Computational Physics III: LU decomposition

Due on May 04, 2023

Salomon Guinchard

Contents

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
```

In order to compare our implementation of the LUP decomposition with the Matlab's one, the relative error between the values of L and L_m and U and U_m are computed, and plotted in Fig.(1), as a function of the size of the matrices for our eigenvalue problem. The error is close to the machine epsilon, which proves that our scheme has been implemented correctly. Note the linear increasing trend with n. The fit shows a value at the origin which is negative. Of course, this result is purely numerical, due to the fact that some outliers biased the curve in a way that the error extrapolated for $n = 0$ would be negative, which makes no sense.

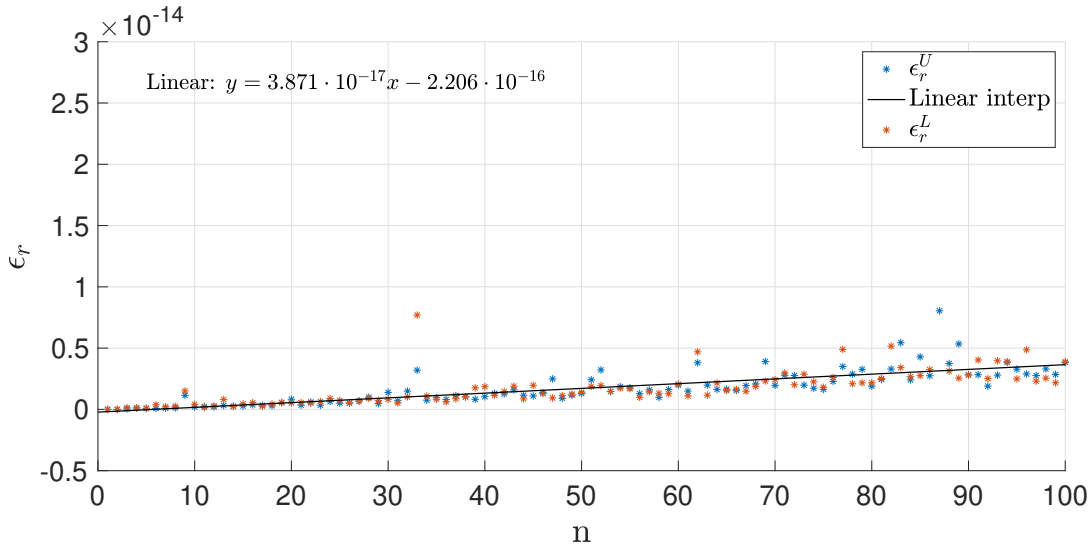

Figure 1: Relative error on the values for the upper and lower triangular matrices obtained via our LU decomposition scheme, and Matlab's one. The subscripts U and L in the legend correspond to upper and lower triangular respectively.

Problem 2

Equivalent resistance of infinite resistor grid

Let us consider a grid of size $N \times (N-1)$ (N odd) of resistors, all connected with their nearest neighbors, and all the same resistance of 1 Ω (see Fig.(2)). In addition, the following quantities are defined according to the following convention: for $1 \leq i \leq N$, $1 \leq j \leq N-1$, $V(i,j)$ is the voltage measured at node (i,j), $I_r(i,j)$ the current from node (i,j) to node $(i+1,j)$ to the right and $I_t(i,j)$ the current from node (i,j) to node $(i,j+1)$ above. Furthermore, open boundary conditions (B.C) are considered, i.e. no current can enter or flow out the system. Finally, the two points of interest are given by

$$A = \left(\frac{N-1}{2}, \frac{N-1}{2}\right), \quad B = \left(\frac{N-1}{2}+2, \frac{N-1}{2}+1\right) \tag{9}$$

and will be used to measure resistance. Indeed, a current of 1A is introduced at site A, while a current of 1A is extracted from the system at site B to ensure that the electrical current conservation.

Let us start by determining the number N_u of unknowns, V, I_r and I_t as well as the number N_c of constraints, given together by the boundary conditions, and Kirchoff's nodes law, and Ohm's law. An examination of Fig(2), with 5 nodes enabling an enumeration, it is possible to extrapolate this result to any $N_{nodes} = N$:

$$N_u = 3N(N-1), \qquad N_c = \underbrace{(N-1)^2 + N(N-2)}_{\text{Ohm's law}} + \underbrace{N(N-1)}_{\text{Law of nodes}} + \underbrace{2N-1}_{B.C} \tag{10}$$

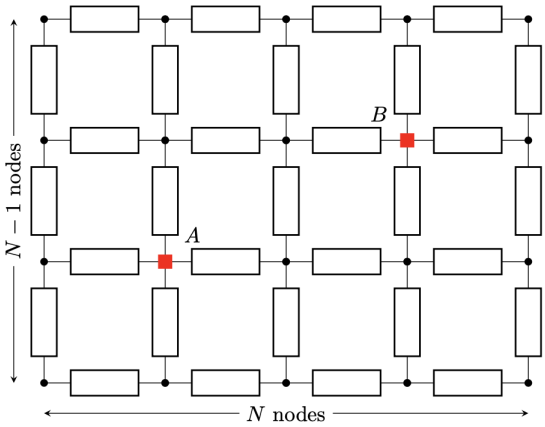

Figure 2: Configuration for the problem described above

In fact, take for example $N = 5$. It follows that $N_u = 3 * 5 * 4 = N_c = 60$, so the problem could hopefully be solved by a matrix inversion.

Let us thus rewrite the system in a matrix form: $A\mathbf{x} = \mathbf{b}$, where A is an $N_c \times N_u$ matrix, $\mathbf{x}$ a N_u-dimensional vector and $\mathbf{b}$ an N_c-dimensional vector. However, it appears (can be checked with Matlab), that the matrix A is singular, that means non invertible. This result was physically expected. The voltages $V(i,j)$ are defined on each node of the mesh, but since a voltage is a potential difference, it needs to be measured between two nodes. Thus, the voltage is a translational invariant, in the sense that $\Delta V = V(i+1,j) - V(i,j) \equiv V(i+1+\delta x, j) - V(i+\delta x, j)$. Hence, an infinite number of voltage values can fit.

Thus, the kernel of A has to be a 1-dimensional vector space. Matlab can provide an orthonormal basis for the kernel of A which is indeed a vector space of dimension 1 (one vector $\mathbf{v_0}$). Numerically, this can be verified since $A \cdot \mathbf{v_0} = \epsilon_m$, where ϵ_m is the machine precision epsilon (~ 0).

The way we decide to bypass this singular determinant problem is to project the problem on the orthogonal complement of $Ker(A)$. To do so, one needs the definition of a projection matrix P_V of size $[N(N-1)+1] \times N(N-1)$ as follows:

$$P_V(V_1, V_2, ..., V_N) = (V_2 - V_1, V_3 - V_1, ..., V_N - V_1) \tag{11}$$

The full projection matrix is a block diagonal matrix

$$P = \begin{pmatrix} P_V & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{pmatrix} \tag{12}$$

where I is a current matrix. It can be easily checked that $\det(PAP^t) \neq 0$. The P projection of the original system can be solved according to

$$PAP^t\mathbf{y} = P\mathbf{b}, \qquad \mathbf{x} = P^t\mathbf{y} \tag{13}$$

Now that $\mathbf{x}$ is known, one can obtain the value of the equivalent resistance R_{eq} between A and B for any N.Note that the current entering the system in A is fixed at 1A, and that the voltage between A and B varies, from Ohm's law, the equivalent resistance is computed to be:

$$R_{\text{eq}} = V\underbrace{\left(\frac{N-1}{2}, \frac{N-1}{2}\right)}_{\text{Point A}} - V\underbrace{\left(\frac{N-1}{2}+2, \frac{N-1}{2}+1\right)}_{\text{Point B}}, \quad \forall N \quad \text{odd} \tag{14}$$

Taking the particular case where $N = 7$, using $\mathbf{x}$ obtained previously, it comes that $R_{\text{eq}} = 0.8462\ \Omega$.

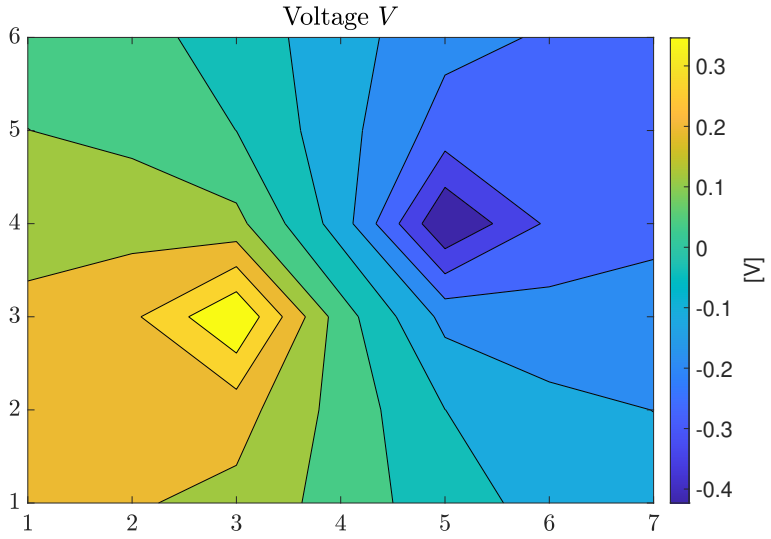

Figure 3: Voltage distribution along the grid $N = 7$, $A = (3, 3)$, $B = (5, 4)$

The voltage and the currents I_r and I_t can be plotted, as in Fig.(3)-(5). The voltage distribution is consistent with the expectations, since injecting a current in A and extracting it in B yields highest values in amplitude of voltage at those positions, in accordance with Ohm's law. The current distributions I_t and I_r are also consistent with our initial assumptions. The boundary conditions are respected in the sense that no current can escape or enter the system.

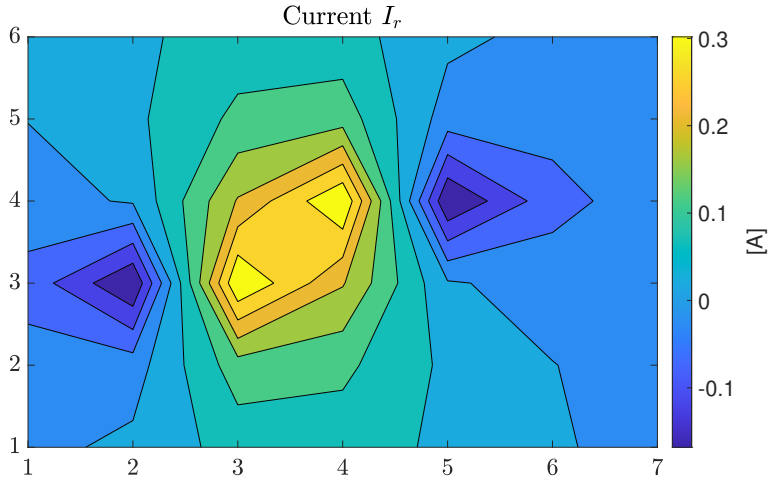

Figure 4: Current I_r distribution for $N = 7$

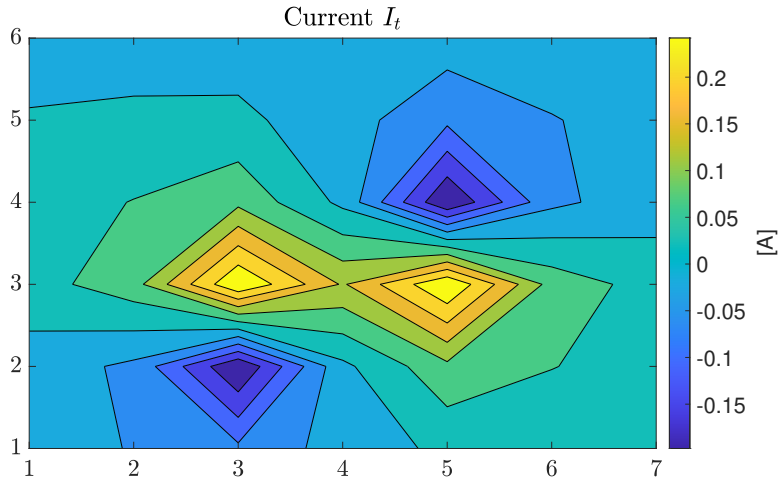

Figure 5: Current I_t distribution for $N = 7$

Finally, it can be verified that Kirschoff's law of nodes holds, i.e. that the algebraic sum of the currents on a node is zero. To verify the law, the sum of the currents flowing in and out of each node can be plotted, as shown in Fig(6). At points A and B, the currents are indeed 1A and -1A, respectively. Elsewhere, the current is zero, as predicted.

Finally, let us study the convergence properties of the equivalent resistance R_{eq} as a function of the size N of the system. More precisely, the question is to determine what size N of the system allows to reach an uncertainty smaller than 10^{-2} Ω on the equivalent resistance.

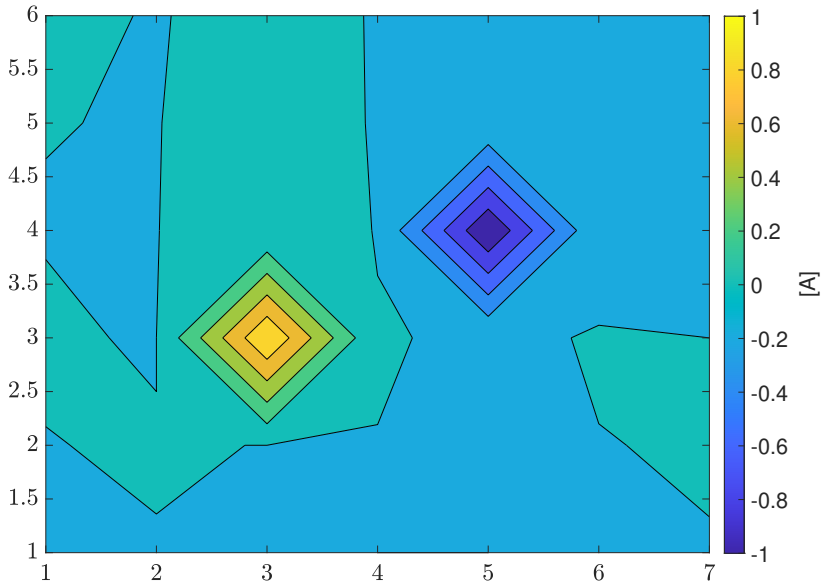

Figure 6:

In order to study the convergence, the size of the system is continuously increased, and the previously developed steps are repeated: generation of A such that $A\mathbf{x} = \mathbf{b}$, generation of P to project on the orthogonal complement of $Ker(A)$, solution of the system from Eq.(13), and finally computation of the equivalent resistance using Eq.(14). The result is shown in Fig.(7). The minimum required size is $N = 13$.

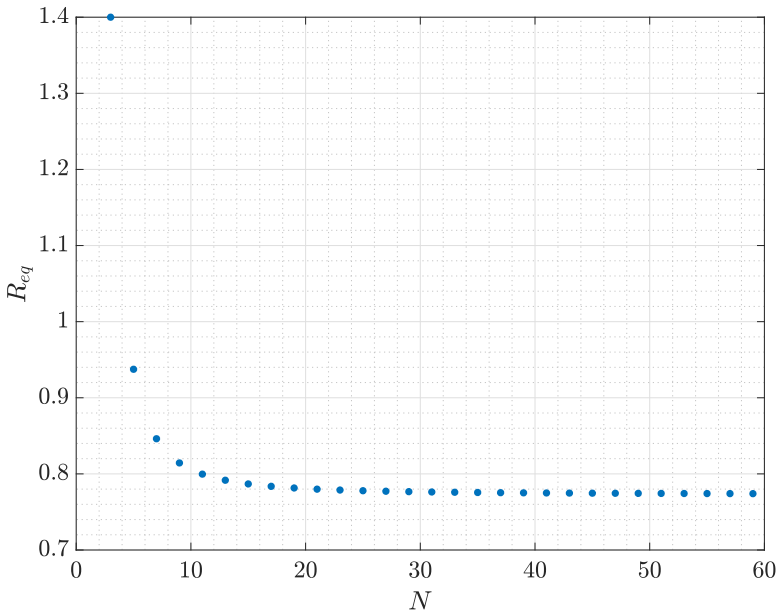

Figure 7:

Note that there exists an analytical solution for the same problem with an infinite grid, yileding $R_{\text{eq}} = 4/\pi - 1/2$. Thus, using the equivalent resistance obtained previously, an estimate of π follows: $\pi \approx 3.14$.

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:

$$\S[u(x,t)] = \int_{t_1}^{t_2} dt \int_0^L dx \left(\frac{1}{2}\rho\Big(\frac{\partial u}{\partial t}\Big)^2 - \frac{1}{2}\kappa\Big(\frac{\partial u}{\partial x}\Big)^2 \right). \tag{15}$$

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

$$\frac{\partial^2 u}{\partial t^2} = \frac{\kappa}{\rho}\frac{\partial^2 u}{\partial x^2}. \tag{16}$$

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.(16),

$$\frac{v''}{v} = \frac{\kappa}{\rho}\frac{w''}{w}, \tag{17}$$

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

$$\frac{w''}{w}\frac{\kappa}{\rho} = c \implies w'' = -\lambda w, \tag{18}$$

where $\lambda = -c\rho/\kappa$. Lambda has the units of m^{-2}, which corresponds to the a wave vector squared. 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.(18), one sees that the solution need to be such that there exists a set of integers k such that $\lambda \propto k^2/L^2$. In fact, the general solution of Eq.(18) can be expressed as:

$$w(x) = A\cos(\sqrt{\lambda}x) + B\sin(\sqrt{\lambda}x), \tag{19}$$

with A and B real valued amplitudes. Together with Dirichlet boundary conditions, one notices that A has to vanish, and that $\lambda = (k\pi/L)^2$, with k non-zero integers.

Problem 4 continued on next page...

To solve this equation, the second order space derivative from Eq.(18) 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:

$$A\mathbf{w} = \lambda\mathbf{w}, \tag{20}$$

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

$$A = \left(\frac{N-1}{2}\right)^2 \begin{pmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & \dots & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & 0 & \dots & -1 & 2 & -1 \\ 0 & 0 & 0 & \dots & -1 & 2 \end{pmatrix} \tag{21}$$

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 are strictly positive, and 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 $w_1 = w_N = 0$. The scalar factor in front of the matrix corresponds to $1/dx^2$, where dx is the distance between two meshpoints. Since for N mesh points, one has $N-1$ intervals, the length of one interval is $dx = L/(N-1) = 1/(N-1)$ in our context.

Now that the problem has been discretized, it is possible to find the lowest eigenvalue λ_0 with the inverse power method that we implemented in the previous part of the report, using a zero shift ($\mu = 0$). The numerically obtained value is $\lambda_0 = 9.7909\ \mathrm{m}^{-2}$. The corresponding vibrating eigenmode is plotted in Fig.(8).

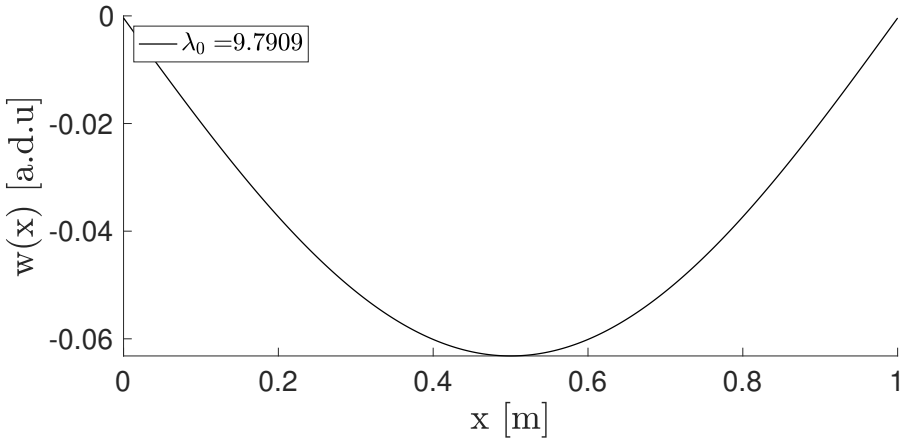

Figure 8: Eigenmode for lowest eigenvalue λ_0.

Note that the result corresponds to the first standing wave mode for a rope with Dirichlet boundary conditions. In Fig.(8), [a.d.u] means arbitrary distance unit, since the units depends on the discretization of the problem and haven't been normalized here.

Now that the lowest eigenvalue has been found, we can focus on higher eigenvalues. To do so, we use the inverse power method with a non-zero shift, and the shift is chosen, for each order, to be the values $\lambda_n := (n^2\pi^2/L^2)$. One notes that the first eigenmode for λ_1 is identical to the mode λ_0. For higher orders, that is higher values of n, we recover the usual standing wave modes, as pictured in Fig.(9).

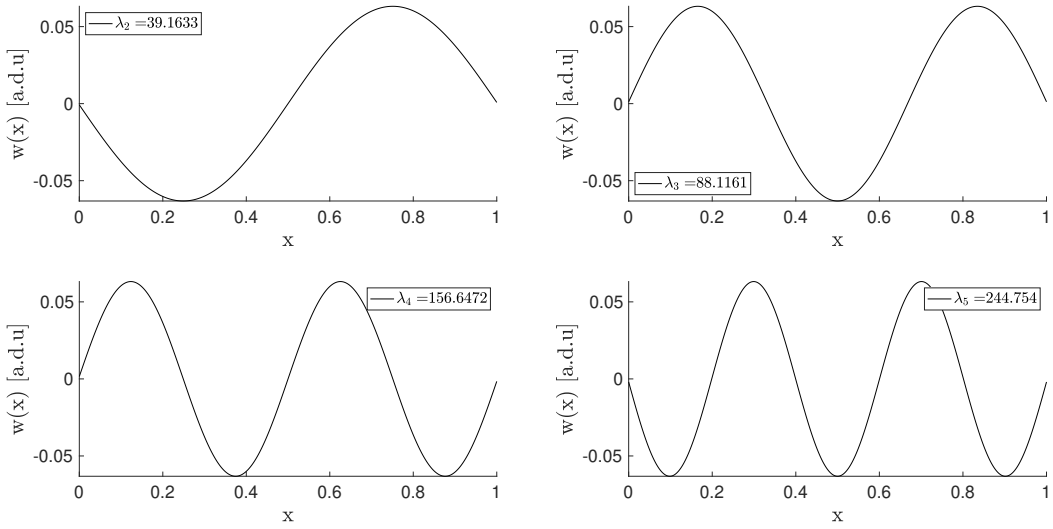

Figure 9: The four lowest values modes above the ground state λ_0. Note that n from λ_n is the number of peaks from each standing wave profile.

One will note that the eigenvalues here have been found with inverse power method with shift, assuming that the the eigenvalue were known to be the values λ_n from the solution of the wave differential equation. In fine, the method worked pretty well, because a comparison of the analytical λ_n and the numerically obtained values shows low error, as in Tab.(1).

Eigenvalues [m^{-2}]	λ_1	λ_2	λ_3	λ_4	λ_5
Analytical	9.8696	39.4784	88.8264	157.9137	246.7401
Numerical	9.7909	39.1633	88.1161	156.6472	244.7540

Table 1: Comparison between the eigenvalues obtained analytically and numerically.

Problem 5

Jacobi Method

Though effective, using the power iterative methods can be problematic as they only allow to determine one eigenvalue at a time. A more efficient way would be to diagonalize the matrix from the eigenvalue problem, which corresponds to finding an eigen-basis in which the matrix is diagonal, with its diagonal elements corresponding to the eigenvalues. In the following, we give two implementations of the Jacobi method, the second being the cyclic Jacobi algorithm, which supposedly performs way less rotations to diagonalize the matrix. A comparison of the two algorithms will be presented after the scripts are introduced.

The matrices treated by the Jacobi methods must be symmetric $N \times N$ matrices. The main idea behind these methods, implemented in the algorithms `eig_j.m` (Jacobi method) and `eig_cj.m` (cyclic Jacobi method), is to reduce the norm of the off-diagonal elements of the matrix A, define as $\text{off}(A) = (\sum_{i,j,i\neq j} a_{i,j}^2)^{1/2}$.

Jacobi & cyclic Jacobi method

Let us explain in more details how both these two methods work. First, it is a matter of identifying the off-diagonal element with the highest amplitude, i.e. finding (i, j) such that $|a_{i,j}| = \max_{l\neq m} |a_{m,n}|$. Once this pair is determined, the Jacobi matrix J_{ac} can be constructed before performing a Jacobi rotation on the matrix A until a certain tolerance is reached. Note however the high computational cost of finding the largest off-diagonal elements, since running through both the rows and the columns can only be done in $\mathcal{O}(n^2)$.

The cyclic Jacob method can drastically reduce the computational cost. Instead of finding the largest off diagonal elements, the method consists in running through the rows.The operating cost of this method is about $\mathcal{O}(n^3 \log(\log(|\epsilon|)))$, where ϵ_m is the machine precision.

In order to compare the efficiency of the two methods, the computational time needed to diagonalize random matrices of increasing size is plotted on the left part of Fig.(10), for both the standard and the cyclic Jacobi algorithms. On the right part of Fig.(10), it is the number of rotations required to perform the diagonalization which is represented. One immediately notices that the cyclic Jacobi is much more efficient.

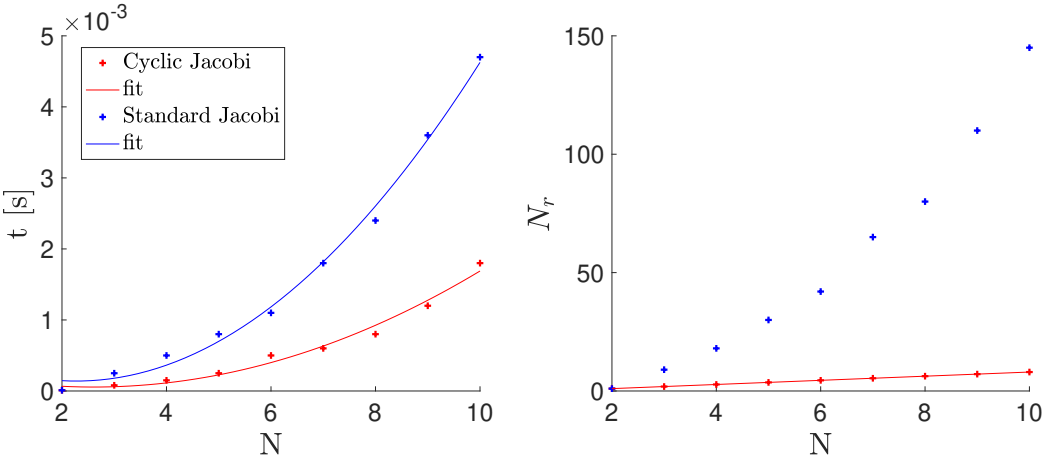

Figure 10: Left: Computational time for both cyclic Jacobi (red) and standard Jacobi (blue) methods - Right: same for the number of rotations.

Problem 6

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.

$$V(x) = \begin{cases} V_0 < 0, & x^2 + y^2/c^2 < r^2 \\ 0, & \text{otherwise} \end{cases} \tag{22}$$

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:

$$\hat{\mathcal{H}}\psi := -\frac{\hbar^2}{2m_e}\Delta\psi + \hat{V}\psi = E\psi, \tag{23}$$

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.(23) 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:

$$-\frac{\hbar^2}{2m_e}\Delta\psi_{i,j} + V_{i,j}\psi_{i,j} = E\psi_{i,j}. \tag{24}$$

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

$$\begin{aligned} \frac{\partial^2\psi}{\partial x^2} &= \frac{1}{(\Delta x)^2}\big(\psi_{i-1,j} - 2\psi_{i,j} + \psi_{i+1,j}\big) \\ \frac{\partial^2\psi}{\partial y^2} &= \frac{1}{(\Delta y)^2}\big(\psi_{i,j-1} - 2\psi_{i,j} + \psi_{i,j+1}\big) \end{aligned} \tag{25}$$

so Eq.(24) can finally be recast as

$$\begin{aligned} -\frac{\hbar^2}{2m_e}\Bigg\{ -2\Big(\frac{\Delta x^2 + \Delta y^2}{\Delta x^2 \Delta y^2}\Big)\psi_{i,j} + \frac{1}{\Delta x^2}\Big(\psi_{i-1,j} + \psi_{i+1,j}\Big) + \frac{1}{\Delta y^2}\Big(\psi_{i,j-1} + \psi_{i,j+1}\Big)\Bigg\} \\ +V_{i,j}\psi_{i,j} = E\psi_{i,j} \end{aligned} \tag{26}$$

Fig.(11) shows the representation of our hamiltonian operator as a sparse matrix. Each blue dot represents a non zero coefficient, acting on the $(1 \times N^2)$ vector containing each grid point. The diagonal corresponds to

Problem 6 continued on next page...

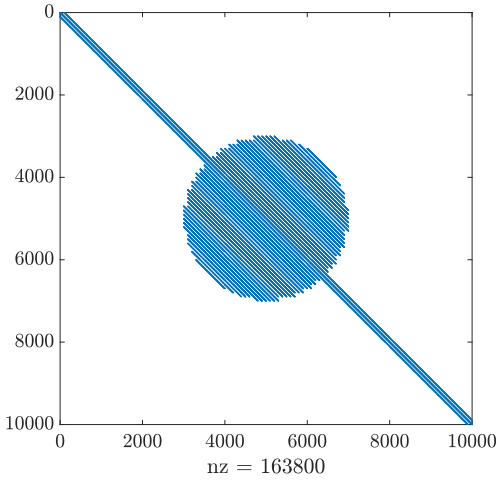

Figure 11: Representation of our hamiltonian operator as a sparse matrix - $c = 1$

the Laplacian term, and the circle to the potential region. Fig.(12) shows the same for a non zero ellipticity, that is for a parameter $c = 2 \neq 1$.

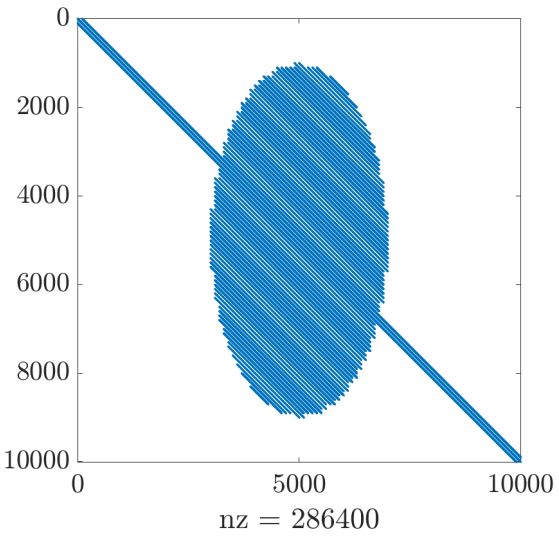

Figure 12: Representation of our hamiltonian operator as a sparse matrix - $c = 2$