Computational Physics III: LU decomposition

Due on May 04, 2023

Salomon Guinchard

Contents

Problem 1 **3**

LU decomposition **3**
1- LU decomposition algorithm 4
2- Ill posed problem for LU without pivoting 4
2- LUP decomposition algorithm 5

Problem 2 **6**

Equivalent resistance of infinite resistor grid **6**

Problem 3 **7**

Power iterations methods **7**
1- Power method 7
2- Inverse power method 8
3- Rayleigh quotient method 8

Problem 4 **9**

Vibrating modes of a string **9**

Problem 5 **11**

2D quantum well **11**

Problem 1

LU decomposition

The LU decomposition of a square matrix corresponds to the expansion of the matrix as a product of a lower triangular (L) and an upper triangular (U) matrices. Given a matrix A, its LU decomposition is then $A \equiv L \cdot U$, where $\cdot$ corresponds to the matricial product. For example, given the linear $4x4$ system of equations:

$$\begin{aligned} 2x_1 + x_2 - x_3 + 5x_4 &= 13 \\ x_1 + 2x_2 + 3x_3 - x_4 &= 37 \\ x_1 + x_3 + 6x_4 &= 30 \\ x_1 + 3x_2 - x_3 + 5x_4 &= 19 \end{aligned} \tag{1}$$

it is possible to solve it by writing in in form of the following matrix equation:

$$A\mathbf{x} = \mathbf{b}, \tag{2}$$

where $\mathbf{x} = (x_1, x_2, x_3, x_4)^T$, $\mathbf{b} = (13, 37, 30, 19)^T$ and A is the matrix containing the coefficients of the system. Performing manually the LU algorithm by hand, Eq.(2) can be solved applying LU decomposition and backward substitution. Since the point of the report is to emphasize the numerical aspect of this decomposition, the calculations are skipped and the result is given below.

$$A = \begin{pmatrix} 2 & 1 & -1 & 5 \\ 1 & 2 & 3 & -1 \\ 1 & 0 & 1 & 6 \\ 1 & 3 & -1 & 5 \end{pmatrix} = \cdot L \cdot U, \tag{3}$$

with

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1/2 & 1 & 0 & 0 \\ 1/2 & -1/3 & 1 & 0 \\ 1/2 & 5/3 & -19/8 & 1 \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} 2 & 1 & -1 & 5 \\ 0 & 3/2 & 7/2 & -7/2 \\ 0 & 0 & 8/3 & 7/3 \\ 0 & 0 & 0 & 111/8 \end{pmatrix} \tag{4}$$

One can easily verify that upon multiplying L and U, we recover A. Hence, applying the following algorithm with L and U defined as above,

1. Perform LU decomposition
2. Perform forward substitution $L\mathbf{y} = \mathbf{b}$
3. Perform backward substitution $U\mathbf{x} = \mathbf{y}$

one gets that $\mathbf{x} = (2, 4, 10, 3)^T$, which is the answer provided by the matlab expression $A/\mathbf{b}$.

Problem 1 continued on next page...

1- LU decomposition algorithm

Listing 1: Matlab script for the LU decomposition without pivoting

```
function [L,U] = lu_decomposition_nopiv(A)
sz=size(A);
L=eye(sz(1));
U=A;
    if sz(1)~=sz(2)
        error('This is not a square matrix.');
    else
        for i=1:sz(1)
            for j=i:sz(1)-1
                L(j+1,i) = U(j+1,i)/U(i,i);
                U(j+1,:) = U(j+1,:)-L(j+1,i)*U(i,:);
            end
        end
    end
end
```

2- Ill posed problem for LU without pivoting

In some cases, certain matrices can become problematic when performing their LU decomposition without pivoting, as in some step of the algorithm, some zeros can appear on the diagonal. Since at the step k, the coefficients $l_{ik}^{(k)} \propto 1/a_{kk}^{(k)}$, if the latter coefficient $1/a_{kk}^{(k)}$ is zero, some infinite values will appear in the process.

It is the case for the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 9 \\ 4 & -3 & 1 \end{pmatrix} \tag{5}$$

as in step $k = 2$, a zero is present on the diagonal for the term a_{22}:

$$A^{(2)} = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0 & 3 \\ 4 & -3 & 1 \end{pmatrix} \tag{6}$$

The final result for the LU algorithm without pivoting is indeed problematic, since L and U respectively read:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & -\text{Inf} & 1 \end{pmatrix} \qquad U = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0 & 3 \\ \text{NaN} & \text{NaN} & \text{Inf} \end{pmatrix}, \tag{7}$$

where Inf result from divisions by 0 and NaN from undetermined forms as $0/0$. In that case, the pivoting becomes necessary. The algorithm implementation (in matlab) for the LU decomposition, with pivoting, yields the following result for the ill-posed situation from above:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/4 & 1/2 & 1 \end{pmatrix} \quad U = \begin{pmatrix} 4 & -3 & 1 \\ 0 & 11/2 & 17/2 \\ 0 & 0 & -3/2 \end{pmatrix} \quad \text{and} \quad P = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \tag{8}$$

The above results are identical to those obtained by the matlab *lu.m* function.

2- LUP decomposition algorithm

Listing 2: Matlab script for the LU decomposition with pivoting

```
function [L,U,P] = lu_decomposition(A)
sz=size(A);
L=eye(sz(1));
P=eye(sz(1));
U=A;
    if sz(1)~=sz(2)
        error('This is not a square matrix.');
    else
         for i=1:sz(1)
             [~,r] = max(abs(U(i:sz(1),i)));
             r=r+i-1;
             if r>=i
                 U([r,i],:)=U([i,r],:);
                 P([r,i],:)=P([i,r],:);
             end
             if i>=2
                L([r,i],1:i-1)=L([i,r],1:i-1);
             end
            for j=i:sz(1)-1
                L(j+1,i) = U(j+1,i)/U(i,i);
                U(j+1,:) = U(j+1,:)-L(j+1,i)*U(i,:);
            end
        end
    end
end
```

Problem 2

Equivalent resistance of infinite resistor grid

Problem 3

Power iterations methods

Iterative power methods provide solutions to eigenvalue problems by mean of algorithms giving sequences of approximate values that converge towards the true solutions of the eigenvalue problem. Usually, the complexity of the problem is of the order of $\mathcal{O}(mn^2)$ where n is dimensionality of the problem and m the number of iterations to reach the desired tolerance.

Here, we only detail the algorithm for the standard power method. A theoretical insight of inverse-power and Rayleigh quotient methods can be found in some literature as [**?**]. The power method is based on the very straightforward principle of applying a matrix several times (hence to a certain power) till a certain tolerance is reached:

1. Given the matrix A for the eigenvalue problem, define a random unit vector $q^{(0)}$
2. The first element of the values sequence is $\lambda^{(0)} = q^{*(0)}Aq^{(0)}$, where * stands for the adjoint of the element
3. While $|\lambda^{(k)} - \lambda^{(k-1)}| > \epsilon$, where ϵ is the tolerance threshold, repeat the following:
   - $q^{(k)} = Aq^{(k-1)}$
   - Normalize $q^{(k)}$
   - $\lambda^{(k)} = q^{*(k)}Aq^{(k)}$
4. Once the tolerance threshold is reached, λ is retained as the last value of the above sequence.

1- Power method

Listing 3: Matlab script for standard power method

```
function [vec,val] = eig_power(A)

    vec = rand(length(A),1); % Initialize container for result
    vec = vec./norm(vec);    % Normalize it
    val = ctranspose(vec)*A*vec;

    temp = val + 1;
    while abs(val-temp)>eps % machine precision epsilon
        vec = A*vec;
        vec = vec/norm(vec);
        temp = val;
        val = ctranspose(vec)*A*vec;
    end
end
```

The documentation on how to use each function is included in each submitted script in the form *MatlabScript.m* but has been removed here for the sake of saving space. However, each of them can be accessed upon typing the command *help FunctionName* in the Matlab command line.

The inverse power method proceeds to a shift in the direction of the initial guessed eigenvalue, before performing the power iterations with the inverse of the shifted matrix.

The Rayleigh quotient method is very close to the inverse power method. The difference resides in the fact that the estimated eigenvalue is replace by the Rayleigh quotient at the end of each iteration.

2- Inverse power method

Listing 4: Matlab script for inverse power method

```
function [vec,val] = eig_ipower(A,t)
    I     = eye(size(A));
    shift = A-t*I;
    vec   = rand(length(A),1); % Initialize container for result
    vec   = vec./norm(vec);    % Normalize
    val   = ctranspose(vec)*A*vec;
    temp = val + 1;

    while abs(val-temp)>eps
        vec = inv(shift)*vec;
        vec = vec/norm(vec);
        temp = val;
        val = ctranspose(vec)*A*vec;
    end
end
```

3- Rayleigh quotient method

Listing 5: Matlab script for the Rayleigh quotient method

```
function [vec,val] = eig_rq(A,t)

    I     = eye(size(A));     % Identity
    chara = A - t * I;        % Characteristic matrix (cf charact. poly.)
    vec = rand(length(A),1); % Generate random vector
    vec = vec./norm(vec);     % Normalize it
    val = ctranspose(vec)*A*vec;
    temp = t;

    while abs(val-temp)>eps  % machine precision epsilon
        vec = (eye(size(chara))/(chara))*vec; % Invert the matrix
        vec = vec/norm(vec);
        temp = val;
        val = ctranspose(vec)*A*vec;
    end

end
```

All the tests from the scripts $test_{p,ip,rq}.m$ have been passed successfully.

Problem 4

Vibrating modes of a string

Considering a vibrating string, and denoting the amplitude of the vibrations by u, one gets that the amplitude will have the following dependence in space and time $u \equiv u(x,t)$. Introducing the constants κ and ρ, corresponding respectively to energy and mass densities per unit length, the action of the problem in the harmonic approximation reads:

$$\S[u(x,t)] = \int_{t_1}^{t_2} dt \int_0^L dx \left(\frac{1}{2}\rho\Big(\frac{\partial u}{\partial t}\Big)^2 - \frac{1}{2}\kappa\Big(\frac{\partial u}{\partial x}\Big)^2 \right). \tag{9}$$

Taking the first variation of the action yields to the following Euler-Lagrange equation, namely the wave equation:

$$\frac{\partial^2 u}{\partial t^2} = \frac{\kappa}{\rho}\frac{\partial^2 u}{\partial x^2}. \tag{10}$$

Applying the following ansatz to decouple the space and time dependences $u(x,t) := v(t)w(x)$, one gets, plugging it in in Eq.(10),

$$\frac{v''}{v} = \frac{\kappa}{\rho}\frac{w''}{w}, \tag{11}$$

where $''$ represents the second order derivative w.r.t the variable it depends on. Since the left term depends on time and the right term depends on space, both terms have to be constant, so focusing on the space term w''/w, one gets that

$$\frac{w''}{w}\frac{\kappa}{\rho} = c \implies w'' = -\lambda w, \tag{12}$$

where $\lambda = -c\rho/\kappa$. Lambda then has the units of a mass/unit energy. If we want to avoid the trivial solution $w \equiv 0$, one has to impose $\lambda \neq 0$. Delving a bit more in the analytical structure of the solutions of Eq.(12), one sees that the solution need to be such that there exists a set of integers k such that $\lambda \propto k^2/L^2$. The second order space derivative from Eq.(12) can be numerically implemented using centered finite differences, having discretized space with N points. Hence, together with the minus term in front of λ, the latter can be rewritten as an eigenvalue problem:

$$A\mathbf{w} = \lambda\mathbf{w}, \tag{13}$$

where the $N \times N$ matrix A is defined as follows:

Problem 4 continued on next page...

$$A = \left(\frac{N-1}{2}\right)^2 \begin{pmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & \dots & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & 0 & \dots & -1 & 2 & -1 \\ 0 & 0 & 0 & \dots & -1 & 2 \end{pmatrix} \tag{14}$$

An obvious comment can be done about the structure of A, that gives out information about the nature of the eigenvalues of A: it is tridiagonal and such that $A^T = A$. So since A is an orthogonal matrix, there exists a basis of eigenvectors in which A can be diagonalized. Moreover, the eigenvalues can be ordered, such that they correspond to different energy states of the vibratory system. As of the vector $\mathbf{w}$, since we chose Dirichlet boundary counditions $u(0,t) = u(L,t)\ \forall t$, one gets that $w_1 = w_N = 0$.

Problem 5

2D quantum well

Let us represent a two-dimensional elliptical quantum well by the following potential, where $V_0 < 0$ is constant, and the ellipticity is set via the parameter c. In our numerical implementation, we'll choose $V_0 = -1.5$ eV and $r = 1$ nm.

$$V(x) = \begin{cases} V_0 < 0, & x^2 + y^2/c^2 < r^2 \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

The electrons moving inside the potential well are described by the time-independent Schrödinger equation, where $\hat{\mathcal{H}}$ is the hamiltonian of the system, ψ and E respectively the wave-function and the energy of the particle:

$$\hat{\mathcal{H}}\psi := -\frac{\hbar^2}{2m_e}\Delta\psi + \hat{V}\psi = E\psi, \tag{16}$$

where we denoted by m_e the electron mass, $\hbar$ is the reduced Planck constant and Δ the Laplace operator. On the 2D cartesian coordinates, the Laplace operator takes the form $\Delta := \partial^2/\partial x^2 + \partial^2/\partial y^2$. As of the implementation, defining a cartesian grid with $N \times N$ points, ψ and V can both be represented as matrices. The values of the matrix associated to ψ will evolve according to Eq.(16) whereas the potential matrix will be fixed. Denoting $\psi(x_i, y_j) = \psi_{i,j}$ and $V(x_i, y_j) = V_{i,j}$, the discretized Schrödinger equation reads:

$$-\frac{\hbar^2}{2m_e}\Delta\psi_{i,j} + V_{i,j}\psi_{i,j} = E\psi_{i,j}. \tag{17}$$

Now using centered finite differences to implement the Laplace operator, that is

$$\begin{aligned} \frac{\partial^2\psi}{\partial x^2} &= \frac{1}{(\Delta x)^2}\big(\psi_{i-1,j} - 2\psi_{i,j} + \psi_{i+1,j}\big) \\ \frac{\partial^2\psi}{\partial y^2} &= \frac{1}{(\Delta y)^2}\big(\psi_{i,j-1} - 2\psi_{i,j} + \psi_{i,j+1}\big) \end{aligned} \tag{18}$$

so Eq.(17) can finally be recast as

$$\begin{aligned} &-\frac{\hbar^2}{2m_e}\left\{ -2\Big(\frac{\Delta x^2 + \Delta y^2}{\Delta x^2 \Delta y^2}\Big)\psi_{i,j} + \frac{1}{\Delta x^2}\Big(\psi_{i-1,j} + \psi_{i+1,j}\Big) + \frac{1}{\Delta y^2}\Big(\psi_{i,j-1} + \psi_{i,j+1}\Big)\right\} \\ &+V_{i,j}\psi_{i,j} = E\psi_{i,j} \end{aligned} \tag{19}$$