Getting started with Julia and the programming project.

There are two data files: prog1b.dat prog1c.dat

they look like...

```
3
  14   32   53
  32   77  128
  53  128  213

3
  14   32   50
  32   77  122
  50  122  194

4
   1    2    3    4
   2   29   36   43
   3   36  109  126
   4   43  126  246

5
 1.50040  1.06180  1.37343  0.75486  0.78857
 1.06180  1.45784  1.41150  0.87909  1.20866
 1.37343  1.41150  2.16213  0.79598  1.17137
 0.75486  0.87909  0.79598  0.73324  0.74008
 0.78857  1.20866  1.17137  0.74008  1.10941
```

line containing dimension of the matrix

the entries in the matrix

blank line seperating each matrix

next matrix

next matrix

! more matrices not shown

after enough web searching we wrote the code which reads in the matrices

```
function dowork(A)
    n=size(A)[1]
    println("The dimension is $n")
    display(A)
end

open("prog1c.dat","r") do fp
    i=0
    while true
        i+=1
        a=readline(fp)
        if a==""
            break
        end
        d=parse(Int,a)
        A=zeros(d,d)
        for i=1:d
            a=readline(fp)
            r=split(a)
            s=(x->parse(Float64,x)).(r)
            A[i,:]=s
        end
        println("Matrix number $i")
        dowork(A)
        a=readline(fp)
    end
end
```

This routine is called once for each matrix.

put your cholesky factor routine here

read dimension

if no more data then stop

convert string to an integer

make a suitably sized matrix to store the data.

loop to read the matrix

convert every entry to floating point

call the work function to perform cholesky

skip the next line

Eigenvalues and Eigenvectors

Iterative approximation

$$Ax = \lambda x$$

Simple idea

Start with $x_0 \in \mathbb{R}^n$ random...

$y_1 = Ax_0$

$x_1 = y_1 / \|y_1\|$

$\vdots$

$y_{n+1} = Ax_n$

$x_n = y_n / \|y_n\|$

Claim
x_n "converge" to eigenvector with λ such that $|\lambda|$ is the largest...

Test the claim

```
julia> A=rand(5,5)          ← random matrix
5×5 Matrix{Float64}:
 0.245698  0.765098  0.942794  0.829685  0.234423
 0.596459  0.471983  0.214947  0.58746   0.279085
 0.270679  0.484195  0.568402  0.502487  0.614909
 0.854123  0.792205  0.160561  0.453048  0.526261
 0.18797   0.822666  0.487708  0.89924   0.96734

julia> x0=rand(5)           ← random vector
5-element Vector{Float64}:
 0.7778515649008082
 0.7694515581781274
 0.5675574884628809
 0.19093313680067303
 0.7330333278830921
```

```
julia> y1=A*x0
5-element Vector{Float64}:
 1.645165945401342
 1.265864117442048
 1.452402978955604
 1.8373407295106854
 1.9368045690759499

julia> x1=y1/norm(y1)
5-element Vector{Float64}:
 0.4470049649740356
 0.3439455740381444
 0.3946296995455089
 0.49922041647898
 0.5262455504755434
```

← start iterating

Make a loop to do it faster

```
julia> x=x1
       for j=1:10
           y=A*x
           x=y/norm(y)
       end

julia> x
5-element Vector{Float64}:
 0.46149788878206716
 0.3462348904295094
 0.39836036541471737
 0.4505457561826266
 0.552683128158838
```

← when entering this block of code press alt+return to put everything together.

after 10 more iterations...

↖ is that an eigenvector?

Check is $Ax = \lambda x$ by dividing each element of Ax by x.

```
julia> A*x ./ x
5-element Vector{Float64}:
 2.7242530972648136
 2.724253047445331
 2.7242530540387158
 2.724253015294663
 2.7242530748676694
```

← these numbers are all pretty much the same and equal to λ.

Press ↑ up arrow (twice) and then ← arrow to edit the loop...

```
julia> x=x1
       for j=1:100
           y=A*x
           x=y/norm(y)
       end
```

move the cursor here and change 10 to 100

Then press return to run it again...

```
julia> x=x1
       for j=1:100
           y=A*x
           x=y/norm(y)
       end

julia> x
5-element Vector{Float64}:
 0.4614978938229284
 0.34623488901764343
 0.39836036456519014
 0.45054574996856933
 0.5526831305121312

julia> A*x ./ x
5-element Vector{Float64}:
 2.724253056139499
 2.724253056139499
 2.724253056139499
 2.7242530561394984
 2.724253056139499
```

eigenvector

note, this would work if any of the entries in x were by chance equal to zero

all the same means very good approximation

common number, is the eigenvalue

Next week we discuss ways to find other eigenvalues and ways to make convergence faster.