TP #1
Diffusion equation

The goal of this study is to implement, analyse and compare different numerical schemes to solve the diffusion equation. The associated lecture notes and the initial programs can be downloaded at: http://userpages.irap.omp.eu/ rbelmont/.

The langage chosen for this work is *Python*. For those who do not know this langage yet, a short note with the main points to know is given at the end.

1 Diffusion equation

Here we want to solve numerically the 1D heat equation for a field $u(t,x)$ on the domain $x \in [0, L]$, a diffusion coefficient D, an initial condition and a set of boundary conditions of Dirichlet type :

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}$$
$$u(0,x) = \sin\left(\frac{\pi x}{L}\right)$$
$$u(t,0) = u(t,L) = 0$$

It is recalled that the CFL parameter for diffusion equations discretized in time and space with steps δt and δx respectively, is:

$$\lambda = \frac{D\delta t}{\delta x^2}$$

The numerical solutions will be compared to the analytical solution of this equation:

$$T(t,x) = \sin\left(\frac{\pi x}{L}\right) e^{-\pi^2 \frac{Dt}{L^2}}$$

With no loss of generality, we can use $L = 1$ m et $D = 1$ m^{2}s^{-1} in the following. To start with, we can use $n_x = 64$ points in x, a conservative CFL condition $\lambda = 0.25$ and integrate up to $t_{\max} = 0.1$s. Then, these parameters will be varied.

2 Explicit FTCS scheme

A partial python code is proposed in file `diff_expl.py`.

Q01– Understand and fill that file with a 2nd order FTCS scheme (*Forward in Time, Centered in Space*).
Q02– Check experimentally the CFL stability criterium. Does it depend in the point number n_x?
Q03– When stable, how do the numerical solution compare to the theoretical ones at large time ?
Q04– What is the effect of n_x ?
Q05– Compare the different kinds of errors when the CFL parameter is varied in the stable range. What does happen for $\lambda \approx 0.17$? Explain these results with the consistency error analysis.

3 Implicit Crank-Nicolson scheme

An initial python program is proposed in file `diff_impl.py`. It assumes that the implicit scheme can be written in a matrix form with a tridiagonal matrix. In that case, its inversion can be performed with algorithms that are much more efficient than in the general case. In python, such matrix inversion is performed with function `tridag` which requires 4 arguments : the 3 diagonals $\{a_i\}, \{b_i\}$ et $\{c_i\}$ and the right hand side $\{s_i\}$.

$$\begin{pmatrix} b_0 & c_0 & 0 & ... & 0 & 0 & 0 \\ a_0 & b_1 & c_1 & ... & 0 & 0 & 0 \\ 0 & a_1 & b_2 & ... & 0 & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & b_{n-3} & c_{n-3} & 0 \\ 0 & 0 & 0 & ... & a_{n-3} & b_{n-2} & c_{n-2} \\ 0 & 0 & 0 & ... & 0 & a_{n-2} & b_{n-1} \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ ... \\ u_{n-3} \\ u_{n-2} \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ ... \\ s_{n-3} \\ s_{n-2} \\ s_{n-1} \end{pmatrix}$$

Q06– Write the Crank-Nicolson scheme for all internal points.
Q08– Show that the system can be written in the matrix form: $M_{ij}u_j = s_i$ where M_{ij} is tridiagonal. Give the terms a_i of the lower diagonal, b_i of the main diagonal, and c_i of the upper diagonal. Compute the components s_i of the right hand side.
Q08– Find the boundary values a_{n_x-2}, c_0, b_0, b_{n_x-1}, s_0 et s_{n_x-1} that correspond to the desired boundary conditions.
Q09– Understand and fill the proposed program with the implicit Crank-Nicholson scheme.
Q10– Check experimentally the effect of the CFL parameter on the stability.
Q11– How does the numerical solution compare to the theoretical one at large time ?
Q12– What is the effect of n_x ?
Q13– How does the CFL parameter influence the solution?

4 To go further

Q014– Modify to favorite scheme to include boundary conditions of Von Neumann type $\partial u/\partial x = 0$, that is with no heat flux at the boundaries.
Q015– The Richardson scheme is a second order scheme centered in space and time. Code that scheme and check its stability against the value of the CFL coefficient.

5 Memo Python

•A python script *myprog.py* is executed from a shell with the command `>python myprog.py` (or from the interactive mode (`ipython`) with command `>run myprog`).

•Tabulations are interpreted to define functions, loops, conditions etc. Be carefull!

•Comments start with symbol `#`.

•Python is optimized to work with arrays and vectors. Do your best to use that spirit and do not work component by component in loops when possible.

•With the *pylab* package, usual operations, functions, and constants are knowns (** pour les puissances, sin(), exp(), abs(), int(), modulo: n%2, π...).

•The syntax for a loop from $i = 0$ to 9 is: `for i in range(0,10):`

•Array elements are accessible with `tab[0]`, `tab[6]`. Attention: the first array index is 0!

•Syntaxe `tab[2:5]` is used to extract the sub-array constituted by elements from 2 to 4 included (not 5 !).

•`linspace(x1,x2,n)` create an array of de n values evenly spaced between x1 and x2 included.

•`zeros(n)` and `ones(n)` create vectors of n zeros and n ones respectively

•`roll(u,-3)` shifts all elements of an array of 3 indices to the left.

•With the *pylab* package, the plotting syntaxe is very similar to matlab.

•for instance, `plot(x,u,'+r')` plots array u as a function of x with red pluses.

•Multiple calls to `plot()` overplot on the same graphics. The final display is performed with command `show()`.