Section 1.2 - Vectors

Maggie Myers
Robert A. van de Geijn

The University of Texas at Austin

Practical Linear Algebra – Fall 2009

Vectors

We will call a one-dimensional array of numbers a (real-valued) (column) vector:

$$x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix},$$

where $\chi_i \in \mathbb{R}$ for $0 \leq i < n$.

The set of all such vectors is denoted by $\mathbb{R}^n$.

A vector has a direction and a length:

- Its direction can be visualized by an arrow from the origin to the point $(\chi_0, \chi_1, \ldots, \chi_{n-1})$.
- Its length is given by the Euclidean length of this arrow: $\|x\|_2 = \sqrt{\chi_0^2 + \chi_1^2 + \cdots + \chi_{n-1}^2}$, which is often called the *two-norm*.
- There are other norms:
    - The "taxi-cab norm" or *one-norm*: $\|x\|_1 = |\chi_0| + |\chi_1| + \cdots + |\chi_{n-1}|$.
    - The "Infinity norm" or *inf-norm*: $\|x\|_\infty = \max(|\chi_0|, |\chi_1|, \cdots, |\chi_{n-1}|)$.
    - The "p-norm", for $p \geq 1$: $\|x\|_p = (|\chi_0|^p + |\chi_1|^p + \cdots + |\chi_{n-1}|^p)^{1/p}$.

Warning

When we talk about the length of a vector, sometimes we mean n, the number of components in the vector, and sometimes we mean the Euclidean length. Usually, it is clear from the context what we mean...

We will try to adhere to the following convention:

- The length of a vector will refer to the Euclidean length (two-norm).
- The size of the vector will refer to the number of components in the vector.

Exercise

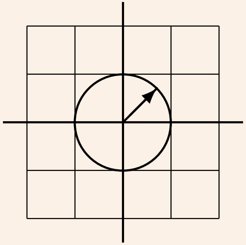
Let $x \in \mathbb{R}^2$ equal $x = \left(\begin{array}{c} 3 \\ 4 \end{array} \right)$. Draw this vector, starting at the origin. What is its length?

Example

Let S be the set of all points in $\mathbb{R}^2$ with the property that a point is in S iff (if and only if) the vector from the origin to that point has two-norm equal to one. Describe the set S.

Answer

S is the set of all point on the circle with center at the origin and radius one:



(Why?)

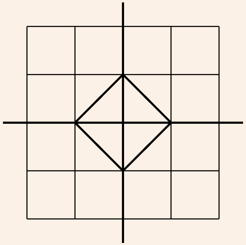
Exercise

Let S be the set of all points in $\mathbb{R}^2$ with the property that a point is in S iff the vector from the origin to that point has one-norm equal to one. Describe the set S.

Exercise

Let S be the set of all points in $\mathbb{R}^2$ with the property that a point is in S iff the vector from the origin to that point has one-norm equal to one. Describe the set S.

Answer



(Why?)

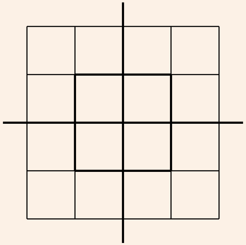
Exercise

Let S be the set of all points in $\mathbb{R}^2$ with the property that a point is in S iff the vector from the origin to that point has inf-norm equal to one. Describe the set S.

Exercise

Let S be the set of all points in $\mathbb{R}^2$ with the property that a point is in S iff the vector from the origin to that point has inf-norm equal to one. Describe the set S.

Answer



(Why?)

Operations with vectors

In our subsequent discussion, let $x, y \in \mathbb{R}^n$ $\alpha \in \mathbb{R}$, with

$$x = \left(\begin{array}{c} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{array} \right) \quad \text{and} \quad y = \left(\begin{array}{c} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{array} \right).$$

Equality

Two vectors are equal if all their components are element-wise equal: $x = y$ if and only if, for $0 \leq i < n$, $\chi_i = \psi_i$.

Copy (assignment)

The *copy* operation assigns the content of one vector to another:

$$y := x$$

(pronounce: y *becomes* x).

Copy (assignment)

The *copy* operation assigns the content of one vector to another:

$$y := x$$

(pronounce: y *becomes* x).

Copy

Algorithm	Mscript
for $i = 0, \ldots, n-1$	`for i=1:n`
$\quad\psi_i := \chi_i$	`  y(i) = x( i );`
endfor	`end`

Warning

- Mscript starts indexing at 1 (as mathematicians usually do).
- We start indexing at 0 (as computer scientists usually do).

Get used to it!

Copy as a routine

```
function [ yout ] = SLAP_Copy( x, y )

% Figure out the size (length) of vector x
[ m_x, n_x ] = size( x );
if ( m_x == 1 )
  n = n_x;
else
  n = m_x;
end

% Create an output vector of the same shape as y
yout = zeros( size( y ) );

for i=1:n
  yout( i ) = x( i );
end

return
```

Comments on copy as a routine

- This routine seems REALLY convoluted.
- The routine can also handle copying to and from (what we will later call) a row vector.
- The reason for why the routine is written in the given way will become clear later.
- YES, we realize that this exact same operation can essentially be implemented as `yout = x`.
- Trust us: things will start falling in place later.

Copy - Cost

Copying one vector to another requires $2n$ memory operations (memops):

- The vector x of length n must be read, requiring n memops.
- The vector y must be written, which accounts for the other n memops.

Note

- We realize that there is also a cost with initializing `yout` to a vector with zeros (`yout = zeros( size( y ) );`).
- This is an artifact of Mscript. In a “real” language like C, this would not be the case (if implemented appropriately).

Scaling (SCAL)

Multiplying vector x by scalar α yields a new vector, $y = \alpha x$.

- y has the same direction as x.
- y is "stretched" (scaled) by a factor α.
- Scaling a vector by α means each of its components, χ_i, is scaled by α:

$$\alpha x = \alpha \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} = \begin{pmatrix} \alpha\chi_0 \\ \alpha\chi_1 \\ \vdots \\ \alpha\chi_{n-1} \end{pmatrix}.$$

Scal

Algorithm	Mscript
for $i = 0, \ldots, n-1$	`for i=1:n`
$\quad \psi_i := \alpha \chi_i$	`  y(i) = alpha * x( i );`
endfor	`end`

Scal as a routine

```
function [ yout ] = SLAP_Scal( alpha, x, y )

% Figure out the size (length) of vector x
[ m_x, n_x ] = size( x );
if ( m_x == 1 )
  n = n_x;
else
  n = m_x;
end

% Create an output vector of the same shape as y
yout = zeros( size( y ) );

for i=1:n
  yout( i ) = alpha * x( i );
end

return
```

Comments on copy as a routine

- This routine seems REALLY convoluted (this is the last time we will say this).
- YES, we realize that this exact same operation can essentially be implemented as `yout = alpha * x`.

Scal - Cost

Scaling a vector requires $2n$ memory operations (memops) and n floating point operations (flops):

- The vector x of length n must be read, requiring n memops.
- Each element must be multiplied by α, requiring n floating point operations (flops).
- The result vector y must be written, which accounts for the other n memops.

Vector addition (ADD)

The vector addition $x + y$ is defined by

$$x + y = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} + \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix} = \begin{pmatrix} \chi_0 + \psi_0 \\ \chi_1 + \psi_1 \\ \vdots \\ \chi_{n-1} + \psi_{n-1} \end{pmatrix}.$$

Note

Clearly, (in exact arithmetic), vector addition commutes: $x + y = y + x$.

Vector addition (ADD)

Algorithm	Mscript

for $i = 0, \ldots, n-1$
    $\zeta_i := \chi_i + \psi_i$
endfor
	`for i=1:n`
	`  z(i) = x( i ) + y( i );`
	`end`

Add - Cost

Adding two vectors requires $3n$ memory operations (memops) and n floating point operations (flops):

- The vectors x and y of size n must be read, requiring $2n$ memops.
- Components must be added, requiring n flops.
- The result vector y must be written, which accounts for the other n memops.

Scaled vector addition (AXPY)

One of the most commonly encountered operations is $y := \alpha x + y$:

$$\alpha x + y = \alpha \left(\begin{array}{c} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{array} \right) + \left(\begin{array}{c} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{array} \right) = \left(\begin{array}{c} \alpha \chi_0 + \psi_0 \\ \alpha \chi_1 + \psi_1 \\ \vdots \\ \alpha \chi_{n-1} + \psi_{n-1} \end{array} \right).$$

The name, AXPY, stands for **a**lpha times **x** **p**lus **y**.

AXPY

Algorithm	Mscript
for $i = 0, \ldots, n-1$	`for i=1:n`
$\psi_i := \alpha \chi_i + \psi_i$	`y(i) = ...`
endfor	`alpha * x(i) + y(i);`
	`end`

AXPY - Cost

The cost of an AXPY is $3n$ memory operations (memops) and $2n$ floating point operations (flops):

- The vectors x and y of size n must be read, requiring $2n$ memops.
- Components of x must be multiplied by α and added to the corresponding component of y, requiring $2n$ flops.
- The result vector y must be written, which accounts for the other n memops.

Dot product (DOT)

The other commonly encountered operation is the dot (inner) product defined by

$$x^T y = \sum_{i=0}^{n-1} \chi_i \psi_i = \chi_0 \psi_0 + \chi_1 \psi_1 + \cdots + \chi_{n-1} \psi_{n-1}.$$

We have seen this before in Section 1.1...

The transition from day k to day $k+1$ is then written as the *matrix-vector product* (multiplication)

$$\begin{pmatrix} \chi_s^{(k+1)} \\ \chi_c^{(k+1)} \\ \chi_r^{(k+1)} \end{pmatrix} = \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} \chi_s^{(k)} \\ \chi_c^{(k)} \\ \chi_r^{(k)} \end{pmatrix},$$

or $x^{(k+1)} = Px^{(k)}$

This is simply a more compact representation (way of writing) the equations

$$\begin{array}{rcrcrcr} \chi_s^{(k+1)} & = & 0.4 \times \chi_s^{(k)} & + & 0.3 \times \chi_c^{(k)} & + & 0.1 \times \chi_r^{(k)} \\ \chi_c^{(k+1)} & = & 0.4 \times \chi_s^{(k)} & + & 0.3 \times \chi_c^{(k)} & + & 0.6 \times \chi_r^{(k)} \\ \chi_r^{(k+1)} & = & 0.2 \times \chi_s^{(k)} & + & 0.4 \times \chi_c^{(k)} & + & 0.3 \times \chi_r^{(k)} \end{array}$$

DOT

Algorithm	Mscript

$\alpha = 0$
for $i = 0, \ldots, n-1$
    $\alpha := \alpha + \chi_i \times \psi_i$
endfor
	`alpha = 0;`
	`for i=1:n`
	`  alpha += x(i) * y(i);`
	`end`

DOT - Cost

The cost of a DOT is (approximately) $2n$ memory operations (memops) and $2n$ floating point operations (flops):

- The vectors x and y of size n must be read, requiring $2n$ memops.
- Corresponding components of x and y must be multiplied and added to α, requiring $2n$ flops.
- The result scalar α must be written, which accounts for the other 1 memops, which can be ignored.

Exercise

Let $x, y \in \mathbb{R}^n$. Show that $x^T y = y^T x$.

Vector length (NORM2)

Recall: The Euclidean length of a vector x (the two-norm) is given by

$$\|x\|_2 = \sqrt{\sum_{i=0}^{n-1} \chi_i^2} = \sqrt{\chi_0^2 + \cdots + \chi_{n-1}^2}.$$

Clearly $\|x\|_2 = \sqrt{x^T x}$.

Cost

If computed with a dot product, it requires approximately n memops and $2n$ flops.

Note

In practice the two-norm is typically not computed via the DOT operation, since it can cause overflow or underflow in floating point arithmetic. Instead, vectors x and y are first scaled to prevent such unpleasant side effects.