A TOUR OF LINEAR ALGEBRA FOR JDEP 384H

Contents

Solving Systems

Solving linear systems is a fundamental activity, practiced from high school forward. Recall that a linear equation is one of the form

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b,$$

where $x_1, x_2, \ldots, x_n$ are the variables and $a_1, a_2, \ldots, a_n, b$ are known constants. A linear system is a collection of linear equations, and a solution to the linear system is a choice of the variables that satisfies every equation in the system. What can we expect?

Example 6. Use geometry to answer the question with one or more equations in 2 unknowns. Ditto 3 unknowns.

Basic Fact (Number of Solutions): A linear system of m equations in n unknowns is either inconsistent (no solutions) or consistent, in which case it either has a unique solution or infinitely many solutions. In the former case, unknowns cannot exceed equations. If the number of equations exceeds the number of unknowns, the system is called **overdetermined**. If the number of unknowns exceeds the number of equations, the system is called **underdetermined**.

While we're on this topic, let's review some basic linear algebra as well:

The general form for a linear system of m equations in the n unknowns $x_1, x_2, \ldots, x_n$ is

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1j}x_j + \cdots a_{1n}x_n & = & b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2j}x_j + \cdots a_{2n}x_n & = & b_2 \\ \vdots & \vdots & \vdots \\ a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{ij}x_j + \cdots a_{in}x_n & = & b_i \\ \vdots & \vdots & \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mj}x_j + \cdots a_{mn}x_n & = & b_m \end{array}$$

Notice how the coefficients are indexed: in the ith row the coefficient of the jth variable, x_j, is the number a_{ij}, and the right hand side of the ith equation is b_i.

The statement "$A = [a_{ij}]$" means that A is a **matrix** (rectangular table of numbers) whose (i, j)th entry, i.e., entry in the ith row and jth column, is denoted by a_{ij}. Generally, the size of A will be clear from context. If we want to indicate that A is an $m \times n$ matrix, we write

$$A = [a_{ij}]_{m,n}.$$

Similarly, the statement "$\mathbf{b} = [b_i]$" means that b is a **column vector** (matrix with exactly one column) whose ith entry is denoted by b_i, and "$\mathbf{c} = [c_j]$" means that c is a **row vector** (matrix with exactly one row) whose jth entry is denoted by c_j. In case the type of the vector (row or column) is not clear from context, the default is a column vector.

How can we describe the matrices of the general linear system described above? First, there is the $m \times n$ **coefficient matrix**

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mj} & \cdots & a_{mn} \end{bmatrix}$$

Notice that the way we subscripted entries of this matrix is really very descriptive: the first index indicates the row position of the entry and the second, the column position of the entry. Next, there is the $m \times 1$

right hand side vector of constants

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_i \\ \vdots \\ b_m \end{bmatrix}$$

Finally, stack this matrix and vector along side each other (we use a vertical bar below to separate the two symbols) to obtain the $m \times (n+1)$ **augmented matrix**

$$\widetilde{A} = [A \mid \mathbf{b}] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & & \vdots & & \vdots & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{in} & b_i \\ \vdots & \vdots & & \vdots & & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mj} & \cdots & a_{mn} & b_m \end{bmatrix}$$

Matrix Arithmetic

Given any two matrices (or vectors) $A = [a_{ij}]$ and $B = [b_{ij}]$ of the same size, we may add them or multiply by a scalar according to the rules

$$A + B = [a_{ij} + b_{ij}]$$

and

$$cA = [ca_{ij}] \,.$$

For each size $m \times n$ we can form a **zero matrix** of size $m \times n$: just use zero for every entry. This is a handy matrix for matrix arithmetic. Matlab knows all about these arithmetic operations, as we saw in our Matlab introduction. Moreover, there is a multiplication of matrices A and B provided that A is $m \times p$ and B is $p \times n$. The result is an $m \times n$ matrix given by the formula

$$AB = \left[\sum_{k=1}^{p} a_{ik} b_{kj} \right] .$$

Again, Matlab knows all about matrix multiplication.

The Basic Rules of Matrix Arithmetic. There are many algebra rules for matrix arithmetic. We won't review all of them here, but one noteworthy fact is that matrix multiplication is not commutative: $AB \neq BA$ in general. Another is that there is no general cancellation of matrices: if $AB = 0$. Another is that there is an **identity matrix** for any square size $n \times n$: I_n is the matrix with ones down the diagonal entries and zeros elsewhere. For example,

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

We normally just write I and n should be clear from context. Algebra fact: if A is $m \times n$ and B is $n \times m$, then $AI = A$ and $IB = B$.

One of the main virtues of matrix multiplication is that is gives us a symbolic way of representing the general linear system given above, namely, we can put it in the simple looking form

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

where A and b are given as above, namely

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

where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix}$$

is the (column) vector of unknowns. Actually, this is putting the cart before the horse because this idea actually inspires the whole notion of matrix multiplication. This form suggests a way of thinking about the solution to the system, namely,

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

The only problem is that this doesn't quite make sense in the matrix world. But we can make it sensible by a little matrix algebra: imagine that there were a multiplicative inverse matrix A^{-1} for A, that is, a matrix such that

$$A^{-1}A = I = AA^{-1}.$$

Then we could multiply both sides of $Ax = b$ on the left by A^{-1} and obtain that

$$\mathbf{x} = A^{-1}\mathbf{b}.$$

Matlab represents this equation by

```
x = A\b
```

In general, a square matrix A may not have an inverse. One condition for an inverse to exist is that the matrix have nonzero determinant, where this is a number associated with a matrix such as one saw in high school algebra with Cramer's rule. We won't go into details here. Of course, Matlab knows all about inverses and we can compute the inverse of A by issuing the commands A^(-1) or inv(A). In fact, Matlab even knows many more subtle matrix constructions. For example, try evaluating `B= A^(1/2)` and follow it with `B*B`.

Sets of *all* matrices or vectors of a given size form "number systems" with a scalar multiplication and addition operation that satisfies a list of basic laws. Such a system is a **vector space.** For example, for vectors from $\mathbb{R}^3$, we have operations

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \\ a_3 + b_3 \end{bmatrix}$$

$$c \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} ca_1 \\ ca_2 \\ ca_3 \end{bmatrix}$$

and similar operations for n-vectors of the vector space $\mathbb{R}^n$ or the vector space of $n \times n$ matrices $\mathbb{R}^{n\times n}$.

Norms and Dot Products

Norms. Before we get started, there are some useful ideas that we need to explore. We already know how to measure the size of a scalar (number) quantity x: use $|x|$ as a measure of its size. Thus, we have a way of thinking about things like the *size* of an error. Now suppose we are dealing with vectors and matrices.

Question: How do we measure the size of more complex quantities like vectors or matrices?

Answer: We use some kind of yardstick called a **norm** that assigns to each vector $\mathbf{x}$ a non-negative number $\|\mathbf{x}\|$ subject to the following norm laws for arbitrary vectors $\mathbf{x}, \mathbf{y}$ and scalar c:

- For $\mathbf{x} \neq \mathbf{0}$, $\|\mathbf{x}\| > 0$ and for $\mathbf{x} = \mathbf{0}$, $\|\mathbf{x}\| = 0$.
- $\|c\mathbf{x}\| = |c| \, \|\mathbf{x}\|$.
- $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$.

Examples: For vector space $\mathbb{R}^n$ and vector $\mathbf{x} = [x_1, x_2; \ldots; x_n]$,

- 1-norm: $\|\mathbf{x}\|_1 = |x_1| + |x_2| + \cdots + |x_n|$
- 2-norm (the default norm):

$$\|\mathbf{x}\|_2 = \left(x_1^2 + x_2^2 + \cdots + x_n^2\right)^{1/2}$$

- ∞-norm:$\|\mathbf{x}\|_\infty = \max\left(|x_1|, |x_2|, \cdots, |x_n|\right)$

Let's do some examples in Matlab, which knows all of these norms:

```
  >x=[1;-3;2;-1], y=[2;0;-1;2]
>norm(x)
>norm(x,1)
>norm(x,2)
>norm(x,inf)
>norm(-2*x),abs(-2)*norm(x)
>norm(x+y),norm(x)+norm(y)
```

Matrix Norms: For every vector norm $\|\mathbf{x}\|$ of n-vectors there is a corresponding induced norm on the vector space of $n \times n$ matrices A by the formula

$$\|A\| = \max_{\mathbf{x} \neq \mathbf{0}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\|.$$

These induced norms satisfy all the basic norm laws and, in addition:

$$\begin{aligned} \|A\mathbf{x}\| &\leq \|A\| \, \|\mathbf{x}\| \text{ (compatible norms)} \\ \|AB\| &\leq \|A\| \, \|B\| \text{ (multiplicative norm)} \end{aligned}$$

One can show directly that if $A = [a_{ij}]$, then

- $\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |a_{ij}|$
- $\|A\|_2 = \sqrt{\rho\left(A^T A\right)}$, where $\rho(B)$ is largest eigenvalue of square matrix B in absolute value.
- $\|A\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}|$

Dot Products. The 2-norm is special. It can be derived from another operation on vectors called the dot product. Here is how it works.

Definition. The **dot product** of vectors $\mathbf{u}, \mathbf{v}$ in $\mathbb{R}^n$ is the scalar

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T \mathbf{v}.$$

This operation satisfies a long list of properties, called the inner product properties:

- $\mathbf{u} \cdot \mathbf{u} \geq 0$ with equality if and only if $\mathbf{u} = \mathbf{0}$.

- $\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{u}$
- $\mathbf{u} \cdot (\mathbf{v} + \mathbf{w}) = \mathbf{u} \cdot \mathbf{v} + \mathbf{u} \cdot \mathbf{w}$
- $c(\mathbf{u} \cdot \mathbf{v}) = (c\mathbf{u}) \cdot \mathbf{v}$

What really makes this idea important are the following facts. Here we understand that $\|\mathbf{u}\|$ means the 2-norm $\|\mathbf{u}\|_2$.

- $$\|\mathbf{u}\| = \sqrt{\mathbf{u} \cdot \mathbf{u}}$$
- If θ is an angle between the vectors represented in suitable coordinate space, then
$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \, \|\mathbf{u}\| \cos \theta.$$
- Once we have the idea of angle between vectors, we can speak of **orthogonal** (perpendicular) vectors, i.e., vectors for which the angle between them is $\pi/2$ ($90°$).
- Given any two vectors $\mathbf{u}, \mathbf{v}$, with $\mathbf{v} \neq \mathbf{0}$, the formula
$$\operatorname{proj}_{\mathbf{v}} \mathbf{u} = \frac{\mathbf{u} \cdot \mathbf{v}}{\mathbf{v} \cdot \mathbf{v}} \mathbf{v}$$
defines a vector parallel to $\mathbf{v}$ such that $\mathbf{u} - \operatorname{proj}_{\mathbf{v}} \mathbf{u}$ is orthogonal to $\mathbf{v}$.

Linear Programming

Here we understand that a vector inequality like $\mathbf{x} \geq \mathbf{0}$ is satisfied if the inequality is satisfied in each coordinate.

With this notation we can express a general linear programming problem as follows:

(LP) Find a value of $\mathbf{x}$ that minimizes the linear function $f(\mathbf{x}) = \mathbf{c}^T \mathbf{x}$ subject to conditions that $A\mathbf{x} \geq \mathbf{b}$, $\mathbf{B}x = \mathbf{c}$ and $\mathbf{x} \geq \mathbf{0}$.

A somewhat restricted form is a linear programming problem in **canonical form**:

(CLP) Find a value of $\mathbf{x}$ that minimizes the linear function $f(\mathbf{x}) = \mathbf{c}^T \mathbf{x}$ subject to conditions that $A\mathbf{x} \geq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$.

Notes:

(1) The problem (CLP) is an optimization problem because its objective is the find an extremum (maximum or minimum) of a scalar function.
(2) The problem is in **canonical form** if it is expressed as above or is a maximization problem or has inequalities the other way: $A\mathbf{x} \leq \mathbf{b}$.
(3) The function $f(\mathbf{x}) = \mathbf{c}^T \mathbf{x}$ is called the **objective function.**
(4) The variables that occur in $f(\mathbf{x})$, i.e., the coordinates of $\mathbf{x}$, are called **decision variables**.

(5) The set of all vectors $\mathbf{x}$ that satisfy the constraints is called the **feasible set**.

A linear programming problem is said to be in **standard form** if the only inequalities are simple ones of the form $\mathbf{x} \geq \mathbf{0}$ and all other constraints on the variables are linear equalities. So such a linear programming problem takes the form

(SLP) Find a value of $\mathbf{x}$ that minimizes the linear function $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x}$ subject to conditions that $A\mathbf{x} = \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$.

In a sense all the problems thus far expressed are equivalent because one form can be put into another with the following tricks:

(1) A linear inequality like $2x_1+x_2 \leq 4$ is turned into an equality by adding in a nonnegative *"slack"* variable x_3 to get $2x_1+x_2+x_3 = 4$ at the price of adding one new variable.
(2) An inequality going the "wrong" way like $2x_1+x_2 \geq 4$ is straightened out by multiplying by -1 to obtain $-2x_1-x_2 \leq -4$. Now add in a slack variable as in 1.
(3) Another way to handle a "wrong" way linear inequality is to subtract a nonnegative *"surplus"* variable x_3 to obtain the equality $2x_1 + x_2 - x_3 = 4$.
(4) An equality like $x_1 + x_2 = 3$ is turned into an inequality by replacing the equation by two inequalities $x_1 + x_2 \leq 3$ and $-x_1 - x_2 \leq -3$.
(5) A maximization problem like maximizing $0.06x_1 + 0.08x_2$ is equivalent to minimizing the negative of the objective function, i.e., minimize $-0.06x_1 - 0.08x_2$.

In general, it is not clear that a linear programming problem even has a solution. If there is a solution, then there is one that occurs on the boundary of the feasible set and more specifically, at a "corner" (called an **extreme point**) of the feasible set. Such a solution is called a **basic feasible solution**. The three possible outcomes are the following:

(1) The feasible set is empty. In this case there is no solution to the linear programming problem.
(2) There are feasible points, but no optimum for the objective function. If this happens, the feasible set is unbounded and we can make the objective function arbitrarily large (in the case of searching for a maximum) by moving along an unbounded line in the feasible set.
(3) The problem has a solution. This always happens when the feasible set is nonempty and bounded, but it may also happen if the feasible set is unbounded. However, there are exactly two possibilities: a unique solution or infinitely many solutions.

In the latter case, every solution can be expressed as a linear combination of basic feasible solutions.

Generally, the solution of (LP) or discovery there is none follows two steps if the so-called **simplex method** is used:

- Phase I: find a basic feasible solution. If none is found, conclude the feasible set is empty and there is no solution.
- Phase II: find an optimal basic feasible solution by moving from one to another that reduces the objective function, or conclude that function values are unbounded and there is no solution.

Other methods for solving (LP) problems that have gained popularity in recent years are the so-called **interior point methods**. They involve ideas from non-linear optimization and are rather more sophisticated than the simplex method. In contrast, the simplex method is simple to understand and has stood the test of time fairly well. It is still the method of choice for small to medium sized linear programming problems.

Eigenvectors and Eigenvalues

An **eigenvalue** for a square $n \times n$ matrix A is a number λ such that for some NONZERO vector $\mathbf{x}$, called an **eigenvector** for λ,

$$A\mathbf{x} = \lambda\mathbf{x}.$$

Eigenvalue Facts:

(1) The eigenvalues of A are precisely the roots of the nth degree polynomial
$$p(\lambda) = |\lambda I - A|,$$
where $||$ denotes determinant.

(2) An $n \times n$ matrix A has exactly n eigenvalues, counting multiplicities and complex numbers.

(3) The eigenvalues of $\alpha I + \beta A$ are just $\alpha + \beta\lambda$, where λ runs over the eigenvalues of A.

(4) More generally, if $r(\lambda) = p(\lambda)/q(\lambda)$ is any rational function, i.e., p, q are polynomials, then the eigenvalues of $r(A)$ are $r(\lambda)$, where λ runs over the eigenvalues of A, provided both expressions make sense.

(5) The **spectral radius** of A is just
$$\rho(A) = \max\{|\lambda| \mid \lambda \text{ is an eigenvalue of } A\}$$

(6) Suppose that $\mathbf{x}_0$ is an arbitrary initial vector, $\{\mathbf{b}_k\}$ is an arbitrary sequence of uniformly bounded vectors, and the sequence

$\{\mathbf{x}_k\}$, $k = 0, 1, 2, \ldots$, is given by the formula

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + \mathbf{b}_k.$$

If $\rho(A) < 1$, then $\{\mathbf{x}_k\}$ is a uniformly bounded sequence of vectors. Such a matrix A is called a **stable** matrix.

(7) If $\rho(A) > 1$, then $\{\mathbf{x}_k\}$ will not be uniformly bounded for some choices of $\mathbf{x}_0$.

(8) **(Principal Axes Theorem)** If A is a (real) symmetric matrix, then there exists an orthogonal matrix P (this means that $P^{-1} = P^T$) such that $P^{-1}AP = D$, a diagonal matrix with real entries. Specifically, the diagonal entries of D are the eigenvalues of A and the columns of P form an orthogonal set of unit-length eigenvectors of A.

(9) **(Spectral Decomposition Theorem)** If A is a (real) symmetric matrix, with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ and corresponding unit-length orthogonal eigenvectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$, then

$$A = \lambda_1 \mathbf{e}_1 \mathbf{e}_1^T + \lambda_2 \mathbf{e}_2 \mathbf{e}_2^T + \cdots + \lambda_n \mathbf{e}_n \mathbf{e}_n^T$$

.

Let's do a few calculations with matrices to illustrate the idea of eigenvalues and eigenvectors in Matlab, which knows all about these quantities.

```
 > A = [3 1 0;-1 3 1;0 3 -2]
> eig(A)
> eig(A)
> [V,D]=eig(A)
> v = V(:,1),lam = D(1,1)
> A*v,lam*v
> x = [1;-2;3]
> b = [2;1;0]
> x = A*x+b % repeat this line
```

Now let's demonstrate boundedness of an arbitrary sequence as above. Try the same with a suitable A.

```
 > A = A/4
> max(abs(eig(A)))
> x = [1;-2;3]
> x = A*x+b % repeat this line
```

Finally, let's find the orthogonal diagonalization guaranteed by the principal axes theorem for a symmetric matrix.

```
 > A = A'*A % need a symmetric matrix..this one's SPD
> [P,D] = eig(A)
> P' - P^(-1) % test for P'=P^(-1)
> e = P(:,1)
> % confirm the spectral decomposition theorem
> lambda1 = D(1,1),lambda2 = D(2,2),lambda3 = D(3,3)
> e1 = P(:,1),e2 = P(:,2),e3 = P(:,3)
> lambda1*e1*e1'+lambda2*e2*e2'+lambda3*e3*e3' - A
```