This project consists of three parts. The first part is worth 60 points, the second part is worth 25 points, and the third part is worth 15 points. You can earn up to 20 extra points for presentation and quality of discussion.

You may work in teams of **up to three** persons. Code for your project should be submitted in either Java, Python, or MATLAB. (Choose only one of these languages to use for the whole project—do not mix and match.) Your final deliverable should be submitted in a single `.zip` archive file. The archive file should be uploaded to the dropbox on T-Square of *one* of your team members by **11:59 p.m.** on **March 31**. The file should contain:

- A `team_members.txt` file listing the name of each person in the project team.
- All source files for each part of the project you submit.
- A `.doc`, `.docx`, or `.pdf` file for each part of the project you submit, containing the written component of the project.
- A `readme` file for each part of the project you submit, explaining how to execute that part of the project.
- For Java or Python users, do not submit a project that has to be run on an IDE. The graders should be able to at least initialize your project on a command line interface, either Linux or Windows.

1 The Hilbert Matrix

1. (10 points) LU-decomposition. For the `lu_fact`, your code will be run against a number of test cases. The test cases will consist of $n \times n$ matrices A of floating-point numbers with linearly independent columns. The output should be two matrices: an $n \times n$ lower triangular matrix L and an upper triangular matrix U such that LU equals the input matrix; and the error $||LU - H||_\infty$. Your code will be inspected manually: no points will be awarded if the function does not implement the LU-decomposition algorithm.

2. QR-factorization. For each of the following functions, your code will be run against a number of test cases. The test cases will consist of $n \times n$ matrices A of floating-point numbers with linearly independent columns. The output should be two matrices: an $n \times n$ orthogonal matrix Q and an $n \times n$ upper-triangular matrix R such that QR equals the input matrix; and the error $||QR - A||_\infty$. Your code will be inspected manually: no points will be awarded if the function does not implement the specified QR-factorization algorithm.

    - (10 points) `qr_fact_househ`
    - (10 points) `qr_fact_givens`

3. The implementation to solve the linear system $A\overline{\mathbf{x}} = \overline{\mathbf{b}}$ based on LU or QR factorizations will be tested on $n \times (n+1)$ augmented matrices $[A|\overline{\mathbf{b}}]$ where A is $n \times n$ and $\overline{\mathbf{v}}$ is $n \times 1$. The output should be the solution vector $\overline{\mathbf{x}}_{sol}$, and the error $||A\overline{\mathbf{x}}_{sol} - \overline{\mathbf{b}}||$. Your code will be inspected manually: no points will be awarded if the function makes use of an inverse matrix algorithm, or if the solution vector is incorrect.
   - (5 points) `solve_lu_b`
   - (5 points) `solve_qr_b`
4. (10 points) Solution and errors for the Hilbert matrix of dimension $n \times n$, $n = 2, 3, \ldots, n$ and $\overline{\mathbf{b}}$ as described in the project. This should be a routine (or set of routines) that calls the appropriate programs and returns a readable output for each n.
5. (10 points) Plots and discussion. It should include plots of the errors obtained for the Hilbert matrix as a function of n. You should have 3 plots for the errors of $\overline{\mathbf{x}}_{sol}$, each obtained with LU and both QR decompositions; one for $||LU - H||_\infty$, and two more for $||QR - H||_\infty$ one for each case of QR-factorization. Your plots will probably look better in a Log scale, and can be grouped in a way that makes comparisons possible. The discussion should address the questions posted in the description of the project.

2 Convolutional codes

1. (4 points) Encoding problem. Your program should generate random binary input stream $\overline{\mathbf{x}}$ of length n determined by the user. Your program should generate the correct matrices A^0 and A^1, and output the encoded word $\overline{\mathbf{y}}$ as a binary stream of appropriate length.
2. Iterative methods for $A\overline{\mathbf{x}} = \overline{\mathbf{b}}$. For each of the following functions, your code will be tested against a number of test cases. The input will be augmented $n \times (n+1)$ matrices $[A|b]$ of floating point numbers, an initial guess vector $\overline{\mathbf{x}}_0$, and an error tolerance number tol. The output of your code should be the the number of iterations required to reach the tolerance; or a report that the method doesn't converge after a fixed number of iterations.
   - (6 points) `jacobi`
   - (6 points) `gauss_seidel`
3. (4 points) Decoding problem. For a given output binary stream $\overline{\mathbf{y}}$, find the input stream $\overline{\mathbf{x}}$ with each of the algorithms, Jacobi and Gauss-Seidel. Your program should produce $\overline{\mathbf{x}}$ as a binary stream.
4. (5 points) Discussion. Comparison of the methods. Address the role of n in each of case of the decoding algorithms.

3 Urban population dynamics

1. (6 points) Power method. Your implementation of the method will be tested against input matrices of $n \times n$, and tolerance number *tol* for which you will need to calculate the largest eigenvalue of the matrix with certain tolerance *tol*; or report that the method did not produce a result with a certain number of iterations.

2. (9 points) Discussion. Answer each of the questions provided in the project description, and discuss your answers.