Computational Physics III: Report 2
Linear systems solving and diagonalization methods

Due on April 30, 2020

April 24, 2020

Raffaele Ancarola

Contents

Introduction

Solving a system of linear equations

A *linear problem* could be defined as a system of which the describing equations are all linear. Furthermore, a linear system is said to be *determined* if the number of equations N is finite and it corresponds to the number of the unknowns. Such a system defined on a field $\mathbb{K}$ takes the advantage to be written in a matrix form:

$$A \cdot \vec{x} = \vec{b} \tag{1}$$

where A is the describing matrix and $\vec{b} \in \mathbb{K}^N$ the affine component of the system, or the components which are independent with respect to the unknows contained in $\vec{x}$. Because the system is *determined*, the condition that A must satisfy is the *inversibility*, then $A \in \mathcal{GL}(N)$ and a solving the system means to find $\vec{x} \in \mathbb{K}^N$ such that eq. (1) is satisfied. There exist various approaches that can reach this aptempt, in this report three cases will be analysed: the *gauss elimination*, the *LU decomposition* and the *diagonalisation*.

Gauss elimination algorithm

The *Gauss elimination* bases to the fact that any square matrix can be decomposed into a finite sequence of elementary operations $\{P_k\}_{1 \leq k \leq M}, M \in \mathbb{N}^*$. There are basically three kinds of them:

- Multipling of a row by a scalar factor $\lambda \in \mathbb{K}$
- Switching a row with another
- Adding a row with a multiple of another

The purpose of this method is to reduce the involved matrix A into the identity applying the same operations to the vector $\vec{b}$, as shown in the equation (2).

$$A = P_1 \cdot ... \cdot P_M \implies \vec{x} = P_M^{-1} \cdot ... \cdot P_1^{-1} \cdot \vec{b}\,, \quad M \in \mathbb{N}^* \tag{2}$$

LU decomposition

The *LU decomposition* is not a direct method which solves a linear system, but it allows to simplify the resolution by decomposing the A matrix into a lower-triangular matrix L and an upper-triangular matrix U. The simplification is due to the major facility to invert the two matrices precedently presented. Once A is decomposed, the process is straigh-forward:

$$A \cdot \vec{x} = L \cdot U \cdot \vec{x} = \vec{b}$$

$$L \cdot \vec{y} = \vec{b} \tag{3}$$

$$U \cdot \vec{x} = \vec{y} \tag{4}$$

Both equations (3) and (4) can be solved sequentially using the *Gauss elimination* method.

Diagonalization: introduction

In case A is a symmetric matrix, the spectral theorem [] states that such a matrix is equivalent (definition of equivalence here: []) to a diagonal matrix D, where the transition matrix P is unitary ($P^{-1} = \bar{P}^T$), then:

$$A = P \cdot D \cdot \bar{P}^T \implies \vec{x} = P \cdot D^{-1} \cdot \bar{P}^T \cdot \vec{b} \tag{5}$$

Generally diagonalization is not used to solve general systems of linear equations, but it's convenient when the problem is related to find the eigen-base related to the eigen-values.

Problem 1

(1) LU decomposition implementation

This algorithm separates the input matrix A into a lower triangular L and an upper triangular U, garanteeing that $A = L \cdot U$. Neverthless, not all the invertible square matrices are purely LU decomposable, then it may happen that the output can result ill formed. The code (1) shows at line 23 that a division by the diagonal values is performed, causing eventually a singularity. A possible work-around is to apply the partial pivoting technique in order to swap the problematic lines. In listing (1) is shown a full implementation with partial pivoting.

Listing 1: *LU decomposition* implementation with partial pivoting

```
1  function [L, U, P] = lu_decomposition(A)
2      [Ni, Nj] = size(A);
3      assert(Ni == Nj, "The input must be diagonal");
4  
5      N = Nj;
6      assert(N > 0, "The input must non empty");
7  
8      L = eye(N); % if zeros doesn't give the same result
9      U = A; % if zeros doesn't give the same result
10     P = eye(N); % identity matrix
11 
12     for k=1:(N-1)
13         % pivoting section
14         [Amax,r] = max(abs(U(k:N, k)));
15         r = r + k - 1;
16         % swap rows if it's not the identity swap operation
17         U([k r],:) = U([r k],:);
18         P([k r],:) = P([r k],:);
19         L([k r], 1:k-1) = L([r k], 1:k-1);
20 
21         % computing LU
22         for i=(k+1):N
23             L(i,k) = U(i,k) / U(k,k);
24             U(i,:) = U(i,:) - L(i,k) * U(k,:);
25         end
26     end
27 end
```

Partial pivoting

The *LU decomposition* algorithm (presented below in exercise 1.1) can easily run into singularities, especially when A presents zeros as diagonal terms. In order to avoid divergent results, it would better select the rows of which element is not zero in the requested columns and swap them with the current one. More precisely, at the k-th step, select the r-th row such that $A_{rk} = \max_{k \leq i \leq N} |A_{ik}|$, then swap rows at the position k and r. If the pivoting is applied the resulting *LU decomposition* won't be anymore like it was defined in the previous section, but a correction to equation (3) must be applied:

$$P \cdot A = L \cdot U \implies L \cdot \vec{y} = P \cdot \vec{b} \tag{6}$$

where P is the orthogonal matrix that accumulated all row switching applications. The rest of the solving method remains unchanged.

(1.1) Solving a linear system

A linear system can be solved applying the LU decomposition and then a gauss elimination process, as shown in the equations (6) and (4).

For example, the system in equation (7) is determined and can be solved using the `solve.m` script. Addictionally the `test_solve.m` script compares with the matlab `x = A \ b` verifying that the solution $\vec{x}$ is given correctly by the `solve.m` script.

$$\begin{cases} 2x_1 + x_2 - x_3 + 5x_4 & = 13 \\ x_1 + 2x_2 + 3x_3 - x_4 & = 37 \\ x_1 + x_3 + 6x_4 & = 30 \\ x_1 + 3x_2 - x_3 + 5x_4 & = 19 \end{cases} \implies A = \begin{pmatrix} 2 & 1 & -1 & 5 \\ 1 & 2 & 3 & -1 \\ 1 & 0 & 1 & 6 \\ 1 & 3 & -1 & 5 \end{pmatrix}, \quad \vec{b} = \begin{pmatrix} 13 \\ 37 \\ 30 \\ 19 \end{pmatrix} \implies \vec{x} = A^{-1} \cdot \vec{b} = \begin{pmatrix} 2 \\ 4 \\ 10 \\ 3 \end{pmatrix} \tag{7}$$

(1.3) Decomposition of a matrix

The example taken in equation (**??**) is a problematic case where a pure LU decomposition doesn't exist. So, the form $P \cdot A = L \cdot U$ is obtainable using the pivoting described in the previous section.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 9 \\ 4 & -3 & 1 \end{pmatrix} \implies L = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.25 & 0.5 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 4 & -3 & 1 \\ 0 & 5.5 & 8.5 \\ 0 & 0 & -1.5 \end{pmatrix} \quad P = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \tag{8}$$

(2) Puzzle board game

The puzzle game problem is reductible t

(3) Helicopter power formula: dimensional analysis

```
% power P2 = 1/9 * P1

A = [1, 0,  1;
     2, -1, 3;
     2, 0,  0]

b = [1; -2; 3]

disp("Solution")
%x = A \ b
x = solve(A,b)
```

The eigenvalue problem and diagonalization

Let $\hat{A}$ be an operator defined over an hilbert space $\mathcal{H}$. By solving an eigenvalue problem is meant to find all vectors (or functions) $x \in \mathcal{H}$ such that there exists a real (or complex) value λ that satisfies the following condition:

$$\hat{A} \cdot x = \lambda \cdot x, \lambda \in \mathcal{K} \tag{9}$$

In the case of this report, the interest is to computationally solve the eigenvalue problem for finite rank operators, which can be expressed as square matrices. So, let N be rank of a square matrix A and $\vec{v} \in \mathcal{K}^N$, then the equation (8) is equivalent to:

$$A \cdot \vec{v} = \lambda \cdot \vec{v}, \lambda \in \mathcal{K} \tag{10}$$

Power method

The power method bases its functioning on the iterative application of a specific operation. The principle is that every iteratio step tends to minimise of the distance between the old evaluated eigen value λ_{k-1} and the current λ_k. Starting by a unitary vector $\vec{v} \in \mathcal{K}^N | \|v\| = 1$, the correspond diagonal value is given by the hermitian scalar product:

$$\tag{11}$$

Jacobi method

Problem 2

(1) Power iteration methods implementation

(2) Eigenmodes of a vibrating string

(3) Jacobi method implementation

(4) Landau levels in a square-lattice model

(a) Grey-scaled image file `stm.png`

(b) Fourier transformed image file `stm.png`

Conclusion

Documentation and sources

[1] https://edu.epfl.ch/coursebook/en/solid-state-physics-i-PHYS-309