tutored at Yale, and spent three years studying in Europe, after which he returned to become a professor at Yale and remained there for the rest of his life. He developed *vector analysis* as an alternative to quaternions.

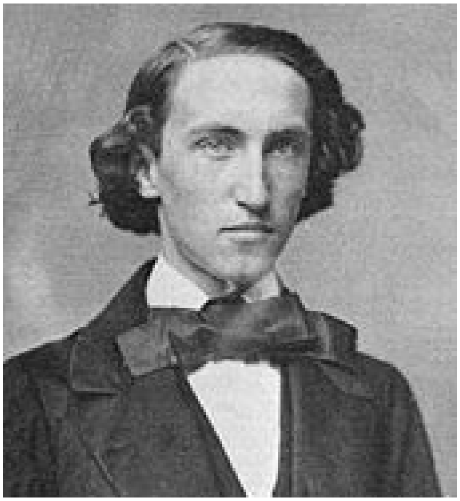


Figure 2.1: Josiah Willard Gibbs, the inventor of vector analysis

For twenty years vector analysis did not appear in published form (the primary source was unpublished notes) until Gibbs finally agreed to publish a book on the topic. It began to displace the theory of quaternions because it was more convenient to use.

However, it had the drawback of having been invented by an American. The eminent British physicist Peter Guthrie Tait, a former student of Hamilton and a partisan of quaternions, attacked mercilessly, writing, for example,

> "Professor Willard Gibbs must be ranked as one of the retarders of ... progress in virtue of his pamphlet on *Vector Analysis*; a sort of hermaphrodite monster."
>
> Tait, *Elementary Treatise on Quaternions*

Today, quaternions are still used, especially in representing rotations in three dimensions. It has its advocates in computer graphics and computer vision. However, it is safe to say that, in the end, vector analysis won out. It is used in nearly every field of science and engineering, in economics, in mathematics, and, of course, in computer science.

2.1 What is a vector?

The word *vector* comes from the Latin for "carrier". We don't plan to study pests; the term comes from a vector's propensity to move something from one location to another.

In some traditional math courses on linear algebra, we are taught to think of a vector as a list of numbers:

$$[3.14159, 2.718281828, -1.0, 2.0]$$

You need to know this way of writing a vector because it is commonly used.[1] Indeed, we will sometimes represent vectors using Python's lists.

Definition 2.1.1: A vector with four entries, each of which is a real number, is called a *4-vector over* $\mathbb{R}$.

The entries of a vector must all be drawn from a single field. As discussed in the previous chapter, three examples of fields are $\mathbb{R}$, $\mathbb{C}$, and $GF(2)$. Therefore we can have vectors over each of these fields.

Definition 2.1.2: For a field $\mathbb{F}$ and a positive integer n, a vector with n entries, each belonging to $\mathbb{F}$, is called an *n-vector over* $\mathbb{F}$. The set of n-vectors over $\mathbb{F}$ is denoted $\mathbb{F}^n$.

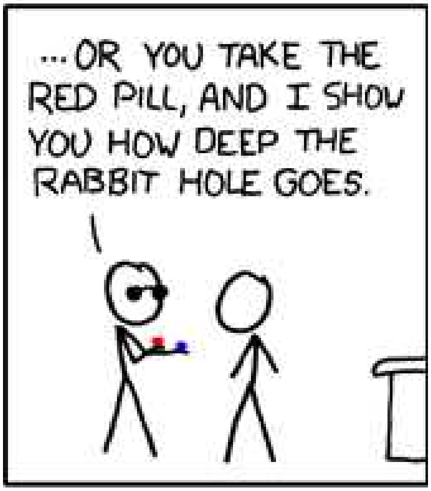
For example, the set of 4-vectors over $\mathbb{R}$ is written $\mathbb{R}^4$.

This notation might remind you of the notation $\mathbb{F}^D$ for the set of *functions* from D to $\mathbb{F}$. Indeed, I suggest you interpret $\mathbb{F}^d$ as shorthand for $\mathbb{F}^{\{0,1,2,3,\ldots,d-1\}}$ According to this interpretation, $\mathbb{F}^d$ is the set of functions from $\{0, 1, \ldots, d-1\}$ to $\mathbb{F}$.

For example, the 4-vector we started with, $[3.14159, 2.718281828, -1.0, 2.0]$, is in fact the function

$$\begin{array}{rcl} 0 & \mapsto & 3.14159 \\ 1 & \mapsto & 2.718281828 \\ 2 & \mapsto & -1.0 \\ 3 & \mapsto & 2.0 \end{array}$$

2.2 Vectors are functions


...OR YOU TAKE THE RED PILL, AND I SHOW YOU HOW DEEP THE RABBIT HOLE GOES.

[1]Often parentheses are used instead of brackets.

excerpt from *Matrix Revisited* (`http://xkcd.com/566/`)

Once we embrace this interpretation—once we accept that vectors are functions—a world of applications opens to us.

Example 2.2.1: Documents as vectors: Here's an example from a discipline called *information retrieval* that addresses the problem of finding information you want from a corpus of documents.

Much work in information retrieval has been based on an extremely simple model that disregards grammar entirely: the *word-bag* model of documents. A document is considered just a *multiset* (also called a *bag*) of words. (A multiset is like a set but can contain more than one copy of an element. The number of copies is called the *multiplicity* of the element.)

We can represent a bag of words by a function f whose domain is the set of words and whose co-domain is the set of real numbers. The image of a word is its multiplicity. Let WORDS be the set of words (e.g. English words). We write

$$f : \text{WORDS} \longrightarrow \mathbb{R}$$

to indicate that f maps from WORDS to $\mathbb{R}$.

Such a function can be interpreted as representing a vector. We would call it a *WORDS-vector over* $\mathbb{R}$.

Definition 2.2.2: For a finite set D and a field $\mathbb{F}$, a *D-vector over* $\mathbb{F}$ is a function from D to $\mathbb{F}$.

This is a computer scientist's definition; it lends itself to representation in a data structure. It differs in two important ways from a mathematician's definition.

- I require the domain D to be finite. This has important mathematical consequences: we will state theorems that would not be true if D were allowed to be infinite. There are important mathematical questions that are best modeled using functions with infinite domains, and you will encounter them if you continue in mathematics.
- The traditional, abstract approach to linear algebra does not directly define vectors at all. Just as a field is defined as a set of values with some operations (+, -, *, /) that satisfy certain algebraic laws, a vector space is defined as a set with some operations that satisfy certain algebraic laws; then vectors are the things in that set. This approach is more general but it is more abstract, hence harder for some people to grasp. If you continue in mathematics, you will become very familiar with the abstract approach.

Returning to the more concrete approach we take in this book, according to the notation from Section 0.3.3, we use $\mathbb{F}^D$ to denote the set of functions with domain D and co-domain $\mathbb{F}$, i.e. the set of all D-vectors over $\mathbb{F}$.

Example 2.2.3: To illustrate this notation for vectors as functions, consider the following:

(a.) $\mathbb{R}^{WORDS}$: The set of all $WORDS$-vectors over $\mathbb{R}$, seen in Example 2.2.1 (Page 82).

(b.) $GF(2)^{\{0,1,\ldots,n-1\}}$: The set of all n-vectors over $GF(2)$

2.2.1 Representation of vectors using Python dictionaries

We will sometimes use Python's lists to represent vectors. However, we have decreed that a vector is a function with finite domain, and Python's dictionaries are a convenient representation of functions with finite domains. Therefore we often use dictionaries in representing vectors.

For example, the 4-vector of Section 2.1 could be represented as {`0:3.14159, 1:2.718281828, 2:-1.0, 3:2.0`}.

In Example 2.2.1 (Page 82) we discussed the word-bag model of documents, in which a document is represented by a WORDS-vector over $\mathbb{R}$. We could represent such a vector as a dictionary but the dictionary would consist of perhaps two hundred thousand key-value pairs. Since a typical document uses a small subset of the words in WORDS, most of the values would be equal to zero. In information-retrieval, one typically has many documents; representing each of them by a two-hundred-thousand-element dictionary would be profligate. Instead, we adopt the convention of allowing the omission of key-value pairs whose values are zero. This is called a *sparse representation*. For example, the document "The rain in Spain falls mainly on the plain" would be represented by the dictionary

```
{'on': 1, 'Spain': 1, 'in': 1, 'plain': 1, 'the': 2, 'mainly': 1,
                                           'rain': 1, 'falls': 1}
```

There is no need to explicitly represent the fact that this vector assigns zero to `'snow'`, `'France'`, `'primarily'`, `'savannah'`, and the other elements of WORDS.

2.2.2 Sparsity

A vector most of whose values are zero is called a *sparse* vector. If no more than k of the entries are nonzero, we say the vector is *k-sparse*. A k-sparse vector can be represented using space proportional to k. Therefore, for example, when we represent a corpus of documents by WORD-vectors, the storage required is proportional to the total number of words comprising all the documents.

Vectors that represent data acquired via physical sensors (e.g. images or sound) are not likely to be sparse. In a future chapter, we will consider a computational problem in which the goal, given a vector and a parameter k, is to find the "closest" k-sparse vector. After we learn what it means for vectors to be close, it will be straightforward to solve this computational problem.

A solution to this computational problem would seem to be the key to *compressing* images and audio segments, i.e. representing them compactly so more can be stored in the same amount of computer memory. This is correct, but there is a hitch: unfortunately, the vectors representing images or sound are not even close to sparse vectors. In Section 5.2, we indicate the way around this obstacle. In Chapter 10, we explore some compression schemes based on the idea.

In Chapter 4, we introduce matrices and their representation. Because matrices are often sparse, in order to save on storage and computational time we will again use a dictionary representation in which zero values need not be represented.

However, many matrices arising in real-world problems are not sparse in the obvious sense. In Chapter 11, we investigate another form of sparsity for matrices, *low rank*. Low-rank matrices arise in analyzing data to discover factors that explain the data. We consider a computational probem in which the goal, given a matrix and a parameter k, is to find the closest matrix whose rank is at most k. We show that linear algebra provides a solution for this computational problem. It is at the heart of a widely used method called *principal component analysis*, and we will explore some of its applications.

2.3 What can we represent with vectors?

We've seen two examples of what we can represent with vectors: multisets and sets. Now I want to give some more examples.

Binary string An n-bit binary string 10111011, e.g. the secret key to a cryptosystem, can be represented by an n-vector over $GF(2)$, $[1, 0, 1, 1, 1, 0, 1, 1]$. We will see how some simple cryptographic schemes can be specified and analyzed using linear algebra.

Attributes In learning theory, we will consider data sets in which each item is represented by a collection of *attribute names* and *attribute values*. This collection is in turn represented by a function that maps attribute names to the corresponding values.

For example, perhaps the items are congresspeople. Each congressperson is represented by his or her votes on a set of bills. A single vote is represented by +1, -1, or 0 (*aye*, *nay*, or *abstain*). We will see in Lab 2.12 a method for measuring the difference between two congresspersons' voting policies.

Perhaps the items are consumers. Each consumer is represented by his or her *age*, *education level*, and *income*, e.g.

```
>>> Jane = {'age':30, 'education level':16, 'income':85000}
```

Given data on which consumers liked a particular product, one might want to come up with a function that predicted, for a new consumer vector, whether the consumer would like the product. This is an example of *machine learning*. In Lab 8.4, we will consider vectors that describe tissue samples, and use a rudimentary machine-learning technique to try to predict whether a cancer is benign or malignant.

State of a system We will also use functions/vectors to represent different states of an evolving system. The state the world might be represented, for example, by specifying the the population of each of the five most populous countries:

```
{'China':1341670000, 'India':1192570000, 'US':308745538,
 'Indonesia':237556363, 'Brazil':190732694}
```

We will see in Chapter 12 that linear algebra provides a way to analyze a system that evolves over time according to simple known rules.

Probability distribution Since a finite probability distribution is a function from a finite domain to the real numbers, e.g.

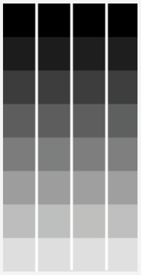
```
{1:1/6, 2:1/6, 3:1/6, 4:1/6, 5:1/6, 6:1/6}
```

it can be considered a vector. We will see in Chapter 12 that linear algebra provides a way to analyze a random process that evolves over time according to simple probabilistic rules. One such random process underlies the original definition of PageRank, the method by which Google ranks pages.

Image A black-and-white 1024×768 image can be viewed as a function from the set of pairs $\{(i, j) \ : \ 0 \leq i < 1024, 0 \leq j < 768\}$ to the real numbers, and hence as a vector. The pixel-coordinate pair (i, j) maps to a number, called the *intensity* of pixel (i, j). We will study several applications of representing images by vectors, e.g. subsampling, blurring, searching for a specified subimage, and face detection.

Example 2.3.1: As an example of a black and white image, consider an 4x8 gradient, represented as a vector in dictionary form (and as an image), where 0 is black and 255 is white:

```
{(0,0): 0,     (0,1): 0,     (0,2): 0,     (0,3): 0,
 (1,0): 32,    (1,1): 32,    (1,2): 32,    (1,3): 32,
 (2,0): 64,    (2,1): 64,    (2,2): 64,    (2,3): 64,
 (3,0): 96,    (3,1): 96,    (3,2): 96,    (3,3): 96,
 (4,0): 128,   (4,1): 128,   (4,2): 128,   (4,3): 128,
 (5,0): 160,   (5,1): 160,   (5,2): 160,   (5,3): 160,
 (6,0): 192,   (6,1): 192,   (6,2): 192,   (6,3): 192,
 (7,0): 224,   (7,1): 224,   (7,2): 224,   (7,3): 224}
```



Point in space We saw in Chapter 1 that points in the plane could be represented by complex numbers. Here and henceforth, we use *vectors* to represent points in the plane, in three dimensions, and in higher-dimensional spaces.

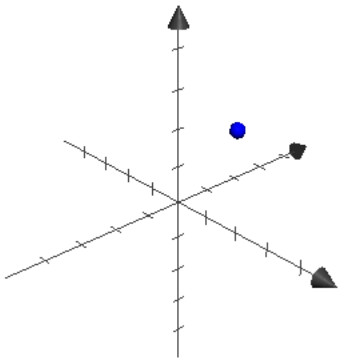
Task 2.3.2: In this task, we will represent a vector using a Python list.
In Python, assign to the variable `L` a list of 2-element lists:

```
>>> L = [[2, 2], [3, 2], [1.75, 1], [2, 1], [2.25, 1], [2.5, 1], [2.75,
  1], [3, 1], [3.25, 1]]
```

Use the `plot` module described in Task 1.4.1 to plot these 2-vectors.

```
>>> plot(L, 4)
```

Unlike complex numbers, vectors can represent points in a higher-dimensional space, e.g. a three-dimensional space:



2.4 Vector addition

We have seen examples of what vectors can represent. Now we study the operations performed with vectors. We have seen that vectors are useful for representing geometric points. The concept of a vector originated in geometry, and it is in the context of geometry that the basic vector operations are most easily motivated. We start with *vector addition*.

2.4.1 Translation and vector addition

We saw in Chapter 1 that *translation* was achieved in the complex plane by a function $f(z) = z_0 + z$ that adds a complex number z_0 to its input complex number; here we similarly achieve translation by a function $f(\boldsymbol{v}) = \boldsymbol{v}_0 + \boldsymbol{v}$ that adds a vector to its input vector.

Definition 2.4.1: Addition of n-vectors is defined in terms of addition of corresponding entries:

$$[u_1, u_2, \ldots, u_n] + [v_1, v_2, \ldots, v_n] = [u_1 + v_1, u_2 + v_2, \ldots, u_n + v_n]$$

For 2-vectors represented in Python as 2-element lists, the addition procedure is as follows:

```
def add2(v,w):
    return [v[0]+w[0], v[1]+w[1]]
```

Quiz 2.4.2: Write the translation "go east one mile and north two miles" as a function from 2-vectors to 2-vectors, using vector addition. Next, show the result of applying this function to the vectors $[4, 4]$ and $[-4, -4]$.

Answer

$f(\boldsymbol{v}) = [1,2] + \boldsymbol{v}$

$$\begin{aligned} f([4,4]) &= [5,6] \\ f([-4,-4]) &= [-3,-2] \end{aligned}$$

Since a vector such as $[1,2]$ corresponds to a translation, we can think of the vector as "carrying" something from one point to another, e.g. from $[4,4]$ to $[5,6]$ or from $[-4,-4]$ to $[-3,-2]$. This is the sense in which a vector is a carrier.

Task 2.4.3: Recall the list L defined in Task 2.3.2. Enter the procedure definition for 2-vector addition, and use a comprehension to plot the points obtained from L by adding $[1,2]$ to each:

```
>>> plot([add2(v, [1,2]) for v in L], 4)
```

Quiz 2.4.4: Suppose we represent n-vectors by n-element lists. Write a procedure addn to compute the sum of two vectors so represented.

Answer

```
def addn(v, w): return [x+y for (x,y) in zip(v,w)]
```

or

```
def addn(v, w): return [v[i]+w[i] for i in range(len(v))]
```

Every field $\mathbb{F}$ has a zero element, so the set $\mathbb{F}^D$ of D-vectors over $\mathbb{F}$ necessarily has a *zero vector*, a vector all of whose entries have value zero. I denote this vector by $\mathbf{0}_D$, or merely by $\mathbf{0}$ if it is not necessary to specify D.

The function $f(\boldsymbol{v}) = \boldsymbol{v} + \mathbf{0}$ is a translation that leaves its input unchanged.

2.4.2 Vector addition is associative and commutative

Two properties of addition in a field are *associativity*

$$(x+y)+z = x+(y+z)$$

and *commutativity*

$$x+y = y+x$$

Since vector addition is defined in terms of an associative and commutative operation, it too is associative and commutative:

Proposition 2.4.5 (Associativity and Commutativity of Vector Addition): For any vectors $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$,

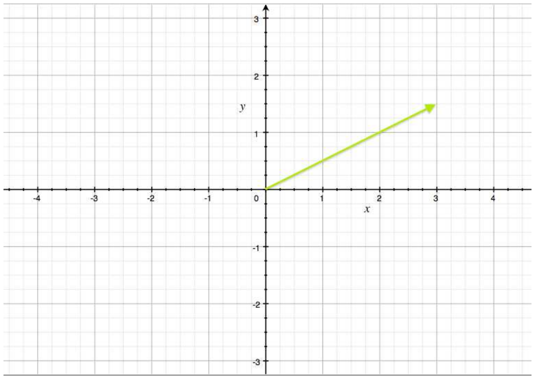
$$(\boldsymbol{u}+\boldsymbol{v})+\boldsymbol{w}=\boldsymbol{u}+(\boldsymbol{v}+\boldsymbol{w})$$

and

$$\boldsymbol{u}+\boldsymbol{v}=\boldsymbol{v}+\boldsymbol{u}$$

2.4.3 Vectors as arrows

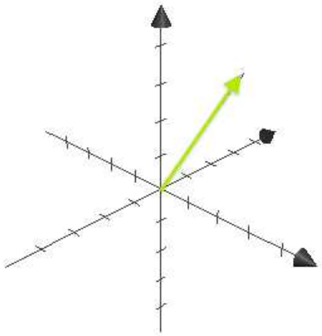
Like complex numbers in the plane, n-vectors over $\mathbb{R}$ can be visualized as arrows in $\mathbb{R}^n$. The 2-vector $[3, 1.5]$ can be represented by an arrow with its tail at the origin and its head at $(3, 1.5)$



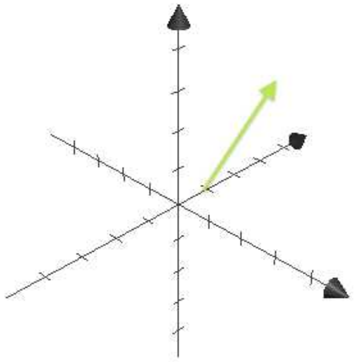
or, equivalently, by an arrow whose tail is at $(-2,-1)$ and whose head is at $(1, 0.5)$:

Exercise 2.4.6: Draw a diagram representing the vector $[-2, 4]$ using two different arrows.

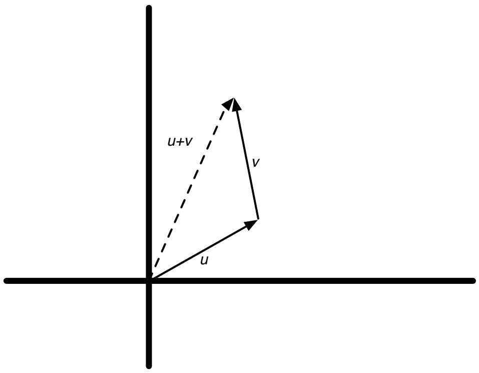
In three dimensions, for example, the vector $[1, 2, 3]$ can be represented by an arrow whose tail is at the origin and whose head is at $[1, 2, 3]$



or by an arrow whose tail is at $[0, 1, 0]$ and whose head is at $[1, 3, 3]$:



Like complex numbers, addition of vectors over $\mathbb{R}$ can be visualized using arrows. To add $\boldsymbol{u}$ and $\boldsymbol{v}$, place the tail of $\boldsymbol{v}$'s arrow on the head of $\boldsymbol{u}$'s arrow, and draw a new arrow (to represent the sum) from the tail of $\boldsymbol{u}$ to the head of $\boldsymbol{v}$.



We can interpret this diagram as follows: the translation corresponding to $\boldsymbol{u}$ can be composed with the translation corresponding to $\boldsymbol{v}$ to obtain the translation corresponding to $\boldsymbol{u}+\boldsymbol{v}$.

Exercise 2.4.7: Draw a diagram illustrating $[-2,4]+[1,2]$.

2.5 Scalar-vector multiplication

We saw in Chapter 1 that *scaling* could be represented in the complex plane by a function $f(z) = r\,z$ that multiplies its complex-number input by a positive real number r, and that multiplying by a negative number achieves a simultaneous scaling and rotation by 180 degrees. The analogous operation for vectors is called *scalar-vector multiplication.* In the context of vectors, a field element (e.g. a number) is called a *scalar* because it can be be used to scale a vector via multiplication. In this book, we typically use Greek letters (e.g. α, β, γ) to denote scalars.

Definition 2.5.1: Multiplying a vector $\boldsymbol{v}$ by a scalar α is defined as multiplying each entry of $\boldsymbol{v}$ by α:

$$\alpha\,[v_1, v_2, \ldots, v_n] = [\alpha\,v_1, \alpha\,v_2, \ldots, \alpha\,v_n]$$

Example 2.5.2: $2\,[5,4,10] = [2\cdot 5, 2\cdot 4, 2\cdot 10] = [10,8,20]$

Quiz 2.5.3: Suppose we represent n-vectors by n-element lists. Write a procedure `scalar_vector_mult(alpha, v)` that multiplies the vector `v` by the scalar `alpha`.

Answer

```
def scalar_vector_mult(alpha, v):
    return [alpha*v[i] for i in range(len(v))]
```

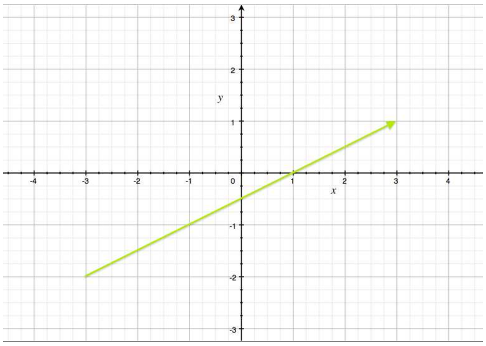
Task 2.5.4: Plot the result of scaling the vectors in `L` by 0.5, then plot the result of scaling them by -0.5.

How shall we interpret an expression such as $2\,[1,2,3]+[10,20,30]$? Do we carry out the scalar-vector multiplication first or the vector addition? Just as multiplication has precedence over addition in ordinary arithmetic, scalar-vector multiplication has precedence over vector addition. Thus (unless parentheses indicate otherwise) scalar-vector multiplication happens first, and the result of the expression above is $[2,4,6]+[10,20,30]$, which is $[12,24,36]$.

2.5.1 Scaling arrows

Scaling a vector over $\mathbb{R}$ by a positive real number changes the length of the corresponding arrow without changing its direction. For example, an arrow representing the vector $[3, 1.5]$ is this:

and an arrow representing two times this vector is this:



The vector $[3, 1.5]$ corresponds to the translation $f(\boldsymbol{v}) = [3, 1.5] + \boldsymbol{v}$, and two times this vector ($[6, 3]$) corresponds to a translation in the same direction but twice as far.

Multiplying a vector by a negative number negates all the entries. As we have seen in connection with complex numbers, this reverses the direction of the corresponding arrow. For example, negative two times $[3, 1.5]$ is $[-6, -3]$, which is represented by the arrow