In this lab, we will examine the LU factorization and some other terminology in solving linear systems.

1 Permutation Matrices

Recall that a permutation matrix P is an identity matrix with the rows (or columns) swapped. A permutation matrix is orthogonal, thus it satisfies the relation

$$P^{-1} = P^T.$$

The matrix below is an example of a permutation matrix.

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

Use MATLAB to verify these properties

```
>> P = [0 1 0 0; 0 0 1 0; 0 0 0 1; 1 0 0 0]
>> A = rand(4)
>> PI = inv(P)
>> P'
>> P*P'
>> P'*P
>> P*A
>> A*P
```

For each of the last 5 commands, explain what you expect the result to be (and why) and state if the result agrees with your expectation.

Using a full $n \times n$ matrix to store a permutation matrix is very wasteful. A better method is to use a vector to represent a permutation matrix. If a vector $p(i) = j$, then the current row i was originally in position j. If we represent the above P in vector form, we get

$$p = \begin{pmatrix} 2 \\ 3 \\ 4 \\ 1 \end{pmatrix}.$$

How do we use a permutation vector to swap the rows? When P is a matrix, we can just compute $P \cdot A$, but this will not work when p is a vector since the dimensions do not match.

Recall that in MATLAB (and F90), a subscript must be an integer, but it does not have to be a scalar. A subscript can also be a vector of integers. Enter the following in MATLAB

```
>> p = [2 3 4 1]'
>> A(p,:)
>> A(:,p)
```

Explain the output from the last 2 commands.

2 LU Algorithm

Download the 2 MATLAB (`mylu.m` and `mylupiv.m`) files from the website into a working directory on your PC. The first of these functions implements Gaussian elimination without pivoting (*i.e.,* row swapping) and the second one uses partial pivoting with the pivoting information returned as a vector. Test the routines by entering the commands below

```
>> clear
>> A = rand(5)
>> [L,U] = mylu(A)    % No partial pivoting algorithm ( no row swapping)
>> [L1,U1,p1] = mylupiv(A)   % Partial pivoting algorithm
>> norm(A-L*U,1)
>> norm(A(p1,:) - L1*U1,1)
```

Recall that the norm (in this case the 1-norm) is a way of computing the magnitude of a matrix. Since we anticipate that $A = L * U$ and $A = P_1^T * L_1 * U_1$ (or equivalently, $P_1 A = L_1 U_1$), the norms $\|A - LU\|_1$ and $\|P_1 A - L_1 U_1\|_1$ should be 0 (or close to it). Answer the questions below:

a) Are L and L_1 the same?

b) What about U and U_1?

c) Was any pivoting done? If so what rows were swapped? Note that only the final row swaps are known and not the intermmediate ones.

d) How large are the norms? Does their size seem reasonable?

Enter the following additional commands:

```
>> [L2,U2,P2] = lu(A)
>> norm(A-P2'*L2*U2,1)
```

This is the intrinsic MATLAB LU routine. Compare L_1 and L_2. Are they the same? What about U_1 and U_2? What about p_1 and P_2? Were the matrices pivoted in the same way? Note that if you ever need to do an LU factorization, you should use the built-in MATLAB version as it will run much faster than the `mylupiv` routine.

3 Using LU to Solve Systems

Recall that the process for Gaussian elimination with partial pivoting is

1) Factor $PA = LU$

2) Solve $Lz = Pb$ for z.

3) Solve $Ux = z$ for b.

Note that if no pivoting is done, then $P = I$. A MATLAB version of the above steps can be expressed as:

```
>> clear
>> A = rand(4)
>> b = rand(4,1)
>> [L,U,p]=mylupiv(A)
>> z = L\b(p)  % Solve Lz = b(p) where b(p) is the permuted b.
>> x = U\z
```

The z vector is a temporary vector and is often of little interest, so the process below can also be used:

```
>> x = U\(L\b(p)) % Parentheses are important here
```

Compare the answer above with the one obtained from the intrinsic MATLAB backslash operator.

```
>> x1 = U\(L\b(p))
>> x2 = A\b
>> norm(x1-x2,1)
```

Again, because these are 2 different approaches for solving the same system, we expect that x_1 and x_2 should be the same. How large is the norm? Are the 2 solutions close to each other?

4 Assessing Accuracy

Both of the solutions computed in the previous section were close to each other, but how close to the exact solution are they? Because the system is randomly generated, the exact solution is not known and cannot be easily computed.

A common trick for assessing accuracy is to pick a b vector that produces a known exact solution, x_{exact}. An analogy can be drawn from the scalar linear equation case. Consider the equation

$$3x = b.$$

By picking the desired exact solution x, we can compute the required b. For example, if we want the exact solution to be $x = 5$, we can just compute $3 * 5 = b$, then solve $3x = 15$.

This can be generalized to the matrix case. Suppose we have a matrix A and we want the exact solution to $Ax = b$ to be a vector of all ones. Then we can do the following

```
>> clear
>> A = rand(4)
>> xexact = ones(4,1)
>> b = A*xexact
```

Now the vector b corresponds to an exact solution of all ones.

Test the accuracy of the 3 methods we have examined by doing the following

```
>> clear
>> A = rand(4)
>> xexact = ones(4,1)
>> b = A*xexact
>> [L,U] = mylu(A)
>> [L1,U1,p1] = mylupiv(A)
>> x1 = U\(L\b)
>> x2 = U1\(L1\b(p1))
>> x3 = A\b
>> error1 = norm(x1-xexact)/norm(xexact)
>> error2 = norm(x2-xexact)/norm(xexact)
>> error3 = norm(x3-xexact)/norm(xexact)
```

Are the norms small? Are the norms the same? If the norms are different, what does this say about the solutions x_1, x_2 and x_3?

5 Assignment

1) Recall that the operation counts for Gaussian Elimination can be expressed as:

   a) LU factorization: $\displaystyle\sum_{k=1}^{n-1}\left(\sum_{i=k+1}^{n}\left(1+\sum_{j=k+1}^{n}2\right)\right)$

   b) Solving $Lz = b$: $\displaystyle\sum_{i=2}^{n}\left(\sum_{j=1}^{i-1}2\right)$

   c) Solving $Ux = z$: $\displaystyle 1+\sum_{i=1}^{n-1}\left(1+\sum_{j=i+1}^{n}2\right)$

   These operation counts ignore the cost of searching for the pivot, performing row swaps and permuting the right hand side vector b. Using either Maple or Mathematica, compute the individual operation

counts as a function of n. Which of these 3 steps is the most expensive? What is the total operation count? HINT: To compute the sum

$$\sum_{i=1}^{n} i^2$$

in Mathematica, use the syntax

```
Sum[i^2,{i,1,n}]
```

To compute the sum in Maple, use the syntax

```
sum(i^2,i=1..n)
```

You might find the `Simplify` and `Factor` commands to also be helpful.

2) NOTE: Remember to use the semicolons on your MATLAB commands for this part. Using a random matrix of size 1000, compute the solution to the system $Ax = b$ (use the trick from Section 4 to generate the right-hand side vector) using the routines `mylu`, `mylupiv` and the intrinsic backslash operator.

   a) Is the solution obtained from using `mylu` more or less accurate than the one obtained from `mylupiv`?

   b) Of the 3 solution approaches (`mylu`, `mylupiv` and the intrinsic matlab \ operator), which is the most accurate?

3) The Hilbert matrix is a matrix whose elements are defined by

$$A_{i,j} = \frac{1}{i+j-1}.$$

You can generate an $n \times n$ Hilbert matrix in MATLAB using the `hilb(n)` command. Repeat the steps from Problem 2 using a 10×10 Hilbert matrix, then answer the following questions:

   a) Was any pivoting done by the `mylupiv` routine?

   b) Is there any difference in the solutions for `mylu` and `mylupiv`?

   c) Are the 3 computed solutions close to each other?

   d) Are they close to the exact solution? Why is this result surprising?