Introduction to Computational Science, Home+Practical Work

Contents

General remarks

The homework part can be handwritten and scanned after to be put in **o**ne pdf file or using any other software like word/LaTex to generate a pdf. Homework part can be written in french or english. It will be the same for the lab report part.

Introduction

The aim of this work will be to study numerical methods and results obtained for an advection-diffusion equation. This equation can be expressed as :

$$\frac{\partial u(x,t)}{\partial t} + V\nabla u(x,t) = C\nabla^2 u(x,t) \tag{1}$$

$$\nabla := \begin{cases} \partial_x \\ \partial_y \\ \partial_z \end{cases}$$

Assuming a domain of size : $[L_x \times L_y] = [2\pi \times 2\pi]$, choosen Δt and Δx must allow a simulation representing $\simeq \mathbf{10}$ times the typical time scale of the considered dynamic.
Diffusion time scale : $\tau_D = L^2/C$
Advection time scale : $\tau_A = L/V$

Values of parameters/grid sizes are not imposed/given, as starting point, you should use mesh size like $[8 \times 8, 16 \times 16, 32 \times 32, \cdots, 256 \times 256]$ and values of C, V in range like $\left[10^{-2}, 10\right]$.

The influence of the discretization scheme will be studied. In particular you will have to compare resolutions using explicit and implit Euler time schemes. This schemes are fast to implement but both have a possible issue which is that, both schemes are only first order schemes $O(\Delta t)$. There exist most advances schemes, as example the Runge-Kutta family schemes. These schemes are iterative schemes in the sense that to compute the function from t to $t + dt$, more than one estimation will be made in the aim to increase the precision of the final estimation. Actually the most used and known method is the 'the Runge-Kutta 4 method' which has a global error of order $O(\Delta t^4)$. You will have to compare these methods and in particular the influence of the method on the stability limits of the scheme.

1 Analytic work

The analytic work will focus on calculation of analytical solution and stability properties of the convection-diffusion equation. The aim will be, at the end, to compare analytic results and results obtained during the labs. This part should be made at home and not during labs.

1.1 Analytical solution

Starting from the advection-diffusion equation with constant coefficients on a periodic domain of size L_x.

$$\frac{\partial u(x,t)}{\partial t} + V\frac{\partial u(x,t)}{\partial x} = C\frac{\partial^2 u(x,t)}{\partial x^2} \tag{2}$$

$$u(x,0) = u_0(x) \tag{3}$$

1. Shows that advection equation ($C = 0$ et $V \neq 0$) has the following solution :

$$u_{adv}(x,t) = u_0(x - V * t) \tag{4}$$

2. Derive the solution associated with diffusion equation ($C \neq 0$ et $V = 0$) (use Fourier mode decomposition) assuming moreover the following condition :

$$\partial_x u(0,t) = \partial_x u(L_x,t)$$

3. Considering diffusion equation, assume that $u_0(x) = A\sin\left(2 * k\frac{\pi}{L_x}x\right)$. Verify in that case that solution has the form $u(x,t) = G(t)u_0(x)$. Shows that $G(t)$ has the form $G(t) = \exp(\gamma t)$ with γ independant of x and t.

4. Verify moreover that using the same initial condition, solution of the complete equation ($C \neq 0$ et $V \neq 0$) can be expressed as :

$$u(x,t) = Ae^{-\gamma t}\sin\left(2 * k\frac{\pi}{L_x}(x - Vt)\right) \tag{5}$$

1.2 Discretized equation

1.2.1 Diffusion

Considering only diffusion equation :

$$\frac{\partial u(x,t)}{\partial t} = C\frac{\partial^2 u(x,t)}{\partial x^2} \tag{6}$$

1. Using Von Neumann's technique, calculate stability criteria for both explicit and implicit euler schemes.

1.2.2 Advection

And now for the advection part,

$$\frac{\partial u(x,t)}{\partial t} + V\frac{\partial u(x,t)}{\partial x} = 0$$

1. Compute stability criteria for explicit and implicit euler using backward derivatives.

2. Does the previous results depends on the sign of V?.

1.2.3 Advection-Diffusion

Finally we will consider the complete equation :

$$\frac{\partial u(x,t)}{\partial t} = -V\frac{\partial u(x,t)}{\partial x} + C\frac{\partial^2 u(x,t)}{\partial x^2}$$

1. Derive discretized equation using backward derivative in space.
2. Calculate the local truncature error associated with explicit Euler scheme. Is it different from the result obtained for the advection only case?

2 1D equation

2.1 Diffusion

Considering only diffusive part in 1D :

$$\frac{\partial u(x,t)}{\partial t} = C\frac{\partial^2 u(x,t)}{\partial x^2} \tag{7}$$

$$u(x, t=0) = A\sin(bx) \tag{8}$$

1. Resolve numerically discrete equation using explicit Euler scheme (scheme=eule). Plot the returned value at last time step.

2. From that, deduce the kind of boundary conditions which are used? How is it implemented in the code?

3. Run new simulations with parameters corresponding to : $C\Delta t/\Delta x^2$ is $< 1/2, = 1/2 - \epsilon, = 1/2, = 1/2 + \epsilon, > 1/2$

4. In the case where $C\Delta t/\Delta x^2 < 1/2$, compute the growth rate of the solution. To compute the growth rate (γ), assume that :

$$u(x,t) = T(t)U(x),$$
$$\text{with : } T(t) = \exp(\gamma t)$$

   Hint : the function *extract_max* might be usefull for this question.

5. Using now the Runge Kutta 4 scheme, and for a given $\Delta x < 1$, do you observe a limit on time step for numerical stability? Estimate this new stability limit with a precision of 10^{-2}.

6. Compare this value with the one corresponding to an explicit Euler scheme.

7. Theoretically, there is no constraint on time step with implicit schemes, is it verified numerically? Comments.

2.2 Advection

This time we consider only advection :

$$\frac{\partial u(x,t)}{\partial t} + V\frac{\partial u(x,t)}{\partial x} = 0$$

1. Which implementation of spatial derivative are implemented in the code? is it backward, centered or forward scheme?

2. What is the physical definition of an advection ? Is radial structure conserved in simulations?

3. Is it the case in the simulations for the two following cases :

   - if a sin function is used for $u_0(x)$
   - if a *gate* function is used for $u_0(x)$

4. Assuming that $max(u(x,t)) = A\exp(\gamma t)$, compute again the value of γ for the 2 previous cases. Compare them.

5. How does the system evolves when $V\Delta t/\Delta x$ is $< 1, \simeq 1, = 1, > 1$

6. As in previous section, estimate the stability limit when you consider the RK4 time scheme.

2.3 Advection-Diffusion

Now we consider the full equation :

$$\frac{\partial u(x,t)}{\partial t} + V\frac{\partial u(x,t)}{\partial x} = C\frac{\partial^2 u(x,t)}{\partial x^2}$$

1. Calculate numerically diffusive growth rate from simulations. Is your result in agreement with expected value?

2. Runs simulations with centered or backward difference for convection. Calculate growth rate, is the value linked to coefficient V?

3. In previous sections, it has been shown that stability criterion exists for diffusion and convection, what happens if we choose parameters which verifies only one of these conditions?

3 Heat equation in 2D

Previous work will be extended to a $2D$ heat equation with a source.

$$\frac{\partial u(x,y,t)}{\partial t} = C\frac{\partial^2 u(x,y,t)}{\partial x^2} + C\frac{\partial^2 u(x,y,t)}{\partial y^2}, \tag{9}$$

Only Explicit Euler (EE) and Rung-Kutta 4 (RK4) time discretizations on a squared spatial grid will be studied in this part ($N_x = N_y$).

3.1 discretization in real space

1. Using the initial condition : $U_0(x,y) = \sin[\pi x/L_x]\sin[\pi y/L_y]$, estimate error with respect to exact solution in the numerical solution for 3 resolutions : 8×8, 32×32, 128×128. Does the relative error remains approximatively constant in time?

2. What are the boundary conditions implemented?

3. Using EE scheme and using previous stability analysis made for the $1D$ case, how can you extend the stability limit in this case? Choosing C, Δx and Δt such that $C\Delta t/\Delta x^2 = .25$, check numerically what is the corresponding numerical limit in y. Conclusion on the new stability limit?

4. For one set on inputs parameters and using RK4 scheme, represent computation time spent from t to $t+1$ for the following resolutions : 8×8, 32×32, 128×128, 256×256. Is it linear with respect to number of points? proportional to N^2? N^3? Remark : you will need the results of the simulation made with a grid size of 128×128 for the next section so backup them.

5. Implement in the code a function null_bc (in file boundary.c if you use the C version, and in the file operators_2d.pyx if you use the python version) which correspond to following condition in y:

$$\partial_y u(x,0,t) = 0, \quad \partial_y u(x,Ly,t) = 0$$

6. How does these new B.C. affects the results. Check numerically if the stability criterion is affected by these new conditions.

3.2 Fourier discretization

Finite difference is not the only spatial discretization which can be used in that case. Assuming periodicity in the y direction, Fourier modes decomposition can also be used :

$$u(x,y,t) \to \sum_{m=0}^{M} u_m(x,t)\exp(\mathrm{i}mk_y y)$$

As consequence, Eq. 9 can be splitted for each mode m :

$$\frac{\partial u_m(x,t)}{\partial t} = C\frac{\partial^2 u_m(x,t)}{\partial x^2} - Cm^2\kappa u_m(x,t), \tag{10}$$

1. Use a set of parameters of the previous section which where stable with a grid size of $Nx\times Ny = 128\times 128$. Use $N_m = 4, 8, 32, 64, 128$ and plot the corresponding results.

2. Compare these results to the corresponding ones with a discretization in real space.

3. for $N_m = 8$and $N_m = 128$, is the maximal time step allowed for stability the same? Estimate it with 1 significant digit and plot the results with respect to the number of modes.

4. Plot computational time from t to $t + 1$ for the same resolutions.

5. Which method seems to be faster? more accurate?

3.3 Conclusions

Using previous results, if you can choose only one method to resolve $1D$ diffusion-advection equation and one method to resolve $2D$ heat equation, which one will you choose in each case and why.

4 Appendices

4.1 Analytical results

4.1.1 General solution of heat equation on a finite domain

1D case

$$\begin{aligned}
\partial_t u(x,t) &= C\partial_x^2 u(x,t) + Du(x,t) + S(x) && (11)\\
\text{Domaine :} \quad & 0 \leq x \leq Lx && (12)\\
u(x,0) &= u_0(x) && (13)\\
u(0,t) &= 0 && (14)\\
u(0,Lx) &= 0 && (15)\\
u(x,t) &= \int_0^{Lx} u_0(x)G(x,\xi,t)\mathrm{d}\xi + \int_0^t \int_0^{Lx} S(\xi)G(x,\xi,t-\tau)\mathrm{d}\xi\mathrm{d}\tau && (16)\\
G(x,\xi,t) &= \frac{2}{Lx}e^{(Dt)} \sum_{n=1}^{\infty} \sin\left(\frac{n\pi x}{Lx}\right)\sin\left(\frac{n\pi\xi}{Lx}\right)\exp\left(-\frac{Cn^2\pi^2 t}{Lx^2}\right) && (17)\\
& && (18)
\end{aligned}$$

2D case

$$\begin{aligned}
\partial_t u(x,y,t) &= C\partial_x^2 u(x,t) + C\partial_y^2 u(x,y,t) + S(x,y) && (19)\\
\text{Domaine :} \quad & 0 \leq x \leq Lx && (20)\\
& -\infty \leq y \leq \infty && (21)\\
u(x,y,0) &= u_0(x,y) && (22)\\
u(0,y,t) &= 0 && (23)\\
u(Lx,y,t) &= 0 && (24)\\
u(x,t) &= \int_{-\infty}^{\infty}\int_0^{Lx} u_0(\xi,\eta)G(x,y,\xi,\eta,t)\mathrm{d}\xi\mathrm{d}\eta && (25)\\
& + \int_0^t \int_0^{Lx}\int_{-\infty}^{\infty} S(\xi,\eta)G(x,y,\xi,\eta,t-\tau)\mathrm{d}\xi\mathrm{d}\eta\mathrm{d}\tau && (26)\\
G(x,\xi,t) &= G_1(x,\xi,t)G_2(y,\eta,t) && (27)\\
G_1(x,\xi,t) &= \frac{2}{Lx}\sum_{n=1}^{\infty}\sin\left(\frac{n\pi x}{Lx}\right)\sin\left(\frac{n\pi\xi}{Lx}\right)\exp\left(-\frac{an^2\pi^2 t}{Lx^2}\right) && (28)\\
G_2(y,\eta,t) &= \frac{1}{2\sqrt{\pi Ct}}\left[\exp\left(-\frac{(y-\eta)^2}{4Ct}\right) - \exp\left(-\frac{(y+\eta)^2}{4Ct}\right)\right] && (29)
\end{aligned}$$

4.2 Numerical schemes

Supposing this convention for discretisation :

$$f(x,t) \to f(x_i,t_j) \to f(x+i\Delta x, t+j\Delta t) \to f_{x+j}^{t+i}$$

4.2.1 Derivates

And considering an equation expressed as

$$\frac{\partial f(x,t)}{\partial t} = L(t, f(x,t)) \tag{30}$$

$$\begin{array}{lrcl} \text{Forward difference :} & \partial_t f(t) & = & \frac{f^{t+1}-f^t}{\Delta t} \\ \text{Backward difference :} & \partial_t f(t) & = & \frac{f^t-f^{t-1}}{\Delta t} \\ \text{Centered difference :} & \partial_t f(t) & = & \frac{f^{t+1}-f^{t-1}}{2\Delta t} \end{array} \tag{31}$$

4.2.2 explicit Euler

$$\frac{f^{t+1}-f^t}{\Delta t} = L(t, f^t) \tag{32}$$

$$f^{t+1} = f^t + \Delta t L(t, f^t) \tag{33}$$

4.2.3 implicit Euler

$$\frac{f^{t+1}-f^t}{\Delta t} = L(t+\Delta t, f^{t+1}) \tag{34}$$

$$f^{t+1} = f^t + \Delta t L(t+\Delta t, f^{t+1}) \tag{35}$$

4.2.4 Cranck-Nicholson

$$\frac{f^{t+1}-f^t}{\Delta t} = \frac{1}{2}L(t, f^t) + \frac{1}{2}L(t+\Delta t, f^{t+1}) \tag{36}$$

$$f^{t+1} = f^t + \Delta t \left(\frac{1}{2}L(t, f^t) + \frac{1}{2}L(t+\Delta t, f^{t+1}) \right) \tag{37}$$

4.2.5 Runge-Kutta 4

$$f^{t+1} = f^t + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{38}$$

$$\text{with} \tag{39}$$

$$k_1 = \Delta t L(t, f^t) \tag{40}$$

$$k_2 = \Delta t L(t + \frac{\Delta t}{2}, f^t + \frac{1}{2}k_1) \tag{41}$$

$$k_3 = \Delta t L(t + \frac{\Delta t}{2}, f^t + \frac{1}{2}k_2) \tag{42}$$

$$k_4 = \Delta t L(t + \Delta t, f^t + k_3) \tag{43}$$

$$\tag{44}$$

4.3 Pre requisites

To be able to complete this lab you will need python and/or a C compiler. To analyze and plot results from simulation you can use any software you prefer (Matlab, Octave, etc). However in the following I will just indicate what are the necessary packages for python on windows/linux.

First of all and in all cases (even for the python version) you will need a C compiler :

- on Linux/Mac the simplest and most frequent choice is gcc. You can install it with a "brew install gcc" on Mac or "aptitude install gcc" on Debian/Ubuntu.

- on Windows, you will need Visual Studio Community Edition $\geq$ 2017. You can download it here :

  `https://visualstudio.microsoft.com/en/vs/`

If you don't want to install the full visual studio software. You can also use only the "Visual Cpp Build Tools" available at this address :

`https://visualstudio.microsoft.com/visual-cpp-build-tools/`

In that case, during install process please ensure the latest versions of MSVCv142 - VS 2019 C++ x64/x86 build tools and Windows 10 SDK are checked.

Now, if python is not installed on your system I recommand you to use the miniconda distribution with python¿=3.6 more than "just" installing python, in particular if you use Windows. You can download *miniconda* from the following website :

`https://docs.conda.io/en/latest/miniconda.html`

Install it as user and not as root/administrator.

And now you can install the requested python packages :

```
python -m pip install cython numpy pyyaml matplotlib pandas
```

or if you have installed python through conda :

```
conda install cython numpy pyyaml matplotlib pandas
conda -c conda-forge install ffmpeg
```

The optional installation of ffmpeg is recommended to see plots as animations.

If you want to use the jupyter notebook interface you will need also to install jupyter

```
python -m pip install jupyter
```

You can start jupyter notebook with the following command in a shell :

```
jupyter notebook
```

4.4 Installation of lab material

All the necessary files for the lab can be downloaded at the following address :
https://github.com/GFuhr/MF_FCM6/zipball/master

4.4.1 jupyter code

jupyter notebooks can be found in the python folder. You have 2 notebooks corresponding to part 1 and 2 of the lab.

4.4.2 python code

python code can be found in the python folder in the subfolders advdiff for TP1 and h2d for TP2. The parameters and the associated descriptions are in the python files *parameters.py*. You can run simulation directly in a shell with the command :

- For section 2 (1D advection-diffusion) :

```
python advdiff/advdiff.py
```

- For section 3.1

```
python h2d/h2d.py
```

- For section 3.2

```
python h2d/h2d_spectral.py
```

In the 3 cases, the main function to run a simulation is the function *simulate*. *simulate* takes 2 optional arguments :

```
verbose[=False] : print simulation progress on screen
save_files[=False] : save each snapshot to a text file.
```

4.4.3 C code

C code can be found in the C folder

Code AdvDiff1D Source code for AdvDiff1D is in folder C/TP1.

how to compile

- On linux , this program can be compiled via the **make** command.
- To run it on linux,after compilation step through **make**, type **./advdiff.exe**
- On Windows, with any IDE editor such as VisualStudio or Code::Blocks, loading files *advdiff.c* and *advdiff.h*.

4.4.4 Initial Conditions for the python version

- "C" : diffusion coefficient
- "V" : advection coefficient
- "Nx" : number of points in x direction
- "Ny" : number of points in y direction (or number of modes associated to the y direction in Fourier space)
- "Nm" : number of modes in the y direction (used only for the last part concerning spectral decomposition)
- "ky" : wave number used for simulations in Fourier space
- "dt" : time step
- "Tmax" : final time
- "Toutput" : intermediate times at which a snapshot of the field is made
- "dx" : step size in x direction (Lx = Nx*dx)

- "dy" : step size in y direction (Ly = Ny*dy)

- "scheme" : numerical scheme used
  "eule" : explicit Euler
  "euli" : implicit Euler
  "eulis" : implicit Euler with relaxations
  "rk4" : Runge-Kutta 4
  "cn" : Cranck-Nicholson

- "boundary" : kind of boundary conditions (used only for the H2D code)
  "default" : default boundary conditions
  "null_bc" : to use the boundary conditions asked in section 4

4.4.5 Initial Conditions for the C version

Time

	$advection$	$diffusion$
$t = 0$	$u^0(x)$	$u^0(x)$

Space

	$advection$	$diffusion$
$x = 0$	$u(t, 0)$	$u(t, 0)$
$x = L_x$		$u(t, L_x)$

Initial Fields

$u^0(x)$	$A \sin(\sigma \frac{\pi}{L_x}(x - x_0))$
$u^0(x)$	$A \exp(-(x - x_0)^2/\sigma^2)$
$u^0(x)$	$\begin{cases} A \text{ if } x \in [x_0 - \sigma/2,\, x_0 + \sigma/2] \\ 0 \text{ else} \end{cases}$

I/O Inputs :
Initial parameters must be entered when you run the program.

- "Nx=?" : number of points in radial direction

- "Npas=?" : number of iterations

- "Nout=?" : every $Nout$ iterations, $u(t, x)$ will be write in a file

- "C=?" : diffusion coefficient

- "V=?" : advection coefficient

- "A=?" : Amplitude of the initial field described in 4.7.2

- "x0=?": parameter x_0 defined in 4.7.2

- "sigma=?": parameter σ defined in 4.7.2

- "Dx=?" : radial step Δx

- "Dt=?" : time step Δt

Outputs :
Everytime you run the program, *Nout* output files will be generated **out_XXXX.dat**. Please notice that files will not be erased between runs. Each file is coded in text format (ASCII) and contains $(Nx-2)$ points with values of u_i^j at time $t = j * Nout$.

4.5 H2D code

4.5.1 Download

Code is in the same archive as previous one :
https://github.com/GFuhr/MF_FCM6/zipball/master,
H2D code is in folder TP2.

4.5.2 Compilation

- on linux, with make command.
- To run it on linux,after compilation step through **make**, type **./bin/h2d_gcc.exe**
- on windows, a "project" file usable with *Code::Blocks* and a "solution" file for *Visual Studio* can be found in folder *TP2/H2D/*

4.5.3 I/O

Inputs :
Initial parameters must be put in file params/params.h. Following parameters can be modified :

- "C" : diffusion coefficient
- "NX" : number of points in x direction
- "NY" : number of points in y direction (or number of modes associated to the y direction in Fourier space)
- "DT" : time step
- "ITER" : number of time iterations
- "LX" : Box size in x direction
- "LY" : Box size in y direction
- "discret" : spacial discretization used, can be "real" or "fourier"
- "scheme" : numerical scheme used
  "eule" : explicit Euler
  "euli" : implicit Euler
  "eulis" : implicit Euler with relaxations
  "rk4" : Runge-Kutta 4
  "cn" : Cranck-Nicholson

Initial fields/source can be modified in file params/functions.c

Outputs :

Two files will be generated at each runs : **H2D_GPLOT_XXXX.dat** and **H2D_OCT_XXXX.dat**. The only difference is that (files $\_OCT\_$) can be used with octave and (files $\_GPLOT\_$) with gnuplot.

Files $\_OCT\_$, contain Ny lines and Nx columns with values of $u(x, y, t)$. Files $\_GPLOT\_$ contain $Nx * Ny$ lines and Nx columns with values of $u(x, y, t)$.

Output format $u(x_i, y_j) \rightarrow u_{i,j}$**:**

Case where "discret=real", files $GPLOT$ **:**

$$\text{Nx*Ny lines, 3 columns} \begin{cases} x_0 & y_0 & u_{i,j} \\ x_1 & y_0 & u_{i,j} \\ \vdots & \vdots & \vdots \\ x_{Nx-1} & y_0 & u_{Nx-1,Ny-1} \\ x_0 & y_1 & u_{0,1} \\ \vdots & \vdots & \vdots \\ x_{Nx-1} & y_{Ny-1} & u_{Nx-1,Ny-1} \end{cases} \tag{45}$$

Case where "discret=fourier", files $GPLOT$ **:**

$$\text{2*Nx*Ny lines, 3 columns} \begin{cases} x_0 & m_0 & \Re(u_{i,j}) \\ x_1 & m_0 & \Re(u_{i,j}) \\ \vdots & \vdots & \vdots \\ x_{Nx-1} & m_0 & \Re(u_{Nx-1,Ny-1}) \\ x_0 & m_1 & \Re(u_{0,1} \\ \vdots & \vdots & \vdots \\ x_{Nx-1} & m_{Ny-1} & \Re(u_{Nx-1,Ny-1}) \\ x_0 & m_0 & \Im(u_{i,j}) \\ x_1 & m_0 & \Im(u_{i,j}) \\ \vdots & \vdots & \vdots \\ x_{Nx-1} & m_0 & \Im(u_{Nx-1,Ny-1}) \\ x_0 & m_1 & \Im(u_{0,1}) \\ \vdots & \vdots & \vdots \\ x_{Nx-1} & m_{Ny-1} & \Im(u_{Nx-1,Ny-1}) \end{cases} \tag{46}$$

Case where "discret=real", files OCT **:**

$$\text{Ny lines, Nx columns} \begin{cases} u_{0,0} & u_{1,0} & \cdots & u_{Nx-1,0} \\ u_{0,1} & u_{1,1} & \cdots & u_{Nx-1,1} \\ \vdots & \vdots & \vdots & \\ u_{0,Ny-1} & u_{1,Ny-1} & \cdots & u_{Nx-1,Ny-1} \end{cases} \tag{47}$$

Case where "discret=fourier", files OCT **:**

$$\text{2*Ny lines, Nx columns} \left\{ \begin{array}{cccc} \Re(u_{0,0}) & \Re(u_{1,0}) & \cdots & \Re(u_{Nx-1,0}) \\ \Re(u_{0,1}) & \Re(u_{1,1}) & \cdots & \Re(u_{Nx-1,1}) \\ \vdots & \vdots & \vdots & \\ \Re(u_{0,Ny-1}) & \Re(u_{1,Ny-1}) & \cdots & \Re(u_{Nx-1,Ny-1}) \\ \Im(u_{0,0}) & \Im(u_{1,0}) & \cdots & \Im(u_{Nx-1,0}) \\ \Im(u_{0,1}) & \Im(u_{1,1}) & \cdots & \Im(u_{Nx-1,1}) \\ \vdots & \vdots & \vdots & \\ \Im(u_{0,Ny-1}) & \Im(u_{1,Ny-1}) & \cdots & \Im(u_{Nx-1,Ny-1}) \end{array} \right. \tag{48}$$

4.6 usefull commands

- to measure computation time, command **time** is used :
  time ./bin/h2d_gcc.exe
  Remark : time measure can be considered reliable only if it's greater than 10 seconds.
- How to plot 3D data with gnuplot. In following, we suppose NX=64 and NY=64

```
gnuplot> set dgrid3d 64,64
gnuplot> set hidden3d
gnuplot> splot "H2D_0000.dat" u 1:2:3 with lines
```

- How to plot 1D data with gnuplot.

```
gnuplot> plot "out_0000.dat" with lines
```

- with Octave
  - to read datas use function *load*
  - to plot datas $u(x, y)$, use function *surf*

```
octave> data=load('H2D_0000.dat');
octave> surf(data)
```

- download code in a terminal :
  wget https://github.com/GFuhr/MF_FCM6/zipball/master
- unzip archive :
  unzip master
- put unzipped files in a directory with a "friendly name" :
  mv GFuhr-MF_FCM6* *newname*
- change directory :
  cd
- create directory mkdir
  mkdir directory_name
- delete file :
  rm filename

- list file in a directory :
  ls directory_name
- list file in current directory :
  ls ./

4.7 Some python commands

- modules to load at startup to be able to use mathematical functions

```
import numpy as np
import matplotlib.pyplot as plt
from os import chdir, getcwd
import sys
```

- change folder chdir('path')
- Load data in a text file, plot output and save result as a png file

```
plt.figure();
data = np.loadtxt('out__0000.dat')
plt.plot(data)
plt.xlabel('x')
plt.ylabel('u(x)')
plt.title('title')
plt.savefigs('plot.png')
plt.show()
```

- Plot data series correspondingto numbered files

```
plt.figure();
for idx in np.arange(0,10):
    data = np.loadtxt('out__{0:04}.dat'.format(idx))
    print(max(data))
    plt.plot(data, label = idx)
plt.legend()
plt.xlabel('x')
plt.ylabel('u(x)')
plt.title('diffusion')
plt.show()
```

- Load and plot H2D data :

```
from h2d_loader import  h2Dloader
data = h2Dloader('H2D_OCT_0430.dat')
data.convertreal()
data.plot()
data.plotmodes()
print data.max()
print data.min()
#data in real format
data = h2Dloader('H2D_OCT_0190.dat')
data.plot()
plt.show()
```

4.7.1 Initial Conditions for the python version

- "C" : diffusion coefficient
- "V" : advection coefficient
- "Nx" : number of points in x direction
- "Ny" : number of points in y direction (or number of modes associated to the y direction in Fourier space)
- "Nm" : number of modes in the y direction (used only for the last part concerning spectral decomposition)
- "ky" : wave number used for simulations in Fourier space
- "dt" : time step
- "Tmax" : final time
- "Toutput" : intermediate times at which a snapshot of the field is made
- "dx" : step size in x direction (Lx = Nx*dx)
- "dy" : step size in y direction (Ly = Ny*dy)
- "scheme" : numerical scheme used
  "eule" : explicit Euler
  "euli" : implicit Euler
  "eulis" : implicit Euler with relaxations
  "rk4" : Runge-Kutta 4
  "cn" : Cranck-Nicholson

- "boundary" : kind of boundary conditions (used only for the H2D code)
  "default" : default boundary conditions
  "null_bc" : to use the boundary conditions asked in section 4

4.7.2 Initial Conditions for the C version

Time

	$advection$	$diffusion$
$t = 0$	$u^0(x)$	$u^0(x)$

Space

	$advection$	$diffusion$
$x = 0$	$u(t, 0)$	$u(t, 0)$
$x = L_x$		$u(t, L_x)$

Initial Fields

$u^0(x)$	$A \sin(\sigma \frac{\pi}{L_x}(x - x_0))$
$u^0(x)$	$A \exp(-(x - x_0)^2/\sigma^2)$
$u^0(x)$	$\begin{cases} A \text{ if } x \in [x_0 - \sigma/2,\, x_0 + \sigma/2] \\ 0 \text{ else} \end{cases}$

I/O Inputs :
Initial parameters must be entered when you run the program.

- "Nx=?" : number of points in radial direction
- "Npas=?" : number of iterations
- "Nout=?" : every $Nout$ iterations, $u(t,x)$ will be write in a file
- "C=?" : diffusion coefficient
- "V=?" : advection coefficient
- "A=?" : Amplitude of the initial field described in 4.7.2
- "x0=?": parameter x_0 defined in 4.7.2
- "sigma=?": parameter σ defined in 4.7.2
- "Dx=?" : radial step Δx
- "Dt=?" : time step Δt

Outputs :
Everytime you run the program, *Nout* output files will be generated **out_XXXX.dat**. Please notice that files will not be erased between runs. Each file is coded in text format (ASCII) and contains $(Nx-2)$ points with values of u_i^j at time $t = j * Nout$.

4.8 H2D code

4.8.1 Download

Code is in the same archive as previous one :
https://github.com/GFuhr/MF_FCM6/zipball/master,
H2D code is in folder TP2.

4.8.2 Compilation

- on linux, with make command.
- To run it on linux,after compilation step through **make**, type **./bin/h2d_gcc.exe**
- on windows, a "project" file usable with *Code::Blocks* and a "solution" file for *Visual Studio* can be found in folder *TP2/H2D/*

4.8.3 I/O

Inputs :
Initial parameters must be put in file params/params.h. Following parameters can be modified :

- "C" : diffusion coefficient
- "NX" : number of points in x direction

- "NY" : number of points in y direction (or number of modes associated to the y direction in Fourier space)

- "DT" : time step

- "ITER" : number of time iterations

- "LX" : Box size in x direction

- "LY" : Box size in y direction

- "discret" : spacial discretization used, can be "real" or "fourier"

- "scheme" : numerical scheme used
  "eule" : explicit Euler
  "euli" : implicit Euler
  "eulis" : implicit Euler with relaxations
  "rk4" : Runge-Kutta 4
  "cn" : Cranck-Nicholson

Initial fields/source can be modified in file params/functions.c

Outputs :
Two files will be generated at each runs : **H2D_GPLOT_XXXX.dat** and **H2D_OCT_XXXX.dat**. The only difference is that (files $\_OCT\_$) can be used with octave and (files $\_GPLOT\_$) with gnuplot.

Files $\_OCT\_$, contain Ny lines and Nx columns with values of $u(x, y, t)$. Files $\_GPLOT\_$ contain $Nx * Ny$ lines and Nx columns with values of $u(x, y, t)$.

Output format $u(x_i, y_j) \rightarrow u_{i,j}$**:**

Case where "discret=real", files $GPLOT$ **:**

$$\text{Nx*Ny lines, 3 columns} \left\{ \begin{array}{ccc} x_0 & y_0 & u_{i,j} \\ x_1 & y_0 & u_{i,j} \\ \vdots & \vdots & \vdots \\ x_{Nx-1} & y_0 & u_{Nx-1,Ny-1} \\ x_0 & y_1 & u_{0,1} \\ \vdots & \vdots & \vdots \\ x_{Nx-1} & y_{Ny-1} & u_{Nx-1,Ny-1} \end{array} \right. \tag{49}$$

Case where "discret=fourier", files $GPLOT$:

$$\text{2*Nx*Ny lines, 3 columns} \left\{ \begin{array}{ccc} x_0 & m_0 & \Re(u_{i,j}) \\ x_1 & m_0 & \Re(u_{i,j}) \\ \vdots & \vdots & \vdots \\ x_{Nx-1} & m_0 & \Re(u_{Nx-1,Ny-1}) \\ x_0 & m_1 & \Re(u_{0,1} \\ \vdots & \vdots & \vdots \\ x_{Nx-1} & m_{Ny-1} & \Re(u_{Nx-1,Ny-1}) \\ x_0 & m_0 & \Im(u_{i,j}) \\ x_1 & m_0 & \Im(u_{i,j}) \\ \vdots & \vdots & \vdots \\ x_{Nx-1} & m_0 & \Im(u_{Nx-1,Ny-1}) \\ x_0 & m_1 & \Im(u_{0,1}) \\ \vdots & \vdots & \vdots \\ x_{Nx-1} & m_{Ny-1} & \Im(u_{Nx-1,Ny-1}) \end{array} \right. \tag{50}$$

Case where "discret=real", files OCT :

$$\text{Ny lines, Nx columns} \left\{ \begin{array}{cccc} u_{0,0} & u_{1,0} & \cdots & u_{Nx-1,0} \\ u_{0,1} & u_{1,1} & \cdots & u_{Nx-1,1} \\ \vdots & \vdots & \vdots & \\ u_{0,Ny-1} & u_{1,Ny-1} & \cdots & u_{Nx-1,Ny-1} \end{array} \right. \tag{51}$$

Case where "discret=fourier", files OCT :

$$\text{2*Ny lines, Nx columns} \left\{ \begin{array}{cccc} \Re(u_{0,0}) & \Re(u_{1,0}) & \cdots & \Re(u_{Nx-1,0}) \\ \Re(u_{0,1}) & \Re(u_{1,1}) & \cdots & \Re(u_{Nx-1,1}) \\ \vdots & \vdots & \vdots & \\ \Re(u_{0,Ny-1}) & \Re(u_{1,Ny-1}) & \cdots & \Re(u_{Nx-1,Ny-1}) \\ \Im(u_{0,0}) & \Im(u_{1,0}) & \cdots & \Im(u_{Nx-1,0}) \\ \Im(u_{0,1}) & \Im(u_{1,1}) & \cdots & \Im(u_{Nx-1,1}) \\ \vdots & \vdots & \vdots & \\ \Im(u_{0,Ny-1}) & \Im(u_{1,Ny-1}) & \cdots & \Im(u_{Nx-1,Ny-1}) \end{array} \right. \tag{52}$$

4.9 usefull commands

- to measure computation time, command **time** is used :
  time ./bin/h2d_gcc.exe
  Remark : time measure can be considered reliable only if it's greater than 10 seconds.

- How to plot 3D data with gnuplot. In following, we suppose NX=64 and NY=64

```
gnuplot> set dgrid3d 64,64
gnuplot> set hidden3d
gnuplot> splot "H2D_0000.dat" u 1:2:3 with lines
```

- How to plot 1D data with gnuplot.

```
gnuplot> plot "out_0000.dat" with lines
```

- with Octave
  - to read datas use function *load*
  - to plot datas $u(x, y)$, use function *surf*

```
octave> data=load('H2D_0000.dat');
octave> surf(data)
```

- download code in a terminal :
  wget https://github.com/GFuhr/MF_FCM6/zipball/master
- unzip archive :
  unzip master
- put unzipped files in a directory with a "friendly name" :
  mv GFuhr-MF_FCM6* *newname*
- change directory :
  cd
- create directory mkdir
  mkdir directory_name
- delete file :
  rm filename
- list file in a directory :
  ls directory_name
- list file in current directory :
  ls ./

4.10 Some python commands

- modules to load at startup to be able to use mathematical functions

```
import numpy as np
import matplotlib.pyplot as plt
from os import chdir, getcwd
import sys
```

- change folder chdir('path')
- Load data in a text file, plot output and save result as a png file

```
plt.figure();
data = np.loadtxt('out__0000.dat')
plt.plot(data)
plt.xlabel('x')
plt.ylabel('u(x)')
```

```
plt.title('title')
plt.savefigs('plot.png')
plt.show()
```

- Plot data series correspondingto numbered files

```
plt.figure();
for idx in np.arange(0,10):
    data = np.loadtxt('out__{0:04}.dat'.format(idx))
    print(max(data))
    plt.plot(data, label = idx)
plt.legend()
plt.xlabel('x')
plt.ylabel('u(x)')
plt.title('diffusion')
plt.show()
```

- Load and plot H2D data :

```
from h2d_loader import  h2Dloader
data = h2Dloader('H2D_OCT_0430.dat')
data.convertreal()
data.plot()
data.plotmodes()
print data.max()
print data.min()
#data in real format
data = h2Dloader('H2D_OCT_0190.dat')
data.plot()
plt.show()
```