```
function myplu(A)
    m,n=size(A)
    if m!=n
        println("Need a square matrix!")
        throw(exit())
    end
    U=zeros(size(A))
    L=zeros(size(A))
    p=zeros(Int,n)
    Ak=copy(A)
    for k=1:n
#       display(Ak)
        i=argmax(abs.(Ak[:,k]))
        p[k]=i
        U[k,:]=Ak[i,:]
        println("About to divide by ",U[k,k])
        L[:,k]=Ak[:,k]/U[k,k]
        Ak=Ak-L[:,k]*U[k,:]'
    end
    return L[p,:],U,p
end
```

divide by max in the column tends to control rounding errors that were made earlier in the computation...

return p since LU ≠ A anymore...

↑ swaps the rows back so L is lower triangular

```
julia> Atrouble=[2 0 4 3 ; -2 0 2 -13; 1 15 2 -4.5 ; -4 5 -7 -10]
4×4 Matrix{Float64}:
  2.0   0.0   4.0    3.0
 -2.0   0.0   2.0  -13.0
  1.0  15.0   2.0   -4.5
 -4.0   5.0  -7.0  -10.0

julia> L,U,p=myplu(Atrouble);
About to divide by -4.0
About to divide by 16.25
About to divide by 5.538461538461538
About to divide by -0.16666666666666664
```

```
julia> L
4×4 Matrix{Float64}:
  1.0     0.0        0.0         -0.0
 -0.25    1.0        0.0         -0.0
  0.5    -0.153846   1.0         -0.0
 -0.5     0.153846   0.0833333    1.0

julia> U
4×4 Matrix{Float64}:
 -4.0   5.0    -7.0       -10.0
  0.0  16.25    0.25       -7.0
  0.0   0.0     5.53846    -9.07692
  0.0   0.0     0.0        -0.166667
```

```
julia> p
4-element Vector{Int64}:
 4
 3
 2
 1
```

← how rows are swapped...

```
julia> L*U
4×4 Matrix{Float64}:
 -4.0   5.0          -7.0   -10.0
  1.0  15.0           2.0    -4.5
 -2.0  -1.38778e-16   2.0   -13.0
  2.0   1.38778e-16   4.0     3.0
```

rounding error...

```
julia> Atrouble[p,:]
4×4 Matrix{Float64}:
 -4.0   5.0   -7.0   -10.0
  1.0  15.0    2.0    -4.5
 -2.0   0.0    2.0   -13.0
  2.0   0.0    4.0     3.0
```

```
julia> using LinearAlgebra
```

```
julia> Z=lu(Atrouble)
LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
4×4 Matrix{Float64}:
  1.0     0.0        0.0        0.0
 -0.25    1.0        0.0        0.0
  0.5    -0.153846   1.0        0.0
 -0.5     0.153846   0.0833333  1.0
U factor:
4×4 Matrix{Float64}:
 -4.0   5.0    -7.0      -10.0
  0.0  16.25    0.25      -7.0
  0.0   0.0     5.53846   -9.07692
  0.0   0.0     0.0       -0.166667

julia> Z.p
4-element Vector{Int64}:
 4
 3
 2
 1
```

built in to library

Same choice of pivots used for builtin routine.

The built-in routine is also $O(n^3)$ running time where $n \times n$ the size of matrix A

I think...

There are algorithms that solve $Ax=b$ in fewer than $O(n^3)$ operations

That algorithm is so complicated that the problem has to be very very very big for it to be worthwhile.

More information next time...

Matrix norms & Vector Norms...

Vector Norms.

Euclidean Norm

$x \in \mathbb{R}^n$ then $\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$

distance of x to the origin.

generalized to vectors of length n.

Pythagorean theorem.

p-norm

$x \in \mathbb{R}^n$ then $\|x\|_p = \sqrt[p]{|x_1|^p + |x_2|^p + \cdots + |x_n|^p}$

when $p=2$ we get Euclidean norm.

$$\|x\|_2 = \sqrt{x \cdot x}$$

this structure involves dot products (and subsequently geometry)

1-norm

$$\|x\|_1 = |x_1| + |x_2| + \cdots + |x_n|$$

simpler to compute... it also measures the "size" of a vector... just differently that Euclidean norm...

∞-norm

$$\|x\|_\infty = \lim_{p \to \infty} \sqrt[p]{|x_1|^p + |x_2|^p + \cdots + |x_n|^p}$$

$$= \max\{|x_1|, |x_2|, \ldots, |x_n|\}$$

intuitively the larger p is the more there there is between $|x_i|^p$ for different i's. and the largest one contributes the most...

graduate students try to prove this limit.

Matrix norms $A \in \mathbb{R}^{n \times n}$

$$\|A\|_F = \sqrt{\sum_{i,j=1}^{n} |A_{ij}|^2}$$ **Frobenius norm:**

Definition 2.7.4 : Induced matrix norm

Given a vector norm $\|\cdot\|_p$, we define an **induced matrix norm** for any $m \times n$ matrix $\mathbf{A}$ as

$$\|\mathbf{A}\|_p = \max_{\|\mathbf{x}\|_p=1} \|\mathbf{A}\mathbf{x}\|_p = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|_p}{\|\mathbf{x}\|_p}.$$

$\|A\|_2 = \max\{\|Ax\|_2 : \|x\|_2 = 1\}$ = not quite as simple formula related to eigenvalues

$\|A\|_1 = \max\{\|Ax\|_1 : \|x\|_1 = 1\}$ = Simple formula

$\|A\|_\infty = \max\{\|Ax\|_\infty : \|x\|_\infty = 1\}$ = Simple formula

```
julia> x=randn(4); xunit=x/norm(x); norm(Atrouble*xunit)
7.135129520059524

julia> x=randn(4); xunit=x/norm(x); norm(Atrouble*xunit)
8.187172882893039

julia> x=randn(4); xunit=x/norm(x); norm(Atrouble*xunit)
13.721318669956165

julia> x=randn(4); xunit=x/norm(x); norm(Atrouble*xunit)
16.23123686821306

julia> x=randn(4); xunit=x/norm(x); norm(Atrouble*xunit)
13.233650481237047

julia> x=randn(4); xunit=x/norm(x); norm(Atrouble*xunit)
12.737534744139348
```

```
julia> x=randn(4); xunit=x/norm(x); norm(Atrouble*xunit)
18.68762368972806
```

even bigger...

could approximate maximum for any p-norm using guess and check...

Builtin routine in Julia, opnorm

```
julia> opnorm(Atrouble)
20.209531674397468
```

The paper under UNR materials has code to compute $\|A\|_p$ for any $p>1$. Extra Credit and graduate students create a Julia code that computes $\|A\|_p$ using the same method.