2.5. Efficiency of matrix computations

Definition 2.5.1 : Asymptotic notation

Let $f(n)$ and $g(n)$ be positive-valued functions. We say $f(n) = O(g(n))$ (read "f is **big-O** of g") as $n \to \infty$ if $f(n)/g(n)$ is bounded above as $n \to \infty$.

We say $f(n) \sim g(n)$ (read "f is **asymptotic** to g") as $n \to \infty$ if $f(n)/g(n) \to 1$ as $n \to \infty$.

$f(n) = n^2 + 100n + 2000$

Claim $f(n) = O(n^2)$ as $n \to \infty$

means $\dfrac{f(n)}{n^2}$ is bounded above as $n \to \infty$

$$\lim_{n\to\infty} \frac{n^2 + 100n + 2000}{n^2} = \lim_{n\to\infty} \left(1 + \frac{100}{n} + \frac{2000}{n^2}\right) = 1$$

bounded

count how many mult to compute

Matrix vector product: Ax where $A \in \mathbb{R}^{n\times n}$, $x \in \mathbb{R}^n$

$$\begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_n^T \end{bmatrix} x = \begin{bmatrix} a_1^T x \\ a_2^T x \\ \vdots \\ a_n^T x \end{bmatrix} = \begin{bmatrix} a_1 \cdot x \\ a_2 \cdot x \\ \vdots \\ a_n \cdot x \end{bmatrix}$$

n dot products
each dot product includes n multiplications and n-1 additions...

total mult is n^2

Alternatively

$$\begin{bmatrix} u_1 & u_2 & \cdots & u_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \overbrace{x_1 u_1 + x_2 u_2 + \dots + x_n u_n}^{n \text{ terms}} = \sum_{k=1}^{n} x_k u_k$$

scalar-vector mult has n multiplications

total mult is n^2.

```
julia> A=rand(5,5)
5×5 Matrix{Float64}:
 0.06591    0.51884    0.242821  0.635408   0.768054
 0.166573   0.841503   0.792264  0.101489   0.765552
 0.638743   0.0827899  0.590156  0.988749   0.180407
 0.933442   0.0659737  0.130125  0.576933   0.293069
 0.945042   0.0258474  0.698698  0.0742852  0.625173
```

5×5 matrix with elements distributed uniform on [0,1].

```
julia> A=randn(5,5)
5×5 Matrix{Float64}:
 0.635926  -1.2457     0.681756  -0.690417  -0.23843
 1.57041    0.913337  -0.674098  -0.916852   0.226277
 0.762292  -0.750406   1.1966     1.03318    0.0188612
 0.76839   -0.955926  -0.740413   0.893575   0.328992
 0.223616   1.35826   -1.08936    0.536711  -1.628
```

5×5 matrix with elements distributed according to a standard normal distribution.

```
julia> x=randn(5)
5-element Vector{Float64}:
  1.072342927649603
 -1.9156173536176098
  0.8729940313426149
 -1.0867791538055305
  2.323197689693588
```

```
julia> A*x
5-element Vector{Float64}:
  3.8597914689011867
  0.8680338411588717
  2.220538329006034
  1.8019858452776782
 -7.678574011792943
```

Question: How long did this computation take?

```
julia> @elapsed A*x
1.5624e-5
```

```
julia> @elapsed A*x
1.6111e-5

julia> @elapsed A*x
1.532e-5
```

similar...

Lots of time must be overhead...

```
julia> @elapsed for n=1:100; A*x; end
5.8694e-5
```

only 5x longer to perform 100 matrix-vector multiplications.

```
julia> @elapsed for n=1:10000; A*x; end
0.002790617
```

```
julia> t100=@elapsed for n=1:100; A*x; end
4.3225e-5

julia> t10000=@elapsed for n=1:10000; A*x; end
0.00409189

julia> t10000=@elapsed for n=1:10000; A*x; end
0.002789153
```

not so predictable

64 times longer

94 times longer

The asymtotic analysis with $O(n^2)$ is when $n \to \infty$ and $n=5$ is not big enough...

```
julia> ns = 1000:1000:5000
       ts = []
       for n in ns
           A = randn(n,n)
           x = randn(n)
           time = @elapsed for j in 1:80; A*x; end
           push!(ts,time)
       end
```

← range of $n \to \infty$

← empty list

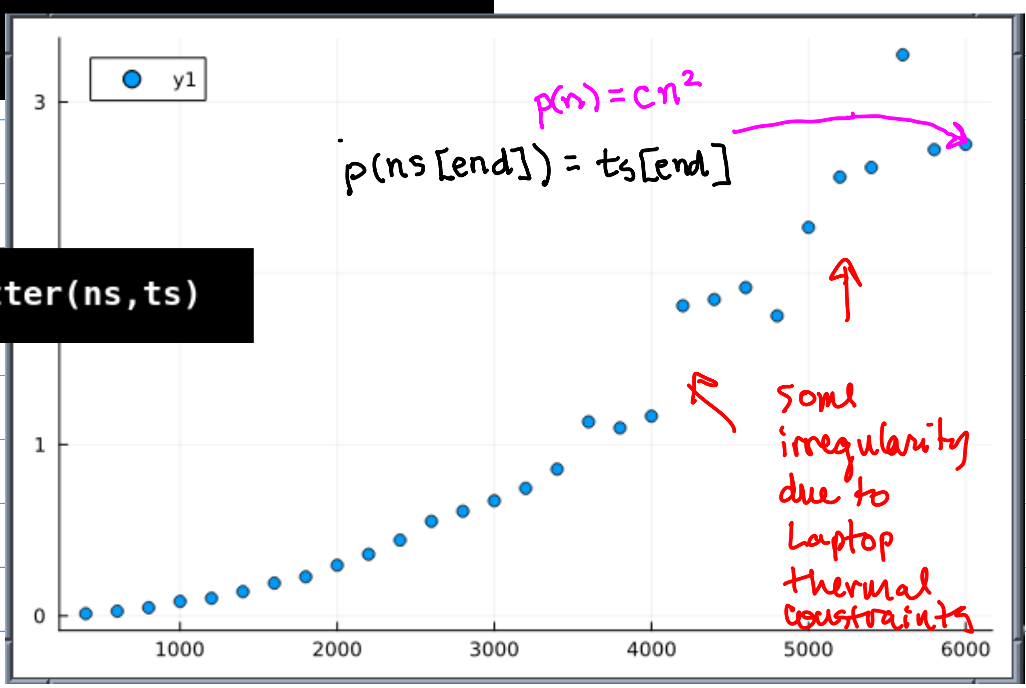
↖ add an element to the end of the list

```
ns = 400:200:6000
ts = []
for n in ns
    A = randn(n,n)
    x = randn(n)
    time = @elapsed for j in 1:80; A*x; end
    push!(ts,time)
end
```

↖ more tests

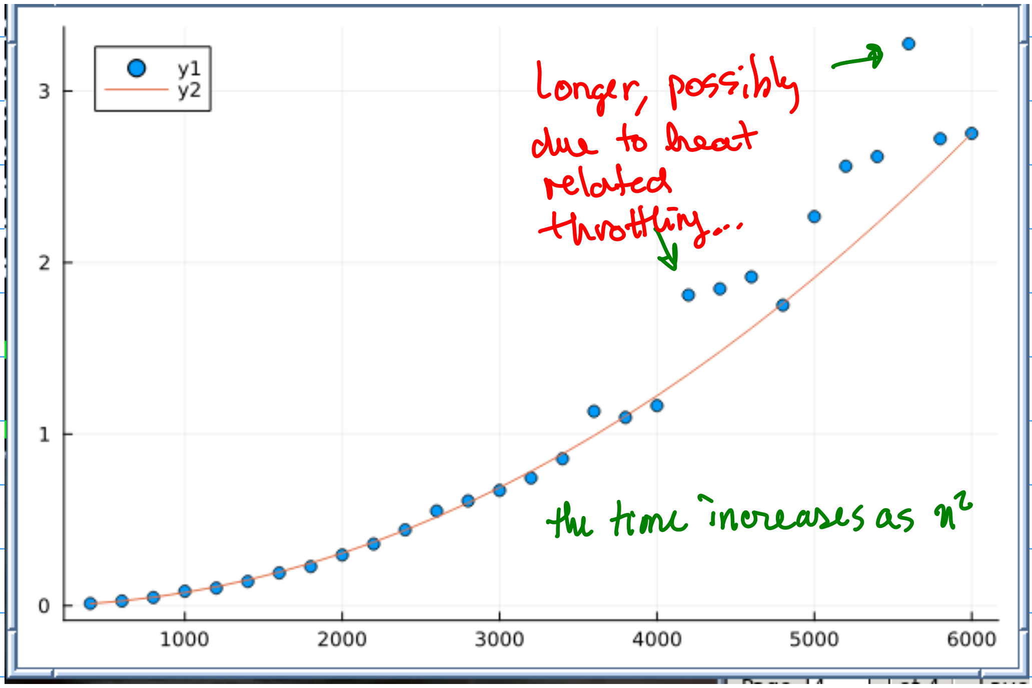
add to the existing list ts

```
scatter(ns,ts)
```



```
plot!(ns,ts[end]*(ns/ns[end]).^2)
```

add to existing plot ...



How many mult as a function of the size n of the matrix?

```
function mylu(A)
        m,n=size(A)
        if m!=n
                println("Need a square matrix!")
                throw(exit())
        end
        U=zeros(size(A))
        L=zeros(size(A))
        Ak=copy(A)
        for k=1:n
                U[k,:]=Ak[k,:]
                L[:,k]=Ak[:,k]/U[k,k]
                Ak=Ak-L[:,k]*U[k,:]'
        end
        return L,U
```

loops n time

no mult here

n divisions

n^2 multiplication

outer product nxn matrix each entry is a mult of ...

total mult + divisions $= n(n+n^2) = O(n^3)$

Check if execution time of mylu(A) scales as O(n^3) next time...