An Introduction to Empirical Modeling

Douglas Robert Hundley
Mathematics Department
Whitman College

January 20, 2004

Contents

Chapter 1

Preface

This is a book about empirical modeling, and it is designed for undergraduates with a background only in calculus and linear algebra. By *empirical*, we mean that there is no underlying analytic model- we are attempting to extract meaning (or patterns) from raw data, and therefore many of the techniques and problems stem from areas of machine learning.

The goals of a course drawn from this text are to:

- Expose undergraduates to a wide range of "new" ideas that are in current use in industry.
- Give students background in manipulating large data sets and introduce the questions that arise.
- Use the Singular Value Decomposition as a centerpiece for our computations- The SVD is arguably the most important theorem from linear algebra, and yet, it receives little attention.
- Get the reader into the computations as quickly as possible so that he/she can begin experimenting independently.

What this book will not do:

- We will not go deeply into statistics. Some might claim that this entire topic should be a statistics course. A better way to view this book is as a way to give students a better understanding of *why* statistics should be used. A student at the end of this course will be ready to ask the right questions, and should seek out statistics as one approach to answering the questions.
- We will not go deeply into optimization theory. An appendix is devoted to the treatment of derivatives for multivariate functions, and some optimization notes are also included. At the undergraduate level, it is appropriate to give the students an intuitive background, and let them experiment.

- We introduce the ideas behind neural networks, and will use them extensively, but much of the statistical theory has been left out. Readers interested in these particular topics can find some great references out there, and we will list them in the appropriate places.

With those caveats in mind, then what is this book? It is a text that is meant to be *introductory*, and it is meant to be a mathematics book. We will use linear algebra extensively, and so we will be looking at data analysis from more of a geometric or topological point of view. For graduate students, this textbook could be treated as a companion text to a book from my colleague, Michael Kirby, "Geometric Data Analysis", which I highly recommend. In fact, much of the research presented in this text was a collaboration between myself and Michael.

In this book, we will often return to the "big picture", and draw connections between courses in order to build abstractions. For example, "data" can come in many forms- I might consider a data "point" as a vector, or as a human face, or as an entire movie (in Part II, Data Representations). I might consider a data point as an entire function (in Part III, Functional Representations). What is unifying in all representations is the concept of the vector space, and this is where we begin.

Another unifying concept of this text is that of *learning*. Machine learning is all about getting representations of data that are sufficiently narrow that we can extract similarities, but are sufficiently broad as to be able to differentiate dissimilarities. For example, in unsupervised learning, we may try to split a data set into classes. The big issue is, of course, how many? In supervised learning, we'll try to incorporate some expert knowledge into the building of an association. This association should be accurate, but at the same time, be broad enough to handle new inputs. We'll start getting into these issues shortly, and will often return to these questions.

Chapter 2

Data

In general, we might think of "data" as the end result of some physical process. That is, measurements of some process have been taken, and these numerical results have been recorded. The data can therefore depend on some temporal process (a *time series*), like the stock values of a certain company during the day, the oil pressure in a running engine, the atmospheric temperature at a given spot on the surface of the earth, the population of rabbits on an island, etc. The data may not depend on time; for example, the pixel values on a photograph, the analysis of a document (perhaps by using a word count), the answers to a questionnaire, the results of a blood test, etc.

In looking at a data set for the first time, you should be asking yourself: What knowledge am I trying to extract from this data? It is in this sense that we think of data as carrying *information*, and it is our task to figure out what that information is[1]. Our task is made harder by the fact that, in general, the data will generally be represented in a form that hides the information (if not, our task is greatly simplified!).

Here are some examples of data. As you read the description, think about what information you might like to extract.

1.

2.1 Representations of Data, Part I

In linear algebra, we learned techniques that allowed us to translate vectors from one basis to another. For example, suppose we have a set of data, $\left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}\right\}$, where $\mathbf{x}^{(i)} \in \mathbb{R}^{10}$. The standard way of representing this data is to use the standard basis, $\mathbf{e}_1, \ldots, \mathbf{e}_{10}$:

$$\mathbf{x}^{(i)} = \sum_{k=1}^{10} x_k^i \mathbf{e}_k$$

[1] You may think we're heading into an area known as *information theory*, but we'll only be looking at it superficially.

where x_k^i is the k^{th} component of vector i. But this seems like overkill- do you really need 10 basis vectors to describe three data points? Here's a better representation:

$$\mathbf{x}^{(1)} = 1 \cdot \mathbf{x}^{(1)} + 0 \cdot \mathbf{x}^{(2)} + 0 \cdot \mathbf{x}^{(3)}$$
$$\mathbf{x}^{(2)} = 0 \cdot \mathbf{x}^{(1)} + 1 \cdot \mathbf{x}^{(2)} + 0 \cdot \mathbf{x}^{(3)}$$
$$\mathbf{x}^{(3)} = 0 \cdot \mathbf{x}^{(1)} + 0 \cdot \mathbf{x}^{(2)} + 1 \cdot \mathbf{x}^{(3)}$$

In this setting, $\mathbf{x}^{(1)}$ is represented as $(1, 0, 0)$, $\mathbf{x}^{(2)}$ is $(0, 1, 0)$, and $\mathbf{x}^{(3)}$ is $(0, 0, 1)$. We are **reducing the dimension** of the set from $\mathbb{R}^{10}$ to $\mathbb{R}^3$. If we were to further assume that $\mathbf{x}^3$ is actually $2\mathbf{x}^{(1)} - 3\mathbf{x}^{(2)}$, then the dimension can be reduced even further:

$$\mathbf{x}^{(1)} \to (1, 0), \quad \mathbf{x}^{(2)} \to (0, 1), \quad \mathbf{x}^{(3)} \to (2, -3)$$

A key idea here is that a data set spans itself.

In general, if we have a basis, $\mathcal{B} = \{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ for some $k-$dimensional vector space, then any point in that space can be written as:

$$\mathbf{x} = \sum_{j=1}^{k} \alpha_i \mathbf{v}_i = B[x]_b$$

where $B = [\mathbf{v}_1 \ , \ldots, \mathbf{v}_k]$. If we have two sets of bases, $\mathcal{B}, \mathcal{C}$, then we can define the change of coordinates via the relations:

$$\mathbf{x} = B[x]_b \ \mathbf{x} = C[x]_c \Rightarrow B[x]_b = C[x]_c$$

so that we can determine either $[x]_b, [x]_c$ given the other set of coordinates.

So, there are an infinite number of choices for the basis. The question of which to choose will be left to consider later on.

A related notion is that a matrix of numbers has a dual nature in that if we are just presented with an $m \times n$ matrix, we do not know if it represents m data points in $\mathbb{R}^n$ or n data points in $\mathbb{R}^m$. We will consider this duality when we introduce the Singular Value Decomposition (SVD).

2.2 Representations of Data, Part II

For data that is a time series (that is, the data is given to us as a string of real numbers), it is a common practice to *embed* the time series to $\mathbb{R}^k$ in the following way:

> Given $x_1, x_2, x_3, x_4, \ldots$, a sequence of numbers, the following represents the data embedded in $\mathbb{R}^2$ via lag 2:
>
> $$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \begin{bmatrix} x_2 \\ x_3 \end{bmatrix}, \begin{bmatrix} x_3 \\ x_4 \end{bmatrix}, \ldots$$

or we could embed the sequence in $\mathbb{R}^3$ via lag 3:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \begin{bmatrix} x_2 \\ x_3 \\ x_4 \end{bmatrix}, \begin{bmatrix} x_3 \\ x_4 \\ x_5 \end{bmatrix}, \ldots$$

and so on. We'll discuss this technique in depth later on.

Why do such a thing? To plot a sequence of real numbers, we normally plot the index i along the horizontal axis, with the value of x_i along the vertical. Therefore, we think of time as progressing along the horizontal axis. The problem is that using this plot makes time leave the graph.

On the other hand, by plotting a sequence of data using a lag 2 vector, the plot is of the ordered pairs (x_i, x_{i+1}). Now time is shown graphically by the path through the data (which will hopefully be confined to some bounded region of space).

2.3 Representing Functions instead of Data

As we have seen, a given set of data can be viewed in terms of its time series:

$$f(t_0), f(t_1), f(t_2), \ldots, f(t_N)$$

but it is interesting to consider other types of representations. The underlying function might be represented by a sum of sines and cosines:

$$f(t) = a_0 + \sum_{k=1}^{m} a_k \sin(kt) + b_k \cos(kt)$$

in which case, f could be represented by its *frequency* content (versus its *temporal* content). That is, I could define f by telling you what frequencies f consists of (that is, I could give you numbers for a_i, b_i) , rather than telling you what f is at various times. We will consider this more in depth in the section on Fourier analysis.

In fact, this is a special case of a more abstract setting. Given that f is some linear combination of functions $\phi_k(t)$,

$$f(t) = \sum_{k=1}^{m} b_k \phi_k(t)$$

f could be defined by stating the weights of the linear combination, b_k. An example we're familiar with is the Taylor series for f based at $t = a$. In this case, $\phi_k(t) = (t-a)^k$, and:

$$f(t) = \sum_{k=0}^{\infty} b_k (t-a)^k$$

where we know that $b_k = \frac{1}{k!} f^{(k)}(a)$, where $f^{(k)}$ is the k^{th} derivative of f, and $f^{(0)} = f$.

From linear algebra, we know that $C[a,b]$ is an infinite dimensional vector space, and so requires an infinite number of basis vectors. In our previous examples, the functions ϕ_k are simply the basis vectors, and the b_k are the weights in that basis. Once we discretize a function to only n data points, the set of functions over those points becomes an $n-$ dimensional space- that is, there are only $n-$ parameters that are required to define any function over those $n-$ points. In fact, this space is isomorphic to $\mathbb{R}^n$ with the standard basis.

The representations given so far are called *Linear Models*. They are called that because the model parameters are simply the weights of a linear combination (but notice that the basis functions themselves are *not* linear!). We will describe some nonlinear models later on.

2.4 Summary

In this chapter, we have discussed the nature of representing data, and the basic types of representation available to us. In ambiguous data, the choice of a basis may carry unintended consequences, as we will discuss in the exercises.

In the first case, we treated data as existing in some linear subspace, and tried to construct a basis for it. In the second two cases, we treat the data as time oriented, and discussed higher dimensional representations via the lag vector, or a functional representation of the data via sums of sines and cosines (although there are many ways of representing a function).

There were two main points of this section: (1) I would like for you to start thinking about what data represents, (2) Is it possible for some essential form of a data set to exist? What would that look like? This is more of a philosophical question, but it is critical in understanding what you want the outcome of data analysis to tell you.

2.5 Exercise Set

1. (Matlab Exercise) Write a Matlab function that will input a time series and output a lagged matrix. The input parameters should include the lag length and a value representing a skip. Some samples of what your code should do follow, where x is a vector of integers from 1 to 10.

    - Function call: `A=lagmatrix(x,4,2)`
    - Output:
    $$A = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \\ 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 10 \end{bmatrix}$$
    - Function call: `A=lagmatrix(x,2,3)`

- Output:

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix}$$

2. Suppose you are told that $1, 3, 5, 7, \ldots$ belongs to class A, and $2, 4, 6, 8, \ldots$ belongs to class B. What information is being carried- What is the underlying process that is producing this data? What does 1.5 produce?

3. Let B, C be two matrices that give two different sets of basis vectors for $\mathbb{R}^2$:

$$B = \begin{bmatrix} -1 & 0 \\ 1 & 2 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 3 \\ -2 & 1 \end{bmatrix},$$

If $[x]_B = [1, 2]^T$, then determine $\mathbf{x}$ and $[x]_C$.

4. What might be considered the opposite of an essential representation has been called a *sparse* representation. That is, the data that we are presented with is the output of some projection, and we wish to find a higher dimensional representation. Some examples:

   - "Shape from Shading" is where we find a three dimensional model of a shape from a two dimensional representation. For example, find the shape of the clown's face from his photo.
   - "The Cocktail Party Problem". It is a well known phenomenon that, if we are in a large room full of people speaking, we can isolate the sound of the person speaking to us. A particular instance of this problem: Given a tape recording of an orchestra, for example, is it possible to isolate each instrument's sound?

   In terms of what we know from linear algebra, why is this such a harder problem that finding an essential representation of data?

5. Recall that Gram-Schmidt orthogonalization inputs a set of vectors,

$$\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$$

, and outputs an orthogonal basis for it, $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$ in the following way:

   - Let $\mathbf{v}_1 = \mathbf{x}_1$
   - $\mathbf{v_2} = x_2 - \dfrac{\mathbf{x}_2 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1} v_1$
   - $\mathbf{v_3} = x_3 - \dfrac{\mathbf{x}_3 \cdot \mathbf{v}_2}{\mathbf{v}_2 \cdot \mathbf{v}_2} v_2 - \dfrac{\mathbf{x}_3 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1} v_1$
   - And so on

   Use Gram-Schmidt to orthogonalize the following set:

$$[3, -4, 5]^T, \quad [-3, 14, -7]^T$$

   First do the orthogonalization with the first vector as $\mathbf{v}_1$. Does your answer change if you use the second vector as $\mathbf{v}_1$?

6. What is the smallest number of basis vectors needed to represent the data:

$$\mathbf{x}_1 = \begin{bmatrix} 2 \\ -1 \\ 2 \\ 0 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 2 \\ -1 \\ 2 \\ 3 \end{bmatrix}, \mathbf{x}_3 = \begin{bmatrix} -1 \\ \frac{1}{2} \\ -1 \\ 0 \end{bmatrix}, \mathbf{x}_4 = \begin{bmatrix} 6 \\ -3 \\ 6 \\ 9 \end{bmatrix}$$

Write down what the low-dimensional representations are for $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$.

7. This series of exercises is designed to get you to consider what happens when we put a time series into $\mathbb{R}^2$ or $\mathbb{R}^3$.

Let $y = \sin(x)$, where x is taken as a sampling of the time interval from $-\pi$ to π. Type in the following script:

```
x=linspace(-pi,pi,3000);  %Get 3000 samples evenly spaced
y=sin(x); for j=2:20
  plot(y(1:j-1),y(j:end))
  pause
end
```

(a) Before beginning, show (graphically) that: $-\cos(t) = \sin\left(t + \frac{\pi}{2}\right)$ and that $-\sin(t) = \sin(t + \pi)$.

(b) Note that the effect of this code is to plot the series y at time t against the series y at time $t+\tau$. Can you predict what will happen? (HINT: Consider what will happen if $\tau = 0$, $\tau = \frac{\pi}{2}$ and $\tau = \pi$ using the previous part).

(c) Run the code and write up your observations: What happens to the curve as j changes?

(d) Try changing y to $\cos(x)$. Any changes?

(e) Try changing x: `x=linspace(0,100*pi,10000)`, and change y to `y=sin(x)+sin((1/sqrt(2))*x)`, and re-run the code. What happens now?

8. The *dimension* of a set varies with the definition used. We explore a couple of these below:

- We will define the Euclidean Dimension (D_e) of a set S to be the (smallest) number of basis vectors needed to describe the given set. This is a global quantity, and is the definition we used in linear algebra.
- The Topological Dimension (D_T) of a set S is something that is local in nature- that is, the topological dimension is something that is determined point by point. To understand this definition, recall that an "$\epsilon-$ neighborhood" (in our usual space R^n) about a point x is the set of points y so that $\|x - y\| < \epsilon$.

- A set has topological dimension 0 if every point has arbitrarily small neighborhoods whose boundaries do not intersect the set.
- A set S has topological dimension k if each point in S has arbitrarily small neighborhoods whose boundaries meet S in a set of topological dimension $k-1$, and k is the least nonnegative integer for which this holds.

(a) Suppose S is a circle in the plane. What is D_e, D_T?

(b) Do the same for the sphere ($\|x\| = 1$) in three dimensions.

(c) Suppose S is the letter "X" in the plane. What is D_e, D_T?

(d) Give some examples of sets S where $D_e = D_T$.

(e) Consider the set S in the plane that consists of a filled in square with a line segment attached to one side. Discuss any problems you might have with computing D_T. In determining D_T, be sure to read its definition carefully.

Part V

Appendices

Appendix A

An Introduction to Matlab

A.1 What is Matlab?

Matlab is a computer program designed to do mathematics. You might think of it as a super-calculator. That is, once Matlab has been started, you can enter computations, and Matlab will produce the results. Matlab has many built-in programs and has some great graphics features that we'll discuss later.

Starting the Matlab Program

The Matlab program physically resides on the computer named "Hope" in the math lab. To run Matlab, you do not have to be physically sitting at Hope; you can be at any of the computer terminals.

Once you have logged into your particular machine, you'll need to log into Hope to make Matlab work. To do this, in your command window type: `ssh hope` and you'll be on that machine. Now just type `matlab` and you'll see the program start. The splashscreen will come up, and you'll get a command window that looks like:

```
                    < M A T L A B >
        Copyright 1984-2001 The MathWorks, Inc.
            Version 6.1.0.450 Release 12.1
                     May 18 2001

  To get started, select "MATLAB Help" from the Help menu.
>>
```

To exit from Matlab, either type `exit` from the command window, or choose `Exit` from the window menu (under File).

How does Matlab work?

You can make Matlab do computations three different ways:

- Type commands directly into the keyboard.
- Have your Matlab commands typed into a separate text file (called a **script file**), and then have Matlab read these commands in. This is very nice- it gives you documentation and allows you to run similar computations several times without having to re-type the commands.
- Define your own functions by typing a separate text file (called an **m-file**).

Saving your work

If you haven't written a script file, but are doing your computations "live", you may want to begin the session by typing: `diary filename`

All subsequent keyboard input and output will then not only be on the computer monitor, but will also be saved as "filename". For example, if you're using Matlab for homework problem 3.1, you may use the command: `diary hw3_1` to save your work.

Important: The "diary" command *must* be used *prior* to typing in the commands you want to save.

Matlab has a very nice text editor that you can use to type out and save Matlab functions and scripts- To access the editor, type `edit` in the Matlab command window.

A.2 Introductory Commands

1. **Arithmetic**

   Matlab understands all of the basic arithmetic functions, `+, -, *, /, ^` are addition, subtraction, multiplication, division and exponentiation. Type them in just as you would write them. For example, 2^5 would be typed as `2^5`.

2. **Trigonometric Functions**

   Matlab understands the basic trig functions sine, cosine and tangent as `sin , cos , tan` . So, for example, the sine of 3.1 would be typed as: `sin(3.1)`

   The number π is used so frequently that Matlab has its (approximate) value built-in as the constant `pi`. For example, $\sin(\pi)$ is typed as `sin(pi)`. Note that π uses a lowercase "P".

3. **Exponential and Logarithmic Functions**

   Matlab does not have the number e built-in as a constant (like π). To take the number e to a power, use the functional form: $\mathrm{e}^x =$ `exp(x)` So if I want the number e, I would type `exp(1)`, and so on.

   For the natural log (log base e), use the notation `log`. For example, $\ln(3)$ is written as `log(3)`. We will only use the natural log- if in the future you want a different base, look up the log command by typing `help log`.

4. **Complex Numbers and Arithmetic**

   Matlab has complex arithmetic built-in. Either the letter i or j can be used to represent $\sqrt{-1}$, but a word of caution is in order here: You can only use i or j for $\sqrt{-1}$ ONLY if you have not previously defined them. If you think you're going to use complex numbers, do not use the letter i for anything but complex arithmetic! Example: `(0.2+3*i)*(5+2*i)` will multiply the two complex numbers together (using complex arithmetic).

A.2.1 Helpful Administrative Commands

The following commands are useful as you begin to use Matlab more and more:

`who` List all variables currently in use.

`whos` List all variables, and their sizes.

`ls` or `dir` List the contents of the current directory.

`cd` Change the directory. For example, `cd examples` would change the current directory to your file named `examples`. To go up the structure instead of down, type `cd ..`

`pwd` Tell me where in the directory structure I'm currently at.

`where` *command* Tell me where *command.m* is located.

`help` *command* List the help file for the function *command*. For example, to get help on the sine function, type `help sin`.

`demo` Lists all the demonstration programs that Matlab came with- This is fun to look at. We don't have all of them; you can go to Matlab's website to look at more: `www.mathworks.com`.

A.2.2 A Programming Note: Assignment v. Equality

In computer programming, the equal sign does not mean mathematical equality. We use the equal sign as an assignment operator. For example,

```
A=3;
```

means to assign the value of 3 to the variable A- If A has not been assigned to anything before, this command will also create that variable.

The format of assignment is always the same: $A = B$ means that we will assign the value of B to the variable A.

Using this, what will the following commands do?

```
x=5;
x=x+3;
```

Answer: First, the value 5 is assigned to the variable x. Next, $x+3$ is computed as 8, and finally, the value of 8 is assigned to the variable x.

Not an example: `5=x;`, you would get an error- the number 5 is not a variable.

The only exception to this rule is when you pass the computer an *equation*. You will know this because you are putting the equation in single quotes. For example,

```
x=solve('3*x+5*exp(x)=0')
```

passes the equation $3x + 5e^x = 0$ into the function `solve`.

A.3 Exercise Set

1. The following script file is an example of Newton's method applied to a function $f(x) = xe^x - \cos(x)$. Recall that Newton's Method solves for x: $f(x) = 0$ by taking an initial guess, x_0, and refines the guess by:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

```
x=0.2;  %Initial guess for solution to f(x)=0
for k=1:5
  y=x*exp(x)-cos(x);
  dy=(x+1)*exp(x)+sin(x);
  x=x-(y/dy)
end
```

Notes about the code:

- We see our first `for` loop. We'll discuss what this does in class.
- Note the use of `%` to make comments.
- Note that `x=x-(y/dy)` does NOT have a semicolon at the end.

(a) Use the `edit` feature to type it in and save it as `newton1.m`

(b) Run the code after you've saved it by typing `newton1` in the command window.

(c) Write down Matlab's output.

(d) To see more significant digits, type `format long`

(e) Type `whos` and write down Matlab's answer. If you're continuing to the next exercise, type `clear` to clear Matlab's memory.

2. Use Matlab's equation solver to do the last exercise by typing:

```
x=solve('x*exp(x)-cos(x)=0')
```

Are your answers the same?

3. Find the 6 **roots of unity** in Matlab by typing

```
x=solve('x^6=1')
```

Write down your answers, and then type `whos`. What is Matlab telling you about what x is? Can you guess?

A.4 Matrices

Matlab was originally designed as a "front end" to access LINPACK and EISPACK, which are numerical linear algebra packages written in FORTRAN. From this beginning, Matlab's basic data type is the matrix.

I enter the following matrix:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$$

as:

```
A=[1 2 3 4; 5 6 7 8];
```

or as:

```
A=[1 2 3 4
5 6 7 8];
```

Note the use of the semicolon: Inside a matrix, the semicolon indicates the end of a row. Outside the matrix, the semicolon suppresses Matlab output. You can also separate numbers using a comma if you'd prefer that. Rows and columns are entered in a corresponding way, as either a $1 \times n$ matrix or as a $n \times 1$ matrix.

We access elements of the matrix in a natural way. For example, the $(2,3)$ element of A is written as `A(2,3)` in Matlab (in this case, $A(2,3)$ is 7). You can change the elements using the assignment operator `=`. For example, if we want to change the $(1,3)$ element of A from 3 to 6, type:

```
A(1,3)=6;
```

Special Commands: The colon operator

- `a:b`

  Produces a listing from a to b in a row:

  $$\texttt{a:b} \text{ gives } [a, a+1, a+2, \ldots, a+n]$$

  where n is the largest integer so that $a+n \leq b$. For example, $x = 2:9$ puts x as a row vector whose elements are the integers from 2 to 9.

- `a:b:c`

  Produces the numbers from a to c by adding b each time:

  $$\texttt{a:b:c} \quad \text{gives} \qquad [a, a+b, a+2b, \ldots, a+nb]$$

  where n is the largest integer so that $a + nb \leq c$. For example, $1:2:7$ returns the numbers $1, 3, 5, 7$. Type the following into Matlab to see what you get: `1:2:8` and `1:0.5:6`

A.4.1 Matlab commands associated with Arrays

- `linspace(a,b,c)`

  Produces c numbers evenly spaced from the number a to the number b (inclusive). For example, `x=linspace(2,3.5,40)` produces 40 numbers evenly spaced beginning with 2 and ending with 3.5.

  SHORTCUT: Leaving off the third number c will give you 100 numbers between a and b (That is, $c = 100$ is the default value.)

  Compare this with the colon operator. We would use the colon operator if we want to define the length between numbers, and use `linspace` if we want to define the endpoints.

- Random arrays (handy if you just need some quick data!)

  `A=rand(m,n)` Produces an $m \times n$ array of random numbers (uniformly distributed) between 0 and 1. If you just want a single random number between 0 and 1, just type `rand`

  `A=randn(m,n)` produces an $m \times n$ array of random numbers (with a normal distribution) with zero mean and unit variance. If you want a single random number (with a normal distribution), just type `randn`

- `A=zeros(m,n)` Produces an $m \times n$ array of zeros.

- `A=ones(m,n)` Produces an $m \times n$ array of ones.

- `A=eye(n)` Produces an $n \times n$ identity matrix.

- `A=repmat(B,m,n)` Matrix A is constructed from matrix (or vector) B by replicating B m times down and n times across.

  Example: Let $B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$. Then `A=repmat(B,2,3)` creates the array:

```
A =
     1     2     1     2     1     2
     3     4     3     4     3     4
     1     2     1     2     1     2
     3     4     3     4     3     4
```

Matrix Arithmetic

- Transposition is denoted by the single quote character `'`. That is, `A'` $= A^T$. (CAUTION: If A contains complex numbers, then `A'` is the *conjugate transpose* of A, sometimes denoted as $A^* = \bar{A}^T$)
- Matrix addition and subtraction is performed automatically and is only defined for matrices of the same size.
- Scalar addition. If we want to add a constant c to every item in an array A, type: `A+c`
- Scalar Multiplication: We can multiply every number in the array by a constant: If A is the array and c is the constant, we would write: `B=c*A`
- Matrix Multiplication: Use the regular multiplication sign for standard matrix multiplication. If A is $m \times n$ and B is $n \times p$, then `A*B` is an $m \times p$ matrix, as we did in linear algebra.
- Elementwise Multiplication. We can multiply and divide the elements of an array A and an array B *elementwise* by `A.*B` and `A./B`

  Exponentiation is done in a similar way. To square every element of an array A, we would write: `A.^2` This is the same as saying `A.*A`
- Functions applied to arrays: Matlab will automatically apply a given function to each element of the array. For example, `sin(A)` will apply the sine function to each element of the array A, and `exp(A)` will apply e^x to each element of the array. If you write your own functions, you should always decide ahead of time how you want the function to operate on a matrix.

Accessing Submatrices

Let A be an $m \times n$ array of numbers. Then:

The notation:	Yields:
`A(i,j)`	The (i,j)th element
`A(i,:)`	The entire ith row
`A(:,j)`	The entire jth column
`A(:,2:5)`	The 2d to fifth columns, all rows
`A(1:4,2:3)`	A 4×2 submatrix

Example: What kind of an array would the following command produce?

```
A([1,3,6],[2,5])
```

A 3×2 matrix consisting of the elements:

$$\begin{array}{cc} A(1,2) & A(1,5) \\ A(3,2) & A(3,5) \\ A(6,2) & A(6,5) \end{array}$$

Example: Create a 5×5 zero array, and change it to:

$$\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}$$

Answer:

```
A=zeros(5,5);  %Create the matrix of zeros
b=[1 2 3;4 5 6; 7 8 9];
A(2:4,2:4)=b;
```

Note also the use of the `%` sign. It is used to denote *comments*; that is, Matlab would ignore everything on the same line after the `%` sign.

Adding/Deleting Columns and Rows:

Its straightforward to insert and/or delete rows and columns into a matrix. Before doing it, we define `[ ]` as "the empty array": the array with nothing in it.

In the following, let A be a 4×5 array, let b be a 1×5 row, and c be a 4×1 column.

Examples of use (each of these are independent from the previous):

The command:	Produces:
`A(1,:)=[];`	Delete the first row.
`A([1,3],:)=[];`	Delete rows 1 and 3.
`A(:,3)=[];`	Delete column 3.
`A(:,1:2:5)=[];`	Delete the odd columns.
`A(1,:)=b;`	Put **b** as row 1.
`A(:,6)=c;`	Add c as the last column.
`d=[c , A(:,1:3)];`	d is c and columns $1-3$ of A.
`A=[A(:,1), c, A(:,2:5)];`	Insert c as column 2 of A, others shift 1 over.
`A=[A(1,:); b; A(2:4,:)];`	Insert b as row 2 of A, others shifted 1 down.

A.4.2 Solving $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$

To solve $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$, Matlab has two basic commands: `x=A\b` or `x=pinv(A)*b`. The command `pinv(A)` computes the *pseudoinverse* of A, which we will discuss later in the section dealing with the Singular Value Decomposition.

In linear algebra, there were three possible outcomes for solving $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$. They were:

1. A unique solution.
2. An infinite number of solutions.
3. No solution.

Matlab will always give exactly one solution. We need to interpret that solution in the second two cases. In the case of an infinite number of solutions (we have free variables in this case, also called *an underdetermined system*), the two methods may give different answers:

- `x=A\b` provides the most zeros in $\mathbf{x}$.
- `x=pinv(A)*b` gives $\mathbf{x}$ with the smallest norm.

Example: Let $A = \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 1 \end{bmatrix}$, with $b = \begin{bmatrix} 9 \\ 3 \end{bmatrix}$. Then the full solution is:

$$\mathbf{x} = \begin{bmatrix} 9+2t \\ 3-t \\ t \end{bmatrix} \tag{A.1}$$

In Matlab, the result of typing `x=A\b` is $x = [0, -15/2, -9/2]^T$ and the result of typing `x=pinv(A)*b` is $x = [4, 11/2, -5/2]$.

(MATLAB HINT: You can get Matlab to return numbers as fractions by typing `format rat`)

In the case of an overdetermined system (a system with no solution), Matlab will automatically return the *least squares* solution- that is, the answer $\mathbf{x}^*$ will be the minimum of $\|A\mathbf{x} - \mathbf{b}\|$:

$$\|A\mathbf{x}^* - \mathbf{b}\| \leq \|A\mathbf{x} - \mathbf{b}\|, \text{ for all } \mathbf{x}$$

In general, you should always use the forward slash (for help, type `help slash`): `x=A\b` which automatically determines a best numerical method. That is, sometimes its best (numerically) not to explicitly compute the pseudoinverse first.

A.5 Exercise Set

1. What is the Matlab command to create the array x which holds the integers: $2, 5, 8, 11, \ldots 89$

2. (Referring to the array above) What would the Matlab command be that zeros out the even-numbered indices (That is, $x(2), x(4), x(6), \ldots$)?

3. What is the difference in Matlab between typing: `x=[1 2 3]` and `x=[1,2,3]` and `x=[1;2;3]`? What happens if you type a semicolon at the end of the commands (i.e., `x=[1 2 3];`)?

4. (Referring to the last question) For each of those, what happens if you type `x.^2+3`? What happens if you forget the period (e.g., `x^2+3`)

5. What do the following commands do: `x=2;3;6;`, `x=2:3:6;`, `a=pi:pi:8*pi;`

6. Describe the output for each of the following Matlab commands. Recall that typing a semicolon at the end of the line suppresses Matlab output- to see the results, leave off the semicolon.

```
A=rand(3,4);
A([1,2],3)=zeros(2,1);
B=sin(A);
C=B+6;
D=2*B';
E=A./2;
F=sum(A.*A);
```

7. What will Matlab do if you type in:

```
A=rand(3,4);
A(:)
A(7)
```

NOTE: This is very bad programming style! Don't do it unless you know what you're doing!!

8. What is the Matlab command to perform the following:
   (a) Given an array x, add 3 to each of its values.
   (b) Given an array A, remove its first column and assign the result to a new array B.

9. What will the following code fragment do?

```
a=1:10;
for k=1:10
  h=ceil(length(a)*rand);
  b(k)=a(h);
  a(h)=[];
end
```

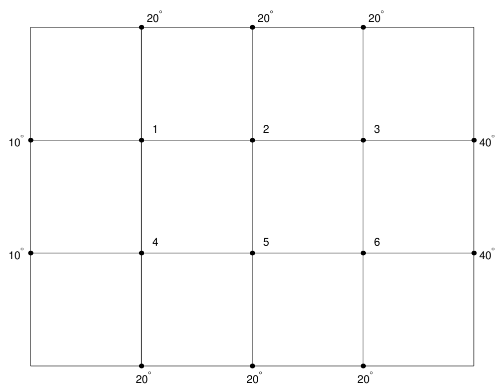
Compare this with `a=ceil(10*rand(10,1))` and `a=randperm(10)`

10. Use the Quick Summary sheet to help you write a code fragment that takes a random matrix X and re-sorts the columns so that the first column has the smallest size and the last column has the greatest size.

11. The plate in in the figure below represents a cross section of a metal beam[1]. Let $T_1, T_2, \ldots, T_6$ denote the temperature at the six interior nodes. The temperature at a node is approximately equal to the average of the four nearest nodes- to the left, above, to the right, and below. For instance,

$$T_1 = (10 + 20 + T_2 + T_4)/4 \text{ or } 4T_1 - T_2 - T_4 = 30$$

Write a system of equations whose solution gives estimates for the temperatures $T_1, \ldots, T_6$ and solve it in Matlab.



[1] See David Lay, Linear Algebra and its Applications, page 12

12. Find the interpolating polynomial $P(t) = a + bt + ct^2$ for the data

$$(1, 12), (2, 15), (3, 16)$$

That is, use Matlab to find a, b, c so that:

$$\begin{aligned} a + b \cdot 1 + c \cdot 1^2 &= 12 \\ a + b \cdot 2 + c \cdot 2^2 &= 15 \\ a + b \cdot 3 + c \cdot 3^2 &= 16 \end{aligned}$$

13. Use calculus to determine t that minimizes $\|x\|$ in Equation A.1. HINT: It suffices to find the minimum of $\|x\|^2$, so we don't have to differentiate the square root. Also give an argument about why this is true.

A.6 How do I get a Plot?

Here's a quick example to get us started:

```
x=linspace(-pi,3*pi,200);
y=sin(x);
plot(x,y);
```

You'll see that we had to create a domain array and a range array for the function. We then plot the arrays. For example,

```
plot([1,2],[3,4]);
```

will plot a line segment between the points $(1, 3)$ and $(2, 4)$. So, Matlab's plotting feature is drawing small line segments between data points in the plane.

A.6.1 Examples

1. Matlab can also plot multiple functions on one graph. For example:

```
x1=linspace(-2,2);
y1=sin(x1);
y2=x1.^2;
x2=linspace(-2,1);
y3=exp(x2);
plot(x1,y1,x1,y2,x2,y3);
```

produces a single plot with all three functions.

2. `plot(x1,y1,'*-');`

Plots the function `y1`, and also plots the symbol `*` where the data points are.

3. `plot(x1,y1,'k*-',x2,y3,'r^-');`

   Plots the function `y1` using a black (k) line with the asterisk at each data point, PLUS plots the function `y2` using a red line with red triangles at each data point.

   The following lists all of the built in colors and symbols that Matlab can use in plotting: (NOTE: You can see this list anytime in Matlab by typing: `help plot`)

Code	Color	Symbol	
y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus
g	green	−	solid
b	blue	*	star
w	white	:	dotted
k	black	−.	dashdot
		−−	dashed

4. The following sequence of commands also puts on a legend, a title, and relabels the $x-$ and $y-$axes: Try it!

```
x=linspace(-2,2);
y1=sin(x);
y2=x.^2;
plot(x,y1,'g*-',x,y2,'k-.');
title('Example One');
legend('The Sine Function','A Quadratic');
xlabel('Dollars');
ylabel('Sense');
```

5. Other Things: If you look at the plotting window from the last example, you'll see lots of things that you can do. For example, there's a zoom in and a zoom out feature. You can also edit the colors and symbols of your plot, and the title, legend and axis labels. Try them out!

A.6.2 Plotting in Three Dimensions

Matlab uses the `plot3` command to plot in three dimensions. We won't be using this feature here. To get more information, either type `help plot3` or refer to the Matlab Graphics Manual.

A.7 M-Files: Functions and Scripts

What is a Matlab Function? A Matlab function is a sequence of commands that uses some input variables and outputs some variables. The following is a very simple Matlab function:

```
function y=square(x)
%FUNCTION Y=SQUARE(X)
%This function inputs a number or an array, and
%  outputs the squares of the numbers.

y=x.^2;
```

You would type this in the editor, then save it as `square.m` (the filename must be the same name as the function, and it must use the `.m` extension).

You'll notice that the first line states "function". This is always the first line of a Matlab function. The remarks that follow the first line are very important. When you type `help square`, these three lines appear. So when you write your own functions, you should include comments so that you can remember how to use it.

The rest of the first line defines what the input variable is (x), and what the output variable is (y).

A Matlab function can produce multiple outputs. For example:

```
function [A,b]=randmatrix(n)
%FUNCTION [A,b]=RANDMATRIX(N)
%Produces an 2n x 2n random matrix A and a random
%column vector b.
nn=2*n;
A=rand(nn,nn);
b=rand(nn,1);
```

To call this function from Matlab, you would write, for example,

`[X,y]=randmatrix(10);`

You'll notice that after running this program, the variables internal to the function (in this case `nn`) disappear. This is one major difference between a *script* and a *function*:

- A **script file** is a text file with a sequence of Matlab commands. It is used by Matlab just as if you were typing the commands in from the keyboard. You should use a script file whenever you are experimenting in Matlab- it makes life a lot easier!

- A **function** in Matlab is like a subroutine in programming. That is, once the function has been called, all of its variables are local to that function- you cannot access them from the keyboard, and the variables are erased once the function is finished.

Both scripts and functions should have the `.m` file extension. We'll get more practice with these a little later.

A.7.1 Debugging Hints

Sometimes when we write a Matlab function, we'll want to stop its execution to see what the function is actually doing. There are a couple of ways to do this-one way is to use the debugger in Matlab's editor. The method below is also very useful.

The **keyboard** command temporarily halts the execution of a function and returns control to the keyboard, and you can use this to see what's happening in a function. To return the control back to Matlab, you would type **return**.

For example, let's turn the Newton's method script file into a function. Edit the file we created earlier as `newton1.m` so that it looks like this:

```
function [z,y,dy]=newton2(x,n)
%FUNCTION [y,dy]=newton2(x,n)
% performs Newton's Method on x*exp(x)-cos(x)
% using initial value x and n iterations.
% The output z gives the refined solution,
% the output y gives the function values and dy
% the corresponding derivatives.

for k=1:n
  y(k) = x*exp(x)-cos(x);
  dy(k) = (x+1)*exp(x)+sin(x);
  z(k) = x-(y(k)/dy(k));
  x = z(k);
end
keyboard
```

Now save the file as `newton2.m`. In the command window, type `help newton2`. You should see the help lines come up. Now run the function by typing in the command window: `[x,y,z]=newton2(0.2,5);`

The program should stop at the keyboard line. Type in `whos` to see the current active variables. NOTICE that the cursor in the command window has changed to `K>>`. This indicates that the keyboard command is active. Type in `return` to get back.

Something to think about: What will Matlab do if you call
`[x,y,z]=newton2(0.2,5);`
and then type in `[x,y,z]=newton2(0.2,5);` at the `K>>` prompt??

Other Hints:

- Always know what the matrix sizes are. If you think the input to a function is a column, you can ensure that it is either by checking the size with `size(x)` or by typing: `x=x(:);`
- You can also type the command: `dbstop if error`

  This will stop Matlab execution and point you to the line in your code where the error occurred. To turn this off, type `dbclear if error`

To see more options, type `help dbstop`

A.8 Exercise Set

1. Let x be a row. What happens if you type `plot(x)`?

2. Let A be a 4×3 matrix. What happens if you type `plot(A)`? Compare this with `plot(A')`.

3. Write a Matlab command to plot $y = e^{5x}$, where $-2 \leq x \leq 2$ using 30 points. Plot both the curve and the actual data points themselves, both in magenta.

4. Write a Matlab function to plot $y = \sin(x)$ in red, $y = \sin(2x)$ in black, and $y = \sin(3x)$ in green, all on the same plot. You can assume that $x \in [-4, 8]$.

5. Find the interpolating polynomial $P(t) = a + bt + ct^2$ for the data

$$(1, 12), (2, 15), (3, 16)$$

   That is, use Matlab to find a, b, c so that:

$$\begin{aligned} a + b \cdot 1 + c \cdot 1^2 &= 12 \\ a + b \cdot 2 + c \cdot 2^2 &= 15 \\ a + b \cdot 3 + c \cdot 3^2 &= 16 \end{aligned}$$

   Plot the data points as red stars and the polynomial as a solid magenta curve. When plotting the polynomial, use more than three domain points, like $t = linspace(0.5, 3.5)$.

6. A model equation for a child's systolic blood pressure p and weight w is given by:

$$a + b\ln(w) = p$$

   where a, b are the model parameters that we need to find using a least squares model. Use the following experimental data to estimate the systolic blood pressure of a child weighing 100 pounds:

w	44	61	81	113	131
p	91	98	103	110	112

   Plot the data (use $\ln(w)$ versus p) and the model equation. When plotting the model equation, use different data, like $t = linspace(40, 140)$.

7. Use the model equation

$$y = A\cos(x) + B\sin(x)$$

to fit the data: $(1, 7.9), (2, 5.4), (3, -0.9)$. That is, use Matlab to find the best A and B. Plot both the data points (as red stars) and the model (as a black curve). Hint: When plotting the model, use more domain points, like $x = linspace(0, 4)$.