VISUALIZING A MATRIX

STUART D. BRORSON *

Abstract. I present several different ways to visualize a matrix by considering its action on vectors – first as a linear transform, then as a quadratic form. My experience with teaching students shows that these geometric visualizations build intuition about matrix transformations and also provide powerful ways to think about constructions like the matrix norm, the EVD, the SVD, and the matrix condition number. Although the mathematics are well known, gathering these visualizations into one place along with some applications helps students understand and appreciate the utility of matrices seen as geometric objects.

Key words. Matrix analysis, linear algebra, numerical linear algebra.

1. Introduction. Everybody learns about matrices early in their mathematical educations. Unfortunately, for many people a matrix remains nothing more than a table of numbers, perhaps along with some rules about how to add or multiply one matrix with another. This is unfortunate since a matrix is so much more than a bunch of numbers! It is possible – even desirable – to think about the geometric properties of a matrix. The goal of this article is to present several different geometric interpretations of a matrix. I believe thinking about a matrix in geometric terms builds intuition which becomes useful when using matrices in solving real-world problems.

Alongside learning about matrices, everybody also learns about vectors – perhaps when still in high school. For vectors the educational situation is better – just about everybody learns to think geometrically about vectors from the time they are introduced. We typically notate a vector as a row or column of numbers, like this

$$u = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \end{pmatrix} \tag{1.1}$$

Now in order to write a numeric representation for a vector like (1.1) we need to assign unit vectors to particular directions in space. However, we are simultaneously taught that a vector can be visualized as an arrow in space. The arrow has a length and points in a specific direction. The arrow has an independent existence outside of any particular coordinate system imposed on the space. Therefore, the actual numbers in (1.1) may change depending upon our choice of coordinate system, but the direction and length of the arrow remain the same.

Examples of two and three dimensional vectors are shown in Figure 1 and Figure 2. The length of a vector is commonly called its norm. The norm of a vector is computed in the 3-dimensional case as

$$\|u\|_2 = \sqrt{u_1^2 + u_2^2 + u_3^2} = \sqrt{\sum_{i=1}^{3} u_i^2} \tag{1.2}$$

More exactly, this is the 2-norm or Euclidian norm of a 3-dimensional vector. The 2-norm is the most frequently used vector norm in engineering and physics applications. Generalizing the definition of norm to vectors of higher dimension is an obvious extension of (1.2).

*Northeastern University, Boston, MA (s.brorson@northeastern.edu)

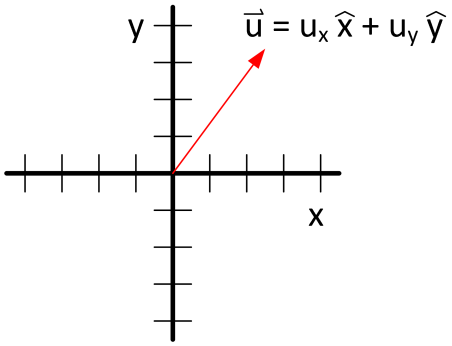

FIG. 1. *A 2-dimensional vector represented as an arrow in the plane.*

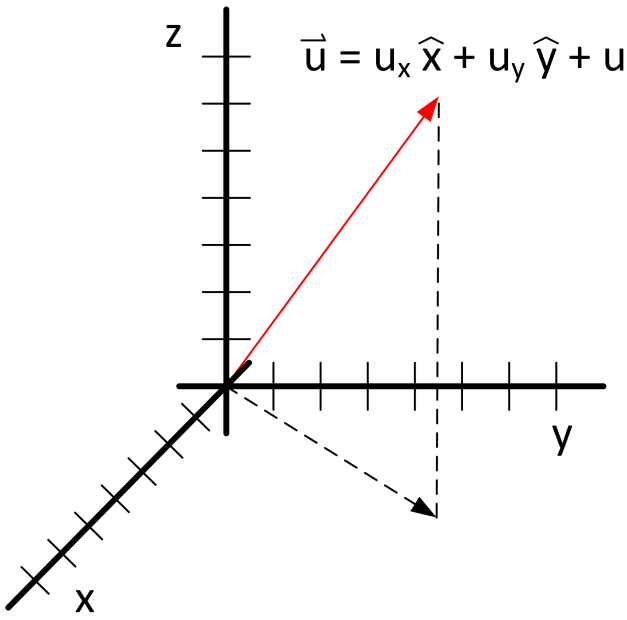

FIG. 2. *A 3-dimensional vector represented as an arrow in space. (The dotted arrow is the projection of the 3-vector onto the x-y plane.)*

Visualizing a vector as an arrow in space serves well in many disciplines, especially in engineering and physics where a vector is used to capture concepts like position in real space, particle velocity, the force impressed on a charge by an electric field, and so on. It gives us a way to think about, for example, the direction of a force as well as its magnitude. However, it can also be applied to more abstract spaces, like a set of measured points $[\xi_1, \xi_2, \xi_3, ...\xi_N]$ taken from an experiment. That is, a set of N measured points can be thought of as an arrow in some N-dimensional space, where each measurement ξ_i corresponds to the amount of that arrow which points in direction i.

2. The matrix as linear transform. Visualizing a vector is easy – the "arrow in space" image is clear and intuitive. But how to visualize a matrix? Is there some geometric picture which we can carry in our heads which captures important properties of a matrix? The answer is "yes" – that's the subject of this article. The starting point to visualizing a matrix is to consider the action of matrix-vector multiplication. For most of this article we will work in 2 dimensions (2x2 matrix), and consider only square matrices. That's so we can draw the geometric actions on a two-dimensional page. In a few examples we will look at a 3x3 matrix in 3-dimensional space. We will also usually use the name A for our matrix, and u and v as the names of vectors. Importantly, we will assume we are looking at a "nice" matrix in a mathematical sense. That is, we will assume the matrix is non-singular, has full rank, and so on. However, the visualizations presented here also help understanding singular matrices, so later in this article we will look at some "bad" matrices. Finally, extending the geometric concepts presented here to four and higher dimensions is straightforward – even if visualizing an object in four dimensions is difficult for ordinary human beings!

Consider multiplying a 2 element vector u by an arbitrary 2x2 matrix A,

$$v = Au$$

The result is another 2 element vector v. Consider this operation as an input-output relationship: the input is the vector u and output is the vector v. Plot the input vector in the 2D plane in red and the output vector in the same plane in blue. The result is shown in Figure 3. Multiplication by A has taken the input vector u and transformed it into the output vector v. This is the first

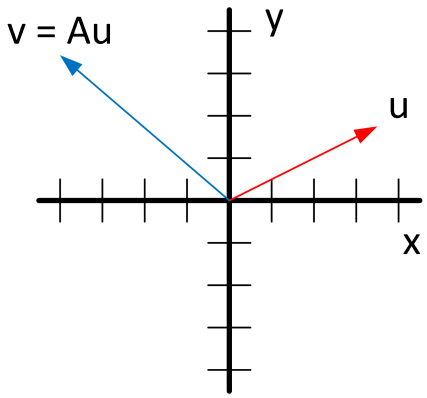

FIG. 3. *Matrix vector multiplication. The input to this visualization is the vector u (red) and the output is the transformed vector v (blue).*

way to think about a matrix: matrix-vector multiplication is an operation which takes a vector and transforms it into a different vector.

2.1. Abstract linear transform. An abstract way to think about this is to replace our 2-dimensional matrix-vector multiplication by the abstract operator $T : \mathbb{R}^2 \to \mathbb{R}^2$ which takes an input 2-vector u and returns an output 2-vector v,

$$v = T(u)$$

It can be shown that the operator T is equivalent to matrix-vector multiplication as long as the following holds true:

- $T(au) = aT(u)$, where a is a scalar
- $T(u + v) = T(u) + T(v)$, where u and v are vectors of the same dimension.

These rules are the definition of a linear transform. Accordingly, matrix-vector multiplication $v = Au$ may be viewed as implementing a linear transform operation which takes u to v. Traditional linear algebra textbooks make a big deal out of this, but for our purposes the important concept is that we can treat matrix-vector multiplication as implementing an operation which behaves like a function, $v = T(u)$. Thought of this way, the objects of our study are the properties of matrix A itself, specifically when A is used as a linear transform $T()$ taking input 2-vectors to output 2-vectors.

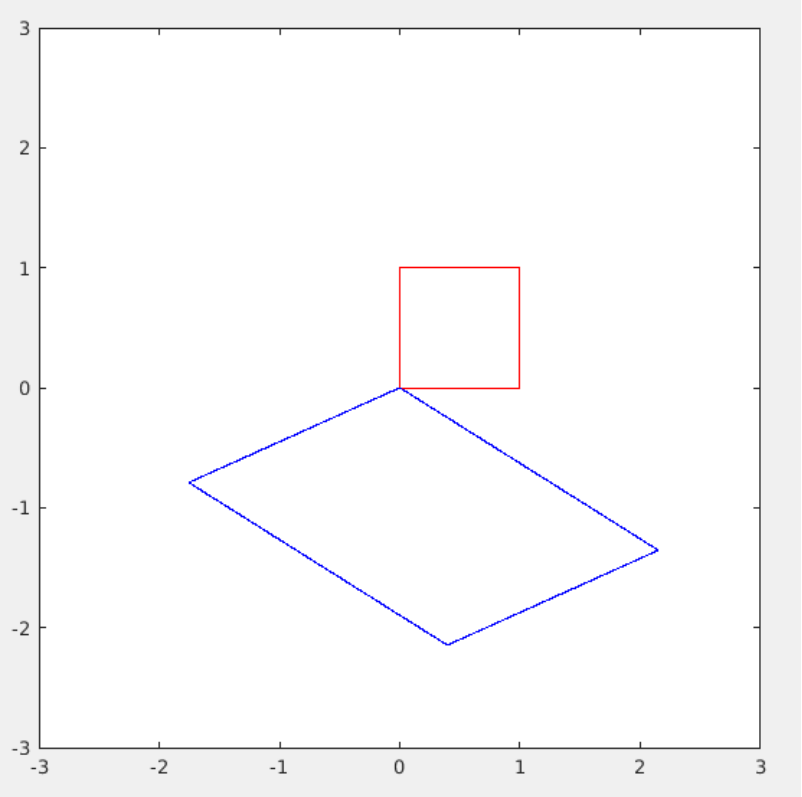

FIG. 4. *The action of matrix-vector multiplication on a unit square. The original unit square is red, the transformed square is blue.*

2.2. Transforming a unit square. The linear transform picture Figure 3 is frequently featured in linear algebra textbooks, so we won't dwell on it here. But a useful extension of this picture involves considering what happens when you draw a square in the x-y plane, and operate on all its points with the matrix A. This is shown in Figure 4, where we create a unit square with one corner lying at the origin. We call the input points u. Upon multiplication with the matrix A the input points u are mapped to the output points v, which form a parallelogram. One vertex of the output parallelogram lies at the origin – multiplying the zero point by A returns zero, so this point

doesn't move. However, all other points in the input square are mapped to some new point on the output shape. What's noteworthy is that the set of output points v always form a parallelogram (as opposed to forming some other shape).

Matlab code to perform this operation is presented in Appendix A.1. If you run this code repeatedly it will generate a new random matrix with each run, and you will get a different output parallelogram each time. This is shown in Figure 5. Examining the results produced by several

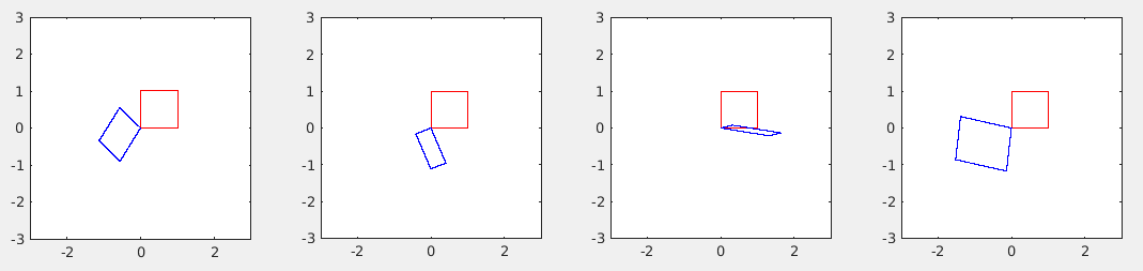

FIG. 5. *Four different parallelograms generated from four different random matrices.*

different matrices shows that the resulting square is both stretched and rotated by the action of A. This is an important observation which we will return to in subsequent sections.

Now let's ask an interesting question: what is the area of the parallelogram generated by $v = Au$? It turns out the area D is given by the determinant, $D = \det(A)$. In our specific 2-dimensional case, if matrix A is

$$A = \begin{pmatrix} \alpha & \delta \\ \beta & \gamma \end{pmatrix}$$

then we have

$$D = \det(A) = \alpha\gamma - \beta\delta \tag{2.1}$$

A "proof without words" of (2.1) is shown in Figure 6. There are a couple of interesting observations to make about (2.1):

- If A is singular, then $\det(A) = 0$. In this case, the parallelogram reduces to a line segment with zero area (or possibly even just a point at the origin if A is the zero matrix). Therefore, even with this simple visualization, the concept of a singular matrix has a geometric interpretation as a degenerate form of the parallelogram.
- For some matrices, $\det(A)$ is negative which implies the parallelogram area can be negative. A negative area can seem unintuitive at first, but if you think of the order in which the two sides are visited you see that one may visit them in two different orders: first right then left, or first left then right. This is similar to computing the "right hand rule" associated with the vector cross product. That is, the area reported by (2.1) is a "signed area" which can take on either sign depending upon the details of how matrix A is constructed.

2.3. Application: Rotation and stretching matrices. The transformation shown in Figure 4 evidences two important operations performed by the transformation: rotation and stretching. That is, the transformation Au has taken the input square, and stretched it and rotated it to create the output square v. Rotation is a "rigid motion" of an object – one which moves the object but

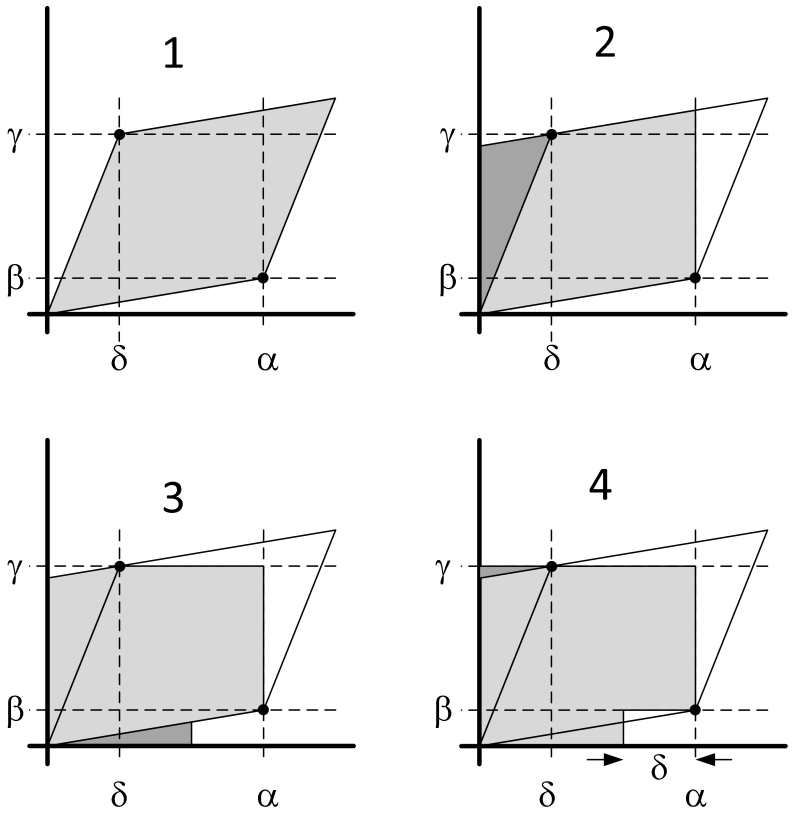

FIG. 6. *Proof that the area of the parallelogram created by $v = Au$ is $\det(A) = \alpha\gamma - \beta\delta$. The proof involves performing rigid motions of parts of the parallelogram so the object is rearranged into a form where Area $= \alpha\gamma - \beta\delta$ may be deduced by inspection.*

doesn't change its shape. Stretching changes the object's shape. (A third effect of matrix multiplication is "inversion", which in two dimensions is achieved by flipping the object over. We'll ignore inversion for now.)

In two dimensions, the rotation transformation may be captured using a so-called rotation matrix, frequently written as

$$R(\theta) = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \tag{2.2}$$

Multiplying a vector by this rotation matrix rotates it about the origin by the angle θ. An example rotation of the square is shown in Figure 7. There are many noteworthy features of rotation matrices. Here are three:

1. Since rotation is a "rigid motion", it preserves the areas of objects being rotated. This is evident upon inspection of the input and output squares shown in Figure 7 – the areas of the two squares are the same. This implies that the determinant of the rotation matrix

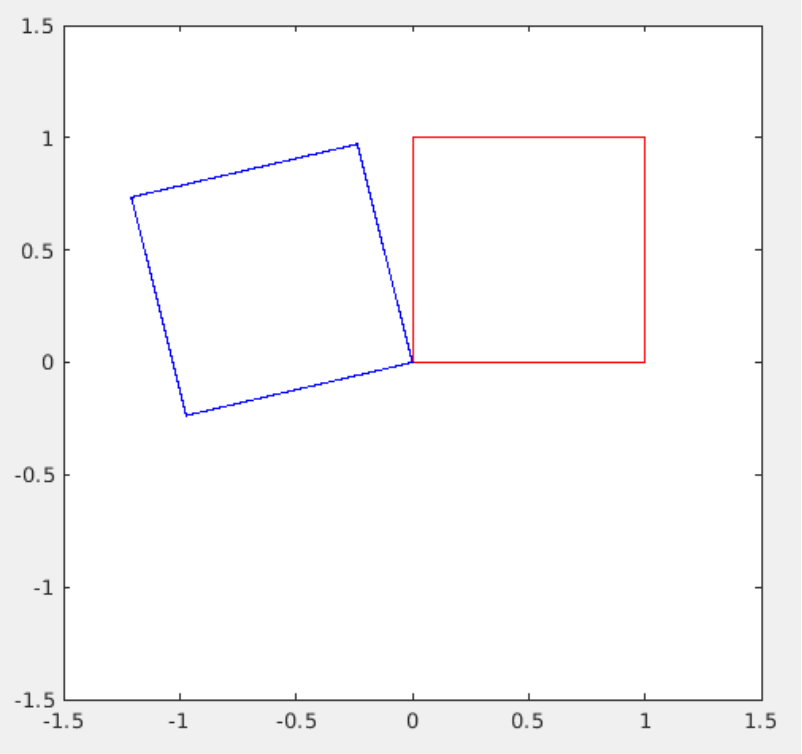

FIG. 7. *Multiplication of the input red square by the rotation matrix $R(\theta)$ takes each point and rotates it by angle θ to create the output blue square.*

must be one, which is easily verified:

$$\det(R) = \det \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} = \cos^2\theta - (-\sin^2\theta) = 1$$

Since the determinant is one, the rotation matrix will preserve the area of any object it operates on. This is a satisfying property since intuitively we expect rotation to preserve area.

2. Cascading two rotations by θ_1 (first) and θ_2 (second) is equivalent to multiplying the two corresponding rotation matrices together. This can be shown by evaluating the matrix multiplication and using trig identities to simplify the result.

$$\begin{aligned} R(\theta_2)R(\theta_1) &= \begin{pmatrix} \cos\theta_2 & \sin\theta_2 \\ -\sin\theta_2 & \cos\theta_2 \end{pmatrix} \begin{pmatrix} \cos\theta_1 & \sin\theta_1 \\ -\sin\theta_1 & \cos\theta_1 \end{pmatrix} \\ &= \begin{pmatrix} \cos\theta_2\cos\theta_1 - \sin\theta_2\sin\theta_1 & \cos\theta_2\sin\theta_1 + \sin\theta_2\cos\theta_1 \\ -\sin\theta_2\cos\theta_1 - \cos\theta_2\sin\theta_1 & -\sin\theta_2\sin\theta_1 + \cos\theta_2\cos\theta_1 \end{pmatrix} \\ &= \begin{pmatrix} \cos(\theta_2+\theta_1) & \sin(\theta_2+\theta_1) \\ -\sin(\theta_2+\theta_1) & \cos(\theta_2+\theta_1) \end{pmatrix} \end{aligned}$$

3. The rotation matrix is a subset of the general set of "orthogonal matrices" [3]. These are matrices made by concatenating orthogonal column vectors like this:

$$\begin{pmatrix} \vdots & \vdots & \vdots \\ q_1 & q_2 & q_3 \\ \vdots & \vdots & \vdots \end{pmatrix}$$

Orthogonal matrices are usually notated using the name Q for the matrix. It is easy to show that orthogonal matrices obey the relationship

$$Q^T Q = Q Q^T = I \tag{2.3}$$

where I is the identity matrix. The proof is based on looking at the dot products of the constituent column vectors – try doing it yourself! Another way of expressing (2.3) is to recognize that the transpose of an orthogonal matrix is its inverse,

$$Q^T = Q^{-1}$$

These facts become useful when studying matrix decompositions such as the SVD, the EVD, and the QR decomposition, which yield an orthogonal matrix as one of the factors.

Another possible action produced by the transformation Au is stretching. For simplicity we specialize to the case of a diagonal matrix,

$$D = \begin{pmatrix} d_1 & 0 \\ 0 & d_2 \end{pmatrix} \tag{2.4}$$

The action of D is to stretch the x axis by factor d_1 and the y axis by factor d_2. This is shown in Figure 8 below, which depicts the action of the matrix $D = \left(\begin{smallmatrix} 1.5 & 0 \\ 0 & 0.8 \end{smallmatrix}\right)$ on the unit square. By inspection it is evident the square grows in the x direction by 1.5 and shrinks in the y direction by 0.8; the output is a rectangle.

We will examine the case of stretching in arbitrary directions in section subsection 3.3 below, which describes the singular value decomposition.

3. Transforming a unit circle. In the last section we examined the action of a 2x2 matrix on the unit square. Rather than visualizing the matrix directly, we looked at its effect on a square in the x-y plane. In this section we look at the effect of a 2x2 matrix on a unit circle (unit ball) in the x-y plane.

Consider how to plot the points on a unit circle. The most common definition of the unit circle is that it is the locus of all points in the x-y plane satisfying

$$x^2 + y^2 = 1 \tag{3.1}$$

One can imagine every point on the circle is represented as the tip of a unit length vector. If you take the collection of unit vectors based at the origin and pointing in all different directions, the set of all vector tips define a circle. Therefore, a different way to create a plot of the unit circle is use a parametric representation. Consider all vectors of form

$$u = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix} \tag{3.2}$$

where $\theta \in [0, 2\pi]$.

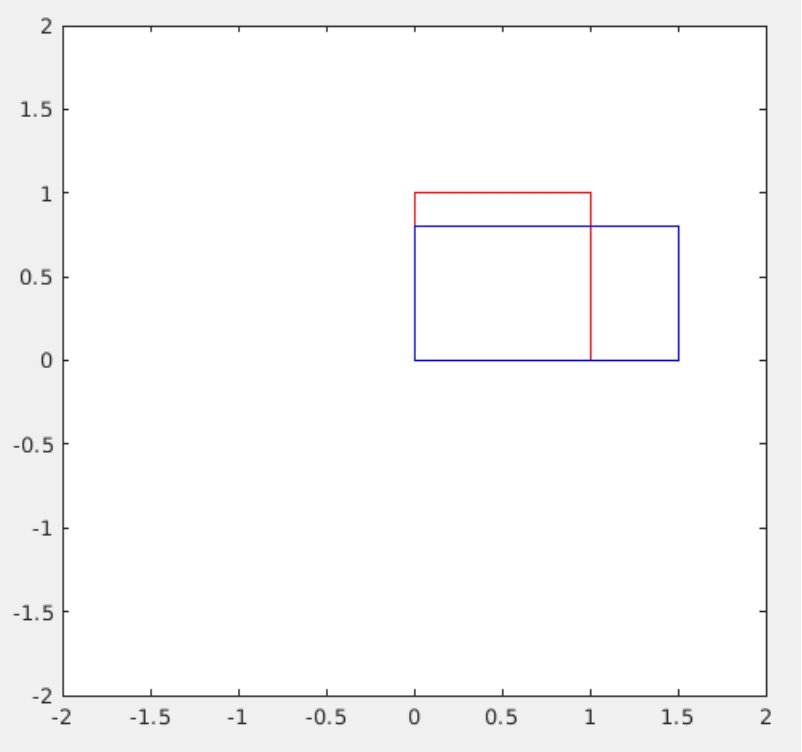

FIG. 8. *Multiplication by diagonal matrix* $D = \left(\begin{smallmatrix} 1.5 & 0 \\ 0 & 0.8 \end{smallmatrix}\right)$ *stretches the input square (red) in the* x *and* y *directions. The result is the output rectangle (blue).*

Each point $u = (x(\theta), y(\theta))^T$ satisfies (3.1), so each of these points lies on the unit circle. Therefore, this representation provides a simple algorithm to create a unit circle in the plane: just sweep θ from 0 to 2π and plot the points $(x(\theta), y(\theta))^T$.

Now we will use the set of unit vectors created by (3.2) to create a visualization for a matrix. This visualization considers how the circle of unit vectors is itself transformed by matrix-vector multiplication. The idea is to take every unit vector u and multiply by the matrix A, giving a new point v. Then plot the set of all points v generated by $v = Au$. A Matlab program which implements this idea is given in Appendix A.2. The circle and its transformed image are shown in Figure 9. The important thing to note in Figure 9 is that the circle has been transformed into an ellipse. The shape, size, and rotation of the ellipse are given by the properties of the matrix A. Different matrices will create different ellipses having different shapes, sizes, and rotations. This is illustrated by running the program in Appendix A.2 several times for different, randomly generated A matrices. The results obtained by operating on a circle using four different random matrices are shown in Figure 10

It may appear this matrix visualization involving transforming unit circles into ellipses is similar to the visualization involving the unit square and is therefore redundant. However, that is not true – several very deep properties of a matrix are better understood by investigating the ellipse visualization. Investigating these properties is the task of the following subsections.

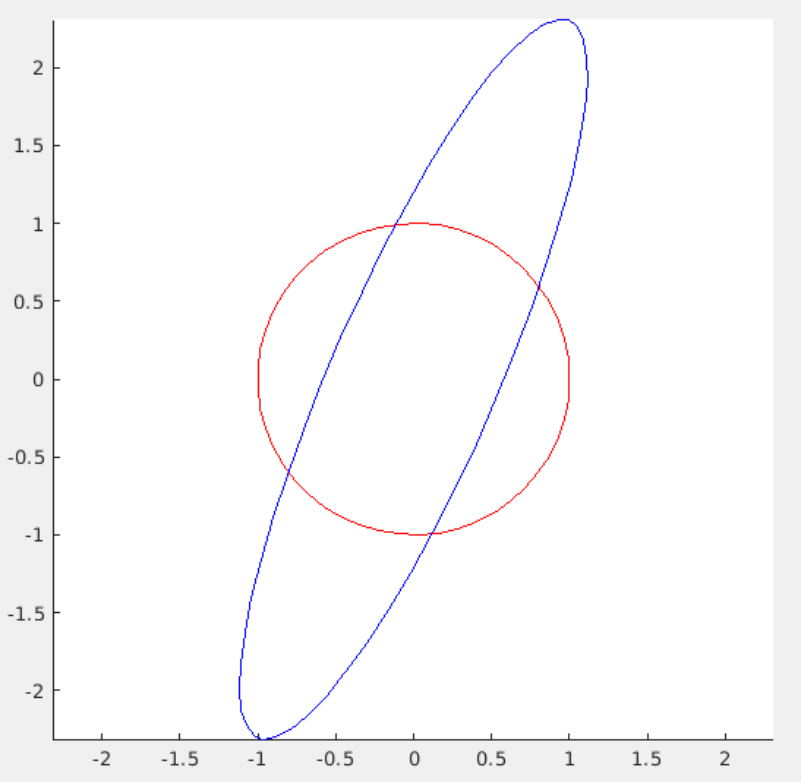

FIG. 9. *Multiplication by matrix A sends every point on the unit circle to a point on an ellipse.*

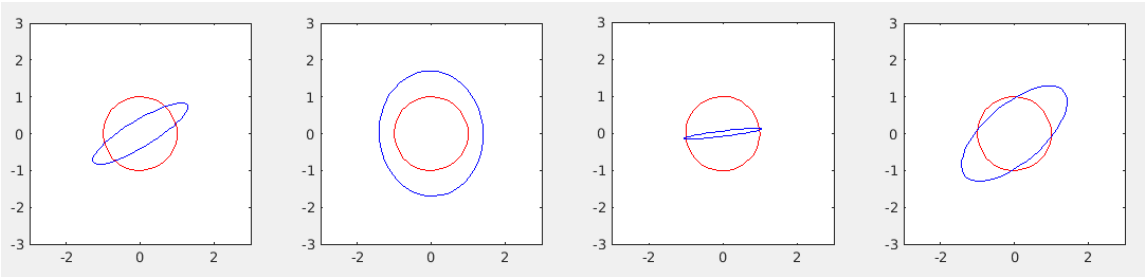

FIG. 10. *Repeated runs of the unit circle program produce different random matrices A which product different output ellipses. The original unit circle is red, the output ellipses are blue.*

3.1. Matrix norm. In general, the norm of any mathematical object is an operation which takes the object and maps it to a positive real scalar. The size of the scalar tries to measure something related to the magnitude of the object – the norm gives a measure how "big" the object is in some sense. For example, the familiar vector norm measures the length of the "arrow in space". The general goal of defining a norm on some object is to capture the idea of "bigness" in a way

which is useful in calculation.

When defining the norm of a new mathematical object, one must take care to satisfy a set of rules about the behavior of a norm. The purpose of the rules is to ensure that the so-defined norm not only measures the "bigness" of the object under consideration, but is also sane. Also, the rules make it possible to compare the "bigness" of two different objects using their norms in a meaningful way.

Here are the rules for creating a norm. They are abstract rules, meaning they hold when computing the norm of any object, whether a vector, a matrix, or some other mathematical construction. If M and N are two of the objects under consideration, their norms must satisfy

- $\|M\| \geq 0$
- $\|aM\| = |a| \, \|M\|$
- $\|M + N\| \leq \|M\| + \|N\|$
- if M is a zero object, then $\|M\| = 0$.

We note that these rules leave a lot of flexibility for defining a norm. This flexibility manifests itself when considering computing the norm of a vector – there are many different ways to compute a vector norm, including:

- L_1 norm: $\|x\| = \sum_i |x_i|$
- L_2 (Euclidian) norm: $\|x\| = \sqrt{\sum_i x_i^2}$
- L_∞ norm: $\|x\| = \max_i |x_i|$

Although having multiple ways to compute a norm might seem redundant, it turns out that different vector norm definitions find application in different places, depending upon the needs of the application and the properties of the particular norm.

What about the norm of a matrix? Similar to vector norm, there are several ways to define a matrix norm [1]. One particularly useful definition generates the so-called "induced norm", which measures how much the matrix A can stretch a vector. Since this matrix norm is based on computing vector norms, one can think of this definition as "induced" by the definition of the vector norm, hence the name.

The induced matrix norm is found as follows. Consider an input vector u with norm $\|u\|$. We use the Euclidian (L_2) norm for the vector norm. Now consider operating on u with A via matrix-vector multiplication, $v = Au$. The output vector v will also have a norm, $\|v\|$. Now ask, what is the relationship of $\|u\|$ to $\|v\|$? Performing this computation is easy. The action of A on u is to stretch and rotate it, giving the output vector v. This is where you can see the idea of the "bigness" of A coming to play – if A doesn't stretch u by very much, then we can think of A as not particularly big. But if A stretches u by a lot, then we can think of A as "big". Therefore, the definition of matrix norm, $\|A\|$, should reflect that notion.

To capture the notion of "bigness" we ask, what is the maximum amount of stretching caused by A? To find this, consider rotating u around the origin while leaving its magnitude fixed. Then look at the length of the transformed vector $v = Au$. In the general case, this vector will have a maximum length for some input vector u. The norm of A will then be the ratio of the output to the input vector lengths:

$$\begin{aligned} \|A\| &= \operatorname*{argmax}_u \frac{\|v\|}{\|u\|} \\ &= \operatorname*{argmax}_u \frac{\|Au\|}{\|u\|} \end{aligned}$$

This is the definition of induced matrix norm ordinarily found in textbooks. Although it looks intimidating, it really says something simple. It says, rotate the input vector u around the origin and look for the output vector v of maximum length. Then, take the ratio of the two vector lengths to get the matrix norm. The factor of $\|u\|$ in the denominator is just a normalization factor – it says that if u is not a unit vector, then the output value should be scaled by the length of u.

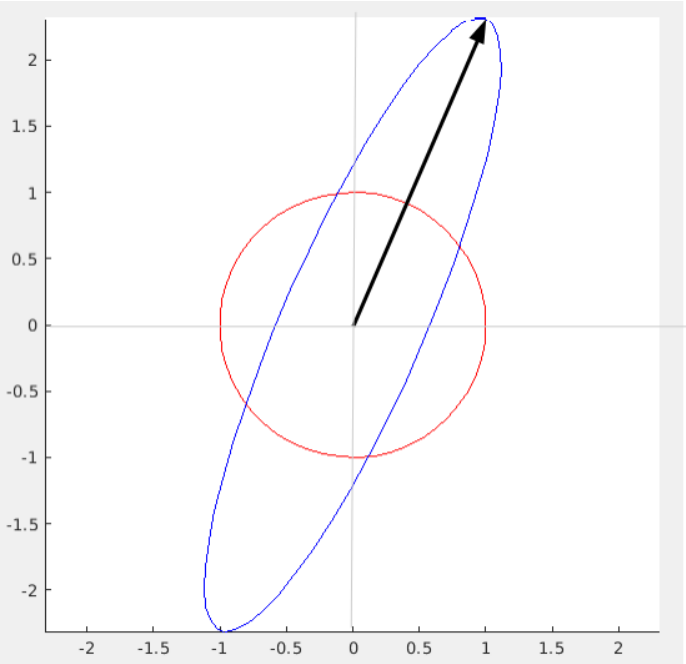

FIG. 11. *The induced matrix norm of A is the length of the longest vector lying on the surface of the ellipse generated by the matrix-vector product $v = Au$, where u is a unit vector.*

A better way to understand this definition of matrix norm is to appeal to the ellipse visualization presented earlier in this section. Using the ellipse visualization, the norm of A is the length of the ellipse's semi-major axis. This is shown in Figure 11, where the arrow identifies the semi-major axis. The length of this arrow is exactly the norm of A.

3.2. Singular value decomposition and the ellipse visualization. The singular value decomposition (SVD) of a matrix is an operation which may be considered as a generalization of the eigenvalue decomposition (EVD) [2]. Unlike the EVD, however, a matrix always has an SVD, regardless of whether it is square or rectangular, singular or not. The SVD separates the matrix A into three different factors (pieces). The decomposition performed by the SVD is

$$A = U\Sigma V^T$$

where U and V are orthogonal matrices, and Σ is a diagonal matrix. The elements on the diagonal of Σ are denoted by σ_i and are called the "singular values" of matrix A. More information about the SVD is available in many linear algebra textbooks, so I will assume you already know about the SVD.

The ellipse visualization of matrix A is related to its SVD on several levels. The first relationship is simple: The two singular values of A are the lengths of the semi-major and semi-minor axes of the ellipse. This is shown in Figure 12, where the semi-major and semi-minor axes are indicated by arrows.

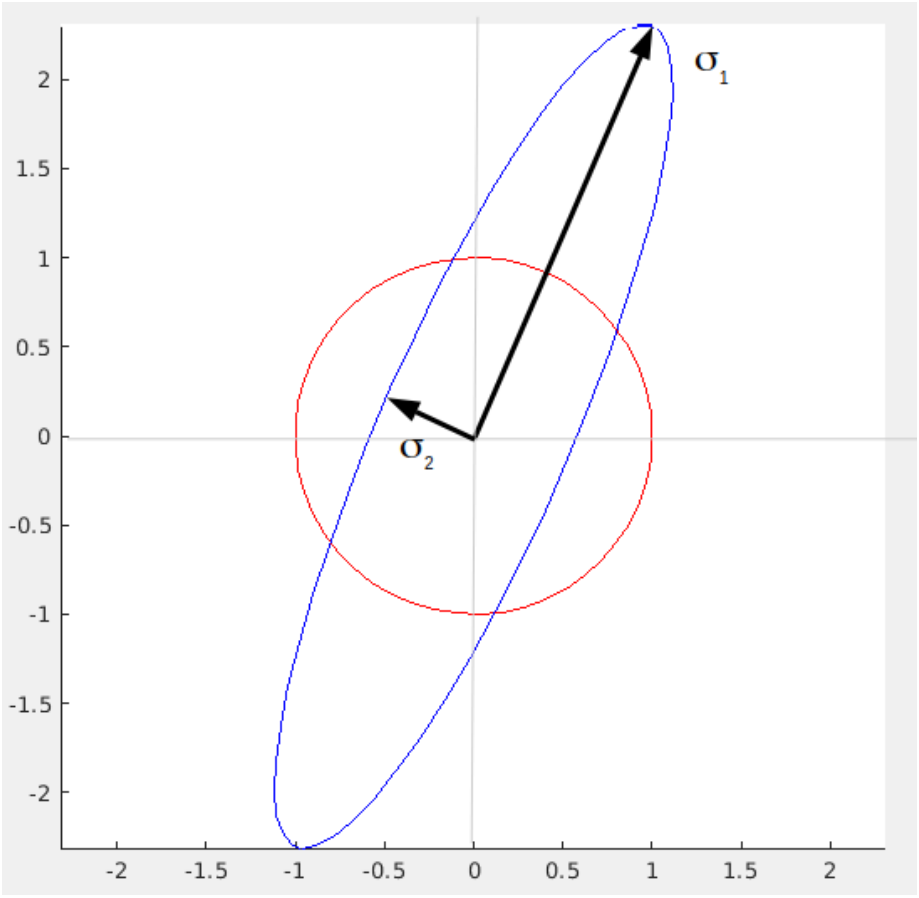

FIG. 12. *The lengths of the semi-major and semi-minor axes of the ellipse are given by the singular values of A: σ_1 and σ_2. The input circle is red, the output ellipse is blue.*

This property of the SVD generalizes to N dimensions – for an $N \times N$ matrix consider the ellipsoid created by matrix-vector multiplication with the unit ball (sphere), $v = Au$. Again, u is the set of unit vectors in N-space. The length of each semi-axis of the output ellipse v is given by the corresponding singular value in the SVD of A. Figure 13 shows a 3-dimensional unit ball and the resulting ellipsoid computed using the visualization library VTK. The numeric values of σ_i are also shown in the figure.

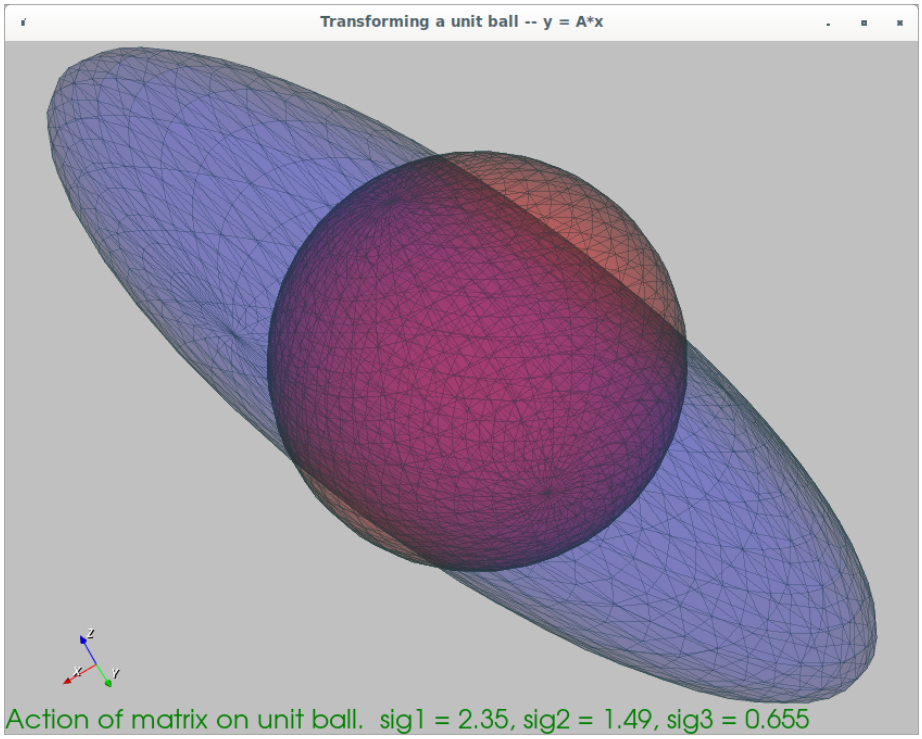

FIG. 13. *For the 3-dimensional ellipsoid created by Au the lengths of the axes of the ellipse are given by the singular values of A: σ_1, σ_2, and σ_3. The input sphere is red, the output ellipsoid is blue.*

3.3. The full action of SVD – rotation, stretching, rotation. The last subsection showed the relationship of the singular values of A to the lengths of the axes of the ellipse created by $v = Au$. There is much more to the relationship. Consider the action of the decomposed matrix A on the unit circle,

$$v = Au = U\Sigma V^T u \tag{3.3}$$

Both U and V are orthogonal matrices. Therefore, the action of A on u can be visualized as a three-step transformation:

1. Multiplication by V^T. Since V is orthogonal (as is V^T), the operation $V^T u$ corresponds to rotating the circle with no stretching.
2. Multiplication by Σ. Since Σ is a diagonal matrix, its action is to stretch the circle in the directions of the underlying coordinate axes (i.e. the basis spanning the space of A). The stretching action is what creates the output ellipse from an input circle.
3. Multiplication by U. Again, because U is orthogonal, multiplication by U corresponds to taking the ellipse and rotating it.

This three-step action is shown in Figure 14 The net result of these three transformations is to take a unit circle and transform it into an ellipse as we have already seen. Now we see the fact that the ellipse's axis lengths are given by the singular values of A is a natural consequence of multiplication by the diagonal matrix $\Sigma = \left(\begin{smallmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{smallmatrix}\right)$ – the ellipse is stretched along axis i by σ_i. Additionally, we

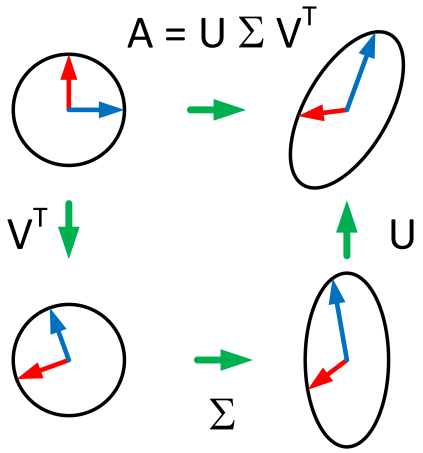

FIG. 14. *The action of A decomposed into the three steps implemented by the SVD. The first action is rotation via the orthonormal matrix V^T, then stretching via the diagonal matrix Σ, and finally a second rotation via the orthonormal matrix U.*

also see why the output ellipse is tilted: the tilt of the ellipse is due to rotation by U as the final step of the three-step transformation.

One final point is worth mentioning: In the last subsection we found that the induced norm of A is the length of the semi-major axis of the ellipse v. Since the ellipse is created by the stretching operations implemented by Σ, the longest axis is created by the largest singular value, σ_1. Therefore, the induced norm of A is also given by the largest singular value, i.e. $\|A\| = \sigma_1$.

3.4. Condition number. Another important property of a matrix A is its condition number [5]. The condition number of a matrix is a scalar which characterizes how close the matrix is to singular. One usually writes the Greek letter κ to stand for the condition number. The condition number is a measure of how sensitive the linear system

$$Ax = b \tag{3.4}$$

is to small perturbations of A and b. In general, the value of x obtained by solving (3.4) will be inexact due to rounding error and other numerical non-idealities. You might think that if A is close to singular, the error incurred in solving (3.4) will be larger than if A is far from singular. This is indeed the case in practice, and condition number κ quantifies this effect.

The condition number is traditionally defined via

$$\kappa = \left\|A^{-1}\right\|\|A\| \tag{3.5}$$

although using this expression to actually compute a value for κ is generally discouraged. (The reason is that computing the matrix inverse A^{-1} is an $O(n^3)$ operation, while the condition number is more easily computed using the SVD, which is typically calculated iteratively.) Nonetheless, you can see from (3.5) the general behavior of κ. On one hand, the most stable matrix is the identity

matrix,

$$I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \ddots \\ 0 & 0 & \ddots & 0 \\ 0 & \ddots & 0 & 1 \end{pmatrix}$$

and for this matrix it's easy to see that the condition number is $\kappa = 1$. (You can use the unit circle matrix visualization to see why this is true.) On the other hand, if A is singular, then $\left\|A^{-1}\right\| = \infty$, so the condition number is also ∞. Therefore, the condition number of any matrix will lie somewhere in the domain $1 \leq \kappa \leq \infty$. The condition number is a reliability measure for the solution x of the linear system $Ax = b$:

- The smaller the condition number (closer to 1), the better behaved is the matrix. That means that solving $Ax = b$ will give good results for x. In this case one says the matrix is "well conditioned".
- The larger the condition number (closer to ∞), the closer is the matrix to singular. In this case, solving the system $Ax = b$ is very sensitive to small changes in A and b (i.e. from round-off) so the result x may have large errors. In this case one says the matrix is "badly conditioned".

This language extends to numeric problems themselves – one can speak of well or badly conditioned problems.

Having examined some properties of the condition number it's time to actually compute it. It turns out that the condition number of matrix A is intimately related to its SVD. In particular, the condition number of A is equal to the ratio of the largest to the smallest singular values of A. That is, if

$$A = U \begin{pmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & \ddots \\ 0 & 0 & \ddots & 0 \\ 0 & \ddots & 0 & \sigma_N \end{pmatrix} V^T$$

then

$$\kappa = \sigma_1 / \sigma_N$$

This can be seen by considering (3.5). The norm of A is the largest singular value σ_1. Meanwhile, taking the SVD of A^{-1} yeilds

$$A^{-1} = (USV^T)^{-1} \quad = (V^T)^{-1} S^{-1} U^{-1} = V S^{-1} U^T$$

and since S is diagonal we have

$$S^{-1} = \begin{pmatrix} 1/\sigma_1 & 0 & 0 & 0 \\ 0 & 1/\sigma_2 & 0 & \ddots \\ 0 & 0 & \ddots & 0 \\ 0 & \ddots & 0 & 1/\sigma_N \end{pmatrix} \tag{3.6}$$

Accordingly, the norm of A^{-1} is its largest singular value, which is $1/\sigma_N$. Therefore, from the definition (3.5) we have $\kappa = \sigma_1/\sigma_N$.

What does this mean for our ellipse visualization? It means that the condition number of A may be read directly from the shape of the ellipse. If the ellipse is close to circular in shape, then the matrix is well conditioned. But if the ellipse is elongated and skinny, then the matrix is badly conditioned. This is illustrated in Figure 15 which shows a series of ellipses corresponding to randomly generated matrices A. Note that the round ellipses have smaller condition numbers than the skinny ones.

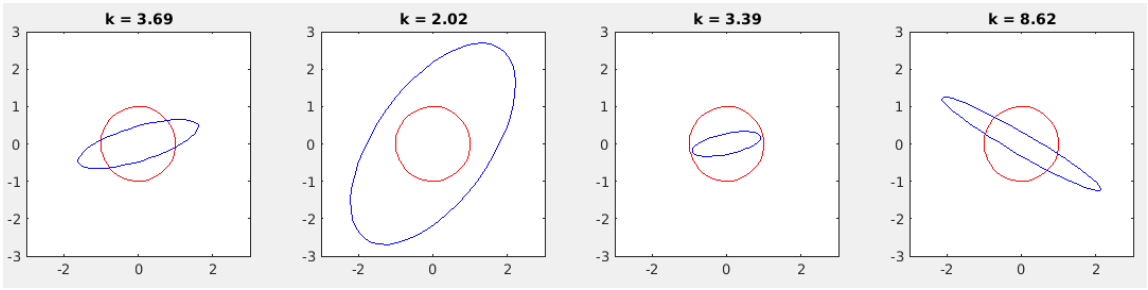

FIG. 15. *Ellipses created by different, random matrices A. Ellipses which are closer to circular have smaller condition numbers κ while ellipses which are skinnier have higher condition numbers.*

This picture extends to a general N-dimensional ellipsoid – if the ellipsoid is close to spherical, the matrix is well conditioned. But if the ellipsoid is squeezed or close to flat in one or more dimensions, then the corresponding matrix is close to singular. In the limit that the matrix is singular, one or more of the singular values are exactly zero, and the corresponding ellipsoid is completely flat in those directions. If the ellipsoid looks like a pancake then the matrix is singular!

3.5. Application: Sensitivity of linear solve. Let's look further into what the condition number κ says about the solution to $Ax = b$. Consider solving this system on a computer using floating point doubles [4]. The mantissa of a double is 53 bits long, which corresponds to roughly 16 decimal digits. If no errors are introduced using doubles to represent the known objects A and b, then the solution x should be valid to around 16 decimal digits. However, errors always creep in to a computer computation performed with finite-length words.

The linear solve operation takes as input the matrix A and the RHS (right hand side) vector b, and outputs the vector x which satisfies $Ax = b$. Now imagine we could calculate the solution x exactly – without any rounding error. What is the difference between this "mathematically true" solution x and the actual solution $\tilde{x}$ obtained from calculation? We call the difference $\delta x = x - \tilde{x}$ the "forward error" of this computation since it measures the distance between the computed and mathematically true solutions which have been arrived at in the "forward direction", i.e. starting from known inputs A and b and arriving at the output $\tilde{x}$.

A different way to think about errors in the linear solve operation involves the so-called backward error [6]. In this case, we start from the computed output $\tilde{x}$ and ask, if we used exact calculation, what is the corresponding input vector $\tilde{b}$ which would give this result? The difference between this input and the actual input, $\delta b = b - \tilde{b}$ is the backward error. We call it the backward error because it works in the "backward direction", i.e. it starts from the output $\tilde{x}$ and infers the input $\tilde{b}$ which gives this output.

It can be shown that the relative (i.e. normalized) forward and backward error are related by [7]

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa \frac{\|\delta b\|}{\|b\|} \tag{3.7}$$

This is a well-known equation which says that a small perturbation or error in the input b is amplified by the condition number κ when computing the output $\tilde{x}$. That is, the condition number measures the sensitivity of the output error to input perturbations. And since rounding error may be regarded as a type of perturbation to the input, the condition number provides a measure of how much error to expect when solving $Ax = b$.

A nice demonstration of this fact can be constructed as follows:

1. Create a random matrix A with a given condition number κ.
2. Create a random vector x. Ordinarily, this is the output of the solve operation but in this example we use it as the starting place for a "round trip" through a solve.
3. Compute $b = Ax$. Ordinarily, b is an input to the solve operation, but in this case we'll regard the system we just constructed from A, b, and x as the "mathematically true" system and look for the effect of perturbations on the $Ax = b$ system.
4. Create a small, random perturbation vector δb whose norm is on the order of the machine epsilon ϵ_o = 1e-16.
5. Create the perturbed RHS vector $\tilde{b} = b + \delta b$.
6. Perform a linear solve on the perturbed system to get $\tilde{x} = A^{-1}\tilde{b}$.
7. Compute the forward error $e_{fwd} = \|x - \tilde{x}\|/\|x\|$.
8. Compute the backward error $e_{bkwd} = \left\|b - \tilde{b}\right\|/\|b\|$.
9. Compute the ratio $\kappa_{comp} = e_{fwd}/e_{bkwd}$.

A Matlab program which implements this algorithm is presented in Appendix A.3. Running the program 10 times using a condition number $\kappa = 100$ gives the following output:

```
>> test_cond_thm(100)
k input = 100.000000, k computed = 62.872969
k input = 100.000000, k computed = 42.993345
k input = 100.000000, k computed = 10.273825
k input = 100.000000, k computed = 13.480149
k input = 100.000000, k computed = 2.830090
k input = 100.000000, k computed = 16.569123
k input = 100.000000, k computed = 27.897132
k input = 100.000000, k computed = 24.836220
k input = 100.000000, k computed = 27.122448
k input = 100.000000, k computed = 29.466046
```

As can be seen, the inferred condition number is bounded by $\kappa = 100$ above. That is, although at least one trial got above 60, the ratio clearly never exceeded 100. This demonstrates that the inequality (3.7) is obeyed, and shows how the condition number characterizes the sensitivity of the linear solve operation to the effects of input error or perturbations.

4. The quadratic form visualization. So far we have studied the ability of a matrix A to act as a linear transform. The idea was to visualize a matrix by thinking about how it transformed various objects via matrix-vector multiplication. In this section we investigate a different way to examine the operation of a matrix – through the creation of quadratic forms.

A quadratic form is a mapping from an N-vector to a scalar, $\mathbb{R}^N \to \mathbb{R}$. For matrices, the mapping is

$$f(u) = u^T A u \tag{4.1}$$

where u is a vector and A is a matrix. We think of the quadratic form as a map which accepts the elements of the vector u as input, and outputs the scalar value $f(u)$. Then, we let u wander all over $\mathbb{R}^N$ and plot the resulting $f(u)$. If u are all values on the 2D plane, then the resulting $f(u)$ is a surface floating above the plane.

Although one my create a quadratic form using any arbitrary matrix, for what follows we will restrict ourselves to symmetric matrices A with all real elements. The reasons for this are:

- Later we will investigate the eigenvalue and eigenvectors of A and their relationship to the visualization. Symmetric matrices with real coefficients have real eigenvalues, making it straightforward to interpret the visualization.
- The surfaces created by the quadratic form are best interpretable for symmetric matrices.

Restricting ourselves to real, symmetric matrices is not a major problem since most real-world problems involve real, symmetric matrices. For example, the matrices obtained from finite difference computations are real and symmetric as a general rule.

An example quadratic form generated by an arbitrarily-chosen matrix is shown in Figure 16. The input matrix is $A = \left(\begin{smallmatrix} 1.9 & 0.3 \\ 0.3 & 0.5 \end{smallmatrix}\right)$. The resulting surface generated by this matrix is a paraboloid which increases quickly in one direction, and more slowly in the perpendicular direction. More

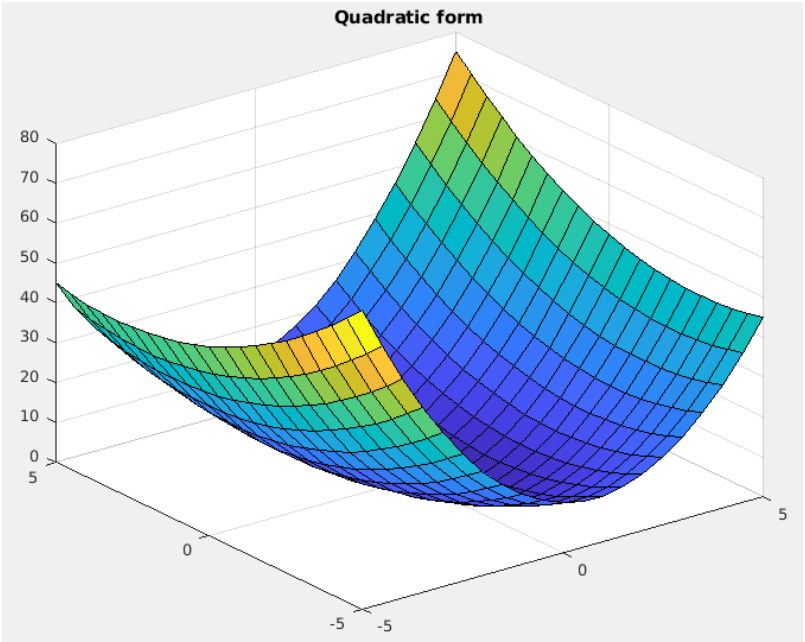

FIG. 16. *Plot of the surface produced by the quadratic form* $f(u) = u^T A u$ *for arbitrarily-chosen symmetric matrix* $A = \left(\begin{smallmatrix} 1.9 & 0.3 \\ 0.3 & 0.5 \end{smallmatrix}\right)$.

examples produced by random symmetric matrices are shown in Figure 17.

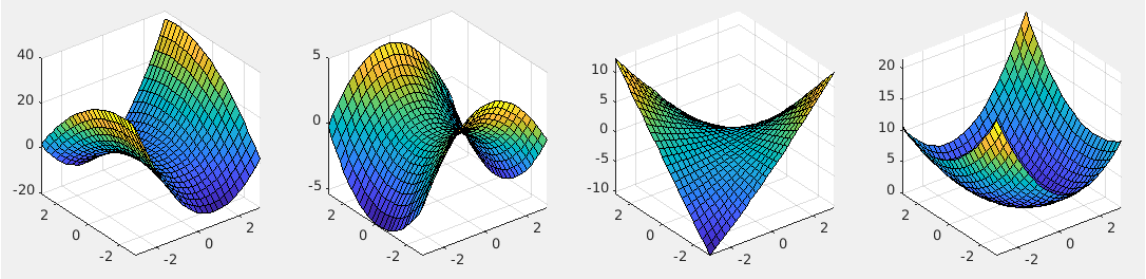

FIG. 17. *Four different quadratic forms created by four different random matrices.*

4.1. Eigenvalues and eigenvectors. Now that we have a matrix visualization involving quadratic forms, what can we learn from it? The shape of the surface carries a lot of information about the matrix A. Figure 18 shows the surfaces obtained from four different matrices:

1. Upward opening paraboloid. $A = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$. Eigenvalues $\lambda_1 = \lambda_2 = 1$.
2. Downward opening paraboloid. $A = \left(\begin{smallmatrix} -1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$. Eigenvalues $\lambda_1 = \lambda_2 = -1$.
3. Saddle (hyperbolic paraboloid). $A = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$. Eigenvalues $\lambda_1 = 1, \lambda_2 = -1$.
4. Cylindrical paraboloid (degenerate form – singular matrix). $A = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$. Eigenvalues $\lambda_1 = 2, \lambda_2 = 0$.

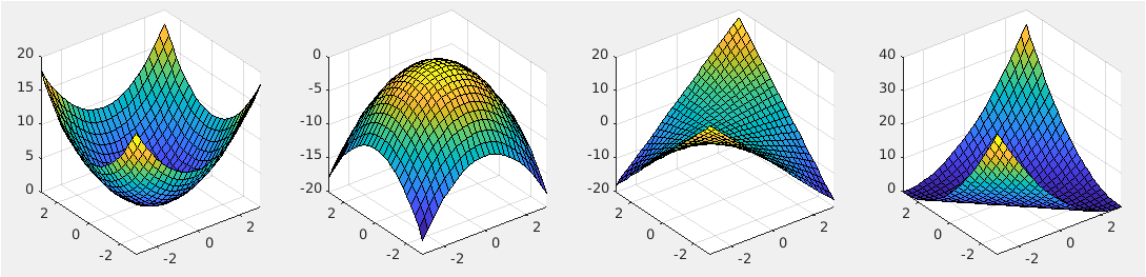

FIG. 18. *From left to right: 1. Upward opening paraboloid: the matrix is positive definite, both eigenvalues are positive. 2. Downward opening paraboloid: the matrix is negative definite, both eigenvalues are negative. 3. Saddle: The matrix is indeterminate, one eigenvalue is positive, one is negative. 4. Cylidrical paraboloid: The matrix is singular, one eigenvalue is positive, the other is zero.*

The question is, is there a way to characterize each matrix which predicts something about the quadratic form surfaces it creates? The answer is yes, and the characterization involves the eigenvalue decomposition (EVD) of the matrix,

$$A = Q \Lambda Q^T \tag{4.2}$$

where Q is an orthonormal matrix, and Λ is a diagonal matrix whose diagonal elements are the eigenvalues of matrix A. I will assume you have some basic familiarity with eigenvalues and eigenvectors; the subject is well treated in the standard linear algebra textbooks. One important thing to note is that by restricting ourselves to real, symmetric matrices, the matrices Q are automatically

orthonormal and the eigenvalues are real. These nice properties are not guaranteed for random, non-symmetric matrices. This is a big reason we have restricted ourselves to real, symmetric matrices in this section.

Here are the points to notice about quadratic forms and the EVD:

- The surface produced by $u^T Au$ is characterized by two curvatures in two orthogonal directions. Positive curvature corresponds to an upward-facing parabola and negative curvature corresponds to a downward-facing parabola. In this case, "curvature" refers to the second derivative of the surface along a particular line passing through the origin.
- The signs of the two eigenvalues of A correspond to the two curvature directions of the surface. A positive eigenvalue corresponds to an upward-facing parabola and a negative eigenvalue corresponds to a downward-facing parabola. If an eigenvalue is zero, it corresponds to one direction of the surface which faces neither up nor down – this is the degenerate case shown in Figure 18
- The two directions of principle curvature are given by the eigenvectors of A. The term "principle curvature" means the minimum and maximum values of curvature along all lines passing through the origin.
- A positive definite matrix is one whose eigenvalues are all positive. Such matrices show up frequently in real-world problems. For a 2x2 matrix the corresponding surface is a paraboloid opening upward, such as is shown in Figure 18. For 3x3 and higher dimensionality matrices the same holds true, except we can't visualize them since we can't visualize surfaces in 4 and higher dimensions.
- A negative definite matrix is one whose eigenvalues are all negative. For a 2x2 matrix the corresponding surface is a paraboloid opening downward, similar to the example shown in Figure 18.
- A matrix whose eigenvalues have mixed sign is referred to as "indefinite". The corresponding surface is a saddle, also shown in Figure 18.

In summary, the eigenvalues of A predict what type of surface will be obtained and the eigenvectors of A predict the directions of maximum and minimum curvature.

4.2. Relation between ellipse and quadratic form visualizations. Consider the case where matrix A is positive definite. The associated quadratic form is an upward-opening paraboloid. Now consider slicing the paraboloid with planes parallel to the x-y plane as shown in Figure 19. The lines of intersection between the paraboloid and the slicing planes are called "levelsets" in optimization speak, or the "isolines" in computer graphics speak. Importantly for us, the lines are manifestly ellipses – a fact that should immediately remind us of the ellipse visualization presented in section 3. The natural question is, is there a relationship between these ellipses and the ellipses generated by matrix-vector multiplication studied in section 3? The answer is yes, the isoline ellipses depicted in Figure 19 and the ellipses depicted in Figure 9 are indeed related. However, the relationship is not as straightforward as you might think. The goal of this subsection is to derive the relationship.

First, recall that the isoline ellipses are described by the following equation:

$$u^T Au = c \tag{4.3}$$

where the constant c measures the height of the cutting plane off the "ground" (i.e. the $z = 0$ plane). For what follows take $c = 1$. Next imagine that A may be decomposed into a product $A = B^T B$. Since A is positive definite we know this is the case – the decomposition is essentially

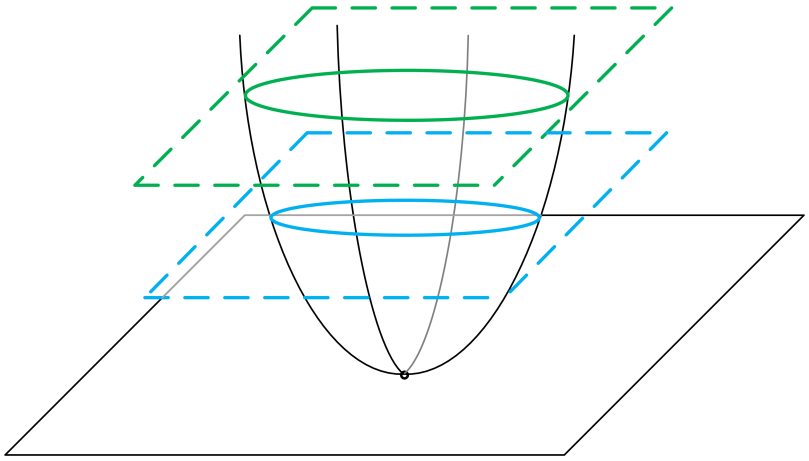

FIG. 19. *The levelsets (isolines) of the sliced paraboliod are ellipses, shown here as solid green and blue lines corresponding to different slicing planes.*

the Cholesky decomposition [8]. Using this, (4.3) becomes

$$1 = u^T B^T B u = (u^T B^T)(Bu) = e^T e \tag{4.4}$$

where $e = Bu$ is a vector. This equation says that the 2-norm of e is one. Therefore, the set of all vectors e are unit vectors, and their tips lie on the the unit circle. This was the starting point of the ellipse visualization back in section section 3! That means the ellipses defined by (4.3) are created by the matrix-vector multiplication

$$u = B^{-1} e$$

where e is the unit circle and B is derived from A via $A = B^T B$.

So the the isoline ellipses and the ellipses generated from the unit circle under matrix-vector multiplication are indeed related. But can we find a straightforward forumula connecting them? Consider the image of the unit circle under A,

$$w = Ae$$

where e is the unit circle and w is its image as an ellipse. Knowing that, and using the definition of B we have

$$w = B^T B e = B^T B B u$$

This says that w is the image of u under matrix-vector multiplication. In particular, the first product Bu returns a unit circle, and then $B^T B$ transforms it into the ellipse w. The relationship between u and w is depicted in Figure 20, which shows the two ellipses, with each computed in two ways:

1. The isoline defined by $1 = u^T A u$ and the ellipse derived from the unit circle $u = B^{-1} e$.
2. The ellipse generated from the transform of the unit circle $y = Ae$ and the transform of the ellipse in item 1, $w = B^T B B u$.

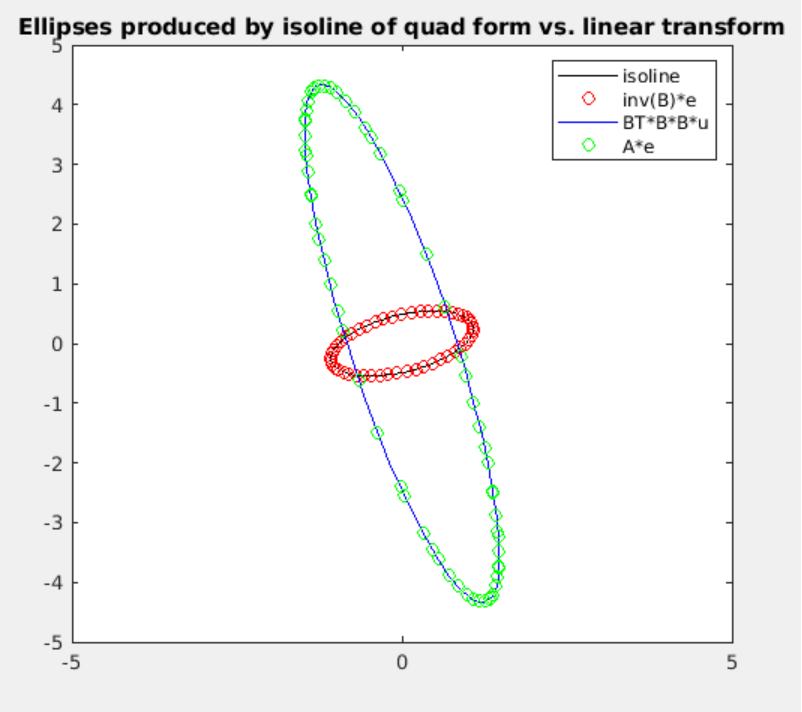

FIG. 20. *Four ellipses associated with matrix A: two generated by the quadratic form, and two generated by the linear transform visualization.*

4.3. Condition number and the quadratic form visualization. Recall using the ellipse visualization in subsection 3.4 as a proxy of the condition number. Can we also use the quadratic form visualization as a way to understand the condition number of A? As usual, when I ask a rhetorical question in this article the answer is "yes".

The keys to this visualization are the eigenvalues of A. It can be shown that the eigenvalues of a symmetric, positive definite matrix are equal to its singular values, $\lambda_i = \sigma_i$. Therefore, the condition number of A is

$$\kappa = \frac{\sigma_{max}}{\sigma_{min}} = \frac{\lambda_{max}}{\lambda_{min}}$$

Recall that the curvature of the paraboloid generated by $f(u) = u^T Au$ in any of the principal directions is proportional to the corresponding eigenvalue λ_i. The larger the λ_i, the larger is the curvature, so the tighter is the parabola. The smaller the λ_i, the smaller is the curvature, and so the more open is the parabola. This effect is shown in Figure 21 which shows the paraboloids generated by two different A matrices, one with condition number 1 and the other with condition number 10. The paraboloid with condition number 10 is quite clearly squeezed in one direction since its principal curvature in that direction is high. The squeezed parabola causes some iterative algorithms like gradient descent to converge slowly, but that is a topic for a different article.

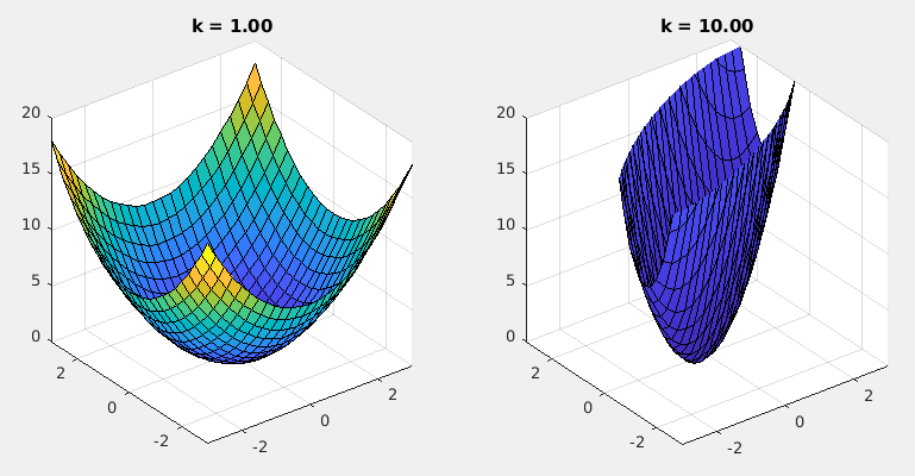

FIG. 21. *On the left is a paraboloid generated by a matrix with condition number* $\kappa = 1$. *On the right is a paraboloid generated by a matrix with condition number* $\kappa = 10$.

5. Summary and review. This paper has presented a guided tour of different ways to think about a matrix in geometric terms. Here is a summary of different matrix visualizations provided by this article:

- Visualize the action of a matrix A transforming the unit square into a parallelogram as shown in Figure 4. The parallelogram's area is given by the determinant of the input matrix A.
- Visualize the action of a matrix A transforming the unit circle centered at the origin. The result is an ellipse as shown in Figure 9. The lengths of the axes of the ellipse are given by the singular values Σ generated by the SVD of A, $A = U\Sigma V^T$.
- Imagine the action of the decomposed matrix $A = U\Sigma V^T$ as three separate actions: 1. Rotation by V^T, 2. stretching by Σ, 3. a final rotation by U. This is shown in Figure 14
- Specializing to the case of real, symmetric A, visualize the surface created by the quadratic form $f(u) = u^T Au$. Examples are shown in Figure 18. The exact shape taken by the surface (paraboloid, saddle, etc.) depends upon the eigenvalues of A.
- A final visualization applies to a positive definite matrix: the cutting planes intersecting the quadratic form $f(u) = u^T Au$ form an ellipse, Figure 19. The ellipse is not the same size nor shape as that found by matrix-vector multiplication, but the two ellipses are related via Cholesky decomposition of matrix A.

My experience is that these visualizations give me insights into the behavior of numerical problems. I hope you find them useful too as you continue your mathematical studies.

Appendix A. Matlab codes.

A.1. Action of a random matrix on a unit square.

```
function transform_square()
  % This function takes a set of points forming a
  % unit square x and plots it.  Then it generates
  % a random 2x2 matrix, A, and forms the product
```

```
  % y = A*x.  It then plots the product
  % on the same plot.

  % Create unit square from 4 sides
  N = 625;  % Number of points per side
  s = linspace(0,1,N);
  z = zeros(size(s));
  o = ones(size(s));

  % Create points. Each point is a 2-element column
  % vector, [x;y]; the points are concatenated into
  % a short, fat 2xN matrix  which holds all the
  % points in the square.
  s1 = [s;z]; % bottom
  s2 = [o;s]; % right side
  s3 = [s;o]; % top
  s4 = [z;s]; % left side
  x = [s1,s2,s3,s4];   % square

  % Plot square
  plot(x(1,:), x(2,:), 'ro', 'MarkerFaceColor', ...
    'red', 'MarkerSize', 0.5)
  hold on
  axis([-3, 3, -3, 3]);

  % Now create random 2x2 matrix A.  The elements
  % have mean 0 and variance 1.
  A = randn(2,2);

  % Now compute matrix-vector product and plot it.
  y = A*x;
  oldplot = plot(y(1,:), y(2,:), 'bo', 'MarkerFaceColor', ...
    'blue', 'MarkerSize', 0.5);

  % Report det(A)
  fprintf('det(A) = %f\n', det(A))
end
```

A.2. Action of a random matrix on a unit circle.

```
function A = ellipses()
  % This fcn generates a random 2x2 matrix A, and
  % then applies it to the unit circle.  It then
  % plots the result.  The goal is to visualize a
  % matrix via its transformation of a unit
  % circle (or ball in ND).

  % Create unit circle u (parameterized by theta)
```

```
  theta = linspace(0, 2*pi, 50);
  u = [cos(theta); sin(theta)];  % Column vector

  % Now create random 2x2 matrix, and apply it to u.
  A = randn(2);
  v = A*u;

  plot(u(1, :), u(2, :), 'r')  % Reference circle is red
  hold on
  plot(v(1, :), v(2, :), 'b')  % Transformed circle is blue
  axis([-3, 3, -3, 3], 'square');
end
```

A.3. Condition number theorem.

```
function test_cond_thm(k)
  % Input: k = condition number of matrix to make

  for i=1:10
    N = 5;
    A = randn_cond(N,N,k);
    x = randn(N,1);
    b = A*x;

    db = randn(N,1)/eps(1);
    bt = b + db;
    xt = A\bt;

    relfwderr = norm(xt-x)/norm(x);
    relbkwderr = norm(bt-b)/norm(b);
    kcomputed = relfwderr/relbkwderr;

    fprintf('k input = %f, k computed = %f \n', k, kcomputed)
  end

end
```

REFERENCES

[1] Eric W. Weisstein, "Matrix Norm." From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/MatrixNorm.html

[2] Nick Higham, "What Is the Singular Value Decomposition?", https://nhigham.com/2020/10/13/what-is-the-singular-value-decomposition/

[3] Nick Higham, "What is an orthogonal matrix?", https://nhigham.com/2020/04/07/what-is-an-orthogonal-matrix/

[4] Cleve Moler, "Floating Point Numbers", https://blogs.mathworks.com/cleve/2014/07/07/floating-point-numbers/

[5] Cleve Moler, "What is the condition number of a matrix?", https://blogs.mathworks.com/cleve/2017/07/17/what-is-the-condition-number-of-a-matrix/

[6] Nick Higham, "What Is Backward Error?", https://nhigham.com/2020/03/25/what-is-backward-error/

[7] For a full discussion, see L. Trefethen, and D. Bau, "Numerical Linear Algebra", SIAM, Philadelphia, (1997)
[8] Nick Higham, "What Is a Cholesky Factorization?", https://nhigham.com/2020/08/11/what-is-a-cholesky-factorization/