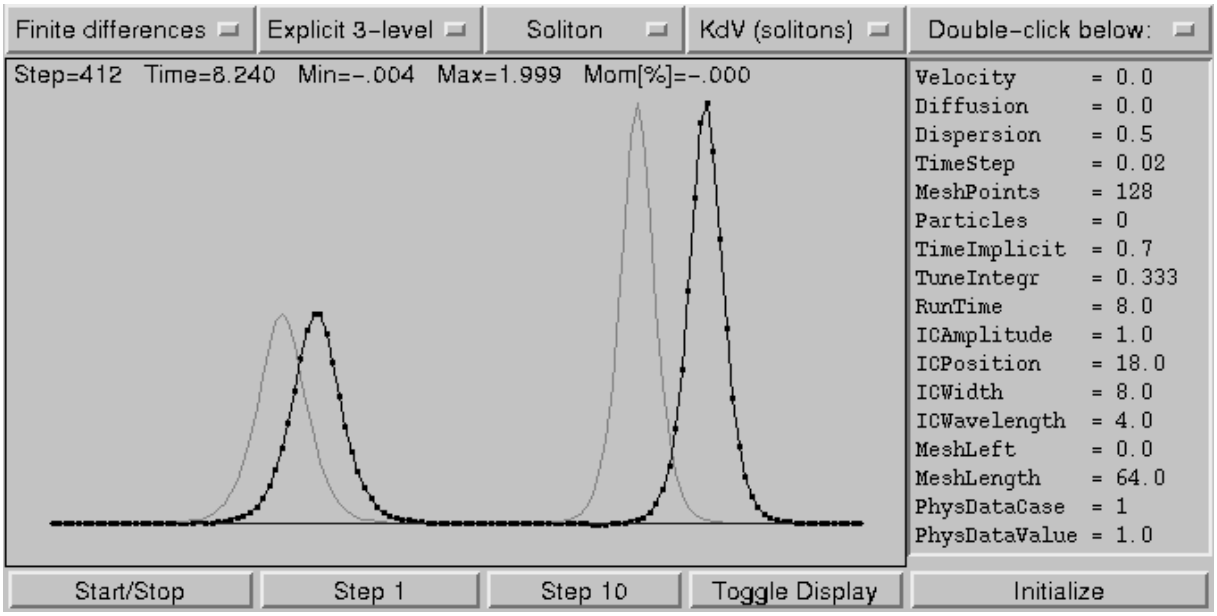
NUMERICAL METHODS FOR PARTIAL DIFFERENTIAL EQUATIONS

http://www.lifelong-learners.com/pde

André Jaun <jaun@kth,se>

April 24, 2003



JBONE applet: the Korteveg-DeVries equation is here solved to propagate two solitons with finite differences.

2003 edition of the Swedish Netuniversity, held simultaneously at the
Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
Royal Institute of Technology, Stockholm, Sweden

This document is typeset with LaTeX 2$_\varepsilon$[1] using the macros in the `latex2html`[2] package.

© A. Jaun, 2003, TRITA-NA-0305, Royal Institute of Technology, Stockholm, Sweden.

[1] http://www.tug.org/

[2] http://cdc-server.cdc.informatik.tu-darmstadt.de/%7Elatex2html

Contents

1 INTRODUCTION **3**
1.1 How to study this course . . . 3
1.2 Differential Equations . . . 5
1.2.1 Ordinary differential equations . . . 5
1.2.2 Partial differential equations . . . 6
1.2.3 Boundary / initial conditions . . . 7
1.2.4 Characteristics and dispersion relations . . . 7
1.2.5 Moments and conservation laws . . . 8
1.3 Prototype problems . . . 8
1.3.1 Advection . . . 9
1.3.2 Diffusion . . . 9
1.3.3 Dispersion . . . 10
1.3.4 Wave-breaking . . . 11
1.3.5 Schrödinger . . . 12
1.4 Numerical discretization . . . 13
1.4.1 Convergence, Richardson extrapolation . . . 13
1.4.2 Sampling on a regular grid . . . 14
1.4.3 Finite elements . . . 15
1.4.4 Splines . . . 15
1.4.5 Harmonic functions . . . 16
1.4.6 Wavelets . . . 17
1.4.7 Sampling with quasi-particles . . . 19
1.5 Computer quiz . . . 21
1.6 Exercises . . . 22
1.7 Further reading . . . 23
1.8 Interactive evaluation form . . . 24

2 FINITE DIFFERENCES **25**
2.1 Explicit 2 levels . . . 25
2.2 Explicit 3 levels . . . 27
2.3 Lax-Wendroff . . . 29
2.4 Leap-frog, staggered grids . . . 30
2.5 Implicit Crank-Nicholson . . . 31
2.5.1 Advection-diffusion equation . . . 31
2.5.2 Schrödinger equation . . . 32
2.6 Computer quiz . . . 34
2.7 Exercises . . . 35
2.8 Further reading . . . 37
2.9 Interactive evaluation form . . . 38

3 FINITE ELEMENT METHOD **39**
3.1 Mathematical background . . . 39
3.2 An engineer's formulation . . . 40
3.3 Numerical quadrature . . . 42
3.4 Implementation and solution . . . 43
3.5 Linear solvers . . . 44

3.6 Variational inequalities . . . 46
3.7 Computer quiz . . . 48
3.8 Exercises . . . 49
3.9 Further Reading . . . 50
3.10 Interactive evaluation form . . . 50

4 FOURIER TRANSFORM **51**
4.1 FFT with the computer . . . 51
4.2 Linear equations. . . . 52
4.3 Aliasing, filters and convolution. . . . 54
4.4 Non-linear equations. . . . 56
4.5 Computer quiz . . . 58
4.6 Exercises . . . 58
4.7 Further Reading . . . 59
4.8 Interactive evaluation form . . . 59

5 MONTE-CARLO METHOD **61**
5.1 Monte Carlo integration . . . 61
5.2 Stochastic theory . . . 61
5.3 Particle orbits . . . 62
5.4 A scheme for the advection diffusion equation . . . 65
5.5 When should you use Monte Carlo methods? . . . 66
5.6 Computer quiz . . . 67
5.7 Exercises . . . 67
5.8 Further Reading . . . 69
5.9 Interactive evaluation form . . . 69

6 LAGRANGIAN METHOD **71**
6.1 Splitting advection from the rest . . . 71
6.2 Cubic-Interpolated Propagation (CIP) . . . 71
6.3 Non-Linear equations with CIP . . . 73
6.4 Quiz . . . 73
6.5 Exercises . . . 74
6.6 Further Reading . . . 74
6.7 Interactive evaluation form . . . 74

7 WAVELETS **75**
7.1 Remain a matter of research . . . 75

8 THE JBONE USER MANUAL **79**
8.1 Source code & installation . . . 79
8.2 Program execution . . . 80
8.3 Program structure & documentation . . . 81
8.4 An object-oriented example: Monte Carlo in JBONE . . . 82
8.5 Interactive evaluation form . . . 82

9 PROBLEM BASED LEARNING ENVIRONEMENT **83**
9.1 Typesetting with TEX 83
9.2 Progamming in JAVA 85
9.3 Parameters and switches in HTML 87
9.4 Interactive evaluation form 88

10 COURSE EVALUATION AND PROJECTS **89**
10.1 Interactive evaluation form 89
10.2 Suggestions for one-week projects 89

PREFACE

This is the syllabus of the courses 2D4232 and 2D5246 taught at the Royal Institute of Technology (KTH, Stockholm) to graduate students from engineering and quantitative social sciences. With the development of collaborative teaching and distance learning, this material is now also shared with the Swiss Federal Institutes of Technology (EPFL, Lausanne), the Swedish Netuniversity and other independent learners from the Internet.

Recognizing the value of an introductory level text describing a whole range of numerical methods for partial differential equations with practical examples, a *problem based learning laboratory* has been developed around this highly interactive document. [3] 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 editable 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 third web edition accessible to everyone on the Internet, I would like to thank Johan Carlsson, Johan Hedin, Thomas Johnson (KTH, Stockholm) and Laurent Villard (EPFL, Lausanne) who have been assisting me over the years and have all significantly contributed to enrich the course material. Ambrogio Fasoli (MIT, Cambridge) provided the measurement of an Alfvén instability to illustrate the importance of aliasing in the digital acquisition of data–an example from fusion energy research where we compare the solutions from numerical models with the world's largest tokamaks.

I hope that this learning environment 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 parts were not original, and the original parts were not good".

André JAUN, Stockholm, 1997–2003

[3] accessible with a java-powered web browser from `http://www.lifelong-learners.com/pde`

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 shows that major steps are taken when a theory exposed by the teacher (in a regular classroom-, a video-lecture or a syllabus) is first reviewed, questionned, discussed with peers (best accross the table during a coffee break, alternatively in a video-conference, a user forum discussion or a computer quiz) and later applied to solve practical problems.

The educational tools that have been developed here reflect this pedagogical understanding: they are meant to be combined in the order / manner that best suits every learner. Along the same line, an assessment of the progress made is performed continuously with a system of bonus points. They reward original contributions from different activities, including questions and answers in the forum discussions and the assignments that are corrected the teacher. Particularly nice solutions are usually shared with the rest of the class.

An example showing how you can study the material during a typical day of an intensive course involves three distinct phases (those market with an asterisk* have been partly restricted and require that you login to enjoy a full pedagogical support).

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

- **Video-Lecture.*** From the course main page[4], 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: syllabus* for an index to the sections where you can find 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 applets.

- **Syllabus.** Repeat the analytical derivations that are on purpose scarce so as to force you to complete the intermediate steps on your own.
- **Applets.*** Perform the numerical experiments that are suggested and modify the applet parameters to challenge your understanding. The original parameters can always be recovered and the applet restarted (check the frequently asked questions FAQ to find out how to do this with PowerToys by pressing CTRL-F5 using Microsoft Explorer and by clicking RIGHT in the white area + press SHIFT + select *Reload* using Netscape).

[4] http://www.lifelong-learners.com/pde

- **Quiz.*** Answer the questions that deal with the core of the subject; review the material when appropriate.
- **Tutorials / Video-Conference.*** Organized for a sufficient number of participants, tutorials (locally) or video-conferences (at a distance) are sometimes organized to discuss and refine an understanding that 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* (others the *world*) forum to obtain and provide help to better understand the material. You are strongly encouraged to discuss related topics and comment answers from your classmates: remember that we judge this virtual classroom activity sufficiently important that your participation is rewarded with bonus points – it doesn't really matter whether your arguments make sense or not, it is the teachers' duty to correct potential errors). Consult the *Forum: rules* and take a minute to think how you will make your contributions beneficial for everyone.

Problem based learning (5h). Most important after acquiring a new skill is to apply and exercise 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* (e.g. 1.00). Below the handout, up to three input windows allow you to submit your solution to the compilers:

- **TeX.*** The first window takes regular text (ASCII) and LaTeX input, 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 solution, how you implement it and discuss the numerical output. Users who are not familiar with LaTeX generally find it easy to modify the templates and use the *symbols* dictionary from the link on the top of the input window.
- **JAVA.*** The content of the second window is compiled into an applet, allowing you to execute your own schemes directly in your browser. Don't worry, your *java* inserts will only be modifications of existing templates so that you really don't need to be a java programer to follow this course. Careful: be sure to always correct all the compiler errors before you switch to a new exercise, or your applet will stop working everywhere. Also, your web browser probably stores old data in it's caching directory; you have to force it to reload your newly compiled version (again, check the frequently asked questions FAQ to find out how to do this). If you don't get immediate programming advice from the *User Forum*, you may temporarily deactivate a problematic scheme using the `/* java comment delimiters */`.
- **Parameters.*** Choose values that highlight the phenomenon you want to illustrate (the parameter *tags* allow you preset default values that can be different in every exercise.

Be sure to submit only one input window at any time and always compile your work before you navigate further to the syllabus or the user Forum. Sometimes, the *Back*

button of your browser restores data you may have lost... but don't count on it! When your solution is ready and documented, click on the *CheckMe* button (appears on the left of every *assignment* after the first compilation) and don't forget to press *Submit Check* (bottom of the table) to send your solution for correction to the teachers. Take into account the modifications that will be suggested until your solution is accepted and your exercise is signalled as passed.

- **Evaluation.*** The last section of every chapter always consists of a short anonymous *evaluation form* where you are kindly requested to communicate your impressions immediately before you start a new chapter. By repeating this as you progress through the course, we will not only maximise your satifaction at the end of the course, but optimize the path leading there.

The amount of work in each module is sufficiently large that it is usually not possible to complete all the course requirements within the short duration of an intensive course; rather than proceeding sequentially, it is important that you at least start one assignment before every topic is discussed in a tutorial / video-conference: remember, these are not lectures and are rather useless if you aren't a little bit familiar with the course material.

Individual work (1 week). In an intensive course, one extra week may be necessary to complete all the work for the assignments. Combine the e-learning tools until you meet the course requirements. Finally, please don't forget to submit your course evaluation allowing us to improve the content 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. The intention is to reward taking a risk (stricktly limited to one week), to assess whether some tools could potentially result in an useful development in the frame of a PhD thesis. A small report with no more than six A4 pages will be published under the course main web page.

1.2 Differential Equations

1.2.1 Ordinary differential equations

Rather than repeating details of elementary numerical analysis, this section is only meant to review a few methods that will be used later in the chapter dealing with the Monte Carlo method for partial differential equations (PDE).

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 an evolution in time

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

The example shows 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.

Introducing a discretization $y_n = y(t_n)$ with a finite number n of time steps $t_n = nh + t_0$, a straight forward manner to solve an ODE is to approximate the derivative with a finite difference quotient forward in time $y' = (y_{n+1} - y_n)/h + \mathcal{O}(h^2)$, leading 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 said to be *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; this is however computationally expensive when the function cannot be inverted analytically.

More precision can be obtained by using a finite difference quotient symmetric in time $y' = (y_{n+1} - y_{n-1})/2h + \mathcal{O}(h^4)$, which is obtained by summing two Taylor series forward and backward. This yields the so-called *midpoint formula*

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

which it is second order accurate, but requires a special initialization to generate the additional values that are needed from the past. Writing the mid-point rule as

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

the initialization problem is cured and results in the well known *second order Runge-Kutta* method

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

Such elementary and more sophisticated methods are commonly available in software packages such as MATLAB; type `help ode23` to find out more about how to use them. Methods for ordinary equations are usually robust, but they become extremely inefficient when the problem is *stiff*, i.e. involves two very different scales that limit the step size to the shorter one even when this is not necessarily relevant (exercise 1.01).

1.2.2 Partial differential equations

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

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

The equation is called *second order* if there are not more than two derivatives acting on the same unknown function f; it is *linear* if D is linear in f, and is *homogeneous* if all the terms in D depend only on f.

Initial value problems involving a combination of linear and non-linear operators

$$\frac{\partial f}{\partial t} = \mathcal{L}f \approx \mathcal{L}_1 f + \mathcal{L}_2 f + \ldots + \mathcal{L}_m f \tag{1.2.2\#eq.2}$$

are often solved with an *operator splitting*. The idea, based on the standard separation of variables, is to decompose the operator into a linear combination with m terms and to solve the entire problem in sub-steps where each part of the operator is evolved separately and the others are kept fixed. One particularly popular splitting is the alternating-direction implicit (ADI) method, where only one spatial dimension is evolved at any time using an implicit scheme. Non-linearities are often split from the linear terms and are treated with appropriate techniques.

1.2.3 Boundary / initial conditions

Depending on the problem, initial conditions (IC)

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

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

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

and are called *Dirichlet (b=0)*, *Neumann (a=0)*, or *Robin (c=0)* conditions. Other forms include the *periodic* condition $f(x_L) = f(x_R)$ with $\{x_L, x_R\} \in \partial\Omega\ \forall t$, and the *outgoing-wave* conditions if the domain is open. To prevent reflections from the computational boundary of an open domain, it can be useful to introduce *absorbing boundary conditions*: they depend on the numerical method, but typically consist of small layers of an artificial material that absorbs the outgoing in a few mesh cells. The *perfectly matched layers* [4] are often used in problems involving electromagnetic waves and a simplified version of that technique will be discussed with the leap-frog method (exercise 2.08).

1.2.4 Characteristics and dispersion relations

The characteristics of a PDE can loosely 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 using an ansatz $f(x,t) = f(x(t),t)$

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

Three categories of equations depend on the sign of the discriminant:

- $B^2 - AC < 0$ 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 themselves and will be exploited numerically in the so-called *Lagrangian methods* in chapter 6.

The local properties of a linear equation can be further analyzed using a harmonic ansatz $f(t,x) = f_0 \exp(-i\omega t + ikx)$, which transforms the differential operators in (t,x) into an

algebraic expression in (ω, k) called *numerical dispersion relation* or *Von Neuman stability relation*:

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} = 0 \qquad \Longrightarrow \qquad -\omega + uk = 0 \tag{1.2.4\#eq.2}$$

Dispersion relations relate the phase velocity $\Re e(\omega)/k$ or the growth rate $\Im m(\omega)$ 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 $\omega\Delta t$. Take for example the wavenumber $k_{\text{eff}} = -if'/f$ that is obtained when a mid-point rule (1.2.1#eq.3) is used to approximate the first derivative of the spatial function $f(x) = \exp(ikx)$:

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

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 wavelength. The forward difference (1.2.1#eq.2)

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

has an imaginary part, showing that small wavelengths $k\Delta x \to \pi$ will be strongly damped $f \propto \exp[-\Im m(k_{\text{eff}})\Delta x]$.

1.2.5 Moments and conservation laws

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

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

Usually, $\mathcal{M}_0$ refers to 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 are a clear signal for a loss in accuracy in the numerical solution as it evolves in time.

1.3 Prototype problems

This section introduces some of the most important initial value problems from science and engineering, illustrated for simplicity in a 1D periodic slab $x \in [0; L[$.

1.3.1 Advection

Also called *convection*, the advection models a streaming of infinitesimal elements in a fluid. It generally appears when a transport process is modeled in a Eulerian representation using 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 advection equation can be solved analytically $f(x,t) = f_0(x-ut)\ \forall f_0 \in \mathcal{C}(\Omega)$, showing explicitly the underlying characteristic $x = ut$. Try the JBONE applet in the document on-line to compute the evolution of a Gaussian pulse with an advection computed using the Lagrangian CIP method from chapter 6.

Numerical experiments: advection

1. Switch the initial condition from a `Gaussian` to `Cosine` and `Box` functions. Observe how the sharp edges disappear in the numerical solution; this is in contradiction with the analytical solution obtained above.
2. Reduce the `Velocity` input parameter and verify that the advection speed is reduced accordingly. Increase the velocity above unity and determine the threshold above which the numerical solution clearly becomes unstable.

Both experiments try to show you that numerical simulations have to be used carefully, withing limits of applicability that will be discussed precisely in the comming sections. Note that for a constant advection speed, the wave equation can always be written in flux-conservative form

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

where second term is the divergence of a flux. This shows explicitly how numerical methods for the advection equation can in principle be used also for wave problems.

1.3.2 Diffusion

At the microscopic scale, *diffusion* is in fact related to a random motion leading 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 diffusion coefficient. For a homogeneous medium, the combined advection-diffusion equation

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

can be solved analytically in terms of the Green function (exercise 1.03)

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

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 diffusion of an initial box function computed using the finite element method from chapter 3.

Numerical experiments: diffusion

1. Switch from an initial `Box` to a `Cosine` function and vary the `ICWavelength` parameter in powers of two: 2, 4, 8, 16, 32. Observe how short spatial scales decay more rapidly than larger scales: measure the decay rate as a function of the wavelength and try to reproduce the analytical result below.
2. Back to the the initial `Box` function, observe how sharp edges disappear quickly, while a large scale structure gradually evolves into a Gaussian – reminder of the Green function from the analytical solution.
3. Add a finite advection `Velocity=0.2` and observe what happens when advection and diffusion are combined.

A harmonic ansatz $f = \exp i(kx - \omega t)$ can be used to analyze the diffusion equation and leads to the dispersion relation

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

This shows that short wavelengths (large k) are more strongly damped than long wavelengths, with a decay rate proportional to $\exp[-Dk^2t]$ as can readily be observed in the numerical experiments on-line.

1.3.3 Dispersion

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

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

The harmonic ansatz in space and time $f = \exp i(kx - \omega t)$ shows that the phase velocity

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

is larger for short wavelengths (large k propagate faster) than long wavelengths (small k propagate slower). In the Korteweg-DeVries equation, this will explain why large amplitude *solitons* with short wavelengths propagate more rapidly than low amplitudes solitons with long wavelengths.

Unfortunately, dispersion does not always have a physical origin: remember (1.2.4#eq.3), showing how a centered finite difference approximation under-estimates first order derivatives of short wavelengths. This is exactly what happens in the 3 levels scheme for the advection equation, where different Fourier components included in the initial square box function propagate with different velocities.

Numerical experiments: dispersion

1. Switch from a `Box` to a `Gaussian` initial condition to make sure that long wavelengths do indeed propagate with the right phase velocity.
2. Pushing the numerical approximation to the extreme, now switch to a `Cosine` with less than two mesh points per wavelength `ICWavelenth=1.9375`. Observe how the short wavelength oscillations propagate backward, i.e. in the opposite direction from what they are expected in presence of advection only.
3. Discuss what happens if a harmonic function is initialized with exactly 2 mesh points per wavelength.

Numerical dispersion affects almost every approximation that will be discussed in this course. It is therefore of crucial importance to make sure that numerical solutions are always dominated by the physical and not the numerical counterpart.

1.3.4 Wave-breaking

Wave-breaking is a non-linearity that is particularly nicely understood when surfing at the see shore, where shallow waters steepen the waves until they break. The process can be modeled theoretically from the advection equation by choosing the velocity proportional to the amplitude $u = f$:

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

Since a local maximum (large f) propagates faster than a local minimum (small f), both will eventually meet; the function clearly becomes multi-valued causing the wave (and our numerical schemes) to break. Sometimes, the 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 diffusion, which damps 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 chapter 2.

Numerical experiments: shock waves

1. Vary the `Diffusion` parameter in the interval $D \in [0.4; 1]$ where the results are more or less satisfactory at the beginning of the simulation when the time reaches $T = 10$. Click in the plot window to measure the propagation speed and count the density of mesh points to evaluate the steepness of the shock front as a function of the diffusion.
2. Observe what happens if the `Diffusion` parameter is chosen outside the interval $D \in [0.4; 1]$.
3. Without spending much effort on a topic that will be discussed further soon, try to modify the simulation parameters to perform a physically relevant simulation using a small `Diffusion=0.1`.

Another type of non-linear equations where the wave-breaking is balanced by dispersion leads to the *Korteweg-DeVries* equation for *solitons*

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

The document on-line shows how large amplitudes solitons (short wavelengths) propagate faster than lower amplitudes (long wavelength), in agreement with the dispersion analysis previously performed in sect.1.3.3.

Numerical experiments: solitons

1. Study the collision that occurs when the time reaches around $T = 104$. Monitor precisely the peak of each pulse as a function of time, to verify that solitons in fact never get superposed – even if the fast pulse ends up in front and eventually propagates independently of the slower pulse.
2. Initialize a Gaussian pulse and observe how it decays into a so-called *train of solitons* that undergo multiple collisions in a periodic domain.

1.3.5 Schrödinger

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

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

In quantum mechanics, the Schrödinger equation is used to evolve the complex wave-function $|\psi >= \psi(x,t)$ describing the probability $< \psi|\psi >= \int_\Omega |\psi|^2 dx$ to 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 and assume a constant potential $V(x) = 0$. The JBONE applet on-line shows the evolution of a low energy $E = - < \psi|\partial_x^2|\psi >$ (long wavelength) particle initially known with a rather good accuracy in space (narrow Gaussian envelope): the wave-function $\Re e(\psi)$ (blue line) oscillates, the probability density $|\psi|^2$ spreads out (black line) showing that the position gradually becomes uncertain.

Numerical experiments: Heisenberg's uncertainty principle

1. Reduce the `ICWavelength` down to 2 and relate an increase in the kinetic energy of a wave-packet to achieve a more accurate localization.
2. Reload the applet using a low kinetic energy `ICWavelength=6` and reduce the initial knowledge of the position `ICWidth=12` to reduce the spreading of the wave-packet – and therefore to reduce the uncertainty in the velocity.

1.4 Numerical discretization

Numerical solutions are obtained by evolving a discrete set of values $\{f_j\}$, $j = 1, N$ with 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 from a "local guru", your choice for a discretization should 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 required anyway,
- the boundary conditions that can be difficult to implement in some methods,
- the structure of the differential operator (the analytic 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 from different methods; it aims at giving you the overview, which is required to make optimal choices to start your numerical modeling – leaving advanced tricks to advanced and specialized literature when this will become necessary.

1.4.1 Convergence, Richardson extrapolation

One of the most important properties of a numerical approximation based on a finite number of data points is that the 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 and 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 acquired in the middle of every interval $x_n = (n + 1/2)/N$. The lowest order approximation $f(0) \approx f^{(N)}(x_0) = (2N)^{-\alpha}$ clearly converges to zero at the origin provided $\alpha \geq 0$. The *convergence rate* can be estimated from the geometric sequence of approximations $\{f^{(i)}\}$, obtained by successively doubling the numerical resolution with $i = N, 2N, 4N$

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

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

Using a sequence where the numerical resolution is doubled for each step $Q = 2$ and where the convergence rate r is known, is possible to improve upon the known values using the so-called *Richardson extrapolation*

$$F^* = F_2 + \frac{F_2 - F_1}{Q^r - 1} \qquad (1.4.1\#\text{eq.2})$$

The more accurate solution F^* is obtained from the evaluation F_1 and its refinement F_2.

1.4.2 Sampling on a regular grid

The main advantage of sampling data on a grid is its 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 (1.4.2#fig.1) shows that the sampled function is unknown

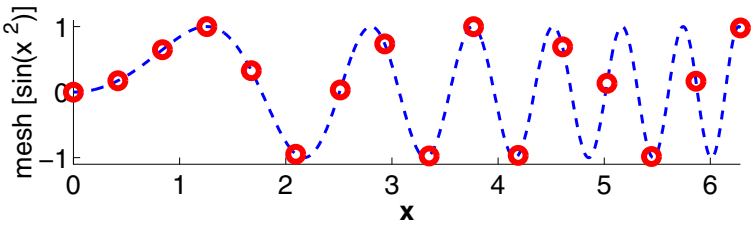


Figure 1.4.2#fig.1: Approximation of $\sin(x^2)$ on a homogeneous mesh.

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

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

Combining Taylor expansions from the same function evaluated on neighboring locations f_{j-1}, f_j, f_{j+1}, f_{j+2}

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

it is easy to obtain the finite difference approximations used in sect.1.2.1 and to verify (exercise 1.06) the formula for the k-th derivative $f^{(k)}$ from Abramowitz [1] §25.1.2

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

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

$$\begin{pmatrix} n \\ k \end{pmatrix} = \frac{n(n-1)\cdots(n-k+1)}{k!}$$

Remember that a discretization on a grid is non-compact: the convergence depends not only on the initial discretization, but also on the interpolation that is required afterwards 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 and need not to be homogeneously distributed. Most common are the normalized "roof-top" functions

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

which yield a piecewise linear approximation for $f(x)$ and a piecewise constant derivative $f'(x)$ that is defined almost everywhere in the interval $[x_1; x_N]$. Boundary conditions are imposed by modifying the functional space $\mathcal{B}$, e.g. by taking "unilateral roofs" at the boundaries.

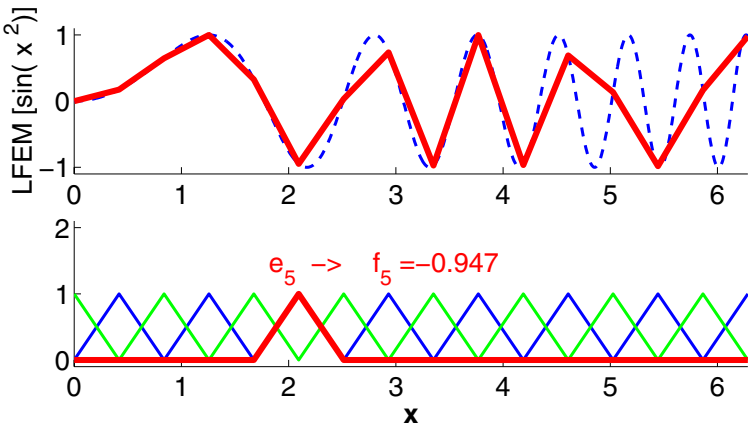


Figure 1.4.3#fig.1: Approximation of $\sin(x^2)$ with homogeneous linear finite elements.

Generalizations with "piecewise constant" or higher order "quadratic" and "cubic" FEM are constructed along the same lines. The capability of densifying the mesh where short spatial scales require a higher accuracy is of particular interest. Figure (1.4.3#fig.1) doesn't exploit this, showing instead what happens when the numerical resolution becomes insufficient: around 20 linear or 2 cubic FEM are typically required per wavelength to achieve a precision around 1%. A minimum of 2 is of course necessary only to resolve the oscillation.

1.4.4 Splines

Starting from an approximation on an inhomogeneous mesh, the idea of splines is to provide a global interpolation that is continuous up to a certain derivative. Using a cubic Hermite polynomial with tabulated values for the function and the second derivative

$$f(x) = Af_j + Bf_{j+1} + Cf''_j + Df''_{j+1} \qquad (1.4.4\#\text{eq.1})$$

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

it is straight forward to first order 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 are calculated rather than measured. Requiring that $f'(x)$ be continuous from one interval to the next, (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 (1.4.4#fig.1) illustrates the procedure and shows the excellent quality of a cubic

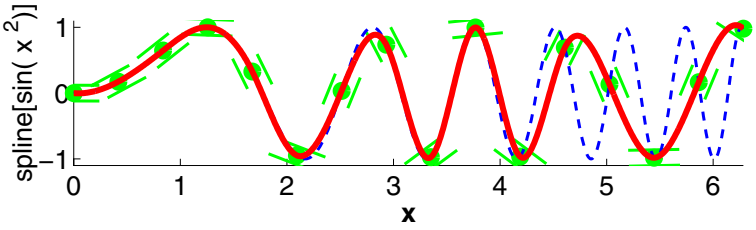


Figure 1.4.4#fig.1: Approximation of $\sin(x^2)$ with cubic Hermite splines.

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

1.4.5 Harmonic functions

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

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

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

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

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

with $W^2 = \exp(2i\pi/2^{M-1})$ until a sum of DFT of length $M = 2$ is obtained. Figure (1.4.5#fig.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.

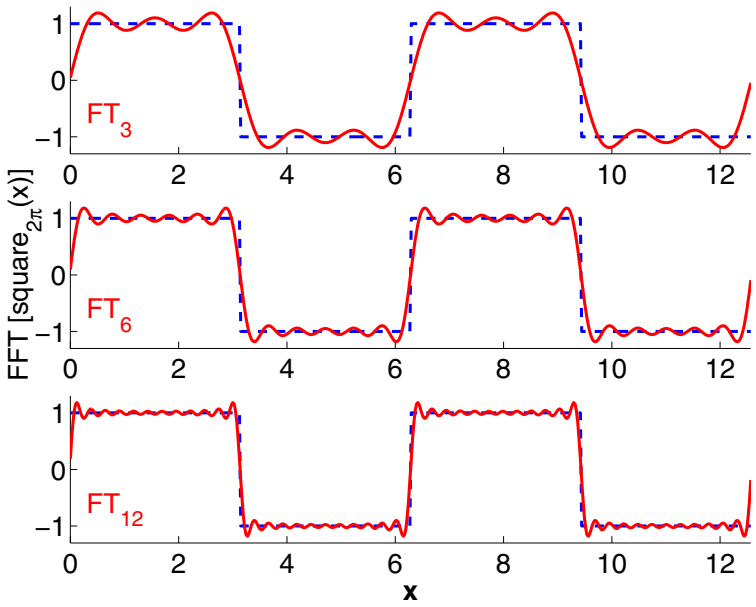


Figure 1.4.5#fig.1: Square wave $\sum_{m=0}^{M} \frac{2}{\pi(2m+1)} \sin\left(\frac{(2m+1)\pi x}{L}\right)$ with $M = 3, 6, 12$.

It should be a surprise to none to hear that a harmonic decomposition is well suited for smooth global functions with long wavelengths $\lambda \sim L$ that result in a narrow spectrum $|k| \leq 2\pi/\lambda \ll \pi/\Delta x$. Note that the convergence with an increasing number of Fourier components is not polynomial. The implementation of non-periodic boundary conditions can sometimes 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

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

This is best illustrated with *Haar wavelets* in figure (1.4.6#fig.1), showing that the piecewise constant approximation at a 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 FFT, Mrs Daubechies proposed a fast $\mathcal{O}(N)$ linear discrete wavelet transformation

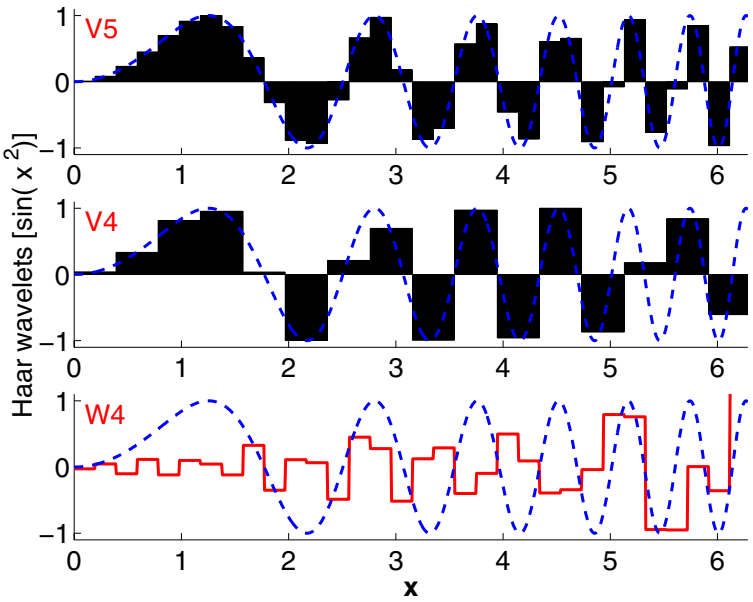


Figure 1.4.6#fig.1: Successive approximation of $\sin(x^2)$ using Haar wavelets.

(DWT) called DAUB4

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

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

(1.4.6#eq.2)

which is applied successively to the function $\tilde{f}$ together with the permutation until only the first two components remain. The inverse is calculated by reversing the 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 (1.4.6#fig.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.

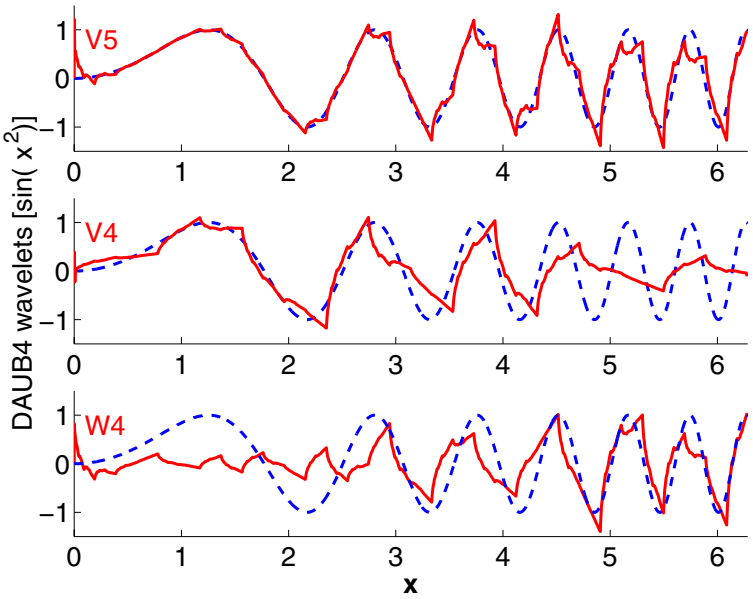


Figure 1.4.6#fig.2: Successive approximation of $\sin(x^2)$ using DAUB4 wavelets.

Note that next to Haar and Daubechies wavelets, a whole family can be generated with a cascading process that has very nicely been illustrated in the Wavelet Cascade Applet[5]. Wavelets are still relatively new and remain a matter of active research [29]: they have been proven useful in approximation theory, but are still being developed both for differential and integral equations. Clearly apparent in the figures (1.4.6#fig.1) and (1.4.6#fig.2) is the difficulty of incorporating the boundary conditions. Another issue that will become clear in the coming sections, is the relatively large cost to calculate overlap integrals.

1.4.7 Sampling with quasi-particles

A completely different way of approximating a function is to use a statistical sampling with 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 the shape, and x_i the position of each particle. In the JBONE applet, the particle shapes are assumed to be Dirac pulses $S_i(x) = \delta(x)$, meaning that the particles are localized in an infinitely small interval around x_i

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

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

For simplicity, the weight is often set to unity $w_i \equiv 1$. As a matter of fact, this form of discretization does 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 with Dirac pulses does converge

$$\int_a^b dx \;\; x^K g(x) = \int_a^b dx \;\; x^K \sum_{i=1}^{\infty} w_i \delta(x - x_i) = \sum_{i=1}^{\infty} w_i x_i^K \qquad (1.4.7\#\text{eq.3})$$

Without a local convergence, it is of course difficult to compare a quasi-particle discretization with a different approximation, based for example on finite elements. The particle solution is therefore projected onto a set of basis functions $\{\varphi_j\}_{j=1}^{N_\varphi}$

$$\begin{aligned} f_\varphi(x) &= \sum_{j=1}^{N_\varphi} \frac{\int f(x)\varphi_j(x))\, d\xi}{\sqrt{\int \varphi_j(\xi)^2\, d\xi}} \varphi_j(x) \\ &= \sum_{j=1}^{N_\varphi} \sum_{i=1}^{N} \frac{\int w_i \delta(\xi - x_i)\varphi_j(\xi)\, d\xi}{\sqrt{\int \varphi_j(\xi)^2\, d\xi}} \varphi_j(x) \\ &= \sum_{j=1}^{N_\varphi} \sum_{i=1}^{N} \frac{w_i \varphi_j(x_i)}{\sqrt{\int \varphi_j(\xi)^2\, d\xi}} \varphi_j(x) \end{aligned} \qquad (1.4.7\#\text{eq.4})$$

In JBONE, the projecting basis functions are piecewise linear roof-top functions. Note that it is important to use the same set of functions for the projection and the plot: figure (1.4.7#fig.1) shows how a Gaussian particle distribution appears after projection, when the result is plotted using piecewise constant (box) and linear (roof-top) functions. The dashed line incorrectly mixes boxes for the projection with roof-tops for the plot.

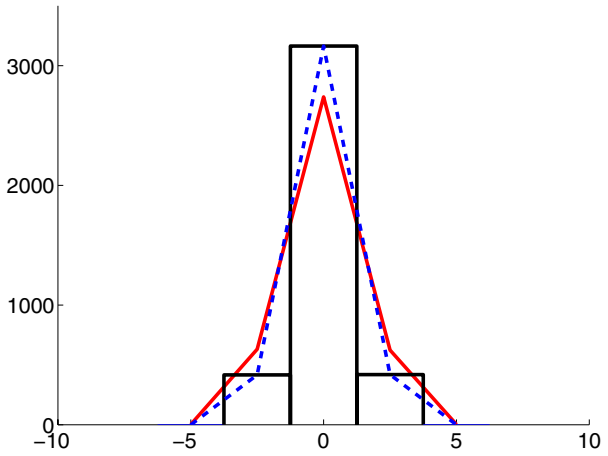


Figure 1.4.7#fig.1: The solid lines are legitimate projections of a Gaussian particle distribution on piecewise constant (box) and linear (roof-top) basis functions. The dashed line illustrates what happens if boxes are used for the projection and mistakenly mixed with roof-tops for the plot.

In the applet 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 (random walkers). 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

This algorithm produces a particle density in the interval $[a; b]$ that is proportional to the initial condition $f_0(x) - f_{\min}$.

1.5 Computer quiz

1. Do you have to register to study this course on-line?
   (a) all the material can be accessed free of charge
   (b) register to gain access to restricted material and pedagogical support
   (c) all the services on this website are for members only

2. The equation $\partial_{xxx} f - \partial_{xx} f \sqrt{1 + (\partial_x f)^2} = \sin(xf)$ with $f(0) = 0$, $f'(0) = 1$
   (a) is an ODE that can be solved for example with the Runge-Kutta method
   (b) is an ODE that can only be solved only with an extra condition f(1)=0
   (c) is a third order PDE that cannot be solved with a Runge-Kutta method

3. The equation $2\partial_{tt} f - 2\partial_{xt} f + 2\partial_{xx} f = 0$ has
   (a) 2 characteristics, i.e. solutions that propagate like waves
   (b) 1 characteristics, i.e. solutions that propagate like heat pulses
   (c) 0 characteristics, i.e. solutions that do no propagate

4. When a heat pulse hits the edge of a perfectly insulated domain
   (a) it leaves the domain while satisfying the Neuman boundary conditions
   (b) it stops and reaches kind of a steady state
   (c) it gets reflected and propagates backward inside the domain

5. Advection and diffusion are not uniquely defined in an inhomogeneous medium
   (a) true: a diffusion coefficient gradient has the same effect as an advection
   (b) false: advection and diffusion are uniquely defined by the prototype problems

6. Successive doubling of the numerical resolution yields values 1.03, 2.98, 4.01, 4.50 that
   (a) converge like the square-root to a value around 6
   (b) converge linearly to a value around 5
   (c) converge quadratically to a value around 5
   (d) do not seem to converge

7. To approximate a Gaussian $\exp(-x^2), x \in [-3; 3]$ with 1% accuracy you need
   (a) approx 100 linear FEM
   (b) approx 1000 linear FEM
   (c) approx 100 particles
   (d) approx 10000 particles

1.6 Exercises

1.00 E-publishing. Familiarize yourself with the electronic submission of assignments and the discussion forums.

1. Follow the TEX link for a list of symbols and type a small text with formulas; learn to interpret the compiler error messages.
2. Change the default JBONE applet parameters to compute the diffusion of a harmonic function; determine the largest diffusion coefficient that seems to give a reasonable result.
3. Read the rules governing the discussion forums and introduce yourself, telling a few words about your background and interests.

1.01 Stiff ODE. Assuming boundary conditions $u(0) = 1; v(0) = 0$, use the MATLAB `ode23` and `ode23s` solvers to integrate the system of equations

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

in the interval $[0; 1]$. Use the variable transformation $u = 2y - z; v = -y + z$ to derive an analytical solution and to show that the problem is stiff. Compare with the solutions obtained first from an integration forward $0 \to 1$ and then, using the terminal value as initial condition, for an integration backward $1 \to 0$.

1.02 Predator-prey model. Study the solutions of the famous Volterra predator-prey model

$$\frac{dx}{dt} = \alpha x - \beta xy; \qquad \frac{dy}{dt} = -\gamma y + \delta xy$$

which predicts the evolution of two populations (x the number of preys and y the number of predators) that depend on each other to survive. 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?

1.03 Fourier-Laplace transform. Solve the advection-diffusion analytically in an infinite 1D slab assuming a constant advection speed u and a constant diffusion coefficient D. *Hint: use a Fourier-Laplace transform first to calculate the evolution of the Green function $G(x - x_0, t)$ obtained from the initial condition $\delta(x - x_0)$. Superpose to describe an arbitrary initial function $f_0(x)$.*

1.04 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 the RMS displacement $\langle z^2 \rangle (t)$ 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 with the ergodicity theorem.*

1.05 Convergence. Calculate an approximation for $\sin(kx)+\cos(kx/3)$, $kx \in [0; 2\pi]$ using four different approximations introduced in the first chapter. Use words to sketch the relative local error at $x = 1$ for $k = 1$ and show how each approximations converges to the analytical value when the numerical resolution increases. What about the first derivative?

1.06 Finite difference quotients. Combine Taylor series expansions with a function $f_j = f(x_j)$ sampled on a regular grid $x_j = j\Delta x$ to reproduce the value and determine the accuracy of the three lowest order finite difference quotients in (1.4.2#eq.2). To implement a Neuman condition at the lower edge of a simulation domain $[x_0; x_N]$, calculate two forward finite difference quotients for the first derivative $f'(x_0)$, first to lowest order $\mathcal{O}(\Delta x)$ and second with a higher accuracy $\mathcal{O}(\Delta x^2)$.

1.07 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.*

1.08 Sampling in a 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.*

These problems can be edited / submitted for correction from your web browser by selecting *assignments* under the course main page.

1.7 Further reading

- **ODE, stiff equations and symplectic integration.**
  Numerical Recipes[6] [25] §16, 16.6, Scientific Computing[7] [15] §9–10, Dahlquist [10] §13, Sanz-Serna [28]

- **PDE properties.**
  Fletcher [14] §2

- **Interpolation and differentiation.**
  Scientific Computing [15] §7–8, Abramowitz [1] §25.2–25.3, Dahlquist [10] §4.6, 4.7, Fletcher [14] §3.2, 3.3

- **FEM approximation.**
  Fletcher [14] §5.3, Johnson [20]

- **Splines.**
  Numerical Recipes [25] §12.0–12.2, Dahlquist [10] §4.8

- **FFT.**
  Numerical Recipes [25] §12.0–12.2

- **Wavelets.**
  Numerical Recipes [25] §13.10, `www.wavelet.org`[8] [29]

[6] http://www.library.cornell.edu/nr/
[7] http://www.cse.uiuc.edu/heath/scicomp/
[8] http://www.wavelet.org/wavelet/index.html

- **Software.**
  Computer Physics Communications Library[9] [12], Guide to Available Mathematical Software[10] [24], Netlib Software Repository[11] [5]

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.

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

2 FINITE DIFFERENCES

The finite difference (FD) method is often used when a function $f_j = f(x_j)$ sampled on a homogeneous mesh $x_j = j\Delta x$ satisfies a differential equation that can simply be approximated with finite difference quotients (1.4.2#eq.2). In *explicit schemes*, the solution is calculated directly in terms of known quantities: explicit schemes are usually easy to implement in a program, but more delicate to execute. *Implicit schemes* are more robust at execution, but result in more complicated codes solving a linear system: this can be slow in 2 or higher dimensions and is then often advantageous to reconsider a more general finite elements solution on an inhomogeneous mesh.

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) for example yields an explicit scheme for the advection-diffusion equation that involves only two time levels

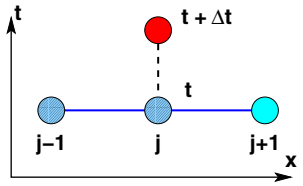


Figure 2.1#fig.1: Explicit 2 levels.

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

The *Courant-Friedrichs-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 relative to the characteristic propagation speed of the mesh $\Delta x/\Delta t$. For every step in time, a new value of the solution `fp[j]` is obtained explicitly from a linear combination involving the nearest neighbors `f[j-1],f[j],f[j+1]`, which are known either from a previous step or the initial condition. This scheme has been implemented in JBONE as

```
for (int j=1; j<n; j++) {
  fp[j]=f[j] -beta *(f[j]-f[j-1])+alpha*(f[j+1]-2.*f[j]+f[j-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 be *AVOIDED* to produce physically meaningful results; to study the numerical properties of a approximation, it is however often illuminating to initialize a box function and check how the intrinsic numerical dispersion / damping of a scheme affects the superposition of both the short and the long wavelengths. In the document on-line, the applet shows the result of an evolution with a constant advection $u = 1$ and no physical diffusion $D = \alpha = 0$. After 128 steps of duration $\Delta t = 0.5$, the

pulse propagates exactly once accross the periodic domain, which is here discretized with 64 mesh points homogeneously distributed over the length $L = 64$ so that $\Delta x = 1$. The lowest order moment (density, (1.2.5#eq.1)) is conserved to a very good accuracy and the function remains positive everywhere as it should . The shape, however, is strongly affected as the sharp edges are smoothed out due intrinsic *numerical damping*.

Numerical experiments: explicit 2-levels

1. Change the initial condition from `Box` to `Cosine`, and vary the wavelength λ = 2–64 mesh points per wavelength to verify that it is the short wavelengths associated with steep edges that are heavily 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, numerical damping can easily cover the physical process you want to model!

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

The properties of a FD approximation are best understood from a *numerical dispersion / Von Neuman stability* analysis, which is carried out on a homogeneous grid $x_j = j\Delta x$ using a hamonic ansatz $f \sim F = \exp(i[kx - \omega t])$. 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 2.1#fig.2 illustrates with vectors in the complex plane how the first three term are combined in the presence of advection only ($\alpha = 0$). Short wavelengths $k\Delta x \simeq \pi/2$ are

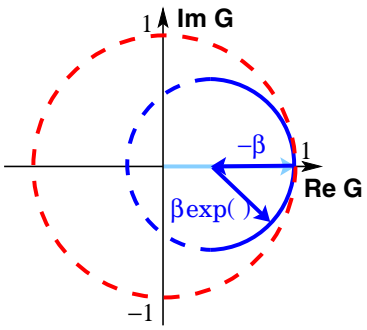


Figure 2.1#fig.2: Numerical dispersion / stability (explicit 2 levels).

heavily damped $|G| < 1$ (area inside the outer circle) if the CFL number is smaller than unity $\beta < 1$; for values exceeding the magic time step where $\beta = 1$, the short wavelengths however grow into a so-called *numerical instability*. Reload the initial applet parameters and try to reproduce this in JBONE.

Numerical experiments: explicit 2-levels (cont)

1. Increase the `TimeStep=1.01` and observe the slowly growing mode with 2 mesh points per wavelength $\lambda = 2\Delta x$, i.e. $k = 2\pi/\lambda = \pi/\Delta x$.
2. Starting from the initial applet parameters, take two steps using a `Cosine` with `ICWavelength=8` mesh points per wavelength to determine the value of the physical `Diffusion` that approximatively doubles the damping observed when $D = 0$ – e.g. doubling the value of the `Min` value displayed in the applet.
3. Choose the value of the numerical parameters (`MeshPoints`, `TimeStep`) to compute a physical evolution of a `Gaussian` pulse with a `ICWidth=2.5` when the advection `Velocity=1` and the `Diffusion=0.1` is relatively small. Make sure that the numerical dispersion and damping are both small in comparison with the physical effect you try to reproduce.

For a diffusive process ($\beta = 0$), the dispersion relation (2.1#eq.3) shows that the 2 levels 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 can conveniently be summarized as

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

It is finally worth pointing out that a backward finite difference for the advection is unstable for negative velocities $u < 0$; this defect of non-centered schemes is sometimes cured with an *upwind difference*, which takes the finite difference forward or backward depending on the local direction of propagation (exercise 2.01).

2.2 Explicit 3 levels

A more accurate scheme in $\mathcal{O}(\Delta x^2 \Delta t^2)$ is obtained from centered differences

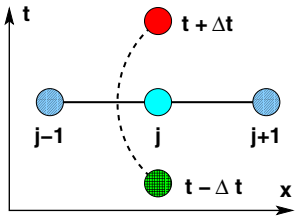


Figure 2.2#fig.3: Explicit 3 levels.

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

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

The scheme involves three time levels (`fm[j], f[j], fp[j]`) and has been implemented in JBONE as

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

A special starting procedure is required to calculate an approximation at a time $-\Delta t$ anterior to the initial condition $t = 0$. This can be implemented by taking one step backwards in time with the explicit 2 levels scheme (2.1#eq.1). The document on-line shows the result obtained using the same parameters for the advection equation as in the previous section. Despite the mathematically higher accuracy in $\mathcal{O}(\Delta x^2 \Delta t^2)$ and the excellent conservation of the lowest order moment, the initial box function is strongly distorted by the *phase errors* created from *numerical dispersion*, when short wavelengths propagate at a different speed from longuer wavelengths due to inaccuracies in the finite difference approximations.

Numerical experiments: explicit 3-levels

1. Switch to `Cosine` and check how the scheme performs for the advection of harmonic functions with `ICWavelength=4` mesh points per wavelength. Measure the propagation speed and compare with the physical `Velocity` that is given as an input parameter. What happens with `ICWavelength=2` mesh point per wavelength?

2. Switch to a `Gaussian` initial condition and observe how the pulse propagates accross the periodic domain. Explain how the initial pulse gets distorted.

3. Switch to `Explicit 2-level` and compare the numerical dispersion / damping in 2 and 3 levels schemes.

4. Back to `Explicit 3-level`, change the sign of `Velocity=-1` and verify that a symmetric scheme indeed works for both directions.

5. Add a small physical `Diffusion=0.1` and verify that the same spatial difference used in (2.1#eq.1) is here in fact always unstable (exercise 2.02).

Because the numerical dispersion slows down short wavelengths $k\Delta x \simeq \pi/2$, the solution can become locally negative even if it was strictly positive at the beginning. This is of course not physical and can be a problem if $f(x,t)$ is a density... Low numerical damping, however, makes the 3 levels scheme attractive when a simple solution is sought only for advection. The choice between a 2 or 3 level scheme ultimately depends on the specific application and always implies a sound monitoring of the numerical parameters at run time.

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* and with any chosen level of 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 $f(x) \to (x_j, f_j)$, $j = 1, N$,

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; for the case of advection (1.3.1#eq.1), 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. Use centered differences for spatial operators

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

The example here yields a second order Lax-Wendroff scheme for advection that is explicit, centered and stable provided that the CFL number $\beta = u\Delta t/\Delta x$ remains below unity:

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

The JBONE applet on-line illustrates how the Lax-Wendroff scheme combines the properties of the 2 and 3 levels schemes from the two previous sections.

Numerical experiments: explicit Lax-Wendroff

1. Examine the propagation and the damping of harmonic functions; try to assess the numerical properties directly from your observations.

2. Test if this scheme also works for negative velocities.

3. Explain what happens when choosing the magic time step for which $\beta = 1$; why can you not rely on this in an inhomogeneous medium?

2.4 Leap-frog, staggered grids

The leap-frog algorithm is often used for the propagation of waves, where a low numerical damping is required with a relatively high accuracy. Relying on two functions $\{f, g\}$ to approximate the scalar wave equation (1.3.1#eq.2) in flux-conservative form, *staggered grids* (where the mesh points are shifted with respect to each other by half an interval, as in fig.2.4#fig.4) are used to evaluate centered differences with an accuracy in $\mathcal{O}(\Delta x^2 \Delta t^2)$:

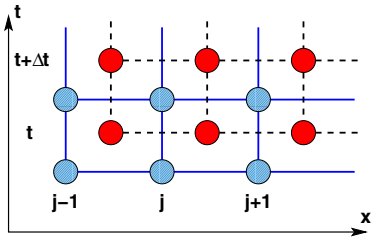


Figure 2.4#fig.4: Staggered grids.

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

The construction of the leap-frog algorithm suggests how the Maxwell equations can be discretized using *finite differences in the time domain (FDTD)* to solve vector equations in terms of the electric and the magnetic fields evaluated on staggered grids (exercise 2.08). The leap-frog algorithm has been implemented in JBONE as

```
for (int j=1; j<=n; j++) {                    //1st equation
  fp[j]=f[j] -beta*(g[j]-g[j-1]); }
fp[0]=f[0] -beta*(g[0]-g[n]);
for (int j=0; j<=n-1; j++) {                  //2nd equation
  gp[j]=g[j] -beta*(fp[j+1]-fp[j]); }
gp[n]=g[n] -beta*(fp[0]-fp[n]);
```

Special care is required when starting the integration, since the initial condition $g_j^{-\Delta t/2}$ determines the direction of propagation (exercise 2.04). In addition to the periodic boundaries shown here above, Dirichlet and absorbing boundary conditions are also frequently used (exercises 2.04 and 2.08).

Numerical experiments: leap-frog

1. Examine the propagation and the damping of harmonic functions; try to assess the numerical properties directly from your observations.
2. Compare the propagation of a box with other explicit FD schemes.

By substitution, note that the leap-frog scheme is equivalent to the 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 linear problem has been solved explicitly in an elegant manner.

2.5 Implicit Crank-Nicholson

As mentioned earlier, *implicit schemes* involve the coupling between unknowns and require solving a linear system. This makes the implementation considerably more complicated, so that the finite element method from sect.3 is often preferable because it offers additional flexibility for the same programming effort.

Two popular applications below are based on the scheme that has originally been proposed by Crank and Nicholson: the first deals with diffusion dominated problems and the second solves the time-dependent Schrödinger equation from quantum mechanics.

2.5.1 Advection-diffusion equation

With Crank-Nicholson, the centered differences in space are evaluated in the same manner and with equal weights for the present and the future values of the function:

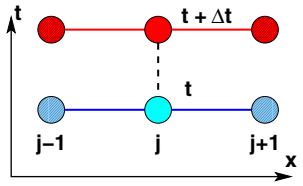


Figure 2.5#fig.5: Implicit Crank-Nicholson.

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

The scheme can conveniently be recast into a linear system with $j = 1, N$ equations

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

where the band matrix on the left has three diagonals and the right hand side can be evaluated explicitly from already known quantities. Performing a von Neumann stability analysis for a pure diffusive process ($u = \alpha = 0$) yields the amplification factor

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

This shows that the Crank-Nicholson scheme is unconditionally stable ($\forall\Delta x$, $\forall\Delta t$) with phase errors affecting mainly short wavelengths $k\Delta x \sim 1$. A similar conclusion can also be reached for the advective part.

The Crank-Nicholson scheme (2.5.1#eq.12) has been implemented in JBONE with

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

The BandMatrix.solve3() method solves the linear system efficiently in $\mathcal{O}(N)$ operations, using an LU-factorization that will be discussed later in sect.3.

The favorable stability property (2.5.1#eq.13) can nicely be exploited in diffusion dominated problems dealing with the evolution of large scale features $\lambda \gg \Delta x$. Starting from a relatively smooth Gaussian pulse that is subject to both advection and diffusion $u = D = 1$, the document on-line shows that a reasonably accurate solution (12 % for the valley to peak ratio when the time reaches 100) can be computed using extremely large time steps $\Delta t = 5$, where $\alpha = \beta = 5$.

Numerical experiments: Crank-Nicholson for advection-diffusion

1. Switch to an initial `Box` function; explain what happens after one time step.
2. Complete the `Implicit 2-level` calculation with a `Box` and terminate with a few tiny `TimeStep=0.1` using the `Explicit 2-level` scheme (2.1#eq.1). Compare the result with the best solution you have obtained so far.

2.5.2 Schrödinger equation

In quantum mechanics, the evolution of a particle is generally modelled with a complex wave function $|\psi >= \psi(x,t)$. If the particle does't decay with time, the total probability of finding this particle somewhere in space of course remains exactly unity at all times $< \psi|\psi >= \int |\psi|^2 dx = 1 \ \ \forall t$. Solving the Schrödinger equation normalized to have Planck's constant $\hbar = 1$, the particle mass $m = 1/2$ and the static potential $V(x)$

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

it is clearly an advantage to formulate a numerical scheme that conserves the probability by keeping the evolution operator unitary even as it is discretized. This can be achieved by writing a formal solution $\psi(x,t) = \exp(-iHt)\psi(x,0)$ in the so-called *Cayley form* with

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

This approximation is unitary, accurate to second-order in time $\mathcal{O}(\Delta t^2)$. It 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 is in fact the same Crank-Nicholson method applied here for the Schrödinger equation:

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

which 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, the scheme 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 j=0; j<=n; j++) {
  z = new Complex(0.,0.5*scale*timeStep*V[j]);
  a.setL(j,mih);                               //Matrix elements
  a.setD(j,pip1.add(z));
  a.setR(j,mih);
  b.setL(j,pih);                               //Right hand side
  b.setD(j,mip1.sub(z));
  b.setR(j,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 j=0; j<=n; j++){                      //Plot norm & real part
  fp[j]=hp[j].norm();
  gp[j]=hp[j].re();
}
```

The code again relies on the complex BandMatrixC.solve3() method to solve the linear system efficiently with $\mathcal{O}(N)$ complex operations using the standard LU-factorization that will be studied in 3.

The document on-line illustrates the scheme with a calculation of the scattering of a wave packet on a square potential barrier. The perfect conservation of the lowest order moment shows that the total probability is indeed conserved for all times.

Numerical experiments: Crank-Nicholson for Schrödinger

1. Modify the energy in the wavepacket `ICWavelength` and verify that the applet indeed reproduces the reflection / transmission coefficients that you expect.
2. Increase the value of the `TimeStep` and explain what you observe.
3. Extend the `RunTime` and observe how quasi-modes are formed, with reflections in the periodic domain that remind what happens to an electron in a crystal.

2.6 Computer quiz

1. A Courant-Friedrichs-Levy (CFL) number larger than unity ($\beta > 1$) means that
   (a) the advection velocity exceeds the characteristic speed of the mesh
   (b) explicit and implicit finite difference schemes are always unstable
   (c) the numerical solution obtained are necessarily inaccurate

2. Dividing the time step by two to converge an explicit scheme for diffusion, you
   (a) multiply the mesh interval size by two
   (b) keep the mesh interval size fixed
   (c) divide the mesh interval size by two
   (d) divide the mesh interval size by four

3. A numerical scheme is necessarily unstable if the amplification factor $G = \exp(-i\omega t)$
   (a) has a real part exceeding unity for one particular wavelength
   (b) has an imaginary part exceeding unity for all the wavelengths
   (c) has a norm exceeding unity for one particular wavelength
   (d) has a real part smaller than zero for short wavelengths

4. Implicit schemes such as Crank-Nicholson
   (a) are always slower than explicit schemes for the same degree of precision
   (b) require the more arithmetic operations per step than explicit schemes in 1D
   (c) require the more arithmetic operations per step than explicit schemes in 2D
   (d) cannot be used for 2D problems

5. An initial box function
   (a) is often used when a physical solution is sought for advection problems
   (b) should always be avoided to minimize inaccuracies in the physical solution
   (c) can make explicit finite difference schemes unstable

6. Compared with other methods, explicit finite difference schemes
   (a) are easy and robust to execute
   (b) are easy to understand and implement in a code
   (c) converge rather slowly
   (d) should be avoided for wave equations

2.7 Exercises

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

2.02 Numerical dispersion. Show 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.

2.03 Shock waves using Lax-Wendroff. Follow the Lax-Wendroff approach to solve the Burger equation (1.3.4#eq.2) using the JBONE applet. Implement a first order scheme and study the numerical convergence for different values of the physical diffusion D. For an extra reward, you can try to go to second order, keeping in mind that it is better not to expand high order non-linear derivatives and to combine $2f(\partial_x f)^2 + f^2\partial_x^2 f = (\partial_x^2 f^3)/3$. 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.*

2.04 Leapfrog resonator. Study the intitialization of the leapfrog scheme (2.4#eq.9) in the JBONE applet and explain how you can affect the direction of propagation of a Gaussian pulse with a proper choice of `gm`. Modify the numerical scheme to incorporate either Dirichlet or Neuman boundary conditions and find a physical analogy that could be modeled with your code.

2.05 European option. Use a finite difference schemes to solve the Black-Scholes equation

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r - D_0) S \frac{\partial V}{\partial S} - rV = 0$$

for the value of a *European Vanilla put option* with a final payoff $Max(E - S, 0)$ assuming the price of the underlying asset $S \in [0; 130]$, a strike 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

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

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

$$\frac{\partial u}{\partial \tau} - \frac{\partial^2 u}{\partial x^2} = 0$$

Derive initial and boundary conditions and test the improved scheme using JBONE.

2.06 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 [2], §8-9). It is shown from perturbative theory that the addition 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, $K = 2\pi/L$ 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.

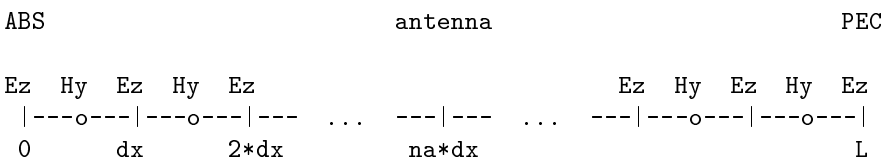
The aim of this exercise is to demonstrate this property. Modify the scheme for the Schroedinger equation (2.5.2#eq.18) to include a periodic potential. Choose the period length for the potential and an amplitude set by `PhysDataValue`. Examine how *plane wave* particles propagate when their wavelength 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.

2.07 Lax-Friedrichs advection. Examine the properties of the symmetric scheme

$$\frac{f_j^{t+\Delta t} - f_j^*}{\Delta t} + u\frac{f_{j+1}^t - f_{j-1}^t}{2\Delta x} = 0, \qquad f_j^* = \left(f_{j+1}^t + f_{j-1}^t\right)/2$$

which has the advantage of working with both positive and negative advection velocities. Start with a an analysis of the numerical stability and dispersion of short wavelength oscillations. Implement the scheme in the JBONE applet and analyze the propagation of a square box. Switch to a short wavelength harmonic function resolved with 8 mesh points per wavelength and verify quantitatively that the numerical decay rate and the phase velocity are in agreement with your findings from the numerical dispersion analysis.

2.08 Electromagnetic waves. Consider first a one dimensional cavity $x \in [0; L]$ surrounded by an absorbing boundary condition (ABC) on the left and perfect electrical conductor (PEC) on the right



An antenna located at $x_{ant} = n_a\Delta x$ is switched on at time $t = 0$, with a Gaussian perturbation of the electric $E_{z,ant} = \exp[-(t - x_a/c)^2/t_w^2]$ and the magnetic field $H_{y,ant} = -E_{z,ant}/Z$, where $Z = \sqrt{\mu/\epsilon}$ is the impedance and $c = 1/\sqrt{\epsilon\mu}$ the speed of propagation in the medium. This generates an electromagnetic pulse with a polarization satisfying the Maxwell equations

$$\begin{cases} \frac{\partial H_y}{\partial t} &= \frac{1}{\mu}\frac{\partial E_z}{\partial x} - \frac{\rho}{\mu}H_y \\ \frac{\partial E_z}{\partial t} &= \frac{1}{\epsilon}\frac{\partial H_y}{\partial x} - \frac{\sigma}{\epsilon}E_z \end{cases}$$

which can be discretized using a so-called Yee scheme and is equivalent to the leap-frog algorithm using staggered grids for $\vec{E}$ and $\vec{B}$. After each update, the antenna source

is superposed to the total field with

$$E_z(x_{ant}, t+\Delta t) = E_z(x_{ant}, t+\Delta t) - CH_y(x_{ant}, t+\Delta t), \qquad C = \frac{\Delta t/(\epsilon \Delta x)}{1+\sigma \Delta t/(2\epsilon)}$$
$$H_y(x_{ant}, t+\Delta t) = H_y(x_{ant}, t+\Delta t) - DE_z(x_{ant}, t+\Delta t), \qquad D = \frac{\Delta t/(\mu \Delta x)}{1+\rho \Delta t/(2\mu)}$$

Account for the PEC setting $E_z(L, t+\Delta t) = 0$ and use a Mur first order ABC

$$E_z(0, t+\Delta t) = E_z(\Delta x, t-\Delta t) + \frac{c\Delta t - \Delta x}{c\Delta t + \Delta x}\left[E_z(\Delta x, t+\Delta t) + E_z(0, t-\Delta t)\right] + \frac{2\Delta x}{c\Delta t + \Delta x}\left[E_z(0,t) + E_z(\Delta x, t)\right]$$

Choose units where the vacuum permittivity $\epsilon_0 = 1$, the permeability $\mu_0 = 1$, the conductivity $\sigma = 0$, the magnetic resistivity $\rho = 0$ and use the `PhysData` parameters in JBONE to define the antenna position n_a and the relative permittivity $\epsilon_r = \epsilon/\epsilon_0$. Verify that the wave is properly reflected on the right and absorbed on the left.

Having completed, tested and documented the first part of this exercise, introduce a dielectric material $\epsilon_r = 4$ into the right half of the cavity

```
ABC          vacuum                        material    (PML) PEC
 |--------+--------------------|====================xxxxxxxx|
 0        xa                  L/2                           L
```

Verify the reflection $R = \frac{\sqrt{\epsilon_r}-1}{\sqrt{\epsilon_r}+1}$ and transmission $T = \frac{2}{\sqrt{\epsilon_r}+1}$ expected theoretically for short wavelengths. Choose an absorbing material with a polynomial increase of the conductivity towards the right while keeping $\rho(x) = Z^2\sigma(x)$: this creates perfectly matched layers (PML), which can be used instead of an ABC to absorb the electromagnetic wave on the right, typically with 4-12 mesh points.

These problems can be edited / submitted for correction from your web browser by selecting *assignments* under the course main page.

2.8 Further reading

- **Eulerian schemes for advection-diffusion problems.**
  Numerical Recipes [25] §19.1–19.2, Boris [9]
- **Computational electromagnetics.**
  EMLIB homepage[12] [22], Jin [19], Taflove [30]
- **Linear solvers.**
  Software from `netlib`[13], `PetSc`[14] and Numerical Recipes[15] [25] §2.4, Dahlquist [10] §6.3, §11.1–11.4, Saad [27].
- **Option pricing.**
  Visit our own distance learning course[16], Björk [7], Duffie [11], Hull [17], Rebonato [26], Wilmott [33].

[12] http://emlib.jpl.nasa.gov/
[13] http://www.netlib.org
[14] http://www.mcs.anl.gov/petsc
[15] http://www.library.cornell.edu/nr/
[16] http://www.lifelong-learners.com/opt

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 ELEMENT METHOD

The finite elements (FEM) method is particularly useful when a robust approximation is sought to solve partial differential equations on an inhomogeneous mesh. Solid mathematical foundations and a great deal of generality allow for different implementations that are only sketched in the first section: an example given for the advection-diffusion equation, for example, shows how the Crank-Nicholson method described in the previous section is only a particular case of a finite elements scheme.

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

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

a mathematician involves first the so-called

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 is carried out using tools from the variational calculus.

Variational principle. 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

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

Integration by parts. If $\vec{w}$ is differentiable at least once in $\mathcal{W} \subset \mathcal{C}^1(\Omega)$, 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

$$\int_\Omega v\nabla^2 u \; dV = -\int_\Omega \nabla v \cdot \nabla u \; dV - \int_{\partial\Omega} v\nabla u \cdot d\vec{S} \qquad (3.1\#\text{eq.7})$$

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 involving non-orthogonal functions are often evaluated with a numerical quadrature that can be easily be handled 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 choices can also be made for the *test functions* $\vec{w}$: among the most popular are the *Galerkin* method, where test and basis functions are both chosen from the same sub-space $\overline{\mathcal{V}} \equiv \overline{\mathcal{W}}$ and the method of *collocation*, which consists in taking Dirac functions $\vec{w} \in \overline{\mathcal{W}} = \{\delta(\vec{x} - \vec{x_j})\}, j = 1, N$ that lead to point-wise evaluations of the integrand on the mesh $\{\vec{x_j}\}, j = 1, N$.

Boundary conditions. If *natural boundary conditions* are not already sufficient, *essential boundary conditions* have to be imposed either by choosing the functional space $\overline{\mathcal{V}}, \overline{\mathcal{W}}$ (e.g. let $\vec{e_1}$ and $\vec{e_N}$ overlap in a periodic domain $[x_1; x_n]$ as in the next section) or by replacing one or several equations of the linear system (e.g. use higher order finite differences to preserve the convergence rate as in exercise 3.02).

3.2 An engineer's formulation

After a section on what a physicist believes is a mathematician's view of the subject, it is time for an example. Using the format of a "recipe" that is applicable to a broad class of practical problems, we show here how the advection-diffusion equation (1.3.2#eq.2) is discretized using Galerkin linear finite elements (FEMs) and how it is implemented.

Derive a weak variational form. "Multiply" your equation by $\forall g \in \mathcal{C}^1(\Omega), \int_\Omega dx \; g^*(x)$ where the complex conjugate 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, so as to avoid having to use quadratic basis functions to discretize the second order diffusion operator

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

Assuming a periodic domain, the surface term has here been canceled, imposing *natural boundary conditions*.

Discretize time with a finite difference backward in time and using a partially implicit evaluation of the unknown $f = (1-\theta)f^t + \theta f^{t+\Delta t}$ where $\theta \in [1/2; 1]$

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

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

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

Discretize space using linear roof-tops and a Galerkin choice for the test functions

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

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

Note how the condition $\forall g \in \mathcal{C}^1([x_L; x_R])$ is here 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 then 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 are all written in terms of *inner products*, involving overlap integrals of the form $(e_i, ue_j') = \int_{x_L}^{x_R} dx \;\; u(x) e_i(x) e_j'(x)$. Reassembling them all in a matrix,

Write a linear system through which the unknown values from the next time step $f_j^{t+\Delta t}$ can implicitly be obtained from the current values f_j^t by solving a linear system

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

To relate the Galerkin linear FEM scheme (3.2#eq.5) with the code that has been implemented in the JBONE applet, it is necessary now to evaluate the overlap integrals. This is usually performed numerically using a quadrature. In the case of a homogeneous mesh $x_j = j\Delta x$, a constant advection $u(x) = u$ and diffusion coefficient $D(x) = D$, the coming section shows how explicit expressions can be obtained from the same rules to define directly the matrix elements.

3.3 Numerical quadrature

Unless the coefficients of the differential equation (here u and D) are singular and require a special treatment somewhere in the solution domain, the precision of the quadrature depends mainly on the choice of the basis and test functions: it is indeed sufficient to simply preserve the convergence rate that is guaranteed by the FEM discretization. Writing the numerical quadrature as a truncated sum

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

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

table 3.3#tab.1 shows how a judicious choice of the abscissas x_i and weights w_i yields an approximation of $\mathcal{O}(f^{(p)})$ that integrates exactly polynomials with a degree up to $(p-1)$.

quadrature	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 3.3#tab.1: Quadrature abscissas and weights for the interval $x_i \in [-1;1]$

For example, a superposition of $n = 4$ terms (where the integrand is evaluated on the specified location $y_i(x_i) \in [a;b]$ and weighted by the factor w_i) is required using a Gaussian quadrature, when four evaluations of the integrand in every mesh interval are used to compute exact overlap integrals involving the product of (two) cubic test and basis functions. In the case of linear roof-top functions, the mid-point and trapezoidal rules are both equally precise and require only one evaluation per interval; although this is slightly more expensive with one extra evaluation per interval, both rules can nicely be combined into a so-called *tunable integration* [8]

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

As will become clear below, this allows to gradually modify the piecewise linear FEM discretization (obtained with $p = 1/3$) into either an equivalent FD discretization (with $p = 1$) or a piecewise constant FEM approximation (with $p = 0$). Apart from the academic interest, this feature can be used to change the slope of the numerical convergence and even to stabilize an approximation that is marginally unstable.

Armed with a variety of quadrature rules, it is now possible to implement the linear FEM scheme sketched in sect.3.2 using a tunable integration (3.3#eq.3) to evaluate the matrix elements in (3.2#eq.5). Since roof-top functions reach only as far as their nearest neighbors, all the matrix elements $\mathbf{A_{ij}}$ with $|i-j| > 1$ vanish, except for the overlap at

the boundaries. Choosing a sequential numbering of the unknowns $\{f_j^{t+\Delta t}\}, j = 1, N$, this results in a tri-diagonal matrix, with two extra elements in the upper-right and lower left corner taking care of the periodicity. It is a good exercise for the reader to verify that all the matrix elements can be written in terms of the integrals

$$\int_{x_{j-1}}^{x_j} e_j e_j dx = \int_{x_j}^{x_{j+1}} e_j e_j dx = (1+p)\frac{\Delta x}{4}, \qquad \int_{x_j}^{x_{j+1}} e_j e_{j+1} dx = \int_{x_j}^{x_{j+1}} e_{j+1} e_j dx = (1-p)\frac{\Delta x}{4}$$
$$\int_{x_j}^{x_{j+1}} e_j e_j' dx = -\int_{x_j}^{x_{j+1}} e_j e_{j+1}' dx = -\frac{1}{2}, \qquad \int_{x_j}^{x_{j+1}} e_j' e_j' dx = -\int_{x_j}^{x_{j+1}} e_j' e_{j+1}' dx = \frac{1}{\Delta x} \tag{3.3\#eq.4}$$

provided that the advection and diffusion coefficients are constant across the intervals as will be assumed here for pedagogical reasons. In a real code constructed for an inhomogeneous medium, each formula is replaced by n function calls returning the value of the integrand (where $u(x_k)$, $D(x_k)$ are evaluated on quadrature points) and the weighted sum (3.3#eq.1) is simply accumulated in the corresponding matrix element.

3.4 Implementation and solution

Substituting the overlap integrals (3.3#eq.4) into (3.2#eq.5) yields the FEM scheme

$$\begin{pmatrix} (1-p)\frac{\Delta x}{4} - \theta\Delta t\frac{u}{2} - \theta\Delta t\frac{D}{\Delta x} \\ (1+p)\frac{2\Delta x}{4} + \theta\Delta t\frac{2D}{\Delta x} \\ (1-p)\frac{\Delta x}{4} + \theta\Delta t\frac{u}{2} - \theta\Delta t\frac{D}{\Delta x} \end{pmatrix}^T \cdot \begin{pmatrix} f_{i-1}^{t+\Delta t} \\ f_i^{t+\Delta t} \\ f_{i+1}^{t+\Delta t} \end{pmatrix} =$$
$$\begin{pmatrix} (1-p)\frac{\Delta x}{4} - (\theta-1)\Delta t\frac{u}{2} - (\theta-1)\Delta t\frac{D}{\Delta x} \\ (1+p)\frac{2\Delta x}{4} + (\theta-1)\Delta t\frac{2D}{\Delta x} \\ (1-p)\frac{\Delta x}{4} + (\theta-1)\Delta t\frac{u}{2} - (\theta-1)\Delta t\frac{D}{\Delta x} \end{pmatrix}^T \cdot \begin{pmatrix} f_{i-1}^{t} \\ f_i^{t} \\ f_{i+1}^{t} \end{pmatrix} \tag{3.4\#eq.1}$$

Following an initialization where the initial condition is discretized with trivial projection on piecewise linear roof-top functions, the advection-diffusion scheme is implemented as

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

Here again, the BandMatrix.solve3() method is used to compute a direct solution in $\mathcal{O}(N)$ operations, using an LU-factorization that will be examined with more details in the coming section. The entire scheme is illustrated on-line with the advection of a box function, calculated using a piecewise linear FEM discretization (with $p = 1/3$) and a partly implicit time integration (with $\theta = 0.55$).

Numerical experiments: tunable finite-elements

1. Vary the `TimeImplicit` parameter $\theta \in [\frac{1}{2}; 1]$ and the `TuneIntegr` parameter $p \in]0; 1]$ to determine which combination has the smallest numerical damping.
2. Repeat the study and try to minimize the phase errors.
3. Introduce a finite physical `Diffusion=1` and adjust the `TimeImplicit` and the `TuneIntegr` parameters to calculate a solution that is physically meaningful even with a large CFL number.

Having spent a considerable effort understanding the FEM formulation, isn't it frustrating to see how similar the code is to the implicit Crank-Nicholson FD scheme in sect.2.5? This should be the main argument for the community who still uses implicit finite difference schemes: with a little bit of thinking but the same computational cost, the finite element scheme is now considerably more flexibile: it is for example easy to densify the mesh[17], modify the partly 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 (3.4#eq.1) are indeed equivalent when using a homogeneous mesh $x_j = j\Delta x$, a time centered integration $\theta = \frac{1}{2}$ and a trapezoidal quadrature $p = 1$.

3.5 Linear solvers

Writing efficient solvers for linear systems is a complete chapter of numerical analysis and involves much more than what can be said in this section. As a user of software libraries such as Netlib [5] or PetSc [3], it is however sufficient to have only a rough idea of what type of solvers exist and how to used them.

Direct LU factorization. Remember that you should a never calculate a matrix inverse to solve a linear system; the solution can be calculated with only one third of the number of operations by decomposing the matrix into *Lower and Upper triangular* parts [3.9] with $\frac{2}{3}N^3$ arithmetic operations $(*, +)$

$$\mathbf{A} \cdot \mathbf{x} = (\mathbf{L} \cdot \mathbf{U}) \cdot \mathbf{x} = \mathbf{L}(\mathbf{U} \cdot \mathbf{x}) = \mathbf{b} \qquad (3.5\#\text{eq.1})$$

and later use a *forward* and *backward substitution* of the right hand side vector involving $2N^2$ operations

$$\begin{aligned} y_1 &= \frac{b_1}{L_{11}}; \quad y_i = \frac{1}{L_{ii}} \left(b_i - \sum_{j=1}^{i-1} L_{ij} y_i \right), \quad i = 2, ...N \\ x_N &= \frac{y_N}{U_{11}}; \quad x_i = \frac{1}{U_{ii}} \left(y_i - \sum_{j=i+1}^{N} U_{ij} x_j \right), \quad j = N-1, ...1 \end{aligned} \qquad (3.5\#\text{eq.2})$$

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

A particularly simple version has been implemented in JBONE for tri-diagonal matrices, where the solve3() method computes the solution directly 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 LU-factorization exist taking advantage of the specific pattern of the matrix $\mathbf{A}$; before you start writing your own, check Netlib and it is likely that you will find a routine tailored for your application.

When the matrix has more than three diagonals, it is important to allow for a *pivoting* during the factorization process – interchanging rows and columns in the matrix to avoid divisions by small numbers on the diagonal. For particularly large problems, note the possibility of storing the matrix on disk in a *frontal implementation* of the same algorithm.

When memory consumption, calculation time or parallelization become an issue in 2, 3 or even higher dimensions, it might be useful to consider *iterative solvers* as an alternative to the direct LU decomposition. The idea behind them is best understood from a *splitting* of the matrix into a sum[18] of a diagonal $\mathbf{D}$, strict lower $-\mathbf{E}$ and upper $-\mathbf{F}$ triangular parts

$$\mathbf{A} = \mathbf{D} - \mathbf{E} - \mathbf{F} \tag{3.5\#eq.3}$$

An iterative solution of the problem

$$(\mathbf{b} - \mathbf{A} \cdot \mathbf{x_{k+1}})_i = 0 \ , \quad i = 1, ...N \tag{3.5\#eq.4}$$

is then sought where $\mathbf{x_{k+1}} = \xi_i^{(k+1)}$ is the $(k+1)$-th approximation of the solution having components from $i = 1, ...N$.

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

$$a_{ii}\xi_i^{(k+1)} = \beta_i - \sum_{i \neq j} a_{ij}\xi_j^{(k)} \tag{3.5\#eq.5}$$

which is equivalent in matrix notation to

$$\mathbf{D} \cdot \mathbf{x_{k+1}} = (\mathbf{E} + \mathbf{F}) \cdot \mathbf{x_k} + \mathbf{b} \tag{3.5\#eq.6}$$

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 possible to immediately use the updated components $\xi_j^{(k+1)}$ using *forward Gauss-Seidel* iterations

$$a_{ii}\xi_i^{(k+1)} + \sum_{j<i} a_{ij}\xi_j^{(k+1)} = \beta_i - \sum_{j>i} a_{ij}\xi_j^{(k)} \tag{3.5\#eq.7}$$

or

$$(\mathbf{D} - \mathbf{E}) \cdot \mathbf{x_{k+1}} = \mathbf{F} \cdot \mathbf{x_k} + \mathbf{b} \tag{3.5\#eq.8}$$

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

$$(\mathbf{D} - \mathbf{F}) \cdot \mathbf{x_{k+1}} = \mathbf{E} \cdot \mathbf{x_k} + \mathbf{b} \tag{3.5\#eq.9}$$

[18] Don't get confused here with a product previously used in the LU factorization!

Successive over-relaxations – SOR are obtained when one of the Gauss-Seidel iterations (3.5#eq.8 or 3.5#eq.9) is linearly combined with the value of the previous step

$$\mathbf{x_{k+1}} = \omega \mathbf{x_k^{GS}} + (1-\omega)\mathbf{x_k} \qquad (3.5\#\text{eq.}10)$$

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

$$\begin{cases} (\mathbf{D} - \omega\mathbf{E}) \cdot \mathbf{x_{k+1/2}} = [\omega\mathbf{F} + (1-\omega)\mathbf{D}] \cdot \mathbf{x_k} + \omega\mathbf{b} \\ (\mathbf{D} - \omega\mathbf{F}) \cdot \mathbf{x_{k+1}} = [\omega\mathbf{E} + (1-\omega)\mathbf{D}] \cdot \mathbf{x_{k+1/2}} + \omega\mathbf{b} \end{cases} \qquad (3.5\#\text{eq.}11)$$

Preconditioning / 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 can be diagonalized approximatively using a preconditioning

$$\mathbf{M^{-1}} \cdot \mathbf{A} \cdot \mathbf{x} = \mathbf{M^{-1}} \cdot \mathbf{b} \qquad (3.5\#\text{eq.}12)$$

where the preconditioner matrix $\mathbf{M}$ is one of

$$\begin{aligned} \mathbf{M^{Jacobi}} &= \mathbf{D} & \mathbf{M^{SOR}} &= \frac{1}{\omega}(\mathbf{D} - \omega\mathbf{E}) \\ \mathbf{M^{GS}} &= \mathbf{D} - \mathbf{E} & \mathbf{M^{SSOR}} &= \frac{1}{\omega(2-\omega)}(\mathbf{D} - \omega\mathbf{E}) \cdot \mathbf{D^{-1}} \cdot (\mathbf{D} - \omega\mathbf{F}) \end{aligned} \qquad (3.5\#\text{eq.}13)$$

The product $\mathbf{s} = \mathbf{M^{-1}} \cdot \mathbf{A} \cdot \mathbf{x}$ is calculated without ever forming the inverse, calculating first $\mathbf{r} = \mathbf{A} \cdot \mathbf{x}$ and then solving the linear system $\mathbf{M} \cdot \mathbf{s} = \mathbf{r}$.

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 large problems that have to be solved many times. Hyperbolic (wave-) problems and non-Hermitian operators are more difficult to solve iteratively and remains a matter of research [19],[18]. Efficient methods in general rely on a suitable *preconditioning* and the rapid convergence of Krylov or Lanczos methods, such as the *generalized minimal residuals (GMRES)* and the *quasi-minimal residuals (QMR)* [3.9].

Note that the matrix-vector multiplications that are used to generate new iterates can be performed in *sparse format*, i.e. using only those matrix elements that are different from zero. This is the reason why iterative methods can potentially be much more economical than direct solvers – if they converge.

3.6 Variational inequalities

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

Mathematically, the obstacle problem amounts to finding a solution $S(x)$ satisfying 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).

This section considers two methods for solving problems that involve inequalities.

Linear complementary formulation. Reassemble all the constraints in a form

$$\mathcal{A}\mathcal{B} = 0, \qquad \mathcal{A} \geq 0, \qquad \mathcal{B} \geq 0 \tag{3.6\#eq.1}$$

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

$$(\mathbf{x} - \mathbf{c})(\mathbf{A}\mathbf{x} - \mathbf{b}) = 0, \qquad \mathbf{A}\mathbf{x} \geq \mathbf{b}, \qquad \mathbf{x} \geq \mathbf{c} \tag{3.6\#eq.2}$$

can be solved with the so-called *projected SOR* method, by replacing (3.5#eq.10) with

$$\mathbf{x_{k+1}} = \max[\mathbf{c}, \omega \mathbf{x_k^{GS}} + (1-\omega)\mathbf{x_k}] \tag{3.6\#eq.3}$$

and making sure that the iteration starts 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 $S - O = 0$; this leads to a complementary problem

$$S''(S - O) = 0 \qquad S - O \geq 0 \qquad -S'' \geq 0 \tag{3.6\#eq.4}$$

which be discretized using finite differences and solved with the projected SOR method (exercise 3.05).

Variational formulation. This approach is particularly 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 constraints 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 subtract them

$$\int_{x_L}^{x_R} -S''(w - S) \geq 0 \tag{3.6\#eq.5}$$

The constraint c now appears only implicitly through the fact that w and S are members of the same sub-space $\mathcal{V}$; all what remains to be done is to prove that the inequality (3.6#eq.5) in fact is a strict equality if the solution satisfies the constraint. After integration by parts and discretization with linear FEMs, the problem can therefore be solved with projected SOR iterations (3.6#eq.3, exercises 3.05 and 3.07).

To prove that (3.6#eq.5) satisfies a strict equality, integrate by part and a discretize $S(x) = \sum_j s_j e_j(x)$ and $w(x) = \sum_j w_j e_j(x)$ with linear finite elements

$$\int dx \left(\sum_i s_i e_i' \right) \left(\sum_j (w_j - s_j) e_j' \right) = \sum_i \sum_j s_i \underbrace{\int dx \; e_i' e_j'}_{\mathbf{M}} (w_j - s_j) \geq 0$$

If the matrix is symmetric $\mathbf{M^T} = \mathbf{M}$ as is the case with diffusion, this simply yields

$$\mathbf{s^T M (w - s)} = \mathbf{(w - s)^T M^T s} = \mathbf{(w - s)^T M s} \geq 0 \qquad (3.6\#\text{eq.6})$$

If the problem is positive definite $\sum_j M_{ij} s_j \geq 0$ as is the case with diffusion and the solution satisfies the constraint $s_i - c_i \geq 0$, the product satisfies the inequality

$$\mathbf{(s - c)^T M s} \geq 0 \qquad (3.6\#\text{eq.7})$$

Since (3.6#eq.6) must hold for the particular choice where the test function coincides with the constraint $\mathbf{w} = \mathbf{c}$, the last two inequalities can only be satisfied simultaneously if they both satisfy the strict equality. □

3.7 Computer quiz

1. Discretizing a weak form with the method of collocations, you use
   (a) 1 evaluations of the PDE coefficients per mesh interval
   (b) 2 evaluations of the PDE coefficients per mesh interval
   (c) a number of evaluations depending on the order of the test function

2. Integration by part of second order differential operators
   (a) is impossible in a cubic Galerkin FEM approximation
   (b) can be used to impose natural boundary conditions
   (c) is mandatory in a cubic Galerkin FEM approximation

3. With linear Galerkin FEM and constant coefficients, exact overlap integrals are
   (a) obtained with 1 point quadratures at the cell boundary (trapezoidal rule)
   (b) obtained with 1 point quadratures in the cell center (mid-point rule)
   (c) obtained with 2 points Gaussian quadratures
   (d) impossible to obtain with a numerical quadrature

4. Rising the implicit time parameter $\theta \in [0.5; 1]$
   (a) increases the numerical accuracy
   (b) increases the numerical damping
   (c) smoothes the physical solution
   (d) none of the above

5. Rising the tunable integration parameter $p \in [0; 1]$
   (a) changes the numerical accuracy
   (b) changes the numerical convergence
   (c) smoothes the physical solution
   (d) none of the above

6. For an efficient solution of a linear Galerkin FEM solution $\mathbf{x} = \mathbf{A^{-1}b}$ or rank n you
   (a) calculate the inverse matrix and multiply in sparse format
   (b) perform an LU-factorization and substitute Ly=b, Ux=y
   (c) perform an LU-factorization and substitute Uy=b, Lx=y
   (d) use a direct solver in 1D, satisfied with a linear scaling in the number of operation
   (e) use an iterative solver in 2D, satisfied with a cubic scaling in operations

3.8 Exercises

3.01 Quadrature. Approximate the integral $\int_0^\pi \sin(x)dx$ with numerical quadratures in two intervals. Compare the analytical result with the values obtained from different tunable integration schemes (3.3#eq.3) and a two point Gaussian quadrature (3.3#eq.1) for which $m = 2$ in table 3.3#tab.1.

3.02 Diffusion in a cylinder. Use a linear Galerkin 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*

$$\frac{\partial T}{\partial t} - \kappa \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T}{\partial r}\right) = 0 \qquad (3.8\#\text{eq.1})$$

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 a trapezoidal quadrature and form a linear system of equations that can be implemented in the JBONE applet. Combine Taylor series at the edge of the numerical domain to obtain a second order accurate finite difference formula to impose a Neuman boundary conditions on the cylinder surface $r = r_c$.

3.03 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.

3.04 Iterative solver. Implement your own iterative–SOR solver in JBONE and study the number of iterations that are required to achieve a 10^{-6} precision with different values of the advection and diffusion coefficients.

3.05 Obstacle problem. Compute a numerical solution for the obstacle problem outlined in sect.3.6. For simplicity, take an obstacle parametrized by the constraint $C(x) = \frac{1}{2} - x^2$ and an elastic string $S(x)$ attached to the edge of the solution interval $S(-1) = S(1) = 0$. *Hint: you may use the projected* `BandMatrix.ssor3()` *solver that is documented in the JBONE tree.*

3.06 Numerical dispersion. Use a local ansatz $f(x,t) \sim \exp(-i\omega t)\sum_j \exp(ikx_j)e_j(x)$ to study how a linear Galerkin FEM discretization of the spatial operator 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.

3.07 American option. Extending your knowledge from exercise 2.05, use a variational formulation with linear Galerkin FEM 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:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r - D_0)S\frac{\partial V}{\partial S} - rV \geq 0$$

Restrict the test function $\phi \in \mathcal{W}$ to a piecewise linear function that remains larger than the terminal payoff $\phi(S,t) \geq \max(E-S,0)$ for all times. Replying on the favorable stability properties of implicit schemes, it is not in fact necessary to make the transformation into log-normal variables. Implement the scheme in JBONE and compare with the solution previously obtained for the *European put* in exercise 2.05.

These problems can be edited / submitted for correction from your web browser by selecting *assignments* under the course main page.

3.9 Further Reading

- **Finite elements.**
  Johnson [20], Fletcher [14], Saad [27] §2.3
- **Quadrature.**
  Abramowitz [1] §25.4.29 with table 25.4, Numerical Recipes [25] §4.5
- **Linear solvers.**
  Software from `netlib`[19], `PetSc`[20], and Numerical Recipes [25] §2.4,
  Dahlquist [10] §6, Saad [27],

3.10 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.

[19] http://www.netlib.org
[20] http://www.mcs.anl.gov/petsc

4 FOURIER TRANSFORM

As mentioned earlier in sect.1.4, it is largely thanks to the possibility of computing efficiently the Fourier transformation with $\mathcal{O}(N \log N)$ operations that *fast Fourier transforms (FFT)* can be considered as a viable alternative to solve partial differential equations.

4.1 FFT with the computer

In the same manner as for the linear solvers, it is sufficient to have a rough idea of the underlying process to efficiently use the FFT routines from software libraries. In fact, you should always privilege vendors implementations, since these have in general been optimized for the specific computer that you are using. Having no library available in JAVA, a couple of routines from Numerical Recipes [25] have here been translated for JBONE and put into the FFT.java class – making as little modification as possible to the original code so as to let you learn more from the book. The Complex.java class has been included to illustrate how rewarding it is to work with Netlib [5] software.

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 has been 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 mesh therefore relies on the command

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

   where `FFT.inXSpace` sets the original location of the variable `myFTObject` in configuration space.

2. To avoid negative indices for periodic functions $f(x+L) = f(x)$, 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 $f(x)$ is real, since $\sin(2\pi x/2)$ is sampled exactly for 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 [25] and indeed very similar to that of many computer vendors.

4.2 Linear equations.

Albeit slower, with $\mathcal{O}(N \log N)$ operations per time-step, than the numerical schemes in $\mathcal{O}(N)$ we have seen so far, FFTs 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

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

The equation can be integrated analytically without any restriction on the time-step; starting directly from the initial condition, this yields for each 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 the 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))); //  condition

  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 read carefully, you probably wonder now about the sign of the phase factor, which is opposite from (4.2#eq.2); the reason is that the phase chosen here for the spatial harmonics is exactly opposite to the one that has been assumed in the FFT routine from Numerical Recipes [25]. This makes the scheme look like as if it evolves backwards in time – it doesn't! Also note that once the initial condition has been transformed to K-space, subsequent transformations back to X-space are in fact only required for plotting.

The document on-line illustrates the advection of a box calculated with the same time step $\Delta t = 0.5$ as previously. The *Gibbs phenomenon* (an artifact from using harmonic functions for the discretization of sharp edges discussed in sect.1.4.5) is clearly visible except when $u\Delta t/\Delta x$ is an integer; but don't be fooled by the plot routine, which (for simplicity, but still incorrectly) draws straight lines between the grid points when it should in fact draw the complete oscillations that are visible in figure 1.4.5#fig.1. The power of the method is evident when taking large time-steps: edit the parameter to `TimeStep=64`, add some diffusion `Diffusion=0.3` and compare the solution obtained with a single step with the result computed using your favorite FD or FEM scheme from the previous sections[21]. This is very nice indeed, but remember that dealing with an inhomogeneous medium $u(x)$, $D(x)$ or complicated boundary conditions is problematic in Fourier space.

[21] Don't forget to reduce the time-step depending on the method!

Numerical experiments: linear equations with FFT

1. Reduce `TimeStep=0.1` and observe how the Gibbs phenomenon appears when the position of the box does not exactly coincide with the mesh points chosen for the plot.

2. Switch to `Finite Differences` and `Finite Elements` to measure the quality of previous schemes for `Velocity=1`, `Diffusion=0.5` by comparing the results with the exact solution obtained with the `Fourier transform`. Hint: re-select the method that is currently being used to re-scale the plot window and monitor `Min / Max` at the end of the evolution.

3. 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* the function over a period L with a finite number of values $N = L/\Delta$, where Δ is the size of the sampling interval, the spectrum gets truncated at the shortest wavelength $\lambda_c = 2\Delta = 2\pi/k_c$ called *Nyquist critical wavelength*: this corresponds to exactly 2 mesh points per wavelength, which does however not mean that shorter wavelengths $|k| > k_c$ do not contribute to the Fourier coefficients (4.1#eq.1). Figure 4.3#fig.1 illustrates how they get *aliased* back into the lower components of the spectrum. This can be important for the digital data acquisition

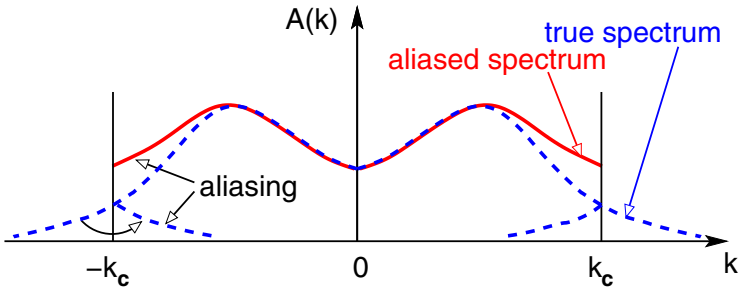


Figure 4.3#fig.1: Aliasing from Fourier components shorter than the Nyquist critical wavelength $|k| > k_c$.

of an experiment, where the signal has to be *low-pass filtered* before it is digitally sampled. 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 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,

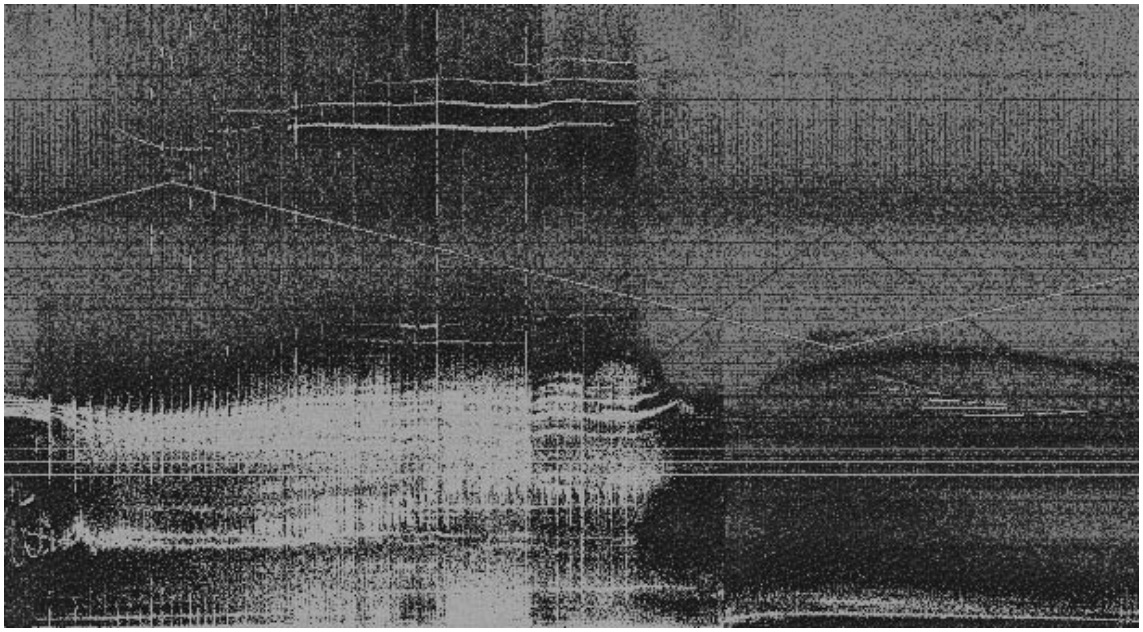


Figure 4.3#fig.2: Experimental spectrum 0-500 kHz digitally recorded during 2 sec from the magnetic perturbations in a fusion plasma in the Joint European Torus. Apart from the Alfvén instabilities which are the subject of this research, you can see the sawteeth-like trace of an exciter antenna reaching 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).

- 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 right from the beginning when the initial condition is first discretized (try to initialize and propagate an aliased cosine with `ICWavelength=1.05` mesh points per wavelength 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). Indeed, think of the *convolution theorem*, which tells 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 incorrectly "pollute" a long wavelength channel 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 the representation to calculate the multiplication of arrays twice the original 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 here of course be formally possible to write the non-linear term as a convolution in K-space, but it is much easier to write and more 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. 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 mentioned 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 [25].

```
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 either calculated without precaution (subject to aliasing) or by temporarily expanding the spectrum by padding the upper part with zeros (cures 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.

Numerical experiments: non-linear equations with FFT

1. Change the switch to `Aliased Convolution` and observe how aliasing pollutes the spectrum with short wavelengths that evolve into a non-linear instability.
2. 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 longer wavelengths (physical) by reducing the `ICAmplitude`. Is such a solution attractive?

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.

4.5 Computer quiz

1. For an efficient FFT, the size of the arrays to be transformed must be
   (a) a multiple of 2
   (b) a power of 2
   (c) a prime number

2. The linear solution computed for the advection-diffusion equation
   (a) converges linearly to the physical value
   (b) converges quadratically to the physical value
   (c) converges faster than any polynomial to the physical value
   (d) is exact

3. Compared with Fourier schemes, aliases in finite differences are
   (a) never a problem (they do not exist)
   (b) rarely a problem (short wavelengths numerically damped by spatial operators)
   (c) as much a problem

4. Compared with Burger's, the Korteweg-DeVries equation is
   (a) less sensitive to aliasing
   (b) equally sensitive to aliasing
   (c) more sensitive to aliasing

4.6 Exercises

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

4.02 Equivalent filter for Zabusky's FD scheme. Study Zabusky's FD scheme for the Korteweg-DeVries equation

$$\frac{f_j^{t+\Delta t}-f_j^{t-\Delta t}}{2\Delta t}+\frac{1}{3}\left[f_{j+1}^t+f_j^t+f_{j-1}^t\right]\frac{f_{j+1}^t-f_{j-1}^t}{2\Delta x}+b\frac{f_{j+2}^t-2f_{j+1}^t+2f_{j-1}^t-f_{j-2}^t}{2\Delta x^3}=0 \qquad (4.6\#\text{eq.1})$$

in the JBONE applet with a dispersion $b = 1/2$. Use an ansatz $f \sim \exp(ikx)$ to calculate the equivalent filter in Fourier space, which is implied when the linear spatial differences are calculated in configuration space. Add this filter to the Fourier scheme in JBONE and test if, after applying the filter, the FD and FFT methods become similar. What remains different? If you want, you can repeat your analysis using a filter for the non-linear term. *Hint: Zabusky's scheme has already been implemented under this link and can be executed after choosing* `Finite differences`, `Explicit 3-level` *and* `KdV (solitons)` *in the applet selectors. Your task is to calculate and to implement a spatial filter function $\mathcal{H}(k\Delta x)$ that emulates in Fourier space the evolution of the Fourier components $f_k^{t+\Delta t} = \mathcal{H} f_k^t$ in manner equivalent to what happens in configuration space using finite differences. Is the linear filter sufficient to cure the problem with aliasing?*

4.03 Prototype problems. Modify the parameters of the non-linear equation 4.4#eq.3 in Fourier space to develop a 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 that you believe is particularly interesting.

4.04 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 $D_{\rm phys}$. Alternatively, you may compare the numerical solution with the analytical value. *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. You may also simply click in the plot area for a rougher estimate.*

These problems can be edited / submitted for correction from your web browser by selecting *assignments* under the course main page.

4.7 Further Reading

- **FFT algorithm.**
  Numerical Recipes [25] §12

4.8 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.

5 MONTE-CARLO METHOD

With the name of a city famous for casinos and gambling, the Monte Carlo (MC) method owes its name to the random numbers that are used to describe possible outcomes of particle trajectories. Contrary to *fluid methods*, which describe local properties with differential changes approximated on a mesh, *particle methods* use the evolution of a large number of samplers to relate the motion at the microscopic scale with statistical changes at the macroscopic scale. The random nature of the method draws on results from the stochastic calculus that will be quickly summarized first for readers who are not at all familiar with this topic.

5.1 Monte Carlo integration

The most common use of Monte Carlo methods is the evaluation of multi-dimensional integrals [25]. Consider first the approximation of an integral obtained from the trapezoidal rule (3.3#eq.1)

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

Instead of a uniform sampling, imagine an evaluation where the positions $\{x_i\}$ are random numbers uniformly distributed in the interval $[a,b]$; this yields a 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 convergence rate is of course lower than from a uniform sampling (5.1#eq.1); the strength however appears for integrals in higher dimensions $d > 2$, where the Monte Carlo error scales as $N^{-1/2}$ irrespective of the number of dimensions – instead of the scaling in $N^{-1/d}$ that is achieved when using a uniform mesh.

5.2 Stochastic theory

This section is intended as a very short introduction into the stochastic theory, which provides the mathematical foundation behind the Monte Carlo method; check Sczepessy [13], Kloeden [23] and Van Kampen [21] for complete courses on the subject.

To begin, remember the definition of an *expected or mean value* $\mathcal{E}$ and the *variance* $\mathcal{V}$

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

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

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

where $f_X(x)$ is the *density distribution function* of a stochastic variable X. The notation $\mathcal{N}(\mu,\sigma)$ refers to the set of *Gaussian* or *normally distributed* stochastic variable X with a 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. Along the same lines, $\mathcal{U}(a,b)$ stands for the set of *uniformly distributed* random numbers in the interval $[a,b]$.

Now consider a class of stochastic processes that play a central role in the Monte Carlo method: W_t is called *a Wiener process* (or in an equivalent manner *Brownian motion*) at a 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.

It turns out that the distribution of Wiener increments $W_{t+\Delta t} - W_t$ 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* (or *Itô calculus*) involves new properties that are fundamentally different from the ordinary (or Riemann) calculus. The reason can be tracked down to the preferred direction of time t in the *Itô integral*, which is defined as the limit of the explicit (forward Euler) discretization

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

for any sequence $\{t_i : t_i \in [0,T], t_{i+1} = t_i + \Delta t_i\}$. Note that the circle $\circ$ here stands for Itô differential and states that dW_t is independent of $b(W_t,t)$; also, an implicit discretization in (5.2#eq.5) would give a fundamentally different result.

Define a property Y_t that is a sum of a Riemann and an Itô integral

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

For an infinitely short time interval, this leads to the *stochastic differential equation*

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

which reduces to an ordinary differential equation in the absence of the stochastic component

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

Stochastic differential equations such as (5.2#eq.7) are used in the Monte-Carlo method to describe the trajectories or orbits of particles.

5.3 Particle orbits

In the Monte Carlo method, the initial condition or indeed any function is discretized using quasi-particles; differential operators are therefore reformulated in terms of particle motions. Remember that a quasi-particle is defined in (1.4.7#eq.1) 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 in the form of Dirac pulses $S_i(x) = \delta(x_i)$. The solution can then be calculated from the particles density obtained from the positions $\{x_i(t)\}$ called *particle orbits*. By construction, the lowest order moment is perfectly conserved, since no particle is lost.

To show the contrast between the *Taylor* and the *Feynman-Kac* theorems, which provide a rigorous foundation for the Monte Carlo method, the *deterministic* and the *stochastic motion* are first considered separately.

Deterministic orbits satisfy 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 is solved numerically using the methods discussed in section 1.2.1, using an explicit or implicit discretization of time

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

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

For an ensemble of N particles with orbits $x_i(t)$, the particle density distribution function $f(x, t)$ is obtained by superposition

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

and is plotted in the applet on-line using the projection (1.4.7#eq.4) from section 1.4.

Numerical experiments: deterministic orbits

1. Study in slow motion how the density function is obtained from a linear assignment on the coarse grid.
2. Increase the number of `Walkers=100`, press `Initialize` and `Start/Stop` to verify that the lowest order moment is exactly conserved for all times. Increase the number of `MeshPoints=64` and discuss whether the solution gets more precise.
3. Increase to `TimeStep=1` or any signed integer; verify that the deterministic motion of a cloud of particles provides in fact an exact description of advection once the initial condition has been discretized.

If a large number of particles approximate a smooth density function and each individual orbit $X(t)$ evolves in a deterministic manner

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

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

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

The theorem makes it possible to study a PDE instead of an N-particle system; the reverse is also true and will be used to solve the advection equation with particles.

Stochastic particle orbits are defined as an ensemble of possible orbits with different probabilities. Imagine a snowflake falling slowly from the sky: the motion is unpredictable and each evolution can be described with an orbit $X(t)$ following the stochastic differential equation

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

where W_t is a Wiener processes (Brownian motion). Averaging over a large number of orbits, the average velocity and the broadening of possible orbits are readily obtained (exercise 5.02) as

$$\frac{\partial}{\partial t}\mathcal{E}[X(t)](x) = v(x,t) \tag{5.3\#eq.8}$$

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

Stochastic orbits can therefore be parametrized with statistical averages

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

and lead to a different result each time the JBONE applet is executed.

Numerical experiments: stochastic orbits

1. Take a few single `Step 1` and convince yourself that the direction where the particle moves does not depend on the past. Re-`Initialize` and verify that the same stochastic differential equation each time produces a different orbit.
2. Increase the number of `Walkers=100` and observe how the cloud of independent particles diffuses in time.
3. Vary the `Velocity` and `Diffusion` parameters, which are linked to the average velocity $v(x,t)$ and the broadening $b(x,t)$ of the orbits.

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; the so-called **Feynman-Kac theorem** tells that a smooth density distribution function $f(x,t)$ in which individual particles move according to

$$dX(t) = v(X(t),t)dt + b(X(t),t) \circ dW_t \tag{5.3\#eq.11}$$

evolves according to the *Fokker-Planck equation*

$$\frac{\partial f}{\partial t} = -\frac{\partial}{\partial x}(vf) + \frac{\partial^2}{\partial x^2}\left(\frac{b^2}{2}f\right) \tag{5.3\#eq.12}$$

which is in fact an advection-diffusion equation.

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

$$\frac{\partial f(x,t)}{\partial t} + \frac{\partial}{\partial x}\left[a(x)f(x,t)\right] = \frac{\partial}{\partial x}\left(D(x)\frac{\partial f}{\partial x}\right), \quad x \in (-\infty,\infty) \tag{5.3\#eq.13}$$

From the Feynman-Kac theorem, it is easy to show (exercise 5.01) that $X(t)$ satisfies the stochastic differential equation

$$\frac{\partial}{\partial t}\mathcal{E}[X(t)](x) = a(X(t)) + \left(\frac{\partial}{\partial x}D(x)\right)_{x=X(t)} \tag{5.3\#eq.14a}$$

$$\frac{\partial}{\partial t}\mathcal{V}[X(t)](x) = 2D(X(t)) \tag{5.3\#eq.14b}$$

showing exactly what evolution is required to reproduce statistically the properties desired at the macroscopic scale.

5.4 A scheme for the advection diffusion equation

Let us use now the Feynman-Kac theorem and evolve a large number of stochastic particle orbits to approximate the advection diffusion equation

$$\frac{\partial f}{\partial t} + \frac{\partial}{\partial x}(vf) - \frac{\partial^2}{\partial x^2}\left(\frac{b^2}{2}f\right) = 0 \qquad (5.4\#\text{eq.1})$$

The rules of the Itô calculus require the time discretization to be explicit

$$X_{t+\Delta t} = X_t + v(X_t,t)\Delta t + b(X_t,t)(W_{t+\Delta t} - W_t) \qquad (5.4\#\text{eq.2})$$

Note that it is in fact possible to construct implicit discretizations of a stochastic differential equation [23], but the construction it not as simple as it is for ordinary equations (5.3#eq.3). Since $(W_{t+\Delta t} - W_t) \in \mathcal{N}\left(0, \sqrt{\Delta t}\right)$, the Wiener process can be written explicitly

$$W_{t+\Delta t} - W_t = \zeta\sqrt{\Delta t} \qquad (5.4\#\text{eq.3})$$

where $\zeta \in \mathcal{N}(0,1)$ are normally distributed random numbers. The central limit theorem states that any large sum involving n random numbers equally distributed with zero mean and unit variance will eventually converge to $\mathcal{N}(0,\sqrt{n})$. Any such random number could therefore be used for ζ provided that the number of time steps used to approximate the differential is large enough; choosing $\zeta \in \mathcal{N}(0,1)$ however leads to the fastest convergence.

A numerical Monte Carlo scheme for the advection-diffusion equation is now constructed, using N particles to sample the possible outcomes of the entire ensemble of orbits

$$X_{t+\Delta t} = X_t + v(X_t,t)\Delta t + \zeta b(X_t,t)\sqrt{\Delta t} \qquad (5.4\#\text{eq.4})$$

This is simply be implemented as

```
for(int j = 0; j < numberOfParticles; j++){
  particlePosition[j] += velocity * timeStep +
    random.nextGaussian() *
    Math.sqrt(2 * diffusCo * timeStep);
  // Insert here your periodic boundary conditions (exercise 5.03)
}
```

and can be tested in the JBONE applet on-line

Numerical experiments: Monte Carlo scheme for advection-diffusion

1. Compare the solution obtained with 128 steps of duration `TimeStep=0.5` with the one achieved with a single step of duration `TimeStep=64`; which is faster and which is more precise?
2. Observe how the periodic boundary conditions have been implemented so that particles can take steps that are larger than the domain size (exercise 5.03).

JAVA is one of the few programming languages that has a normal pseudo random numbers generator $\mathcal{N}(0,1)$. Other programming languages generally provide only uniform pseudo random numbers $\mathcal{U}(0,1)$ and the **Box Müller method** can then be used to transfrom them into $\mathcal{N}(0,1)$ with the algorithm:

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

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

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

The rest of the implementation is equivalent if one remembers the definition of the command `x+=dx` as being equivalent to `x=x+dx`.

5.5 When should you use Monte Carlo methods?

Particle methods generally are efficient for problems dealing with a large number of dimensions and complex geometries: table 5.5#tab.1 suggests that Monte Carlo methods become more efficient than FD or FEM when $d/n > 2$ (in the limit of $N \to \infty$ and neglecting computer hardware issues). The MC method is also well suited for problems with complicated

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 5.5#tab.1: Accuracy for a problem with d-dimensions.

boundary conditions. Imagine for example a 3D cube with a bouncing ball 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 the MC method, but nearly untraceable with a fluid method.

Parallelization is easy and efficient if the problem is linear and the particles are decoupled: just run a copy of the simulation program on several machines simultaneously. The final result is obtained 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#eq.4). Be careful, however, to seed the random number generators differently on the each machines, or the simulations will all be identical!

Unfortunately, the MC method is 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 Computer quiz

1. If one single integration with 100 samples falls within 1% of the analytical value
   (a) 10000 samples yield an error smaller than 0.01%
   (b) 10000 samples yield an error around 0.1%
   (c) 10000 samples yield an error that is likely to be larger than 0.1%

2. In a Wiener process, the increment immediately following an increment of 0.23 is
   (a) likely to be negative rather than positive
   (b) uniformly distributed in [-1;1]
   (c) uniformly distributed in [0;1]
   (d) independent of the past realisation

3. In a Brownian motion, the increment (mean free path) is
   (a) propotional to the square-root of the diffusion coefficient
   (b) propotional to the diffusion coefficient
   (c) independent of the diffusion coefficient

4. The numerical error in our MCM scheme for advection-diffusion is
   (a) proportional to the square-root of the time step
   (b) proportional to the time step
   (c) independent of the time step

5. Monte Carlo discretization of non-linear equations such as Burger's is
   (a) impossible
   (b) particularly slow
   (c) difficult to parallelize

5.7 Exercises

5.01 Expected value and variance. Calculate the statistical parameters $\frac{\partial}{\partial t}\mathcal{E}[X(t)](x)$ and $\frac{\partial}{\partial t}\mathcal{V}[X(t)](x)$, in the stochastic orbits $X(t)$, where the density distribution function $f(x,t)$ of a particle is given by

$$\frac{\partial f(x,t)}{\partial t} + \frac{\partial}{\partial x}\left[a(x)f(x,t)\right] = \frac{\partial}{\partial x}\left(D(x)\frac{\partial f}{\partial x}\right), \quad x \in (-\infty,\infty) \tag{5.7\#eq.1}$$

Take the initial condition $X(0) = x_i$, i.e. the density distribution is

$$f(x,0) = \delta(x - x_i) \tag{5.7\#eq.2}$$

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

$$\frac{\partial}{\partial t}\mathcal{E}[X(t)](x) = \frac{\partial}{\partial t}\int_{-\infty}^{\infty} f(x,t)x\,dx = \int_{-\infty}^{\infty}\frac{\partial f(x,t)}{\partial t}x\,dx \tag{5.7\#eq.3}$$

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

5.02 Diffusion statistics. Use the stochastic equation (5.3#eq.7) and the definitions

$$\begin{aligned}\frac{\partial}{\partial t}\mathcal{E}[X(t)](x) &= \lim_{dt\to 0}\frac{1}{dt}\mathcal{E}[dX(t)]\\ \frac{\partial}{\partial t}\mathcal{V}[X(t)](x) &= \lim_{dt\to 0}\frac{1}{dt}\mathcal{E}[dX(t)^2]\end{aligned}$$

to calculate the average particle velocity (5.3#eq.8) and the broadening of the distribution of possible orbits (5.3#eq.9). Combine this with the results from exercise 5.01, and show that the advection-diffusion equation (5.3#eq.13) is related to the Fokker-Planck equation (5.3#eq.12).

Comment: together with exercise 5.01, you have now 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.12). If you would extend this to include all moments, you would in fact prove the Feynman-Kac theorem.

5.03 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. Note that* `lim[1]` *includes the extra mesh cell that connects consecutive domains and is not plotted.*

5.04 Steady state with velocity gradient. Simulate the equation

$$\frac{\partial f}{\partial t} = -\frac{\partial}{\partial x}\left(\frac{(x_0 - x)}{s}f\right) + D\frac{\partial^2 f}{\partial x^2} \qquad (5.7\#\text{eq.}4)$$

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

5.05 Diffusion coefficient gradient. Simulate the equation

$$\frac{\partial f}{\partial t} = -u\frac{\partial f}{\partial x} + \frac{\partial}{\partial x}\left(D\left[\frac{1}{4} - \left(\frac{x - (x_\mathrm{R} + x_\mathrm{L})/2}{x_\mathrm{R} - x_\mathrm{L}}\right)^2\right]\frac{\partial f}{\partial x}\right) \qquad (5.7\#\text{eq.}5)$$

Why is the motion of the pulse retarded at the right boundary? Play with u and D and determine for which values the particles can cross the boundary. *Hint: use the increments (5.3#eq.14b) calculated in exercise 5.01.*

5.06 Dynamics of a crowd. Simulate people walking using the continuity equation

$$\frac{\partial P}{\partial t} + \frac{\partial}{\partial x}v(P)P = 0 \qquad (5.7\#\text{eq.}6)$$

where P denotes the local people density. Indeed, the speed at which a person walks depends strongly on the density 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 in reality people do not always know where they are going? It may therefore be appropriate to include a diffusion term, making the behavior even more interesting. The complete equation to simulate the crowd is

$$\frac{\partial P}{\partial t} + \frac{\partial}{\partial x} v(P) P - D \frac{\partial^2 P}{\partial x^2} = 0 \qquad (5.7\#\text{eq.}7)$$

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

These problems can be edited / submitted for correction from your web browser by selecting *assignments* under the course main page.

5.8 Further Reading

- **Stochastic Differential Equations**
  Kloeden [23], Sczepessy [13].
- **Stochastic processes in physics and chemistry**
  Van Kampen [21].
- **Particle Methods**
  Birdsall[22] [6] and Hockney [16]
- **Examples of Monte Carlo integration**
  Option pricing[23], Ising model[24]
- **Monte Carlo Methods**
  Monte Carlo Methods[25] in WWW virtual library[26] including C++ classes maintained by Taygeta Scientific Incorporated[27]

5.9 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.

[22] http://ptsg.eecs.berkeley.edu/
[23] http://www.lifelong-learners.com/opt
[24] http://www2.truman.edu/%7Evelasco/ising.html
[25] http://random.mat.sbg.ac.at/others/
[26] http://vlib.org/Overview.html
[27] http://www.taygeta.com/stochastics.html

6 LAGRANGIAN METHOD

Rather than solving the convective derivative df/dt with advection directly in *Eulerian coordinates* as for example in the Lax-Wendroff method (sect.2.3), the idea behind Lagrangian schemes is to split the equation and solve for the convective derivative separately by propagating the initial conditions along the characteristics.

6.1 Splitting advection from the rest

Start by *splitting* the entire evolution into a sequence of alternating advection and non-advection phases

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

The advection part (including the convective derivative) is first evolved by propagating the solution along the characteristics; this is achieved using a Lagrangian method such as the explicit cubic-interpolated propagation (CIP) scheme, which is explicit and has no restriction on the time step. After this interpolation from the characteristics back to Eulerian coordinates, the second non-advection part can then be solved using one of the methods that have been discussed previously.

6.2 Cubic-Interpolated Propagation (CIP)

Introduced a decade ago by Yabe and Aoki [34], a whole family of schemes have been proposed along the same lines, relying on different interpolations to propagate the solution along the characteristics.

Using a cubic-Hermite polynomial, the discretized function and its first derivative $\{x_j, f_j, f'_j\}$ is 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 f, f' satisfy the master advection equation

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

and each is split into an advection (top) and non-advection phase (bottom)

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

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

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

without any restriction on the size or the sign of the Courant-Friedrich-Lewy (CFL) number $\beta = u\Delta t/\Delta x$ (exercise 6.01).

For pedagogical reasons, the scheme implemented in JBONE assumes that $\beta \in [0;1]$ so that both quantities $(f_{j+1}^{t+\Delta t}, f_{j+1}'^{t+\Delta t})$ can be interpolated from only one piece of the piecewise continuous polynomial, namely F_{j+1} defined in the interval $[x_j; x_{j+1}]$. After an initialization where the function is discretized with cubic-Hermite polynomials by sampling on a grid and the derivative are calculated with centered finite differences, the CIP scheme is implemented 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 j=0; j<n; j++) {
  a=dx[0]*(df[j]+  df[j+1])-2*(f[j+1]-f[j]);
  b=dx[0]*(df[j]+2*df[j+1])-3*(f[j+1]-f[j]);
  fp[j+1]=  f[j+1] -beta*(dx[0]*df[j+1]-beta*(b-beta*a));
 dfp[j+1]= df[j+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 document on-line illustrates the high quality of this approach, which combines a low level of dispersion with low damping.

Numerical experiments: cubic interpolated propagation

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

2. Switch to `Cubic---Splines` to test an alternative scheme, where cubic splines (1.4.4#eq.1) are used instead of Hermite polynomials to interpolate from any interval in the periodic domain. This allows for arbitrarily large CFL numbers $|\beta| > 1$ as can be tested by choosing `TimeStep=21.333` and changing the sign of the `Velocity`.

Some additional bookkeeping is of course necessary in a code that is intended for $|\beta| > 1$: exercise 6.01 deals with exactly this problem and can be implemented in a similar manner as illustrated with the `Cubic---Spline` scheme.

6.3 Non-Linear equations with CIP

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

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

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

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

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

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

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

$$\vec{f}_j^{\,*} = \vec{f}_j^{\,t} + \vec{G}_j \Delta t \tag{6.3\#eq.4}$$

where the super-script $(^*)$ refers to an intermediate step. To avoid an explicit evaluation of $\vec{G'_j}$, the equation for the derivative is computed with

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

The advection phase can then be evolved in the same manner as before (eq.6.2#eq.4), by shifting the cubic-Hermite polynomials along the characteristics (exercise 6.04).

6.4 Quiz

1. Lagrangian methods are specially designed to take care of
   (a) advection dominated problems
   (b) diffusion dominated problems
   (c) non-linear equations

2. The numerical error from a CIP scheme with advection only
   (a) increases with the number of steps
   (b) increases with the time step
   (c) does not depend on the time stepping

6.5 Exercises

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

6.02 Diffusion in CIP. Use your favorite method to implement the diffusion phase with CIP in the JBONE applet. Discuss the merits of such a splitting.

6.03 Lagrangian method with splines. Analyze and discuss the CIP scheme using cubic-splines to provide an interpolation for arbitrarily large time-steps.

6.04 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 and compare with the solutions obtained with other methods.

These problems can be edited / submitted for correction from your web browser by selecting *assignments* under the course main page.

6.6 Further Reading

- **Cubic-Interpolated Propagation (CIP).**
  Yabe and Aoki [34], [31], [32]

6.7 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.

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 number $\mathcal{O}(N)$ of operations as there are unknowns, relying on recent advances in iterative methods (sect.3.5) 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 still limited to a number of links to papers maintained on a web site from MathSoft[28]:

1. D. M. Bond and S. A. Vavasis, Fast Wavelet Transforms for Matrices Arising From Boundary Element Methods.[29]
2. T. Chan, W. Tang and W. Wan, Wavelet sparse approximate inverse preconditioners[30]
3. P. Charton and V. Perrier, Factorisation sur Bases d'Ondelettes du Noyeau de la Chaleur et Algorithmes Matriciels Rapides Associes[31]
4. P. Charton and V. Perrier, Towards a Wavelet Based Numerical Scheme for the Two-Dimensional Navier-Stokes Equations.[32]
5. P. Charton and V. Perrier, A Pseudo-Wavelet Scheme for the Two-Dimensional Navier-Stokes Equations.[33]
6. S. Dahlke and A. Kunoth, Biorthogonal Wavelets and Multigrid.[34]
7. S. Dahlke and I. Weinreich, Wavelet-Galerkin Methods: An Adapted Biorthogonal Wavelet Basis.[35]
8. S. Dahlke and I. Weinreich, Wavelet Bases Adapted to Pseudo-Differential Operators.[36]
9. W. Dahmen and A. Kunoth, Multilevel Preconditioning.[37]
10. R. Glowinski, T. Pan , R. O. Wells, Jr. and X. Zhou, Wavelet and Finite Element Solutions for the Neumann Problem Using Fictitious Domains[38]
11. R. Glowinski, A. Rieder, R. O. Wells, Jr. and X. Zhou, A Wavelet Multigrid Preconditioner for Dirichlet Boundary Value Problems in General Domains.[39]

[28] http://www.mathsoft.com/wavelets.html
[29] ftp://ftp.tc.cornell.edu/pub/tech.reports/tr174.ps
[30] ftp://ftp.math.ucla.edu/pub/camreport/cam96-33.ps.gz
[31] ftp://ftp.lmd.ens.fr/MFGA/pub/wavelets/produits2d.ps.Z
[32] ftp://ftp.lmd.ens.fr/MFGA/pub/wavelets/iciam95.ps.Z
[33] ftp://ftp.lmd.ens.fr/MFGA/pub/wavelets/ns.ps.Z
[34] ftp://ftp.igpm.rwth-aachen.de/pub/dahlke/dksh.ps.Z
[35] ftp://ftp.igpm.rwth-aachen.de/pub/ilona/wega.ps.Z
[36] ftp://ftp.igpm.rwth-aachen.de/pub/ilona/wega2.ps.Z
[37] ftp://ftp.igpm.rwth-aachen.de/pub/dahmen/mulpre.ps.gz
[38] ftp://cml.rice.edu/pub/reports/9201.ps.Z
[39] 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[40]

13. F. Heurtaux, F. Planchon and M. V. Wickerhauser, Scale Decomposition in Burgers' Equation[41]

14. A. Jiang, Fast wavelet based methods for certain time dependent problems[42]

15. A. Kunoth, Multilevel Preconditioning – Appending Boundary Conditions by Lagrange Multipliers.[43]

16. J. Lewalle, Wavelet Transforms of some Equations of Fluid Mechanics[44]

17. J. Lewalle, Energy Dissipation in the Wavelet-Transformed Navier-Stokes Equations[45]

18. J. Lewalle, On the effect of boundary conditions on the multifractal statistics of incompressible turbulence[46]

19. J. Lewalle, Diffusion is Hamiltonian[47]

20. D. Lu, T. Ohyoshi and L. Zhu, Treatment of Boundary Conditions in the Application of Wavelet-Galerkin Method to a SH Wave Problem[48]

21. A. Rieder and X. Zhou, On the Robustness of the Damped V-Cycle of the Wavelet Frequency Decompositions Multigrid Method[49]

22. A. Rieder, R. O. Wells, Jr. and X. Zhou, A Wavelet Approach to Robust Multilevel Solvers for Anisotropic Elliptic Problems.[50]

23. A. Rieder, R. O. Wells, Jr. and X. Zhou, On the Wavelet Frequency Decomposition Method[51]

24. O. V. Vasilyev and S. Paolucci, A Dynamically Adaptive Multilevel Wavelet Collocation Method for Solving Partial Differential Equations in a Finite Domain.[52]

25. O. V. Vasilyev, S. Paolucci and M. Sen, A Multilevel Wavelet Collocation Method for Solving Partial Differential Equations in a Finite Domain.[53]

26. R. O. Wells, Jr. and X. Zhou, Wavelet Solutions for the Dirichlet Problem[54]

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

27. R. O. Wells, Jr. and X. Zhou, Wavelet Interpolation and Approximate Solution of Elliptic Partial Differential Equations[55]

28. R. O. Wells, Jr. and X. Zhou, Representing the Geometry of Domains by Wavelets with Applications to Partial Differential Equations[56]

29. R. O. Wells, Jr., Multiscale Applications of Wavelets to Solutions f Partial Differential Equations[57]

[55] ftp://cml.rice.edu/pub/reports/9203.ps.Z
[56] ftp://cml.rice.edu/pub/reports/9214.ps.Z
[57] ftp://cml.rice.edu/pub/reports/9409.ps.Z

8 THE JBONE USER MANUAL

This chapter is not at all intended for the first part of the course, where the learning laboratory environement provides strong to implement only small modifications in the applet. The user manual, however, can be useful when deeper modifications are needed to carry our the projects or when the code is installed locally on your computer.

8.1 Source code & installation

The *Java Bed for ONE* dimensional evolution equations JBONE provides a flexible learning environement to test and compare numerical schemes in a JAVA applet. The full package consists of JAVA sources files `*.java`, installation instructions in the `README` file and a number of UNIX specific configuration files `config*` and compiler rules `Makefile*`.

Download. The JBONE source code can be obtained free of charge for personal use from the course main page[58]. After registration, the server will send a password that gives you access to the 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 ~                              # Somewhere in your directory
gunzip pde-course-*.tar.gz        # Uncompress
tar xvf pde-course-*.tar          # Unbundle
cd  applet
configure                         # Create Makefile for your system
```

A default `Makefile` is provided if the last command fails on an older platform.

Compile under UNIX. Different targets are built using the commands

```
make all                          # Everything below
make jbone                        # Compile java files
make docs                         # Generate documentation
```

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
```

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`.

Program listing. The substance of the program listing[59] (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

[58] http://www.lifelong-learners.com/pde
[59] http://www.lifelong-learners.com/pde/applet/listing.php3#rights

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>
    <center><table>
     <caption align=bottom><strong>
      <a href=../SYL/help_run.php3
         onMouseover="window.status='JBONE  #  parameters'; return true"
        >JBONE applet:</a></strong>  press <b>Start/Stop</b>
      to run the simulation.
     </caption>
    <tr><td>
     <applet codebase=jbone/ code=jbone.class
             align=center width=780 height=400>
       <param name=pde                value="Advection">
       <param name=method             value="Lagrangian">
       <param name=ic                 value="Gaussian">
       <param name=Velocity           value=1.>
     </applet>
```

```
    </td></tr></table></center>
    <br>
  </body>
</html>
```

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
- `MCPSolution.java` — for Monte Carlo particle schemes
- `MCSSolution.java` — for Monte Carlo sampling 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 `MCPSolution`. 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 `MCPDrawParticlesSolution`

8.5 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.

9 PROBLEM BASED LEARNING ENVIRONEMENT

This chapter gives a short introduction, advice and links to the documentation of tools that are used to run this course in a virtual university environement. Most of the tables can be consulted directly when needed on-line, simply by following the link above the input windows and using the browser Back button to recover your input.

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

where `(\ref{advection})` appears in the text as (9.1#eq.1) and can be used to reference your equations 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{itemize} \item \end{itemize}
\begin{quotation} \end{quotation}	\begin{enumerate} \item \end{enumerate}
\begin{center} \end{center}	\begin{description} \item \end{description}
\begin{verse} \end{verse}	\begin{equation} \label{key} \end{equation}
\begin{verbatim} \end{verbatim}	\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} \\
\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

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

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

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

9.2 Progamming in JAVA

Numerical schemes submitted from the Java window are automatically inserted in the JBONE source code (e.g. solution 2.01) 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 may need to carry out your 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[60] from Sun Microsystems.

JBONE = Java Bed for ONE dimensional problems.

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 $f(t - \Delta t)$. For example, the simple loop

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

   computes an artificial evolution where the function decays by 2% every step.
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 has in fact to be defined in the Java window.

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 k	`int k = runData.getNumberOfParticles();`
normalized advection β	`double beta;`
normalized diffusion α	`double alpha;`

[60] http://java.sun.com/docs/books/tutorial/

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]`).

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 (3) 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 refers to the element `c[0]`, you get 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
- **Walkers** 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", "Lagrangian methods", "Particle methods","Sampling methods".
- **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", "Lax-Friedrichs", "Leap-frog (FDTD)", "European naive", "European vanilla", "American naive", "American vanilla", "Tune Integration", "Expanded convol.", "Aliased convol.", "CubicHermite FEM", "Cubic—Splines", "MC norm. walk", "MC norm. walk pp", "MC log-normal", "MC log-normal pp", "My scheme", "Exercise 2.01", "Exercise 2.03", 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="$user_dir/applet/" code=$applet
        align=center width=780 height=400>
       <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=Walkers             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 2.01">
       <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.

9.4 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.

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; please submit a short description to the course leader and don't overestimate the work you can achieve during a week! For those who need some suggestions, here are a number of projects listed 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 for diffusion with FEM. 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; this leads to an expression for the 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 a diffusion dominated problem in a strongly inhomogeneous medium.

The uncertain future of a predator-prey system. During uncertain (stochastic) feeding conditions, the Volterra-Lotka system of equations (exercise 1.02) 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. Implement a computational scheme to solve the equation describing forced oscillation in a weakly absorbing and 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 $\nu \ll \omega_0 \simeq 2\pi c/L$ that 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 5.04) 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 with `discretize(new ShapeNumerical(this))`. The particle weight should be used to increase the accuracy in discretization. To complete this project, you will have to change the projection, the discretization and learn the Java `Vector` class.

Option pricing. Having implemented the FD and FEM schemes for the price of European and American options (exercises 2.05 and 3.07), complete this project implementing a MC solver for a out-barrier option, which looses all its value when the underlying exceeds a certain value. Validate each scheme with comparisons in the parameter range where all three overlap.

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 medium 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.

References

[1] M. Abramowitz and A. Stegun. *Handbook of Mathematical Functions.* Dover Publications, New York, tenth edition, 1972.

[2] N.W. Ashcroft and N.D. Mermin. *Solid State Physics.* Saunders, 1976.

[3] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. PETSc home page. The PETSc home page[61], 1999.

[4] J. P. Bérenger. A perfectly matched layer for the absorption of electomagnetic waves. *Journal of Computational Physics*, 114:185, 1994.

[5] S. Betts, S. Browne, J. Dongarra, E. Grosse, P McMahan, and T. Rowan. Netlib Software Repository[62].

[6] C. K. Birdsall and A. B. Langdon. *Plasma physics via computer simulation.* McGraw-Hill, 1985. The Plasma Theory and Simulation Group Homepage[63].

[7] T. Bjork. Examples in optimization theory. unpublished lecture notes, Inst. for Optimization and System Theory, KTH, SE-100 44 Stockholm, Sweden, 1995.

[8] A. Bondeson and G. Y. Fu. Tunable integration scheme. *Computer Physics Communications*, 66:167, 1991.

[9] J. Boris. *Methods in Computational Physics, Vol.16.* Academic, 1976.

[10] G. Dahlquist and Å Bjorck. *Numerical Methods.* Prentice-Hall, 1974.

[11] D. Duffie. *Dynamic Asset Pricing Theory.* Princeton University Press, 1992.

[12] J. Eastwood. Computer Physics Communications Library[64].

[13] A. Sczepessy et al. Stochastic Differential Equations[65]. unpublished lecture notes, Nada, KTH, SE-100 44 Stockholm, Sweden, 2001.

[14] C. A. Fletcher. *Computational Techniques for Fluid Dynamics.* Springer, second edition, 1991.

[15] Heath. *Scientific Computing: An Introductory Survey*[66]. McGraw Hill, second edition, 2002.

[16] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles.* Adam Hilger, 1988.

[17] J.C. Hull. *Options Futures and Other Derivative Securities.* Prentice-Hall International, New Jersey, fourth edition, 2000.

[61] http://www.mcs.anl.gov/petsc
[62] http://www.netlib.org
[63] http://ptsg.eecs.berkeley.edu/
[64] http://www.cpc.cs.qub.ac.uk/cpc/
[65] http://www.nada.kth.se/%7eszcepessy/
[66] http://www.cse.uiuc.edu/heath/scicomp/

[18] A. Jaun, K. Blomqvist, A. Bondeson, and T. Rylander. Iterative solution of global electromagnetic wavefields with finite elements. *Computer Physics Communications*, 135:74, 2001.

[19] J. M. Jin. *The Finite Element Method in Electromagnetics.* John Wiley, 1993.

[20] C. Johnson. *Numerical Solution of PDEs by the FEM Method.* Studentlitteratur – Lund, Sweden, 1987.

[21] N.G. Van Kampen. *Stochastic Processes in Physics and Chemistry.* North-Holland Personal Library, 1992.

[22] D. Katz and T. Cwik. EMLIB homepage[67].

[23] P. E. Kloeden and E. Platen. *Numerical Solution of Stochastic Differential Equations.* Springer-Verlag, second corrected printing edition, 1995.

[24] National Institute of Standards and Technology (NIST). Guide to Available Mathematical Software[68].

[25] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipies*[69]. Cambridge University Press, second edition, 1992.

[26] R. Rebonato. *Interest-Rate Option Models.* Wiley & Sons Ldt, UK, 1996.

[27] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, 1996.

[28] J. M. Sanz-Serna and P. Calvao. *Numerical Hamiltonian Problems.* Number 7 in Applied Mathematics Computation. Chapman & Hall, 1994.

[29] W. Sweldens. The Wavelet Home Page[70].

[30] A. Taflove and S. Hagness. *Computational Electromagnetics: the FDTD method.* Artech House, 2000.

[31] H. Takewaki and T. Yabe. The cubic-interpolated pseudo particle (cip) method. *Computer Physics Communications*, 70:355, 1987.

[32] T. Utsumi, T. Kunugi, and T. Aoki. Stability and accuracy of the cip scheme. *Computer Physics Communications*, 101:9, 1997.

[33] P. Wilmott, J. Dewynne, and S. Howison. *Option Pricing.* Oxford Financial Press, 1993.

[34] T. Yabe and T. Aoki. A universal solver for hyberbolic equations by cip. *Computer Physics Communications*, 66:219, 1991.

[67] http://http://emlib.jpl.nasa.gov/
[68] http://gams.nist.gov
[69] http://www.library.cornell.edu/nr/
[70] http://www.wavelet.org/wavelet/index.html