Review of Linear Algebra and Overview of MATLAB

January 9, 2013

1.1 Matrices and Matrix Operations

Let A be an $m \times n$ matrix. This is the basic MATLAB datatype. The size of A can be determined in several ways:

```
[m,n]=size(A)
m=size(A,1)
n=size(A,2)
```

We can access the entry in the i^{th} row and the j^{th} column by

```
A(i,j)
```

Colon Notation

The i^{th} row and j^{th} column are obtained by

```
A(i,:)
A(:,j)
```

Matrix multiplication is defined as

$Ax = x_1v_1 + x_2v_2 + \ldots x_nv_n$ where $A = [v_1, v_2, \ldots, v_n]$ and $x = [x_1, x_2, \ldots x_n]^T$, the $n \times 1$ column matrix. The usual operator ^{T}, is the *transpose* which turns rows into columns and columns into rows. In MATLAB the transpose is `A'`. In MATLAB Ax becomes

```
A*x=x(1)*A(:,1)+x(2)*A(:,2)+...+ x(n)*A(:,n)
```

and the more general AB is $[Au_1, \ldots, Au_k]$ where $B = [u_1, \ldots, u_k]$.

If v is a vector with integer entries in the range $[1, n]$, then

```
A(:,v)
```

will produce

```
[A(:,v(1)),A(:,v(2)),...,A(:,v(k))]
```

where `k=length(v)`. Note that the order of the columns is determined by v. Using vectors v and u we can carve out submatrices of A with `A(u,v)`.

The Equation $Ax = b$

The usual solution method for the matrix equation $Ax = b$ is based on Gaussian Elimination. The MATLAB function `rref` (an acronym for reduced row echelon form) will do complete Gaussian Elimination. The solution to $Ax = b$, *when A is a square matrix and there is a unique solution* is obtained by `rref([A,b])` with the solution appearing in augmented column. MATLAB's $A\backslash b$ will also find this solution. The backslash operator is much more sophisticated than we have let on and we will return to it later. There is code for a function called `badgauss` at the end of this chapter which will use Gaussian elimination to find an upper triangular form.

The *identity matrix* is usually written I_n. In MATLAB this is `eye(n)`. An $n \times n$ matrix A is *invertible* if there is another $n \times n$ matrix B where

$$BA = AB = I_n$$

In MATLAB the inverse is called from either `inv(A)` or $A\hat{-1})$. Numerically it is both faster and more accurate to solve $Ax = b$ with $A\backslash b$ rather than $A^{-1}b$.

In MATLAB the single use of the equality sign `t = e` makes an assignment of expression `e`, the value on the right hand side, to the variable name `t` on the left hand side. It does not matter if the value on the right hand side is a scalar, a vector or a matrix. Thus, the elementary row operations are easily produced in MATLAB.

Type 2, multiply a row by a scalar $r \neq 0$

```
A(i,:)=r*A(i,:)
```

Type 3, add r times the i^{th} row to the j^{th} row

```
A(j,:)=r*A(i,:)+A(j,:)
```

Type 1, switch two rows

```
A([i,j],:)=A([j,i],:)
```

This latter is a MATLAB trick, since as we have seen above, the order of the entries in the vectors determines the order that the rows or columns are returned.

An *elementary matrix* is obtained by applying an elementary row operation to the identity matrix. By multiplying a matrix on the left by an elementary matrix, the outcome is the same as performing an elementary row operation to the matrix. Thus Gaussian Elimination yields

$\mathtt{rref}(A) = E_k \dots E_1 A$

Since each of the elementary matrices is invertible, so is the product $B = E_k \dots E_1$ and if $\mathtt{rref}(A) = I_n$, then B is the inverse of A. The simple way to find B is

$\mathtt{rref}([A, I_n) = [R, B]$

where $R = I_n$ and $B = A^{-1}$. when A is invertible. The MATLAB function `inv(A)` will compute the inverse of `A` when it exists.

The preferred method (and we will discuss later why it is preferred) for solving $Ax = b$ is to reduce $[A, b]$ to upper triangular form and then find the solution using *backward substitution.* The code for `backsub` is listed at the end of this chapter.

1.2 Vectors and Subspaces

If $v_1, \dots, v_n$ are vectors, then a *linear combination* is any vector of the form

$$v = \alpha_1 v_1 + \dots + \alpha_n v_n$$

The span of $v_1, \dots, v_n$ written $\mathrm{span}(v_1, \dots, v_n)$ is the set of all linear combinations of $v_1, \dots, v_n$. The *Column Space of* A is the span of the columns of the matrix A. Equivalently, $\mathrm{Col}(A) = \{Ax \ : \ x \in R^n \ \}$ by virtue of the definition given above for matrix multiplication. *The Null Space,* written $\mathrm{Nul}(A)$, and defined by

$$\mathrm{Nul}(A) = \{x \in R^n \ : \ Ax = \vec{0} \ \}$$

The vectors $v_1, \dots, v_n$ are *linearly independent* if $\alpha_1 v_1 + \dots + \alpha_n v_n = \vec{0}$ implies $\alpha_1 = \alpha_2 = \dots = \alpha_n = 0$. This is equivalent to saying that

$$\mathtt{rref}(A) = \begin{pmatrix} I_n \\ 0 \end{pmatrix}$$

where $A = [v_1, \dots, v_n]$ and the 0 indicates the possibility of rows of zeros.

A *basis* for a subspace $\mathcal{S}$ is a collection of vectors $v_1, \dots, v_n \in \mathcal{S}$ where

(1) $v_1, \dots, v_n$ are linearly independent

(2) $\mathrm{span}(v_1, \dots, v_n) = \mathcal{S}$

The *dimension of* $\mathcal{S}$ written $\dim(\mathcal{S})$ is the number of vectors in any basis. This is properly defined as all bases have the same number of elements. A basis for $\mathrm{Col}(A)$ is given by the columns of A which correspond to the leading ones of $\mathrm{rref}(A)$. Thus, the $\mathrm{rank}(A)$ is $\dim(\mathrm{Col}(A))$. There is a MATLAB function `rank` which will compute the rank. This turns out to be the number of nonzero rows in $\mathtt{rref}(A)$ and it is the same as the maximal number of linearly independent columns of A.

A basis for the Null Space is given by Null Space Algorithm which produces a set of linearly independent vectors which correspond to the non-leading columns of $\mathrm{rref}(A)$. If n is the number of columns of A then the number of non-leading columns is $n - r$ where $r = \mathrm{rank(A)}$. Thus,

$$\dim(\mathrm{Nul}(A)) = n - \mathrm{rank}(A))$$

The MATLAB command `null(A)` or `null(A,'r')` will find a basis for $\mathrm{Nul}(A)$).

1.3 Orthogonality

If u and v are vectors of the same length then the dot product

$$u \cdot v = u_1 v_1 + u_2 v_2 + \ldots + u_n v_n$$

which in MATLAB is just `u'*v` or `v'*u` assuming that u and v are $n \times 1$ column vectors. We say that u and v are perpendicular if $u \cdot v = 0$. If $\mathcal{S}$ is a subspace, we define the *orthogonal complement of* $\mathcal{S}$ or the *perp of* $\mathcal{S}$ to be

$$\mathcal{S}^{\perp} = \{x \in R^n \ : \ x \cdot y = 0 \text{ for all } y \in \mathcal{S} \ \}$$

There are simple duality formulas for the perps of our favorite subspaces

$$\text{Col}(A)^{\perp} = \text{Nul}(A^T)$$
$$\text{Nul}(A)^{\perp} = \text{Col}(A^T)$$

Projections

Let b be any vector. The *projection* of b into the Column Space of A is defined to be the unique $p \in \mathcal{S}$ for which there is a $q \in \mathcal{S}^{\perp}$, with $p + q = b$. Now suppose that $\mathcal{S} = \text{Col}(A)$ and define

$$p = A(A^T A)^{-1} A^T b$$

The inverse in this formula exists provided $\text{rank}(A) = n$. It is clear that $p \in \mathcal{S}$ and that $p + q = b$ if we define $q = b - p$. Thus we will show that $q \in \mathcal{S}^{\perp}$. Since $\text{Col}(A)^{\perp} = \text{Nul}(A^T)$, it suffices to show that $A^T q = \vec{0}$.

$$A^T q = A^T(b - p) = A^T b - A^T p = A^T b - A^T A (A^T A)^{-1} A^T b = A^T b - A^T b = \vec{0}.$$

In MATLAB this is easily computed with either `A*(inv(A'*A)*(A'*b))` or `A*((A'*A)'\(A'*b))`.

Orthogonal Matrices

A collection of vectors $u_1, \ldots, u_n \in R^m$ is *orthonormal* if $u_i \cdot u_j = 0$ when $i \neq j$ and $u_i \cdot u_i = 1$ for all i. A matrix U is *orthogonal* if the columns of U are orthonormal vectors. It is easily checked that $U^T U = I_n$. Since $Ux = \vec{0}$ has a unique solution, (note that $U^T U x = x = \vec{0}$) the columns of U are linearly independent. Now if we substitute U for A in the projection formula we get

$$p = U(U^T U)^{-1} U^T b = U U^T b$$

Thus, UU^T is the projection matrix into $\text{Col}(U)$. If U is a square matrix, then the columns of U form a basis for R^n and so $U^T U = I_n$ making U invertible with $U^T = U^{-1}$.

The *norm* or *2-norm* (there are other norms which will be defined later) of a vector $x = [x_1, \ldots, x_n]^T$ is

$$\|x\|_2 = norm(x) = \sqrt{x_1^2 + x_2^2 + \ldots x_n^2}$$

Simple algebra shows that if $u \perp v$, then

$$\|u + v\|^2 = \|u\|^2 + \|v\|^2$$

or

$$\text{norm}(u + v) = \text{norm}(u) + \text{norm}(v)$$

If U is orthogonal, then notice the following:
(1) $Ux \cdot Ux = x \cdot x$
  since $Ux \cdot Ux = x^T U^T U x = x^T x = x \cdot x$.
(2) $\|Ux\| = \|x\|$
  since $\|Ux\| = \sqrt{Ux \cdot Ux} = \sqrt{x \cdot x} = \|x\|$.

Least Squares

The *least squares solution* to $Ax = b$ is the vector x_{LS} such that norm$(Ax_{LS} - b)$ is minimal. This norm is called the *residual* and the vector $Ax_{LS} - b$ is called the *residual vector.* If $p = Ax_{LS}$ is the projection into the column space, then for any x, $Ax - p$ and $p - b$ are perpendicular, so

norm$(Ax - b)$ =norm$(Ax - p + p - b)$ =norm$(Ax - p)$+norm$(p - b)$ >norm$(Ax - p)$

and so norm$(Ax - p)$ is minimal. Thus, we can choose $x_{LS} = (A^T A)^{-1} A^T$ as the least squares solution. The MATLAB backslash operator will compute the least squares solution automatically as `A\b.` Suppose, for example, we knew that $U^T AU = D$ where U is orthogonal and D is diagonal, then $\|Ax - b\| = \|D - UbU^T\|$ and we could solve the least squares problem with a diagonal matrix D.

Gram-Schmidt Orthogonalization

Given a linearly independent collection of vectors $v_1, \ldots, v_n$ we can produce an orthonormal collection $u_1, \ldots, u_n$ where span$(v_1, \ldots, v_n) = span(u_1, \ldots, u_n)$. This is an iterative process as follows:

Step 1: $u_1 = v_1/\text{norm}(v_1)$ and $U = [u_1]$.

Step k+1: $w = v_{k+1} - UU^T v_{k+1}$, $u_{k+1} = w/\text{norm}(w)$ and $U = [U, u_{k+1}]$.

The MATLAB code for this is given in Section 1.5. The MATLAB function `orth(A)` will find an orthonormal basis for the column space of `A.` Similary, `null(A)` will find an orthonormal basis for the null space of `A.`

1.4 Sample MATLAB Programs.

Gaussian Elimination

The MATLAB function `badgauss` is a simplistic code for Gaussian Elimination. It is included here as an example of a MATLAB function but also as a starting point for future codes which build on Gaussian Elimination. A is assumed to be a $m \times n$ matrix where $n \geq m$. This code is fatally missing a check for division by 0.

```
function B=badgauss(A)
m=size(A,1);
B=A;
for i=1:m-1
    for j=i+1:m
        r=B(j,i)/B(i,i);
        B(j,:)=B(j,:)-r*B(i,:);
    end
end
```

Backward Substitution

Here is a program for backward substitution. It is not the most efficient, but we will work on that later.

```
function x=backsub(A,b)
n=size(A,2);
x=zeros(n,1);
for i=n:-1:1,
    x(i)=(b(i)-A(i,:)*x)/A(i,i);
end
```

The Gram-Schmidt Orthogonalization Process

```
function B=grmsch(A)
n=size(A,2);
w=A(:,1);
B=w/norm(w);
for i=2:n
    w=A(:,i);
    w=w-B*(B'*w);
    B=[B,w/norm(w)];
end
```