Exercise sessions 6–7: LU decomposition and systems of linear equations.

Exercise 1: Implementation of the LU Decomposition [12 pts]

1. Solve the following system of linear equations:

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

   by performing the LU decomposition and backward substitution. Verify the solution explicitly by comparing with the one obtained using Matlab's matrix operations (ex. `X=A\B`).

2. Implement the LU decomposition of a square matrix. Test your implementation on random matrices of sizes up to 100. Compare your implementation with Matlab's `lu` function plotting the difference between the solutions provided by the two methods. *N.B.*: Matlab typically performs row permutations. Indeed, `[L,U,P] = lu(A)` gives L, U and the permutation matrix P. In order to compare properly, run your LU function on $A' = PA$, where P is taken from Matlab's output.

3. Test your implementation on the following matrix:

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 9 \\ 4 & -3 & 1 \end{pmatrix}$$

   Is your result meaningful? Comment on the origin of the observed problem (if any).

4. Implement the LU decomposition with pivoting. Test your implementation of LU decomposition on the above matrix. Your function should accept a square matrix A and return 3 square matrices L, U, P akin to the Matlab's function `lu`. Take a look into the supplied `test_lu.m` to make sure your implementation is correct.

Listing 1: 'lu_decomposition.m'

```
function [ L, U, P ] = lu_decomposition(A)

% lu_decomposition computes the LU decomposition with pivoting
% for a square matrix A
% Return lower L, Upper U and permuation matrix P such that P*A=L*U
[ L U P]=lu(A)  % Replace this line with your own implementation
                % not using Matlab's lu function
end
```

Submit: `lu_decomposition.m` [6 pts] and **ALL** other scripts. In the report, the correct solution of the system of linear equations (1), the comparison between your and MatLab's implementation of the LU decomposition (2) and the discussion (3), each give [2 pts].

Exercise 2: Equivalent resistance of an infinite resistor grid [18 pts]

Files: `test_ab.m, test_p.m`

Let us consider an infinite two-dimensional grid of resistors, all connected with their nearest neighbors, and all the same resistance of 1 Ω. The goal of this exercise is to compute the resistance between two nodes connected by a "knight move" (see Figure 1). In order to do this, finite grids of increasing size will be considered, and a convergence study will performed in order to estimate the infinite grid resistance.

More concretely, let us consider a $N \times (N-1)$ grid of nodes, for N odd. Setting the origin in the bottom left corner, one can define the following quantities, for $1 \leq i \leq N$, $1 \leq j \leq N-1$:

- $V(i,j)$ – voltage measured at node (i,j);
- $I_r(i,j)$ – current from node (i,j) to node $(i+1,j)$ on the right;
- $I_t(i,j)$ – current from node (i,j) to node $(i,j+1)$ above;

Figure 2 illustrates these definitions. Open boundary conditions are considered: no current can flow out or enter the system. The position of the two sites of interest is given by:

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

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

To measure the resistance, a current of 1 A is pumped into the system at site A, and 1 A is extracted at site B to ensure current conservation.

1. For an arbitrary size N, how many unknown quantities V, I_r and I_t do we have ? Write the constraints given by the boundary conditions, the Kirschoff's current law for each node, and the Ohm's law for each resistor. How many constraints do we have? A priori, is it possible to solve this problem with a simple matrix inversion ?

2. Cast these constraints into a system of linear equations $Ax = b$, where A is the $N_c \times N_u$ matrix, x the N_u-dimensional vector, and b the N_c-dimensional vector. N_c is the number of constraints in the

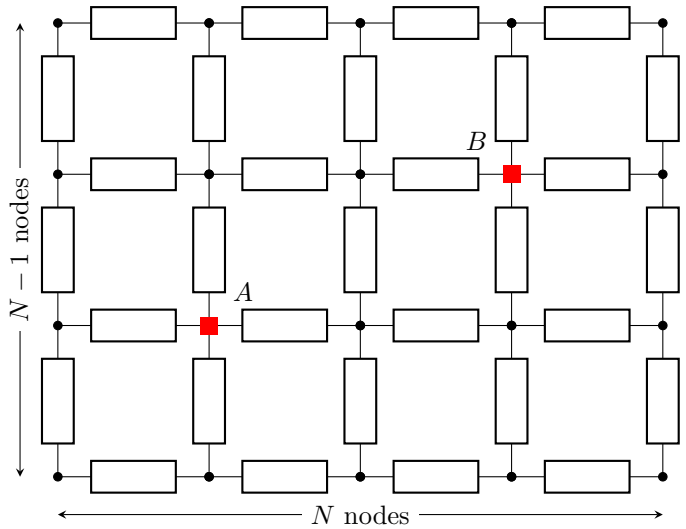


Figure 1: A finite grid of resistors for $N = 5$. The resistance is measured between two nodes marked in red.

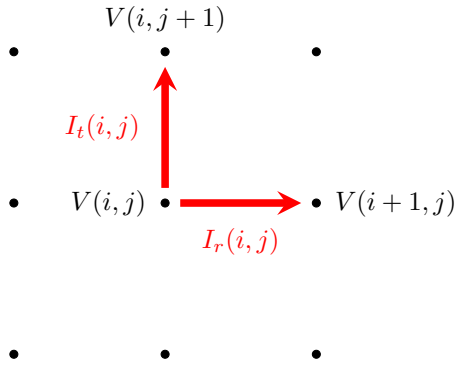


Figure 2: Definition of the voltage and currents for each node.

system, and N_u the number of unknown quantities. Write a subroutine `generate_ab.m` that generates A and b for a given system size N:

```
function [A, b] = generate_ab(N)
%% replace with the actual number of constraints and unknown quantities
Nc = 10
Nu = 42

%% replace with the actual expressions
A = zeros(Nc, Nu)
b = zeros(Nc, 1)
end
```

Use the following convention for the state vector x: the first dimension corresponds to $V(1,1)$, then the second to $V(1,2)$, etc. This is done so that x can be reshaped to a $N \times (N-1) \times 3$ tensor, the last index being $1, 2$ or 3 for respectively V, I_r and I_t[1]. See Equation 3 for clarity. Check that your script works using the provided `test_ab.m` script.

$$x = \Big[\underbrace{V(1,1), V(1,2), \ldots, V(1,N-1), V(2,1), \ldots,}_{N(N-1) \text{ voltages}} \underbrace{I_r(1,1), \ldots,}_{N(N-1) \text{ rightwards currents}} \underbrace{I_t(1,1), \ldots, I_t(N,N-1)}_{N(N-1) \text{ upwards currents}}\Big] \tag{3}$$

3. Verify numerically that the determinant of A is zero: the matrix is not invertible. Physically, why is it expected ? (i.e. why are there infinitely many solutions for the voltages ?). From a physical point of view, what should be the dimensions of the kernel of A ? Give it a basis, and verify it numerically.

4. One way to solve this issue is to project the problem on the subspace orthogonal to the kernel of A. On this subspace, A is invertible by definition. To do this, define the $[N(N-1)+1] \times N(N-1)$ projection matrix P_V as:

$$P_V(V_1, V_2, \ldots, V_n) = (V_2 - V_1, V_3 - V_1, \ldots, V_n - V_1) \tag{4}$$

[1]This order is the fastest to extract V, I_r and I_t without any cache miss and unnecessary copy of the data. It is due to the fact that MATLAB, like Fortran or Julia, and unlike Python or C/C++, is a column-major language (cf. `https://en.wikipedia.org/wiki/Row-\_and\_column-major\_order`).

This matrix only acts on the space of voltages. Lift it trivially to the space of voltages and currents by defining the full projection as a block diagonal matrix:

$$P = \begin{pmatrix} P_V & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{pmatrix} \tag{5}$$

Using the following template, write a function to generate the (full) projection matrix for a given system size N in a script `projection.m`, and verify it is correct using the provided `test_p.m` script.

```
function P = projection(N)
%% replace with the actual expression
P = zeros(3*N*(N-1)-1, 3*N*(N-1));
end
```

5. Verify that the projection of A, that is PAP^T, is invertible. Solve the system $PAP^T y = Pb$ (the projection of the original system by P). One can retrieve a state in the original full space with $x = P^T y$. Compute the value of the equivalent resistance between A and B for $N = 7$.

6. Plot the voltage, I_r and I_t. Verify graphically that the boundary conditions are respected. Also plot the sum of all currents flowing inside each node to verify that Kirschoff's law for currents is respected.

7. Perform a convergence study of the equivalent resistance as a function of the system size. How large N needs to be to have an uncertainty smaller than 10^{-2} Ω ?

8. An analytical solution exists for this problem with an infinite grid, yielding $R = \frac{4}{\pi} - \frac{1}{2}$ (see Ref.[2]). Using your results, estimate the value of π.

Submit: `generate_ab.m` [5 pts], `projection.m` [5 pts] and **ALL** other scripts. The report must include: discussion of the number of constraints and degrees of freedom [2 pts], discussion of the dimension of the kernel of A, and the physical interpretation [2 pts], plots of the currents and voltages [2 pts], convergence study of the computed resistance [2 pts].

[2] József Cserti. Application of the lattice Green's function for calculating the resistance of an infinite network of resistors (`https://arxiv.org/abs/cond-mat/9909120`).

Exercise sessions 8–9: Diagonalization algorithms and eigenvalue problems

Exercise 1 [4 pts] Power iteration methods

Files: `test_power.m, test_ipower.m, test_rq.m, rmg.m`

The power iteration methods for solving eigenvalue problems are very straightforward to implement: in order to obtain the eigenvector corresponding to a certain (smallest, largest, closest) eigenvalue of a matrix, one has to multiply an initial vector by a specific matrix multiple times.

Your task is to implement the power method, the inverse power method with shift and the Rayleigh quotient algorithm. The following is the template for the Rayleigh quotient function:

Listing 1: 'eig_rq.m'

```
function [vec,val] = eig_rq(input_matrix, target)

% eig_rq computes the closest eigenvector and eigenvalue
% of a given matrix.
%
% Arguments:
%
%     input_matrix (2D complex Hermitian matrix): matrix
%     for the eigenvalue problem;
%
%     target (real scalar): an estimation to the eigenvalue;
%
% Returns: a right eigenvector and the corresponding eigenvalue
% of a matrix.

% Replace these lines with your own implementation
vec = zeros(length(input_matrix),1);
val = 0;

end
```

For your convenience, we provide the test scripts your code has to pass, see the **Files** section.

Submit: Matlab functions `eig_power.m`, `eig_ipower.m` and `eig_rq.m` with the your implementations.

Exercise 2 [10 pts] Eigenmodes of a vibrating string

Consider a simple model of a vibrating string. Let $u(x,t)$ be the displacement field of a string with fixed boundary conditions, i.e. $u(0,t) = u(L,t) = 0$, where L is the length of the string. In the harmonic approximation, the action is given by

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

where ρ is the mass density of the string and κ is the elastic constant, both assumed to be homogeneous along the string. Using the least action principle, one can easily find the equations of motion for the displacement

field u

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

The goal of this exercise is to find numerically, using the inverse power method, the first few eigenmodes of vibration of the string.

1. Introduce the Ansatz $u(x,t) = v(t)w(x)$ and argue that the differential equation for the spatial function can be reduced to $-w'' = \lambda w$. What are the conditions on λ to obtain a non-trivial ($w(x) \neq 0$) solution? What are the units of λ and what physical quantity does it correspond to?

2. Discretize the problem by representing $w(x)$ using N equidistant points w_i. Approximate the second derivative with finite differences. From now on, assume $L = 1$ m, $\rho = 1/3$ kg/m, $\kappa = 3$ J/m and rewrite the problem at hand as an eigenvalue problem

$$A\vec{w} = \lambda\vec{w}\,. \tag{3}$$

   What is the form of the matrix A? What are its properties and what does it tell you about its eigenvalues? In your report, please explain your implementation of the boundary conditions.

3. Using the inverse power method previously implemented, find the smallest eigenvalue λ_0, give it correct units, and plot its corresponding eigenvector.

4. Using the inverse power method with shift, find the lowest four eigenvalues above λ_0 and plot their corresponding eigenvectors. Explain your strategy for finding these four lower eigenvalues.

Submit: Your report as a PDF file as well as your scripts. Answers to questions and explanations in (1) and (2) give [2 pts]. The remaining results in (3) and (4) give [3 pts] each.

Exercise 3 [6 pts] Jacobi method

Files: `test_j.m`, `rmg.m`

1. Implement the classic Jacobi method for calculating eigenvalues and eigenvectors of symmetric real matrices. Use the following function template:

Listing 2: 'eig_j.m'

```
function val = eig_j(input_matrix)

% eig_j computes the eigenvalues of a matrix.
%
% Arguments:
%
%     input_matrix (2D real symmetric matrix): matrix
%     for the eigenvalue problem;
%
% Returns: an array with eigenvalues.

% Replace the following with your implementation
val = zeros(length(input_matrix),1);

end
```

   Test your implementation using the test scripts provided.
   Hint: The efficiency of your Jacobi method code can be significantly improved by avoiding matrix multiplications in favor of vector operations.

2. Implement the cyclic Jacobi method. Compare the performance of the classic and cyclic algorithms by studying the following two quantities – computer time and the number of Jacobi rotations required for performing diagonalization – for a range of matrix sizes. Discuss the results obtained.

Submit: Your implementations of the Jacobi and cyclic Jacobi methods as functions `eig_j.m` and `eig_cj.m` ([2 pts] each); discussion of the performance in the report [2 pts].

Exercise 4: Two-dimensional quantum well [10 pts]

In solid-state systems, potential profiles can be tailored by means of varying chemical composition in order to spatially localize electrons. Consider a 2D quantum well, in which electrons are subject to the following potential profile:

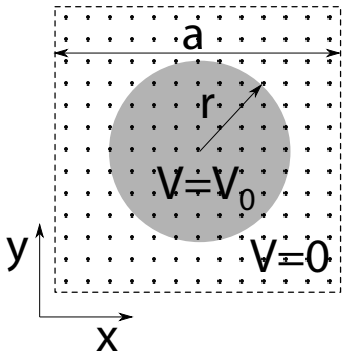
$$V(x,y) = \begin{cases} V_0 < 0, & x^2 + \frac{y^2}{c^2} < r^2 \\ 0, & \text{otherwise} \end{cases}$$

where $V_0 = -1.5$ eV is the width of the potential and $r = 1$ nm. Parameter $c = 1$ corresponds to the circular shape quantum well, while $c > 1$ makes it elliptic.

Compute the matrix representation of the Hamiltonian operator by discretizing space in 2D :

$$H = -\frac{\hbar^2}{2m_{\text{el}}}\Delta + V(x)$$

where m_{el} is the free electron mass and $\hbar$ is the reduced Planck's constant. The free electron mass is $m_{\text{el}} = 9.10938291 \cdot 10^{-31}$ kg. Other constants needed are: $\hbar = 1.05457172647 \cdot 10^{-34}$ J $\cdot$ s, 1 eV $= 1.60217656535 \cdot 10^{-19}$ J. Use a grid of $N \times N$ points in a square region defined by the range parameter a.



In the following, assume $N = 100$ and $a = 5r$.

1. Consider the case of $c = 1$, which correspond to the potential sketched in the figure. Find the number of bound solutions of this problem (i.e. negative eigenvalues). We recommend you to use Matlab's diagonalization routine `eigs` specially designed to deal with sparse matrices. Find the three lowest energy states, list their eigenvalues ϵ_i (in eV) and visualize the wavefunctions $\psi_i(\vec{r})$ as well as the probability densities $\psi_i^\star(\vec{r})\psi_i(\vec{r})$. You can use Matlab's function `pcolor` for this purpose. Comment on the results obtained.

2. Investigate one of the scattering solutions at a positive energy of approximately 1 eV. Visualize the wavefunction and the probability density. Calculate the probability of finding particle inside the quantum well for this state and compare this value with the one obtained for three bound solutions.

3. Consider the case of $c > 1$. Find the number of bound states as a function of c. Explain the observed trend.

4. Compare the wavefunctions and the eigenvalues corresponding to the three lowest energy states for $c = 1$ and some value $c > 1$ considered above. Explain the observed result.

Submit: your report as a PDF file as well as your scripts. The points are distributed in the following way:

- [4 pts] for the number of bound solutions, plots of the wavefunctions, probability densities and the discussion of results;
- [2 pts] for the calculation of the probability for (i) the three lowest states and (ii) the scattering state;
- [2 pt] for the plot of the number of bound states as a function of parameter c;
- [2 pt] discussion of the $c = 1$ and $c > 1$ cases.