**LABORATORIO #4**

* [Introducción](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-4#INTRODUCCION)
* [Systemas Múltiples](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-4#SISTEMAS%20MULTIPLES)
* [Determinante de PLU](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-4#DETERMINANTE_DE_PLU)
* [Inversa de PLU](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-4#INVERSA_DE_PLU)
* [Sistemas transpuestos](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-4#SISTEMAS_TRANSPUESTOS)
* [Asignación](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-4#ASIGNACION)

**Introducción**

En el laboratorio anerior, se introdujo la idea de resolver un sistema lineal de ecuaciones **A\*x=b** al construir la factorización PLU de **A**.

A continuación se pondrá un poco más de atención a la factorización PLU. En particular, se verá como esta herramienta se puede utilizar para esolver múltiples sistemas lineales, calcular el determinente o inversa, u obtener la solución. Estas aplicaciones hacen de la factorización PLU una herramienta muy util.

**Sistemas múltiples**

La factorización **PLU** utiliza muchos recursos, por lo que se debe de considerar si se puede utilizar para otros propósitos. De hecho, un de usos es el de obtener la solución de sistemas múltiples, donde la matriz del sistema es fija pero el vector independiente toma varios valores.

Si ya se ha resuelto **A\*x1=b1**, entonces se puede hacer dos cosas:

* se factoriza **A** para obtener **P, L, U**;
* se utiliza los factores para resolver el sistema lineal.

El primer paso toma un tiempo. Si se quiere resolver otro sistema **A\*x2=b2**, no se necesita factorizar nuevamente la matriz **A**; ya se ha hecho, por lo que se pasa de inmediato a la fase de solución. Un ejemplo de la solución de dos sistemas sería:

**[ P, L, U ] = ge\_pp ( A )**   
**x1 = plu\_solve ( P, L, U, b1 )**   
**x2 = plu\_solve ( P, L, U, b2 )**

**Ejercicio** - use una factorización para resolver dos sistemas. Utilice la matriz de Frank de tamaño **n=5**, y realice:

**A = frank(n) x1 = [1 : n]'**   
**b1 = A \* x1**   
**b2 = A(:,n-2)**

Factorice **A** una vez, y resuelva el sistema 1, y después resuelva el sistema 2. Fíjese que se ha definido **b2**, pero si observa bien, se ha definido con la columna **n-2** de **A**. Puede Ud. decir cuál es la solución de **x2** a partir de esto?

If both right hand sides are available at one time, MATLAB would actually prefer to solve them together. To do this, we would have to set up a single **N** by **2** array for the right hand side, and make sure our *plu\_solve* routine was written properly to handle simultaneous multiple right hand sides.

**M File** - modify your *plu\_sol* routine so that it can handle multiple right hand sides, stored in a single array **b**. Here are some suggestions:

[ n, nrhs ] = size ( b ) ... *(figure out number of RHS's)*  
  
 z = P' \* b ... *(can stay the same!)*  
  
 y = zeros ( n, nrhs )  
 for k = 1 : nrhs  
 for j = 1 : n  
 y(j,k) = ?  
 for i = j+1 : n  
 z(i,k) = z(i,k) - L(i,j) \* y(j,k);  
 end  
 end  
 end  
  
 ...similar changes for the U code.

**Exercise** - test your modified solver. Use the **dif2** matrix of size **n=5**, and set

**A = dif2(n) b(:,1) = A \* [1:n]' b(:,2) = A(:,n-2)**

Factor **A** once, and then solve both linear systems by a single call to *plu\_sol*.

**Determinant from PLU**

The determinant is easy to compute because of the following facts:

* The determinant of a permutation matrix is +1 or -1;
* The determinant of a triangular matrix is the product of its diagonal elements;
* The determinant of a product is the product of the determinants;

In other words, we have:

**det ( A ) = det ( P ) \* det ( L ) \* det ( U ) = (+1 or -1) \* U11 \* U22 ... \* Unn.**

**M File** - Write a routine called *plu\_det.m* which computes the determinant of a matrix given its PLU factorization. The routine should have the form:

**det = plu\_det ( P, L, U )**

Copy the M file *p\_det.m* from the web page, which will compute the determinant of **P**. You can either call *p\_det* from within your routine, or simply copy the code into your routine. You should be able to write the rest of the routine yourself.

**Exercise** - Use your routine *plu\_det.m* to compute the determinants of a few matrices:

**Matrix N Determinant plu\_det(A)  
  
 DIF2 5 -6.0 \_\_\_\_\_\_\_\_\_\_\_  
 DIF2 20 21.0 \_\_\_\_\_\_\_\_\_\_\_  
 Frank 5 1.0 \_\_\_\_\_\_\_\_\_\_\_  
 Hilbert 5 3.7E-12 \_\_\_\_\_\_\_\_\_\_\_**

**Inverse from PLU**

We could try to compute the inverse of **A** by computing the inverses of each of the PLU factors and multiplying them out. (This isn't too hard.) But a better way, which is faster, and re-uses code we've already written and tested, is to construct the inverse by solving the following set of linear systems:

**A \* X = I**

where **I** is the identity matrix. (The identity matrix of order **n** is called **eye(n)** in MATLAB.) The solution matrix **X** will actually be the inverse. Because we've fixed up our *plu\_solve* routine, we can do this in a single operation.

**Exercise** - Create a routine called *inverse.m* which computes the inverse of a matrix. The routine should have the form

**function B = inverse ( A )**

Inside of your routine, what do you have to do in order to compute the inverse? Test your routine on the 5 by 5 **dif2** matrix, and compare your results to the output of the official MATLAB **inv** command, or the *dif2\_inv* routine I gave you.

**Transposed Systems**

I said that if you had the PLU factors of a matrix **A**, you could also use them to solve linear systems involving the transpose matrix, that is, equations of the form:

**A' \* x = b**

In the assignment, I'm going to ask you to try to write the code to do this, so pay attention!

Consider the fact that if

**A = P \* L \* U**

then the transpose matrix has the factorization

**A' = U' \* L' \* P'**

If we needed to solve systems involving **A'**, we could go ahead and compute the PLU factorization of **A'**, but with a little work, we can use this **U'L'P'** factorization instead.

*(PLU)' Solution Algorithm*: To solve the transposed linear system **A'\*x=b**, given factors **P**, **L**, **U** (which are factors for the untransposed matrix) and right hand side **b**,

1. Solve **U' \* z = b**;
2. Solve **L' \* y = z**;
3. Solve **P' \* x = y**.

**Assignment**

**Assignment** - Make a copy of your code *plu\_solve.m*, calling it *plut\_solve.m*. Modify this code so that it can solve a linear system involving the transposed matrix, given PLU factors of the original matrix.

*Design suggestions*: - To start this code, simply take the three sections of your original program, and reverse their order. Now rewrite each section so that it's solving the transposed equation. Start with the transposed **P** system, which is easy. To solve the transposed **L** system, you have to realize that **L'** is a unit upper triangular matrix, and that its (I,J) entry is stored in **L(J,I)**. So essentially, rewrite the code to look like your upper triangular solve code, but adjust for the fact the your matrix now has a unit diagonal, and the actual elements have the indices reversed. Finally, try to set up the **U'** solver, which looks like your old **L** solver, but now the lower triangular matrix doesn't have a unit diagonal, and, again, the entry indices are flipped.

**Test**: When you think your code is working, carry out the following test:

**n = 5 A = frank ( n )**   
**[P,L,U] = ge\_pp ( A )**  
**x = [1:n]'**   
**b = A' \* x**   
**x2 = plut\_solve ( P, L, U, b )**

*(Do not try your code on the Hilbert or dif2 matrices. They are bad examples for this problem, because they are symmetric!)*

**Results**: If your code has passed the test I just descrbed, it is probably working correctly. I want to take a look at your code (that's all). Mail your **plut\_solve.m** code to me.