Faculté de physique et ingénierie
Université de Strasbourg

Université de Strasbourg

Numerical Simulation Project

Solving Poisson Equation with Fast Fourier Transform

Auteurs :
Laureen Barbiche
Amar Bellahsene
Enguerrand Decroix-Têtu
Léo Huber
Ahmed Kasri
Quentin Rossi

Tuteurs :
Christian Boily
Romain Schotter

M1 Physique Fondamentale et Appliquée

Année Universitaire 2021-2022

Table des matières

Introduction

A central concern of engineering mathematics involves the transformation of equations into a coordinate system where expressions simplify, decouple, and are useful to computation and analysis. The most fundamental coordinates transformation was introduced by Joseph Fourier in the early 1800's to investigate the theory of heat. Fourier introduced the concept that sine and cosine functions of increasing frequency provide an orthogonal basis for the space of solution functions. Indeed, the Fourier transform basis of sines and cosines serve as eigenfunctions of the heat equation, with the specific frequencies serving as the eigenvalues. Fourier's seminal work provided the mathematical foundation for Hilbert spaces, approximation theory, and the subsequent revolution in analytical and computational mathematics. Fast forward 200 years, and the Fast Fourier Transform (FFT) has become the cornerstone of computational mathematics, enabling real-time image and audio compression, global communication networks, modern devices and hardware, numerical physics and engineering at scale, and advanced data analysis. The fast Fourier transform has had a more significant and profound role in shaping the modern world than any other algorithm to date. As a motivation for this numerical simulation project, we were asking ourselves, in what extent FFT algorithm can be useful and efficient to compute a Gravitational Field on a $2D$ grid generated by a density of stars ? In this report, we will begin by explaining briefly the Fast Fourier Transform principle and then, present the different applications applying FFT made during this semester.

I- Fast Fourier Transform (FFT) Principle

I.1- Fourier Series and Fourier transform : theoretical aspects

Before describing the computational implementation of Fourier transforms on vectors of data, here we introduce the analytic Fourier series and Fourier transform, defined for continuous functions. Naturally, the discrete and continuous formulations should match in the limit of data with infinitely fine resolution.

Fourier Series

A fundamental result in Fourier analysis is that if $f(x)$ is periodic and smooth, then it can be written in terms of a Fourier series, which is an infinite sum of cosines and sines of increasing frequency. In particular, if $f(x)$ is 2π-periodic, it may be written as :

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos(kx) + b_k \sin(kx)$$

The coefficients a_k and b_k, which can viewed as the coordinates obtained by projecting the function onto the orthogonal cosine and sine basis, given by the inner products Here, the computation of functions using samples will be done by numerical integral calculations of these coefficients, using several possible methods (Newton-Cotes formulas). More frequencies we take in the summation more accuracy we get in the final function we are trying to recover.

Fourier Transform and application to PDEs and ODEs

The Fourier transform integral is essentially the limit of a Fourier series as the length of the domain goes to infinity, which allows us to define a function defined on $(-\infty, +\infty)$ without repeating :

$$\begin{cases} \tilde{f}(\omega) = \mathcal{F}\left(f(x)\right) = \displaystyle\int_{-\infty}^{\infty} f(x) e^{-i\omega x}\, dx \\ \\ f(x) = \mathcal{F}^{-1}\left(\tilde{f}(\omega)\right) = \dfrac{1}{2\pi} \displaystyle\int_{-\infty}^{\infty} \tilde{f}(\omega) e^{i\omega x}\, d\omega \end{cases}$$

The Fourier transform is particularly useful because of a number of properties, for example how derivatives of functions behave in the Fourier transform domain. These properties have been used extensively for data analysis and scientific computing to solve PDEs accurately and efficiently. In our case we will use to solve Poisson equation, and calculate the gradient of a 2 dimensions function.

Derivatives of Functions The Fourier transform of the second derivative of a function is given by :

$$\mathcal{F}\left(\frac{d^2}{du^2} f(u)\right) = -\left|\omega\right|^2 \mathcal{F}\left(f(u)\right)$$

This property, in our case, will be used to solve the Poisson equation for different functions.

Convolution theorem The convolution of two functions is particularly well-behaved in the Fourier domain, being the product of the two Fourier transformed functions. The theorem tells us that :

$$\mathcal{F}\left((f \star g)\right)(k) = \mathcal{F}(f)(k) \times \mathcal{F}(g)(k)$$

This property will be used to calculate the $2D$ Gravitational Potential by computing the FFT of the Mass Density and the Green Function.

I.2- Fourier Analysis in practice : From DFT to FFT

Until now, we have considered the Fourier series and Fourier transform for continuous functions $f(x)$. However, when computing and working with "real-data", it is necessary to approximate the Fourier transform on discrete vectors of data. The resulting discrete Fourier transform (DFT) is an algorithm that gives a discretized version of the Fourier series for vectors of data $f = \{f_1, ..., f_n\}$ obtained by discretizing the function $f(x)$ at a regular spacing Δx.The DFT is useful for numerical approximation and computation, but it has a poor time computation for very large number on which we make the sampling (for example in $2D$ a $N \times N$ matrix). It is actually a simple formulation that involves a computational time that goes as $O(N^2)$ operations.

Discrete Fourier Transform (DFT)

Although we will always use the FFT for computations, it is illustrative to begin with the simplest formulation of the DFT in order to understand the results given by FFT since, they give the same type of results using different algorithmic approaches. the DFT is a linear operator (a matrix) that maps the N data points in f to the frequency domain $\tilde{f}$ ($f_1, .., f_k, .., f_N \rightarrow \tilde{f}_1, .., \tilde{f}_k, .., \tilde{f_N}$). The discrete Fourier transform (DFT) is given by :

$$\tilde{f}_k = \sum_{j=0}^{N-1} f_j e^{-i\frac{2\pi}{N}jk}$$

And the inverse discrete Fourier transform ($iDFT$) is given by :

$$f_k = \frac{1}{N}\sum_{j=0}^{N-1} \tilde{f}_j e^{i\frac{2\pi}{N}jk}$$

Fast Fourier Transform (examples : $FFTW$ and $FFTN$)

In 1965, J.W.Cooley and J.W.Tukey developed the revolutionary fast Fourier transform (FFT) algorithm that scales as $O(N \times \ln(N))$ with respect to the number of points N on the grid. Their algorithm was based on a fractal symmetry in the Fourier Transform that allows an N dimensional DFT to be solved with a number of smaller dimensional DFT computations. In our python numerical simulation, we used the *pyFFTW* library which implements FFT and the $FFTN$ *numpy* module, we will be able to map the data points on $1D$ and $2D$ grid to frequency domain and use Fourier Transform mathematical properties to de-noize a sample, or solve Poisson equation using either derivative property or convolution one.

II- Applications of the FFT in our numerical simulations

We will aim to show first how to de-noise a signal with FFT considering $1D$ and $2D$ samples. Then we go through solving $2D$ Poisson equation for 2 functions always with FFT and the property of the Fourier transform of the second derivative (Laplacian). After doing so, we calculate the Static Gravitational Field by solving the $2D$ Poisson equation, this time using the Convolution Property. Finally, we compute a time-integrator to get the trajectory of a mass in an axial symmetrical Potential.

II.1- 1D and 2D noise filtering

As an example for the $1D$ noise filtering, we took the sum of 2 sines with 2 different frequencies $f(x) = \sin(x) + \sin(2x)$. For the noise filtering on a $2D$ grid, we took $f(x, y) = \sin(2\pi * x) + \cos(\pi * y)$.

Addition of artificial noise

The objective was to create artificial noisy signal. To get such a sample, for each value of $f(x)$ or $f(x, y)$ on the grid, we add a random value using the method *random.randn*() from *Numpy* library. It generates random floats sampled from a Gaussian distribution of mean 0 and variance 1. It is at this point that FFT

can be very useful, since by taking the FFT of the initial noisy signal, we can calculate the mean value of the FFT and suppress in the Fourier space the frequencies corresponding to noise. We can go back to the real space, and compare the de-noise signal and the noise-less signal. The FFT was handled by $pyFFTW$ library.

FFT results By plotting the signals (see notebook code 1) using matplotlib, we see that we recovered with a great precision the initial functions. The mean-values and the standard deviation of the de-noise samples and the noise-less ones are similar.

II.2- Solve 2D Poisson equation for 2 different functions

In this part, the idea was to solve the Poisson equation for 2 functions as examples, with known solutions. The finite difference method is a very popular and simple way to calculate derivatives but it's not so efficient at small orders and also leads to very high order number of operations, in a $2D$ grid we should have $O(N*N)$ operations, for large N number of points, the numerical calculation takes too much time. The objective was to know if FFT can actually provide solutions with good accuracy since it provide an algorithm with a less consuming calculation time for large N ($O(N*log(N))$, and find at least one possible limitation of this method.

Solving Poisson equation : Theoretical aspects

Considering the Poisson equation in 2 dimensions, it involves the Laplacian operator that we consider here in a Cartesian coordinate system. By solving the Poisson Equation analytically, we get the analytical expression $\Phi(x,y)$ for a given known function $f(x,y)$. Here we provides the examples we took for the simulations :

$$f(x,y) = 6xy(1-y) - 2x^3 \quad \longrightarrow \quad \Phi(x,y) = y(1-y)x^3$$

The function $f(x,y)$ and the solution $\Phi(x,y)$ of the Poisson equation given above are represented on a $2D$ grid (see Notebook code 2) and they are obviously not periodic.

$$f(x,y) = \sin(x) + \sin(y) \quad \longrightarrow \quad \Phi(x,y) = \sin(x) + \sin(y)$$

The function and the solution are here periodic with the feature of being the same with respect to the Laplacian operator.
Now, the idea is to use the second derivative Fourier transform property to solve the Poisson equation. In fact, in the Fourier space, we have :

$$\mathcal{F}\left(\Delta\Phi(x,y)\right) = \mathcal{F}\left(f(x,y)\right) \quad \Longrightarrow \quad \mathcal{F}\left(\Phi(x,y)\right)(\omega) = \frac{-1}{|\omega|^2}\mathcal{F}\left(f(x,y)\right)$$

So to get the solution in the Fourier space, we have to take the Fourier transform of $f(x,y)$ divided by the square modulus of the frequencies. The inverse Fourier Transform is then a way to get the searched solution. Using the $pyFFTW$ library, we perform a discrete Fourier Transform, FFT, of the function $f(x,y)$ and we divide all these values at each grid points by it corresponding square modulus of the frequency. To get back to the solution in real space we perform an inverse-FFT. We compare the known solution to the simulated one :

FFT Simulation with Non-Periodic Function

For the Non-periodic Function, the final result is not so good, we don't even have the good orders of magnitude. The FFT solution provided is very flat at the borders whereas the known solution admit maximum values at one edge of the grid. We represented both solutions (see Notebook code 02). We think that this result is actually relevant to point out one of the feature of the FFT, the fact that at the edges of the $2D$ grid, the knows solution has non zeros values. The non periodicity of the function and the solution, and the non zero values at the edges of grids put the FFT process in a bad situation. Actually, the FFT algorithm is useful for relatively smooth functions, periodic or vanishing at the edges of the grid.

FFT simulation with periodic function

To confirm this hypothesis, we try the solution provided by FFT for a sine function. We represented both solutions, the known (sine solution) and the FFT result on the $2D$ grid (see Notebook code 02). The FFT solution is very close to the known one with a good accuracy. We calculate the mean-value and the standard

deviation of $\Phi(x,y)$ on the $2D$ grid comparing the known solution to the FFT solution. This confirm that FFT can be used to solve Poisson equation providing that the function is periodic or vanishing at the edges, and relatively smooth.

II.3- Solve 2D Poisson equation for static gravitational field

To illustrate the convolution property in the Fourier space, the idea is to solve the standard equation of sources $S(x,y)$ (in our case gravitational) whose distribution in space ($2D$ space, discretized by a grid) that gives rise to a potential $\Phi(x,y)$ at any point (x,y). This equation is actually the $2D$ Poisson equation provided that $f(x,y) = S(x,y)$. For the case of a mass distribution in space we will have : $S(x,y) = 4\pi * G * \rho(x,y)$ (where G is the Newton gravitational constant and $\rho(x,y)$ the density of mass in the $2D$ space). By considering the Green function $G(x,y)$, it is possible to demonstrate mathematically that the solution of the Poisson equation $\Phi(x,y)$ in the form of a convolution such that :

$$\Phi(x) = \int_{-\infty}^{\infty} G(x-u)S(u)\ du$$

Where the Green function is :

$$G(\overrightarrow{x}) \equiv -\frac{1}{4\pi}\frac{1}{|\overrightarrow{x}|}$$

And where the gravitational potential $\Phi(x,y)$ verifies :

$$\Delta\Phi(x,y) = S(x,y)$$

Solving Poisson Equation : Convolution Theorem

Writing $\Phi(x,y)$ in the form of a convolution gives an opportunity to use the Fourier Transform property :

$$\mathcal{F}\left(\Phi(x)\right) = \mathcal{F}\left(G(x,y)\right)(k) \times \mathcal{F}\left(S(x,y)\right)(k)$$

So the numerical calculation will consists in computing $S(x,y)$ and $G(x,y)$ on a $2D$ grid, than perform an the FFT of each of these functions, the product gives $\Phi(x,y)$ in the Fourier space. The inverse-FFT provide the final result of the potential.

Interpolation of Stars on a 2D grid

The computation of $G(x,y)$ on the grid is straightforward, we only have to make a loop on the grid assigning values of Green function after the computation of the distance to the center. But for $S(x,y)$ it is more complicated. In fact the idea of our algorithm is to pick positions of stars in a file with (x,y) coordinates and search for an interpolation scheme to assign a density of mass on each point of the grid considering the location of the stars. So we have N Stars and we apply a Particle mesh scheme to get $S(x,y)$, its relative algorithmic simplicity and speed (the running time scales as $O(N_p)+O(N_c*\ln(N_c))$, where N_p is the number of particles and N_c is the number of grid cells). The Particle mesh scheme considered was : the Cloud in Cell (CIC) scheme where particles are squares (in $2D$ grid) of uniform density and of one grid cell size. In practice, we loop over particles and assign the density with this method. There is a simpler interpolation but not so efficient : Nearest grid point (NGP) scheme where particles are point-like and all of particle's mass is assigned to the single grid cell that contains it. We represented the density of mass after CIC interpolation (see Notebook code 3) and we recover the spiral shape of a galaxy on a grid using the file containing 6000 positions of stars.

Comparison : FFT Results / Direct Numerical Calculation

We perform the FFT calculations of $G(x,y)$ and $S(x,y)$ using either $pyFFTW$ or $FFTn$ libraries in order to compare them, after making the product and the inverse-FFT, we have the potential $\Phi(x,y)$ of a spiral galaxy. The potential has this spiral shape for both FFT calculation but using $pyFFTW$ there is a strange artefact for which we don't know the reason. $FFTn$ does an impressive job. We represented the potential after interpolation for both (see Notebook code 3). Considering the time (number of operations) to perform the CIC scheme interpolation $O(N_s)+O(N_c\ln(N_c))$, where N_s is the number of stars and N_c is the number of grid cells, we have to add the FFT operations of order $O(N_c\ln(N_c))$. So it goes fast at large number of cells. We made the comparison with a straightforward calculation of the potential :

$$\Phi(\overrightarrow{r}) = -\sum_{i=1}^{N_S}\frac{Gm_i}{|\overrightarrow{r}-\overrightarrow{r_i}|}$$

At each point of the grid, we loop over all the stars positions to get the potential in this point. This procedure cost a lot, for each loop over the grid, there is a loop over all the stars positions, it is of order $O(N_c * N_c * N_s)$. However, it allows us to compare the FFT computations with a "theoretical result". The mean-value of $FFTn$ has the same order of magnitude with respect to the straightforward computation, but $pyFFTW$ is 1 order lower confirming this artefact we saw graphically.

II.4- Dynamics of Moving Bodies

The main idea now is to go through dynamics by integrating the orbit of a celestial body using a system of 2 differential equations in time :

$$\begin{cases} \dot{\overrightarrow{x}} = \overrightarrow{v} \\ \dot{\overrightarrow{v}} = -\nabla\left(\Phi(\overrightarrow{x})\right) \end{cases}$$

We begin by considering a simple axial symmetrical potential created by a mass located at the center of the grid. We didn't arrive to the point where we code a program to make the integration self-coherent but we think that we have all the elements to do it, we propose the procedure at the end of this report.

Axial Symmetric Potential

We performed the calculation of this potential using $FFTN$ library since it provide the best results (no artefact). The idea was then to get the gradient of the potential by the mean of a second order finite difference method. As a remark, we could also perform the gradient of the potential using the property of the Fourier transform of the first derivative of the potential (as we did it in the case of the second derivative to solve Poisson equation). We represented the potential and the gradient obtained for one gravitational source located in the center of the grid (see Notebook code 04). We have then to perform a time-integration of a mass in this potential created and get the kinematic of the point. We used a first order Eulerian scheme (we could use more accurate as Range-Kutta of order 2 or 4). We succeeded to get an orbit with a 1 body system.

Self-coherence

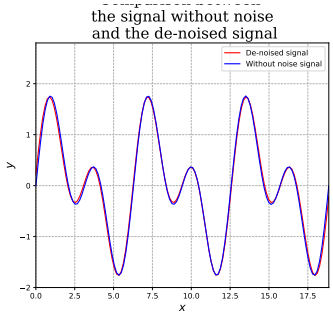
Going from this simple example, we can make an auto-coherent procedure to get to positions of all the stars that generate the field. the idea is to re-use the code we made for $2D$ static potential, with the CIC interpolation of the stars, then calculate the gradient in order to make a time integration of all the stars having the new velocities and the new positions of the stars. Then, we go back to the computation of the new potential by an interpolation of the new stars positions by CIC and solving the Poisson equation by FFT, we update again the particle positions and velocities with time-integration, etc.. There is a huge loop over time, to get positions of all the stars at each time. Unfortunately, we didn't have enough time to finish this code, losing too much time to succeed the static computation due to coding errors.

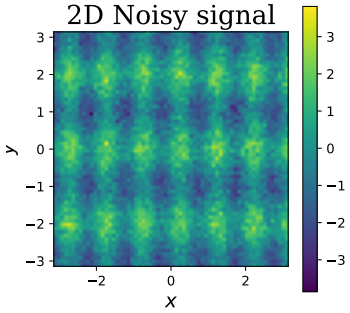
Conclusion

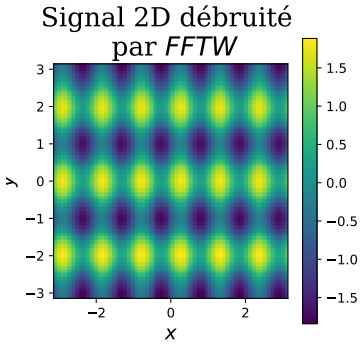
As a conclusion we can say that FFT procedure is very efficient in many cases since we succeded to determine the static gravitational potential with high precision, de-noising sample and find solution of Poisson Equation for some functions. it is also cheap in terms of time computation in comparison to straighforward computations.

Appendix

Some images related to the denoising part :



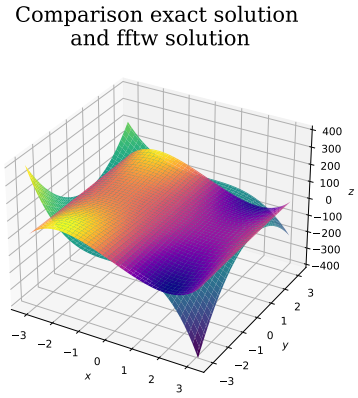


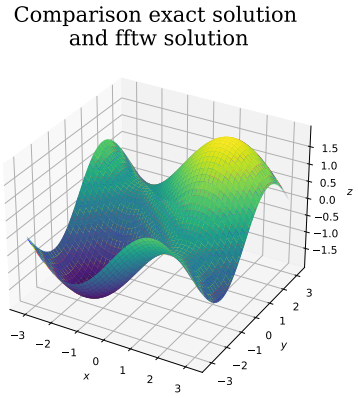


Comparison original/de-noised signals

Noised and de-noised signals in 2D

Some images related to the solutions of the Poisson equation for 2 functions :

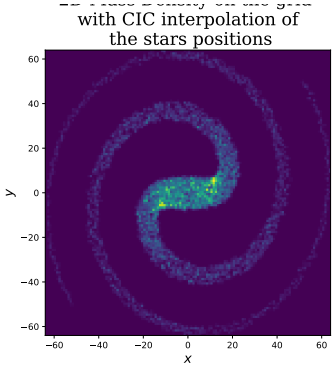


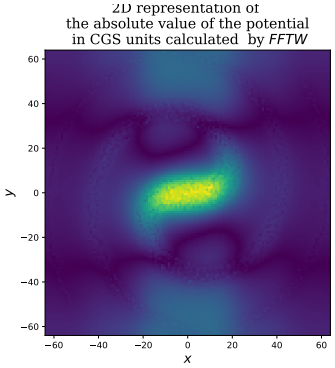


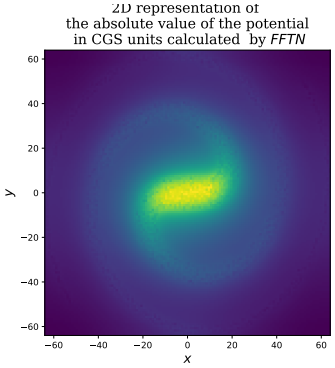
Comparison exact/fftw for non periodic

Comparison exact/fftw periodic

Some images related to the 2D gravitationnal problem :







2D Mass density

2D potential by $FFTW$

2D potential by $fftn$

If you want more wonderfull images, check our codes ! :)