In science and engineering an important goal is to become a skilled practitioner by doing it yourself. The computer labs provide computational experience related to the analytic theory presented in the lectures.

Matrix Norms

This lab is about computing the matrix norm of $A \in \mathbf{R}^{m\times n}$ given by

$$\|A\| = \max\big\{\, \|Ax\| : \|x\| \le 1 \,\big\}.$$

Note the matrix norm is defined in terms of vector norms. Thus, to every kind of vector norm corresponds a different matrix norm.

The vector norm

$$\|x\|_\infty = \max\big\{\, |x_i| : i = 1, \ldots, n \,\big\}$$

was considered in the text. In that case, the corresponding matrix norm

$$\|A\|_\infty = \max\big\{\, \|Ax\|_\infty : \|x\|_\infty \le 1 \,\big\}$$

may be computed as

$$\|A\|_\infty = \max\Big\{\, \sum_{j=1}^{n} |a_{ij}| : i = 1, \ldots, m \,\Big\}.$$

In this lab we consider the Euclidean norm $\|x\|_2 = \sqrt{x \cdot x}$ and recall the formula for computing the corresponding matrix norm

$$\|A\|_2 = \max\big\{\, \lambda_i^{1/2} : i = 1, \ldots, n \,\big\}$$

where λ_i are the eigenvalues of $B = A^T A$. Upon defining the spectrum

$$\sigma(B) = \big\{\, \lambda : \lambda \text{ is an eigenvalue of } B \,\big\}.$$

we obtain

$$\|A\|_2 = \max\big\{\, \lambda^{1/2} : \lambda \in \sigma(A^T A) \,\big\}.$$

This is why $\|A\|_2$ is called the spectral norm. The goal in this lab is to find the spectral norm of an individualized 5×4 matrix.

Each person in the lab will have a different matrix. Your individualized matrix may be obtained by clicking on the following link:

`https://fractal.math.unr.edu/~ejolson/466-23/specnorm/snmatrix.cgi`

Please do not use anyone else's matrix for this lab.

Upon clicking on the link, I obtained

Your matrix in Julia-compatible code is

```
A=[  0.23  0.29 -1.85 -3.48;
     4.24  4.00  2.36  4.82;
     4.56 -2.97  1.90 -0.51;
    -1.18 -2.55 -4.86  0.50;
     3.31 -2.04 -2.58  0.35 ]
```

The following discussion concerns the matrix which appeared when I clicked the above link. That matrix is different than what you will obtain when you click the same link. To finish this lab please repeat these same steps but for your own individualized matrix.

The Statistical Approach

Consider again the definition

$$\|A\|_2 = \max\big\{\, \|Ax\|_2 : \|x\|_2 \le 1 \,\big\}.$$

For notational convenience the subscripts on the norms will be dropped throughout the rest of this lab so that $\|A\| = \|A\|_2$, $\|Ax\| = \|Ax\|_2$ and $\|x\| = \|x\|_2$. An intuitive way to approximate $\|A\|$ would be to randomly select vectors x such that $\|x\| \le 1$, plug them in to $\|Ax\|$ and then choose the largest resulting value for $\|A\|$.

Given a non-zero vector $x \in \mathbf{R}^n$ with $\|x\| < 1$ define the corresponding unit vector $z = x/\|x\|$. Since

$$\|Az\| = \left\| A\frac{x}{\|x\|} \right\| = \frac{1}{\|x\|}\|Ax\| \ge \|Ax\|,$$

one immediately has

$$\|A\| = \max\big\{\, \|Az\| : \|z\| = 1 \,\big\}.$$

Thus, it is sufficient to randomly test unit vectors to approximate $\|A\|$.

Before writing the code, reflect for a moment that this method of finding $\|A\|$ is likely to be inefficient and of low accuracy. Such is typical for calculations that follow directly from the definition. The advantage is such a calculation is less likely to have mathematical errors or software bugs. Moreover, having two different ways to solve the same problem serves as a check of correctness for the algorithms we implement later.

Since computers—even ChatGPT—will happily produce answers that are completely wrong without warning or feelings of guilt, it is important to work in such a way that provides consistency checks on the final answer. The point here is that one wouldn't be using a computer in the first place if the answer were already known.

Julia has a built-in function called `rand(`n`)` that makes random vectors. This function samples a random vector $V \in \mathbf{R}^n$ where each V_i is uniformly distributed in $[0, 1)$. Thus, `rand(2)` always produces a vector in the first quadrant. To produce random vectors in all directions, apply the transform $X_i = 2V_i - 1$ so X_i is uniformly distributed in $[-1, 1)$.

For example,

```
julia> x1=2*rand(4).-1
4-element Vector{Float64}:
  0.5769703226246823
 -0.7694043276720857
  0.04742602668985452
  0.8611118814209817

julia> x2=2*rand(4).-1
4-element Vector{Float64}:
 -0.9776982954790019
  0.1938427905675142
 -0.03880922639686535
  0.2911117285223299
```

produces two random samples in $\mathbf{R}^4$. Note that since these are random variables you will get different vectors each time they are sampled.

Load your individualized matrix A into Julia and then compute $\|Az\|$ where $z = x/\|x\|$ is the unit vector corresponding to $x =$ `x1` and $x =$ `x2`. For the matrix downloaded earlier it follows that

```
julia> A=[  0.23  0.29 -1.85 -3.48;
            4.24  4.00  2.36  4.82;
            4.56 -2.97  1.90 -0.51;
           -1.18 -2.55 -4.86  0.50;
            3.31 -2.04 -2.58  0.35 ]
5×4 Matrix{Float64}:
  0.23   0.29  -1.85  -3.48
  4.24   4.0    2.36   4.82
  4.56  -2.97   1.9   -0.51
 -1.18  -2.55  -4.86   0.5
  3.31  -2.04  -2.58   0.35

julia> using LinearAlgebra

julia> norm(A*x1/norm(x1))
5.981497801013638

julia> norm(A*x2/norm(x2))
6.516247741068198
```

Taking the largest of the two implies

$$\|A\| \geq 6.516247741068198.$$

Now check a million different directions for a sharper bound. First, create the function `sample` to make generating samples easier.

```
julia> function sample()
           while true
               x=2*rand(4).-1
               r=norm(x)
               if r>0
                   return x/r
               end
           end
       end
```

```
sample (generic function with 1 method)
```

The condition $r > 0$ avoids the chance that the zero vector was chosen at random. Though zero is unlikely, checking could be important when creating a very large number of samples.

To make sure everything is fine, test the function interactively.

```
julia> sample()
4-element Vector{Float64}:
  0.37505747491848035
 -0.23812495410387083
  0.5682134947095993
 -0.6926484109349471

julia> sample()
4-element Vector{Float64}:
 -0.2890023067322923
  0.7717879209913837
  0.15436044914663802
 -0.5449714886619847
```

Now run a loop to compute the sharper bound

```
julia> Abound=0.0
       for i=1:1000000
           z=sample()
           t=norm(A*z)
           if Abound<t
               Abound=t
           end
       end

julia> Abound
9.204731194747062
```

This implies

$$\|A\| \geq 9.204731194747062.$$

Your individualized matrix A may have a much different bound. On the other hand, even for the same matrix the bound will be slightly different each time due to the randomness.

Before finishing this section note that Julia allows the same computation to be performed by passing an iterator to a function as

```
julia> maximum(norm(A*sample()) for i=1:1000000)
9.20451625282672
```

This functional approach looks simpler and is sometimes clearer to reason about, but there is no speed advantage either way. Often I find loops easier because they express the algorithm in explicit steps in much the same way as the hardware actually works.

The Spectral Approach

In this section we set $B = A^T A$ and then use the power method to find the largest eigenvalue λ_1 of B. It follows that $\|A\| = \lambda_1^{1/2}$.

We shall use the scaled power method which renormalizes the approximation $w^{(j)}$ of the eigenvector at each iteration to prevent numerical overflow and underflow. Recall our individualized matrix $A \in \mathbf{R}^{5\times 4}$. Consequently $B \in \mathbf{R}^{4\times 4}$ and so $n = 4$.

Let $w^{(0)} \in \mathbf{R}^4$ be an initial approximation of an eigenvector corresponding to λ_1. Almost any non-zero vector will work. Set $y^{(0)} = w^{(0)}$ and repeatedly carry out the following steps for $j = 1, 2, 3, \ldots$ until the approximation in step (c) converges.

(a) Calculate $w^j = By^{(j-1)}$.

(b) Find p such that $|w_p^{(j)}| = \max\left\{\, |w_i^{(j)}| : i = 1, \ldots, n \,\right\}$.

(c) Evaluate the approximation $\lambda_1^{(j)} \approx \dfrac{w^{(j)} \cdot y^{(j-1)}}{y^{(j-1)} \cdot y^{(j-1)}}$.

(d) Calculate $y^{(j)} = w^{(j)}/w_p^{(j)}$.

The following Julia code chooses $w^{(0)}$ randomly, iterates 30 times and prints the value of $\lambda_1^{(j)}$ at each iteration.

```
julia> B=A'*A
```

```
        w=sample()
        y=copy(w)
        for j=1:30
            w=B*y
            p=argmax(abs.(w))
            lambda1=(w'*y)/(y'*y)
            println("lambda1^($j)=$lambda1")
            y=w/w[p]
        end
lambda1^(1)=69.47112082263426
lambda1^(2)=82.17910796253742
lambda1^(3)=84.19541584480642
lambda1^(4)=84.59223680274377
lambda1^(5)=84.68990112869551
...omitted...
lambda1^(27)=84.72747261391937
lambda1^(28)=84.72747261391945
lambda1^(29)=84.72747261391945
lambda1^(30)=84.72747261391943
```

Taking square roots as

```
julia> sqrt(84.72747261391943)
9.204752718781718
```

implies that $\|A\| \approx 9.204752718781718$. It is satisfying to observe this approximation is similar to the bound obtained with the statistical approach.

The Built-in Matrix Norm

Julia has a built-in function called `opnorm` which uses the spectral approach to find $\|A\|$. The main improvement is an algorithm more complicated and efficient than the power method is used to find the largest eigenvalue of B.

The result for the matrix downloaded earlier is

```
julia> opnorm(A)
9.204752718781718
```

Not surprisingly the result is comparable to what was obtained earlier.

At this point you may be wondering why one went to so much trouble to approximate $\|A\|$ in the previous sections when a built-in function that does the same thing already exists. My understanding is that doing it yourself builds intuition about how various algorithms works in practice along with the computational skills needed to tackle problems that haven't been solved before. At the same time, if it becomes necessary to find a matrix norm in a practical application the built-in function would be preferred.

Submitting Your Work

A single PDF file should be submitted for grading that contains

- A program that approximates $\|A\|_2$ for your individualized matrix in three ways: Using the statistical approach with a million samples, using the spectral approach with 30 iterations and finally using `opnorm`.
- The output from running that program.

After debugging and making sure your program runs correctly, you may prepare your submission by typing

```
$ julia matnorm.jl >matnorm.out
$ j2pdf -o submit04.pdf matnorm.jl matnorm.out
```

Before uploading, check `submit04.pdf` using the PDF previewer with

```
$ evince submit04.pdf &
```

to make sure the output from the three computations is consistent. Please reboot into Microsoft Windows before leaving the lab.