Math 4610 Lecture Notes

Matrix Operations in Computational Mathematics *

Joe Koebbe

October 28, 2019

*These notes are part of an Open Resource Educational project sponsored by Utah State University

Matrix Operations: Basic Operations on Vectors

The central algorithms used in most computer simulations involve computer codes that perform basic vector and matrix operations. Operations like matrix-vector multiplications are central to methods for solving linear systems of equations and determination of eigenvalues and eigenvectors. In this section of the class notes, we will cover a number of basic vector and matrix operations. Suppose that we want to compute the solution of a linear system of equations of the form

$$A\ \mathbf{x} = \mathbf{b}$$

where A is the coefficient matrix, $\mathbf{b}$ is an input vector, and we want to find $\mathbf{x}$ is the vector of unknowns that we are interested in finding. Knowing what we know about roundoff error and machine precision representation of numbers, any solution we obtain will most likely be an approximation of the solution.

We will need some way to measure or compute the error associated with the approximation obtained. There are as many ways to measure errors as on can think of. However, we will settle for three standard meausres of error introduced in the next section of the notes. To implement the error definitions, we will need to write codes that perform basic vector operations one would learn about in a standard linear algebra course in a college curriculum. The vector operations we will begin with are (1) vector addition, (2) vector subtraction, (3) scalar multiplication, and (4) then to errors and other vector operations.

Matrix Operations: Basic Operations on Vectors

For a set of objects to be a vector space we need to have closure of addition and scalar multiplication. In fact, if V is a linear vector space, then for any $\mathbf{u}$ and $\mathbf{v}$ in the set, then the sum of the vectors is in V. That is $\mathbf{w}$, defined by

$$\mathbf{w} = \mathbf{u} + \mathbf{v}$$

is also in the set of vectors. Also, if $\mathbf{v} \in V$ and a is any real number then $\mathbf{v}$ defined by

$$\mathbf{v} = a\ \mathbf{v}$$

is also in V. Based on this idea, we should at least implement a pair of codes - one that compute the sum of two vectors and one that computes a scalar multiple of a vector.
Almost all of the work we will do will involve real vectors of length n. A vector will be of the form $(v_1, v_2, \ldots, v_n)^T$ where each component, $v_i, i = 1, \ldots, n$ is a real number. The superscript, T, denotes the transpose. It should be noted that there is a differnce between row and column vectors in linear algebra. In our vector notation, we will by default assume that a given vector is a column vector. Transposing a column vector results in a row vector and vice versa. This convention lends itself to operations like matrix-vector multiplication.
So, let's start with the sum of two vectors. The conventional definition of addition of vectors is a componentwise definition. If $\mathbf{u}$ and $\mathbf{v}$ are two vectors of length n. Then

$$\mathbf{u} + \mathbf{v} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_1 + v_1 \\ u_2 + v_2 \\ \vdots \\ u_n + v_n \end{bmatrix}$$

For completeness, we can also write down a relationship between row and column vectors. That is,

$$\mathbf{v}^T = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}^T = \begin{bmatrix} v_1, v_2, \vdots v_n \end{bmatrix}$$

and

$$\begin{bmatrix} v_1, v_2, \vdots v_n \end{bmatrix}^T \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \mathbf{v}$$

This is one of the easiest operations that we can code. There are a few things that might need checking. For example, we need to check the dimension of the two vectors. Without any checks on dimensions, the code, written in Java might look like the following.

```
public double[] vecadd(double[] u, double[] v) {
  //
  // get the dimension of the first vector and instantiate some storage for
  // the output vector
  // -----------------
  //
  int n = u.length;
  double [] output = new double[n];
  //
  // compute the componentwise sum for the vector
```

```
  // ------------------------------------------
  //
  for(int i=0; i<n; i++) output[i] = u[i] + v[i];
  //
  // return the output
  // -----------------
  //
  return output;
  //
}
```

Before coding a scalar multiple method for our library, it would be best to write a code for computing the difference in a pair of vectors. Given two vectors $\mathbf{u}$ and $\mathbf{v}$, the formula for the difference is the following.

$$\mathbf{u} - \mathbf{v} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} - \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_1 - v_1 \\ u_2 - v_2 \\ \vdots \\ u_n - v_n \end{bmatrix}$$

It is an easy exercise to write a code that will compute the difference of two vectors. All that needs to happen is to give the routine a unique name and change the addition to a subtraction. For exaxmple,

```
public double[] vecsub(double[] u, double[] v) {
  //
  // get the dimension of the first vector and instantiate some storage for
  // the output vector
  // -----------------
  //
  int n = u.length;
  double [] output = new double[n];
  //
  // compute the componentwise sum for the vector
  // ------------------------------------------
  //
  for(int i=0; i<n; i++) output[i] = u[i] - v[i];
  //
  // return the output
  // -----------------
  //
  return output;
  //
}
```

This particular routine can be used as a means to compute the difference between two vectors as a part of measuring errors in vector approximations.
The last code to write in this section is the scalar multiplication code. The vector operation is defined by the following. For any vector, $\mathbf{v}$, and number, a, scalar multiplication is defined by

$$a\ \mathbf{v} = a \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} a\ v_1 \\ a\ v_2 \\ \vdots \\ a\ v_n \end{bmatrix}$$

and the code to implement this operation is given by

```
public double[] scalvec(double a, double[] v) {
  //
  // get the dimension of the first vector and instantiate some storage for
  // the output vector
  // -----------------
  //
  int n = u.length;
  double [] output = new double[n];
  //
  // compute the componentwise sum for the vector
  // -------------------------------------------
  //
  for(int i=0; i<n; i++) output[i] = a * v[i];
  //
  // return the output
  // -----------------
  //
  return output;
  //
}
```

The methods developed in this section of the notes provides a starting point for a number of other methods. It will be easy to cut and paste and rename methods to do most of the basic matrix operations

Matrix Operations: The l_2 Length of a Vector

In many problems it will be necessary to measure the length or magnitude of a vector. So, we need some way to determine this length of a vector. Actually, there are infinitely many ways to do this. The approach most students see first involves the Euclidean distance in multiple dimensions. That is, the length can be defined as follows.

$$||v|| = (v_1^2 + v_2^2 + \cdots + v_n^2)^{1/2}$$

where the notation, $|| * ||$, provides a mathematical notation for the magnitude of the vector. This can be implemented easily into a method or routine that will return this length of a vector.

```
public double l2norm(double[] v) {
  double sum = 0.0;
  //
  // extract the length of the vector
  // --------------------------------
  //
  int n = v.length;
  //
  // compute the sum of squares
  // --------------------------
  //
  for(int i=0; i<n; i++) sum = sum + v[i] * v[i];
  //
  // return the value
  // ----------------
  //
  return Math.sqrt(sum);
  //
}
```

Note that the Java package including the square root function provides the last piece of the code.

Matrix Operations: Definition of the Norm of a Vector

There are many ways to compute the length of a vector. In this section we will consider several ways to do this. However, we need to have a general definition of the length of a vector. To start, the definition of a norm is given.

Definition 1 *Suppose that V is a vector space and* $\mathbf{u}$ *and* $\mathbf{v}$ *are any two vectors in V. Also, assume a is an arbitrary number. The norm of a vector is a function*

$$||*|| : V \to \Re$$

such that

1. $||v|| = 0$ *if and only if* $\mathbf{x} = \mathbf{0}$,
2. $||a\mathbf{v}|| = |a|\ ||\mathbf{v}||$, *and*
3. $||\mathbf{u} + \mathbf{v}|| \leq ||\mathbf{u}|| + ||\mathbf{v}||$.

The Euclidean length of a vector can be shown to satisfy the definition of a norm. The norm as defined in the code written for the length and thus the name **l2norm** for the method. There are an infinite number of ways to compute norms on vector spaces. A general definition for a norm on n-dimensional space is the following.

$$||v||_p = (v_1^p + v_2^p + \cdots + v_n^p)^{1/p} = \left(\sum_{i=1}^{n} |v_i|^p \right)^{1/p}$$

for any integer p. The Euclidean norm is obtained when $p = 2$. The reality in computational mathematics, there are three norms of interest including the 2-norm. The other two norms are the 1-norm and the other is the infinity-norm. The definitions for these two norms is given below.

$$||v||_1 = |v_1| + |v_2| + \cdots + |v_n| = \sum_{i=1}^{n} |v_i|$$

and

$$||v||_\infty = \max_{1 \leq i \leq n} |v_i|$$

are the definitions we will use in our work. The gold standard for measuring length is typically the 2-norm. However, in some problems the 1-norm and infinity-norm are easier to compute in many problems.
The 1-norm and infinity-norm can easily be coded into methods as was done above to the 2-norm. Using the 2-norm code, the other two methods just need a couple of minor changes to implement the other two norms into their own methods.

Matrix Operations: Errors in Vector Approximations

Just as the errors in approximating roots of functions of a single variable, having a way to compute the magnitude of the error in approximating one vector $\mathbf{v}$ by another vector $\mathbf{u}$. We can use the definitions of norms earlier in this section to define a consistent formula for the vector approximation error. If we use a generic definition of the norm of a vector, we can define

$$\text{absolute error} = ||\mathbf{v} - \mathbf{u}||$$

and

$$\text{relative error} = \frac{||\mathbf{v} - \mathbf{u}||}{||\mathbf{u}||}$$

These formulas should look familiar when compared to error measurement in root finding problems.
A code that will implement the absolute error with the 2-norm might look like

```
public double absl2err(double[] u, double[] v) {
  double sum = 0.0;
  //
  // extract the length of the vector
  // --------------------------------
  //
  int n = v.length;
  double diff = 0.0;
  for(int i=0; i<n; i++) {
    diff = u[i] - v[i];
    sum = sum + diff * diff;
  }
  //
  // return the norm of the difference
  // ---------------------------------
  //
  return Math.sqrt(sum);
  //
}
```

Of course, we could reuse code that we have already written as follows. Provided that the methods have been created and test and inserted in some sort of archive that is available for our use. A first simpler version is the following.

```
public double absl2err(double[] u, double[] v) {
  double [] diff = null;
  int n = v.length;
  diff = vecsub(u, v);
  return l2norm(diff);
}
```

An even more concise version of the method might like the following.

```
public double absl2err(double[] u, double[] v) {
  return l2norm(vecsub(u, v));
}
```