C. Boily, R. Schotter

Term Papers

Numerical Analysis

& Computational Physics

WINTER 2022

University of Strasbourg

Front page image credit: wikipedia / Julia set entry.

PUBLISHED BY UNIVERSITY OF STRASBOURG

TUFTE-LATEX.GOOGLECODE.COM

First printing, February 2022

Contents

Presentation and objectives

The overall objective of these notes is to introduce the term papers that will start during week 4 (26 January 2022). The list of projects is available on Moodle/Ernest in pdf format. If you have any problems accessing them, you can contact us by email: romain.schotter@unistra.fr or christian.boily@astro.unistra.fr.

Although you have full flexibility to use online resources and to cooperate with each other, the project you develop should reflect your personal input: a team report will be the basis for the oral interview at the end of the semester. A stated objective of the term projects is to favoured team work and the share of the work in teams of 5 to 6 students.

Evaluation elements

Your grades will focus in equal measures on:

(1) the quality of your written report;

(2) on the computer programs you developed;

(3) the quality of the oral presentation ; and

(4) the results obtained and their interpretation.

You will have to deposit an archive containing your report in pdf format, your programs, a file containing the instructions necessary for the **execution** as well as the input files and scripts (if you have any) by 6:00 pm on **Wednesday, March 23, 2022**; the archive will be open on Monday, March 11. More details will be communicated nearer the date of submission.

The Written Report: What to include

The report must present in words the summary of your work: context, objectives, methods of resolution, results and their interpretation, conclusion and opening to other related problems.

The report should **not exceed 5 pages** inclusive of all information. It should be in sync with the oral presentation, and one should complement the other. So for example, it is conceivable to cover the details of a graphing or data reduction in the report, so that less time is spent on it during the oral presentation. A bibliographic appendix can be added to the report (it will not be counted as part of the 5-page essay).

The oral interview: technical details

Each team of students must prepare an individual presentation that they will present as a team in front of the jury. The team has to decide how best to convey the results of their investigation (a single spokesperson, a split presentation, etc). Each team will have 15 minutes to present and then up to 45 minutes to answer questions about the oral presentation and the written report. The presentation will be made using a video projector [standard vga or hdmi plug in]. You may use the software of your choice to prepare your slides (latex, [libre]office, powerpoint, etc.); however, it is required to prepare a pdf version of your presentation in case of a technical issue on the day of the interview. Overheads must be submitted in pdf format on Moodle no later than **Monday, May 14 at midnight**. If you have animations and wish to use your own computer, it will be possible to do so. However, please bring a version of your presentation on a USB key if possible.

The oral interview: date and place

Presentations will take place on May??, 2022 (tbc). Room assignments will be communicated to you during the week: conflicting schedule will be taken into account as much as it is possible in the attribution of the order of passage (of the teams).

Programs: layout details

- The code must be readable and indented;
- It must be judiciously commented and documented;
- The program must be modular, scalable and configurable as much as possible:
  - Use of functions, structures, classes, ...
  - Use of input parameters, files (input/output), ...

  Although it is not required, it is possible to develop your project in Python3.8 by creating your own classes. Doing so will demonstrate

your full mastery of the language, and will contribute to a better overall grade.

Note : The use of libraries and/or graphic software is allowed and even encouraged. However, this use of resources must be mentioned during the oral presentation.

The oral presentation: important points

- Please make the best use of the time allotted to you.
- Be pedagogical in your presentation of the topic, without being too simplistic.
- **Project management** ;
  You can present your scientific approach, the choices you have made, the structure of your program (without technical details, with program (without technical details, with diagrams),...
- Numerical methods ; A detailed analysis of the numerical method used is required: evolution of the results according to the parameters of the method (convergence, stability, etc.) parameters of the method (convergence, stability, ...), etc. However, this analysis can be based on what has been covered in the report (see remark above).
- A quantitative analysis of the physical results obtained must be presented with all the scientific rigor that is required:
  - Quantities must have units
  - Graphs must have titles, axes, legends, etc.
  - Think about the pertinence of the chosen representation (choice of axes, scale, ...)
  - The graphics must have titles, axes, legends, ...
- **Interpretation** ; The results must be commented and interpreted. Be critical with the results obtained (uncertainties, limitations of the model, ...)

Notice

This document is a rough transcription of the project proposed this year. The actual material covered in the classroom may add to this material, however the topics remain identical or very close to what is shown herein.

Bibliography & Internet References

Some useful references for this course include textbooks in both French and English:
J. P. Demailly (1996). *Analyse numérique et équations différentielles*. First Ed. EDP Sciences, Paris France
Randall J. LeVeque (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations*. SIAM, Philadelphia, USA
W.H. Press et al. (1992). *Numerical Recipes*. Cambridge: The University Press

The support material for the lectures can be found on the web at:

Numerical Analysis and Computational Physics (n.d.). ENT/Moodle Programmation M1 UE3. URL: `https://moodle3.unistra.fr/course/view.php?id=4540`

Summary of chapter bibliography

Demailly, J. P. (1996). *Analyse numérique et équations différentielles*. First Ed. EDP Sciences, Paris France.
LeVeque, Randall J. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations*. SIAM, Philadelphia, USA.
Press, W.H. et al. (1992). *Numerical Recipes*. Cambridge: The University Press.

1 The Schroedinger equation

1.1 Overview of principles

The non-relativistic Schrödinger equation (1.1) drives the time-evolution of a quantum mechanical wave function $\Psi(x,t)$ of a particle, subject to a Hamiltonian field:

$$i\hbar\frac{\partial}{\partial t}\psi(x,t) = H\Psi(x,t) = \left[\frac{p^2}{2m} + V(x)\right]\Psi(x,t) = \left[-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x)\right]\Psi(x,t) \tag{1.1}$$

with the vector position, t the time, and $i^2 = -1$ denotes the imaginary number. The wave function is therefore a complex quantity. The Planck constant h is here divided by 2π, so $\bar{h} = h/2\pi$. The partial derivatives with respect to time and space link the classical position-velocity interpretation of the motion of a point-mass object[1], to the quantum mechanics interpretation of a wave probability density interpretation of the motion of an object. The wave function $\Psi(x,t)$ can be expanded as the product of two functions of a single free variable, one the position, the other the time, so:

[1] Note how the momentum-frequency relation of de Broglie is important in making this step.

$$\Psi(\mathbf{x},t) \equiv \psi(\mathbf{x})\,\phi(t) \tag{1.2}$$

and we will drop the bold face on when working in one dimension. We can split the variables with as solution

$$\phi(t) = \exp(-i\,Et/h)$$

where E is the mechanical energy, and hence E/h has dimensions of a frequency. Substituting in Eq. (1.3) leads to the **stationary Schrödinger** equation,

$$-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\psi(x) + V(x)\psi(x) = E\psi(x)\ . \tag{1.3}$$

An *eigenstate* is defined by the *eigenfunction* $\psi(x)$ of fixed mechanical energy E. In general one can make a parallel with the situation of point-mass of classical mechanics evolving in a constrained

(bounded) volume by a potential $V(x)$. When $E > V(x)$ in that volume, then the eigenstates allowed by the Schrödinger equation are quantified. We then say that the eigenfunction ψ_n matches an eigenvalue of energy $E = E_n$, $n = 1, 2, ...$ The ensemble of states is discrete.

1.2 *Analytical tests: the free particle $V(x) = 0$*

As an appetizer, consider the case when the binding potential $V(x)$ is everywhere void. We seek a solution analytically in one-dimensional space.

a) Find the analytical solution to this problem, and validate the numerical integration procedure using (for example) the Numerov integration scheme. Care should be taken to identify the boundary conditions and the choices made (units, gauge, ..) ;

b) Show that the choice of length and energy are arbitrary, then exploit this to defined integration units well-suited to your problem ;

c) Recover the analytical wavefunction $\psi(x)$ from your numerical scheme, using the mid-point matching technique at $x = 0$ (see Fig. 1.1 for explanations).

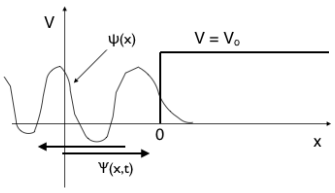

Figure 1.1: The sketch shows the possible behaviour of a wave function at / near the boundary where the potential V changes rapidly.

1.3 *1D Schrödinger wave for various potentials*

Formulation

The equation to solve for eigenstates of the system in one dimension is

$$\begin{aligned} &\frac{d^2}{dx^2}\psi(x) + J(x)\psi(x) = 0 \\ &J(x) \equiv \frac{2m}{\hbar^2}\left[E - V(x)\right] \ . \end{aligned} \tag{1.4}$$

It is useful to list the properties of the wave function $\psi(x)$ for given potential properties, such as

- When $V(x)$ admits a symmetry, say with respect to the origin of the coordinates $x = 0$, then setting $x_{min} = -x_{max}$ immediately implies $\psi(x_{min}) = \psi(x_{max})$;
- The energy E of a particle of mass m is the sum of potential + kinetic energies. If the potential V is defined as a negative quantity, then a bound eigenstate must have $\min(V) < E < 0$;

- The system of equations (1.4) is best rewritten as two first-order differential equations.

In the following, we specify which potentials V aught to be considered for the project.

1.4 Potentials

We define three potentials represented schematically on Fig. 1.2 respectibely as

a) Flat potential. We set
$$V(x) = V_o$$
in the interval $x_{min} \leq x \leq x_{max}$ and otherwise $V(x) \to \infty$ everywhere else. This is the situation when a particle is trapped by insulating walls;

b) Quadratic potential. Many continuous curves car be approximated by a quadratic shape over a small volume. If we set
$$\Delta = (x_{max} - x_{min})/2,\ \Sigma = (x_{max} + x_{min})/2\,,$$
then $V(x_{min} < x < x_{max}) = V_o\ \left(1 - [x - \Sigma]^2/\Delta^2\right)$, and $V = 0$ otherwise (outside the cavity) ;

c) The Lennard-Jones potential. This one admits a familiar, step form near the edges, where we find
$$V(x) = V_o \left(\left[\frac{x_{min}}{x}\right]^6 - \left[\frac{x_{max}}{x}\right]^{12} \right)$$
which admits a minimum value $= V_o/4$ at $x = 2^{1/6} x_{min}$.

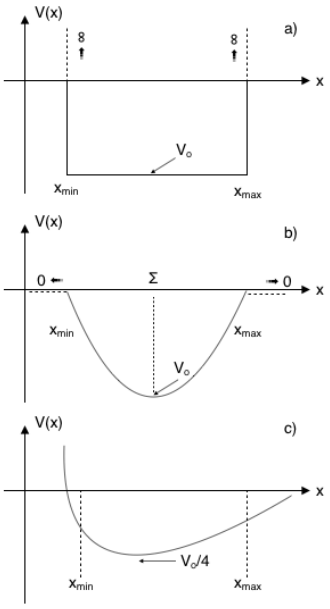

Figure 1.2: The various potentials for which eigenstates are sought.

1.5 Project work sheet

Methodology

The mid-point matching technique implies that one may treat the boundary conditions simultaneously at the inner- and outer boundary. Thus the "matching" of pairs of curves, one integrated from the inner boundary, outward, the other integrated from the outer boundary, inward, must occur at some point $x = x_m$ chosen by the user. For a physically realistic solution, we must find $E > \min(V)$ inside the bounded volume. We should check that the solution obtained numerically $\psi(x_m)$ is continuous in both order zero and to first order. If we denote with arrows the direction of the integration path, so " $<$ " for

outward $\rightarrow$ inward ; and " $>$ " for inward $\rightarrow$ outward, then these matching conditions read

$$\begin{aligned} \Psi_<(x_m) &= \Psi_>(x_m) \\ \left.\frac{d}{dx}\Psi_<(x)\right|_{x_m} &= \left.\frac{d}{dx}\Psi_>(x)\right|_{x_m} . \end{aligned} \tag{1.5}$$

Questions

Any numerical integration will not meet these conditions exactly and one will have to assess whether the differences are due to *e.g.* the choice of E, the numerical scheme and its parameters (space steps, ..) ?

Determine whether the amplitude of the wave function is constrained at either points x_{min} and x_{max}: do you have the freedom to fix one value for instance ? Does this have any impact on your integration strategy ?

When two eigenstates ψ_1 and ψ_2 have been obtained, of energy respectively E_1 and E_2, they represent independent (and so, orthogonal) states. The mathematical expression of this is known in integral form

$$\int_{x_{min}}^{x_{max}} \psi_{E_1}(x)\,\psi_{E_2}(x)\ \mathrm{d}x = 0 . \tag{1.6}$$

Show that this relation is validated for at least one pair of solution, for as many potentials V as possible.

Procedure, strategy

1. Start with the simplest, flat potential to calibrate the integration procedure and control your boundary conditions ;

2. Try and maintain a code that is as modular as possible using function calls, separate text files, classes and methods ;

3. Use the Unix **make** utility to allow target compilations of the code ;

4. Check your equations and make sure that you choose a gauge that allows to limit the dynamic range of the numbers as much as possible, allowing for greater accuracy and speed.

1.6 References

On top of the lecture notes that we have / are currently provided and which will be updated throughout the semester, these are additional references that you may find helpful:

- R. Eisberg, R. Resnick, Quantum Physics, 2ème édition, New York: Wiley, 1985
- Wikipedia : `https://en.wikipedia.org/wiki/Schrodinger_equation`
- (In French) Entretien avec M. Combescure, CNRS (2007) : **Physique quantique et information** (introduction)
- Numerov numerical method: `https://en.wikipedia.org/wiki/Numerov%27s_method`
- Lecture notes by J. Polonyi.

2 *Modelling road traffic*

2.1 Objectives

This project consists of simulating the evolution of road traffic using methods similar to those used in hydrodynamics. A macroscopic approach will characterize road traffic in terms of density and vehicle flows.

2.2 The problem set in context

Traffic modeling is an important societal issue. A great deal of research has been carried out on this subject for several decades and new models are still on the test bench in an attempt to optimise traffic organisation and in particular urban traffic (optimisation of the length of traffic lights in cities, etc.). There are two approaches to simulating road traffic. The first microscopic approach consists in simulating the behaviour of each motorist. Such an approach makes it possible to account for the diversity of vehicles and their drivers[1] as well as to take into account the interactions between drivers (adaptation of the distance between vehicles, lane changes, etc.). However, this approach is limited in terms of calculation time, particularly when dealing with a large number of vehicles or when integrating over large time intervals or long distances. A second macrosopic approach is to look at quantities with larger distance scales. In such an approach, only "average" behaviour is considered, not individual behavior.

Road traffic can then be modelled by equations that resemble hydrodynamic problems: road traffic behaves like a fluid whose density varies over time.

[1] As far as the vehicles are concerned, we will take into account characteristics such as length, power, braking capacity, ... As far as drivers are concerned, their reaction time will be taken into account, as well as their behaviour: changing lanes, use of secondary routes, etc ...

2.3 Modelling the traffic

To proceed, we need to define the relevant quantities to write down traffic.

Physical paramters

- Traffic density, defined at every point along the route:

$$\rho(x,t) = \lim_{\Delta x \to 0} \frac{N_{\Delta x}(x,t)}{\Delta x} \tag{2.1}$$

  This is *de facto* the average number of vehicles per unit length.

- The car flow, that is, the average number of cars per unit of time:

$$q(x,t) = \lim_{\Delta t \to 0} \frac{N_{\Delta t}(x,t)}{\Delta t} \tag{2.2}$$

The average speed of motorists at x at time t, is related to the two variables just mentioned by the relationship:

$$v(x,t) = \frac{q(x,t)}{\rho(x,t)} \, . \tag{2.3}$$

Equations

Just as hydrodynamics uses the principle of conservation of mass, so here we must respect the conservation of the number of cars. This leads us to the following equation:

$$\frac{\partial \rho}{\partial t} = -\frac{\partial q}{\partial x} \tag{2.4}$$

To solve this problem with two unknowns, we need to introduce a new relationship to express q as a function of ρ: we talk about **fundamental diagram**, which is equivalent to a state equation in thermodynamics.

There are more than twenty basic schemes that can be grouped into two main categories:

a) **First-order Models:**
The flow depends only on density: $q(x,t) = q(\rho(x,t))$ = cρ(x,t) where c is not necessarily a constant, but can change with x and/or time t.

There are several possible first-order models, here are some of them:

$$q(\rho) = v_M \rho(x,t)(1 - \frac{\rho}{\rho_M}) \tag{2.5}$$

$$q(\rho) = v_M \rho(x,t) e^{-\frac{\rho}{\rho_M}} \tag{2.6}$$

$$q(\rho) = v_M \rho(x,t) e^{-\frac{1}{a}(\frac{\rho}{\rho_M})^a} \tag{2.7}$$

where v_M is the maximum speed and ρ_M is the maximum density. The maximum speed corresponds to the type of road considered

(city, highway, freeway, ..). The maximum density corresponds to a traffic jam situation and therefore depends on the average distance between vehicles when they are stopped. The treatment is then of the type *transport* (advection).

b) **Second-order Models** :
The flow now depends on the density and its derivative:

$$q(x,t) = q(\rho, \frac{\partial \rho}{\partial x}) \ .$$

We end up with a *transport-diffusion* equation in this case. We may use the relation

$$q = c\rho - \alpha \frac{\partial \rho}{\partial x} \tag{2.8}$$

Depending on the model, α may be a constant (disturbance propagation rate) or some function of ρ.[2]

[2] For more information, please refer to page 25 of the MSc Thesis (in french) by G. Costeseque at this **url**.

To model traffic, you need to know the expression $q(rho, t)$. You can search for other, more elaborate models in the literature if you wish.

The equation can then be re-expressed as an equation at the first degree of density, *rho*, using partial derivation:

$$\frac{\partial \rho}{\partial t} = -\frac{\partial q}{\partial \rho}\frac{\partial \rho}{\partial x} \tag{2.9}$$

Example

As an example, we will adopt the fundamental diagram of the equation (2.5). We then obtain the partial derivation

$$\frac{\partial q}{\partial \rho} = v_M \left[1 \times (1 - \frac{\rho}{\rho_M}) + \rho \times (-\frac{1}{\rho_M})\right] = v_M \left[1 - 2\frac{\rho}{\rho_M}\right] \ .$$

Injecting this expression into the differential equation (2.9) results in a transport-type equation for the density; we find

$$\frac{\partial \rho}{\partial t} = -v_M \left[1 - 2\frac{\rho}{\rho_M}\right] \times \frac{\partial \rho}{\partial x} \, . \tag{2.10}$$

We must then adopt a numerical scheme to integrate over time the density ρ (and also the flow, q).

2.4 *Numerical scheme*

First, we discretize the space in N intervals of length Δx as shown in figure 2.1. The scales of distances considered for the study of the problem must be large in relation to the size of a vehicle, typically

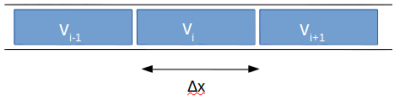

Figure 2.1: Illustration of the discretization of a stretch of road.

from a few tens to a few hundreds of meters. It's a configurable parameter.

We will consider the evolution of the system by discretizing the time in Δt intervals. The time step Δt must be less than the characteristic time it takes a vehicle to travel Δx. It is also a configurable parameter. What is your choice of parameters, expressed as the Courant-Friedrich-Levy (CFL) inequality?

We will use the **Godunov scheme** to solve for that problem. We will adopt the change of variables for a discrete representation of the continuous quantities ρ, q: $\rho \to u$, and $q \to f$ to respect conventions often encountered in literature.

Godunov integration method

First of all, initial conditions have to be set, *i.e.* know the values of the average velocity and density in each cell. Godunov's scheme allows the system to evolve over time in small steps Δt. For a given time, we determine the density values in each interval of the route:

$$u_i^n = \rho(i\Delta x, n\Delta t)$$

Use the following expression for the discrete form of the basic equation (2.4):

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{\Delta x}(f_{i+1}^n - f_i^n) \tag{2.11}$$

(Note the syntax with index for the f flow, identical to the one for the density, u). This expression to the right of the equality for the f gradient depends on the chosen scheme: we give here the form *forward in space*.

In the case of an approach to first order, according to the equation 2.5, we would obtain from (2.11) the expression :

$$u_i^{n+1} = u_i^n - v_M \times (1 - 2\frac{u_i^n}{\rho_M}) \times \frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n) \tag{2.12}$$

Exercise

In the case of a second-order approach where we have the equation:

$$q = v_M \rho(x,t)(1 - \frac{\rho}{\rho_M}) - \alpha \frac{\partial \rho}{\partial x} \tag{2.13}$$

Show that it's possible to use the following numerical scheme:

$$u_i^{n+1} = u_i^n + \alpha \times \frac{\Delta t}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n) - v_M \times (1 - 2\,\frac{u_i^n}{\rho_M}) \times \frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n) \tag{2.14}$$

Once the velocity v_M is known, along with all the integration parameters, we will be able to trace back the density u and the flow f.

Workplan

To solve the problem, it is necessary to specify the boundary conditions at the ends of the stretch of road (ρ and q at these points) and initial conditions (knowledge of ρ and v when t=0) at any point along the route.

Road map and traffic conditions

The problem can be dealt with in a number of contexts, starting with the simplest. Here are some examples of possible "routes":

- A closed car racetrack (boundary conditions: $\rho(0,t) = \rho(L,t)$)
- A stretch of motorway
- ascending / descending paths, roadwork ..

As far as boundary conditions are concerned, we can look at the following cases:

- inflow increasing over time
- with increasing inflow and congested traffic
- 'red-light' start at one end, other scenarios, ..

Results and analysis

You can illustrate the behaviour of the numerical method graphically as a function of the chosen parameters and the model under consideration. You will be able to represent the evolution of road traffic characteristics as a function of time.

Bibliography and external links:

- The Wikipedia entry for road **traffic models**
- HDR Thesis by M. Goatin (French/English) at `https://tel.archives-ouvertes.fr/file/index/docid/765410/filename/hdrGoatin.pdf`

To go further ..

- You should process different fundamental diagrams and compare them.
- You can also try to deal with non-homogeneous cases, i.e. when there are changes in maximum speed on a circuit or when the number of lanes evolves (roadwork, ..).
- Beware: non-homogeneous cases may be handled incorrectly by the Godunov scheme, so you can explore its limitations. If you wish, you can also search the bibliography for more advanced schemes.
- etc

3 Spectral analysis with Fourier transforms

3.1 Overview of principles

THE SUBJECT TAKES UP the theoretical notions of the Fourier transform as seen in mathematics and analytical physics classes. We will aim to implement these codes in their current state: solving the Poisson equation on a grid with one (1D) or two (2D) dimensions, with mesh = n or $n_x \times n_y$, respectively.

Material provided

The Python FFTW module PYFFTW can be installed with the *pip* utility ; or, you may download and install it from the github repository `https://github.com/pyFFTW/pyFFTW`. Examples how to use the package are given on the github page, and a notebook FFTW_EX.IPYNB is also available on the Ernest/moodle and Slack website for this course.

Results / report

You will save your results/programmes on ENT in a directory called *M1-Projects*. There you will store an archive of your source programmes and results; the compressed archive, however, cannot exceed 50 MB of memory space.

3.2 Theory of signal spectral modes

Formulas, definitions: Fourier space

We imagine a continuous curve in space, $h(x)$, and we will limit the analysis to one dimension for simplicity. First of all, note that the integral I in (3.1) below of the product of two cosines, covering the whole space, cancels (I = 0) if $k \neq k'$, and $I = 1$ if $k = k'$. A similar result also applies to sine functions. Trigonometric functions are said to be orthogonal to the integral over the real domain.

$$
\begin{aligned}
I &\equiv \lim_{\Delta X \to \infty} \frac{1}{2\Delta X} \int_{-\Delta X}^{+\Delta X} \cos(kx)\cos(k'x)dx \\
&= \lim_{\Delta X \to \infty} \frac{1}{2\Delta X} \times \left[\frac{\cos(k+k')x}{k+k'} + \frac{\cos(k-k')x}{k-k'} \right]_{-\Delta X}^{+\Delta X} \\
&\leq \lim_{\Delta X \to \infty} \frac{1}{2\Delta X} \times \left[\frac{2}{k+k'} + \frac{2}{k-k'} \right]_{0}^{+\Delta X} = 0
\end{aligned}
\tag{3.1}
$$

Consequently, the trigonometric functions can be used as a (orthogonal) basis for the reconstruction of any continuous function over the entire x-domain:] - infinite, + infinite [.

The transform of $h(x)$ is a function $H(k)$, written in k wave-number space, or, *Fourier space*, such that

$$
h(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} H(k) e^{-ikx} dk \;\Leftrightarrow\; H(k) = \int_{-\infty}^{+\infty} h(x) e^{ikx} dx \; . \tag{3.2}
$$

with $\exp(ikx)$ the Euler formula for trigonometric functions; remember that the wave-number $k = 2\pi/\lambda$ (which explains the factor $1/2\pi$ in front of the integral for $h[x]$).

Convolution theorem

The *convolution* of two functions h and g is the integral: convolution(g,h) =

$$
\int_{-\infty}^{+\infty} g(u)h(x-u)du \equiv g * h(x) \tag{3.3}
$$

The convolution theorem says that the convolution of two functions in space is the product of their transformation in Fourier space:

$$
g \star h(x) \Leftrightarrow G(k)H(k) \tag{3.4}
$$

The convolution of a pair of functions (g,h) in real space allows us to express this integral in Fourier space: the solution we are looking for is obtained by an inverse transform from Fourier space to real space.

Application to the gravitational field of a self-gravitating system

An example to illustrate these ideas is the standard equation of sources (electrostatic or gravitational) whose distribution in space[1] gives rise to a potential at any point $\mathbf{x}$. If the distribution of sources is denoted $S(\mathbf{x})$ and the potential it gives rise to, $\Phi(\mathbf{x})$, then the pair must satisfy the Poisson equation

$$
\nabla^2 \Phi(\mathbf{x}) = S(\mathbf{x}) \; . \tag{3.5}
$$

[1] This can be a space of any dimension, from 1 to 3: the coordinates define a *vector space*.

The problem is to solve Poisson's equation for a given source distribution. These are distributed on a Cartesian grid (of constant h-pitch) and we will limit ourselves to one dimension for simplicity.

For the case of a mass distribution in space we will have (for a density ρ)

$$S(\boldsymbol{x}) = 4\pi G\rho(\boldsymbol{x}) \; . \tag{3.6}$$

For numerical programming, it is conventional to set the gravitational constant G = 1 (a specific choice of gauge).

Green's function

The Green function $G_r(\mathbf{x})$ is a weight function that allows the desired integral to be grasped; by posing

$$G_r(\mathbf{x}) \equiv -\frac{1}{4\pi}\,\frac{1}{|\mathbf{x}|} \tag{3.7}$$

It can be shown that the Laplacian, applied to this function, leads to Dirac's "delta" function,

$$\nabla^2 G_r(\mathbf{x}) = \delta(\mathbf{x}) \; . \tag{3.8}$$

Reminder: The Dirac function allows you to write $\int f(\mathbf{x})\delta(\mathbf{x} - \mathbf{x}_o)d^n\mathbf{x} = f(\mathbf{x}_o)$. The Dirac function therefore makes it possible to evaluate the integrator at any point in the integration domain.

By searching for the solution for $\phi(\mathbf{x})$ in the form of a convolution $G_r \star S(\mathbf{x})$, i.e.

$$\Phi(x) = \int_{-\infty}^{+\infty} G_r(x-u)S(u)\,du \tag{3.9}$$

and this solution must satisfy Poisson's equation

$$\nabla^2\Phi(x) = \int_{-\infty}^{+\infty} \nabla^2 G_r(x-u)S(u) + G_r(x-u)\nabla^2(S[u])\,du = S(x) + \int_{surface} G_r\nabla S\cdot d\mathbf{v} = S(x) \tag{3.10}$$

where the last equality in (3.10) is obtained by applying Gauss's theorem and by admitting that the gradient of the sources, evaluated on the surface bounding the integration volume, tends towards zero. In one dimension, the surface is interpreted as a point infinitely far from the origin. Note that Green's function decreases as $1/|\mathbf{x}|$ and itself tends to zero in amplitude.

Convolution and resolution

The convolution theorem makes it possible to write[2]

[2] The sub-script k indicates the Fourier space.

$$\Phi_k(\mathbf{k}) = G_{r,k}(\mathbf{k}) \cdot S_k(\mathbf{k}) \,, \tag{3.11}$$

which are all complex quantities in principle. The potential $\Phi(\mathbf{x})$ is then found by means of an inverse Fourier transform. In this last step, the choice is made by default to normalize the potential such that

$$\lim_{|\mathbf{x}|\to\infty} \Phi(\mathbf{x}) = 0. \tag{3.12}$$

3.3 *Worksheet, objectives*

C++ programming & the FFTW library: compilation, linking, execution

If you are comfortable with the C++ style of programming, then you may write the core of your programmes in C++ and use Python wrapper functions to interface it with the integrated work-environment and plotting methods.

The FFTW3 library quickly implements Fourier transforms according to Tukey and Cooley's fast algorithm based on the descent of a binary tree. The official site of this public library is

HTTP://WWW.FFTW.ORG/ .

You will find links to detailed documentation, should you wish to have more information on the package (and/or the latest version). The actual fftw.3.3.2 FFTw.tar archive file can be downloaded from the Ernest website.

Use one of the gcc compilers (c++, c++-9.3 or 10) under Ubuntu to compile the test codes of the FFTw.tar archive that comes as project support material.

Python programming, numpy module

The fftw3 library is already integrated to the NUMPY module of Python. Note that older version of Python3 come with a package called SCIPY which may instead contain the fft modules. If you choose to code in Python, you are still bound to meet the expectations of validations and tests suggested below. You may also develop classes (with attributes, methods, ..) to showcase your command of object-oriented programming.

We suggest that you use the pyFFTW module presented in the introduction, as it is closer in performance to the best. optimised version of the FFTw library (written in C originally).

Work programme

1. **Phase 1**. Getting around

   To get your hands dirty, use the example codes provided (in C++ and in the Jupyter Notebook FFTW_ex.ipynb) and reproduce the data obtained for 1d, 2d (simple signals) and then to solve the Poisson equation on a grid (see the code Poisson.c for the C++ implementation). Feel free to rewrite these codes as needed, or to port them to Python (the algorithm is the same in any case).

   For C++ programming, don't forget to check the links to the .h files (function headers) needed by the program: the file fftw3.h is local, and is included in the archive. Three examples of use of the library (single.c, single2d.c and Poisson.c) have been compiled and executed (see ".log" files in the archive).

2. **Phase 2**. Adapting the codes, performance

   To appreciate the performance of the "FFTW" algorithm, it is necessary to modify the parameters and function calls to take the runtime in relation to the size of the grid (or mesh) used. You should focus on two-dimensional configurations, starting either with the code signal2d.c or Poisson.c. Using the time functions of standard libraries, to obtain a scaling law of the computing time as a function of the net size of the grid (nx x ny). The grid will therefore have to be modified. The POISSON.C code must allow to calculate the gravitational potential of N mass elements, the sources: a classical summation algorithm would require $\sim N^2$ operations. A key question is: What is the advantage of using the FFT algorithm? In practice, the grid points do not coincide with the position of the N sources: explain how to reconstruct the density profile (we are talking about the S signal here) from the (vectorial) position of the sources. Is this algorithm a hindrance to the performance of the FFT transform, does it slow it down? Discuss the scaling of runtime with the number of sources, N.

3. **Phae 3**. Dynamics of moving bodies

   When we want to integrate the orbit of a celestial body, we have to integrate a system of two differential equations in time :

$$\begin{aligned} \dot{\boldsymbol{x}} &= \mathbf{v} \\ \dot{\mathbf{v}} &= -\boldsymbol{\nabla}\Phi(\mathbf{x})\,. \end{aligned} \tag{3.13}$$

   To calculate the gradient of the potential, one may choose between two possibilities:

a) Calculate the potential by FFT and then find the gradient again, for example by means of a finite difference diagram throughout the grid;

b) Compute the Fourier transform of the gradient itself.

Make a choice between either method (a) and (b) and implement a time-integrator such as one that was developed during the first semester (Eulerian, Runge-Kutta, ABM, ..).

Going further, auto-coherence

Using the source codes provided, program one or other of the methods discussed in the previous sub-section for the case of distribution (of sources S) respecting axial symmetry around the origin. You can limit yourself to the integration of a *single* orbit (N = 1) but at least discuss the general case where N is chosen by the user and can be large: this is the case of gravitational sources on the scale of galaxies, for example.

What is missing in your program to make it self-coherent? Indicate what needs to be added to the programme to make it coherent with the Poisson solution. Is there a relationship between the number of sources N and the grid dimensions $n_x \times n_y$? Let the user specify the number of elements N, the initial conditions (positions $\boldsymbol{x}$ and velocity $\boldsymbol{v}$) of each element, and the total integration time. Define a quantity (such as angular momentum or mechanical energy) that should be conserved throughout integration time. Experiment with the number of sources and your grid parameters to check how well these quantities are conserved in practice.

3.4 References

Here are references that you may find helpful:

- Binney, J. & Tremaine, S. 2008. Galactic Dynamics (Princeton: Princeton University Press)
- Wikipedia : The fast Fourier transform

4 Hamiltonian Chaos in Galaxies

4.1 Overview of principles

The motion of stars in galaxies is governed by gravity. In the weak field limit, the Newtonian gravitational field is conserved which allows for a significant simplification of the analysis of their motion (owing to constraints such as the integrals of motion).

In this project, we will explore the transition from a highly ordered configuration of the potential, when most orbits are regular and nearly-closed orbits, to highly irregular, **chaotic** orbits that come to be an important contribution to the dynamics as a whole.

Figure 4.1: Examples of two galaxies: one with a near-axisymmetric disc (far left), the other with a strong $m = 2$ barred mode. Credits HST Heritage Collection.

4.2 Equations

The equations of motions are those of classical mechanics

$$\frac{d}{dt}\mathbf{v} = -\nabla\Phi(\mathbf{x}, t) \tag{4.1}$$

$$\frac{d}{dt}\mathbf{x} = \mathbf{v}\,. \tag{4.2}$$

To make the situation more concrete we will work in the local Universe, at low cosmological redshift, when galaxies have acquired all their mass and are in quasi-steady equilibrium. The potential will be taken a fixed in a first instance.

Potential

Most galaxies are made of a flat disc, with central bulge of stars. The ratio of bulge to disc mass may vary wildly, yet normally one would have that the disc dominates the global potential. One of the main characteristics of this potential is that it will lead (asymptotically) to a velocity field of near-constant celerity. In other words,

$$\lim_{r\to\infty} v_c^2 = r\nabla\Phi = \textit{constant}\,, \tag{4.3}$$

and this is robust against the morphological details of the galaxy. A potential that can recover this property yet is simple to handle is called **the logarithmic** potential, which takes the mathematical form

$$\Phi(\mathbf{x}) = \frac{v_o^2}{2}\ln\left|\frac{x^2+\left(\frac{y}{b}\right)^2+\left(\frac{z}{c}\right)^2}{r_c^2}\right| \tag{4.4}$$

In that definition we have introduced a reference unit of length (r_c) along with dimension-less numbers $c \leq b \leq 1$. Clearly the limit $c \to 0$ gives rise to a flat distribution of matter in the x-y plane; we are interested in three-dimensional motion. The mass profile of the galaxy gives rise to a volume density $\rho(\mathbf{x})$ which satisfies Poisson's equation

$$\nabla^2\Phi = 4\pi G\rho(\mathbf{x})\,. \tag{4.5}$$

Initial conditions

The set of free parameters $\{b, c\}$ define the morphology of the potential : when $c < b = 1$, the potential admit a symmetry with respect to the z-azis ; hence in steady-state the Hamiltonian is a conserved quantity as is the z-component of the angular momentum,

$$\mathbf{L}_z = \mathbf{R}\times\mathbf{p}_\theta$$

where we have defined R the cylindrical radius, θ the ascending angle and z the third dimension of cylindrical coordinates. These will be useful when treating a thin galactic disc: in observations of galaxies, it is often found that the ratio of characteristic vertical- to radial scale lengths is less than $1/20$; this is therefore a control value for c.

To integrate orbits in the potential (4.4) any combination of phase-space coordinates $(\mathbf{x}, \mathbf{v})$ are possible, however bound-states are those for which the **total binding energy** E of a star is negative, where

$$E = \tfrac{1}{2}\frac{\mathbf{p}^2}{m} + \Phi(\mathbf{x}) = \tfrac{1}{2}m\mathbf{v}^2 + \Phi(\mathbf{x}) < 0\,. \tag{4.6}$$

Hence it is a good idea to fix the range of binding energy at given space coordinates $\{\mathbf{x}\}$, and control the quality of the orbit passing through that point by attributing, firstly, a value of energy E to an orbit, and then, secondly, an angular momentum component.

Orthogonal start space

To reduce the search for orbital initial configurations, we appeal to the continuity equation and say that any point in space can be visited by an orbit - as long as there is no singularity anywhere in the system. So there is an enormous freedom in choosing the orbits to integrate.

One approach that has a lot of success is called the **orthogonal start**, *i.e.* orthogonal velocity vector with respect to one of the main axes of the potential. Since we have adopted (implicitly) the x-axis to coincide with the semi-major axis of the potential (4.4), we may use the x-coordinate as free parameter and set $z = y = 0$ initially. The start conditions at $t = 0$ are then

$$\text{start conditions: } x = x_o; y = z = 0;\ v_x = 0;\ v_y, v_z \neq 0\ . \tag{4.7}$$

We can of course express these conditions in cylindrical coordinates,

$$\text{(Cylindrical) start conditions: } R = x_o; \theta = z = 0;\ v_R = 0;\ v_\theta, v_z \neq 0\ . \tag{4.8}$$

Either way, a good reference for the attribution of velocity vector components is to compute the angular momentum of the instantaneous circular orbit at the start location, $\mathbf{L} = \mathbf{R} \times \mathbf{v}_c$, where the circular velocity is defined in (4.3). Since the energy $E < 0$ is fixed, it will be possible to determine whether the orbit traced by the star is elongated and "bar-line", or rather circular or "loop-like", by comparing its angular momentum components to the reference circular motion. Clearly exactly circular orbits would require $b = 1$ at least.

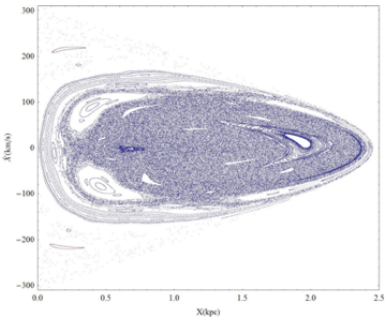

Figure 4.2: The figure shows a Poincaré section in phase-space axes of x and $\dot{x}$. A large sea of chaos is clearly visible, together with circumscribed islands of regular motion. Credits T. Chatzopoulos.

4.3 *Project work sheet*

Figure 4.2 should help get some orientation into the problem. The figure shows a cross section in phase-space, located at $y = 0, z > 0$. If we recall that the orbital coordinates are six-dimensional, then using integrals of motion such as L_z and the Hamiltonian $H = E$ together with the constraints on y and z leaves exactly two degrees of freedom to play with, say x and $\dot{x}$. Thus if there exists a third integral of motion, unknown to us in analytical form, then correlations will appear

between x and $\dot{x}$, giving rise to **islands of regularity** between the two quantities. In other words, regular orbits will belong to a family of attractors in that plane, and they will describe regular patterns over a small area. If one identifies such an island, then it is possible to **fine-tune** the orbital initial conditions so as to recover a single point in the $x - \dot{x}$ plane: we have now identified a **closed orbit**, *i.e.* the orbit always crosses the plane at the same coordinates, and we may say that this orbit is the **shephard** "on the island".

The purpose of the project is therefore to explore how many orbits gravitate around islands of stability, and how many end up in a sea of chaos. That, in relation with the global morphological parameters b, c.

Methodology

The potential being a given, we require an integration technique to recover the orbits. One may try low order integrators such as the Runge-Kutta (4th order) or the simpler but simplectic 2nd order Verlet integrator (a.k.a. *leap-frog* or *saute-moutons*). One may easily adapt a code developed during the semester to do this, or download a library from the specialised website (such as some GitHub entries).

Having a good integrator it will then be necessary to develop a package to construct a phase-space Poincaré section such as shown on Fig. 4.2. Note that the key issue is that the coordinates should be refined to improve accuracy, since in general the time-stepping will be too large to intercept the $y = 0$ axis, for example.

Procedure, strategy

1. Start with the simplest, spherical potential to calibrate the integration procedure and control your boundary conditions: identify and recover integrals of motion, closed orbits for that case ;

2. modify the shape of the potential only one parameter at a time, starting with c to recover a thin axi-symmetric disc ;

3. Try and maintain a code that is as modular as possible using function calls, separate text files, classes and methods ;

4. Use the Unix **make** utility to allow target compilations of the code ;

5. Check your equations and make sure that you choose integration parameters that are suited to your problem (the choice of timestep will be crucial) ;

6. If the procedure is well tested, shift to a **rotating frame** of constant angular speed Ω. In this new, non-inertial frame, the potential pattern remains constant but now fictitious forces imply that H is replaced as a conserved quantity by the Jacoby integral J.

Going further ..

There are other potentials that actually are better suited to describe galaxies: you may want to compare your results for the logarithmic potential with those for the **Miyamoto-Nagai** potential, which introduces a central "bulge" - similar to galaxies - but remains axisymmetric;

Explore the constraints on the vertical motion that is allowed on orbits that should remain confined to a small vertical height (set by c);

Try to think of a way to recover a relation between the orbits that you have integrated, and the density profile that should give rise to the potential: how would you check for self-consistency ? You may want to look at **quadratic programming** methods if you want to explore this.

Suppose you have a telescope to observe your galaxy, equipped with a spectrograph. How would you reconstruct the signature of a "chaotic" velocity field, what might be its signature?

4.4 References

Here are references that you may find helpful:

- Binney, J. & Tremaine, S. 2008. Galactic Dynamics (Princeton: Princeton University Press)
- Wikipedia : Poincaré maps
- Miyamoto-Nagai potential : see the Python Galpy package at `https://github.com/jobovy/galpy`

5 Implementation of your own MC simulation

5.1 Objectives

The project consists of setting up your own simulation based on Monte-Carlo methods. A simulation of the experiment SiTrInEO will be implemented, as well as particle-matter interactions within detectors.

Material provided

No special material is needed for this project, everything can be done from scratch.

Results / report

You will save your results/programmes on ENT in a directory called *M1-Projects*. There you will store an archive of your source programmes and results; the compressed archive, however, cannot exceed 50 MB of memory space.

5.2 Overview

Introduction

Simulation plays an important role in research activities ; it can be used for studying feasibility of a project, for comparing theoretical model with observations, for correction of real data, etc. They are handled by software, usually developed by huge collaboration, with a large number of dependencies. Learning how to use it may then be a complicated process, and may not be worth it especially when the wanted simulation is actually quite simple. In this case, it would be more time-efficient to develop his own simulation. That is the goal of this project.

Simplified Monte Carlo (MC) simulation made to test rapidly some features are usually called toy-MC. Here, we propose to make such a simulation of a real experiment : the SIlicon TRacker with

International Education Objectives (SiTrInEO). The objective of that table-top tracker is to reconstruct trajectories of charged particles and measure their momentum. This is why it is composed of two pairs of silicon detectors separated by a magnetic field (see Fig.5.1) : when the charged particle enters inside the magnetic field region, its trajectory will curve and from the bending is extracted the momentum. The bending can be measured by comparing the direction of the track before and after the magnetic field thanks to, respectively, the first and last two layers.

In this project, the students will have the opportunity to set up a toy MC simulation of that experiment and use it to reconstruct the momentum of the particle of their interest (e, p, ...).

Assumptions

In order to facilitate the implementation, some restrictions will be imposed :

- For the sake of simplicity, the simulation will be done only in 2D, as presented in Fig.5.1.
- The experiment is not done in vacuum but in the atmosphere. However, the air will be neglected.
- Whatever the particle, it will be considered as point-like.
- Originally, the particles to be measured are electrons emitted by a radioactive source in the MeV range. Here, the particles can be electrons, protons, kaons,... emitted by a particle gun in extremely narrow momentum window (for example, a Dirac peak around 2 MeV).

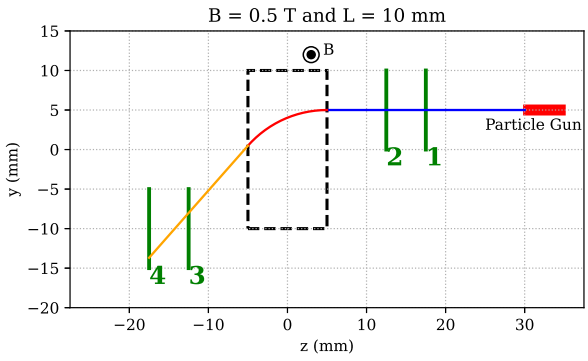

Figure 5.1: Simulation of the SiTrINEO experiment. Here, this is the track of a 2 MeV electron.

Concepts of tracking

The tracking consists in reconstructing the momentum of charged particles based on their hit positions on the sensors. If no magnetic field is applied, particle trajectories would be straight lines whatever their momentum ; there would be no way to distinguish a 1 MeV track from 10 GeV one. For this exact reason, a magnetic field is applied : the bending is different depending on the track momentum.

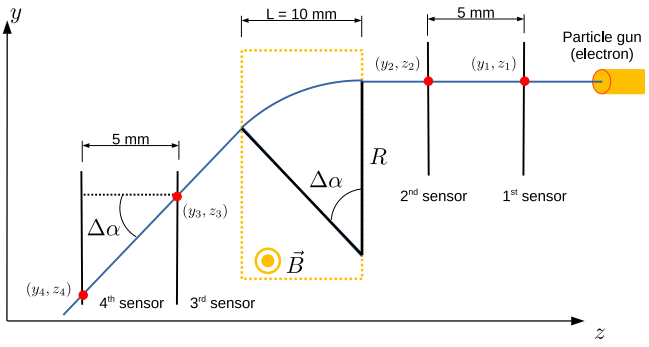

Figure 5.2: Schematic of the SiTrINEO project. Here, the angle α in the subject is $\Delta\alpha$ on the figure.

Generally in particle physics experiments, the generated magnetic field is homogeneous in order to maximise the detector acceptance. In such field, the motion of a charged particle is determined by the Lorentz force :

$$\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} = q\mathbf{v} \times \mathbf{B} \tag{5.1}$$

One can show quickly from eq. 5.1 that the momentum is simply the product of the charge q, the magnetic field B and the curvature radius R :

$$p = qBR = \frac{qBL}{\sin\alpha} \tag{5.2}$$

Be careful with the units : p is in GeV/c, $q = 0.3$ for a particle with a charge of 1.6×10^{-19} C, B is expressed in Tesla and L is in m.

Since $\sin\alpha = L/R$, the momentum can be expressed using the deviation angle α and the width of the magnetic field region L. The last expression of the momentum is quite convenient because α can be easily extracted from the hit position on the four sensors. Since the track is perpendicular to the two first detection layers, α is given by :

$$\alpha = \tan^{-1}\left(\frac{y_4 - y_3}{z_4 - z_3}\right). \tag{5.3}$$

Pixel firing threshold

The silicon detectors in this experiment are CMOS sensors – as in smartphone's camera – with a thickness of 14 μm. These sensors are

constituted of pixels, that will be fired – meaning activated – each time an ionizing particle traverses it. Ionizing particles may come from cosmic tracks, natural radioactivity,... so the pixels are always activated. To reduce this background, a threshold is applied : if the energy deposited is not larger that a certain value, the pixel is not fired. The energy deposited is evaluated in number of electron-hole pairs created and, in the experiment, the typical threshold is about 100 e^-.

The energy deposition of a charged particle is stochastic ; in detectors of moderate thickness, it follows a Landau distribution, whose most probable value Δ_p and width w are defined such as :

$$\Delta_p = \xi\Big[\ln\frac{2mc^2\beta^2\gamma^2}{I} + \ln\frac{\xi}{I} + j - \beta^2 - \delta(\beta\gamma)\Big] \quad ; \quad w = 4.018\xi \quad (5.4)$$

where $\xi = (K/2)\langle Z/A\rangle(x/\beta^2)$ with x the detector thickness in g/cm^2, I is the average excitation energy, $\beta = v/c$, $\gamma = 1/\sqrt{1-\beta^2}$ the Lorentz factor, c is the speed of light, m is the mass of the incident particle, $j = 0.200$ and δ is the density effect correction. For silicon detectors, $\langle Z/A\rangle = 0.49930$, $\rho = 2.200$ g/cm^3, and I is given by :

$$I = \begin{cases} 12 + \frac{Z}{Z}[\text{eV}] & (Z < 13) \\ 9.76 + 58.8Z^{-1.19}[\text{eV}] & (Z \geq 13) \end{cases} \quad (5.5)$$

Finally, one can extract the number of electron-hole pair created by computing the ratio of the energy deposited by the minimum energy required to create an electron-hole pair(W_i). For silicon, W_i = 3.6 eV.

So, there can be cases where a track traverses a sensor but the energy deposited is not enough to trigger the pixel. In this case, particle momentum can not be evaluated because one of the four necessary point is missing.

Multiple scattering

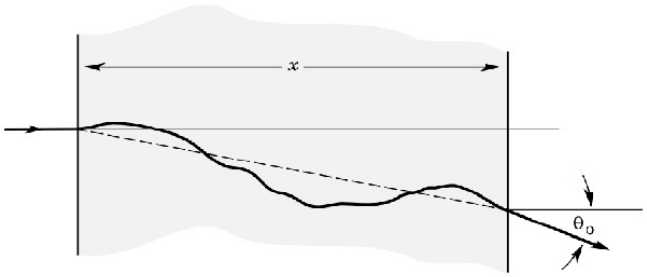

Figure 5.3: Illustration of multiple scattering on a material of thickness x.

One effect that greatly influences the tracking is multiple Coulomb scattering. Charged particles traversing a medium are deflected by

many small angle scatters ; most of them are due to Coulomb scattering from nuclei, hence the fact that this is called multiple Coulomb scattering. In order to be as close as possible to the real experiment, this effect should also be integrated in the simulation.

The multiple scattering is very well described by the Molière theory. In first approximation, the angular distribution follows a centered gaussian of width θ_0, with θ_0 is given by :

$$\theta_0 = \frac{13.6 \text{ MeV}}{\beta c p} z \sqrt{x/X_0} \left[1 + 0.038 \ln\left(x/X_0\right)\right] \tag{5.6}$$

where p, βc, z are the momentum, the velocity and the charge number of the incident particle. x is the thickness of the material and X_0 is the radiation length. In our case, the detectors are mainly composed of silicon, whose radiation length is about 27.05 g/cm^2.

5.3 *What you should do and paths of exploration*

To start with, you should start implementing a simulation similar to the one in Fig.5.1. This is an ideal case (all the particles have the same energy, no energy loss) so at this stage, no random number generation is necessary. Momentum reconstruction can already be introduced in the simulation.

Once this is done, one can start to implement the pixel firing threshold and the multiple coulomb scattering, as explained in previous sections. These are stochastic phenomena, so random number generation should intervene here.

Finally, students can use the simulation to study...:

- ...the momentum spectra for different energies and type of particles.
- ...the limit of momentum reconstruction in the conditions of Fig.5.1.
- ...the impact of multiple scattering as a function of the particle energy.
- ...the influence of a small opening angle on the particle gun.
- ...the influence of energy spectra of the particle gun (for example, switch from a perfect Dirac peak to a Gaussian).

5.4 *References*

- SiTrInEO web site : https://sites.google.com/site/sitrineo/
- Particle Data Group : https://pdg.lbl.gov/2021/reviews/contents_sports.html

6 (Nuclear) missile interception

6.1 Objectives

The goal is to implement a game based on adjustments from a χ^2 fit and a Kalman filter. At the first order, fit can be realised using Python modules, but ideally, an implementation of these two approaches would be highly appreciated.

Material provided

For this project, a basic script for producing a Matplotlib animation is provided (see Fig.6.1). χ^2 fit and Kalman filter can be done using, for example, the modules Scipy.optimize and FilterPy.

Results / report

You will save your results/programmes on ENT in a directory called *M1-Projects*. There you will store an archive of your source programmes and results; the compressed archive, however, cannot exceed 50 MB of memory space.

6.2 Overview

Introduction

Some phenomena or physical processes, in order for them to be understood, may require modelling. The objective is to propose the most satisfying description with the fewest number of parameters. Once they all have been estimated – directly or indirectly from experimental measurements – the model can then be used to make predictions, which will be compared to experimental results. Depending on the outcome of this comparison, the model may be validated or invalidated. The core of this project is to estimate the value of the parameters of a given model, and use them to produce predictions.

Here is the scenario : your worst enemy considers that enough is enough, and decides to launch a (nuclear) missile targeted on your

position. There is no time to evacuate, the only escape is to intercept the missile on its course. Thankfully, you also have missiles. Your strategy is to sample (with some imprecision) the position of the missile, derive its trajectory, predict the impact point and launch a counter-measure missile. If the former manages to intercept the missile, you've succeeded ! If not, let's hope you can find a shelter...

The goal is to implement this game and visualize it using a MATPLOTLIB animation. The visualisation makes part of the project but not the animation. So students should not waste time debugging the animation ; if you encounter problems related to the animation, report it to the supervisor.

Assumptions

In order to facilitate the implementation, some assumptions will be made :

- The only force acting on the initial missile or the counter-measure is the weight, *ie* there are no losses.
- The missile follows a parabolic trajectory.
- Only one counter-measure may be launched.

The animation

The animation will be realised with the module MATPLOTLIB. The user should be able to select the initial conditions of the missile, meaning x_0, y_0, v_{x_0} and v_{y_0}. Once they have been entered, the animation should start. Two additional buttons should be present :

- *Measure* : if it is pressed, a measurement of the current position of the missile is performed. After a certain number of measurements, a fitting procedure will be launched automatically and the predicted trajectory of the missile should be plotted.
- *Launch* : once the user considers that the quality of the fit (and thus the predicted trajectory of the missile) is satisfying, he can press this button to fire the counter-measure ; its parameters have to be evaluated automatically based on the predicted trajectory.

If the counter-measure misses, the message "Game Over" is displayed.

The missile position measurement

As in real life, each measurement must be erroneous at some extent. Thus, errors have to introduced in our model, and this can be

achieved this way. The measured position will corresponds to the actual/analytical missile position shifted by a randomly generated number – which corresponds to the error –, assuming a centered Gaussian distribution of standard deviation σ. The numerical value of this parameter is to be defined by the students.

Concerning the uncertainty, to begin with, one may consider it as constant for all the points and equal to 1σ or 2σ. Later – if time allows it –, variable uncertainties could be introduced.

Evaluation of the trajectory

The trajectory of the missile is already known : by assumption, it follows a parabolic trajectory depending on four parameters (there is actually a fifth one, the mass of the missile, but it will be considered as fixed and known in the entire project). Based on experimental measurements, one can attempt to find the parameters minimizing the difference between the theoretical model and data points. This method, called a fit procedure, is the one we are going to follow in this project. Two procedures shall be investigated : a global and a local fit.

6.3 Global fit

Introduction

In the case of a global fit, the most used technique is the least square method. The popularity of this method comes from the Gauss-Markov theorem, which demonstrates that, in the case of a linear model, the least square method is the best unbiased estimator of the coefficients. To illustrate the estimation of the parameters with this method, let's consider a set of N data points (x_i, y_i) and with an uncertainty σ_i associated to each y_i. One wants to adjust these points with a function $y = f(x)$. We define the quantity χ^2 such as :

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{f(x_i) - y_i}{\sigma_i} \right)^2 \tag{6.1}$$

The optimal parameters are the ones minimizing the quantity χ^2. In the case of a function f of k parameters λ, trying to minimize of the χ^2 results in finding the values of λ_k for which the partial derivatives are equal to zero.

$$\frac{\partial \chi^2}{\partial \lambda_1} = 0 \tag{6.2}$$

$$\frac{\partial \chi^2}{\partial \lambda_2} = 0 \tag{6.3}$$

$$\vdots \tag{6.4}$$

$$\frac{\partial \chi^2}{\partial \lambda_k} = 0 \tag{6.5}$$

This methodology can be generalized to every function $y = f(x, \vec{\lambda})$, where $\vec{\lambda}$ corresponds to the set of parameters of the model. It is extremely important that the number of experimental data points is greater than the number of parameters of the model. For a given adjustment, a compatibility between model and experiment can be done : the minimized χ^2 should follow a χ^2 law with n degree of freedom where $n = N_{\text{points}} - N_{\text{parameters}} - 1$. There are methods to calculate the associated probability.

There exists plenty of algorithm to solve minimizing problems, and it's up to students to choose the algorithm. If time allows it, at least the conjugated gradient method should be implemented.

Conjugated gradient descent

This method is not only efficient to find the minimum of the gradient of the function, but it is also fast since it adjusts the step used to get the next estimation of the minimum : as a second step, the method uses the conjugated direction. The algorithm can implemented in the following way :

- Starting from a point $\vec{x_0}$, the gradient of the function at this point is calculated : $\vec{\nabla} f|_{\vec{x_0}} = \vec{p_0}$, where $\vec{p_0}$ is called the first direction of descent.
- In the $\vec{p_0}$ direction, the value of α_0 minimizing $f(\vec{x_0} + \alpha_0 \vec{p_0})$ is evaluated.
- The next value of the parameter vector is then defined as $\vec{x_1} = \vec{x_0} + \alpha_0 \vec{p_0}$.
- The new direction $\vec{p_1}$ can be calculated from the gradient $\nabla f|_{\vec{x_1}}$.
- From the two calculated gradients, one can compute their ratio to the square, β_1 :

$$\beta_1 = \frac{\| \nabla f|_{\vec{x_1}} \|^2}{\| \nabla f|_{\vec{x_0}} \|^2} \tag{6.6}$$

- The new direction, $\vec{p_2}$ is gien by : $\vec{p_2} = \nabla f|_{\vec{x_1}} + \beta_1 \vec{p_1}$.

- New values of direction and β are calculated by iteration :

$$\beta_{k+1} = \frac{\| \nabla f|_{\vec{x_{k+1}}} \|^2}{\| \nabla f|_{\vec{x_k}} \|^2} \tag{6.7}$$

$$\vec{p_{k+1}} = \nabla f|_{\vec{x_{k+1}}} + \beta_{k+1}\vec{p_k} \tag{6.8}$$

- Finally, the algorithm stops when

$$\frac{\| \nabla f|_{\vec{x_k}} \|}{\sqrt{1+ \| \nabla f|_{\vec{x_k}} \|}} < \delta, \tag{6.9}$$

  with $\delta = 10^{-8}$.

6.4 *Local fit*

Introduction

The global fit requires the entire set of data points in order for the procedure to be executed. On the contrary, the local fit adjusts the model parameters point by point (hence the expression *local fit*). A typical situation of the use of such an adjustment arises when the number of experimental measurements is not fixed, and more points may still be introduced in the data sample. Either one redo a fit from scratch with the global fit, or one keeps the previous adjustment and takes into account the new point with a certain weight. This is the foundation of the Kalman filters.

The Kalman filter is an iterative procedure estimating the state of a linear dynamical system. The filter is composed of three phases :

- **Prediction** : from the state at the iteration k given by the observations up to step $k-1$, predict the new state (state at the iteration $k+1$).
- **Update** : Compare the predicted state with the observation at $k+1$, and update the predicted state. Move to the new state $(k+2)$ by going back at the first phase.
- **Smoothing** : After n observations, re-execute the first and second steps backward, meaning from the n-th observation to the first one, in order to smooth/precise the estimation of the parameters.

The first and second phases are the core of the Kalman filtering. The third phase is not a mandatory part of the filter ; it is one of its advantage over global fit, though. The smoothing phase can be run as many times as needed, and this will ultimately lead to a more accurate estimate of the parameter.

Implementation

If you have time to implement the Kalman filter, ask the supervisor to provide you equations of the filter.

6.5 What you should do and paths of exploration

To get started, students should download the script *MissileInterpectionAnimation.py* in order to produce the animation presented in Fig.6.1. Inside the script, there are two empty functions associated to the buttons *Measure* and *Launch* ; implement them as described in the previous sections. Concerning the fit, you should start by using the methods already implemented in the module SCIPY.OPTIMIZE. If it works, continue with the implementation of the Kalman filter with the module FILTERPY.

One can already compare these two approach by looking at the :

- the precision of the adjustments, and
- the execution speed of the adjustments.

Finally, if there is still enough time, you can implement the χ^2 fit and the Kalman filter, as explained in the dedicated subsections.

6.6 References

Here are links that you may find useful :

- Documentation of SCIPY.OPTIMIZE : https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html
- Documentation of FILTERPY : https://filterpy.readthedocs.io/en/latest/

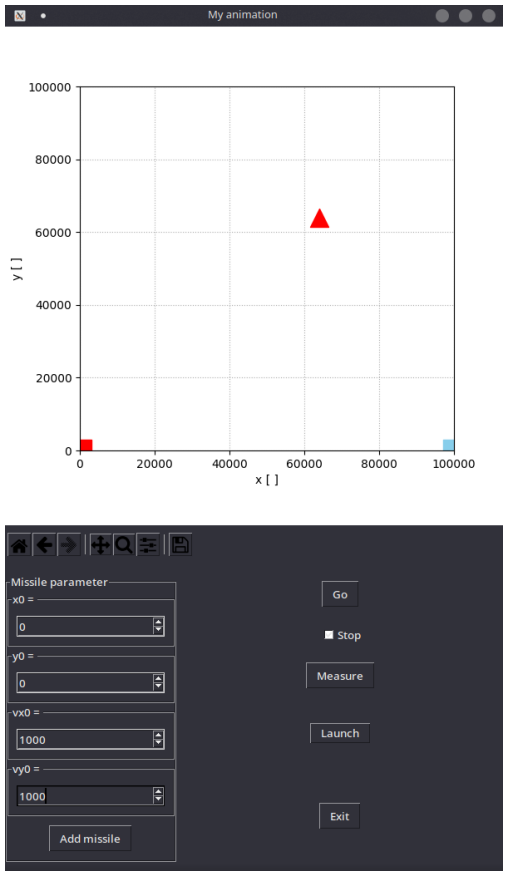

Figure 6.1: Screenshot of the animation produced by *MissileInterpectionAnimation.py*. The red triangle represents the missile, the red square is the enemy base and the blue square corresponds to your position, the targeted position of the missile.

Index