## Week 11 — *External libraries: application to a heat equation solver*

The goal of this exercise is to use an external library made for scientific computing and adapt it to our needs through a wrapping interface.
We will use the C library FFTW (Fastest Fourier Transform in the West) to code an efficient heat equation solver. The following points will be considered for the grading:

- Proper usage of git (meaningful comments, several commits with developments steps, use of .gitignore)
- Code works as intended
- Readability of the code (meaningful variable names, comments)
- Minimal documentation: README file

**Use the starting point from GIT for this exercise**

### Exercise 1: *Code discovery*

By consulting the provided code, you will notice the addition of a few classes. You can regenerate the documentation with the provided `Doxyfile` or doing build the 'doc' target. In particular, pay attention to additional classes:

- `MaterialPoint`
- `Matrix`
- `MaterialPointsFactory`
- `FFT`: The FFTW wrapping interface

Describe in your README how the particles are organized in the various objects.

### Exercise 2: *Link to FFTW*

In this exercise, you will modify the `CMakeLists.txt` file to *dynamically link* your executable against the FFTW library.

1. Install FFTW

   ```
   sudo apt install libfftw3-dev
   ```

2. Modify the `CMakeLists.txt` to link your executable against FFTW. CMake must first find the library. Make an option so that the user can choose to activate (or not) the FFT part of the code.

3. Verify that your executable (and tests) have been successfully linked by running:

   ```
   ldd <your-executable>
   ```

   and with the command

   ```
   nm -DAC <your-library>
   ```

**Exercise 3:** ***Interface implementation***

In this exercise, you will implement a simple wrapping interface to FFTW that will work with the classes we have created in our code, as to avoid exposing FFTW-specific code.

1. Familiarize yourself with the API of FFTW through its documentation (in particular the multi-dimensional DFT tutorial, the complex DFT page and the links therein).
2. Write the forward and backward transform functions for the `Matrix` class. Check your results with the provided test for the forward transform and implement a test for the backward transform.
3. Write the computeFrequencies function which computes the wavenumbers and check your results by comparing with the numpy.fft.fftfreqs routine. Implement a test for the computeFrequencies.

**Exercise 4:** ***Solver implementation***

We now implement a solver of the transient heat equation in two dimensions:

$$\rho C \frac{\partial \theta}{\partial t} - \kappa \left( \frac{\partial^2 \theta}{\partial x^2} + \frac{\partial^2 \theta}{\partial y^2} \right) = h_v,$$

where $\rho$ is the mass density, $C$ is the specific heat capacity, $\kappa$ is the heat conductivity, $h_v$ is the volumetric heat source and the primary unknown is $\theta$, the temperature.

We choose an approach based on Fourier series to solve the above equation. The Fourier transform has the nice property that it transforms differential operators into mere multiplications, so that the heat transfer equation becomes:

$$\rho C \frac{\partial \hat{\theta}}{\partial t} + \kappa \hat{\theta}(q_x^2 + q_y^2) = \hat{h}_v$$

where $\hat{\theta}$ is the Fourier transform of $\theta$ and $q_x$, $q_y$ are coordinates in Fourier space. We now have an ordinary differential equation that can be solved with an explicit time integrator:

1. By Fourier transform: $\theta_n \rightarrow \hat{\theta}_n$
2. $\frac{\partial \hat{\theta}_n}{\partial t} = \frac{1}{\rho C} \left( \hat{h}_v - \kappa \hat{\theta}_n (q_x^2 + q_y^2) \right)$
3. By inverse Fourier transform: $\frac{\partial \hat{\theta}_n}{\partial t} \rightarrow \frac{\partial \theta_n}{\partial t}$
4. $\theta_{n+1} = \theta_n + \Delta t \frac{\partial \theta_n}{\partial t}$

You have to implement and test this Euler integration scheme:

1. Implement the `ComputeTemperature` subclass that will compute a full step of the time integration. Please pay attention to dimensions, since we manipulate particles and not a continuum.
2. Implement a validating test for an initial homogeneous temperature and no heat flux.
3. Implement a validating test for a volumetric heat source of the form

$$h_v(x, y) = \left( \frac{2\pi}{L} \right)^2 \sin \left( \frac{2\pi x}{L} \right) \tag{1}$$

   and the initial temperature field that should respect equilibrium, for a domain $[-1, 1] \times [-1, 1]$. Equilibrium temperature field is given by :

$$\theta(x, y) = \sin \left( \frac{2\pi x}{L} \right) \tag{2}$$

4. (Optional) Implement a validating test for a volumetric heat source of the form

$$h_v(x, y) = \begin{cases} 1 & \text{if } x = \frac{1}{2} \\ -1 & \text{if } x = -\frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

and the initial temperature that should respect equilibrium. The domain spans $[-1, 1] \times [-1, 1]$. The equilibrium temperature field, for $\rho = 1$, $C = 1$ and $\kappa = 1$ is given by :

$$\theta(x, y) = \begin{cases} -x - 1 & \text{if } x \leq -\frac{1}{2} \\ x & \text{if } -\frac{1}{2} < x \leq \frac{1}{2} \\ -x + 1 & \text{if } x > \frac{1}{2} \end{cases} \tag{4}$$

5. (Optional) Implement a python script to generate a heat distribution within a provided radius as given in Equation (**??**).

$$h_v(x, y) = \begin{cases} 1 & \text{if } x^2 + y^2 < R \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

   where $R$ is a provided radius. We also apply a boundary condition to the temperature field: it should be zero on the boundary of the domain and not evolve with time. Explain in the README how you integrate this condition within the existing code.

6. Describe how to launch such a stimulation which will produce dumps observable with Paraview for a grid of $512 \times 512$ particles.