Week 13 — *Eigen: linear algebra applied to the heat equation*

The goal of this exercise is to use the external library Eigen made for high performance vector and linear algebra computations.
We will use it to a heat equation implicit solver based on the finite-difference scheme (see the 1D application example on the Wikipedia page).
Use the starting point from GIT for this exercise

Exercise 1: *Code discovery*

We have replaced a few classes that were previously implemented by hand with Eigen classes.

Exercise 2: *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 ρ is the mass density, C is the specific heat capacity, κ is the heat conductivity, h_v is the volumetric heat source and the primary unknown is θ, the temperature.

This time, we implement a finite-differences scheme. The idea of this scheme is to approximate the derivatives $\frac{\partial^2 \theta}{\partial x^2}$ and $\frac{\partial^2 \theta}{\partial y^2}$ by their finite difference approximation, that is (for a given time t):

$$\frac{\partial^2 \theta}{\partial x^2}(x,y) \approx \frac{\theta(x-\Delta x, y) - 2\theta(x,y) + \theta(x+\Delta x, y)}{\Delta x^2} \quad \text{and} \quad \frac{\partial^2 \theta}{\partial y^2}(x,y) \approx \frac{\theta(x, y-\Delta y) - 2\theta(x,y) + \theta(x, y+\Delta y)}{\Delta y^2}$$

Or in discretized form, if $\delta\theta_{i,j} = \frac{\partial^2 \theta}{\partial x^2}(x_i, y_j) + \frac{\partial^2 \theta}{\partial y^2}(x_i, y_j)$ and $\Delta x = \Delta y = a$:

$$\delta\theta_{i,j} = \frac{\theta_{i-1,j} + \theta_{i,j-1} - 4\theta_{i,j} + \theta_{i+1,j} + \theta_{i,j+1}}{a^2} \tag{1}$$

This time, we implement an implicit time solver (backward Euler) for the diffusion equation. We introduce the finite difference for the time derivative:

$$\frac{\partial \theta}{\partial t}(t^{n+1}) = \frac{\theta^{n+1} - \theta^n}{\Delta t} \tag{2}$$

Pluging equations (1) and (2) into the heat equation and solving for θ^{n+1} gives:

$$\theta_{i,j}^{n+1} - \frac{\Delta t \kappa}{\rho C} \delta\theta_{i,j}^{n+1} = \frac{\Delta t}{\rho C} h_{v,i,j}^{n+1} + \theta_{i,j}^n \tag{3}$$

Let us collect all unknowns $\theta_{i,j}^{n+1}$ into a large vector $\mathbf{\Theta}^{n+1}$, then equation (3) turns into a (sparse) linear system of equations:

$$\left(\mathbf{I} - \frac{\Delta t \kappa}{\rho C} \mathbf{\Delta}_{\text{fd}} \right) \mathbf{\Theta}^{n+1} = \frac{\Delta t}{\rho C} \mathbf{H}_v^{n+1} + \mathbf{\Theta}^n \tag{4}$$

where $\mathbf{\Delta}_{\text{fd}}$ is a matrix with coefficients resulting from (1) and $\mathbf{H}_v^{n+1}$ collects into a large vector the value of the internal heat at each particle.

1. Create a class `ComputeTemperatureFiniteDifferences`.
2. Assemble the right hand side of equation (4) into an Eigen `VectorXd`. We have extended the `Eigen::Matrix` class in the file `matrix_eigen_addons.hh` to have a simple iterator. This means you can use range-for loops and standard algorithms for complex vector manipulation. For more help, please refer: Eigen VectorXd.
3. Assemble the matrix of system (4) into an Eigen `SparseMatrix`.

```
Eigen::SparseMatrix<double> A(rows, cols); // by default column major
A.insert(i, j) = ... // specific entry at i^th row and j^th column
A.coeffRef(i, j) += ... // to add to the previous value at  i^th row and j^th
  column
```

   For more help, please refer: Eigen SparseMatrix.

4. Solve the system using `SparseLU`. The Eigen Sparse solver expects a compressed and column major `Eigen::SparseMatrix`. For more help, please refer: Eigen Sparse LU Solver.
5. Write tests for you implementation (you can re-use last week's tests).
6. How many times are you factorizing the matrix in a simulation? Is it necessary? What can you change in the code to avoid uncessary factorizations?
7. Compare the performance of the finite-differences solver to the FFT solver.