1. In the derivation of the LU decomposition algorithm the following result is used:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & \cdots & 0 \\
0 & 1 & 0 & 0 & \cdots & 0 \\
0 & 0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & 0 & \cdots & 1
\end{bmatrix}

^{-1} \begin{bmatrix}
1 & 0 & 0 & 0 & \cdots & 0 \\
0 & 1 & 0 & 0 & \cdots & 0 \\
0 & 0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & 0 & \cdots & 1
\end{bmatrix} =

\begin{bmatrix}
1 & 0 & 0 & 0 & \cdots & 0 \\
0 & 1 & 0 & 0 & \cdots & 0 \\
0 & 0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & 0 & \cdots & 1
\end{bmatrix}

\text{for } i = 1, 2, \ldots, n. \text{ Show that that it is correct.}

2. [465 only] AFCNM, Chapter 5, exercise 6.

2. [565 only] Assuming no pivoting is needed (to avoid breakdown or to ensure numerical stability), devise an efficient way to arrange the computations for solving an \(n\)-by-\(n\) linear system with non-zero entries in the coefficient matrix only in the first and last rows and columns and also in the two main diagonals. In the case of a 9 \(\times\) 9 matrix, the non-zero entries would appear as follows:

\[
\begin{bmatrix}
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\end{bmatrix}

\begin{bmatrix}
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\end{bmatrix}

\begin{bmatrix}
x \\
\end{bmatrix} =

\begin{bmatrix}
b \\
\end{bmatrix}.

\text{where } \bullet \text{'s represent a non-zero entry and blanks represent zero entries. It is sufficient to illustrate the efficient method for this } 9 \times 9 \text{ case. However, for the general } n\text{-by-}n \text{ case, determine the number of operations (additions/subtractions and multiplications/divisions) your algorithm requires (these should both be } O(n)\text{). Your method should not introduce a non-zero value in any step where there was previously (or initially) a zero.}

3. Let \(T\) be a (diagonally dominant) tridiagonal matrix, \(A\) be a symmetric positive definite matrix, and \(B\) and \(C\) be full nonsingular matrices. Assume all of these matrices are of size \(n\)-by-\(n\). Let \(f(x)\) be defined as follows

\[
f(x) = x^T B^{-1} C T^{-1} A^{-1} x + b^T B^{-T} x,
\]
where \( \mathbf{x} \) and \( \mathbf{b} \) are column vectors of size \( n \).

(a) Describe how to efficiently evaluate the function \( f(\mathbf{x}) \) using a combination of the various techniques we have discussed for solving linear systems (i.e. LU, Cholesky, and Crout (or Thomas)). Your description can be high level in terms of what algorithms you are going to use, but you must detail all steps required to compute \( f(\mathbf{x}) \), including when backward and forward substitutions should be used.

(b) Implement the algorithm you came up with in part (a) in MATLAB (or other suitable language) in a general manner so that it can be used for evaluating the function \( f(\mathbf{x}) \) for many different \( A, B, C, T, \mathbf{b}, \mathbf{x} \).

Note that the MATLAB command for computing an LU factorization of a matrix \( B \) with partial pivoting is \texttt{lu}:

\[
\text{[L,U,p]} = \text{lu}(B,’vectorize’);
\]

The \texttt{mldivide} (or \( / \)) function in MATLAB can be used with \( L \) and \( U \) to do the forward and backward substitution, e.g. to solve \( Bx = \mathbf{b} \) after the LU factorization is computed one can do:

\[
\mathbf{x} = U \backslash (L \backslash \mathbf{b}(p));
\]

The command for computing the Cholesky factorization is \texttt{chol}, e.g. if \( A \) is positive definite then \( \mathbf{R} = \text{chol}(A) \);

\[
\text{gives an upper triangular matrix } \mathbf{R} \text{ such that } \mathbf{R}^T \mathbf{R} = A. \text{ You can use } \mathbf{R} \text{ computed in this manner with the mldivide function to obtain a solution to } \mathbf{A} \mathbf{x} = \mathbf{b} \text{ similar to what was done for LU.}
\]

The Crout (or Thomas) algorithm can be applied to diagonally dominant, tridiagonal systems using the MATLAB sparse matrix tools, in particular the \texttt{spdiags} function. As an example, the tridiagonal system

\[
T = \begin{bmatrix}
-2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2
\end{bmatrix}
\]

can be created with the command:

\[
\text{T = spdiags(ones(10,1)*[1 -2 1,-1:1,10,10]);}
\]

Solving a system \( \mathbf{T} \mathbf{x} = \mathbf{b} \) with \( \mathbf{T} \) created in this manner is as simple as \( \mathbf{x} = \mathbf{A} \backslash \mathbf{b} \). This will use Crout’s algorithm to solve the system.

(c) Test your code using the following definitions (as given in MATLAB) for \( \mathbf{T}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{b}, \) and \( \mathbf{x} \), and some value of \( n \geq 10 \):

\[
\text{T = spdiags(ones(n,1)*[1 -2 1,-1:1,n,n]);}
\]

\[
\text{A = tril(rand(n));}
\]

\[
\text{A = A*A’;}
\]

\[
\text{B = rand(n);}
\]

\[
\text{C = rand(n);}
\]

\[
\text{b = rand(n,1);}
\]

\[
\text{x = ones(n,1);}
\]

Note: no where in your code should you actually compute the inverse of a matrix.
4. Pentadiagonal linear systems have the general form

\[
\begin{bmatrix}
a_1 & c_1 & e_1 & & \\
b_1 & a_2 & c_2 & e_2 & \\
d_1 & b_2 & a_3 & c_3 & e_3 & \\
 & d_2 & b_3 & a_4 & c_4 & e_4 & \\
 & & \ddots & \ddots & \ddots & \ddots & \\
 & & & d_{n-4} & b_{n-3} & a_{n-2} & c_{n-2} & e_{n-2} & \\
 & & & d_{n-3} & b_{n-2} & a_{n-1} & c_{n-1} & \\
 & & & d_{n-2} & b_{n-1} & a_n & 
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
\vdots \\
x_{n-2} \\
x_{n-1} \\
x_n
\end{bmatrix}
= \begin{bmatrix}
f_1 \\
f_2 \\
f_3 \\
f_4 \\
\vdots \\
f_{n-2} \\
f_{n-1} \\
f_n
\end{bmatrix}
\]

and arise in enough applications (although not as much as tridiagonal systems) that it makes sense to design special algorithms for solving them.

(a) Derive a fast algorithm for solving these systems using a similar approach to what was done in class and in the book for solving tridiagonal systems (you can assume no pivoting is necessary). Fast here means it should take \(O(n)\) operations.

(b) Determine the exact number of operations your algorithm requires for solving a general \(n\)-by-\(n\) pentadiagonal system.

(c) Implement your algorithm as a function in MATLAB (or other similar language) that takes as input the vectors \(a, b, c, d, e,\) and \(f\) containing the respective entries in the system above.

(d) Test your code from part (c) using the values \(a_i = i,\) for \(i = 1, \ldots, n,\) \(b_i = c_i = -(i+1)/3,\) for \(i = 1, \ldots, n-1,\) \(d_i = e_i = -(i+2)/6,\) for \(i = 1, \ldots, n-2,\) and \(f_1 = 1/2, f_2 = 1/6, f_i = 0,\) for \(i = 3, \ldots, n-2, f_{n-1} = 1/6,\) and \(f_n = 1/2.\) Use \(n = 100\) and \(n = 1000\) and verify that algorithm gives the correct answer.

5. Consider the linear system of equation \(Ax = b,\) where

\[
A = \begin{bmatrix}
1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\
1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\
1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\
1/5 & 1/6 & 1/7 & 1/8 & 1/9 \\
1/6 & 1/7 & 1/8 & 1/9 & 1/10
\end{bmatrix}, \quad b = \begin{bmatrix}
0.882 \\
0.744 \\
0.618 \\
0.521 \\
0.447
\end{bmatrix}
\]

Suppose you have in some way obtained the approximate solution vector to this system:

\[
\hat{x} = \begin{bmatrix}
-2.1333 \\
0.6258 \\
17.4552 \\
-11.8692 \\
-1.4994
\end{bmatrix}
\]

It is then easy to show that the residual becomes exactly

\[
\hat{r} = b - A\hat{x} = \begin{bmatrix}
-0.00001 \\
0.00001 \\
-0.00001 \\
0.00001 \\
-0.00001
\end{bmatrix}
\]

(a) Does this imply that \(\hat{x}\) is close to the exact solution \(x?\)

(b) Use MATLAB to obtain an accurate solution to the given system.

(c) Use MATLAB again to obtain a condition number for \(A.\) Use the appropriate result on perturbations of the right hand side (RHS) of a linear system to confirm that this very small residual indeed is big enough to allow for the solution to be as far away from the correct one as occurs in this example.
Hint: The $A$ matrix can be constructed easily in MATLAB using the function `hankel` (use MATLAB help to see why). The following code constructs $A$:

$$A = 1./\text{hankel}(2:6,6:10).$$

The vector $b$ can be entered into MATLAB manually:

$$b = [0.882 0.744 0.618 0.521 0.447]' .$$

The linear system $Ax = b$ can be solved in MATLAB using the `mldivide` command or simply the backslash operator “\”:

$$x = A\backslash b .$$

The condition number (with respect to the two-norm) can be computed using the MATLAB function `cond`:

$$\text{cond}(A) .$$

6. The growth factor for Gaussian elimination (see page 108 of AFCNM) applied to the matrix $A$ is given by

$$g_n(A) = \frac{\max_{i,j,k} \|a_{i,j}^{(k)}\|}{\max_{i,j} \|a_{i,j}\|} ,$$

where $a_{i,j}^{(k)}$ is the entry $(i, j)$ entry of $A$ at the $k$th step of Gaussian elimination.

(a) Add code to the `gepp.m` function (or an equivalent in another language) to compute the growth factor and return it as an output.

(b) Use your code to compute the growth factor of the 21-by-21 matrix

$$A = \begin{bmatrix}
1 & 0 & \cdots & \cdots & 0 & 1 \\
-1 & 1 & 0 & \cdots & 0 & 1 \\
-1 & -1 & 1 & \cdots & \cdots & 1 \\
\vdots & \ddots & -1 & \cdots & 0 & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & 1 \\
-1 & \cdots & \cdots & \cdots & -1 & 1
\end{bmatrix} .$$

Report the value you get and turn in your modified code. If you did things correctly, you should get $g_{21}(A) = 2^{20} .$

(c) [565 only] Show that the growth for the matrix in $A$ (but for the general $n$-by-$n$ case) is exactly $g_n(A) = 2^{n-1} .$

7. [565 only] Complete pivoting.

(a) Using the `gepp.m` code posted on the course webpage as a template, implement a function for doing Gaussian elimination with complete pivoting. Your function should return one matrix containing the upper and unit lower triangular matrices (similar to `gepp.m`), a vector $p$ representing the row permutations that were performed, and a vector $q$ representing the column permutations that were performed. A possible function declaration would look like

$$[\text{LU}, p, q] = \text{gecp}(A) .$$

(b) Using the `forsub.m` and `backsub.m` codes in conjunction with your `gecp` code from part (a), solve the linear system

$$A = \begin{bmatrix}
1 & 1 & 1^2 & \cdots & 1^{n-1} \\
1 & 2 & 2^2 & \cdots & 2^{n-1} \\
1 & 3 & 3^2 & \cdots & 3^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & n-1 & (n-1)^2 & \cdots & (n-1)^{n-1} \\
1 & n & n^2 & \cdots & n^{n-1}
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
\vdots \\
x_{n-1} \\
x_n
\end{bmatrix} = \begin{bmatrix}
1^{n-1} \\
2^{n-1} \\
3^{n-1} \\
\vdots \\
(n-1)^{n-1} \\
(n-1)^n
\end{bmatrix} ,$$

for $n = 12$. Report the max-norm of the residual in your computed solution and the max-norm of the error (you should easily be able to figure out exact solution).

(c) Compare your results from part (b) with the Gaussian elimination with partial pivoting, using, for example, the `gepp.m` code.