starting from the source files `*.java`.

[56] http://pde.fusion.kth.se
[57] http://pde.fusion.kth.se

Program listing. The substance of the program listing[58] (excluding the graphics user interface) can be consulted directly in your web browser. If you read this document on-line, the previous link illustrates how markers of the form `//TAG_rights_TAG//` have been inserted in the source code to target specific sections, enabling the web browser to jump directly to a section of interest. The column on the left tells you from which file the program instruction comes from.

8.2 Program execution

The JBONE code can be executed either as an independent JAVA program or as an applet, the latter adding the possibility of running it in your web browser.

There are 3 manners to execute. On a UNIX platform, type

```
java jbone                      # Run program interpreter
appletviewer run.html           # Run applet with default
                                #   arguments in run.html
```

The third manner, of course, is to open the file `run.html` directly with your web browser using the address `file://absolute_path/run.html`. You need to recompile with the command `make all` after every modification. Because the web browser tend to use buffered data, you must **PRESS** `SHIFT` **AND SELECT** `View->RELOAD` to force it to load the newly compiled code.

Preset with HTML tags. Running applets has the advantage of choosing the run time arguments in the calling HTML file. Take for example run.html:

```
<html>
  <head>
    <! -- @(#)run.html
    Andre JAUN (jaun@fusion.kth.se) and Johan HEDIN (johanh@fusion.kth.se)
    (C) 1999-2001, all rights reserved. This shareware can be obtained
    without fee by e-mail from the authors. Permission to use, copy, and
    modify the source and its documentation for your own personal use is
    granted provided that this copyright notice appears in all copies.
    -->
    <title>JBONE</title>
  </head>
  <body>
    <h1>JBONE scratch-pad</h1>
    Select the switches and modify the parameters to study and compare
    different numerical schemes<br><p>
    <applet codebase="../jbone/" code=jbone.class
            align=center width=740 height=340>
       <param name=pde                value="Advection">
       <param name=method             value="Lagrangian">
       <param name=ic                 value="Gaussian">
       <param name=Velocity           value=1.>
    </applet><p>
  </body>
</html>
```

[58] http://www.fusion.kth.se/courses/jbone/listing.html#rights

The default parameters are here modified to calculate the advection of a Gaussian pulse using the CIP / Lagrangian scheme from sect.6.

8.3 Program structure & documentation

Using an object oriented language such as JAVA, it is important to realize that you don't have to read the entire listing to understand and even modify the code. This section gives you some hints as where to find information and what needs to be done to add a new scheme.

Documentation. It is generated and automatically updated with the command `make docs`, using the comments `/** */` that preceed the declarations in the JBONE source code. Check how this happens in the method `Mesh.point()` from the `Mesh` object in the file `Mesh.java`

```
/** Coordinate value
    @param i The index of a coordinate
    @return The value of a coordinate */
public double   point(int i) { return x[i]; }
```

Follow the links to verify where `Mesh.point()` appears in the program tree and the name index, defining everything you need to know to obtain a mesh point coordinate: give it an integer index and it will return the corresponding real position. Unless you want to modify the properties of the Mesh object, you never need to know where and how the position was stored!

All you need to modify. To complete most of the projects, it is sufficient to modify or add some small sections in the sub-classes of the `Solution` hierarchy, i.e. the files

- `FDSolution.java` — for finite difference schemes
- `FEMSolution.java` — for finite elements schemes
- `FFTSolution.java` — for Fourier methods
- `MCMSolution.java` — for Monte-Carlo schemes
- `CHASolution.java` — for Lagarangian schemes

Having identified the section with a specific choice of the selectors your task consists in defining new values `fp[]` from the old `f[]`. If you add a new combination of selectors, you need to define it in `Solution.hasOption()` to finally make it appear at run time.

After each modification, remember that you need to recompile with the command `make all`. And you must press `SHIFT` and select `View->RELOAD` to force the browser to load the new compiled code.

8.4 An object-oriented example: Monte-Carlo in JBONE

The Monte Carlo solver in JBONE is different from the finite differences and finite elements solvers in the sense that the solution is represented by the set of particles and the function f is just used for diagnostics. As shown in the class tree, the solvers have been divided into particle methods and fluid methods. The discretization with particles is contained in the class `ParticleSolution` and the Monte Carlo time stepping and boundary conditions in the class `MCMSolution`. The class `ParticleSolution` contains a vector of the particles phase

space coordinates. Since `f[]` is only defined as a projection onto the roof-top base, the advantage of an objected oriented methods becomes clear, as e.g. the method `limits()` can here be overridden from the definition in `Solution`. Indeed, check how `limits()` in `FluidSolution` is computed directly from the solution `f[]`, whereas in `ParticleSolution` the method `generateDistribution()` needs to be called prior to finding min and max of `f[]`. Everything the class `jbone` needs to know is that the solver is a sub-class of `Solution`. How and where things are computed, is none of `jbone`'s business! Also note how the solutions to exercises 5.3–5.5 will be inherited in the class `MCMDrawParticlesSolution`

9 LEARNING LABORATORY ENVIRONEMENT

This chapter gives a short introduction, advice and links to the further documentation for the tools that are used to run this course in a virtual university environement. Most of the tables can be consulted directly when needed, by following the link above the input windows and using the browser Back button to recover input. They are here only given for reference.

9.1 Typesetting with TEX

The text input in the first window is typeset using the TEXlanguage and is translated into HTML with the tth compiler installed on our server. You have to view documents using the Western character set ISO-8859, which is generally set by default in recent browsers. If this page doesn't display the symbols correctly, please refer to the frequently asked questions FAQ link on the course main page.

TEXbasics.

Normal ASCII input is interpreted in *text mode* and TEXcommands starting with the backslash character \ are used for formatting. Mathematical symbols are typed in *math mode* delimited by two dollar signs ($\partial_t f$ yields $\partial_t f$) or in an equation:

```
\begin{equation}\label{advection}
\frac{d}{dt}f \equiv
\frac{\partial f}{\partial t}
  +u\frac{\partial f}{\partial x}=0
\end{equation}
```

yields $$\frac{d}{dt}f \equiv \frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} = 0 \quad (9.1\#\mathrm{eq}.1)$$

where (`\ref{advection}`) yields (9.1#eq.1) and can be used for reference within the document. You can also add links and HTML inserts using

```
\href{http://address}{text} create a link from text to http://address
\special{html:stuff} inserts HTML stuff
```

Character type and size.

Rom \textrm{}	*Ital* \textit{}	**Bold** \textbf{}	Type \texttt{}
Rom \mathrm{}	*Ital* \mathit{}	**Bold** \mathbf{}	Type \mathtt{}
small \small{}	normal \normalsize{}	large \large{}	Large \Large{}

Special characters and accents (text mode).

$ \$	& \&	% \%	# \#	{ \{	} \}	_ _
é \'{e}	è \`{e}	ê \^{e}	ë \"{e}	ç \c{c}	
† \dag	‡ \ddag	§ \S	¶ \P	© \copyright	£ \pounds

Greek letters (math mode).

α \alpha	β \beta	γ \gamma	δ \delta	ϵ \epsilon	ε \varepsilon
ζ \zeta	η \eta	θ \theta	ϑ \vartheta	ι \iota	κ \kappa
λ \lambda	μ \mu	ν \nu	ξ \xi	o o	π \pi
ϖ \varpi	ρ \rho	ϱ \varrho	σ \sigma	ς \varsigma	τ \tau
υ \upsilon	ϕ \phi	φ \varphi	χ \chi	ψ \psi	ω \omega
Γ \Gamma	Δ \Delta	Θ \Theta	Λ \Lambda	Ξ \Xi	Π \Pi
Σ \Sigma	Υ \Upsilon	Φ \Phi	Ψ \Psi	Ω \Omega	

Binary operation and relation symbols (math mode).

$\pm$ \pm	$\mp$ \mp	$\times$ \times	$\div$ \div	$*$ \ast	$\circ$ \circ
$\bullet$ \bullet	$\cdot$ \cdot	$\cap$ \cap	$\cup$ \cup	$\dagger$ \dagger	$\ddagger$ \ddagger
$\leq$ \leq	$\geq$ \geq	$\ll$ \ll	$\gg$ \gg	$\subset$ \subset	$\supset$ \supset
$\subseteq$ \subseteq	$\supseteq$ \supseteq	$\in$ \in	$\ni$ \ni	$\equiv$ \equiv	$\approx$ \approx
$\sim$ \sim	$\simeq$ \simeq	$\neq$ \neq	$\propto$ \propto	$\perp$ \perp	$\mid$ \mid
$\parallel$ \parallel					

Arrows and miscellaneous symbols (math mode).

$\leftarrow$ \leftarrow	$\rightarrow$ \rightarrow	$\Leftarrow$ \Leftarrow	$\Rightarrow$ \Rightarrow
$\leftrightarrow$ \leftrightarrow	$\Leftrightarrow$ \Leftrightarrow	$\uparrow$ \uparrow	$\downarrow$ \downarrow
$\Uparrow$ \Uparrow	$\Downarrow$ \Downarrow	$\mapsto$ \mapsto	$\aleph$ \aleph
$\hbar$ \hbar	$\imath$ \imath	ℓ \ell	$\wp$ \wp
$\Re$ \Re	$\Im$ \Im	$\prime$ \prime	$\emptyset$ \emptyset
∇ \nabla	$\surd$ \surd	$\|$ \—	$\angle$ \angle
$\forall$ \forall	$\exists$ \exists	\ \backslash	∂ \partial
∞ \infty	$\clubsuit$ \clubsuit	$\diamondsuit$ \diamondsuit	$\heartsuit$ \heartsuit
$\spadesuit$ \spadesuit			

Operations and functions (math mode).

$\sum$ \sum	$\prod$ \prod	$\int$ \int	$\oint$ \oint	$\sqrt{a}$ \sqrt{a}
a^b a^{b}	a_{ij} a_{ij}	$\sinh$ \sinh	$\arccos$ \arccos	$\cos$ \cos
$\arcsin$ \arcsin	$\sin$ \sin	$\arctan$ \arctan	$\tan$ \tan	$\arg$ \arg
$\cot$ \cot	$\cosh$ \cosh	$\det$ \det	$\dim$ \dim	$\exp$ \exp
$\lim$ \lim	$\ln$ \ln	$\log$ \log	$\max$ \max	$\min$ \min
$\tanh$ \tanh	$\frac{a}{b}$ \frac{a}{b}			

Format, list and equations.

\begin{quote} \end{quote}
\begin{quotation} \end{quotation}
\begin{center} \end{center}
\begin{verse} \end{verse}
\begin{verbatim} \end{verbatim}
\begin{itemize} \item \end{itemize}
\begin{enumerate} \item \end{enumerate}
\begin{description} \item \end{description}
\begin{equation} \label{key} \end{equation}
\begin{equation*} \end{equation*}

Tables (text mode) and arrays (math mode).

```
\begin{tabular}{|llc|}
\multicolumn{2}{c}{ITEM} &
  \multicolumn{1}{c}{PRICE} \\
gnat & (dozen) & 3.24 \\
gnu  & (each)  & 24.00
\end{tabular} \\
```

yields

ITEM		PRICE
gnat	(dozen)	3.24
gnu	(each)	24.00

```
\begin{equation*}
\begin{array}{clcr}
a+b+c & uv    & x-y & 27 \\
a+b   & u+v   & z   & 134 \\
a     & 3u+vw & xyz & 2,978
\end{array}
\end{equation*}
```

yields

$$\begin{array}{clcr} a+b+c & uv & x-y & 27 \\ a+b & u+v & z & 134 \\ a & 3u+vw & xyz & 2,978 \end{array}$$

```
\begin{eqnarray}
\lefteqn{a+b+c=} \nonumber \\
& & c+d+e+f+g+h  \nonumber
x & < & y
\end{eqnarray}
```

yields

$$\begin{aligned} a+b+c= & \\ & c+d+e+f+g+h \\ x \;&<\; y \end{aligned} \qquad (9.1\#\text{eq.2})$$

9.2 Progamming in JAVA

Numerical schemes submitted from the Java window are automatically inserted in the JBONE source code (e.g. exercise 2.1) and have first to be compiled on our server before you can download and execute them locally in your browser. This page introduces a limited number of Java commands you have to know when you carry our the assignments. More details concerning the JBONE applet can be found in the program tree, the name index and finally in the program listing. For tutorial in Java programming, consult the tutorial[59] from Sun Microsystems.

JBONE variables.

From the list of run parameters (an object called `runData`), the JBONE applet defines the (double = 16 digits precision real, int = up to 9 digits signed integer) local variables

advection velocity u	`double velocity = runData.getVelocity();`
diffusion coefficient D	`double diffusCo = runData.getDiffusion();`
dispersion coefficient υ	`double disperCo = runData.getDispersion();`
time step Δt	`double timeStep = runData.getTimeStep();`
number of mesh points n	`int n = runData.getNumberOfMeshPoints();`
number of particles NP	`int np = runData.getNumberOfParticles();`
normalized advection β	`double beta;`
normalized diffusion α	`double alpha;`

and computes the evolution of the variables and arrays defining the solution (an object called `solution`)

time t	`double time;`
mesh x_i, intervals Δx_i	`double[] x,dx;`
$f(x), g(x) \in \mathcal{R}$	`double[] f,g;`
$h(x), s(x) \in \mathcal{C}$	`Complex[] h,s;`
$f(t-\Delta t)$, $f(t)$, $f(t+\Delta t)$	`double[] fm,f,fp, gm,g,gp;`
$h(t-\Delta t)$, $h(t)$, $h(t+\Delta t)$	`Complex[] hm,h,hp, sm,s,sp;`
$\partial_x f$, $\partial_x g$, $\partial_x h$	`double[] dfm,df,dfp, dgm,dg,dgp;`
	`Complex[] dhm,dh,dhp;`
particle positions x_{pi}	`double[] particlePosition;`

where the index of every array starts with zero (`x[0]`) and finishes with one element less than its size (`x[x.length-1]`).

JBONE = Java Bed for ONE dimensional evolution.

In a simple evolution, which can schematically be written as

1. For $t = 0$ use the initial condition to define $f(t)$
2. Plot $f(t), g(t)$
3. Define new value $f(t + \Delta t)$ in terms of current $f(t)$ and past values $f(t - \Delta t)$

[59] http://java.sun.com/docs/books/tutorial/index.html

4. Shift time levels $t \to t + \Delta t$ and copy the arrays $f(t + \Delta t) \to f(t) \to f(t - \Delta t)$
5. Goto 2 until finished

only the third step (in red) has in fact to be defined in the Java window. For example, the simple loop

```
for (int i=0; i<=n; i++) {
  fp[i]=0.98*f[i];
}
```

computes an artificial evolution where the initial condition decays by 2% every time step.

Debugging.

Having corrected all the compiler errors unfortunately doesn't mean that your scheme immediately behaves the way you want! You may then have to monitor the value of different variables, inserting statements such as

```
System.out.println("Value fp["+i+"] = "+fp[i]);
```

after the second line in the example above (9.2) to print the values of the array `fp` to the *Java Console*. Submit a scheme with such a print statement to the compiler, open the *Java Console* of your browser (with Netscape select *Communicator* → *Tools* → *Java Console*, with Explorer first select *Tool* → *Internet Options* → *Advanced* → *Java console enabled* and then *View* → *Java Console*) and advance the calculation one step in the applet. From the values that appear in the *Console*, it is generally possible to track down all the mistakes.

Common errors.

To avoid first difficulties when you carry out your assignments, note that

- every new variable (not listed in the variable index) has to be declared; memory for arrays and objects must be allocated explicitly with the command `new`

```
int i = 3;                 // Declare i as an integer
double[] c;                // Declare c[] array 16 digits nbrs
c = new double[i];         // Memory for c[0], c[1], c[2]
BandMatrix  A;             // Declare A as a BandMatrix object
A = new BandMatrix(3,10); // Memory for 3 bands with 10 doubles
```

- if you forget to attribute memory in the example above and a statement suddendly refers to the element `c[0]` results in the infamous `java.lang.NullPointerException` error; accessing `c[3]` yields the `java.lang.ArrayIndexOutOfBoundsException:3` error, because the first element of an array in Java always starts with the index number 0.
- the assigning equal sign is denoted by as single = and the comparing equal sign by a double ==

```
int a = 42;
if(a == 17)
  System.out.println("a is equal to 17");
if(a != 17)
  System.out.println("a is not equal to 17");
```

will print the text "`a is not equal to 17`" to the *Java Console*.

9.3 Parameters and switches in HTML

Editable parameters.

The following list of input parameters can be used to change the defaults pre-defined in the JBONE code:

- **Velocity** the advection velocity u (or the market interest rate r)
- **Diffusion** the diffusion D (or the market volatility σ)
- **Dispersion** the dispersion
- **TimeStep** the time step Δt
- **MeshPoints** the number of mesh points N
- **Particles** the number of particles N_p
- **TimeImplicit** the time implicit parameter θ
- **TuneIntegr** the tunable integration parameter p
- **ICAmplitude** the initial condition amplitude
- **ICPosition** the initial condition position
- **ICWidth** the initial condition width
- **ICWavelength** the initial condition wavelength
- **RunTime** the physical run time, T=$n\Delta t$
- **MeshLeft** the left position of the mesh x_0
- **MeshLength** the length of the simulation box x_N
- **PhysDataCase** the type of Physical Data, e.g potential shape
- **PhysDataValue** a Physical Data parameter, e.g potential amplitude

Selectors.

The selectors appear over the JBONE plot window and allow you to choose the problem you want to solve. Careful, white spaces count!

- **method** selects the numerical method. Choices include "Finite differences", "Finite elements", "Fourier transform", "Monte-Carlo", "Lagrangian".
- **scheme** selects a particular scheme in a given method. Choices include "Standard scheme", "Explicit 2-level", "Explicit 3-level", "Implicit 2-level", "Expl-LaxWendroff", "Impl-LaxWendroff", "Leap-frog (FDTD)", "European naive", "European vanilla", "American vanilla", "Tunable Integration", "Expanded convol.", "Aliased convol.", "CubicHermite FEM", "Cubic – Splines", "Forward Euler", "Forward Euler pp", "My scheme", "Exercise 2.1", "Exercise 2.3", etc
- **ic** selects the type of initial condition. Choices include "Box", "Gaussian", "Cosine", "Soliton", "WavePacket", "PutOption".
- **pde** selects the type of PDE. Choices include "Advection", "Burger (shock)", "KdV (solitons)", "Schroedinger", "Black-Scholes", "My equation 1", "My equation 2", "Exercise".

TAG defaults.

The JBONE applet is included an HTML document with a header of the form

```
<applet codebase="./jbone/" code=jbone.class
        align=center width=720 height=340>
       <param name=Velocity            value=0.>
       <param name=Diffusion           value=0.>
       <param name=Dispersion          value=0.>
       <param name=TimeStep            value=0.5>
       <param name=MeshPoints          value=64>
       <param name=Particles           value=0>
       <param name=TimeImplicit        value=0.7>
       <param name=TuneIntegr          value=0.333>
       <param name=ICAmplitude         value=1.>
       <param name=ICPosition          value=18.>
       <param name=ICWidth             value=8.>
       <param name=ICWavelength        value=4.>
       <param name=RunTime             value=128.>
       <param name=MeshLeft            value=0.>
       <param name=MeshLength          value=64.>
       <param name=PhysDataCase        value=1>
       <param name=PhysDataValue       value=1.>
       <param name=method              value="Finite differences">
       <param name=scheme              value="Exercise 2.1">
       <param name=ic                  value="Gaussian">
       <param name=pde                 value="Exercise">
</applet>
```

where the first two lines specify the path name of the executable, the position and the size of the window. The *TAGS* that follow define the default values of parameters and switches that are set when the applet appears in the web page.

10 COURSE EVALUATION AND PROJECTS

10.1 Interactive evaluation form

Your anonymous opinion is precious. Please fill-in the anonymous form in the web edition. Thank you very much in advance!

10.2 Suggestions for one-week projects

The best ideas for a small one-week project stems directly from your own field ! For those however who want some suggestions, here is a list of projects more or less in rising order of difficulty.

Diffusion in an inhomogeneous medium. Let the advection $u(x)$ and diffusion coefficients $D(x)$ vary in space, and solve the inhomogeneous advection-diffusion equation (1.3.2#eq.2) with finite elements. Using a Gaussian quadrature instead of the analytical calulation of the inner products (3.3#eq.4).

Inhomogeneous mesh with FEMs. Add the capability of refining the mesh using the finite elements method. Integrate analytically a sum of Lorentzian functions $L_{[x_j,w_j]}(x) = w_j \left[w_j^2 + (x-x_j)^2\right]^{-2}$ defining the locations x_j and the weights w_j of the regions you want to refine and derive the expression for an inhomogeneous distribution of mesh points

$$X_{packed}(x) = px + (1-p)\frac{\sum_{j=1}^{N} \arctan\left(\frac{x-x_j}{w_j}\right) + \arctan\left(\frac{x_j}{w_j}\right)}{\sum_{j=1}^{N} \arctan\left(\frac{1-x_j}{w_j}\right) + \arctan\left(\frac{x_j}{w_j}\right)} \qquad (10.2\#\text{eq.1})$$

Study the numerical convergence of the advection-diffusion problem in a strongly inhomogeneous medium.

Iterative GMRES solver for wave-equations. Add a modular GMRES solver in JBONE. Start by defining a matrix-vector product in sparse format and implement the iterative scheme by testing it first for a diffusion dominated problem; switch to a wave equation once everything appears to work. Compare the efficiency of the new iterative method with a direct solution for a diffusion dominated problem using an increasing amount of advection.

The uncertain future of a predator-prey system. During uncertain (stochastic) feeding conditions, the Volterra-Lotka system of equations (exercise 1.2) could be replaced by a stochastic differential equation describing an ensemble of possible outcomes. Study the evolution of the density distribution of the possible population using a Monte Carlo method; check the possibility of an extinction simply by "bad luck". Show that your solution converges for small time steps.

Wave equation as a driven problem. Use two different methods to solve the equation describing forced oscillation in a weakly absorbing bounded medium

$$\frac{\partial^2 f}{\partial t^2} - c^2 \frac{\partial^2 f}{\partial x^2} = S_{\omega_0}(x,t) - 2\nu \frac{\partial f}{\partial t} \qquad (10.2\#\text{eq.2})$$

Choose a source $S_{\omega_0}(x,t) = S_0 \exp(x^2/\Delta^2)\sin(\omega_0 t)$ smoothly distributed inside the cavity $x \in [-L/2; L/2]$ and a sink $\nu \ll \omega_0 \simeq 2\pi c/L$ which is sufficiently small to allow the perturbations to propagate and reflect. Vary the driving frequency ω_0 and study the possibility of exciting resonances inside the cavity.

Particle weight. Apply individual weights w_i to the particles in the Monte Carlo solver. Try to reduce the amount of noise at low levels in a steady state solution (e.g. exercise 5.4) by splitting particles with $w_i > w_{\text{limit}}(x)$. Try to find a good maximum weight function $w_{\text{limit}}(x)$. After a while some particles will be to heavy. Solve this by re-discretizing the distribution function with a call to `discretize(new ShapeNumerical(this));` The particle weight should be used to increase the accuracy in discretization. This project involves changing the projection, the discretization and learning the `Vector` class in Java.

Option pricing. Study the finite difference schemes proposed for the price of a European put option in financial and lognormal variables (exercise 2.5) and implement the implicit finite element scheme for an American put (exercise 3.5) in financial variables. Complete the project by implementing a Monte-Carlo solver along the same lines.

Bose-Einstein condensation. Start with the linear Schrödinger equation

$$i\frac{\partial \psi}{\partial t} = \left[-\frac{\partial^2}{\partial x^2} + V(x)\right]\psi \qquad (10.2\#\text{eq.}3)$$

normalized with Plank's constant $\hbar = 1$ and a particle mass $m = 1/2$ and use two different schemes to calculate the scattering of a wavepacket by a simple one-dimensional potential $V(x)$. Having validated your schemes with analytical results, solve the non-linear equation describing the Bose-Einstein condensation in a parabolic trapping potential $V(r) = \frac{\alpha}{2}r^2$ with cylindrical symmetry

$$i\frac{\partial \psi}{\partial t} = \left[-\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial}{\partial r}\right) + V(r) + 2\pi a \mid \psi \mid^2\right]\psi \qquad (10.2\#\text{eq.}4)$$

where the parameter a defines the scattering length.

Slowing down of beams. The slowing down of a beam in a collisional media is given by

$$\frac{\partial f(x,v,t)}{\partial t} + v\frac{\partial f(x,v,t)}{\partial x} + \frac{\partial}{\partial v}[-v_p v f(x,v,t)] = \frac{\partial}{\partial v}\left(D_p \frac{\partial f(x,v,t)}{\partial v}\right) \qquad (10.2\#\text{eq.}5)$$

Use Monte Carlo for solving the beam distribution function. Identify the drift and diffusion coefficients in x and v. Start the pulse to the left with $v = v_0$. Modify v_p and D_p in order to get a nice slowing down of the pulse.

KdV with wavelets.

References

[1] J. P. Bérenger. A perfectly matched layer for the absorption of electomagnetic waves. *Journal of Computational Physics*, 114:185, 1994.

[2] M. Abramowitz and A. Stegun. *Handbook of Mathematical Functions*. Dover Publications, New York, tenth edition, 1972.

[3] W. Sweldens. The Wavelet Home Page[60].

[4] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipies*[61]. Cambridge University Press, second edition, 1992.

[5] J. M. Sanz-Serna and P. Calvao. *Numerical Hamiltonian Problems*. Chapman & Hall, 1994.

[6] G. Dahlquist and Å Bjorck. *Numerical Methods*. Prentice-Hall, 1974.

[7] C. A. Fletcher. *Computational Techniques for Fluid Dynamics*. Springer, second edition, 1991.

[8] C. Johnson. *Numerical Solution of PDEs by the FEM Method*. Studentlitteratur – Lund, Sweden, 1987.

[9] K. Appert. Experimentation numérique. unpublished lecture notes, CRPP-EPFL, CH-1015 Lausanne, Switzerland, 1988.

[10] National Institute of Standards and Technology (NIST). Guide to Available Mathematical Software[62].

[11] J. Eastwood. Computer Physics Communications Library[63].

[12] I. M.Ryzhik I. S. Gradshteyn. *Table of integrals, series and products*. Academic Press, New York, 1980.

[13] N.W. Ashcroft and N.D. Mermin. *Solid State Physics*. Holt-Saunders International Editions, 1976.

[14] J. Boris. *Methods in Computational Physics, Vol.16*. Academic, 1976.

[15] J. M. Jin. *The Finite Element Method in Electromagnetics*. John Wiley, 1993.

[16] D. Katz and T. Cwik. EMLIB homepage[64].

[17] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, 1996.

[18] P. Wilmott, J. Dewynne, and S. Howison. *Option Pricing*. Oxford Financial Press, 1993.

[19] T. Bjork. Examples in optimization theory. unpublished lecture notes, Inst. for Optimization and System Theory, KTH, SE-100 44 Stockholm, Sweden, 1995.

[60] http://www.wavelet.org/wavelet/index.html
[61] http://www.ulib.org/webRoot/Books/Numerical_Recipes/bookcpdf.html
[62] http://gams.nist.gov
[63] http://www.cpc.cs.qub.ac.uk/cpc/
[64] http://http://emlib.jpl.nasa.gov/

[20] D. Duffie. *Dynamic Asset Pricing Theory*. Princeton University Press, 1992.

[21] R. Rebonato. *Interest-Rate Option Models*. Wiley & Sons Ldt, UK, 1996.

[22] A. Bondeson and G. Y. Fu. Tunable integration scheme. *Computer Physics Communications*, 66:167, 1991.

[23] S. Betts, S. Browne, J. Dongarra, E. Grosse, P McMahan, and T. Rowan. Netlib Software Repository[65].

[24] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. PETSc home page. The PETSc home page[66], 1999.

[25] A. Jaun, K. Blomqvist, A. Bondeson, and T. Rylander. Iterative solution of global electromagnetic wavefields with finite elements. *Computer Physics Communications*, 2001.

[26] P. E. Kloeden and E. Platen. *Numerical Solution of Stochastic Differential Equations*. Springer-Verlag, second corrected printing edition, 1995.

[27] N.G. Van Kampen. *Stochastic Processes in Physics and Chemistry*. North-Holland Personal Library, 1992.

[28] J. Carlsson. The Monte-Carlo method – a cookbook approach[67]. Alfvén Laboratory, KTH, SE-100 44 Stockholm, Sweden, 1997.

[29] C. K. Birdsall and A. B. Langdon. *Plasma physics via computer simulation*. McGraw-Hill, 1985. The Plasma Theory and Simulation Group Homepage[68].

[30] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. Adam Hilger, 1988.

[31] T. Yabe and T. Aoki. A universal solver for hyberbolic equations by cip. *Computer Physics Communications*, 66:219, 1991.

[32] H. Takewaki and T. Yabe. The cubic-interpolated pseudo particle (cip) method. *Computer Physics Communications*, 70:355, 1987.

[33] T. Utsumi, T. Kunugi, and T. Aoki. Stability and accuracy of the cip scheme. *Computer Physics Communications*, 101:9, 1997.

[65] http://www.netlib.org
[66] http://www.mcs.anl.gov/petsc
[67] http://fedu52.fed.ornl.gov/%7ecarlsson/MonteCarlo/cookbook/notes.html
[68] http://ptsg.eecs.berkeley.edu/