Article Image

Linear Systems with Numpy and Scipy

Table of Contents

Introduction

In the following we want to study how to use `numpy` and `scipy` to solve Linear Systems with Python. Most if the function in Numpy and Scipy for Linear Algebra are contained in the sub-packages `np.linalg` and `scipy.linalg`, as you will see in the forllowing.

To fix the notation, given a matrix $A \in \mathbb{R}^{n \times n}$ and a vector $y \in \mathbb{R}^n$, *solving* a linear system means finding (when exists) a vector $x \in \mathbb{R}^n$ such that it solves

$$Ax = y \tag{1}$$

This is not hard to do in `numpy`, since it implements a function `np.linalg.solve`, taking as input a 2-dimensional array `A` and a 1-dimensional array `y`, and returns the solution `x` to the linear system. In particular:

```
# Generates the problem
A = np.array([[1, 1, 1], [2, 1, 2], [0, 0, 1]])
y = np.array([0, 1, 0])

# Solve the system
x_sol = np.linalg.solve(A, y)
print(f"The solution is {x_sol}.")
```

Testing the accuracy

You already studied that, when the matrix A is ill-conditioned, the solution of a linear system won't be correct, since the small perturbations on y introduced by the floating point system will be amplified and the corresponding solution will be drammatically distant to the true solution. To check how accurate our computed solution is to the true solution of the system (i.e. to quantify the amplification of the perturbation on the data), it is common to use the relative error, which is defined as

$$E(x_{true}, x) = \frac{||x_{true} - x||_2}{||x_{true}||_2} \tag{2}$$

Clearly, the problem is that if our algorithm fails in recovering the true solution due to the ill-conditioning of the system matrix A, how can we compute the true solution x_{true}, required to compute $E(x_{true}, x)$? The solution is to build a **test problem**.

Creating a Test Problem

Consider a matrix $A \in \mathbb{R}^{n \times n}$ and assume we want to test the accuracy of an algorithm solving systems involving A. Fix an n-dimensional vector $x_{true} \in \mathbb{R}^n$, and compute $y = Ax_{true}$. Clearly, this procedure defines a linear system

$$Ax = y \tag{3}$$

of which we know that x_{true} is a solution, since we built the term b accordingly. Now, when we apply our algorithm to that linear system, we get a solution x_{sol}, approximately solving $Ax = y$. Given that, we can always compute the relative error $E(x_{true}, x_{sol})$ asssociated to the solution obtained by the algorithm. In numpy, this can be simply done as

```
import numpy as np

# Setting up the dimension
n = 10

# Creating the test problem
A = np.random.randn(n, n) # n x n random matrix
x_true = np.ones((n, ))   # n-dimensional vector of ones

y = A @ x_true # Compute the term y s.t. x_true is a sol.

# Solving the system with numpy
x_sol = np.solve(A, y)

# Computing the accuracy
E_rel = np.linalg.norm(x_true - x_sol, 2) / np.linalg.norm(x_true, 2)
print(f"The relative error is {E_rel}")
```

Condition number

You should already know that the conditioning of an $n \times n$ matrix A can be quantified by a term called **condition number** which, whenever A is invertible, is defined as

$$k_p(A) = ||A||_p ||A^{-1}||_p \tag{4}$$

Where $p \geq 1$ idenfities the norm on which the condition number is computed.

An invertible matrix A is said to be ill-conditioned when its condition number grows exponentially with the dimension of the problem, n.

The condition number is related to the accuracy of the computed solution of a linear system by the following inequality

$$\frac{||\delta x||}{||x||} \leq k(A) \Big(\frac{||\delta A||}{||A||} + \frac{||\delta y||}{||y||} \Big) \tag{5}$$

which implies that the relative error on the computed solution is big whenever $k(A)$ is big. Moreover, note that as a consequence of the formula above, the accuracy of a computed solution is partially a proprierty of the condition number of A itself, meaning that *no algorithm* is able to compute an accurate solution to an ill-conditioned system.

Computing the p-condition number of a matrix A in Numpy is trivial, just use the function `np.linalg.cond(A, p)` to compute $k_p(A)$.

Solving Linear System by Matrix Splitting

As you should know, when the matrix A is unstructured, the linear system $Ax = y$ can be efficiently solved by using LU Decomposition. In particular, with Gaussian elimination algorithm, one can factorize any non-

singular matrix $A \in \mathbb{R}^{n \times n}$ into:

$$A = PLU \tag{6}$$

where $L \in \mathbb{R}^{n \times n}$ is a lower-triangular matrix, $U \in \mathbb{R}^{n \times n}$ is an upper-triangular matrix with all ones on the diagonal and $P \in \mathbb{R}^{n \times n}$ is a permutation matrix (i.e. a matrix obtained by permutating the rows of the identity matrix). If the decomposition is computed without pivoting, the permutation matrix equals the identity. Note that the assumption that A is non-singular is not restrictive, since it is a necessary condition for the solvability of $Ax = y$.

Since P is an orthogonal matrix, $P^{-1} = P^T$, thus

$$A = PLU \iff P^T A = LU \tag{7}$$

Since linear systems of the form

$$Lx = y \quad \text{and} \quad Ux = y \tag{8}$$

can be efficiently solved by the Forward (Backward) substitution, and the computation of the LU factorization by Gaussian elimination is pretty fast ($O(n^3)$ floating point operations), we can use that to solve the former linear system.

Indeed,

$$Ax = y \iff P^T Ax = P^T y \iff LUx = P^T y \tag{9}$$

then, by Forward-Backward substitution, this system can be solved by subsequently solve

$$Lz = P^T y \quad \text{then} \quad Ux = z \tag{10}$$

whose solution is a solution for $Ax = y$.

Even if this procedure is automatically performed by the `np.linalg.solve` function, we can unroll it with the functions `scipy.linalg.lu(A)` and `scipy.linalg.solve_triangular(A, b)`, whose documentation can be found here and here.

> ***Exercise:*** Write a function that takes as input a non-singular matrix $A \in \mathbb{R}^{n \times n}$ and a vector $y \in \mathbb{R}^n$ and returns the solution $x \in \mathbb{R}^n$ of $Ax = y$ without using `np.linalg.lu`.

▶ Visualize the solution

Homework

Please refer to the Homework PDF on Virtuale.

Written by

Davide Evangelista

Published 17 Sep 2022