

Math 131/135/194, Fall 2004

Applied Calculus

Prof. Kaplan & Flath

Macalester College

Linear Combinations of Vectors

Goal of this lab

To visualize the geometry of a linear combination of vectors and understand when solutions to matrix equations can be exact and when they are only approximate.

Introduction

A vector is a mathematical concept that corresponds more or less to the everyday meaning of a direction, e.g., northwest, to the right, up, and so on. When we speak of a linear combination of vectors, we mean the set of points that can be reached by following the various directions in that set. For example, given the set of directions North and East, we can reach anywhere on a horizontal surface. We do this by taking steps in the North direction (the steps might be backwards, so we would be moving South) and then in the East direction. Notice that following these two directions we can't ever move off the surface, because we don't have Up in our set.

In this lab, we're going to explore how to reach a given target by taking steps forward or backward along the directions in a given set. Of course, we will want to be more formal than this. Rather than "set of directions" we will say **matrix**. Rather than "direction" we will say **vector**. A matrix is a set of vectors.

Here's an example, in symbolic notation. Suppose we are given two vectors \vec{u} and \vec{v} . How do we find the linear combination that will bring us to a specific target coordinate, \vec{b} ? That is, what are the scalar constant values x_1 and x_2 such that

$$x_1\vec{u} + x_2\vec{v} = \vec{b}$$

Some other questions related to this. Is there such a linear combination at all? If so, is it unique or is there more than one successful linear combination? If not, how can we get as close as possible to the target?

Another, equivalent symbolic form of the problem involves matrices and the operation of matrix multiplication. In this notation, the problem is written

$$\begin{pmatrix} | & | \\ \vec{u} & \vec{v} \\ | & | \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} | \\ \vec{b} \\ | \end{pmatrix} \quad (1)$$

Most people look at equations such as the ones above and assume, because the problem is stated abstractly, that the

problem is difficult. This is not at all the case. To prove this, we are going to solve some problems of this form graphically, using a video-game type interface.

Later on, we will use R to solve matrix equations numerically. But our video game is written in a system called Matlab. You don't need to learn anything about Matlab, just follow the instructions.

1. Start up the Matlab program following the instructions given in class.

2. Type in a command which will look something like
 \gg `cd g:/mathcs/math194`

3. Type in three additional commands to create three vectors:

\gg `u = [3; 1]`

\gg `v = [-1; 3]`

\gg `b = [5; 9]`

Later on we will create other vectors following the same pattern. Note that square brackets and the semi-colon are used to assemble the components of the vectors. (R uses a completely different system, unfortunately, so don't worry about mastering the Matlab syntax.)

4. The program that helps you to solve the matrix system in Equation 1 is called `addvec`. You start it like this:

\gg `addvec(b, u, v)`

Note the order of the arguments: the target point comes first.

5. Once you have entered this command, a graphics window should appear on the screen. (If you have previously used the command, the graphics window may be under some other windows — raise it to the top in the conventional way.)

In this window, the target point \vec{b} is drawn in red (just as in Equation 1), and similarly, the vectors \vec{u} and \vec{v} are drawn in blue and green respectively. When we start, the multipliers x_1 and x_2 are set to 1. The dashed lines

shows the directions of the vectors; the solid lines shows where we get when we x_1 steps in the first direction and x_2 steps in the second direction.

Notice that one of the solid lines is thicker than the other. This is the “active direction,” a terminology that has nothing to do with mathematics but is needed for us to play the video game.

MOVE THE MOUSE CURSOR so that it is in the window. Once you have done this, the keyboard becomes active. There are several things you can accomplish by pressing the appropriate key. But remember, keep the mouse cursor in the graphics window; if you move the cursor out of the window, the keyboard becomes inactive.

q Quit the game. At this point, the values of x_1 and x_2 will be printed in the Matlab command window — the window where you entered the `addvec` command in the first place.

→ Move further in the active direction. Mathematically, this is done by increasing the size of x_i . Notice that all the other vectors are redrawn at the same time.

← Move backwards in the active direction. More precisely, make the multiplier x_i smaller.

f Change the sign of x_i . This flips us to the other side of the origin, $(0,0)$ which is at the center of the graphics window.

n Make the next direction active.

m Make the previous direction active.

x Report the current values of the multipliers x_1 and x_2 .

b Make the region being viewed bigger, so you can see more of the space. This will make the vectors appear smaller.

s Make the region being viewed smaller.

Practice with all of these keyboard commands until you have an intuitive understanding of what each of them does.

In this video game, when we say “Solve Equation 1,” we mean, play the game, growing, shrinking, and flipping the vectors until the tip of the vector sum is on the target point **b**. At that point, the values of x_1 and x_2 are the solutions we seek.

6. Press the q key to exit the game and set yourself up for the next problem.

Now comes some work:

Solutions in \mathbb{R}^2

1 Solve Equation 1 for the values $\vec{u} = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$, $\vec{v} = \begin{pmatrix} -1 \\ 3 \end{pmatrix}$, and $\vec{b} = \begin{pmatrix} 5 \\ 9 \end{pmatrix}$. Record your values of x_1 and x_2 .

2 Solve Equation 1 as in (1), but swap the value of \vec{u} and \vec{v} .

3 Solve the equation $x_1\vec{u} = \vec{b}$ for the numerical values given in (1). An exact solution is one where you reach the target point. (In the video game, there is some slop due to the imprecision of the graphical display. We’ll disregard this.) Can you find an exact solution? If not, what is the “best” solution you can find? Record this.

4 Solve the equation $x_1\vec{u} + x_2\vec{v} + x_3\vec{w} = \vec{b}$. As numerical values, use the same ones as in (1), and set

```
>> w = [-4; 1]
```

You use `addvec` in the same way as before, but give **w** as an additional argument:

```
>> addvec(b, u, v, w)
```

Note that the solution returned by `addvec` has three components. These are x_1 , x_2 , and x_3 , the multipliers on \vec{u} , \vec{v} , and \vec{w} .

Find one solution and record it. Then find another, different solution and record that too. How many different solutions are there?

5 Write a few sentences describing in what circumstances there will be a unique, exact solution for Equation 1, when there will be many exact solutions, and when there will be no exact solutions.

6 Explain what you think the word “best” should mean when we speak of finding the best solution even though there is no exact solution.

Solutions in \mathbb{R}^3

It’s really pretty easy to find solutions in \mathbb{R}^2 ; even a child can do it. Now let’s consider the situation in \mathbb{R}^3 , when the vectors refer to directions in 3-dimensional space.

The program `addvec` works the same way with 3-dimensional vectors as with 2-dimensional vectors. To illustrate, let’s define

```
>> b = [5;4;5]
```

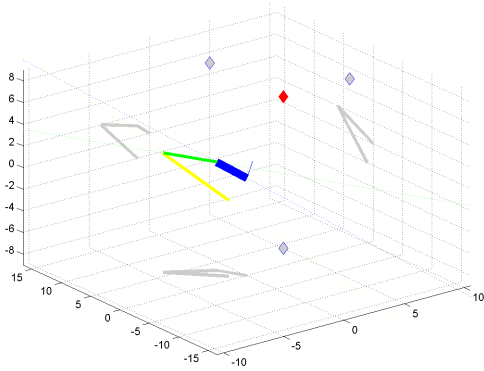
```
>> u = [-1;3;1]
```

```
>> v = [-3;3;1]
```

```
>> w = [3;-5;-4]
```

```
>> addvec(b,u,v,w)
```

The situation will now look like the figure:



The vectors are being plotted in a perspective format that mimics 3 dimensions. To help you interpret the situation, the “shadows” of the vectors and the target point are also plotted. These shadows are the projections down onto the (x, y) plane, the (y, z) plane, and the (x, z) plane.

Use the same keyboard commands to grow and shrink the coefficients x_1 , x_2 , and x_3 to bring the vector sum to the target point, that is, to solve the equations. Keep in mind that you need to reach the target point in all three dimensions at once. It’s really easy to get the tip of the vector sum over the target point, but you need to make sure to get all of the shadows at their shadow target points as well. Only then do you know that you have reached the target.

Experiment with trying to solve the matrix equations. Most people find it hard.

But, with an additional technique, finding the solution becomes easy. This technique is to pick up the displayed box and rotate it so that you are looking directly down one of the vectors. In this perspective, that vector becomes invisible. (Think of looking at a broomstick. When you move so that you are looking straight down the stick, its image becomes very small.)

We provide a keyboard command for this:

c Rotate the displayed box so that you are looking down the active direction.

Once you have done this, growing, shrinking, or flipping the active direction doesn’t change the way the position of the other vectors. Or, at least it appears that this is the case. Look at the shadows — they are moving even if the colored vectors are not.

Experiment with using the “c” command. Try to find an algorithm that allows you to solve the equations. Once you know the trick, finding the solution is very simple (although it involves keeping careful track of which vector is active.)¹

7 Solve the matrix system involving 3 vectors in \mathbb{R}^3 given above. Record your solution.

¹Here’s the trick, but read this only after you have tried to figure it out for yourself. Use “c” to look down one of the vectors, say \vec{u} . Then, **using only the other vectors**, bring the vector sum over the target point. It probably won’t match in 3 dimensions, but that’s OK. Finally, use “c” to project down onto one of the vectors that you have just adjusted and, adjusting only the length of \vec{u} , bring the vector sum over the target point. It will match in all the shadows as well — the matrix equation has been solved!

8 Add in a 4th vector — any vector you like — show that there are multiple solutions. Record the vectors and two or three of your solutions.

9 Try to solve the system using only the \vec{u} and \vec{v} vectors. Can you do it? If not, find the best solution you can and record it.

10 Let’s add in a new third vector:

```
>> wnew = 2*u + v
```

```
>> addvec(b, u, v, wnew)
```

Try to solve the system using only these three vectors. Can you do it? If not, find the best solution you can and record it.

Linear Combinations Numerically Using R

We’ll drop the video game now, and work using computer operations that accomplish the same thing. The two main R-language operators that we will use are `solve` and `%*%`. The later one is “matrix multiplication.”

Exact Solution

Suppose that we wish to solve the following equation with 3 vectors:

$$x_1 \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + x_2 \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} + x_3 \begin{pmatrix} 2 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \\ 8 \end{pmatrix}$$

or equivalently

$$\begin{pmatrix} 1 & 1 & 2 \\ 2 & 0 & 1 \\ 3 & -1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \\ 8 \end{pmatrix}.$$

We do so by typing the following commands into R

```
> A = matrix(c(1,2,3,1,0,-1,2,1,2),3,3)
> A (this shows you A to see if you typed it correctly)
> b = matrix(c(4,4,8),3,1)
> b (this shows you b to see if you typed it correctly)
```

Once the matrix and the target point are entered, the solution \vec{x} is easily found:

```
> x = solve(A,b)
> x
```

Try it! Record your answer, and check your answer by hand to see if you believe it.

Once you have checked it by hand, you can check it using the computer’s matrix multiplication operator to compute $A\vec{x}$:

```
> A %% x
```

Note that in mathematical multiplication, the multiplication operator is written implicitly by juxtaposing the two quantities to be multiplied. That the operation is matrix multiplication and not ordinary (scalar) multiplication is something that the reader learns to infer by context. On the computer, however, we need to specify explicitly that we want matrix multiplication. Just to reinforce this, try the statement for ordinary multiplication and see what happens

```
> A * x # not the right thing to do
```

Ordinary (scalar) multiplication is commutative, that is xy is the same as yx . But this isn't true of matrix multiplication; it will always be (in this class, at least) the matrix first then the vector \vec{x} .

Redundancy

Now solve the following equation in the same way:

$$\begin{pmatrix} 1 & -1 & 5 \\ 2 & 0 & 4 \\ 3 & 1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 4 \end{pmatrix}.$$

No exact solution can be found because the matrix is "singular." The problem is that the matrix doesn't have 3 independent vectors, but three vectors on the same plane. One is redundant! The third vector is 2 times the first minus 3 times the second. The first two vectors will span the same plane that all 3 vectors span.

Throw the 3rd vector away and instead solve

$$\begin{pmatrix} 1 & -1 \\ 2 & 0 \\ 3 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 4 \end{pmatrix}$$

and check by hand that your solution is correct.

Deficiency

Solve $\begin{pmatrix} 1 & -1 \\ 2 & 0 \\ 3 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$ as follows:

```
> A = matrix(c(1,2,3,-1,0,1),3,2)
> b = matrix(c(1,0,1),3,1)
> x = solve(A,b) % Read the note below!
```

Important note: Depending on what version of R you are using, you may get an error message from the above

```
> x = solve(A,b)
Error in solve.default(a, b) : A (3 x 2) must be square
```

If you do get such an error, then replace the `solve` command with this one

```
> x = lsfit(A,b,intercept=F)$coef
```

This is much more verbose and ugly, but accomplishes just what `solve` is meant to do.

Now check your answer by hand. What happened! It is not correct!

In this case, it is not possible to write \vec{b} as a linear combination of the two vectors in \mathbf{A} because \vec{b} is not in the span of \mathbf{A} . In other words, if you look at all possible linear combinations of the vectors in \mathbf{A} you get a plane in 3-space, and \vec{b} is not on that plane!

The solution that R gives is called a least-squares solution. Let's see why. First we compute $\mathbf{A}\vec{x}$ as follows

```
> p = A %% x
```

The vector \vec{p} is now equal to the matrix product $\mathbf{A}\vec{x}$, and so $\vec{p} = x_1 \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + x_2 \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$, where x_1 and x_2 are the

coordinates in \mathbf{x} that you found above. If \vec{b} had been on the plane \vec{p} would equal \vec{b} .

Let's see how far \vec{p} is from \vec{b} by computing the *residual* vector $\vec{r} = \vec{b} - \vec{p}$ as follows

```
> r = b - p
```

Now, by hand, compute the dot products $\langle \vec{r}, \vec{u} \rangle$ and $\langle \vec{r}, \vec{v} \rangle$, where \vec{u} and \vec{v} are the columns of \mathbf{A} . We find that \vec{r} is orthogonal to both \vec{u} and \vec{v} so it is orthogonal to the plane that they span.

Now, compute the length of \vec{r} . Since \vec{r} is orthogonal to the plane containing \vec{u} and \vec{v} it is the shortest residual vector of all possible vectors on the plane. Thus, \vec{p} is the closest vector to \vec{b} that is on the plane, and the distance from \vec{p} to \vec{b} is the length of \vec{r} .

11 Repeat the task of finding the solutions you found using the video game, but this time using R. Just write down the R solution next to the one you recorded from the video game. Make sure to make it clear which solution is which.

We have been treating these matrix equations very abstractly up to now. But they are widely used in practice, indeed, they are fundamental to scientific computing in all fields. These methods are used, for example, in computing the pictures found by CT (computed tomography) images, economic and statistical modelling, numerical weather prediction, searching for documents in Google. The computer methods that allow large problems, involving vectors with 100s of thousands of components rather than the 2 or 3 we have considered here, have been a triumph of numerical mathematics in the second half of the 20th century. You do not need to understand these methods, and the sophistication of tools such as `solve`, but you do need to understand the geometry of vectors and solutions.

[Added in proof: The built-in `matrix` command is a bit hard to use. We have provided you with two new functions, `vec` and `mat`, that can be made available by sourcing the file `matrix.r`. The `vec` function collects scalar arguments into a vector. The `mat` function collects vectors into a matrix. For example:

```
> source('matrix.r') % do this once in each session
> b = vec(4,4,8)
> b
      [,1]
[1,]    4
[2,]    4
```

```
[3,]    8
> A = mat( vec(1,2,3), vec(1,0,-1), vec(2,1,2))
> A
      [,1] [,2] [,3]
[1,]    1    1    2
[2,]    2    0    1
[3,]    3   -1    2
> solve(A,b)
      [,1]
[1,]    1
[2,]   -1
[3,]    2
```