CS412: INTRODUCTION TO NUMERICAL ANALYSIS 10/26/10

Lecture 14: Linear Algebra

Instructor: Professor Amos Ron Scribes: Mark Cowlishaw, Nathanael Fillmore

1 Introduction

In the last few classes, we have discussed several methods for approximating the behavior of a function f over an interval $[a, b]$. We have seen several methods for approximating f using the idea of interpolation, including polynomial interpolation, cubic spline interpolation, and cubic Hermite spline interpolation. As we have seen, interpolation by cubic splines and cubic Hermite splines can be both fast and accurate. However, in practice, interpolation is not always used for approximating functions. Are there reasons why interpolation might be a bad idea for certain function approximation problems? The answer is yes, there are two main reasons why interpolation might be inappropriate for a particular function approximation problem:

- There may be too many data points. As we have seen, the cost of interpolation increases with the number of interpolation points. If we have data with thousands or millions of points, the cost of interpolation may be prohibitive.
- The function values may only be approximate. In scientific applications, "noisy" data values are quite common. If data values are inexact, then matching all data points exactly makes our approximation too complex and too sensitive to error. Instead, we would like to choose an approximation from a set of simpler functions, as shown in Figure 1

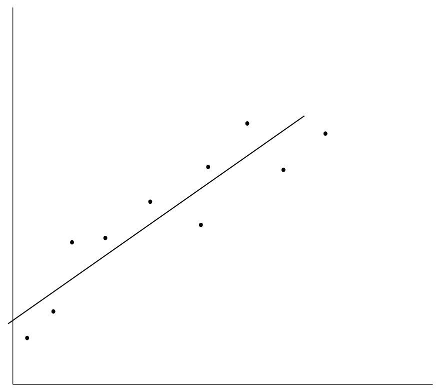

Figure 1: Example of Function Approximation using Noisy Data Values

Information theory tells us that if we are approximating a function using noisy data values, we would like to use the simplest solution (in this case meaning the lowest degree polynomial) that fits the data values. In practice, this means that we first decide the degree d of the polynomial we would like to use to fit the data, then we find the specific degree d polynomial that minimizes the error over the set of data points. In this case, *error* is some measure of the distance between our representation and the data points.

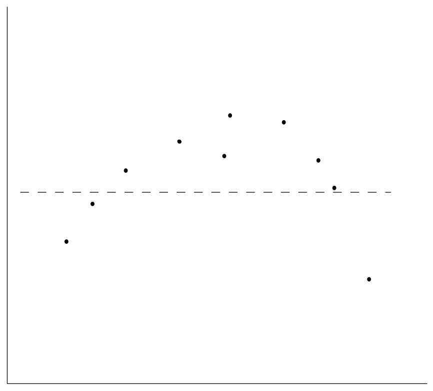

Figure 2: Example of Data Values Difficult to Fit with a Linear Approximation

It is possible to choose a representation that is too simple to fit the data. For example, if we try to fit the data in Figure 2 with a line, there will be no good choices. That is to say, we will be able to find the line that minimizes error, but the error would be unacceptably large. In this case, choosing a higher-degree polynomial representation would make sense. However, in this class, we will mainly be interested in finding the *best* solution given a particular, fixed representation.

1.1 Measures of Error

Imagine that we have $N + 1$ points $\{(x_0, y_0), (x_1, y_1), \ldots, (x_N, y_N)\}$. In previous lectures, our approach was to interpolate these $N + 1$ points by a polynomial or a spline, and, assuming that the y-values were taken from an underlying function, our goal was to minimize our interpolant's error with respect to this underlying function everywhere on an interval. In this setting, we did not worry about error at the interpolation points themselves, since by construction the error was always zero.

In this lecture, our goal is instead to *approximate* the underlying function - so there could be error even at the "interpolation" points $(x_i, f(x_i))$.

Thus, the first question we must address when we study function approximation is this: what is the error at the "interpolation" points (x_i, y_i)? Suppose we have, somehow or other, come up with an approximating polynomial p. We define an error vector

$$\begin{bmatrix} e_0 \\ \vdots \\ e_N \end{bmatrix} := \begin{bmatrix} y_0 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} p(x_0) \\ \vdots \\ p(x_N) \end{bmatrix}$$

The ith entry in this vector measures the error of p with respect to the true value at the ith "interpolating" point x_i. Now suppose our competitor comes up with a different approximating polynomial $\hat{p}$. We define an analogous error vector

$$\begin{bmatrix} \hat{e}_0 \\ \vdots \\ \hat{e}_N \end{bmatrix} := \begin{bmatrix} y_0 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} \hat{p}(x_0) \\ \vdots \\ \hat{p}(x_N) \end{bmatrix}$$

How can we show that our error vector is better than his? We need to compare vectors of numbers, and it is not immediately obvious what is the best way to do so.

For example, suppose my error vector is

$$e = \begin{bmatrix} .1 \\ -.1 \\ .1 \end{bmatrix}$$

and my competitor's error vector is

$$\hat{e} = \begin{bmatrix} .2 \\ -.05 \\ 0 \end{bmatrix}$$

We need to map each vectors to a single number in order to compare them. Our first impulse may be to map our error

$$e \mapsto \sqrt{.1^2 + (-.1)^2 + .1^2} = 0.17$$

and our competitor's error

$$\hat{e} \mapsto \sqrt{.2^2 + (-.05)^2 + 0^2} = 2$$

It looks like we win big! But perhaps our competitor will argue that we should instead map the error vectors to numbers as

$$e \mapsto |.1| + |-.1| + |.1| = .3$$
$$\hat{e} \mapsto |.2| + |-.05| + |0| = .25$$

Now our competitor is doing better.

Thus we see that there is no universal way to associate size to a vector of numbers. We also see that our choice how to measure size can make a big difference - indeed, the difference between algorithms that measure error in one way or another is sometimes huge. How can we make an informed choice?

As we will see, measuring the size of vectors is a basic problem in linear algebra.

2 Vector Norms

In linear algebra, the size of a vector $\vec{v}$ is called the *norm* of $\vec{v}$. We will discuss a few different kinds of norms. For this discussion, we define a vector $\vec{v}$ as an ordered tuple of numbers.

$$\vec{v} = (v_1, v_2, \ldots, v_n) \qquad (v_i \in \mathbb{C} \text{ for } i = 1, 2, \ldots, n)$$

(1) One Norm $\|\vec{v}\|_1$

The one-norm (also known as the L_1-norm, ℓ_1 norm, or mean norm) of a vector $\vec{v}$ is denoted $\|\vec{v}\|_1$ and is defined as the sum of the absolute values of its components:

$$\|\vec{v}\|_1 = \sum_{i=1}^{n} |v_i| \tag{1}$$

for example, given the vector $\vec{v} = (1, -4, 5)$, we calculate the one-norm:

$$\|(1, -4, 5)\|_1 = |1| + |-4| + |5| = 10$$

(2) Two Norm $\|\vec{v}\|_2$

The two-norm (also known as the L_2-norm, ℓ_2-norm, mean-square norm, or least-squares norm) of a vector $\vec{v}$ is denoted $\|\vec{v}\|_2$ and is defined as the square root of the sum of the squares of the absolute values of its components:

$$\|\vec{v}\|_2 = \sqrt{\sum_{i=1}^{n} |v_i|^2} \tag{2}$$

for example, given the vector $\vec{v} = (1, -4, 5)$, we calculate the two-norm:

$$\|(1, -4, 5)\|_2 = \sqrt{|1|^2 + |-4|^2 + |5|^2} = \sqrt{42}$$

(3) Infinity Norm $\|\vec{v}\|_\infty$

The infinity norm (also known as the L_∞-norm, ℓ_∞-norm, max norm, or uniform norm) of a vector $\vec{v}$ is denoted $\|\vec{v}\|_\infty$ and is defined as the maximum of the absolute values of its components:

$$\|\vec{v}\|_\infty = \max\{|v_i| : i = 1, 2, \dots, n\} \tag{3}$$

for example, given the vector $\vec{v} = (1, -4, 5)$, we calculate the infinity-norm:

$$\|(1, -4, 5)\|_\infty = \max\{|1|, |-4|, |5|\} = 5$$

(4) p-Norm $\|\vec{v}\|_p$

In general, the p-norm (also known as L_p-norm or ℓ_p-norm) for $p \in \mathbb{N}$, $1 \leq p < \infty$ of a vector $\vec{v}$ is denoted $\|\vec{v}\|_p$ and is defined as:

$$\|\vec{v}\|_p = \sqrt[p]{\sum_{i=1}^{n} |v_i|^p} \tag{4}$$

There is an important qualitative difference between the 1- and 2-norms:

- Small entries in a vector contribute more to the 1-norm of the vector than to the 2-norm. For example, if $v = (.1, 2, 30)$, the entry .1 contributes .1 to the 1-norm $\|v\|_1$ but contributes roughly $.1^2 = .01$ to the 2-norm $\|v\|_2$.
- Large entries in a vector contribute more to the 2-norm of the vector than to the 1-norm. In the example $v = (.1, 2, 30)$, the entry 30 contributes only 30 to the 1-norm $\|v\|_1$ but contributes roughly $30^2 = 900$ to the 2-norm $\|v\|_2$.

Thus in our setting, when we want to compare the sizes of error vectors, we should use the 1-norm if we prefer a few large errors to an accumulation of small errors - if we minimize error in the 1-norm, most entries in the error vector will be 0, and a few may be relatively large. In the 2-norm, there will be lots of small errors but few large ones. If we want to avoid outliers and don't mind having many small (but nonzero) errors, we should use the 2-norm.

3 Matrix Norms

We will also find it useful to measure the size matrices. Thus, we define the *matrix norm* as the "size" of a matrix. We do not measure matrices in the same way we measure vectors, however. That is, given a matrix (a_{ij}) we do *not* take the one-norm by adding up the absolute values of the individual components a_{ij}. Instead, we consider the effect the matrix has when multiplied by vectors.

A matrix A can be seen as a map from vectors to vectors, for example the matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ -4 & -5 & 6 \end{bmatrix}$$

can be seen as a map from $\mathbb{R}^3$ to $\mathbb{R}^2$ that takes a vector v to another vector Av:

$$v = \begin{bmatrix} v(1) \\ v(2) \\ v(3) \end{bmatrix} \mapsto Av = \begin{bmatrix} v(1) + 2v(2) + 3v(3) \\ -4v(1) - 5v(2) + 6v(3) \end{bmatrix}.$$

We would like to measure how much a matrix A can amplify a vector $\vec{v}$ (that is, increase its norm), but notice that a given matrix A will have different effects on different vectors $\vec{v}$, for example:

$$\begin{bmatrix} 10^6 & 10^6 \\ 10^6 & 10^6 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} 10^6 & 10^6 \\ 10^6 & 10^6 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \cdot 10^6 \\ 2 \cdot 10^6 \end{bmatrix}$$

When we take the norm of a matrix A, we would like to find the maximum expansion it can cause to any vector $\vec{v}$. More formally:

DEFINITION 3.1 (Norm of a Matrix). *Given a particular vector norm $\| \|$, and $M \times N$ matrix A, the norm of A is defined as follows:*

$$\|A\| = \max\{\|A\vec{v}\| : \|\vec{v}\| = 1\}$$

Equivalently, $\|A\| = \max\{\frac{\|Av\|}{\|v\|} : v \neq 0\}$.

Note that this definition does not provide an explicit formula for calculating the norm, but, in some cases, calculating the matrix norm is not too difficult. To illustrate, consider the following example.

EXAMPLE 3.1. *Determine the one-norm of the matrix:*

$$A \;=\; \begin{bmatrix} 1 & -3 \\ 2 & 8 \end{bmatrix}$$

To solve this problem, consider the effect that matrix A has on the simplest unit vectors, $(1,0)^T$ and $(0,1)^T$:

$$\begin{bmatrix} 1 & -3 \\ 2 & 8 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad \left\| \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right\|_1 = |1| + |2| = 3$$
$$\begin{bmatrix} 1 & -3 \\ 2 & 8 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -3 \\ 8 \end{bmatrix} \qquad \left\| \begin{bmatrix} -3 \\ 8 \end{bmatrix} \right\|_1 = |-3| + |8| = 11$$

It turns out that 11 is the maximum norm of any unit vector multiplied by A, so that $\|A\|_1 = 11$. In general, if one splits a matrix A into its column vectors.

$$A_{M\times N} = [A_1, A_2, \ldots A_N]$$

then the one-norm of A is the maximum of the one-norms of the column vectors A_i of A.

$$\|A\|_1 = \max\{\|A_i\| : A_i \text{ is a column vector of } A\}$$

Similarly, in general if one splits a matrix A into its row vectors

$$A_{M\times N} = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_M \end{bmatrix}$$

then the ∞-norm of A is the maximum of the one-norms of the row vectors vectors A_j of A:

$$\|A\|_\infty = \max\{\|A_j\| : A_j \text{ is a row vector of } A\}$$

For example, let

$$A = \begin{bmatrix} 1 & 2 & 3 \\ -4 & -5 & 6 \end{bmatrix}$$

It is easy to see that, for the purpose of getting the biggest $\|Av\|_\infty$, the "best" unit vectors in the ∞-norm are those the form

$$\begin{bmatrix} \pm 1 \\ \pm 1 \\ \pm 1 \end{bmatrix}$$

where the signs are chosen to avoid cancellation when multiplying by a row of A. In the case of our matrix A above, the biggest we can get is

$$\left\| \begin{bmatrix} 1 & 2 & 3 \\ -4 & -5 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\|_\infty = \left\| \begin{bmatrix} 6 \\ 3 \end{bmatrix} \right\|_\infty = 6 \quad \text{maximizing the first entry of the output vector}$$

$$\left\| \begin{bmatrix} 1 & 2 & 3 \\ -4 & -5 & 6 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} \right\|_\infty = \left\| \begin{bmatrix} 0 \\ 15 \end{bmatrix} \right\|_\infty = 15 \quad \text{maximizing the second entry of the output vector}$$

So $\|A\|_\infty$ is 15.

Algorithms for finding the two-norms (and higher p-norms) of a matrix are more complex, and we do not discuss them here. We can get lower bounds on the two norm by taking any vector with 2-norm equal to 1, e.g.,

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},$$

multiplying our matrix by this vector, e.g.,

$$\begin{bmatrix} 1 & 2 & 3 \\ -4 & -5 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -4 \end{bmatrix},$$

and finally computing the two norm of the output

$$\left\| \begin{bmatrix} 1 \\ -4 \end{bmatrix} \right\|_2 = \sqrt{17}.$$

So $\|A\|_2 \geq \sqrt{17}$.

4 Linear Systems

Linear systems appear quite frequently in scientific applications, perhaps not so much because real-world problems are actually linear as because linear systems are easy to solve and often provide good approximations for real-world problems. Recall that a simple system of linear equations:

$$\begin{aligned} 2x_1 - 3x_2 &= 7 \\ 3x_1 + 6x_2 &= -1 \end{aligned}$$

can be rewritten in matrix form:

$$\begin{bmatrix} 2 & -3 \\ 3 & 6 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 7 \\ -1 \end{bmatrix}$$

In general, we can write any linear system of M equations in N unknowns in the form:

$$A_{M\times N}\vec{x}_{N\times 1} = \vec{b}_{M\times 1}$$

In this class, we shall discuss some of the key issues of linear systems, including:

1. **Algorithms** - What methods can we use to find numerical solutions to linear systems.
2. **Complexity** - What is the cost of each algorithm. This will determine how large a linear system can be solved in a reasonable amount of time.
3. **Conditioning** - How *good* is the system we are trying to solve? Some (ill-conditioned) systems of equations will pose problems for any numerical algorithm.
4. **Stability** - How *reliable* are the algorithms.

It turns out that all of the direct, finite algorithms for solving linear systems depend on the concept of *factorization.* Factorization algorithms use the idea that there are some kinds of *simple* matrices which make linear equations easy to solve, for example:

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

We can find the solution to this system by inspection:

$$\begin{aligned} x_1 &= 1/2 \\ x_2 &= 4/3 \end{aligned}$$

Factorization algorithms find ways to transform a general linear system into an equivalent linear system composed of such simple matrices. We will see examples of factorization algorithms next time.