Méthodes numériques sous Python
S02 - Linear Algebra

Cyril Desjouy, Oliver Dazel

22 novembre 2017

1 Introduction

The discretization of a linear physics problem always leads to a linear system of finite dimension. It is a matrical problem of the form :

$$[\mathbf{M}]\mathbf{X} = \mathbf{F}. \tag{1}$$

$[\mathbf{M}]$ is a $(n \times n)$ invertible matrix, $\mathbf{F}$ is a vector of length n, these two elements are known. $\mathbf{X}$ is also a vector of length n and is the unknown of the problem.

We are, in this activity, considering several numerical methods to solve this problem.

We will consider a simple problem as a case of application : the discretization of Helmholtz equation for in a dimension n space lead to the following linear system :

$$\begin{bmatrix} 1-k^2 & -1 & 0 & \dots & 0 \\ -1 & 2-k^2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2-k^2 & -1 \\ 0 & \dots & 0 & -1 & 2-k^2 \end{bmatrix} \mathbf{X} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}. \tag{2}$$

In the following we will consider that $k = 0$.

— Write a Python function called `matrixM()` that generates the matrix $\mathbf{M}$ and the vector $\mathbf{F}$

2 Direct method : Gauss method

The Gauss method consists in replacing the initial system by an equivalent one which is upper triangular. This one is obtained step by step by using linear combination whose objective is to successively introduce zeros in the lines of the lower diagonal. The starting point is matrix $[a_{ij}]$ and left hand side $[b_i]$.

The first step then consist in replacing the $n-1$ last equations with ones not involving x_1. Let now define the first pivot π_1 by $\pi_1 = a_{11}$. It then leads to :

$$\begin{bmatrix} \pi_1 & a_{12} & \dots & a_{1n} \\ 0 & a_{22} - \dfrac{a_{21}a_{12}}{\pi_1} & \dots & a_{2n} - \dfrac{a_{21}a_{1n}}{\pi_1} \\ \vdots & \vdots & \vdots & \\ 0 & a_{n2} - \dfrac{a_{n1}a_{12}}{\pi_1} & \dots & a_{nn} - \dfrac{a_{n1}a_{1n}}{\pi_1} \end{bmatrix} \mathbf{X} = \begin{bmatrix} b_1 \\ b_2 - \dfrac{a_{21}b_1}{\pi_1} \\ \vdots \\ b_2 - \dfrac{a_{n1}b_1}{\pi_1} \end{bmatrix}. \tag{3}$$

The second step is devoted to column 2. This process is then iterated until the last column. This triangular system is then solved by starting from component n until component 1 of the unknown vector.

- Write a function called `triangularise()` that implements the first phase for the method, i.e that makes an upper triangular matrix from the matrix M
- Write a function called `solve_gauss()` that uses the `triangularise()` function to solve the problem
- Note that the inputs of this function will be the matrix **M** and the vector **F**, and the outputs will be the solution **X** of the problem and a string describing the convergence of the method.
- Compare the solution obtained from the Gauss method with the one obtained from the built-in function `solve` provided by the submodule `linalg` of `numpy`.

3 Iterative methods I : Jacobi and Gauss-Seidel

Jacobi and Gauss-Seidel methods are based on the following principle : the [**M**] matrix is first decomposed under the following form :

$$[\mathbf{M}] = [\mathbf{D}] - [\mathbf{E}], \tag{4}$$

with [**D**] a matrix "easily invertible".

These methods are iterative methods. They start from an initial vector $\mathbf{X}^0$ which is arbitrary. If a frequency sweep is considered the initial vector at a given frequency can, for example be the solution at the previous frequency. An iterative sequence is then built based on the following relation :

$$\mathbf{x}^{k+1} = [\mathbf{D}]^{-1}\left([\mathbf{E}]\mathbf{x}^k + \mathbf{F}\right). \tag{5}$$

This sequence can reach convergence only if the following condition is achieved : $\left\|[\mathbf{D}]^{-1}[\mathbf{E}]\right\| < 1$. In case of convergence, a key issue is the number of iteration to be considered. One possible way is to compare the norm of the residual $\|\mathbf{r}_k\| = \left\|[\mathbf{M}]\mathbf{x}^k - \mathbf{F}\right\|$ to a value ϵ which can be set by the user or fixed in the program. It is also very useful, in case of a slow convergence to impose a maximum number of iterations.

We will consider here two schemes : Jacobi and Gauss-Seidel Methods. They only differ on the decomposition :
- For the Jacobi method [**D**] is the diagonal of [**A**].
- For the Gauss Seidel method [**D**] is the upper triangular of [**A**].

- Write two functions called `solve_jacobi()` and `solve_seidel()` that implement the algorithms to solve the problem using the two methods above mentioned.
- Note that these functions will take as inputs the matrix **M**, the vector **F**, the value ϵ for the criterion for the norm and N the maximum number of iterations. The output of these functions will be the solution **X** of the problem and a string describing the convergence of the method.
- Compare the solutions obtained from this two methods with the one obtained from the built-in function `solve` provided by the submodule `linalg` of `numpy`.

4 Iterative methods II : Conjugate gradient

The conjugate gradient method is another iterative method used to solve systems of linear equations. You will find a very good description of this method, as well as the algorithm on its dedicated webpage on wikipedia : `https://en.wikipedia.org/wiki/Conjugate_gradient_method`

- Write a function called `conjugate_grad()` that implements the algorithms to solve the problem using the conjugate gradient method
- Note that this function will take as inputs the matrix $\mathbf{M}$, the vector $\mathbf{F}$, the value ϵ for the criterion for the norm and N the maximum number of iterations. The output of these functions will be the solution $\mathbf{X}$ of the problem and a string describing the convergence of the method.
- Compare the solution obtained from the conjugate gradient method with the one obtained from the built-in function `solve` provided by the submodule `linalg` of `numpy`.