AM205: Assignment 2 (due 5 PM, October 10)

Program files: A number of program and data files for this homework can be downloaded as a single ZIP file from the course website.

1. **Norms and Newton root-finding.** Define a matrix

$$A = \begin{bmatrix} 3 & -1 \\ 1 & 0 \end{bmatrix}, \tag{1}$$

which represents a linear transformation in $\mathbb{R}^2$.

   (a) Find four points $b \in \mathbb{R}^2$ such that $||b||_2 = 1$ and $||Ab||_2 = 1$. Plot the two curves $||x||_2 = 1$ and $||Ax||_2 = 1$ and mark the points b on this plot.

   (b) Find four points $c \in \mathbb{R}^2$ such that $||c||_\infty = 1$ and $||Ac||_\infty = 1$. Plot the two curves $||x||_\infty = 1$ and $||Ax||_\infty = 1$ and mark the points c on this plot.

   (c) Consider the function $f : \mathbb{R}^2 \to \mathbb{R}^2$ defined as

$$f(d) = (||d||_4 - 1, ||Ad||_4 - 1), \tag{2}$$

   where $d \in \mathbb{R}^2$. Write a program to find four solutions to the equation $f(d) = (0, 0)$ using the vector generalization of the Newton root finding method.[1] Plot the two curves $||x||_4 = 1$ and $||Ax||_4 = 1$ and mark the solutions d on this plot.

   (d) Show that the families of points b, c, and d (12 points in total) lie on two straight lines, and explain why this is true.

2. **LU factorization for binary numbers.** In this course we usually calculate using the set of real numbers $\mathbb{R}$. This question takes a different approach of calculating using the binary set $\mathbb{B} = \{0, 1\}$ with just two elements. Within $\mathbb{B}$, addition is defined as

$$0 + 0 = 0, \qquad 0 + 1 = 1, \qquad 1 + 1 = 0,$$

corresponding to regular addition, and then taking the remainder after division by two. Multiplication is defined as

$$0 \times 0 = 0, \qquad 0 \times 1 = 0, \qquad 1 \times 1 = 1.$$

Subtraction is the same as addition, so that $x - y = x + y$ for all $x, y \in \mathbb{B}$. For division, $x/1 = x$ for all $x \in \mathbb{B}$, and $x/0$ is undefined, giving a "division by zero" error. With these rules $\mathbb{B}$ becomes a *field*,[2] in that addition and muliplication have all the usual properties. In most cases, linear algebra calculations that work for numbers in $\mathbb{R}$ will work just as well for numbers in $\mathbb{B}$, as long as the arithmetic operations are interpreted as defined above.

[1] You can make use of linear algebra routines in your code, but the actual Newton iteration should be coded yourself, without using a library function.

[2] More specifically, $\mathbb{B}$ is the field of integers modulo p for $p = 2$, which is frequently studied in number theory courses.

There are various ways to write programs that calculate in $\mathbb{B}$. In the homework files, there are programs `bin_mul.py` and `bin_mul.m` that demonstrate this in Python and MATLAB, respectively. They both implement matrix multiplication in $\mathbb{B}$, using the test matrices

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}.$$

The two programs calculate LU and show that it is equal to A. We now consider adapting the algorithms presented in class to carry out the LU factorization to solve linear systems in $\mathbb{B}$.

(a) Write a function `fsolve` that takes a lower triangular matrix L and vector b and returns the solution x to the linear system $Lx = b$ using forward substitution described in the lectures. If any diagonal element of L is zero, the function should give an error and report that the matrix is singular.

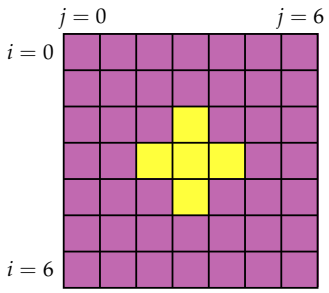
(b) Write a function `rsolve` that takes an upper triangular matrix U and vector b and returns the solution x the the linear system $Ux = b$ using reverse substitution described in the lectures. If any diagonal element of U is zero, the function should give an error and report that the matrix is singular.

(c) Write a program to calculate the LU factorization with partial pivoting as described in the lecture 7 slides. The program should return P, L, and U so that $PA = LU$, and it should also work on singular matrices.

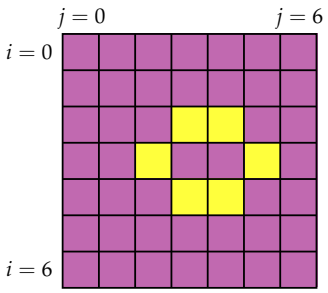
(d) In the homework files, there are two directories called `q2_small` and `q2_large`. Each has a text file containing a binary matrix A and a text file containing source data b. Find the solutions x to both linear systems $Ax = b$.

(e) **Optional.** What is the probability that a random $n \times n$ binary matrix will be singular?

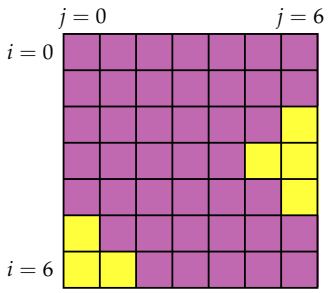
3. **The light game.** An electronic children's toy consists of a 7×7 grid of lights, which are initially all switched off. Pressing on a light toggles it on or off, and toggles its orthogonally adjacent neighbors on or off. A single press in the interior of the grid therefore creates set of lights in the shape of a plus sign, while several presses may lead to more complicated patterns. Three examples of the lights after different presses (i, j) are shown below.



Press $(3, 3)$



Press $(3, 3)$ and $(3, 4)$



Press $(6, 0)$ and $(3, 6)$

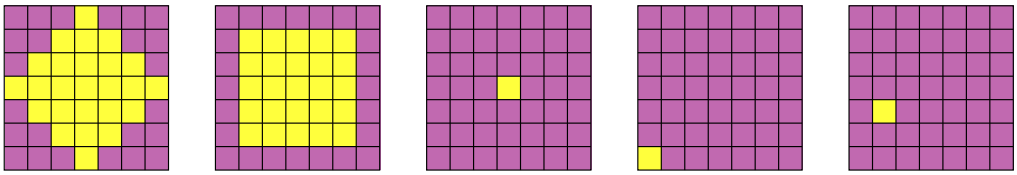
(a) Let x be a vector in $\mathbb{B}^{49}$ that represents which lights have been pressed, and let b be a vector in $\mathbb{B}^{49}$ that represents which lights are lit. Write a program that creates a 49×49

binary matrix A such that

$$Ax = b \tag{3}$$

in the binary arithmetic scheme introduced in Question 2.

(b) The toy presents different patterns of lights and the aim is to determine the correct presses to switch off all of the lights. For each of the patterns given below, use your binary LU solver from Question 2 to determine the correct presses.



For each case, present your results as in a 7×7 grid, such as by using the `spy` command in NumPy and MATLAB. In addition, create a light pattern of your own[3] and solve it.

(c) For the 7×7 grid the matrix A in Eq. 3 is non-singular, so that every combination of lights can be created with presses. However, this is not always the case for a general $m \times n$ grid. For each $m \times n$ grid with $m, n \in \{1, 2, \ldots, 9\}$ determine the dimension of the null space, $f(m, n) = mn - \text{rank } A$.[4]

(d) **Optional.** For the 5×5 grid, find two linearly independent press patterns that leave all of the lights switched off. For the 4×4 grid, find four linearly independent press patterns that leave all of the lights switched off.

(e) **Optional.** Suppose that you play the game on a polyhedron, so that pressing on a face lights up that face, as well as the neighboring faces that share an edge. For the five platonic solids, what is the dimension of the null space of A?

4. **Difficult cases for LU factorization.** Matrices of the form

$$G_n = \begin{bmatrix} 1 & 0 & 0 & 0 & \ldots & 1 \\ -1 & 1 & 0 & 0 & \ldots & 1 \\ -1 & -1 & 1 & 0 & \ldots & 1 \\ -1 & -1 & -1 & 1 & \ldots & 1 \\ \vdots & & & & \ddots & \vdots \\ -1 & -1 & -1 & -1 & \ldots & 1 \end{bmatrix} \tag{4}$$

in $\mathbb{R}^{n \times n}$ are examples of the very rare cases in which Gaussian elimination is unstable.

(a) Write a function `generate_g` that returns G_n.

(b) Write a program that measures the time $t(n)$ taken to run `generate_g` as a function of n, for $n = 10, 20, \ldots, 1000$. Make a plot of $t(n)$ as a function of n.[5] You should find that the $t(n) \sim \alpha n^{\beta}$. Determine α and β and discuss whether the value of β is reasonable, given the number of operations that `generate_g` does.

[3] If you wish to get creative, your pattern can be on a different-sized grid than 7×7. Any particularly awesome examples will be mentioned in class.

[4] Note that $f(m, n)$ is equal to the number of zero diagonal entries in U, in the LU factorization of A.

[5] For examples of how to time functions, see the `lu_time.py` and `chol_time.py` examples from lecture 8.

(c) For $n = 10, 20, \ldots, 200$, set $x = [1, 1, \ldots, 1]^T \in \mathbb{R}^n$ and construct a right-hand side vector $b = G_n x$. Solve the system $G_n \hat{x} = b$ using the LU factorization.[6] Plot the 2-norm relative error as a function of n and explain why we consider Gaussian elimination with partial pivoting to be numerically unstable in this case. In addition, make a plot that shows that the inequality

$$\frac{\|x - \hat{x}\|_2}{\|\hat{x}\|_2} \leq \kappa(G_n, 2) \frac{\|r(\hat{x})\|_2}{\|G_n\|_2 \|\hat{x}\|_2} \tag{5}$$

is satisfied, where $\kappa(G_n, 2)$ is the condition number of G_n with respect to the 2-norm, and $r(\hat{x}) = b - G_n \hat{x}$ is the residual.

5. **Singular Value Decomposition (SVD) and Principal Component Analysis (PCA).** Suppose we are given a data set represented by a matrix $X \in \mathbb{R}^{m \times n}$. One way to analyze this data is to perform PCA, which consists of computing the eigenvalue decomposition of the matrix XX^T. The eigenvectors of XX^T are referred to as principal components, and the eigenvalues related to the variance in the data set associated with the principal components.

   (a) We discussed the relationship between the eigenvalue decomposition of XX^T and the SVD of X in lectures. Based on this relationship, show that the principal components are the same as the left singular vectors of X.

   (b) It can be less accurate to perform PCA by computing eigenvalues of XX^T as opposed to using the SVD. For example, consider the matrix

$$X = \begin{bmatrix} 1 & 10^{-8} & 0 & 0 & 0 \\ 1 & 0 & 10^{-8} & 0 & 0 \\ 1 & 0 & 0 & 10^{-8} & 0 \\ 1 & 0 & 0 & 0 & 10^{-8} \end{bmatrix}. \tag{6}$$

   What are the exact eigenvalues of XX^T? You can either find these by hand, or you can make use of a symbolic computing environment like Mathematica or Maple.

   (c) Write a program to numerically compute (i) the eigenvalues of XX^T, and (ii) the squares of the singular values of X. Which method, (i) or (ii), is better in this case?

   (d) We will now make use of PCA to analyze the data provided in the file `q5.txt`. Recall from linear algebra that the orthogonal projection of a vector $x \in \mathbb{R}^m$ onto a subspace W of $\mathbb{R}^m$ is is defined as

$$v^\mathsf{T}(Px - x) = 0 \qquad \forall v \in W, \tag{7}$$

   where $Px \in W$ denotes the projection of x onto W. This formula states that the projection error $Px - x$ is orthogonal to W. The principal components of X provide an orthogonal basis that we can use to form a low-dimensional approximation to X via orthogonal projection. Compute the orthogonal projection of each column of X onto the first two principal components, and form a 2D scatter plot of the projected data. In your plot, the coefficients of the first and second principal components should correspond to the horizontal and vertical axes, respectively.

[6] In NumPy the routine `numpy.linalg.solve` uses the LU factorization. In MATLAB, the "backslash" operator uses the LU factorization.

(e) The total projection error in part (c) is given by

$$E_P = \left(\sum_{j=1}^{20} ||Px_j - x_j||_2^2 \right)^{1/2}, \tag{8}$$

where x_j denotes the jth column of X. Give an analytical formula for E_P that involves the singular values of X. *(Hint: For P denoting the orthogonal projection operator onto the span of the first two principal components, it can be shown that the matrix $[Px_1|Px_2|\ldots|Px_n] \in \mathbb{R}^{10\times 20}$ minimizes the total projection error, as defined in Eq. 8, among all rank 2 matrices.)*

In addition, calculate E_P directly based on your results for Px_j for $j = 1, \ldots, 20$ from (a). Verify that your calculated value agrees with the result of evaluating your analytical formula for E_P.