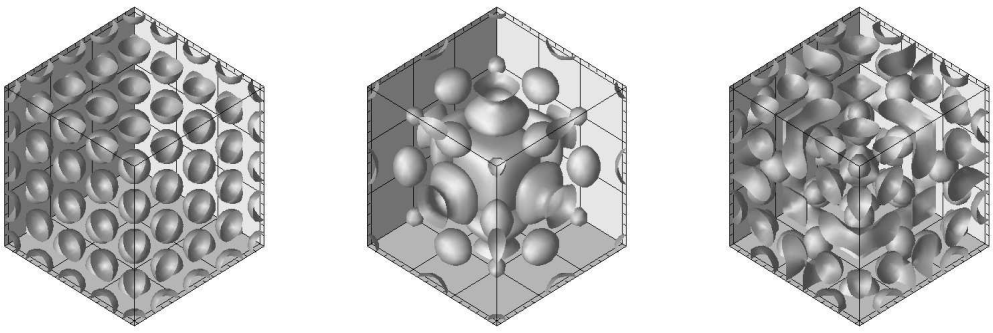
AMATH 582
Computation Methods for Data Analysis*

J. Nathan Kutz†

February 26, 2010

Abstract

This course is a survey of computational methods used for extracting meaningful results out of experimental or computational data. The central focus is on using a combination of spectral methods, statistics and linear algebra to analyze data and determine trends which are statistically significant.



*These notes are intended as the primary source of information for AMATH 582. The notes are incomplete and may contain errors. Any other use aside from classroom purposes and personal research please contact me at kutz@amath.washington.edu.

†Department of Applied Mathematics, Box 352420, University of Washington, Seattle, WA 98195-2420 (kutz@amath.washington.edu).

Contents

1 Statistical Methods and Their Applications

Our ultimate goal is to analyze highly generic data arising from applications as diverse as imaging, biological sciences, atmospheric sciences, or finance, to name a few specific examples. In all these application areas, there is a fundamental reliance on extracting meaningful trends and information from large data sets. Primarily, this is motivated by the fact that in many of these systems, the degree of complexity, or the governing equations, are unknown or impossible to extract. Thus one must rely on data, its statistical properties, and its analysis in the context of spectral methods and linear algebra.

1.1 Basic probability concepts

To understand the methods required for data analysis, a review of probability theory and statistical concepts is necessary. Many of the ideas presented in this section are intuitively understood to most students in the mathematical, biological, physical and engineering sciences. Regardless, a review will serve to refresh one with the concepts and will further help understand their implementation in MATLAB.

Sample space and events

Often one performs an experiment whose outcome is not known in advance. However, while the outcome is not known, suppose that the set of all possible outcomes is known. The set of all possible outcomes is the *sample space* of the experiment and is denoted by S. Some specific examples of sample spaces are following:

1. If the experiment consists of flipping a coin, then
$$S = \{H, T\}$$
where H denotes an outcome of heads and T denotes an outcome of tails.

2. If the experiment consists of flipping two coins, then
$$S = \{(H, H), (H, T), (T, H), (T, T)\}$$
where the outcome (H, H) denotes both coins being heads, (H, T) denotes the first coin being heads and the second tails, (T, H) denotes the first coin being tails and the second being heads, and (T, T) denotes both coins being tails.

3. If the experiment consists of tossing a die, then
$$S = \{1, 2, 3, 4, 5, 6\}$$
where the outcome i means that the number i appeared on the die, $i = 1, 2, 3, 4, 5, 6$.

Any subset of the sample space is referred to as an event, and it is denoted by E. For the sample spaces given above, we can also define an event.

1. If the experiment consists of flipping a single coin, then
$$E = \{H\}$$
denotes the event that a head appears on the coin.

2. If the experiment consists of flipping two coins, then
$$E = \{(H, H), (H, T)\}$$
denotes the event where a head appears on the first coin .

3. If the experiment consists of tossing a die, then
$$E = \{2, 4, 6\}$$
would be the event that an even number appears on the toss.

For any two events E_1 and E_2 of a sample space S, the *union* of these events $E_1 \cup E_2$ consists of all points which are either in E_1 or in E_2 or in both E_1 and E_2. Thus the event $E_1 \cup E_2$ will occur if either E_1 or E_2 occurs. For these same two events E_1 and E_2 we can also define their *intersection* as follows: E_1E_2 consists of all points which are both in E_1 and E_2, thus requiring that both E_1 and E_2 occur simultaneously. Figure 1 gives a simple graphical interpretation of the probability space S along with the events E_1 and E_2 and their union $E_1 \cup E_2$ and intersection E_1E_2.

To again make connection to the sample spaces and event examples given previously, consider the following unions and intersections:

1. If the experiment consists of flipping a single coin and $E_1 = \{H\}$ and $E_2 = \{T\}$ then
$$E_1 \cup E_2 = \{(H, T)\} = S$$
so that $E_1 \cup E_2$ is the entire sample space S and would occur if the coin flip produces either heads or tails.

2. If the experiment consists of flipping a single coin and $E_1 = \{H\}$ and $E_2 = \{T\}$ then
$$E_1E_2 = \emptyset$$
where the null event $\emptyset$ cannot occur since the coin cannot be both heads and tails simultaneously. Indeed if two events are such that $E_1E_2 = \emptyset$, then they are said to be mutually exclusive.

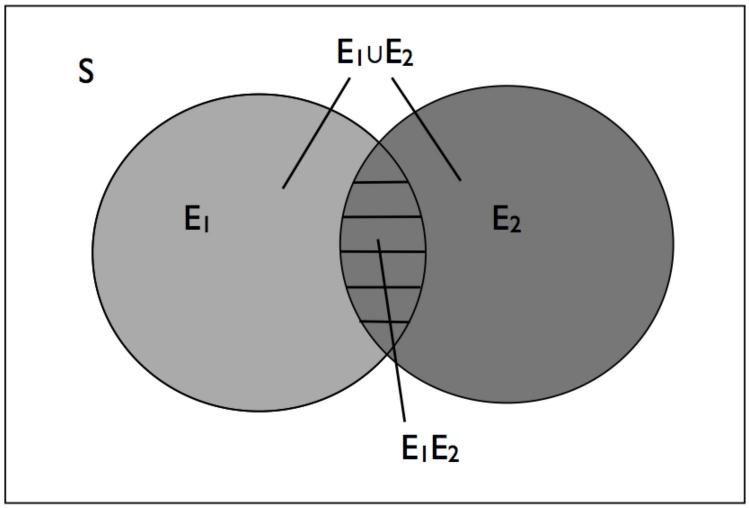


Figure 1: Venn diagram showing the sample space S along with the events E_1 and E_2. The union of the two events $E_1 \cup E_2$ is denoted by the gray regions while the intersection E_1E_2 is depicted in the cross-hatched region where the two events overlap.

3. If the experiment consists of tossing a die and if $E_1 = \{1, 3, 5\}$ and $E_2 = \{1, 2, 3\}$ then
$$E_1E_2 = \{1, 3\}$$
so that the intersection of these two events would be a roll of the die of either 1 or 3.

The union and intersection of more than two events can be easily generalized from these ideas. Consider a set of events $E_1, E_2, \cdots, E_N$, then the union is defined as $\cup_{n=1}^{N} E_n = E_1 \cup E_2 \cup \cdots \cup E_N$ and the intersection is defined as $\prod_{n=1}^{N} E_n = E_1E_2 \cdots E_N$. Thus an event which is in any of the E_n belongs to the union whereas an even needs to be in all E_n in order to be part of the intersection ($n = 1, 2, \cdots, N$). The compliment of an event E, denoted E^c, consists of all points in the sample space S not in E. Thus $E \cup E^c = S$ and $EE^c = \emptyset$.

Probabilities defined on events

With the concept of sample spaces S and events E in hand, the probability of an event $P(E)$ can be calculated. Thus the following conditions are necessary in defining a number to $P(E)$:

- $0 \leq P(E) \leq 1$
- $P(S) = 1$
- For any sequence of events $E_1, E_2, \cdots E_N$ that are mutually exclusive so that $E_i E_j = \emptyset$ when $i \neq j$ then

$$P\left(\cup_{n=1}^{N} E_n\right) = \sum_{n=1}^{N} P(E_n)$$

The notation $P(E)$ is the probability of the event E occurring. The concept of probability is quite intuitive and it can be quite easily applied to our previous examples.

1. If the experiment consists of flipping a fair coin, then

$$P(\{H\}) = P(\{T\}) = 1/2$$

   where H denotes an outcome of heads and T denotes an outcome of tails. On the other hand, a biased coin that was twice as likely to produce heads over tails would result in

$$P(\{(H\}) = 2/3, \;\; P(\{T\}) = 1/3$$

2. If the experiment consists of flipping two fair coins, then

$$P(\{(H, H)\} = 1/4$$

   where the outcome (H, H) denotes both coins being heads.

3. If the experiment consists of tossing a fair die, then

$$P(\{1\}) = P(\{2\}) = P(\{3\}) = P(\{4\}) = P(\{5\}) = P(\{6\}) = 1/6$$

   and the probability of producing an even number

$$P(\{2, 4, 6\}) = P(\{2\}) + P(\{4\}) + P(\{6\}) = 1/2.$$

This is a formal definition of the probabilities being functions of events on sample spaces. On the other hand, our intuitive concept of the probability is that if an event is repeated over and over again, then with probability one, the proportion of time and event E occurs is just $P(E)$. This is the frequentists viewpoint.

A few more facts should be placed here. First, since E and E^c are mutually exclusive and $E \cup E^c = S$, then $P(S) = P(E \cup E^c) = P(E) + P(E^c) = 1$. Additionally, we can compute the formula for $P(E_1 \cup E_2)$ which is the probability that either E_1 or E_2 occurs. From Fig. 1 it can be seen that the probability

$P(E_1) + P(E_2)$, which represents all points in E_1 and all points in E_2, is given by the gray regions. Notice in this calculation, that the overlap region is counted twice. Thus the intersection must be subtracted so that we find

$$P(E_1 \cup E_2) = P(E_1) + P(E_2) - P(E_1E_2)\,. \tag{1.1.1}$$

If the two events E_1 and E_2 are mutually exclusive then $E_1E_2 = \emptyset$ and $P(E_1 \cup E_2) = P(E_1) + P(E_2)$ (Note that $P(\emptyset) = 0$).

As an example, consider tossing two fair coins so that each of the points in the sample space $S = \{(H,H),(H,T),(T,H),(T,T)\}$ is equally likely to occur, or said another way, each has probability of $1/4$ to occur. Then let $E_1 = \{(H,H),(H,T)\}$ be the event where the first coin is a heads, and let $E_2 = \{(H,H),(T,H)\}$ be the event that the second coin is a heads. Then

$$\begin{aligned} P(E_1 \cup E_2) &= P(E_1) + P(E_2) - P(E_1E_2) \\ &= \frac{1}{2} + \frac{1}{2} - P\{(H,H)\} = \frac{1}{2} + \frac{1}{2} - \frac{1}{4} = \frac{3}{4} \end{aligned}$$

Of course, this could have also easily been computed directly from the union $P(E_1 \cup E_2) = P(\{(H,H),(H,T),(T,H)\}) = 3/4$.

Conditional probabilities

Conditional probabilities concerns, in some sense, the interdependency of events. Specifically, given that the event E_1 has occurred, what is the probability of E_2 occurring. This conditional probability is denoted as $P(E_2|E_1)$. A general formula for the conditional probability can derived from the following argument: If the event E_1 occurs, then in order for E_2 to occur, it is necessary that the actual occurrence be in the intersection E_1E_2. Thus in the conditional probability way of thinking, E_1 becomes our new sample space, and the event that E_1E_2 occurs will equal the probability of E_1E_2 relative to the probability of E_1. Thus we have the conditional probability definition

$$P(E_2|E_1) = \frac{P(E_2E_1)}{P(E_1)}\,. \tag{1.1.2}$$

The following are examples of some basic conditional probability events.

1. If the experiment consists of flipping two fair coins, what is the probability that both are heads given that at least one of them is heads? With the sample space $S = \{(H,H),(H,T),(T,H),(T,T)\}$ and each outcome as likely to occur as the other. Let E_2 denote the event that both coins are heads, and E_1 denote the event that at least one of them is heads, then

$$P(E_2|E_1) = \frac{P(E_2E_1)}{P(E_1)} = \frac{P(\{(H,H)\})}{P(\{(H,H),(H,T),(T,H)\})} = \frac{1/4}{3/4} = \frac{1}{3}$$

2. Suppose cards numbered one through ten are placed in a hat, mixed and drawn. If we are told that the number on the drawn card is at least five, then what is the probability that it is ten? In this case, E_2 is the probability ($= 1/10$) that the card is a ten, while E_1 is the probability ($= 6/10$) that it is at least a five. then

$$P(E_2|E_1) = \frac{1/10}{6/10} = \frac{1}{6}\,.$$

Independent events

Independent events will be of significant interest in this course. More specifically, the definition of independent events determines whether or not an event E_1 has any impact, or is correlated, with a second event E_2. By definition, two events are said to be *independent* if

$$P(E_1E_2) = P(E_1)P(E_2)\,. \tag{1.1.3}$$

By Equation (1.1.2), this implies that E_1 and E_2 are independent if

$$P(E_2|E_1) = P(E_2)\,. \tag{1.1.4}$$

In other words, the outcome of obtaining E_2 does not depend upon whether E_1 occurred. Two events that are not independent are said to be *dependent*.

As an example, consider tossing two fair die where each possible outcome is an equal $1/36$. Then let E_1 denote the event that the sum of the dice is six, E_2 denote the event that the first die equals four and E_3 denote the event that the sum of the dice is seven. Note that

$$\begin{aligned}
&P(E_1E_2) = P(\{(4,2)\}) = 1/36\\
&P(E_1)P(E_2) = 5/36 \times 1/6 = 5/216\\
&P(E_3E_2) = P(\{(4,3)\}) = 1/36\\
&P(E_3)P(E_2) = 1/6 \times 1/6 = 1/36
\end{aligned}$$

Thus we can find that E_2 and E_1 cannot be independent while E_3 and E_2 are, in fact, independent. Why is this true? In the first case where a total of six (E_2) on the die is sought, then the roll of the first die is important as it must be below six in order to satisfy this condition. On the other hand, a total of seven can be accomplished (E_3) regardless of the first die roll.

Bayes's Formula

Consider two events E_1 and E_2. Then the event E_1 can be expressed as follows

$$E_1 = E_1E_2 \cup E_1E_2^c\,. \tag{1.1.5}$$

This is true since in order for a point to be in E_1, it must either be in E_1 and E_2, or it must be in E_1 and not in E_2. Since the intersections E_1E_2 and $E_1E_2^c$ are obviously mutually exclusive, then

$$\begin{aligned} P(E_1) &= P(E_1E_2) + P(E_1E_2^c) \\ &= P(E_1|E_2)P(E_2) + P(E_1|E_2^c)P(E_2^c) \\ &= P(E_1|E_2)P(E_2) + P(E_1|E_2^c)(1 - P(E_2)) \end{aligned} \tag{1.1.6}$$

This states that the probability of the event E_1 is a weighted average of the conditional probability of E_1 given that E_2 has occurred and the conditional probability of E_1 given that E_2 has not occurred, with each conditional probability being given as much weight as the event it is conditioned on has of occurring. To illustrate the application of Bayes's formula, consider the two following examples:

1. Consider two boxes: the first containing two white and seven black balls, and the second containing five white and six black balls. Now flip a fair coin and draw a ball from the first or second box depending on whether you get heads or tails. What is the conditional probability that the outcome of the toss was heads given that a white ball was selected? For this problem, let W be the event that a white ball was drawn and H be the event that the coin came up heads. The solution to this problem involves the calculation of $P(H|W)$. This can be calculated as follows:

$$\begin{aligned} P(H|W) &= \frac{P(HW)}{P(W)} = \frac{P(W|H)P(H)}{P(W)} \\ &= \frac{P(W|H)P(H)}{P(W|H)P(H) + P(W|H^c)P(H^c)} \\ &= \frac{2/9 \times 1/2}{2/9 \times 1/2 + 5/11 \times 1/2} = \frac{22}{67} \end{aligned} \tag{1.1.7}$$

2. A laboratory blood test is 95% effective in determining a certain disease when it is present. However, the test also yields a false positive result for 1% of the healthy persons tested. If 0.5% of the population actually has the disease, what is the probability a person has the disease given that their test result is positive? For this problem, let D be the event that the person tested has the disease, and E the event that their test is positive. The problem involves calculation of $P(D|E)$ which can be obtained by

$$\begin{aligned} P(D|E) &= \frac{P(DE)}{P(E)} \\ &= \frac{P(E|D)P(D)}{P(E|D)P(D) + P(E|D^c)P(D^c)} \end{aligned}$$

$$= \frac{(0.95)(0.005)}{(0.95)(0.005)+(0.01)(0.995)}$$
$$= \frac{95}{294} \approx 0.323 \to 32\%$$

In this example, it is clear that if the disease is very rare so that $P(D) \to 0$, it becomes very difficult to actually test for it since even with a 95% accuracy, the chance of the false positives means that the chance of a person actually having the disease is quite small, making the test not so worth while. Indeed, if this calculation is redone with the probability that one in a million people of the population have the disease ($P(D) = 10^{-6}$), then $P(D|E) \approx 1 \times 10^{-4} \to 0.01\%$.

Equation (1.1.6) can be generalized in the following manner: suppose $E_1, E_2, \cdots, E_N$ are mutually exclusive events such that $\cup_{n=1}^{N} E_n = S$. Thus exactly one of the events E_n occurs. Further, consider an event H so that

$$P(H) = \sum_{n=1}^{N} P(HE_n) = \sum_{n=1}^{N} P(H|E_n)P(E_n)\,. \tag{1.1.8}$$

Thus for the given events E_n, one of which must occur, $P(H)$ can be computed by conditioning upon which of the E_n actually occurs. Said another way, $P(H)$ is equal to the weighted average of $P(H|E_n)$, each term being weighted by the probability of the event on which it is conditioned. Finally, using this result we can compute

$$P(E_n|H) = \frac{P(HE_n)}{P(H)} = \frac{P(H|E_n)P(E_n)}{\sum_{n=1}^{N} P(H|E_n)P(E_n)} \tag{1.1.9}$$

which is known as Bayes' formula.

1.2 Random variables and statistical concepts

Ultimately in probability theory we are interested in the idea of a *random variable*. This is typically the outcome of some experiment that has an undetermined outcome, but whose sample space S and potential events E can be characterized. In the examples already presented, the toss of a die and flip of a coin represent two experiments whose outcome is not known until the experiment is performed. A random variable is typically defined as some real-valued function on the sample space. The following are some examples.

1. Let X denote the random variable which is defined as the sum of two fair dice, then the random variable is assigned the values

$$P\{X = 2\} = P\{(1,1)\} = 1/36$$

$$
\begin{aligned}
P\{X=3\} &= P\{(1,2),(2,1)\} = 2/36 \\
P\{X=4\} &= P\{(1,3),(3,1),(2,2,)\} = 3/36 \\
P\{X=5\} &= P\{(1,4),(4,1),(2,3),(3,2)\} = 4/36 \\
P\{X=6\} &= P\{(1,5),(5,1),(2,4),(4,2),(3,3)\} = 5/36 \\
P\{X=7\} &= P\{(1,6),(6,1),(2,5),(5,2),(3,4),(4,3)\} = 6/36 \\
P\{X=8\} &= P\{(2,6),(6,2),(3,5),(5,3),(4,4)\} = 5/36 \\
P\{X=9\} &= P\{(3,6),(6,3),(4,5),(5,4)\} = 4/36 \\
P\{X=10\} &= P\{(4,6),(6,4),(5,5)\} = 3/36 \\
P\{X=11\} &= P\{(5,6),(6,5)\} = 2/36 \\
P\{X=12\} &= P\{(6,6)\} = 1/36\,.
\end{aligned}
$$

Thus the random variable X can take on integer values between two and twelve with the probability for each assigned above. Since the above events are all possible outcomes which are mutually exclusive, then

$$P\left(\cup_{n=2}^{12}\{X=n\}\right) = \sum_{n=2}^{12} P\{X=n\} = 1$$

Figure 2 illustrates the probability as a function of X as well as its distribution function.

2. Suppose a coin is tossed having probability p of coming up heads, until the first heads appears. Then define the random variable N which denotes the number of flips required to generate the first heads. Thus

$$
\begin{aligned}
P\{N=1\} &= P\{H\} = p \\
P\{N=2\} &= P\{(T,H)\} = (1-p)p \\
P\{N=3\} &= P\{(T,T,H)\} = (1-p)^2 p \\
&\vdots \\
P\{N=n\} &= P\{(T,T,\cdots,T,H)\} = (1-p)^{n-1}p
\end{aligned}
$$

and again the random variable takes on discrete values.

In the above examples, the random variables took on a finite, or countable, number of possible values. Such random variables are *discrete* random variables. The cumulative distribution function, or more simply the distribution function $F(\cdot)$ of the random variable X (See Fig. 2) is defined for any real number b $(-\infty < b < \infty)$ by

$$F(b) = P\{X \le b\}\,. \tag{1.2.1}$$

Thus the $F(b)$ denotes the probability of a random variable X taking on a value which is less than or equal to b. Some properties of the distribution function are as follows:

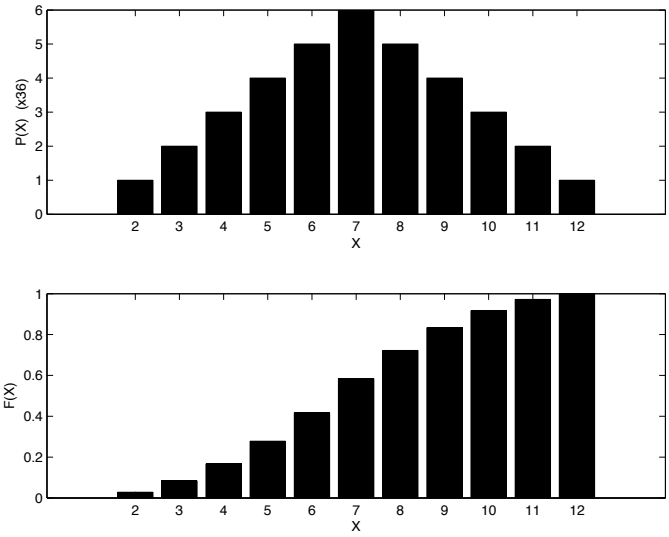


Figure 2: (a) Random variable probability assignments $P(X)$ for the sum of two fair dice, and (b) the distribution function $F(X)$ associated with the toss of two fair dice.

- $F(b)$ is a nondecreasing function of b,
- $\lim_{b\to\infty} F(b) = F(\infty) = 1$,
- $\lim_{b\to-\infty} F(b) = F(-\infty) = 0$.

Figure 2(b) illustrates these properties quite nicely. Any probability question about X can be answered in terms of the distribution function. The most common and useful example of this is to consider

$$P\{a < X \leq b\} = F(b) - F(a) \tag{1.2.2}$$

for all $a < b$. This result, which gives the probability of $X \in (a, b]$, is fairly intuitive as it involves first computing the probability that $X \leq b$ $(F(b))$ and then subtracting from this the probability that $X \leq a$ $(F(a))$.

For a discrete random variable, the overall probability function can be defined using the *probability mass function* of X as

$$p(a) = P\{X = a\}\,. \tag{1.2.3}$$

The probability mass function $p(a)$ is positive for at most a countable number of values a so that

$$p(x) = \begin{cases} > 0 & x = x_n \ (n = 1, 2, \cdots, N) \\ = 0 & x \neq x_n \end{cases} \tag{1.2.4}$$

Further, since X must take on one of the values of x_n, then

$$\sum_{n=1}^{N} p(x_n) = 1\,. \tag{1.2.5}$$

The cumulative distribution is defined as follows:

$$F(a) = \sum_{x_n<a} p(x_n)\,. \tag{1.2.6}$$

In contrast to discrete random variables, *continuous random variables* have an uncountable set of possible values. Many of the definitions and ideas presented thus far for discrete random variables are then easily transferred to the continuous context. Letting X be a continuous random variable, then there exists a non-negative function $f(x)$ defined for all real $x \in (-\infty, \infty)$ having the property that for any set of real numbers

$$P\{X \in B\} = \int_B f(x)dx\,. \tag{1.2.7}$$

The function $f(x)$ is called the *probability density function* of the random variable X. Thus the probability that X will be in B is determined by integrating over the set B. Since X must take on some value in $x \in (-\infty, \infty)$, then

$$P\{X \in (-\infty, \infty)\} = \int_{-\infty}^{\infty} f(x)dx = 1\,. \tag{1.2.8}$$

Any probability statement about X can be completely determined in terms of $f(x)$. If we are determine the probability that the event $B \in [a, b]$ then

$$P\{a \le X \le b\} = \int_a^b f(x)dx\,. \tag{1.2.9}$$

If in this calculation we let $a = b$, then

$$P\{X = a\} = \int_a^a f(x)dx = 0\,. \tag{1.2.10}$$

Thus the probability that a continuous random variable assumes a *particular* value is zero.

The cumulative distribution function $F(\cdot)$ can also be defined for a continuous random variable. Indeed, it has a simple relationship to the probability density function $f(x)$. Specifically,

$$F(a) = P\{X \in (-\infty, a]\} = \int_{-\infty}^{a} f(x)dx\,. \tag{1.2.11}$$

Thus the density is the derivative of the cumulative distribution function. An important interpretation of the density function is obtained by considering the following

$$P\left\{a - \frac{\epsilon}{2} \leq X \leq a + \frac{\epsilon}{2}\right\} = \int_{a-\epsilon/2}^{a+\epsilon 2} f(x)dx \approx \epsilon f(a)\,. \tag{1.2.12}$$

The *sifting* property of this integral illustrates that the density function $f(a)$ measures how likely it is that the random variable will be near a.

Having established key properties and definitions concerning discrete and random variables, it is illustrative to consider some of the most common random variables used in practice. Thus consider first the following discrete random variables:

1. **Bernoulli Random Variable**: Consider a trial whose outcome can either be termed a success or failure. Let the random variable $X = 1$ if there is a success and $X = 0$ if there is failure, then the probability mass function of X is given by

$$p(0) = P\{X = 0\} = 1 - p$$
$$p(1) = P\{X = 1\} = p$$

   where p $(0 \leq p \leq 1)$ is the probability that the trial is a success.

2. **Binomial Random Variable**: Suppose n independent trials are performed, each of which results in a success with probability p and failure with probability $1-p$. Denote X as the number of successes that occur in n such trials. Then X is is a binomial random variable with parameters (n, p) and the probability mass function is given by

$$p(j) = \left(\begin{array}{c} n \\ j \end{array} \right) p^j (1-p)^{n-j}$$

   where $\left(\begin{array}{c} n \\ j \end{array} \right) = n!/((n-j)!j!)$ is the number of different groups of j objects that can be chosen from a set of n objects. For instance, if $n = 3$ and $j = 2$, then this binomial coefficient is 3!/(1!2!)=3 representing the fact that there are three ways to have successes in three trials: $(s, s, f), (s, f, s), (f, s, s)$.

3. **Poisson Random Variable**: A Poisson random variable with parameter λ is defined such that

$$p(n) = P\{X = n\} = e^{-\lambda}\frac{\lambda_n}{n!} \quad n = 0, 1, 2, \cdots$$

   for some $\lambda > 0$. The Poisson random variable has an enormous range of applications across the sciences as it represents, to some extent, the exponential distribution of many systems.

In a similar fashion, a few of the more common continuous random variables can be highlighted here:

1. **Uniform Random Variable:** A random variable is said to be uniformly distributed over the interval $(0, 1)$ if its probability density function is given by
$$f(x) = \begin{cases} 1, & 0 < x < 1 \\ 0, & \text{otherwise} \end{cases} .$$
More generally, the definition can be extended to any interval (α, β) with the density function being given by
$$f(x) = \begin{cases} \frac{1}{\beta-\alpha}, & \alpha < x < \beta \\ 0, & \text{otherwise} \end{cases} .$$
The associated distribution function is then given by
$$F(a) = \begin{cases} 0, & a < \alpha \\ \frac{a-\alpha}{\beta-\alpha}, & \alpha < a < \beta \\ 1, & a \geq \beta \end{cases}$$

2. **Exponential Random Variable**: Given a positive constant $\lambda > 0$, the density function for this random variable is given by
$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & x \leq 0 \end{cases}$$
This is the continuous version of the Poisson distribution considered for the discrete case. Its distribution function is given by
$$F(a) = \int_0^a \lambda e^{-\lambda x} dx = 1 - e^{-\lambda a}, \;\; a > 0 \, .$$

3. **Normal Random Variable**: Perhaps the most important of them all is the normal random variable with parameters μ and σ^2 defined by
$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}, \;\; -\infty < x < \infty, \, .$$
Although we have not defined them yet, the parameters μ and σ refer to the mean and variance of this distribution respectively. Further, the Gaussian shape produces the ubiquitous bell-shaped curve that is always talked about in the context of large samples, or perhaps grading. Its distribution is given by the error function $erf(a)$, one of the most common functions for look-up tables in mathematics.

Expectation, moments and variance of random variables

The expectation value of a random variable is defined as follows:

$$E[X] = \sum_{x:p(x)>0} xp(x)$$

for a discrete random variable, and

$$E[X] = \int_{-\infty}^{\infty} xf(x)dx$$

for a continous random variable. Both of these definitions illustrate the that the expectation is a weighted average whose weight is the probability that X assumes at that value. Typically, the expectation is interpreted as the average value of the random variable. Some examples will serve to illustrate this point:

1. The expected value $E[X]$ of the roll of a fair die is given by

$$E[X] = 1(1/6) + 2(1/6) + 3(1/6) + 4(1/6) + 5(1/6) + 6(1/6) = 7/2$$

2. The expectation $E[X]$ of a Bernoulli variable is

$$E[X] = 0(1-p) + 1(p) = p$$

   showing that the expected number of successes in a single trial is just the probability that the trial will be a success.

3. The expectation $E[X]$ of a binomial or Poisson random variable is

$$\begin{aligned} E[X] &= np \quad \text{binomial} \\ E[X] &= \lambda \quad \text{Poisson} \end{aligned}$$

4. The expectation of a continuous uniform random variable is

$$E[X] = \int_{\alpha}^{\beta} \frac{x}{\beta - \alpha} dx = \frac{\alpha + \beta}{2}$$

5. The expectation $E[X]$ of an exponential or normal random variable is

$$\begin{aligned} E[X] &= 1/\lambda \quad \text{exponential} \\ E[X] &= \mu \quad \text{normal} \end{aligned}$$

The idea of expectation can be generalized beyond the standard expectation $E[X]$. Indeed, we can consider the expectation of any function of the random

variable X. Thus to compute the expectation of some function $g(X)$, the discrete and continuous expectations become:

$$E[g(X)] = \sum_{x:p(x)>0} g(x)p(x) \tag{1.2.13a}$$

$$E[g(X)] = \int_{-\infty}^{\infty} g(x)f(x)dx \tag{1.2.13b}$$

This also then leads us to the idea of higher-moments of a random variable, i.e. $g(X) = X^n$ so that

$$E[X^n] = \sum_{x:p(x)>0} x^n p(x) \tag{1.2.14a}$$

$$E[X^n] = \int_{-\infty}^{\infty} x^n f(x)dx\,. \tag{1.2.14b}$$

The first moment is essentially the mean, the second moment will be related to the variance, the third and fourth moments measure the *skewness* (asymmetry of the probability distribution) and the *kurtosis* (the flatness or sharpness) of the distribution.

Now that these probability ideas are in place, the quantifies of greatest and most practical interest can be considered, namely the ideas of mean and variance (and standard deviation, which is defined as the square root of the variance). The mean has already been defined, while the variance of a random variable is given by

$$Var[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2 \tag{1.2.15}$$

The second form can be easily manipulated from the first. Note that depending upon your scientific community of origin $E[X] = \bar{x} =< x >$. It should be noted that the normal distribution has $Var(X) = \sigma^2$, standard deviation σ, and skewness of zero.

Joint probability distribution and covariance

The mathematical framework is now in place to discuss what is perhaps, for this course, the most important use of our probability and statistical thinking, namely the relation between two or more random variables. Ultimately, this assesses whether or not two variables depend upon each other. A joint cumulative probability distribution function of two random variables X and Y is given by

$$F(a,b) = P\{X \le a, Y \le b\}, \;\; -\infty < a, b < \infty\,. \tag{1.2.16}$$

The distribution of X or Y can be obtained by considering

$$F_X(a) = P\{X \le a\} = P\{X \le a, Y \le \infty\} = F(a,\infty) \tag{1.2.17a}$$

$$F_Y(b) = P\{Y \le b\} = P\{Y \le b, X \le \infty\} = F(\infty,b)\,. \tag{1.2.17b}$$

The random variables X and Y are jointly continuous if there exists a function $f(x,y)$ defined for real x and y so that for any set of real numbers A and B

$$P\{X \in A, Y \in B\} = \int_A \int_B f(x,y)dxdy\,. \tag{1.2.18}$$

The function $f(x,y)$ is called the joint probability density function of X and Y.

The probability density of X or Y can be extracted from $f(x,y)$ as follows:

$$P\{X\in A\} = P\{X\in A, Y\in(-\infty,\infty)\} = \int_{-\infty}^{\infty}\int_A f(x,y)dxdy = \int_A f_X(x)dx \tag{1.2.19}$$

where $f_X(x) = \int_{-\infty}^{\infty} f(x,y)dy$. Similarly, the probability density function of Y is $f_Y(y) = \int_{-\infty}^{\infty} f(x,y)dx$. As previously, to determine a function of the random variables X and Y, then

$$E[g(X,Y)] = \begin{cases} \sum_y \sum_x g(x,y)p(x,y) & \text{discrete} \\ \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x,y)f(x,y)dxdy & \text{continuous} \end{cases} \tag{1.2.20}$$

Jointly distributed variables lead naturally to a discussion of whether the variables are independent. Two random variables are said to be independent if

$$P\{X \le a, Y \le b\} = P\{X \le a\}P\{Y \le b\} \tag{1.2.21}$$

In terms of the joint distribution function F of X and Y, independence is established if $F(a,b) = F_X(a)F_Y(b)$ for all a, b. The independence assumption essentially reduces to the following:

$$f(x,y) = f_X(x)f_Y(y)\,. \tag{1.2.22}$$

This further implies

$$E[g(X)h(Y)] = E[g(X)]E[h(Y)]\,. \tag{1.2.23}$$

Mathematically, the case of independence allows for essentially a separation of the random variables X and Y.

What is, to our current purposes, more interesting is the idea of dependent random variables. For such variables, the concept of covariance can be introduced. This is defined as

$$\mathrm{Cov}(X,Y) = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X]E[Y]\,. \tag{1.2.24}$$

If two variables are independent then $\mathrm{Cov}(X,Y) = 0$. Alternatively, as the $\mathrm{Cov}(X,Y)$ approaches unity, it implies that X and Y are essentially directly, or strongly, correlated random variables. This will serve as a basic measure of the statistical dependence of our data sets. It should be noted that although two independent variables implies $\mathrm{Cov}(X,Y) = 0$, the converse is not necessary true, i.e. $\mathrm{Cov}(X,Y) = 0$ does not imply independence of X and Y.

MATLAB commands

The following are some of the basic commands in MATLAB that we will use for computing the above mentioned probability and statistical concepts.

1. **rand(m,n)**: This command generates an $m \times n$ matrix of random numbers from the uniform distribution on the unit interval.
2. **randn(m,n)**: This command generates an $m \times n$ matrix of normally distributed random numbers with zero mean and unit variance.
3. **mean(X)**: This command returns the mean or average value of each column of the matrix **X**.
4. **var(X)**: This computes the variance of each row of the matrix **X**. To produce and unbiased estimator, the variance of the population is normalized by $N - 1$ where N is the sample size.
5. **std(X)**: This computes the standard deviation of each row of the matrix **X**. To produce and unbiased estimator, the standard deviation of the population is normalized by $N - 1$ where N is the sample size.
6. **cov(X)**: If **X** is a vector, it returns the variance. For matrices, where each row is an observation, and each column a variable, the command returns a covariance matrix. Along the diagonal of this matrix is the variance of each column.

1.3 Hypothesis testing and statistical significance

With the key concepts of probability theory in hand, statistical testing of data can now be pursued. But before doing so, a few key results concerning a large number of random variables or large number of trials should be explicitly stated.

Limit theorems

Two theorems of probability theory are important to establish before discussing testing of hypothesis via statistics: they are the *strong law of large numbers* and the *central limit theorem*. They are, perhaps, the most well-known and most important practical results in probability theory.

Theorem: *Strong law of large numbers:* Let $X_1, X_2, \cdots$ be a sequence of independent random variables having a common distribution, and let $E[X_j] = \mu$. Then, with probability one,

$$\frac{X_1 + X_2 + \cdots + X_n}{n} \to \mu \text{ as } n \to \infty \tag{1.3.1}$$

As an example of the strong law of large numbers, suppose that a sequence of independent trials are performed. Let E be a fixed event and denote by $P(E)$ the probability that E occurs on any particular trial. Letting

$$X_j = \begin{cases} 1, & \text{if } E \text{ occurs on the } j\text{th trial} \\ 0, & \text{if } E \text{ does not occur on the } j\text{th trial} \end{cases} \tag{1.3.2}$$

The theorem states that with probability one

$$\frac{X_1 + X_2 + \cdots + X_n}{n} \to E[X] = P(E)\,. \tag{1.3.3}$$

Since $X_1 + X_2 + \cdots + X_n$ represents the number of times that the event E occurs in n trials, this result can be interpreted as the fact that, with probability one, the limiting proportion of time that the event E occurs is just $P(E)$.

Theorem: *Central Limit Theorem:* Let $X_1, X_2, \cdots$ be a sequence of independent random variables each with mean μ and variance σ^2. Then the distribution of the quantity

$$\frac{X_1 + X_2 + \cdots + X_n - n\mu}{\sigma\sqrt{n}} \tag{1.3.4}$$

tends to the standard normal distribution as $n \to \infty$ so that

$$P\left\{\frac{X_1 + X_2 + \cdots + X_n - n\mu}{\sigma\sqrt{n}} \le a\right\} \to \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{a} e^{-x^2/2} dx \tag{1.3.5}$$

as $n \to \infty$. The most important part of this result: it holds for *any* distribution of X_j. Thus for a large enough sampling of data, i.e. a sequence of independent random variables that goes to infinity, one should observe the ubiquitous bell-shaped probability curve of the normal distribution. This is a powerful result indeed, provided you have a large enough sampling. And herein lies both its power, and susceptibility to overstating results. The fact is, the normal distribution has some very beautiful mathematical properties, leading us to consider this central limit almost exclusively when testing for statistical significance.

As an example, consider a binomially distributed random variable X with parameters n (n is the number of events) and p (p is the probability of a success of an event). The distribution of

$$\frac{X - E[X]}{\sqrt{Var[X]}} = \frac{X - np}{\sqrt{np(1-p)}} \tag{1.3.6}$$

approaches the standard normal distribution as $n \to \infty$. Empirically, it has been found that the normal approximation will, in general, be quite good for values of n satisfying $np(1-p) \ge 10$.

Statistical decisions

In practice, the power of probability and statistics comes into play in making decisions, or drawing conclusions, based upon a limited sampling or information information of an entire *population* (of data). Indeed, it is often impossible or impractical to examine the entire population. Thus a sample of the population is considered instead. This is called the *sample*. The goal is to infer facts or trends about the entire population with this limited sample size. The process of obtaining samples is called *sampling*, while process of drawing conclusions about this sample is called *statistical inference*. Using these ideas to make decisions about a population based upon the sample information is termed *statistical decisions*.

In general, these definitions are quite intuitive and also quite well known to us. Statistical inference, for instance, is used extensively in polling data for elections. In such a scenario, county, state, and national election projections are made well before the election based upon sampling a small portion of the population. Statistical decision making on the other hand is used extensively to determine whether a new medical treatment or drug is, in fact, effective in curing a disease or ailment.

In what follows, the focus will be upon statistical decision making. In attempting to reach a decision, it is useful to make assumptions or guesses about the population involved. Much like an analysis proof, you can reach a conclusion by assuming something is true and proving it to be so, or assuming the opposite is true and proving that it is false, for instance, by a counter example. There is no difference with statistical decision making. In this case, all the statistical tests are hypothesis driven. Thus a *statistical hypothesis* is proposed and its validity investigated. As with analysis, some hypothesis are explicitly constructed for the purpose of rejection, or nullifying the hypothesis. For example, to decide whether a coin is fair or loaded, the hypothesis is formulated that the coin is fair with $p = 0.5$ as the probability of heads. To decide if one procedure is better or worse than another, the hypothesis formulated is that there is no difference between the procedures. Such hypotheses are called *null hypothesis* and denoted by H_0. Any hypothesis which differs from a given hypothesis is called an *alternative hypothesis* and is denoted by H_1.

Tests for statistical significance

The ultimate goal in statistical decision making is to develop tests that are capable of allowing us to accept or reject a hypothesis with a certain, quantifiable confidence level. Procedures that allow us to do this are called *tests of hypothesis, tests of significance* or *rules of decision*. Overall, they fall within the aegis of the ideas of statistical significance.

In making such statistical decisions, there is some probability that erroneous conclusion will be reached. Such errors are classified as follows:

- **Type I error**: If a hypothesis is rejected when it should be accepted.
- **Type II error**: If a hypothesis is accepted when it should be rejected.

As an example, suppose a fair coin ($p = 0.5$ for achieving heads or tails) is tossed 100 times and unbelievably, 100 heads in a row results. Any statistical test based upon assuming that the coin is fair is going to be rejected and a Type I error will occur. If on the other hand, a loaded coin twice as likely to produce a heads than a tails ($p = 2/3$ for achieving a heads and $1 - p = 1/3$ for producing a tails) is tossed 100 times and heads and tails both appear 50 times, then a hypothesis about a biased coin will be rejected and a Type II error will be made.

What is important in making the statistical decision is the *level of significance* of the test. This probability is often denoted by α, and in practice it is often taken to be 0.05 or 0.01. These correspond to a 5% or 1% level of significance, or alternatively, we are 95% or 99% confident respectively that our decision is correct. Basically, in these tests, we will be wrong 5% or 1% of the time.

Hypothesis testing with the normal distribution

For large samples, the central limit theorem asserts that the statistics has a normal distribution (or nearly so) with mean μ_s and standard deviation σ_s. Thus many statistical decision tests are specifically geared to the normal distribution. Consider the normal distribution with mean μ and variance σ^2 such that the density is given by

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} \quad -\infty < x < \infty \tag{1.3.7}$$

and where the cumulative distribution is given by

$$F(x) = P(X \leq x) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{x} e^{-(x-\mu)^2/2\sigma^2} dx\,. \tag{1.3.8}$$

With the change of variable

$$Z = \frac{X - \mu}{\sigma} \tag{1.3.9}$$

the *standard normal density function* is given by

$$f(z) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \tag{1.3.10}$$

with the corresponding distribution function (related to the error function)

$$F(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-u^2/2} du = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{z}{\sqrt{2}}\right)\right]. \tag{1.3.11}$$

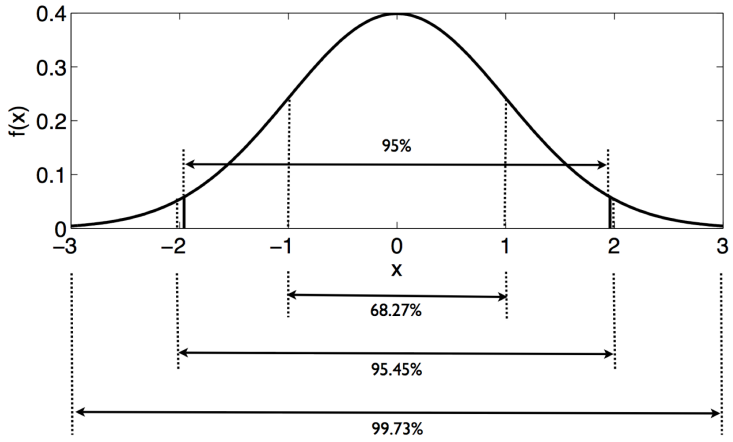


Figure 3: Standard normal distribution curve $f(x)$ with the first, second and third standard deviation markers below the graph along with the corresponding percentage of the population contained within each. Above the graph, markers are placed at $x = -1.96$ and $x = 1.96$ within which 95% of the population is contained. Outside of this range, a statistical test with 95% confidence would result in a conclusion that a given hypothesis is false.

The standard normal density function has mean zero ($\mu = 0$) and unit variance ($\sigma = 1$). The standard normal curve is shown in Fig. 3. In this graph, several key features are demonstrated. On the bottom, the first, second, and third standard deviation markers indicate that 68.72%, 95.45% and 99.73% percent of the population are contained within the respective standard deviation markers. Above the graph, a line is drawn at $x = -1.96$ and $x = 1.96$ indicated the 95% confidence markers. Indeed, the graph indicates two key regions: the *region of acceptance of the hypothesis* for $x \in [-1.96, 1.96]$ and the *region of nonsignificance* for $|x| > 1.96$.

In the above example, interest was given to the extreme values of the statistics, or the corresponding z value on both sides of the mean. Thus both tails of the distribution mattered. This is then called a *two-tailed test*. However, some hypothesis tests are only concerned with one side of the mean. Such a test, for example, may be evaluating whether one process is better than another versus whether it is better or worse. This is called a *one-tailed test*. Some examples will serve to illustrate the concepts. It should also be noted that a variety of tests can be formulated for determining confidence intervals, including the Student's t-test (involving measurements on the mean), the chi-square test (involving tests on the variance), and the F-test (involving ratios of variances). These will not be discussed in detail here.

Example: A fair coin. Design a decision rule to test the null hypothesis H_0 that a coin is fair if a sample of 64 tosses of the coin is taken and if the level of significance is 0.05 or 0.01.

The coin toss is an example of a binomial distribution as a success (heads) and failure (tails). Under the null hypothesis that the coin is fair ($p = 0.5$ for achieving either heads or tails) the binomial mean and standard deviation can be easily calculated for n events:

$$\mu = np = 64 \times 0.5 = 32$$
$$\sigma = \sqrt{np(1-p)} = \sqrt{64 \times 0.5 \times 0.5} = 4\,.$$

The statistical decision must now be formulated in terms of the standard normal variable Z where

$$Z = \frac{X - \mu}{\sigma} = \frac{X - 32}{4}\,.$$

For the hypothesis to hold with 95% confidence (or 99% confidence), then $z \in [-1.96, 1.96]$. Thus the following must be satisfied:

$$-1.96 \leq \frac{X - 32}{4} \leq 1.96 \;\rightarrow\; 24.16 \leq X \leq 39.84$$

Thus if in a trial of 64 flips heads appears between 25 and 39 times inclusive, the hypothesis is true with 95% confidence. For 99% confidence, heads must appear between 22 and 42 times inclusive.

Example: Class scores. An examination was given to two classes of 40 and 50 students respectively. In the first class, the mean grade was 74 with a standard deviation of 8, while in the second class the mean was 78 with standard deviation 7. Is there a statistically significant difference between the performance of the two classes at a level of significance of 0.05?

Consider that the two classes come from two populations having respective means μ_1 and μ_2. Then the following null hypothesis and alternative hypothesis can be formulated.

$$H_0 : \mu_1 = \mu_2, \text{ and the difference is merely due to chance}$$
$$H_1 : \mu_1 \neq \mu_2, \text{ and there is a significant difference between classes}$$

Under the null hypothesis H_0, both classes come from the same population. The mean and standard deviation of the differences of these is given by

$$\mu_{\bar{X}_1 - \bar{X}_2} = 0$$
$$\sigma_{\bar{X}_1 - \bar{X}_2} = \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}} = \sqrt{\frac{8^2}{40} + \frac{7^2}{50}} = 1.606$$

where the standard deviations have been used as estimates for σ_1 and σ_2. The problem is thus formulated in terms of a new variable representing the difference of the means

$$Z = \frac{\bar{X}_1 - \bar{X}_2}{\sigma_{\bar{X}_1 - \bar{X}_2}} = \frac{74 - 78}{1.606} = -2.49$$

(a) For a two-tailed test, the results are significant if $|Z| > 1.96$. Thus it must be concluded that there is a significant difference in performance to the two classes with the second class probably being better
(b) For the same test with 0.01 significance, then the results are significant for $|Z| > 2.58$ so that at a 0.01 significance level there is no statistical difference between the classes.

Example: Rope breaking (Student's t-distribution). A test of strength of 6 ropes manufactured by a company showed a breaking strength of 7750lb and a standard deviation of 145lb, whereas the manufacturer claimed a mean breaking strength of 8000lb. Can the manufacturers claim be supported at a significance level of 0.05 or 0.01?

The decision is between the hypotheses

$$H_0 : \mu = 8000\text{lb}, \text{ the manufacturer is justified}$$
$$H_1 : \mu < 8000\text{lb}, \text{ the manufacturer's claim is unjustified}$$

where a one-tailed test is to be considered. Under the null hypothesis H_0 and the small sample size, the Student's t-test can be used so that

$$T = \frac{\bar{X} - \mu}{S}\sqrt{n-1} = \frac{7750 - 8000}{145}\sqrt{6-1} = -3.86$$

As $n \to \infty$, the Student's t-distribution goes to the normal distribution. But in this case of a small sample, the degree of freedom is of this distribution is 6-1=5 and a 0.05 (0.01) significance is achieved with $T > -2.01$ (or $T > -3.36$). In either case, with $T = -3.86$ it is extremely unlikely that the manufacture's claim is justified.

2 Time-Frequency Analysis: Fourier Transforms and Wavelets

Fourier transforms, and more generally, spectral transforms, are one of the most powerful and efficient techniques for solving a wide variety of problems arising the physical, biological and engineering sciences. The key idea of the Fourier transform, for instance, is to represent functions and their derivatives as sums of cosines and sines. This operation can be done with the Fast-Fourier transform

(FFT) which is an $O(N \log N)$ operation. Thus the FFT is faster than most linear solvers of $O(N^2)$. The basic properties and implementation of the FFT will be considered here along with other time-frequency analysis techniques, including wavelets.

2.1 Basics of Fourier Series and the Fourier Transform

Fourier introduced the concept of representing a given function by a trigonometric series of sines and cosines:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos nx + b_n \sin nx) \quad x \in (-\pi, \pi]\,. \tag{2.1.1}$$

At the time, this was a rather startling assertion, especially as the claim held even if the function $f(x)$, for instance, was discontinuous. Indeed, when working with such series solutions, many questions arise concerning the convergence of the series itself. Another, of course, concerns the determination of the expansion coefficients a_n and b_n. Assume for the moment that Eq. (2.1.1) is a uniformly convergent series, then multiplying by $\cos mx$ gives the following uniformly convergent series

$$f(x)\cos mx = \frac{a_0}{2}\cos mx + \sum_{n=1}^{\infty} (a_n \cos nx \cos mx + b_n \sin nx \cos mx)\,. \tag{2.1.2}$$

Integrating both sides from $x \in [-\pi, \pi]$ yields the following

$$\int_{-\pi}^{\pi} f(x)\cos mx dx = \frac{a_0}{2}\int_{-\pi}^{\pi} \cos mx dx + \sum_{n=1}^{\infty}\left(a_n \int_{-\pi}^{\pi} \cos nx \cos mx dx + b_n \int_{-\pi}^{\pi} \sin nx \cos mx dx\right). \tag{2.1.3}$$

But the sine and cosine expressions above are subject to the following *orthogonality* properties.

$$\int_{-\pi}^{\pi} \sin nx \cos mx dx = 0 \quad \forall\ n, m \tag{2.1.4a}$$

$$\int_{-\pi}^{\pi} \cos nx \cos mx dx = \begin{cases} 0 & n \neq m \\ \pi & n = m \end{cases} \tag{2.1.4b}$$

$$\int_{-\pi}^{\pi} \sin nx \sin mx dx = 0 \begin{cases} 0 & n \neq m \\ \pi & n = m \end{cases}\,. \tag{2.1.4c}$$

Thus we find the following formulas for the coefficients a_n and b_n

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos nx dx \quad n \geq 0 \tag{2.1.5a}$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin nx dx \quad n > 0 \tag{2.1.5b}$$

As L in this case is pi

which are called the Fourier coefficients.

The expansion (2.1.1) must, by construction, produce 2π-periodic functions since they are constructed form sines and cosines on the interval $x \in (-\pi, \pi]$. In addition to these 2π-periodic solutions, one can consider expanding a function $f(x)$ defined on $x \in (0, \pi]$ as an *even periodic function* on $x \in (-\pi, \pi]$ by employing only the cosine portion of the expansion (2.1.1) or as an *odd periodic function* on $x \in (-\pi, \pi]$ by employing only the sine portion of the expansion (2.1.1).

Using these basic ideas, the complex version of the expansion produces the Fourier series on the domain $x \in [-L, L]$ is given by

$$f(x) = \sum_{-\infty}^{\infty} c_n e^{in\pi x/L} \quad x \in [-L, L] \tag{2.1.6}$$

with the corresponding Fourier coefficients

$$c_n = \frac{1}{2L} \int_{-L}^{L} f(x) e^{-in\pi x/L} dx \,. \tag{2.1.7}$$

Although the Fourier series is now complex, the function $f(x)$ is still assumed to be real. Thus the following observations are of note: (i) c_0 is real and $c_{-n} = c_n*$, (ii) if $f(x)$ is even, all c_n are real, and (iii) if $f(x)$ is odd, $c_0 = 0$ and all c_n are purely imaginary. These results follow from Euler's identity: $\exp(\pm ix) = \cos x \pm i \sin x$. The convergence properties of this representation will not be considered here except that it will be noted that at discontinuity of $f(x)$, the Fourier series converges to the mean of the left and right value of the discontinuity.

The Fourier Transform

The *Fourier Transform* is an integral transform defined over the entire line $x \in [-\infty, \infty]$. The Fourier transform and its inverse are defined as

gives a function of the wavenumber k

$$F(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f(x) dx \tag{2.1.8a}$$

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ikx} F(k) dk \,. \tag{2.1.8b}$$

There are other equivalent definitions. However, this definition will serve to illustrate the power and functionality of the Fourier transform method. We again note that formally, the transform is over the entire real line $x \in [-\infty, \infty]$ whereas our computational domain is only over a finite domain $x \in [-L, L]$. Further, the Kernel of the transform, $\exp(\pm ikx)$, describes oscillatory behavior. Thus the Fourier transform is essentially an eigenfunction expansion over all continuous wavenumbers k. And once we are on a finite domain $x \in [-L, L]$, the continuous eigenfunction expansion becomes a discrete sum of eigenfunctions and associated wavenumbers (eigenvalues).

Derivative Relations

The critical property in the usage of Fourier transforms concerns derivative relations. To see how these properties are generated, we begin by considering the Fourier transform of $f'(x)$. We denote the Fourier transform of $f(x)$ as $\widehat{f(x)}$. Thus we find

$$\widehat{f'(x)} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f'(x) dx = f(x) e^{-ikx}|_{-\infty}^{\infty} + \frac{ik}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f(x) dx \,. \tag{2.1.9}$$

Assuming that $f(x) \to 0$ as $x \to \pm\infty$ results in

$$\widehat{f'(x)} = \frac{ik}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f(x) dx = ik\widehat{f(x)} \,. \tag{2.1.10}$$

Thus the basic relation $\widehat{f'} = ik\widehat{f}$ is established. It is easy to generalize this argument to an arbitrary number of derivatives. The final result is the following relation between Fourier transforms of the derivative and the Fourier transform itself

$$\widehat{f^{(n)}} = (ik)^n \widehat{f} \,. \tag{2.1.11}$$

This property is what makes Fourier transforms so useful and practical.

As an example of the Fourier transform, consider the following differential equation

$$y'' - \omega^2 y = -f(x) \quad x \in [-\infty, \infty] \,. \tag{2.1.12}$$

We can solve this by applying the Fourier transform to both sides. This gives the following reduction

$$\begin{aligned} &\widehat{y''} - \omega^2 \widehat{y} = -\widehat{f} \\ &-k^2 \widehat{y} - \omega^2 \widehat{y} = -\widehat{f} \\ &(k^2 + \omega^2)\widehat{y} = \widehat{f} \\ &\widehat{y} = \frac{\widehat{f}}{k^2 + \omega^2} \,. \end{aligned} \tag{2.1.13}$$

To find the solution $y(x)$, we invert the last expression above to yield

$$y(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ikx} \frac{\widehat{f}}{k^2 + \omega^2} dk \,. \tag{2.1.14}$$

This gives the solution in terms of an integral which can be evaluated analytically or numerically.

fast fourier transform is an ALGORITHM for quickly performing forward and backward fourier transforms. As seen above, these transforms can be used for solving systems of differential equations.

The Fast Fourier Transform

The Fast Fourier transform routine was developed specifically to perform the forward and backward Fourier transforms. In the mid 1960s, Cooley and Tukey developed what is now commonly known as the FFT algorithm [3]. Their algorithm was named one of the top ten algorithms of the 20th century for one reason: the operation count for solving a system dropped to $O(N \log N)$. For N large, this operation count grows almost linearly like N. Thus it represents a great leap forward from Gaussian elimination and LU decomposition. The key features of the FFT routine are as follows:

1. It has a low operation count: $O(N \log N)$.
2. It finds the transform on an interval $x \in [-L, L]$. Since the integration Kernel $\exp(ikx)$ is oscillatory, it implies that the solutions on this finite interval have periodic boundary conditions.
3. The key to lowering the operation count to $O(N \log N)$ is in discretizing the range $x \in [-L, L]$ into 2^n points, i.e. the number of points should be $2, 4, 8, 16, 32, 64, 128, 256, \cdots$.
4. The FFT has excellent accuracy properties, typically well beyond that of standard discretization schemes.

We will consider the underlying FFT algorithm in detail at a later time. For more information at the present, see [3] for a broader overview.

The practical implementation of the mathematical tools available in MATLAB is crucial. This lecture will focus on the use of some of the more sophisticated routines in MATLAB which are cornerstones to scientific computing. Included in this section will be a discussion of the Fast Fourier Transform routines (*fft, ifft, fftshift, ifftshift, fft2, ifft2*), sparse matrix construction (*spdiag, spy*), and high end iterative techniques for solving $\mathbf{Ax} = \mathbf{b}$ (*bicgstab, gmres*). These routines should be studied carefully since they are the building blocks of any serious scientific computing code.

Fast Fourier Transform: FFT, IFFT, FFTSHIFT, IFFTSHIFT2

i.e. we have a function on [-L, L] then x will be 2^n linearly spaced points between -L and L.

As we see on the next page, we do not want the first endpoint to be one of these points, so actually
x2 = linspace(-L, L, 2^n + 1)
x = x2[1:end]

The Fast Fourier Transform will be the first subject discussed. Its implementation is straightforward. Given a function which has been discretized with 2^n points and represented by a vector $\mathbf{x}$, the FFT is found with the command *fft(x)*. Aside from transforming the function, the algorithm associated with the FFT does three major things: it shifts the data so that $x \in [0, L] \to [-L, 0]$ and $x \in [-L, 0] \to [0, L]$, additionally it multiplies every other mode by -1, and it assumes you are working on a 2π periodic domain. These properties are a consequence of the FFT algorithm discussed in detail at a later time.

To see the practical implications of the FFT, we consider the transform of a Gaussian function. The transform can be calculated analytically so that we have the exact relations:

$$f(x) = \exp(-\alpha x^2) \quad \rightarrow \quad \widehat{f}(k) = \frac{1}{\sqrt{2\alpha}} \exp\left(-\frac{k^2}{4\alpha}\right). \tag{2.1.15}$$

A simple MATLAB code to verify this with $\alpha = 1$ is as follows

```
clear all; close all;  % clear all variables and figures

L=20;   % define the computational domain [-L/2,L/2]
n=128;  % define the number of Fourier modes 2^n

x2=linspace(-L/2,L/2,n+1);  % define the domain discretization
x=x2(1:n);   % consider only the first n points:  periodicity

u=exp(-x.*x);          % function to take a derivative of
ut=fft(u);             % FFT the function
utshift=fftshift(ut); % shift FFT

figure(1), plot(x,u)     % plot initial gaussian
figure(2), plot(abs(ut))      % plot unshifted transform
figure(3), plot(abs(utshift)) % plot shifted transform
```

ignore the starting point?

The second figure generated by this script shows how the pulse is shifted. By using the command *fftshift*, we can shift the transformed function back to its mathematically correct positions as shown in the third figure generated. However, before inverting the transformation, it is crucial that the transform is shifted back to the form of the second figure. The command *ifftshift* does this. In general, unless you need to plot the spectrum, it is better not to deal with the *fftshift* and *ifftshift* commands. A graphical representation of the *fft* procedure and its shifting properties is illustrated in Fig. 4 where a Gaussian is transformed and shifted by the *fft* routine.

we cannot perform the fourier transform inverse on the shifted one! Only on the original

FFT versus Finite-Difference Differentiation

Taylor series methods for generating approximations to derivatives of a given function can also be used. In the Taylor series method, the approximations are *local* since they are given by nearest neighbor coupling formulas of finite differences. Using the FFT, the approximation to the derivative is *global* since it uses an expansion basis of cosines and sines which extend over the entire domain of the given function.

Taylor series = local approximations to derivative based on fixed starting point

FFT = global approximations to the derivative based on entire interval.

The calculation of the derivative using the FFT is performed trivially from the derivative relation formula (2.1.11). As an example of how to implement

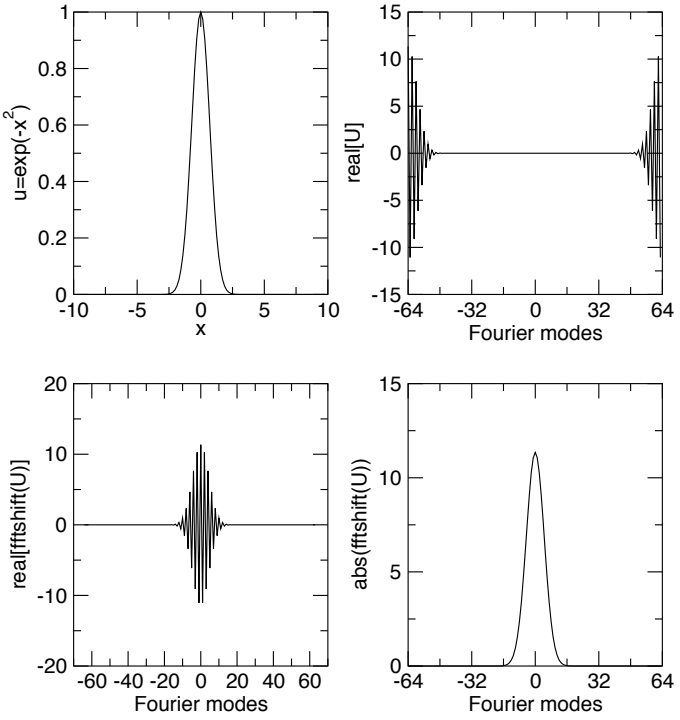


Figure 4: Fast Fourier Transform U = fft(u) of Gaussian data illustrating the shifting properties of the FFT routine. Note that the *fftshift* command restores the transform to its mathematically correct, unshifted state.

this differentiation formula, consider a specific function:

$$u(x) = \text{sech}(x) \tag{2.1.16}$$

which has the following derivative relations

$$\frac{du}{dx} = -\text{sech}(x)\tanh(x) \tag{2.1.17a}$$

$$\frac{d^2u}{dx^2} = \text{sech}(x) - 2\,\text{sech}^3(x)\,. \tag{2.1.17b}$$

A comparison can be made between the differentiation accuracy of finite difference formulas of Tables **??-??** and the spectral method using FFTs. The following code generates the derivative using the FFT method along with the finite difference $O(\Delta x^2)$ and $O(\Delta x^4)$ approximations considered with finite differences.

```
clear all; close all;  % clear all variables and figures
```

```
L=20;   % define the computational domain [-L/2,L/2]
n=128;  % define the number of Fourier modes 2^n

x2=linspace(-L/2,L/2,n+1);  % define the domain discretization
x=x2(1:n);   % consider only the first n points:  periodicity
dx=x(2)-x(1);  % dx value needed for finite difference
u=sech(x);     % function to take a derivative of
ut=fft(u);     % FFT the function
k=(2*pi/L)*[0:(n/2-1) (-n/2):-1]; % k rescaled to 2pi domain

% FFT calculation of derivatives

ut1=i*k.*ut;         % first derivative
ut2=-k.*k.*ut;       % second derivative
u1=real(ifft(ut1)); u2=real(ifft(ut2)); % inverse transform
u1exact=-sech(x).*tanh(x);  % analytic first derivative
u2exact=sech(x)-2*sech(x).^3;  % analytic second derivative

% Finite difference calculation of first derivative

 % 2nd-order accurate
 ux(1)=(-3*u(1)+4*u(2)-u(3))/(2*dx);
 for j=2:n-1
     ux(j)=(u(j+1)-u(j-1))/(2*dx);
 end
 ux(n)=(3*u(n)-4*u(n-1)+u(n-2))/(2*dx);

 % 4th-order accurate
 ux2(1)=(-3*u(1)+4*u(2)-u(3))/(2*dx);
 ux2(2)=(-3*u(2)+4*u(3)-u(4))/(2*dx);
 for j=3:n-2
   ux2(j)=(-u(j+2)+8*u(j+1)-8*u(j-1)+u(j-2))/(12*dx);
 end
 ux2(n-1)=(3*u(n-1)-4*u(n-2)+u(n-3))/(2*dx);
 ux2(n)=(3*u(n)-4*u(n-1)+u(n-2))/(2*dx);

figure(1)
plot(x,u,'r',x,u1,'g',x,u1exact,'go',x,u2,'b',x,u2exact,'bo')
figure(2)
subplot(3,1,1), plot(x,u1exact,'ks-',x,u1,'k',x,ux,'ko',x,ux2,'k*')
axis([-1.15 -0.75 0.47 0.5])
subplot(3,1,2), plot(x,u1exact,'ks-',x,u1,'kv',x,ux,'ko',x,ux2,'k*')
axis([-0.9376 -0.9374 0.49848 0.49850])
```

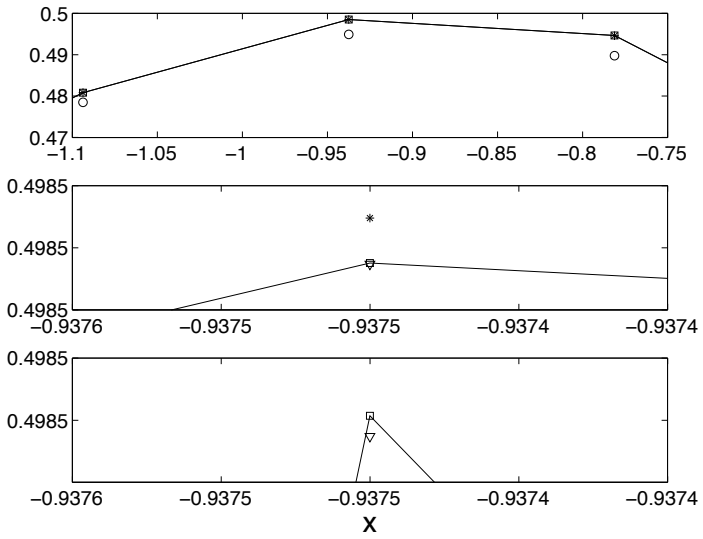


Figure 5: Accuracy comparison between second- and fourth-order finite difference methods and the spectral FFT method for calculating the first derivative. Note that by using the *axis* command, the exact solution (line) and its approximations can be magnified near an arbitrary point of interest. Here, the top figure shows that the second-order finite difference (circles) method is within $O(10^{-2})$ of the exact derivative. The fourth-order finite difference (star) is within $O(10^{-5})$ of the exact derivative. Finally, the FFT method is within $O(10^{-6})$ of the exact derivative. This demonstrates the spectral accuracy property of the FFT algorithm.

```
subplot(3,1,3), plot(x,u1exact,'ks-',x,u1,'kv',x,ux,'ko',x,ux2,'k*')
axis([-0.9376 -0.9374 0.498487 0.498488])
```

Note that the real part is taken after inverse Fourier transforming due to the numerical round off which generates a small $O(10^{-15})$ imaginary part.

Of course, this differentiation example works well since the periodic boundary conditions of the FFT are essentially satisfied with the choice of function $f(x) = \mathrm{sech}(x)$. Specifically, for the range of values $x \in [-10, 10]$, the value of $f(\pm 10) = \mathrm{sech}(\pm 10) \approx 9 \times 10^{-5}$. For a function which does not satisfy periodic boundary conditions, the FFT routine leads to significant error. Much of this error arises from the discontinuous jump and the associated *Gibb's phenomena* which results when approximating with a Fourier series. To illustrate this, Fig. 6 shows the numerically generated derivative to the function $f(x) = \tanh(x)$, which is $f'(x) = \mathrm{sech}^2(x)$. In this example, it is clear that the FFT method

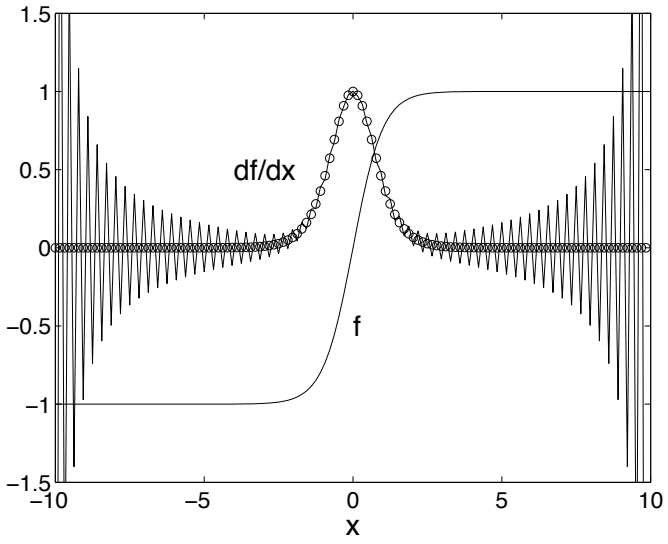


Figure 6: Accuracy comparison between the fourth-order finite difference method and the spectral FFT method for calculating the first derivative of $f(x) = \tanh(x)$. The fourth-order finite difference (circle) is within $O(10^{-5})$ of the exact derivative. In contrast, the FFT approximation (line) oscillates strongly and provides a highly inaccurate calculation of the derivative due to the non-periodic boundary conditions of the given function.

is highly undesirable, whereas the finite difference method performs as well as before.

Higher dimensional Fourier transforms

For transforming in higher dimensions, a couple of choices in MATLAB are possible. For 2D transformations, it is recommended to use the commands *fft2* and **ifft2**. These will transform a matrix **A**, which represents data in the x and y direction respectively, along the rows and then columns. For higher dimensions, the **fft** command can be modified to **fft(x,[],N)** where N is the number of dimensions. Alternatively, use **fftn(x)** for multi-dimensional FFTs.

2.2 FFT Application: Radar Detection and Filtering

FFTs and other related frequency transforms have revolutionized the field of digital signal processing and imaging. The key concept in any of these applications is to use the FFT to analyze and manipulate data in the frequency domain.

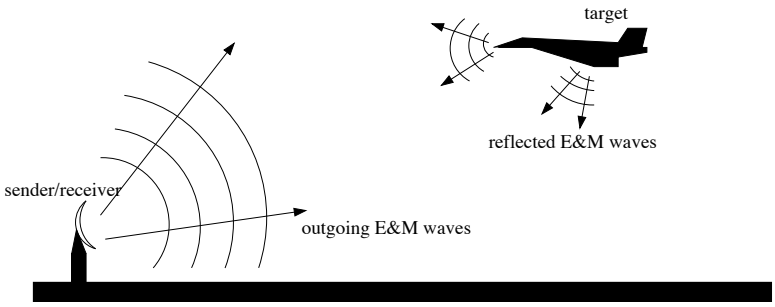


Figure 7: Schematic of the operation of a radar detection system. The radar emits an electromagnetic (E&M) field at a specific frequency. The radar then receives the reflection of this field from objects in the environment and attempts to determine what the objects are.

There are other methods of treating the signal in the time and frequency domain, but the simplest to begin with is the Fourier transform.

The Fourier transform is also used in quantum mechanics to represent a quantum wavepacket in either the spatial domain or the momentum (spectral) domain. Quantum mechanics is specifically mentioned due to the well-known *Heisenberg Uncertainty Principle*. Simply stated, you cannot know the exact position or momentum of a quantum particle simultaneously. This is simply a property of the Fourier transform itself. Essentially, a narrow signal in the time domain corresponds to a broadband source, whereas a highly confined spectral signature corresponds to a broad time domain source. This has significant implication for many techniques for signal analysis. In particular, an excellent decomposition of a given signal into its frequency components can render no information about *when* in time the different portions of signal actually occurred. Thus there is often a competition between trying to localize both in time and frequency a signal or stream of data. Time-frequency analysis is ultimately all about trying to resolve by the time and frequency domain in an efficient and tractable manner. Various aspects of this time-frequency processing will be considered in later lectures, including wavelet based methods of time-frequency analysis.

trade off between time resolution and freq resolution (where freq is occuring and detectable freq range)

At this point, we discuss a very basic concept and manipulation procedure to be performed in the frequency domain: noise attenuation via frequency (band-pass) filtering. This filtering process is fairly common in electronics and signal detection. As a specific application or motivation for spectral analysis, consider the process of radar detection depicted in Fig. 7. An outgoing electromagnetic field is emitted from a source which then attempts to detect the reflections of the emitted signal. In addition to reflections coming from the desired target, reflections can also come from geographical objects (mountains, trees, etc.) and atmospheric phenomena (clouds, precipitation, etc.). Further, there may be

other sources of the electromagnetic energy at the desired frequency. Thus the radar detection process can be greatly complicated by a wide host of environmental phenomena. Ultimately, these effects combine to give at the detector a noisy signal which must be processed and filtered before a detection diagnosis can be made.

It should be noted that the radar detection problem discussed in what follows is highly simplified. Indeed, modern day radar detection systems use much more sophisticated time-frequency methods for extracting both the position and spectral signature of target objects. Regardless, this analysis gives a high-level view of the basic and intuitive concepts associated with signal detection.

To begin we consider a ideal signal, i.e. an electromagnetic pulse, generated in the time-domain.

```
clear all; close all;

L=30;    % Total time slot to transform
n=512;   % number of Fourier modes 2^7
t2=linspace(-L,L,n+1); t=t2(1:n);  % time discretization
k=(2*pi/(2*L))*[0:(n/2-1) -n/2:-1];  % frequency components of FFT

u=sech(t);       % ideal signal in the time domain
figure(1), subplot(3,1,1), plot(t,u,'k'), hold on
```

why strip off the first point?

This will generate an ideal hyperbolic secant shape in the time domain. It is this time domain signal that we will attempt to reconstruct via denoising and filtering. Figure 8(a) illustrates the ideal signal.

In most applications, the signals as generated above are not ideal. Rather, they have a large amount of noise integrated within them. Usually this noise is what is called *white noise*, i.e. a noise which effects all frequencies the same. We can add white noise to this signal by considering the pulse in the frequency domain.

```
noise=1;
ut=fft(u);
utn=ut+noise*(randn(1,n)+i*randn(1,n));
un=ifft(utn);
figure(1), subplot(3,1,2), plot(t,abs(un),'k'), hold on
```

These three lines of code generate the Fourier transform of the function along with the vector **utn** which is the spectrum of the given signal with a complex and Gaussian distributed (mean zero, unit variance) noise source term added in. Figure 8 shows the difference between the ideal time-domain signal pulse and the more physically realistic pulse for which white noise has been added. In these figures, a clear *signal*, i.e. the original time-domain pulse, is still detected even in the presence of noise.

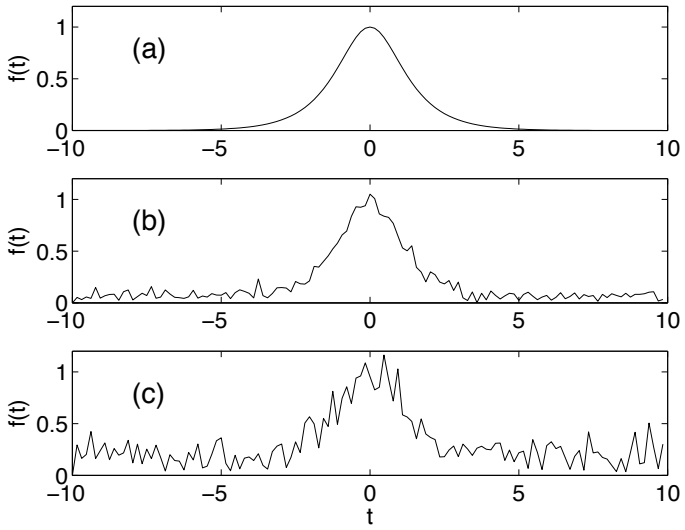


Figure 8: Ideal time-domain pulse (a) along with two realizations of the pulse with increading noise strength (b) and (c). The noisy signals are what are typically expected in applications.

The fundamental question in signal detection now arises: is the time-domain structure received in a detector simply a result of noise, or is there an underlying signal being transmitted. For small amounts of noise, this is clearly not a problem. However, for large amounts of noise or low-levels of signal, the detection becomes a nontrivial matter.

In the context of radar detection, two competing objectives are at play: for the radar detection system, an accurate diagnosis of the data is critical in determining the presence of an aircraft. Competing in this regard is, perhaps, the aircrafts objective of remaining invisible, or undetected, from the radar. Thus an aircraft attempting to remain invisible will attempt to reflect as little electromagnetic signal as possible to the radar. This can be done by, for instance, covering the aircraft with materials that absorb the radar frequencies used for detection, i.e. stealth technology. An alternative method is to fly another aircraft through the region which bombards the radar detection system with electromagnetic energy at the detection frequencies. Thus the aircraft attempting to remain undetected may be successful since the radar system may not be able to distinguish the difference in the sources of electromagnetic fields. This second option is, in some sense, like a radar jamming system. In the first case, the radar detection system will have a small signal-to-noise ratio and denoising will be critical to accurate detection. In the second case, an accurate time-

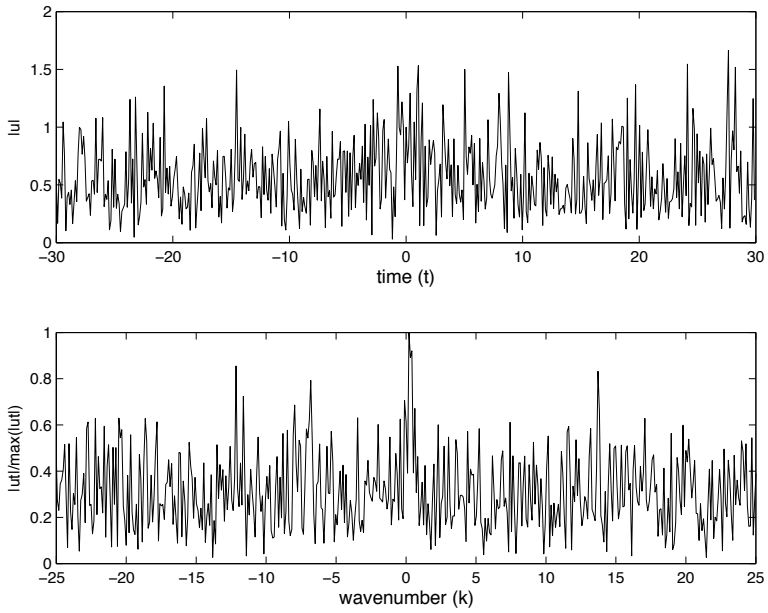


Figure 9: Time-domain (top) and frequency-domain (bottom) plots for a single realization of white-noise. In this case, the noise strength has been increased to ten, thus burying the desired signal field in both time and frequency.

frequency analysis is critical for determining the time localization and positions of the competing signals.

The focus of what follows is to apply the ideas of spectral filtering to attempt to improve the signal detection by denoising. Spectral filtering is a method which allows us to extract information at specific frequencies. For instance, in the radar detection problem, it is understood that only a particular frequency (the emitted signal frequency) is of interest at the detector. Thus it would seem reasonable to filter out, in some appropriate manner, the remaining frequency components of the electromagnetic field received at the detector.

filtering out unwanted frequencies (i.e. we know our desired freq is within a specific range)

Consider a very noisy field created around the hyperbolic secant of the previous example (See Fig. 9):

```
noise=10;
ut=fft(u);
unt=ut+noise*(randn(1,n)+i*randn(1,n));
un=ifft(unt);

subplot(2,1,1), plot(t,abs(un),'k')
axis([-30 30 0 2])
```

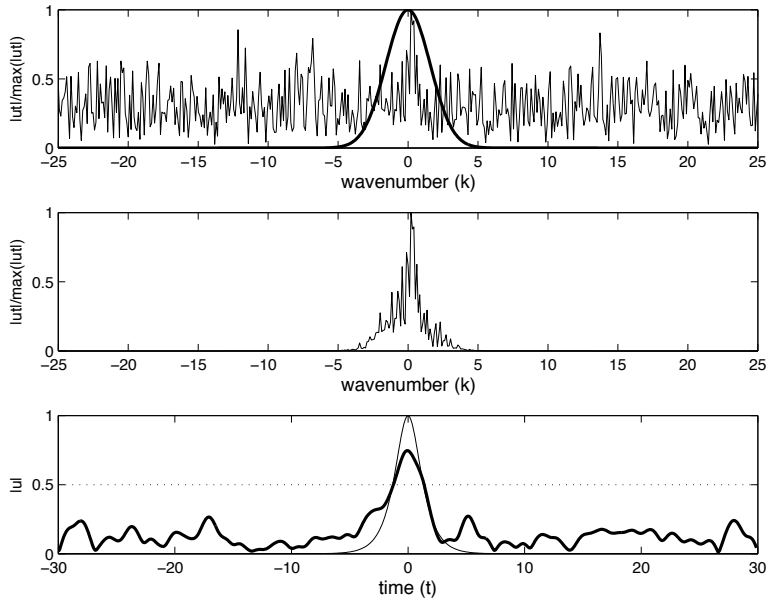


Figure 10: (top) White-noise inundated signal field in the frequency domain along with a Gaussian filter with bandwidth parameter $\tau = 0.2$ centered on the signal center frequency. (middle) The post-filtered signal field in the frequency domain. (bottom) The time-domain reconstruction of the signal field (bolded line) along with the ideal signal field (light line) and the detection threshold of the radar (dotted line).

```
xlabel('time (t)'), ylabel('|u|')
subplot(2,1,2)
plot(fftshift(k),abs(fftshift(unt))/max(abs(fftshift(unt))),'k')
axis([-25 25 0 1])
xlabel('wavenumber (k)', ylabel('|ut|/max(|ut|)')
```

Figure 9 demonstrates the impact of a large noise applied to the underlying signal. In particular, the signal is completely buried within the white-noise fluctuations, making the detection of a signal difficult, if not impossible, with the unfiltered noisy signal field.

Filtering can help significantly improve the ability to detect the signal buried in the noisy field of Fig. 9. For the radar application, the frequency (wavenumber) of the emitted and reflected field is known, thus spectral filtering around this frequency can remove undesired frequencies and much of the white-noise picked up in the detection process. There are a wide range of filters that can be applied to such a problem. One of the simplest filters to consider here is the

tau = bandwidth = how wide you want your filter to be; how close to the desired freq should be the signals we are filtering for

k0 = center freq; what freq we are looking for; filter out signals too far away from this

mathematically, we can see that frequencies k such that
| k - k0| is large will be flattened to zero, so the filter behaves as expected.

Gaussian filter (See Fig. 10):

$$\mathcal{F}(k) = \exp(-\tau(k-k_0)^2) \tag{2.2.1}$$

is it really low pass? Freqs too far below k0 will also be squished to zero

where τ measures the bandwidth of the filter, and k is the wavenumber. The generic filter function $\mathcal{F}(k)$ in this case acts as a low-pass filter since it eliminates high-frequency components in the system of interest. Note that these are high-frequencies in relation to the center-frequency ($k = k_0$) of the desired signal field. In the example considered here $k_0 = 0$.

Application of the filter attenuates strongly those frequencies away from center frequency k_0. Thus if there is a signal near k_0, the filter isolates the signal input around this frequency or wavenumber. Application of the filtering results in the following spectral processing in MATLAB:

```
filter=exp(-0.2*(k).^2);
unft=filter.*unt;
unf=ifft(unft);
```

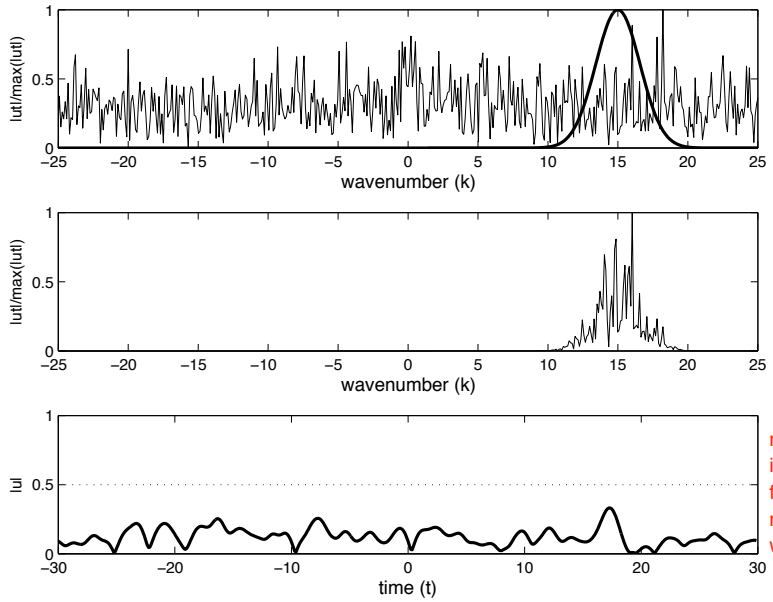
The application of these lines of code are illustrate in Fig. 10 where the white-noise inundated signal field in the spectral domain is filtered with a Gaussian filter centered around the zero wavenumber $k_0 = 0$ with a bandwidth parameter $\tau = 0.2$. The filtering extracts nicely the signal field despite the strength of the applied white-noise. Indeed, Fig. 10 (bottom) illustrates the effect of the filtering process and its ability to reproduce an approximation to the signal field. In addition to the extracted signal field, a detection threshold is placed (dotted line) on the graph in order to illustrate that detector would read the extracted signal as a target.

As a matter of completeness, the extraction of the electromagnetic field is also illustrated when not centered on the center frequency. Figure 11 shows the field that is extracted for a filter centered around $k_0 = 15$. This is done easily with the MATLAB commands

```
filter=exp(-0.2*(k-15).^2);
unft=filter.*unt;
unf=ifft(unft);
```

In this case, there is no signal around the wavenumber $k_0 = 15$. However, there is a great deal of white-noise electromagnetic energy around this frequency. Upon filtering and inverting the Fourier transform, it is clear from Figure 11 (bottom) that there is no discernible target present. As before, a decision threshold is set at $|u| = 0.5$ and the white noise fluctuations clearly produce a field well below the threshold line.

In all of the analysis above, filtering is done on a given set of signal data. However, the signal data is evolving in time, i.e. the position of the aircraft is probably moving. Moreover, one can imagine that there is some statistical



no significant signals after filtering frequencies. i.e. no signficant signals around the range the filter searched fro (wave number 15) which makes sense as our signal is centered at 0 with some noise.

Figure 11: (top) White-noise inundated signal field in the frequency domain along with a Gaussian filter with bandwidth parameter $\tau = 0.2$ centered on the signal frequency $k = 15$. (middle) The post-filtered signal field in the frequency domain. (bottom) The time-domain reconstruction of the signal field (bolded line) along the detection threshold of the radar (dotted line).

chance that a type I or type II error can be made in the detection. Thus a target may be identified when there is no target, or a target has been missed completely. Ultimately, the goal is to use repeated measurements from the radar of the airspace in order to try to avoid a detection error or failure. Given that you may launch a missile, the stakes are high and a high level of confidence is needed in the detection process. It should also be noted that filter design, shape, and parameterization play significant roles in designing optimal filters. Thus filter design is an object of strong interest in signal processing applications.

2.3 FFT Application: Radar Detection and Averaging

The last section clearly shows the profound and effective use of filtering in cleaning up a noisy signal. This is one of the most basic illustrations of the power basic signal processing. Other methods also exist to help reduce noise and generate better time-frequency resolution. In particular, the filtering example given fails to use two key facts: the noise is white, and radar will continue to produce more signal data. These two facts can be used to produce useful information

about the incoming signal. In particular, it is known, and demonstrated in the last section, that white-noise can be modeled adding a normally distributed random variable with zero mean and unit variance to each Fourier component of the spectrum. The key here: with zero mean. Thus if average over many signals, the white-noise should, on average, add up to zero. A very simple concept, yet tremendously powerful in practice for signal processing systems where there is continuous detection of a signal.

use the fact that the noise has zero mean (likely) to improve the filtering process>

To illustrate the power of denoising the system by averaging, the simple analysis of the last section will be considered for a time pulse. In beginning this process, a time window must be selected and its Fourier resolution decided:

```
clear all; close all; clc

L=30;  % total time slot
n=512; % Fourier modes
t2=linspace(-L,L,n+1); t=t2(1:n);
k=(2*pi/(2*L))*[0:(n/2-1) -n/2:-1]; ks=fftshift(k);
noise=10;
```

In order to be more effective in plotting our spectral results, the Fourier components in the standard shifted frame (**k**) and its unshifted counterpart (**ks**) are both produced. Further, the noise strength of the system is selected.

In the next portion of the code, the total number of data frames, or realizations of the incoming signal stream, is considered. The word realizations is used here to help underscore the fact that for each data frame, a new realization of the white-noise is produced. In the following code, one, two, five, and one hundred realizations are considered.

```
labels=['(a)';'(b)';'(c)';'(d)'];
realize=[1 2 5 100];
for jj=1:length(realize)

  u=sech(t); ave=zeros(1,n);
  ut=fft(u);
  for j=1:realize(jj)
    utn(j,:)=ut+noise*(randn(1,n)+i*randn(1,n));
    ave=ave+utn(j,:);
    dat(j,:)=abs(fftshift(utn(j,:)))/max(abs(utn(j,:)));
    un(j,:)=ifft(utn(j,:));
  end
  ave=abs(fftshift(ave))/realize(jj);

  subplot(4,1,jj)
  plot(ks,ave/max(ave),'k')
```

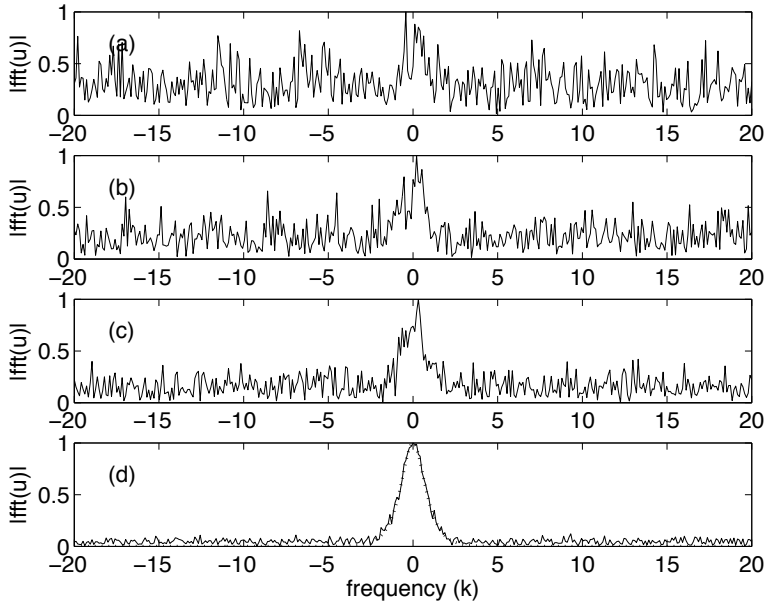


Figure 12: Average spectral content for (a) one, (b) two, (c) five and (d) one hundred realizations of the data. The dotted line in (d) represents the ideal, noise-free spectral signature.

```
  set(gca,'Fontsize',[15])
  axis([-20 20 0 1])
  text(-18,0.7,labels(jj,:),'Fontsize',[15])
  ylabel('|fft(u)|','Fontsize',[15])

end
hold on
plot(ks,abs(fftshift(ut))/max(abs(ut)),'k:','Linewidth',[2])
set(gca,'Fontsize',[15])
xlabel('frequency (k)')
```

Figure 12 demonstrates the averaging process in the spectral domain as a function of the number of realizations. With one realization, it is difficult to determine if there is a true signal, or if the entire spectrum is noise. Already after two realizations, the center frequency structure starts to appear. Five realizations has a high degree of discrimination between the noise and signal and 100 realizations is almost exactly converged to the ideal, noise-free signal. As expected, the noise in the averaging process is eliminated upon averaging since its mean is zero. To view a few of the data realizations used in computing the averaging

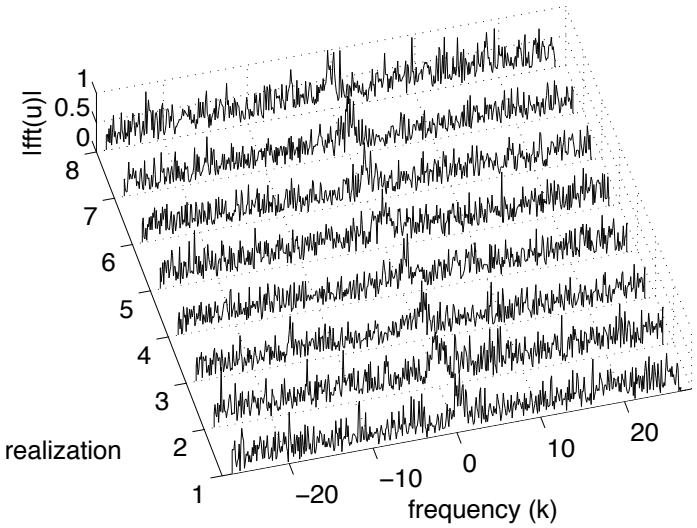


Figure 13: Typical time domain signals that would be produced in a signal detection system. At different times of measurement, the added white-noise would alter the signal stream significantly.

in Fig. 12, the first eight time domain signals used to compute the average for 100 realizations is shown in Fig. 13. The following code is used to produce the waterfall plot used:

```
figure(2)
waterfall(ks,1:8,dat(1:8,:)), colormap([0 0 0]), view(-15,80)
set(gca,'Fontsize',[15],'Xlim',[-28 28],'Ylim',[1 8])
xlabel('frequency (k)'), ylabel('realization'),zlabel('|fft(u)|')
```

Figures 12 and 13 illustrate how the averaging process can ultimately extract a clean spectral signature. As a consequence, however, you completely loose the time-domain dynamics. In other words, one could take the cleaned up spectrum of Fig. 12 and inverse the Fourier transform, but this would only give the average time domain signal, but not how it actually evolved over time. To consider this problem more concretely, consider the following bit of code that generates a time-varying signal shown in Fig. 14. There are a total of 21 realizations of data in this example.

your time series signal is only the time average of your true signal even though all the noise has "canceled out"

```
slice=[0:0.5:10];
[T,S]=meshgrid(t,slice);
[K,S]=meshgrid(k,slice);
```

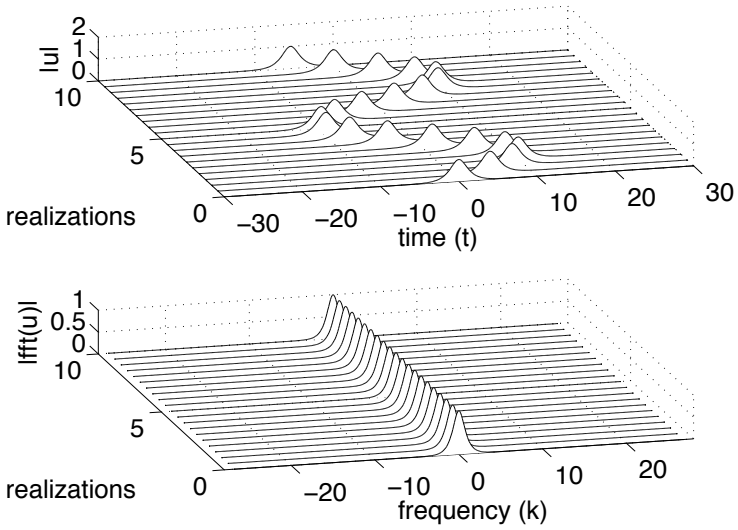


Figure 14: Ideal time-frequency behavior for a time-domain pulse that evolves dynamically in time. The spectral signature remains unchanged as the pulse moves over the time domain.

```
U=sech(T-10*sin(S)).*exp(i*0*T);
subplot(2,1,1)
waterfall(T,S,U), colormap([0 0 0]), view(-15,70)
set(gca,'Fontsize',[15],'Xlim',[-30 30],'Zlim',[0 2])
xlabel('time (t)'), ylabel('realizations'), zlabel('|u|')

for j=1:length(slice)
  Ut(j,:)=fft(U(j,:));
  Kp(j,:)=fftshift(K(j,:));
  Utp(j,:)=fftshift(Ut(j,:));
  Utn(j,:)=Ut(j,:)+noise*(randn(1,n)+i*randn(1,n));
  Utnp(j,:)=fftshift(Utn(j,:))/max(abs(Utn(j,:)));
  Un(j,:)=ifft(Utn(j,:));
end
figure(3)
subplot(2,1,2)
waterfall(Kp,S,abs(Utp)/max(abs(Utp(1,:)))), colormap([0 0 0]), view(-15,70)
set(gca,'Fontsize',[15],'Xlim',[-28 28])
xlabel('frequency (k)'), ylabel('realizations'), zlabel('|fft(u)|')
```

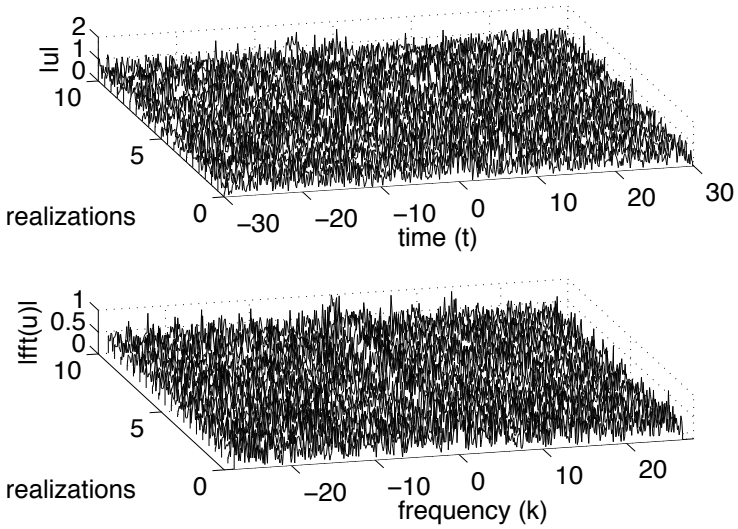


Figure 15: A more physically realistic depiction of the time-domain and frequency spectrum as a function of the number of realizations. The goal is to extract the meaningful data buried within the noise.

If this was a radar problem, the movement of the signal would represent an aircraft moving in time. The spectrum, however, remains fixed at the transmitted signal frequency. Thus it is clear even at this point that averaging over the frequency realizations produces a clean, localized signature in the frequency domain, whereas averaging over the time-domain only smears out the evolving time-domain pulse.

The ideal pulse evolution in Fig. 14 is now inundated with white-noise as shown in Fig. 15. The noise is added to each realization, or data measurement, with the same strength as shown in Figs. 12 and 13. The following code plots the noise time-domain and spectral evolution:

```
figure(4)
subplot(2,1,1)
waterfall(T,S,abs(Un)), colormap([0 0 0]), view(-15,70)
set(gca,'Fontsize',[15],'Xlim',[-30 30],'Zlim',[0 2])
xlabel('time (t)'), ylabel('realizations'), zlabel('|u|')
subplot(2,1,2)
waterfall(Kp,S,abs(Utnp)), colormap([0 0 0]), view(-15,70)
set(gca,'Fontsize',[15],'Xlim',[-28 28])
xlabel('frequency (k)'), ylabel('realizations'), zlabel('|fft(u)|')
```

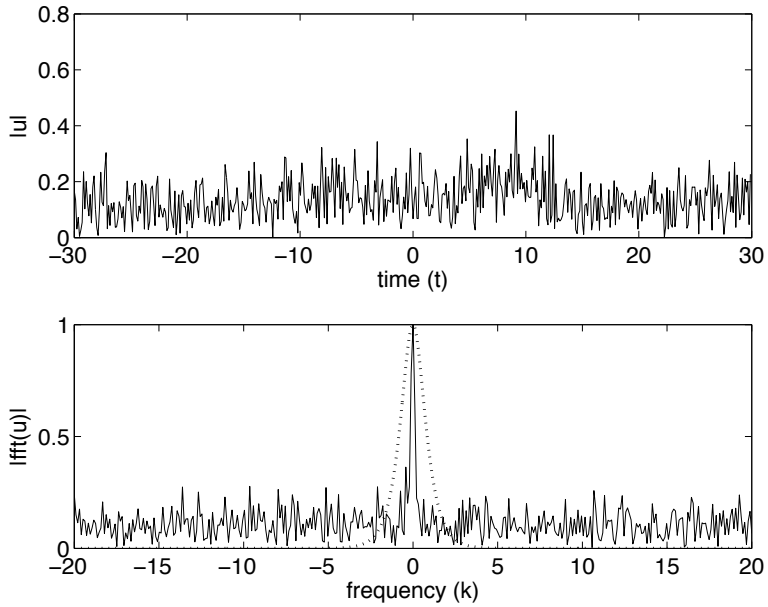


Figure 16: Averaged time-domain and spectral profiles for the 21 realizations of data shown in Fig. 15. Even with a limited sampling, the spectral signature at the center frequency is extracted.

The obvious question arises about how to cleanup the signal and whether something meaningful can be extracted from the data or not. After all, there are only 21 data slices to work with. The following code averages over the 21 data realizations in both the time and frequency domains

```
figure(5)
Uave=zeros(1,n); Utave=zeros(1,n);
for j=1:length(slice)
  Uave=Uave+Un(j,:);
  Utave=Utave+Utn(j,:);
end
Uave=Uave/length(slice);
Utave=fftshift(Utave)/length(slice);

subplot(2,1,1)
plot(t,abs(Uave),'k')
set(gca,'Fontsize',[15])
xlabel('time (t)'), ylabel('|u|')
subplot(2,1,2)
```

```
plot(ks,abs(Utave)/max(abs(Utave)),'k'), hold on
plot(ks,abs(fftshift(Ut(1,:)))/max(abs(Ut(1,:))),'k:','Linewidt',[2])
axis([-20 20 0 1])
set(gca,'Fontsize',[15])
xlabel('frequency (k)'), ylabel('|fft(u)|')
```

Figure 16 shows the results from the averaging process for a nonstationary signal. The top graph shows that averaging over the time domain for a moving signal produces no discernible signal. However, averaging over the frequency domain produces a clear signature at the center-frequency of interest. Ideally, if more data is collected a better average signal is produced. However, in many applications, the acquisition of data is limited and decisions must be made upon what are essentially small sample sizes.

because the signal is a wave! it moves!

2.4 Time-Frequency Analysis: Windowed Fourier Transforms

The Fourier transform is one of the most important and foundational methods for the analysis of signals. However, it was realized very early on that Fourier transform based methods had severe limitations. Specifically, when transforming a given time signal, it is clear that all the frequency content of the signal can be captured with the transform, but the transform fails to capture the *moment in time* when various frequencies were actually exhibited. Figure 17 shows a proto-typical signal that may be of interest to study. The signal $S(t)$ is clearly comprised of various frequency components that are exhibited at different times. For instance, at the beginning of the signal, there are high frequency components. In contrast, the middle of the signal has relatively low frequency oscillations. If the signal represented music, then the beginning of the signal would produce high notes while the middle would produce low notes. The Fourier transform of the signal contains all this information, but there is no indication of *when* the high or low notes actually occur in time. Indeed, by definition the Fourier transform eliminates all time-domain information since you actually integrated out all time in Eq. (2.1.8).

The obvious question to arise is this: what is the Fourier transform good for in the context of signal processing? In the previous sections where the Fourier transform was applied, the signal being investigated was fixed in frequency, i.e. a sonar or radar detector with a fixed frequency ω_0. Thus for a given signal, the frequency of interest did not shift in time. By then using different measurements in time, a signal tracking algorithm could be constructed. Thus an implicit assumption was made about the invariance of the signal frequency. Ultimately, the Fourier transform is superb for one thing: characterizing *stationary* or periodic signals. Informally, a stationary signal is such that repeated measurements of the signal in time yield an average value that does not change in time. Most signals, however, do not satisfy this criteria. A song, for instance, changes its

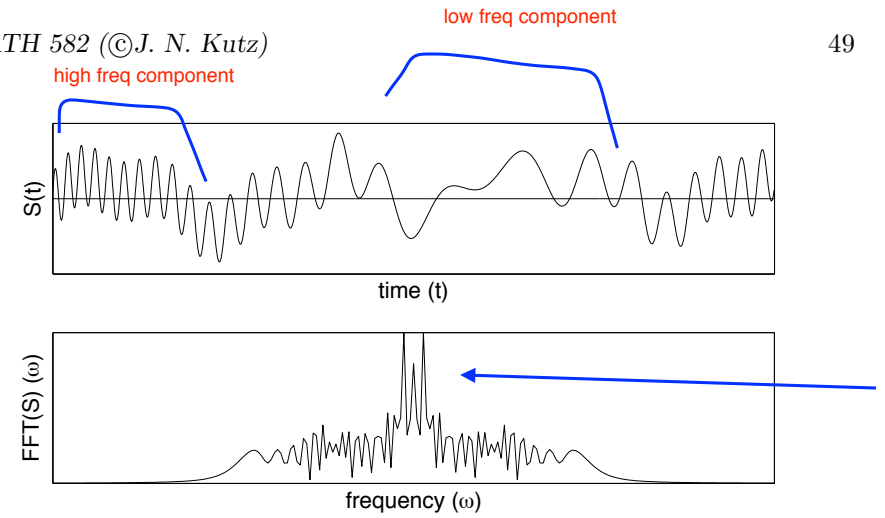


Figure 17: Signal $S(t)$ and its normalized Fourier transform $\hat{S}(\omega)$. Note the large number of frequency components that make up the signal.

average Fourier components in time as the song progresses in time. Thus the generic signal $S(t)$ that should be considered, and that is plotted as an example in Fig. 17 is a *non-stationary signal* whose average signal value does change in time. It should be noted that in our application of radar detection of a moving target, use was made of the stationary nature of the spectral content. This allowed for a clear idea of where to filter the signal $\hat{S}(\omega)$ in order to reconstruct the signal $S(t)$.

Having established the fact that the direct application of the Fourier transform provides a nontenable method for extracting signal information, it is natural to pursue modifications of the method in order to extract time and frequency information. The most simple minded approach is to consider Fig. 17 and to decompose the signal over the time domain into separate time frames. Figure 18 shows the original signal $S(t)$ considered but now decomposed into four smaller time windows. In this decomposition, for instance, the first time frame is considered with the remaining three time frames zeroed out. For each time window, the Fourier transform is applied in order to characterize the frequencies present during that time frame. The highest frequency components are captured in Fig. 18(a) which is clearly seen in its Fourier transform. In contrast, the slow modulation observed in the third time frame (c) is devoid of high-frequency components as observed in Fig. 18(c). This method thus exhibits the ability of the Fourier transform, appropriately modified, to extract out both time and frequency information from the signal.

The limitations of the direct application of the Fourier transform, and its inability to localize a signal in both the time and frequency domains, was realized very early on in the development of radar and sonar detection. The Hungarian physicist/mathematician/electrial engineer Gábor Dénes (Physics Nobel Prize

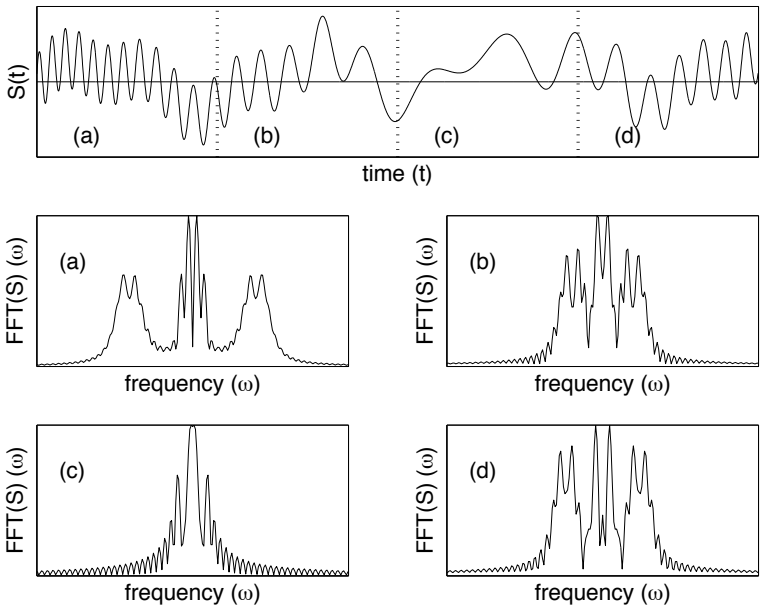


Figure 18: Signal $S(t)$ decomposed into four equal and separate time frames (a), (b), (c) and (d). The corresponding normalized Fourier transform of each time frame $\hat{S}(\omega)$ is illustrated below the signal. Note that this decomposition gives information about the frequencies present in each smaller time frame.

in 1971 for the discovery of holography in 1947) was first to propose a formal method for localizing both time and frequency. His method involved a simple modification of the Fourier transform kernel. Thus Gábor introduced the kernel

$$g_{t,\omega}(\tau) = e^{i\omega\tau} g(\tau - t) \tag{2.4.1}$$

where the new term to the Fourier kernel $g(\tau - t)$ was introduced with the aim of localizing both time and frequency. The *Gábor transform*, also known as the *short-time Fourier transform (STFT)* is then defined as the following:

"window" is center at time tau.

Gabor (short-time Fourier) transform

$$\mathcal{G}[f](t,\omega) = \tilde{f}_g(t,\omega) = \int_{-\infty}^{\infty} f(\tau)\bar{g}(\tau - t)e^{-i\omega\tau} d\tau = (f, \bar{g}_{t,\omega}) \tag{2.4.2}$$

matrix!

where the bar denotes the complex conjugate of the function. Thus the function $g(\tau - t)$ acts as a time filter for localizing the signal over a specific window of time. The integration over the parameter τ slides the time-filtering window down the entire signal in order to pick out the frequency information at each instant of time. Figure 19 gives a nice illustration of how the time filtering scheme of Gábor works. In this figure, the time filtering window is centered at

i.e. we could define g(x) to be

1 if -1 < x < 1
0 otherwise

this would create a window of width 1 so when you integrate over tau g(tau - t), only tau values within +- 1 of t will be considered as part of the signal

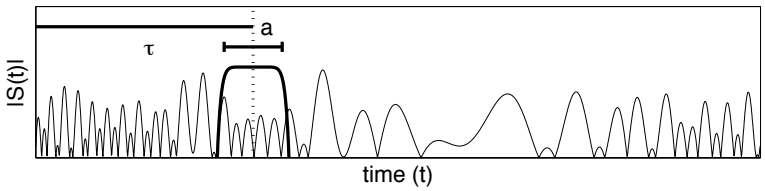


Figure 19: Graphical depiction of the Gábor transform for extracting the time-frequency content of a signal $S(t)$. The time filtering window $g(\tau - t)$ is centered at τ with width a.

τ with a width a. Thus the frequency content of a window of time is extracted and τ is modified to extract the frequencies of another window. The definition of the Gábor transform captures the entire time-frequency content of the signal. Indeed, the Gábor transform is a function of the two variables t and ω.

A few of the key mathematical properties of the Gábor transform are highlighted here. To be more precise about these mathematical features, some assumptions about commonly used $g_{t,\omega}$ are considered. Specifically, for convenience we will consider g to be real and symmetric with $\|g(t)\| = 1$ and $\|g(\tau - t)\| = 1$ where $\|\cdot\|$ denotes the L_2 norm. Thus the definition of the Gábor transform, or STFT, is modified to

$$\mathcal{G}[f](t,\omega) = \tilde{f}_g(t,\omega) = \int_{-\infty}^{\infty} f(\tau)g(\tau - t)e^{-i\omega\tau}d\tau \tag{2.4.3}$$

with $g(\tau - t)$ inducing localization of the Fourier integral around $t = \tau$. With this definition, the following properties hold

1. The energy is bounded by the Schwarz inequality so that
$$|\tilde{f}_g(t,\omega)| \leq \|f\|\|g\| \tag{2.4.4}$$

2. The energy in the signal plane around the neighborhood of (t,ω) is calculated from
$$|\tilde{f}_g(t,\omega)|^2 = \left|\int_{-\infty}^{\infty} f(\tau)g(\tau - t)e^{-i\omega\tau}d\tau\right|^2 \tag{2.4.5}$$

3. The time-frequency spread around of a Gábor window is computed from the variance, or second moment, so that
$$\sigma_t^2 = \int_{-\infty}^{\infty}(\tau - t)|g_{t,\omega}(\tau)|^2 d\tau = \int_{-\infty}^{\infty}\tau^2|g(\tau)|^2 d\tau \tag{2.4.6a}$$
$$\sigma_\omega^2 = \frac{1}{2\pi}\int_{-\infty}^{\infty}(\nu - \omega)^2|\tilde{g}_{t,\omega}(\nu)|^2 d\nu = \frac{1}{2\pi}\int_{-\infty}^{\infty}\nu^2|\tilde{g}(\nu)|^2 d\nu \tag{2.4.6b}$$
where $\sigma_t\sigma_\omega$ is independent of t and ω and is governed by the Heinsenberg uncertainty principle.

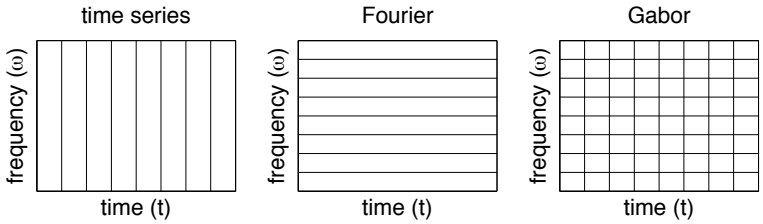


Figure 20: Graphical depiction of the difference between a time series analysis, Fourier analysis and Gábor analysis of a signal. In the time series method, good resolution is achieved of the signal in the time domain, but no frequency resolution is achieved. In Fourier analysis, the frequency domain is well resolved at the expense of losing all time resolution. The Gábor method, or short time Fourier transform, is constructed to give both time and frequency resolution. The area of each box can be constructed from $\sigma_t^2\sigma_\omega^2$.

4. The Gábor transform is linear so that

$$\mathcal{G}[af_1 + bf_2] = a\mathcal{G}[f_1] + b\mathcal{G}[f_2] \tag{2.4.7}$$

5. The Gábor transform can be inverted with the formula

$$f(\tau) = \frac{1}{2\pi}\frac{1}{\|g\|^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\tilde{f}_g(t,\omega)g(\tau - t)e^{i\omega\tau}d\omega dt \tag{2.4.8}$$

where the integration must occur over all frequency and time-shifting components. This double integral is in contrast to the Fourier transform which requires only a single integration since it is a function, $\hat{f}(\omega)$, of the frequency alone.

Figure 20 is a cartoon representation of the fundamental ideas behind a time series analysis, Fourier transform analysis and Gábor transform analysis of a given signal. In the time series method, good resolution is achieved of the signal in the time domain, but no frequency resolution is achieved. In Fourier analysis, the frequency domain is well resolved at the expense of losing all time resolution. The Gábor method, or short time Fourier transform, trades away some measure of accuracy in both the time and frequency domains in order to give both time and frequency resolution simultaneously. Understanding this figure is critical to understanding the basic, high-level notions of time-frequency analysis.

In practice, the Gábor transform is computed by discretizing the time and frequency domain. Thus a discrete version of the transform (2.4.2) needs to be considered. Essentially, by discretizing, the transform is done on a lattice of

Gabor is a trade off of between time resolution and frequency resolution!

i.e. we have the question which frequencies are occuring when?

If you let me shrink the window, I can be more precise at where the frequencies are occuring, but I won't be able to find freqs which are too large to fit in the window.

If I widen the window, I can find a wider range of frequencies but can be less specific on where they are occuring in time.

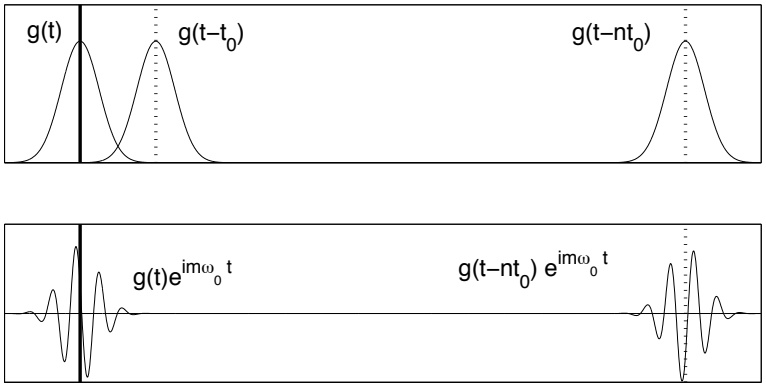


Figure 21: Illustration of the discrete Gábor transform which occurs on the lattice sample points Eq. (2.4.9). In the top figure, the translation with $\omega_0 = 0$ is depicted. The bottom figure depicts both translation in time and frequency. Note that the overlap of the Gábor frames (windows) overlap so that good resolution of the signal can be achieved in both time and frequency since $0 < t_0, \omega_0 < 1$.

time and frequency. Thus consider the lattice, or sample points,

$$\nu = m\omega_0 \tag{2.4.9a}$$

$$\tau = nt_0 \tag{2.4.9b}$$

where m and n are integers and $\omega_0, t_0 > 0$ are constants. Then the discrete version of $g_{t,\omega}$ becomes

$$g_{m,n}(t) = e^{i2\pi m\omega_0 t} g(t - nt_0) \tag{2.4.10}$$

and the Gábor transform becomes

$$\tilde{f}(m,n) = \int_{-\infty}^{\infty} f(t)\bar{g}_{m,n}(t)dt = (f, g_{m,n}) \ . \tag{2.4.11}$$

Note that if $0 < t_0, \omega_0 < 1$, then the signal is over-sampled and time frames exist which yield excellent localization of the signal in both time and frequency. If $\omega_0, t_0 > 1$, the signal is under-sampled and the Gábor lattice is incapable of reproducing the signal. Figure 21 shows the Gábor transform on lattice given by Eq. (2.4.9). The overlap of the Gábor window frames ensures that good resolution in time and frequency of a given signal can be achieved.

Drawbacks of the Gábor (STFT) transform

Although the Gábor transform gives a method whereby time and frequency can be simultaneously characterized, there are obvious limitations to the method.

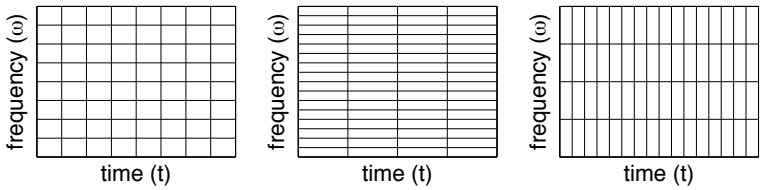


Figure 22: Illustration of the resolution trade-offs in the discrete Gábor transform. The left figure shows a time filtering window that produces nearly equal localization of the time and frequency signal. By increasing the length of the filtering window, increased frequency resolution is gained at the expense of decreased time resolution (middle figure). Decreasing the time window does the opposite: time resolution is increased at the expense of poor frequency resolution (right figure).

Specifically, the method is limited by the time filtering itself. Consider the illustration of the method in Fig. 19. The time window filters out the time behavior of the signal in a window centered at τ with width a. Thus when considering the spectral content of this window, any portion of the signal with a wavelength longer than the window is completely lost. Indeed, since the Heinsenberg relationship must hold, the shorter the time filtering window, the less information there is retained concerning the frequency content. In contrast, longer windows retain more frequency components, but this comes at the expense of losing the time resolution of the signal. Figure 22 provides a graphical description of the failings of the Gábor transform. Specifically the trade offs that occur between time and frequency resolution, and the fact that high-accuracy in one of these comes at the expense of resolution in the other parameter. This is a consequence of a fixed window time filtering window.

Other short-time Fourier transform methods

The Gábor transform is not the only windowed Fourier transform that has been developed. There are several other well-used and highly developed STFT techniques. Here, a couple of these more highly used methods will be mentioned for completeness [4].

The *Zak transform* is closely related to the Gábor transform. It is also called the *Weil-Brezin* transform in harmonic analysis. First introduced by Gelfand in 1950 as a method for characterizing eigenfunction expansions in quantum mechanical system with periodic potentials, it has been generalized to be a key mathematical tool for the analysis of Gábor transform methods. The Zak transform is defined as

$$\mathcal{L}_a f(t,\omega) = \sqrt{a} \sum_{n=-\infty}^{\infty} f(at+an)e^{-i2\pi n\omega} \tag{2.4.12}$$

where $a > 0$ is a constant and n is an integer. Two useful and key properties of this transform are as follows: $\mathcal{L}f(t, \omega + 1) = \mathcal{L}f(t, \omega)$ (periodicity) and $\mathcal{L}f(t + 1, \omega) = \exp(i2\pi\omega)\mathcal{L}f(t, \omega)$ (quasi-periodicity). These properties are particularly important for considering physical problems placed on a lattice.

The *Wigner-Ville Distribution* is particularly important transform in the development of radar and sonar technologies. Its various mathematical properties make it ideal for these applications and provides a method for achieving great time and frequency localization. The Wigner-Ville transform is defined as

$$\mathcal{W}_{f,g}(t, \omega) = \int_{-\infty}^{\infty} f(t + \tau/2)\bar{g}(t - \tau/2)e^{-i\omega}\tau d\tau \tag{2.4.13}$$

where this is a standard Fourier kernel which transforms the function $f(t + \tau/2)\bar{g}(t - \tau/2)$. This transform is nonlinear since $\mathcal{W}_{f_1+f_2,g_1+g_2} = \mathcal{W}_{f_1,g_1} + \mathcal{W}_{f_1,g_2} + \mathcal{W}_{f_2,g_1} + \mathcal{W}_{f_2,g_2}$ and $\mathcal{W}_{f+g} = \mathcal{W}_f + \mathcal{W}_g + 2\Re\{\mathcal{W}_{f,g}\}$.

Ultimately, alternative forms of the STFT are developed for one specific reason: to take advantage of some underlying properties of the physical or mathematical properties of a given system. It is rare that a method developed for radar would be broadly applicable to other physical systems unless they were operating under the same physical principles. Regardless, one can see that specialty techniques exist for time-frequency analysis of different systems.

2.5 Time-Frequency Analysis and Wavelets

The Gábor transform established two key principles for joint time-frequency analysis: *translation* of a short-time window and *scaling* of the short-time window to capture finer time resolution. Figure 19 shows the basic concept introduced in the theory of windowed Fourier transforms. Two parameters are introduced to handle the *translation* and *scaling*, namely τ and a. The shortcoming of this method is that it trades off accuracy in time (frequency) for accuracy in frequency (time). Thus the fixed window size imposes a fundamental limitation on the level of time-frequency resolution that can be obtained.

A simple modification to the Gábor method is to allow the scaling window (a) to vary in order to successively extract extract out improvements in the time resolution. In other words, first the low-frequency (poor time resolution) components are extracted using a broad scaling window. The scaling window is subsequently shortened in order to extract out higher-frequencies and better time resolution. By keeping a catalogue of the extracting process, both excellent time and frequency resolution of a given signal can be obtained. This is the fundamental principle of *wavelet theory*. The term wavelet means little wave and originates from the fact that the scaling window extracts out smaller and smaller pieces of waves from the larger signal.

Wavelet analysis begins with the consideration of a function known as the

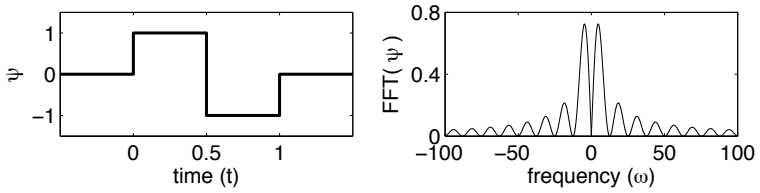


Figure 23: Representation of the compactly supported Haar wavelet function $\psi(t)$ and its Fourier transform $\hat{\psi}(\omega)$. Although highly localized in time due to the compact support, it is poorly localized in frequency with a decay of $1/\omega$.

mother wavelet:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) \tag{2.5.14}$$

where $a \neq 0$ and b are real constants. The parameter a is the scaling parameter illustrated in Fig. 19 whereas the parameter b now denotes the translation parameter (previously denoted by τ in Fig. 19). Unlike Fourier analysis, and very much like Gábor transforms, there are a vast variety of mother wavelets that can be constructed. In principle, the mother wavelet is designed to have certain properties that are somehow beneficial for a given problem. Thus depending upon the application, different mother wavelets may be selected.

Ultimately, the wavelet is simply another expansion basis for representing a given signal or function. Thus it is not unlike the Fourier transform which represents the signal as a series of sines and cosines. Historically, the first wavelet was constructed by Haar in 1910 [5]. Thus the concepts and ideas of wavelets are one century old. However, their widespread use and application did not become prevalent until the mid-1980s. The Haar wavelet is given by the piecewise constant function

$$\psi(t) = \begin{cases} 1 & 0 \leq t < 1/2 \\ -1 & 1/2 \leq t < 1 \\ 0 & \text{otherwise} \end{cases} . \tag{2.5.15}$$

Figure 23 shows the Haar wavelet step function and its Fourier transform which is a sinc like function. Note further that $\int_{-\infty}^{\infty} \psi(t)dt = 0$ and $\|\psi(t)\|^2 = \int_{-\infty}^{\infty} |\psi(t)|^2 dt$. The Haar wavelet is an ideal wavelet for describing localized signals in time (or space) since it has compact support. Indeed, for highly localized signals, it is a much more efficient to use the Haar wavelet basis than the standard Fourier expansion. However. the Haar wavelet has poor localization properties in the frequency domain since it decays like a sinc function in powers of 1ω. This is a consequence of the Heinsenberg uncertainty principle.

To represent a signal with the Haar wavelet basis, the translation and scaling operations associated with the mother wavelet need to be considered. Depicted

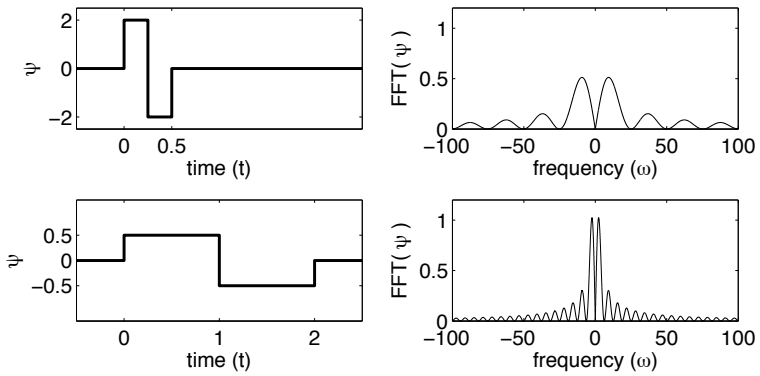


Figure 24: Illustration of the compression and dilation process of the Haar wavelet and its Fourier transform. In the top row, the compressed wavelet $\psi_{1/2,0}$ is shown. Improved time resolution is obtained at the expense of a broader frequency signature. The bottom row, shows the dilated wavelet $\psi_{2,0}$ which allows it to capture lower-frequency components of a signal.

in Fig. 23 and given by Eq. (2.5.15) is the wavelet $\psi_{1,0}(t)$. Thus its translation is zero and its scaling is unity. The concept in reconstructing a signal using the Haar wavelet basis is to consider decomposing the signal into more generic $\psi_{m,n}(t)$. By appropriate selection of the m and n (integers), finer scales and appropriate locations of the signal can be extracted. For $a < 1$, the wavelet is a compressed version of $\psi_{1,0}$ whereas for $a > 1$, the wavelet is a dilated version of $\psi_{1,0}$. The scaling parameter a is typically taken to be a power of two so that $a = 2^j$ for some integer j. Figure 24 shows the compressed and dilated Haar wavelet for $a = 0.5$ and $a = 2$, i.e. $\psi_{1/2,0}$ and $\psi_{2,0}$. The compressed wavelet allows for finer scale resolution of a given signal while the dilated wavelet captures low-frequency components of a signal by having a broad range in time.

The simple Haar wavelet already illustrates all the fundamental principles of the wavelet concept. Specifically by using scaling and translation, a given signal or function can be represented by a basis of functions which allow for higher and higher refinement in the time resolution of a signal. Thus it is much like the Gábor concept, except that now the time window is variable in order to capture different levels of resolution. Thus the large scale structures in time are captured with broad time-domain Haar wavelets. At this scale, the time resolution of the signal is very poor. However by successive rescaling in time, a finer and finer time resolution of the signal can be obtained along with its high-frequency components. The information at the low and high scales is all preserved so that a complete picture of the time-frequency domain can be constructed. Ultimately, the only limit in this process is the number of scaling levels to be considered.

The wavelet basis can be accessed via an integral transform of the form

$$(Tf)(\omega) = \int_t K(t,\omega) f(t) dt \tag{2.5.16}$$

where $K(t,\omega)$ is the kernel of the transform. This is equivalent in principle to the Fourier transform whose kernel are the oscillations given by $K(t,\omega) = \exp(-i\omega t)$. The key idea now is to define a transform which incorporates the mother wavelet as the kernel. Thus we define the *continuous wavelet transform (CWT):*

$$\mathcal{W}_\psi[f](a,b) = (f, \psi_{a,b}) = \int_{-\infty}^{\infty} f(t)\bar{\psi}_{a,b}(t) dt \tag{2.5.17}$$

Much like the windowed Fourier transform, the CWT is a function of the dilation parameter a and translation parameter b. Parenthetically, a wavelet is admissible if the following property holds: can be made

$$C_\psi = \int_{-\infty}^{\infty} \frac{|\hat{\psi}(\omega)|^2}{|\omega|} d\omega < \infty \tag{2.5.18}$$

where the Fourier transform of the wavelet is defined by

$$\hat{\psi}_{a,b} = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} e^{-i\omega t} \psi\left(\frac{t-b}{a}\right) dt = \frac{1}{\sqrt{|a|}} e^{-ib\omega} \hat{\psi}(a\omega)\,. \tag{2.5.19}$$

Thus provided the admissibility condition (2.5.18) is satisfied, the wavelet transform can be well defined.

As an example of the admissibility condition, consider the Haar wavelet (2.5.15). Its Fourier transform can be easily computed in terms of the sinc-like function:

$$\hat{\psi}(\omega) = ie^{-i\omega/2} \frac{\sin^2(\omega/4)}{\omega/4}\,. \tag{2.5.20}$$

Thus the admissibility constant can be computed to be

$$\int_{-\infty}^{\infty} \frac{|\hat{\psi}(\omega)|^2}{|\omega|} d\omega = 16 \int_{-\infty}^{\infty} \frac{1}{|\omega|^3} \left|\sin\frac{\omega}{4}\right|^4 d\omega < \infty\,. \tag{2.5.21}$$

This then shows that the Haar wavelet is in the admissible class.

Another interesting properties of the wavelet transform is the ability to construct new wavelet bases. The following theorem is of particular importance.

Theorem: *If ψ is a wavelet and ϕ is a bounded integrable function, then the convolution $\psi \star \phi$ is a wavelet.*

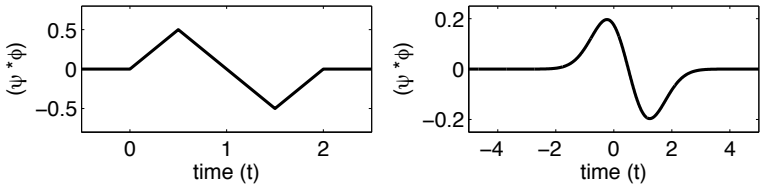


Figure 25: Convolution of the Haar wavelet with the functions (2.5.22) (left panel) and (2.5.23) (right panel). The convolved functions can be used as the mother wavelet for a wavelet basis expansion.

In fact, from the Haar wavelet (2.5.15) we can construct new wavelet functions by convolving with for instance

$$\phi(t) = \begin{cases} 0 & t < 0 \\ 1 & 0 \le t \le 1 \\ 0 & t \ge 1 \end{cases} \tag{2.5.22}$$

or the function

$$\phi(t) = e^{-t^2} . \tag{2.5.23}$$

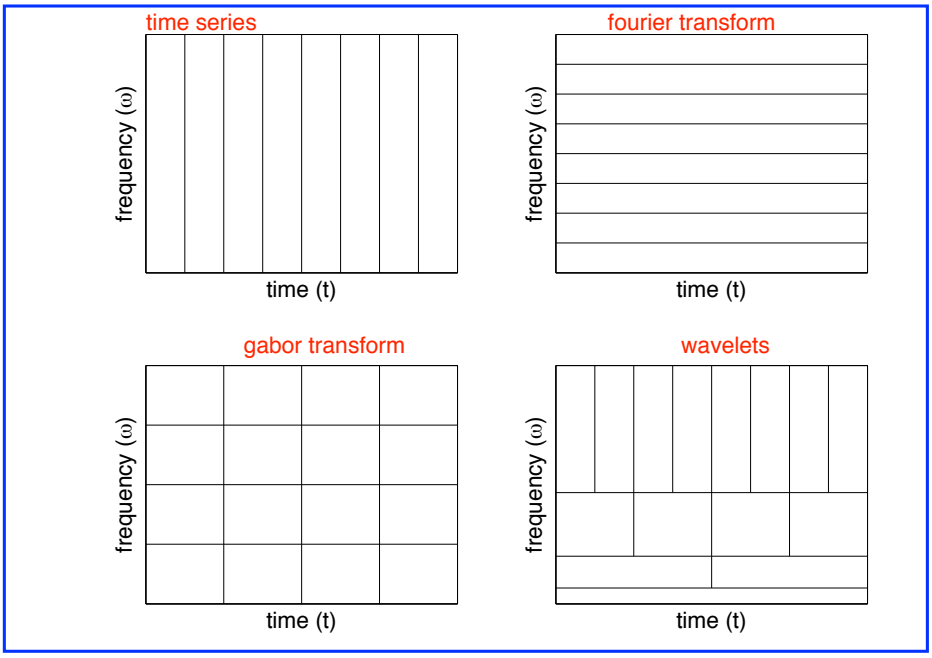
convolution f and g

int (f(t) g(t - tau) dtau) from - infinity to infinity

The convolution of these functions ϕ with the Haar wavelet ψ (2.5.15) are produced in Fig. 25. These convolutions could also be used as mother wavelets in constructing a decomposition of a given signal or function.

The wavelet transform principle is quite simple. First, the signal is split up into a bunch of smaller signals by translating the wavelet with the parameter b over the entire time domain of the signal. Second, the same signal is processed at different frequency bands, or resolutions, by scaling the wavelet window with the parameter a. The combination of translation and scaling allows for processing of the signals at different times and frequencies. Figure 25 is an upgrade of Fig. 20 that incorporates the wavelet transform concept in the time-frequency domain. In this figure, the standard time-series, Fourier transform and windowed Fourier transform are represented along with the multi-resolution concept of the wavelet transform. In particular, the box illustrating the wavelet transform shows the multi-resolution concept in action. Starting with a large Fourier domain window, the entire frequency content is extracted. The time window is then scaled in half, leading to finer time resolution at the expense of worse frequency resolution. This process is continued until a desired time-frequency resolution is obtained. This simple cartoon is critical for understanding wavelet application to time-frequency analysis.

Example: The Mexican Hat Wavelet. One of the more common wavelets is the Mexican hat wavelet. This wavelet is essentially a second moment of a Gaussian in the frequency domain. The definition of this wavelet and its



trade off between frequency and time resolution.

Figure 26: Graphical depiction of the difference between a time series analysis, Fourier analysis, Gábor analysis and wavelet analysis of a signal. This figure is identical to Fig. 20 but with the inclusion of the time-frequency resolution achieved with the wavelet transform. The wavelet transform starts with a large Fourier domain window so that the entire frequency content is extracted. The time window is then scaled in half, leading to finer time resolution at the expense of worse frequency resolution. This process is continued until a desired time-frequency resolution is obtained.

transform are as follows:

$$\psi(t) = (1-t^2)e^{-t^2/2} = -d^2/dt^2\left(e^{-t^2/2}\right) = \psi_{1,0} \tag{2.5.24a}$$

$$\hat{\psi}(\omega) = \hat{\psi}_{1,0}(\omega) = \sqrt{2\pi}\omega^2 e^{-\omega^2/2} \tag{2.5.24b}$$

The Mexican wavelet has excellent localization properties in both time and frequency due to the minimal time-bandwidth product of the Gaussian function. Figure 27 (top panels) shows the basic Mexican wavelet function $\psi_{1,0}$ and its Fourier transform, both of which decay in t (ω) like $\exp(-t^2)$ ($\exp(-\omega^2)$). The Mexican hat wavelet can be dilated and translated easily as is depicted in Fig. 27 (bottom panel). Here three wavelets are depicted: $\psi_{1,0}$, $\psi_{3/2,-3}$ and $\psi_{1/4,6}$. This shows both the scaling and translation properties associated with any wavelet function.

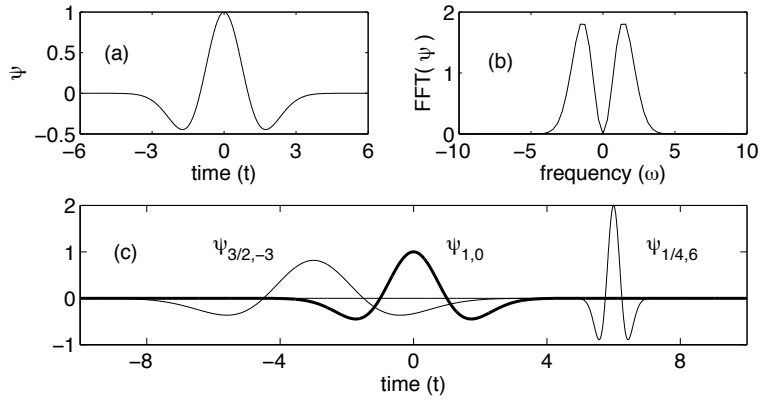


Figure 27: Illustration of the Mexican hat wavelet $\psi_{1,0}$ (top left panel), its Fourier transform $\hat{\psi}_{1,0}$ (top right panel), and two additional dilations and translations of the basic $\psi_{1,0}$ wavelet. Namely the $\psi_{3/2,-3}$ and $\psi_{1/4,6}$ are shown (bottom panel).

To finish the initial discussion of wavelets, some of the various properties of the wavelets are discussed. To begin, consider the time-frequency resolution and its localization around a given time and frequency. These quantities can be calculated from the relations:

$$\sigma_t^2 = \int_{-\infty}^{\infty} (t - < t >)^2 |\psi(t)|^2 dt \tag{2.5.25a}$$

$$\sigma_\omega^2 = \frac{1}{2\pi} \int_{-\infty}^{\infty} (\omega - < \omega >)^2 |\hat{\psi}(\omega)|^2 d\omega \tag{2.5.25b}$$

where the variances measure the spread of the time and frequency signal around $< t >$ and $< \omega >$ respectively. The Heisenberg uncertainty constrains the localization of time and frequency by the relation $\sigma_t^2 \sigma_\omega^2 \geq 1/2$. In addition, the CWT has the following mathematical properties

1. **Linearity:** the transform is linear so that

$$\mathcal{W}_\psi(\alpha f + \beta g)(a,b) = \alpha \mathcal{W}_\psi(f)(a,b) + \beta \mathcal{W}_\psi(g)(a,b)$$

2. **Translation:** the transform has the translation property

$$\mathcal{W}_\psi(T_c f)(a,b) = \mathcal{W}_\psi(f)(a, b-c)$$

where $T_c f(t) = f(t-c)$.

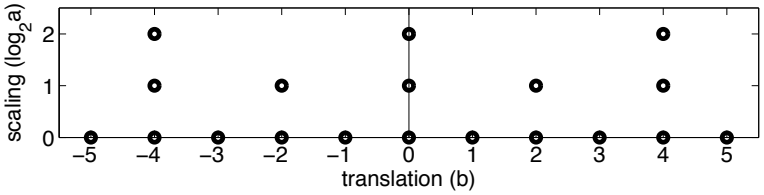


Figure 28: Discretization of the discrete wavelet transform with $a_0 = 2$ and $b_0 = 1$. This figure is a more formal depiction of the multi-resolution discretization as shown in Fig. 26.

3. **Dilation:** the dilation property follows

$$\mathcal{W}_\psi(D_c f)(a,b) = \frac{1}{\sqrt{c}}\mathcal{W}_\psi(f)(a/c, b/c)$$

where $c > 0$ and $D_c f(t) = (1/c)f(t/c)$.

4. **Inversion:** the transform can be inverted with the definision

$$f(t) = \frac{1}{C_\psi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\mathcal{W}_\psi(f)(a,b)\psi_{a,b}(t)\frac{dbda}{a^2}$$

where is becomes clear why the admissibility condition $C_\psi < \infty$ is needed.

To conclude this section, consider the idea of discretizing the wavelet transform on a computational grid. Thus the transform is defined on a lattice so that

$$\psi_{m,n}(x) = a_0^{-m/2}\psi\left(a_0^{-m}x - nb_0\right) \tag{2.5.26}$$

where $a_0, b_0 > 0$ and m, n are integers. The *discrete wavelet transform* is then defined by

$$\begin{aligned}\mathcal{W}_\psi(f)(m,n) &= (f, \psi_{m,n}) \\ &= \int_{-\infty}^{\infty} f(t)\bar{\psi}_{m,n}(t)dt \\ &= a_0^{-m/2}\int_{-\infty}^{\infty} f(t)\bar{\psi}(a_0^{-m}t - nb_0)dt\,. \end{aligned} \tag{2.5.27}$$

Futhermore, if $\psi_{m,n}$ are complete, then a given signal or function can be expanded in the wavelet basis:

$$f(t) = \sum_{m,n=-\infty}^{\infty}(f, \psi_{m,n})\psi_{m,n}(t)\,. \tag{2.5.28}$$

This expansion is in a set of wavelet frames. It still needs to be determined if the expansion is in terms of a set of basis functions. It should be noted that the scaling and dilation parameters are typically taken to be $a_0 = 2$ and $b_0 = 1$, corresponding to dilations of 2^{-m} and translations of $n2^m$. Figure 28 gives a graphical depiction of the time-frequency discretization of the wavelet transform. This figure is especially relevant for the computational evaluation of the wavelet transform. Further, it is the basis of fast algorithms for multi-resolution analysis.

2.6 Multi-Resolution Analysis and the Wavelet Basis

Before proceeding forward with wavelets, it is important to establish some key mathematical properties. Indeed, the most important issue to resolve is the ability of the wavelet to actually represent a given signal or function. In Fourier analysis, it has been established that any generic function can be represented by a series of cosines and sines. Something similar is needed for wavelets in order to make them a useful tool for the analysis of time-frequency signals.

http://mathworld.wolfram.com/L2-Function.html

L^2 —> its square is integrable and hence the function dot product is defined on it?

The concept of a wavelet is simple and intuitive: construct a signal using a single function $\psi \in L^2$ which can be written $\psi_{m,n}$ and that represents binary dilations by 2^m and translations of $n2^{-m}$ so that

$$\psi_{m,n} = 2^{m/2}\psi\left(2^m(x - n/2^m)\right) = 2^{m/2}\psi(2^m x - n) \tag{2.6.29}$$

For us to construct a basis, we must be able to construct pairwise orthogonal wavelets

(suppose we couldn't, but could generate a basis. From that basis we could use like Grahm Schmidt to generate an orthonormal basis, a contradiction

where m and n are integers. The use of this wavelet for representing a given signal or function is simple enough. However, there is a critical issue to be resolved concerning the *orthogonality* of the functions $\psi_{m,n}$. Ultimately, this is the primary issue which must be addressed in order to consider the wavelets as basis functions in an expansion. Thus we define the orthogonality condition as

$$(\psi_{m,n}, \psi_{k,l}) = \int_{-\infty}^{\infty} \psi_{m,n}(x)\psi_{k,l}(x)dx = \delta_{m,k}\delta_{n,l} \tag{2.6.30}$$

where δ_{ij} is the Dirac delta defined by

$$\delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases} \tag{2.6.31}$$

where i, j are integers. This statement of orthogonality is generic and in most functional spaces with a defined inner product.

The importance of orthogonality should not be underestimated. It is very important in applications where a functional expansion is used to approximate a given function or solution. In what follows, two examples are given concerning the key role of orthogonality.

Fourier expansions. Consider representing an even function $f(x)$ over the domain $x \in [0, L]$ with a cosine expansion basis. By Fourier theory, the function can be represented by

$$f(x) = \sum_{n=0}^{\infty} a_n \cos \frac{n\pi x}{L} \tag{2.6.32}$$

where the coefficients a_n can be constructed by using inner product rules and orthogonality. Specifically, by multiplying both sides of the equation by $\cos(m\pi x/L)$ and integrating over $x \in [0, L]$, i.e. taking the inner produce with respect to $\cos(m\pi x/L)$, the following result is found:

$$(f, \cos m\pi x/L) = \sum_{n=0}^{\infty} a_n (\cos n\pi x/L, \cos m\pi x/L)\,. \tag{2.6.33}$$

This is where orthogonality plays a key role, the infinite sum on the right hand side can be reduced to a single index where $n = m$ since the cosines are orthogonal to each other

without this orthogonality property, it would be near impossible to deal with this infinite series and calculate the coefficients

$$(\cos n\pi x/L, \cos m\pi x/L) = \begin{cases} 0 & n \neq m \\ L & n = m \end{cases}\,. \tag{2.6.34}$$

Thus the coefficients can be computed to be

$$a_n = \frac{1}{L} \int_0^L f(x) \cos \frac{n\pi x}{L} dx \tag{2.6.35}$$

and the expansion is accomplished. Moreover, the cosine basis is complete for even functions and any signal or function $f(x)$ can be represented, i.e. as $n \to \infty$ in the sum, the expansion converges to the given signal $f(x)$.

Eigenfunction expansions: The cosine expansion is a subset of the more general eigenfunction expansion technique that is often used to solve differential and partial differential equations problems. Consider the nonhomogeneous boundary value problem

$$Lu = f(x) \tag{2.6.36}$$

where L is a given self-adjoint, linear operator. This problem can be solved with an eigenfunction expansion technique by considering the associated eigenvalue problem of the operator L:

$$Lu_n = \lambda_n u_n\,. \tag{2.6.37}$$

The solution of (2.6.36) can then be expressed as

$$u(x) = \sum_{n=0}^{\infty} a_n u_n \tag{2.6.38}$$

provided the coefficients a_n can be determined. Plugging in this solution to (2.6.36) yields the following calculations

$$\begin{aligned} Lu &= f \\ L(\sum a_n u_n) &= f \\ \sum a_n L u_n &= f \\ \sum a_n \lambda_n u_n &= f\,. \end{aligned} \tag{2.6.39}$$

Taking the inner product of both sides with respect to u_m yields

$$\begin{aligned} (\sum a_n \lambda_n u_n, u_m) &= (f, u_m) \\ \sum a_n \lambda_n (u_n, u_m) &= (f, u_m) \\ a_m \lambda_m &= (f, u_m) \end{aligned} \tag{2.6.40}$$

where the last line is achieved by orthogonality of the eigenfunctions $(u_n, u_m) = \delta_{n,m}$. This then gives $a_m = (f, u_m)/\lambda_m$ and the eigenfunction expansion solution is

$$u(x) = \sum_{n=0}^{\infty} \frac{(f, u_n)}{\lambda_n} u_n\,. \tag{2.6.41}$$

Provided the u_n are a complete set, this expansion is guaranteed to converge to $u(x)$ as $n \to \infty$.

Orthonormal wavelets

The preceding examples highlight the importance of orthogonality for representing a given function. A wavelet ψ is called orthogonal if the family of functions $\psi_{m,n}$ are orthogonal as given by Eq. (2.6.30). In this case, a given signal or function can be uniquely expressed with the doubly infinite series

$$\boxed{f(t) = \sum_{n,m=-\infty}^{\infty} c_{m,n} \psi_{m,n}(t)} \tag{2.6.42}$$

where the coefficients are given from orthogonality by

$$\boxed{c_{m,n} = (f, \psi_{m,n})\,.} \tag{2.6.43}$$

The series is guaranteed to converge to $f(t)$ in the L^2 norm.

The above result based upon orthogonal wavelets establishes the key mathematical framework needed for using wavelets in a very broad and general way. It is this result that allows us to think of wavelets philosophically as the same as the Fourier transform.

Multi-Resolution Analysis (MRA)

The power of the wavelet basis is its ability to take a function or signal $f(t)$ and express it as a limit of successive approximations, each of which is a finer and finer version of the function in time. These successive approximations correspond to different resolution levels.

A *multi-resolution analysis*, commonly referred to as an MRA, is a method that gives a formal approach to constructing the signal with different resolution levels. Mathematically, this involves a sequence

$$\{V_m : \ m \in \text{integers}\} \tag{2.6.44}$$

of embedded subspaces of L^2 that satisfies the following relations:

1. The subspaces can be embedded in each other so that
$$\cdots \subset V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \cdots V_m \subset V_{m+1} \cdots$$
2. The union of all the embedded subspaces spans the entire L^2 space so that
$$\cup_{m=-\infty}^{\infty} V_m$$
is dense in L^2
3. The intersection of subspaces is the null set so that
$$\cap_{m=-\infty}^{\infty} V_m = \{0\}$$
4. Each subspace picks up a given resolution so that $f(x) \in V_m$ if and only if $f(2x) \in V_{m+1}$ for all integers m.
5. There exists a function $\phi \in V_0$ such that
$$\{\phi_{0,n} = \phi(x-m)\}$$
is an orthogonal basis for V_0 so that
$$\|f\|^2 = \int_{-\infty}^{\infty} |f(x)|^2 dx = \sum_{-\infty}^{\infty} |(f, \phi_{0,n})|^2$$
The function ϕ is called the *scaling function* or *father wavelet*.

If $\{V_m\}$ is a multiresolution of L^2 and if V_0 is the closed subspace generated by the integer translates of a single function ϕ, then we say ϕ generates the MRA.

One remark of importance: since $V_0 \subset V_1$ and ϕ is a scaling function for V_0 and also for V_1, then

$$\phi(x) = \sum_{-\infty}^{\infty} c_n \phi_{1,n}(x) = \sqrt{2} \sum_{-\infty}^{\infty} c_n \phi(2x-n) \tag{2.6.45}$$

where $c_n = (\phi, \phi_{1,n})$ and $\sum_{-\infty}^{\infty} |c_n|^2 = 1$. This equation, which relates the scaling function as a function of x and $2x$ is known as the *dilation equation*, or *two-scale equation*, or *refinement equation* because it reflects $\phi(x)$ in the refined space V_1 which as the finer scale of 2^{-1}.

Since $V_m \subset V_{m+1}$, we can define the orthogonal complement of V_m in V_{m+1} as

$$V_{m+1} = V_m \oplus W_m \tag{2.6.46}$$

where $V_m \perp W_m$. This can be generalized so that

$$\begin{aligned} V_{m+1} &= V_m \oplus W_m \\ &= (V_{m-1} \oplus W_{m-1}) \oplus W_m \\ &\vdots \\ &= V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_m \\ &= V_0 \oplus (\oplus_{n=0}^{m} W_n) \,. \end{aligned} \tag{2.6.47}$$

As $m \to \infty$, it can be found that

$$V_0 \oplus (\oplus_{n=0}^{\infty} W_n) = L^2 \,. \tag{2.6.48}$$

In a similar fashion, the resolution can rescale upwards so that

$$\oplus_{n=-\infty}^{\infty} W_n = L^2 \,. \tag{2.6.49}$$

Moreover, there exists a scaling function $\psi \in W_0$ (the mother wavelet) such that

$$\psi_{0,n}(x) = \psi(x - n) \tag{2.6.50}$$

constitutes an orthogonal basis for W_0 and

$$\psi_{m,n}(x) = 2^{m/2} \psi(2^m x - n) \tag{2.6.51}$$

is an orthogonal basis for W_m. Thus the mother wavelet ψ spans the orthogonal complement subset W_m while the scaling function ϕ spans the subsets V_m. The connection between the father and mother wavelet are shown in the following theorem.

Theorem: If $\{V_m\}$ is a MRA with scaling function ϕ, then there is a mother wavelet ψ

$$\psi(x) = \sqrt{2} \sum_{-\infty}^{\infty} (-1)^{n-1} \bar{c}_{-n-1} \phi(2x - n) \tag{2.6.52}$$

where

$$c_n = (\phi, \phi_{1,n}) = \sqrt{2} \int_{-\infty}^{\infty} \phi(x) \bar{\phi}(2x - 1) dx \,. \tag{2.6.53}$$

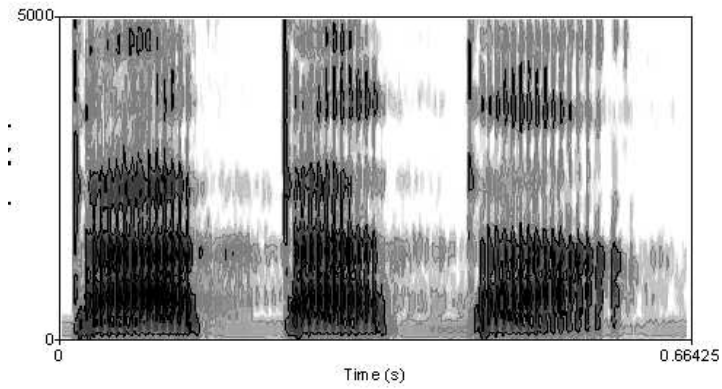


Figure 29: Spectrogram (time-frequency) analysis of a male human saying "ta ta ta."

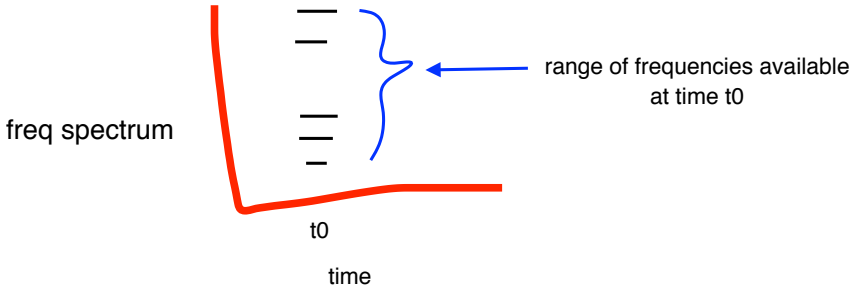
That is, the system $\psi_{m,n}(x)$ is an orthogonal basis of L^2.

This theorem is critical for what we would like to do. Namely, use the wavelet basis functions as a complete expansion basis for a given function $f(x)$ in L^2. Further, it explicitly states the connection between the *scaling function* $\phi(x)$ (father wavelet) and *wavelet function* $\psi(x)$ (mother wavelet). It is only left to construct a desirable wavelet basis to use. As for wavelet construction, the idea is to build them to take advantage of certain properties of the system in the wavelet construction so that it gives an efficient and meaningful representation of your time-frequency data.

2.7 Spectrograms and the Gábor transforms in MATLAB

The aim of this lecture will be to use MATLAB's fast Fourier transform routines modified to handle the Gábor transform. The Gábor transform allows for a fast and easy way to analysis both the time and frequency properties of a given signal. Indeed, this windowed Fourier transform method is used extensively for analyzing speech and vocalization patterns. For such applications, it is typical to produce a *spectrogram* that represents the signal in both the time and frequency domain. Figures 29 and 30 are produced from the vocalization patterns in time-frequency of a humans saying "ta ta ta" and a killer whale vocalizing to other whales. The time-frequency analysis can can be used to produce speech recognition algorithms given the characteristic signatures in the time-frequency domains of sounds. Thus spectrograms are a sort of fingerprint of sound.

To understand the algorithms which produce the spectrogram, it is informative to return to the characteristic picture shown in Fig. 19. This demonstrates



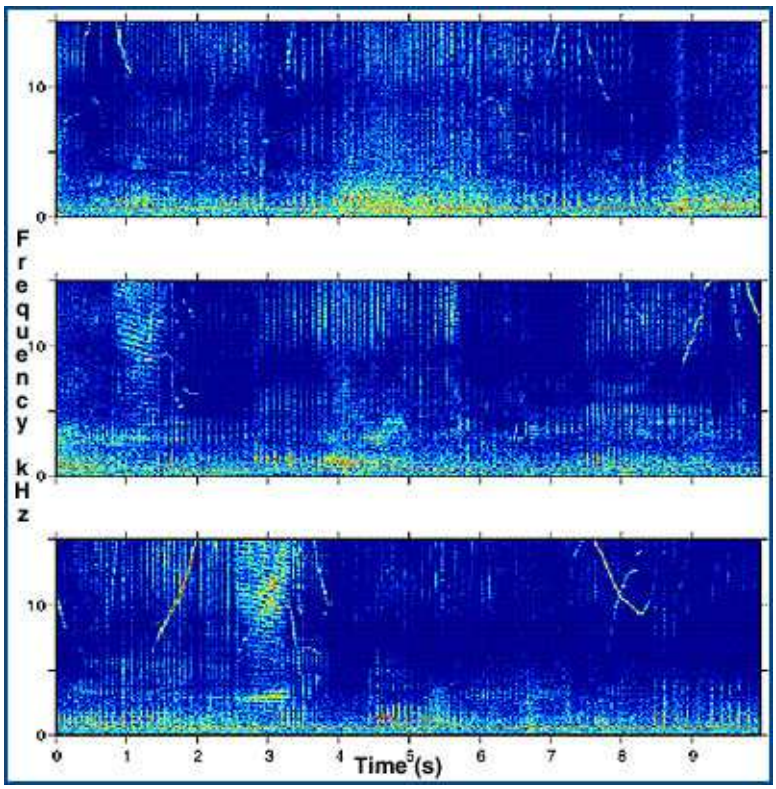


Figure 30: Spectrogram (time-frequency) analysis of a killer whale vocalization over three periods of time.

the action of an applied time filter in extracting time localization information. To build a specific example, consider the following MATLAB code that builds a time domain (t), its corresponding Fourier domain (ω), a relatively complicated signal ($S(t)$), and its Fourier transform ($\hat{S}(\omega)$).

```
clear all; close all; clc
L=10; n=2048;
t2=linspace(0,L,n+1); t=t2(1:n);
k=(2*pi/L)*[0:n/2-1 -n/2:-1]; ks=fftshift(k);

S=(3*sin(2*t)+0.5*tanh(0.5*(t-3))+ 0.2*exp(-(t-4).^2)...
  +1.5*sin(5*t)+4*cos(3*(t-6).^2))/10+(t/20).^3;
St=fft(S);
```

signal goes from time 0 to L in 2048 points

FFT thinks its working on domain 2 pi. This scales our domain [0, 10] into 2pi

k = wave numbers

The signal and its Fourier tranform can be plotted with the commands

L = the spatial domain of our signal; i..e “window” we want to look at our signal in; i.e. if our signal is L periodic?
n = number of different frequencies to look for between [0, L]. i..e calculate freq values for wavenumber k = L/n, 2L / n

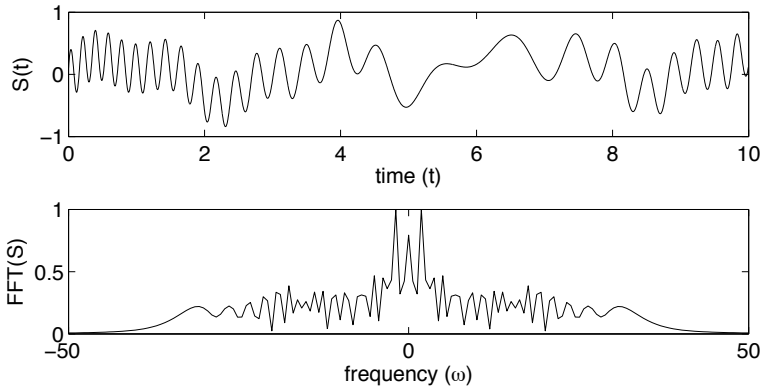


Figure 31: Time signal and its Fourier transform considered for a time-frequency analysis in what follows.

```
figure(1)
subplot(3,1,1)  % Time domain
plot(t,S,'k')
set(gca,'Fontsize',[14]),
xlabel('Time (t)'), ylabel('S(t)')

subplot(3,1,2)  % Fourier domain
plot(ks,abs(fftshift(St))/max(abs(St)),'k');
axis([-50 50 0 1])
set(gca,'Fontsize',[14])
xlabel('frequency (\omega)'), ylabel('FFT(S)')
```

Figure 31 shows the signal and its Fourier transform for the above example. This signal $S(t)$ will be analyzed using the Gábor transform method.

The simplest Gábor window to implement is a Gaussian time-filter centered at some time τ with width a. As has been demonstrated, the parameter a is critical for determining the level of time-resolution versus frequency resolution in a time-frequency plot. Figure 32 shows the signal under consideration with three filter widths. The more narrow the time-filtering, the better resolution in time. However, this also produces the worst resolution in frequency. Conversely, a wide window in time produces much better frequency resolution at the expense of reducing the time resolution. A simple extension to the existing code produces a signal plot along with three different filter widths of Gaussian shape.

```
figure(2)
width=[10 1 0.2];
for j=1:3
```

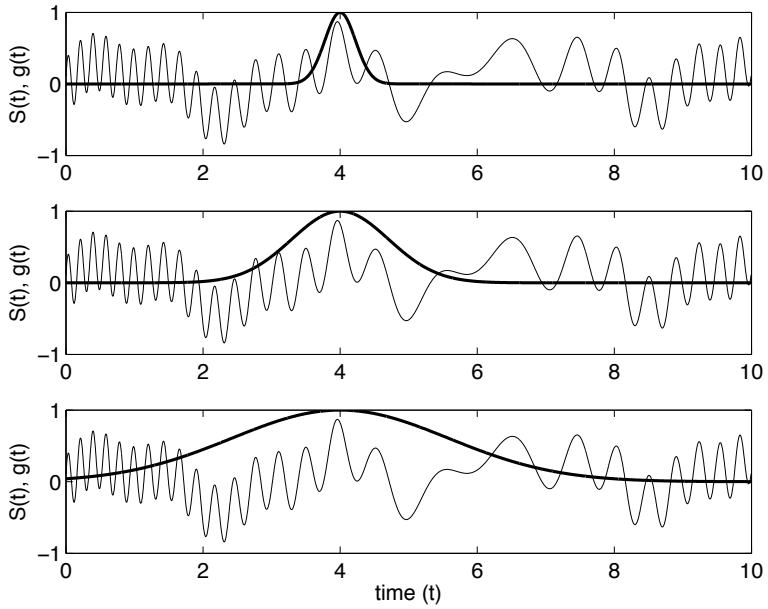


Figure 32: Time signal $S(t)$ and the Gábor time filter $g(t)$ (bold lines) for three different Gaussian filters: $g(t) = \exp(-10(x-4)^2)$ (top), $g(t) = \exp(-(x-4)^2)$ (middle), and $g(t) = \exp(-0.2(x-4)^2)$ (bottom). The different filter widths determine the time-frequency resolution. Better time resolution gives worse frequency resolution and vice-versa due to the Heinsenberg uncertainty principle.

i.e. our Gabor window
choose window to
fit filtering needs
(i.e. width; change
to exp(^ 10)
to make wider window
with steeper drop off

Fourier transforms run into Gibbs Phenomenom with sharp edges
frequency filter; WANT a SMOOTH filter with FFT

```
  g=exp(-width(j)*(t-4).^2);
  subplot(3,1,j)
  plot(t,S,'k'), hold on    signal over time
  plot(t,g,'k','Linewidth',[2])  filter over time
  set(gca,'Fontsize',[14])
  ylabel('S(t), g(t)')
end
xlabel('time (t)')
```

The key now for the Gábor transform is to multiply the time filter Gábor function $g(t)$ with the original signal $S(t)$ in order to produce a windowed section of the signal. The Fourier transform of the windowed section then gives the local frequency content in time. The following code constructs the windowed Fourier transform with the Gábor filtering function

$$g(t) = e^{-a(t-b)^2}. \tag{2.7.54}$$

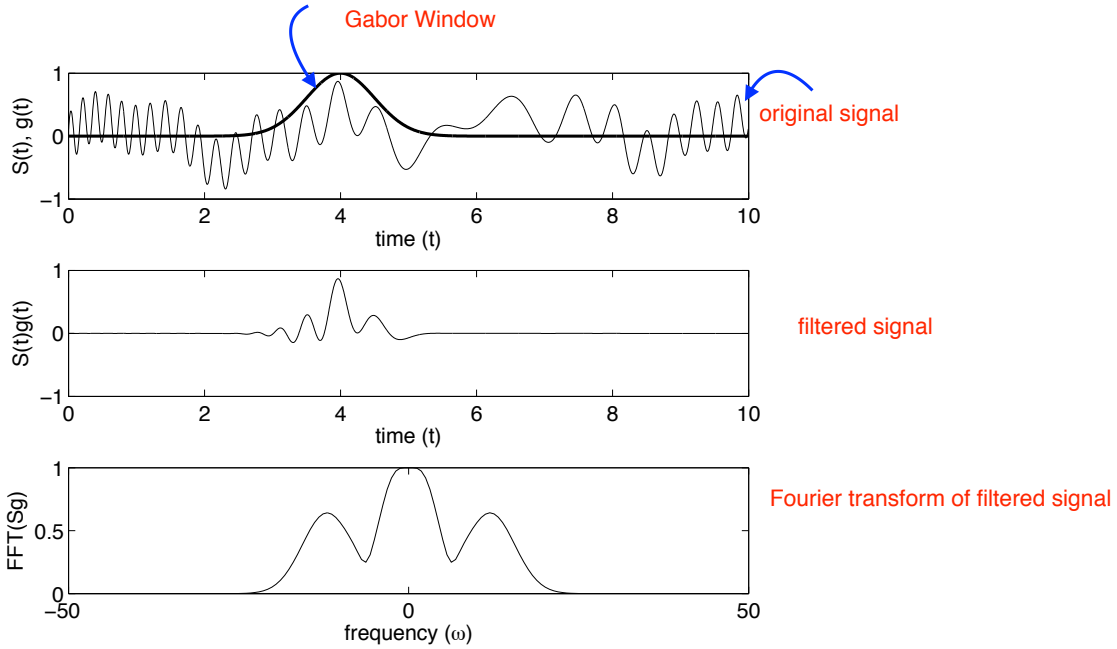


Figure 33: Time signal $S(t)$ and the Gábor time filter $g(t) = \exp(-2(x-4)^2)$ (bold line) for a Gaussian filter. The product $S(t)g(t)$ is depicted in the middle panel and its Fourier transform $\hat{Sg}(\omega)$ is depicted in the bottom panel. Note that the windowing of the Fourier transform can severely limit the detection of low-frequency components.

The Gaussian filtering has a width parameter a and translation parameter b. The following code constructs the windowed Fourier transform using the Gaussian with $a = 2$ and $b = 4$.

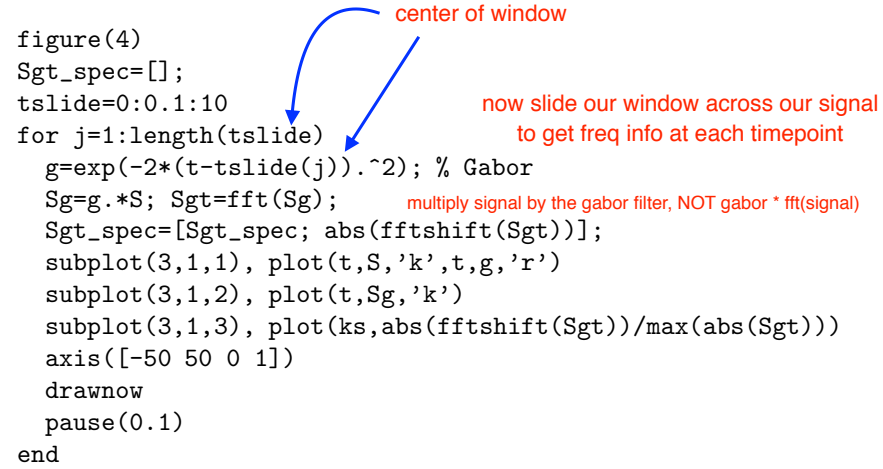
```
figure(3)
g=exp(-2*(t-4).^2);
Sg=g.*S;
Sgt=fft(Sg);

subplot(3,1,1), plot(t,S,'k'), hold on
  plot(t,g,'k','Linewidth',[2])
  set(gca,'Fontsize',[14])
  ylabel('S(t), g(t)'), xlabel('time (t)')
subplot(3,1,2), plot(t,Sg,'k')
  set(gca,'Fontsize',[14])
  ylabel('S(t)g(t)'), xlabel('time (t)')
subplot(3,1,3), plot(ks,abs(fftshift(Sgt))/max(abs(Sgt)),'k')
  axis([-50 50 0 1])
```

```
set(gca,'Fontsize',[14])
ylabel('FFT(Sg)'), xlabel('frequency (\omega)')
```

Figure 33 demonstrates the application of this code and the windowed Fourier transform in extracting local frequencies of a local time window.

The key to generating a spectrogram is to now vary the position b of the time filter and produce spectra at each location in time. In theory, the parameter b is continuously translated to produce the time-frequency picture. In practice, like everything else, the parameter b is discretized. The level of discretization is important in establishing a good time-frequency analysis. Specifically, finer resolution will produce better results. The following code makes a dynamical *movie* of this process as the parameter b is translated.




```
figure(4)
Sgt_spec=[];
tslide=0:0.1:10
for j=1:length(tslide)
  g=exp(-2*(t-tslide(j)).^2); % Gabor
  Sg=g.*S; Sgt=fft(Sg);
  Sgt_spec=[Sgt_spec; abs(fftshift(Sgt))];
  subplot(3,1,1), plot(t,S,'k',t,g,'r')
  subplot(3,1,2), plot(t,Sg,'k')
  subplot(3,1,3), plot(ks,abs(fftshift(Sgt))/max(abs(Sgt)))
  axis([-50 50 0 1])
  drawnow
  pause(0.1)
end
```

This movie is particularly illustrative and provides an excellent graphical representation of how the Gábor time-filtering extracts both local time information and local frequency content. It also illustrates, as the parameter a is adjusted, the ability (or inability) of the windowed Fourier transform to provide accurate time and frequency information.

The code just developed also produces a matrix **Sgt_spec** which contains the Fourier transform at each slice in time of the parameter b. It is this matrix that produces the spectrogram of the time-frequency signal. The spectrogram can be viewed with the commands

```
pcolor(tslide,ks,Sgt_spec.'), shading interp
set(gca,'Ylim',[-50 50],'Fontsize',[14])
colormap(hot)
```

Modifying the code slightly, a spectrogram of the signal $S(t)$ can be made for three different filter widths $a = 5, 1, 0.2$ in Eq. (2.7.54). The spectrograms are shown in Fig. 34 where from left to right the filtering window is broadened from

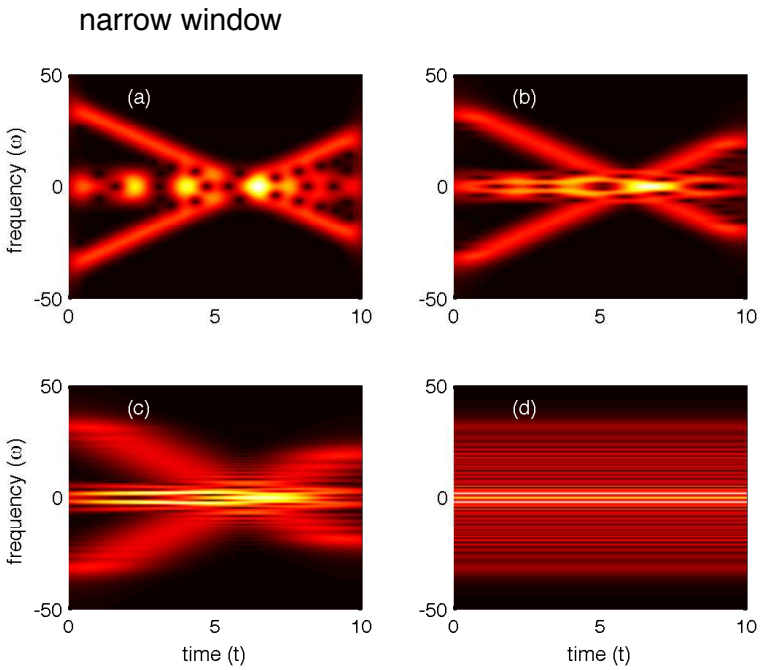


Figure 34: Spectrograms produced from the Gábor time-filtering Eq. (2.7.54) with (a) $a = 5$, (b) $a = 1$ and (c) $a = 0.2$. The Fourier transform, which has no time localization informatoin is depicted in (d). It is easily seen that the window width trades off time and frequency resolution at the expense of each other. Regardless, the spectrogram gives a visual time-frequency picture of a given signal.

a = window width

matlab has a generic spectrogram function but ofc you get a lot more freedom doing your own analysis

$a = 5$ to $a = 0.2$. Note that for the left plot, strong localization of the signal in time is achieved at the expense of suppressing almost all the low-frequency components of the signal. In contrast, the right most figure with a wide temporal filter preserves excellent resolution of the Fourier domain but fails to localize signals in time. Such are the tradeoffs associated with a fixed Gábor window transform.

2.8 MATLAB Filter Design and Wavelet Toolboxes

The applications of filtering and time-frequency analysis are so ubiquitous across the sciences, that MATLAB has developed a suite of toolboxes that specialize in these applications. Two of these toolboxes will be demonstrated in what follows. Primarily, screenshots will gives a hint of the functionality and versatility of the toolboxes.

The most remarkable part of the toolboxes is that it allows for entry into high

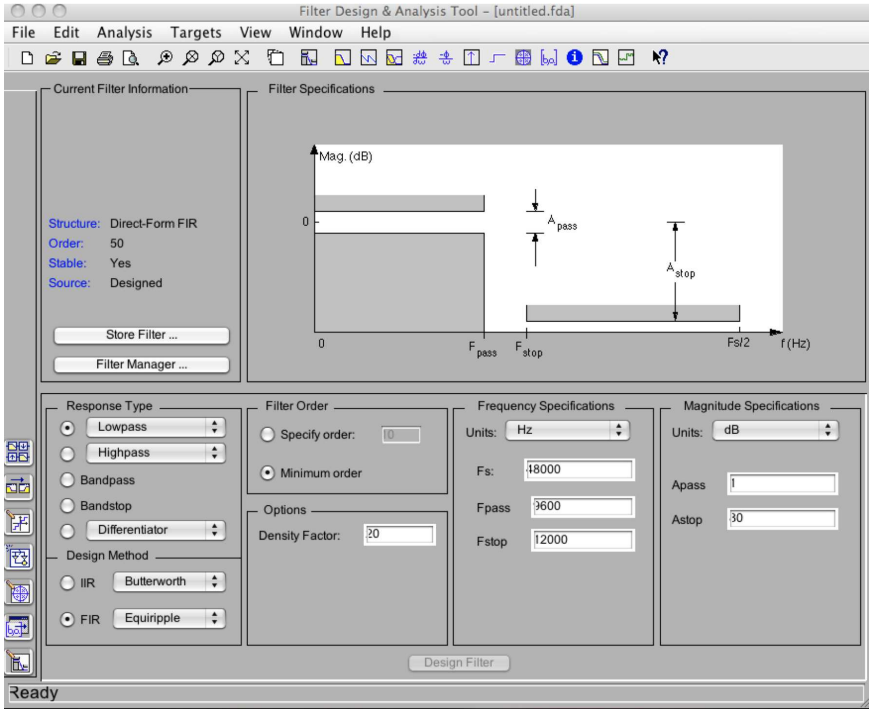


Figure 35: Screenshot of the filter design and analysis toolbox GUI. The top left contains information on the current filter design while the top right is a graphical representation of the filter characteristics. The lower portion of the GUI is reserved for the user interface functionality and design. Indeed, the type of filter, its spectral width, whether it is low-pass (filters high frequencies), high-pass (filters low frequencies), band-pass (filters a specified band of frequencies) or band-stop (attenuates a band of frequencies) can be chosen.

level signal processing and wavelet processing almost immediately. Indeed, one hardly needs to know anything to begin the process of analyzing, synthesizing, and manipulating data to one's own ends. For the most part, each of the toolboxes allows you to begin usage once you upload your signal, image or data. The only drawback is cost. For the most part, many academic departments have access to the toolboxes. And if you are part of a University environment, or enrolled in classes, the student versions of the toolboxes can be purchased directly from MATLAB at a very reasonable price.

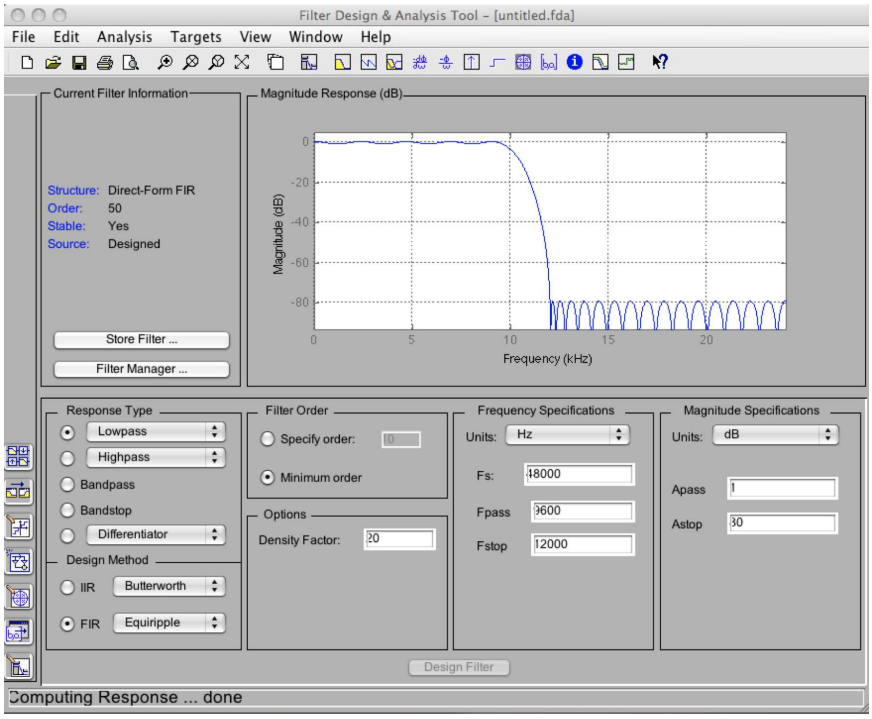


Figure 36: Screenshot of the magnitude response of the filter. The magnitude response, along with a host of other important characteristics of the filter can be obtained from the drag-down menu of the **Analysis** button on the top.

Filter Design and Analysis Toolbox

The filter design toolbox is ideally suited to help create and engineer and ideal filter for whatever application you may need. The user has the option of using command line codes to directly access the filter design subroutines, or one can use the extremely efficient and useful GUI (graphical user interface) from MATLAB. Figure 35 shows a typical screenshot that MATLAB produces upon startup of the filter design toolbox. The command to start the toolbox is given by

```
FDAtool
```

This launches the highly intuitive GUI that guides you through the process of designing your ideal filter. Figure 36 demonstrates, for instance, the magnitude response of the filter along with its frequency and magnitude of response. The magnitude response can be accessed by dragging down the **Analysis** button at the top of the GUI panel. In addition to the amplitude response, the phase

response, a combination of phase and amplitude response, group-velocity response, etc. Depending upon your needs, complete engineering of the filter can be performed with set objectives and performance aims.

As with all toolbox design and information, it can be exported from MATLAB for use in other programs or with other parts of MATLAB. The exporting of data, as well as saving of a given filter design, can all be performed from the **file** drag-down menu at the top right of the GUI. Moreover, if you desire to skip the filter design GUI, then all the filter functions can be accessed directly from the command line window. For instance, the command

```
[B,A]=butter(N,Wn)
```

generates a Butterworth digital and analog filter design. The parameter N denotes the Nth order lowpass digital Butterworth filter and returns the filter coefficients in length $N+1$ vectors B (numerator) and A (denominator). The coefficients are listed in descending powers of z. The cutoff frequency Wn must be $0.0 < Wn < 1.0$, with 1.0 corresponding to half the sample rate. And besides this, you now know the butter command in MATLAB.

Wavelet Toolbox

Just like the signal processing, filter design toolbox, the wavelet toolbox it an immensely powerful toolbox for signal processing, image processing, multiresolution analysis and any manner of time-frequency analysis. The wavelet toolbox is comprised of a large number of modules that allow for various applications and methodologies. To access the toolbox, simply type

```
wavemenu
```

and a selection of menu items is generated. Figure 37 demonstrates the assortment of possible uses of the wavelet toolbox that are available upon launching the wavelet toolbox. In addition to such applications as signal processing and time frequency analysis of one-dimensional signals or functions, one can proceed to denoise or manipulate image data. The program can also produce wavelet transforms and output the coefficients of a continuous or discrete wavelet transform.

As a specific example, consider the *wavelet 1-D* button and the example files it contains. The example files are accessed from the **file** button at the top left of the GUI. In this case, an example analysis of a noisy step function signal is considered. It should be noted that your own signal could have easily been imported from the **file** button with the **load** menu item. In this example, a signal is provided with the full wavelet decomposition at different levels of resolution. It is easy to see that the given signal is decomposed into its *big* features followed by its *finer* features as one descends vertically down the plots. In the standard example, the *sym* wavelet is used with five levels of resolution.

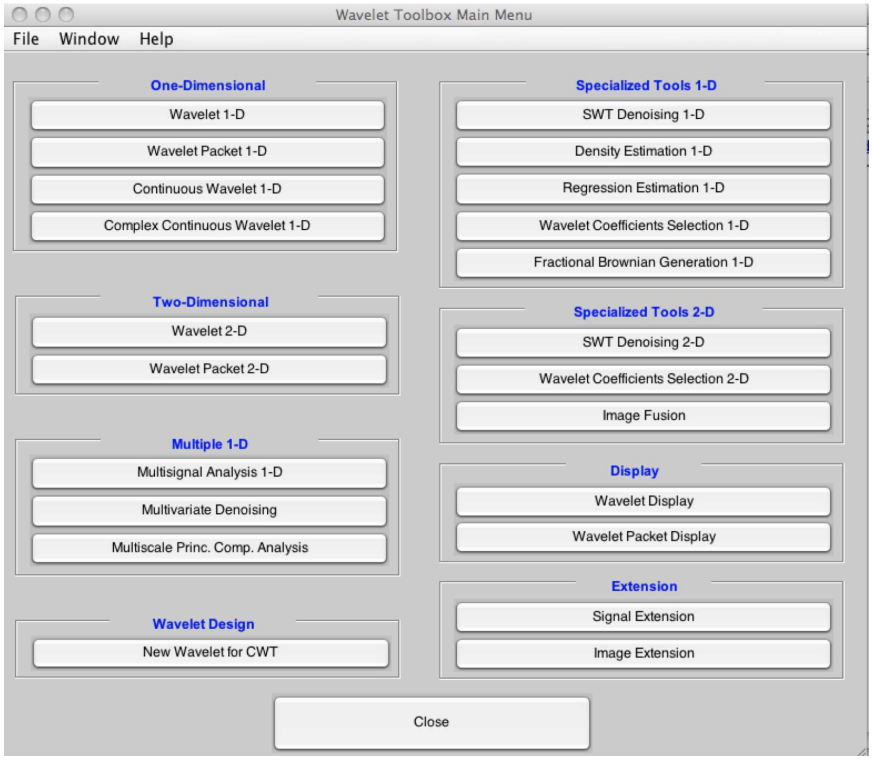


Figure 37: Screenshot of the entry point of the wavelet toolbox. A myriad of applications are available by simply clicking on one of the menu items. Both 1D and 2D problems can be analyzed with the toolbox..

This shows the progression, as discussed in the wavelets lecture, of the multi-resolution analysis. The wavelet analysis can be easily modified to consider other wavelet bases. In fact, in addition to the sym wavelet, one can choose the Haar wavelets, Daubechies wavelets, and Coifman wavelets among others. This remarkable packaging of the best time-frequency domain methods is one of the tremendous aspects of the toolbox. In addition to the decomposition, a histogram, the statistics, or compression of the signal can be performed by simply clicking one of the appropriate buttons on the right.

The wavelet toolbox also allows for considering a signal with a wavelet expansion. The continuous wavelet 1-D provides a full decomposition of the signal into its wavelet basis. Figure 40 shows a given signal along with the calculation of the wavelet coefficients $C_{a,b}$ where a and b are dilation and translation operations respectively. This provides the basis for a time-frequency analysis. Additionally, given a chosen level of resolution determined by the parameter

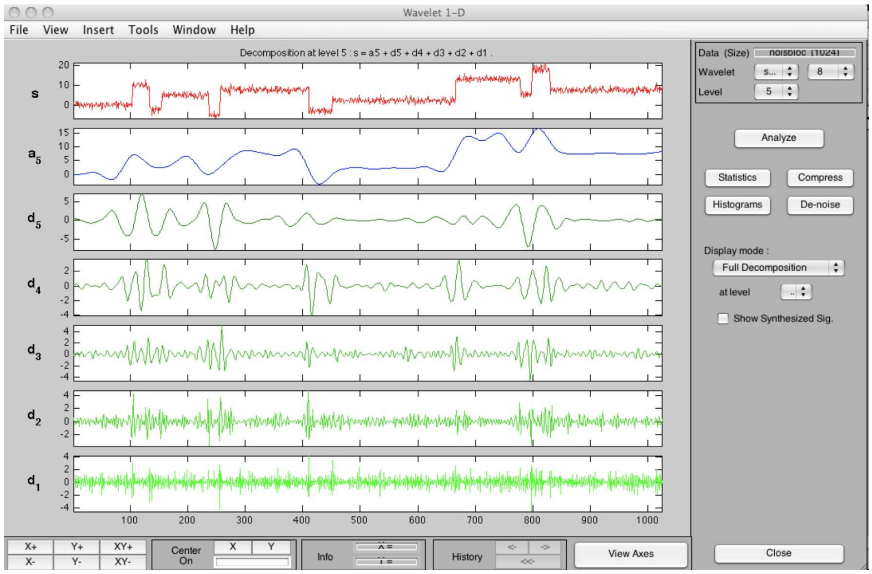


Figure 38: Screenshot of the wavelet 1-D with a noisy step function signal. A multi-resolution decomposition of the signal is performed with *sym* wavelets.

a, then the third panel shows the parameter $C_{a,b}$ for $a = a_{\max}/2$. The local maxima of the $C_{a,b}$ function is also shown. As before, different wavelet bases may be chosen along with different levels of resolution of a given signal.

Two final applications are demonstrated: one is an imagine denoising application, and the second is an imagine multi-resolution (compression) analysis. Both of these applications have important applications for imagine clean-up and size reduction, therefore they are prototypical examples of the power of the 2-D wavelet tools. In the first applications, illustrated in Fig. 41, a noisy image is considered. The application allows you to directly important your own imagine or data set. Once imported, the image is decomposed at the different resolution levels. To denoise the image, *filters* are applied at each level of resolution to remove the high-frequency components of each level of resolution. This produces an exceptional, wavelet based algorithm for denoising. As before, a myriad of wavelets can be considered and full control of the denoising process is left to the user. Figure 42 demonstrates the application of a multi-resolution analysis to a given image. Here the image is decomposed into various resolution levels. The program allows the user complete control and flexibility in determining the level of resolution desired. By keeping fewer levels of resolution, the image quality is compromised, but the algorithm thus provides an excellent compressed image. The discrete wavelet transform (**dwt**) is used to decompose the image. The inverse discrete wavelet transform (**idwt**) is used to extract back the im-

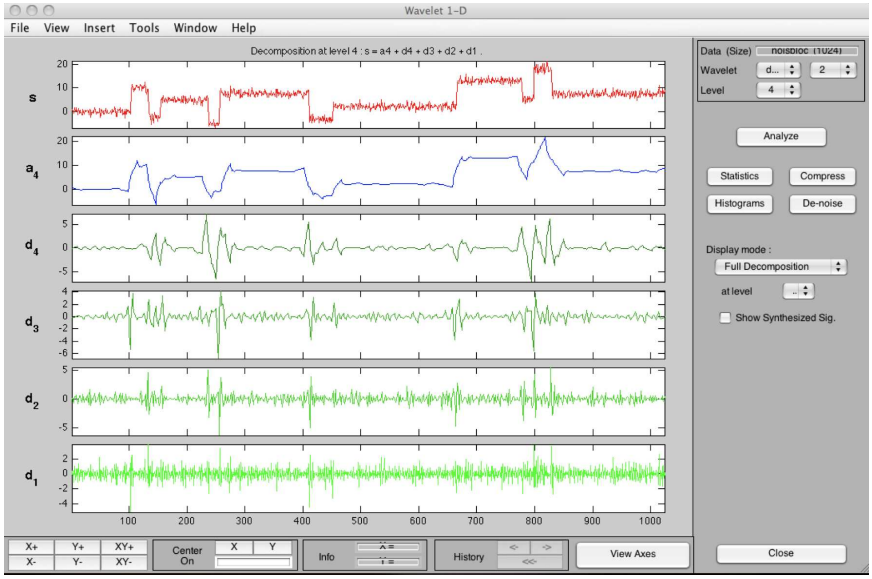


Figure 39: Screenshot of the wavelet 1-D with Daubechies wavelets.

age at an appropriate level of multi-resolution. For this example, two levels of decomposition are applied.

The wavelets transform and its algorithms can be directly accessed from the command line, just like the filter design toolbox. For instance, to apply a continuous wavelet transform, the command

```
coeffs=cwt(S,scales,'wname')
```

can be used to computed the real or complex continuous 1-D wavelet coefficients. Specifically, it computes the continuous wavelet coefficients of the vector S at real, positive *scales*, using wavelet whose name is *wname* (for instance, haar). The signal S is real, the wavelet can be real or complex.

3 Image Processing and Analysis

Over the past couple of decades, imaging science has had a profound impact on science, medicine and technology. The reach of imagining science is immense, spanning the range of modern day biological imaging tools to computer graphics. But the imaging methods are not limited to visual data. Instead, a general mathematical framework has been developed by which more general data analysis can be performed. The next set of subsections deal with some basic tools for image processing or data analysis.

MATLAB commands

flipud
= flips matrix values as image is stored upside down?

pcolor
=

not necessarily just pictures;
these images could be the products of other mathematical objects, they could be data-based like ultrasounds

image type Frequencies

ultrasound images: 20Hz-20KHz, 1MHz
infrared/thermal: > 700 nm
tomography (CAT scan; bones)
MRIs (biological tissue)
radar/sonar: 1cm - 1m wavelength
digital photos

WATCH OUT FOR image data type in MATLAB
usually images stored as arrays of uint8, 8-bit integers

1) have to convert to doubles before performing analysis/operations —> double(image_matrix)

2) elements of uint8 CANNOT go above 255, so be careful with your order of operations!

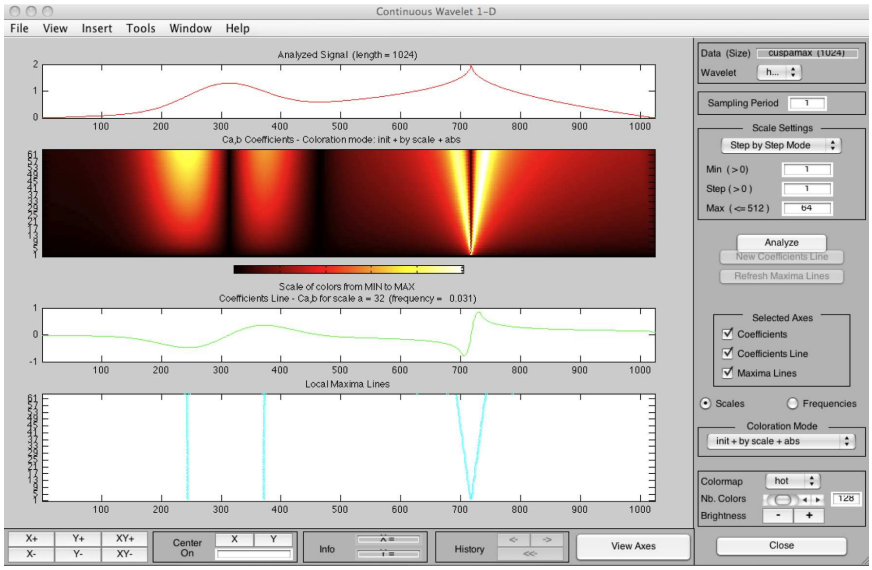


Figure 40: Screenshot of the continuous wavelet 1-D application where a signal is decomposed into a given wavelet basis and its wavelet coefficients are determined.

3.1 Basic concepts and analysis of images

Imaging science is now ubiquitous. Indeed, it now dominates many scientific fields as a primary vehicle for achieving information and informing decisions. The following are a sample of the technologies and applications where imaging science is having a formative impact:

- **Ultrasound:** ultrasound refers to acoustic waves whose frequencies are higher than the audible range of human ears. Typically this is between 20Hz to 20KHz. Most medial ultrasound devices are above 1MHz and below 20MHz. Anybody who has had a baby will proudly show you their ultrasound pictures.

- **Intrared/thermal imaging:** Imperceptible to the human eye, infrared (IR) signatures correspond to wavelengths longer than 700nm. However, aided by signal amplification and enhancement devices, humans can perceive IR signals from above 700nm to a few microns.

- **Tomography (CAT scans):** Using the first Nobel Prize (1895) ideas of Wilhelm Röntgen for X-rays, tomography has since become the standard in the medical industry as the wavelengths, ranging from nanomenters to

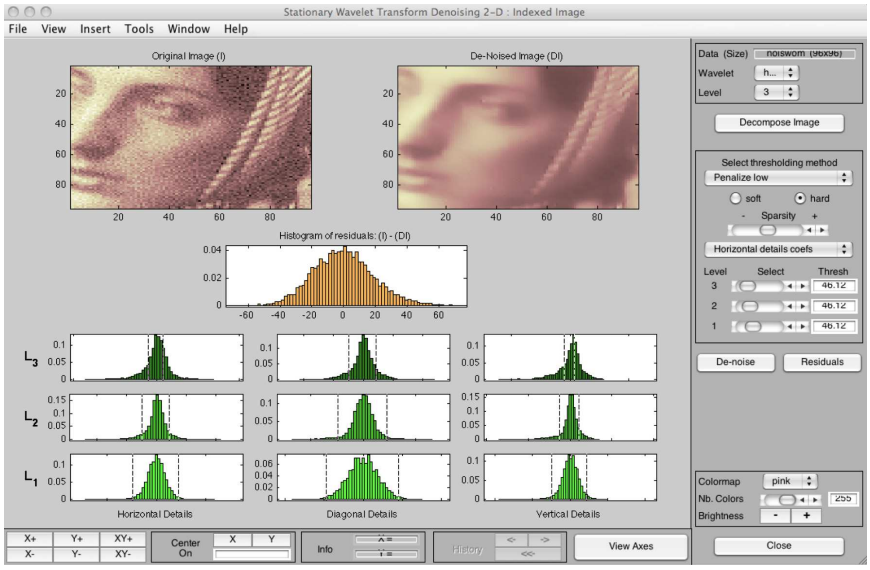


Figure 41: Screenshot of the denoising application using the wavelet bases.

picometers, can easily penetrate most biological tissue. Imaging hardware and software have made this the standard for producing medical images.

- **Magnetic resonance imaging (MRI):** By applying a strong magnetic field to the body, the atomic spins in the body are aligned. High-frequency RF pulses are emitted into a slice plan of the external field. Upon turning off the RF waves, the relaxation process reveals differentials in tissue and biological material, thus giving the characteristics needed for imaging.

- **Radar and sonar imaging:** Radar uses radio waves for the detection and ranging of targets. The typical wavelengths are in the microwave range of 1cm to 1m. Applied to imaging, radar becomes a sophisticated tool for remote sensing applications. Sonar works in a similar fashion, but with underwater acoustic waves.

- **Digital photos:** Most of us are well aware of this application as we often would like to remove red-eye, undo camera blurring from out-of-focus shots or camera shakes, etc. Many software packages already exist to help you improve your image quality.

- **Computer graphics:** Building virtual worlds and targeted images falls within the aegis of the computer science community. There are many mathematically challenging issues related to representation of real-life images.

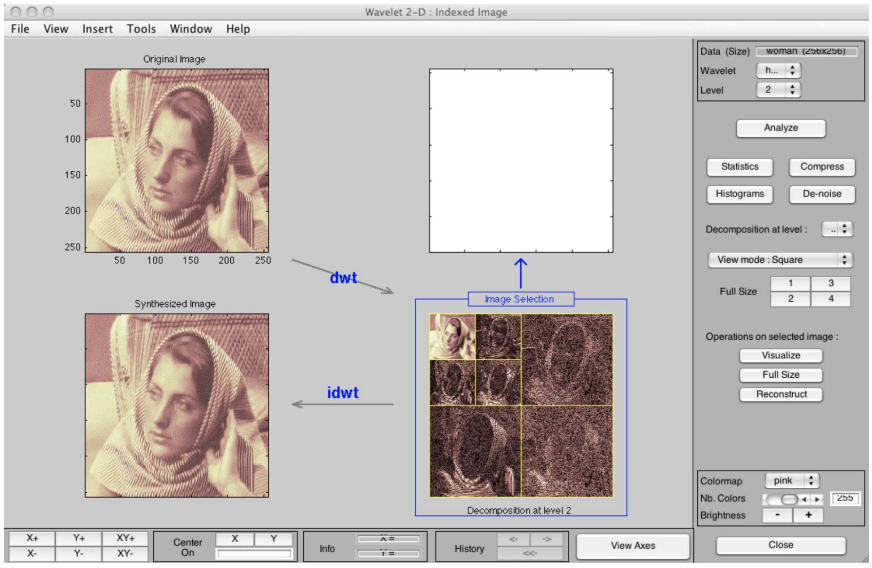


Figure 42: Screenshot of the decomposition of an image into different resolution levels. This application is ideal for image compression.

In the applications above, it goes without saying that most images acquired from experiment or enhancement devices are rarely without imperfections. Often the imperfections can be ignored. However, if the noise or imperfections in the images are critical for a decision making process, i.e. detection via radar of aircraft or detection of a tumor in biological tissue, then any enhancement of the image that could contribute to a statistically better decision is of high value.

Image processing and analysis addresses the fundamental issue of allowing an end user to have enhanced information, or statistically more accurate data, of a given image or signal. Thus mathematical method are the critical piece in achieving this enhancement. Before delving into specifics of mathematical methods that can be brought to bear on a given problem, it is important to classify the types of image analysis objectives that one can consider. The following gives a succinct list of image processing tasks:

OBJECTIVES!

What we might try to do with the aforementioned data types.

- **image contrast enhancement:** Often there can be little contrast between different objects in a photo or in a given data set. Contrast enhancement is a method that tries to promote or enhance differences in an image in order to achieve sharper contrasts.

- **image denoising:** This is often of primary interest in analyzing data sets. Can noise, from either measurements or inaccuracies, be removed to give a more robust and fundamental picture of the underlying dynamics

of a system.

- **image deblurring:** The leading cause of bad pictures: camera shakes and out-of-focus cameras. Image processing methods can often compensate and undo these two phenomena to produce a sharper image that is greatly deblurred. This is a remarkable application of image analysis, and is already an application that is included in many digital photo software bundles.

- **inpainting:** If data is missing over certain portions of an image or set of data, inpainting provides a mathematical framework for systematically generating the missing data. It accounts for both edges and the overall structure surrounding the missing pixels.

- **segmentation (edge detection):** Segmentation breaks up an image into blocks of different structures. This is particularly important for biomedical applications as well as military applications. Good algorithms for determining segmentation locations are critical.

The mathematical task is then to develop methods and algorithms capable of performing a set of tasks that addresses the above list. In the subsections that follow, only a small subset of these will be considered. However, a comprehensive mathematical treatment of the above issues can be performed [6].

Mathematical approaches

Given the broad appeal of imaging sciences and its applicability in such a wide variety of fields, it is not surprising that an equally diverse set of mathematical methods has been brought to bear on image processing. As with the previous lists of applications and tasks associated with image processing, the following list highlights some of the broad mathematical ideas that are present in thinking about image analysis.

Morphological approach: In this approach, one can think of the entire 2D domain of an image as a set of subdomains. Each subdomain should represent some aspect of the overall image. The fundamental idea here is to decompose the domain into fundamental shapes based upon some structure element S. The domain is then decomposed in a binary way, a small patch of the image (defined by the structure element S) is *in* the object of interest or *out* of the object of interest. This provides an efficient method for edge detection in certain sets of data or images.

Fourier analysis: As with time-frequency analysis, the key idea is to decompose the image into its Fourier components. This is easily done with a two-dimensional Fourier transform. Thus the image becomes a collection of Fourier

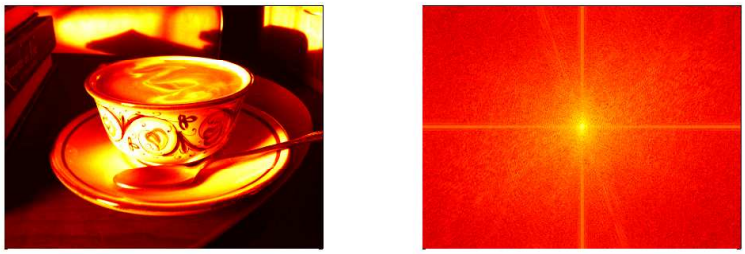


Figure 43: Image of a beautifully made cappuccino along with its Fourier transform (log of the absolute value of the spectrum) in two dimensions. The strong vertical and horizontal lines in the spectrum correspond to vertical and horizontal structures in the photo. Looking carefully at the spectrum will also reveal diagonal signatures associated with the diagonal edges in the photo.

modes. Figure 43 shows a photo and its corresponding 2D Fourier transform. The drawback of the Fourier transform is the fact that the Fourier transform of a sharp edged object results in a sinc-like function that decays in the Fourier modes like $1/k$ where k is the wavenumber. This slow decay means that localization of the image in both the spatial and wavenumber domain is limited. Regardless, the Fourier mode decomposition gives an alternative representation of the image. Moreover, it is clear from the Fourier spectrum that a great number of the Fourier mode components are zero or nearly so. Thus the concept of image compression can easily be seen directly from the spectrum, i.e. by saving 20% of the dominant modes only, the image can be nearly constructed. This then would compress the image five fold. As one might also expect, filtering with the Fourier transform can also help process the image to some desired ends.

Wavelets: As with the time-frequency analysis, wavelets provide a much more sophisticated representation of an image. In particular, wavelets allow for exceptional localization in both the spatial and wavenumber domain. In the wavelet basis, the key quantities are computed from the wavelet coefficients

$$c_\alpha = (u(x,y), \psi_\alpha) \tag{3.1.55}$$

where the wavelet coefficient c_α is equivalent to a Fourier coefficient for a wavelet ψ_α. Since the mid-1980s, the wavelet basis has taken over as the primary tool for image compression due to its excellent space-wavenumber localization properties.

Stochastic modeling: Consideration can be made in image processing of the statistical nature of the data itself. For instance, many images in nature,

such as clouds, trees, sky, sandy beaches, are fundamentally statistical in nature. Thus one may not want to denoise such objects when performing image analysis tasks. In the stochastic modeling approach, hidden features are explored within a given set of observed data which accounts for the statistical nature of many of the underlying objects in an image.

Partial differential equations and diffusion: For denoising applications, or applications in which the data is choppy or highly pixelated, smoothing algorithms are of critical importance. In this case, one can take the original data as the initial conditions of a diffusion process so that the image undergoes, in the simplest case, the evolution

$$\frac{\partial u}{\partial t} = D\nabla^2 u \tag{3.1.56}$$

where $\nabla^2 = \partial_x^2 + \partial_y^2$. This diffusion process provides smoothing to the original data file u. The key is knowing when the smoothing process should be stopped so as not to remove too much image content.

Loading images, additive noise and MATLAB

To illustrate some of the various aspects of image processing, the following example will load the ideal image, apply Gaussian white noise to each pixel and show the noisy image. In MATLAB, most standard image formats are easily loaded. The key to performing mathematical manipulations is to then transform the data into double precision numbers which can be transformed, filtered, and manipulated at will. The generic data file uploaded in MATLAB is *uint8* which is an integer format ranging from 0 to 255. Moreover, color images have three sets of data for each pixel in order to specific the RGB color coordinates. The following code uploads and image file (600x800 pixels). Although there are 600x800 pixels, the data is stored as a 600x800x3 matrix where the extra dimensions contain the color coding. This can be made into a 600x800 matrix by turning the image into a black-and-white picture. This is done in the code that follows:

```
clear all; close all; clc;
A=imread('photo','tiff');   % load image
Abw=rgb2gray(A);         % make image black-and-white
subplot(2,2,1), image(A); set(gca,'Xtick',[],'Ytick',[])
subplot(2,2,3), imshow(Abw);

A2=double(A);          % change form unit8 to double
Abw=double(Abw);
```

Note that at the end of the code, the matrices produced are converted to double precision numbers.

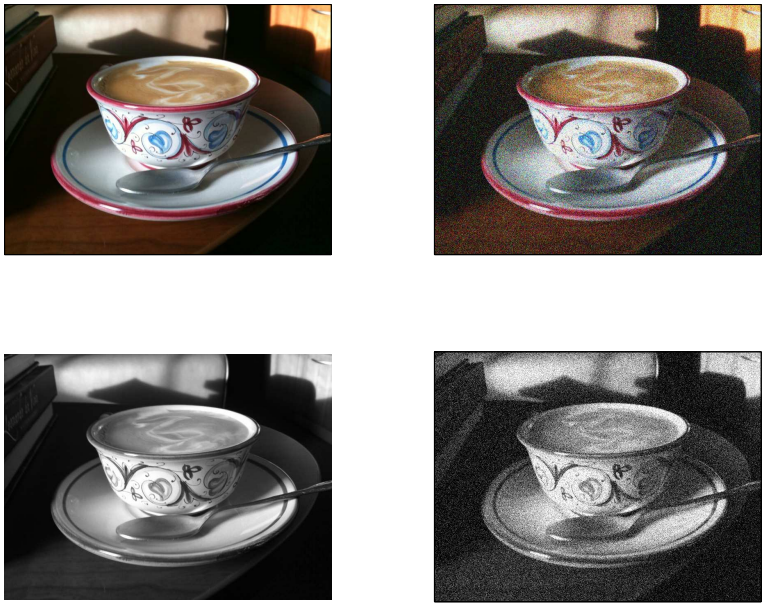


Figure 44: Screenshot of the decomposition of an image into different resolution levels. This application is ideal for image compression.

To add Gaussian white noise to the images, either the color or black-and-white versions, the **randn** command is used. In fact, noise is added to both pictures and the output is produced.

```
noise=randn(600,800,3);  % add noise to RGB image
noise2=randn(600,800);  % add noise to black-and-white

u=uint8(A2+50*noise);   % change from double to uint8
u2=uint8(Abw+50*noise2);

subplot(2,2,2), image(u); set(gca,'Xtick',[],'Ytick',[])
subplot(2,2,4), image(u2); set(gca,'Xtick',[],'Ytick',[])
```

Note that at the final step, the images are converted back to standard image formats used for .JPEG or .TIFF images. Figure 44 demonstrates the plot produced from the above code. The ability to load and manipulate the data is fundamental to all the image processing applications to be pursued in what follows.

As a final example of data manipulation, the Fourier transform of the above image can also be considered. Figure 43 has already demonstrated the result of this code. But for completeness, it is illustrated here

```
Abw2=Abw(600:-1:1,:);
figure(2)
subplot(2,2,1), pcolor(Abw2), shading interp,
colormap(hot), set(gca,'Xtick',[],'Ytick',[])

Abwt=abs(fftshift(fft2(Abw2)));
subplot(2,2,2), pcolor(log(Abwt)), shading interp,
colormap(hot), set(gca,'Xtick',[],'Ytick',[])
```

Note that in this set of plots, we are once again working with matrices so that commands like **pcolor** are appropriate. These matrices are the subject of image processing routines and algorithms.

3.2 Linear filtering for image denoising

As with time-frequency analysis and denoising of time signals, many applications of image processing deal with cleaning up images from imperfections, pixelation, and graininess, i.e. processing of noisy images. The objective in any image processing application is to enhance or improve the quality of a given image. In this section, the filtering of noisy images will be considered with the aim of providing a higher quality, maximally denoised image.

The power of filtering has already been demonstrated in the context of radar detection applications. Ultimately, image denoising is directly related to filtering. To see this, consider Fig. 43 of the last section. In this image, the ideal image is represented along with the log of its Fourier transform. Like many images, the Fourier spectrum is particularly sparse (or nearly zero) for most high-frequency components. Noise, however, tends to generate many high-frequency structures on an image. Thus it is hoped that filtering of high-frequency components might remove unwanted noise fluctuations or graininess in an image. The top two panels of Fig. 45 show an initial noisy image and the log of its Fourier transform. These top two panels can be compared to the ideal image and spectrum shown in Fig. 43. The code used to generate these top two panels is given by the following:

convert the uint8 type which we cannot really work with into doubles.

```
A=imread('photo','tiff'); Abw=rgb2gray(A);
Abw=double(Abw);

B=Abw+100*randn(600,800);
Bt=fft2(B);  Bts=fftshift(Bt);

subplot(2,2,1), imshow(uint8(B)), colormap(gray)
subplot(2,2,2), pcolor((log(abs(Bts)))); shading interp
colormap(gray), set(gca,'Xtick',[],'Ytick',[])
```

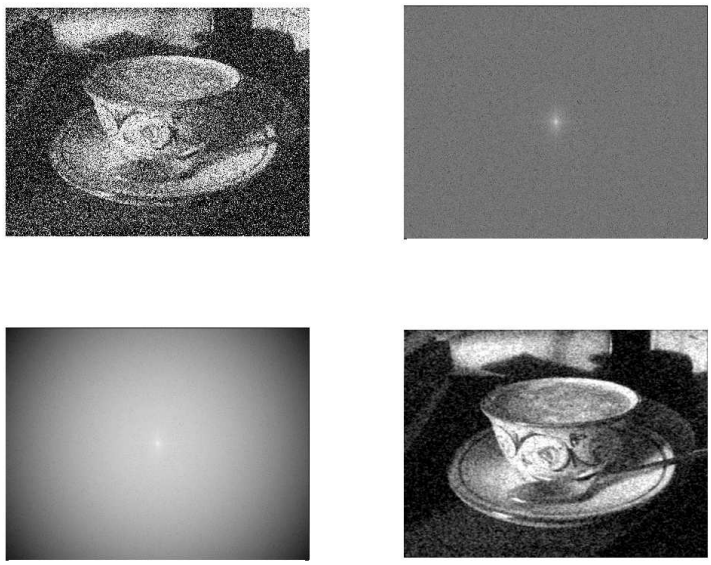


Figure 45: A noisy image and its Fourier transform are considered in the top panel. The ideal image is shown in Fig. 43. By applying a linear Gaussian filter, the high-frequency noise components can be eliminated (bottom left) and the image quality is significantly improved (bottom right).

Note that the uploaded image is converted to a double precision number through the **double** command. It can be transformed back to the image format via the **uint8** command.

Linear filtering can be applied in the Fourier domain of the image in order to remove the high-frequency scale fluctuations induced by the noise. A simple filter to consider is a Gaussian that takes the form

$$F(k_x, k_y) = \exp(-\sigma_x(k_x - a)^2 - \sigma_y(k_y - b)^2) \tag{3.2.57}$$

where σ_x and σ_y are the filter widths in the x and y directions respectively and a and b are the center-frequency values for the corresponding filtering. For the image under consideration, it is a 600x800 pixel image so that the center-frequency components are located at $k_x = 301$ and $k_y = 401$ respectively. To build a Gaussian filter, the following lines of code are needed:

```
kx=1:800; ky=1:600; [Kx,Ky]=meshgrid(kx,ky);
F=exp(-0.0001*(Kx-401).^2-0.0001*(Ky-301).^2);
Btsf=Bts.*F;
```

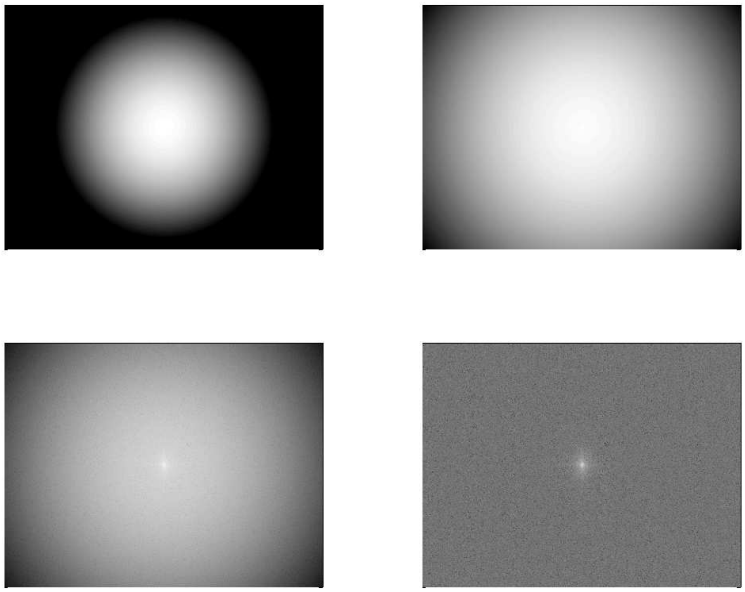


Figure 46: Comparison of the log of the Fourier transform for three different values of the filtering strength, $\sigma_x = \sigma_y = 0.01, 0.001, 0.001$ (top left, top right, bottom left respectively), and the unfiltered image (bottom right).

```
subplot(2,2,3), pcolor(log(abs(Btsf))); shading interp
colormap(gray), set(gca,'Xtick',[],'Ytick',[])

Btf=ifftshift(Btsf); Bf=ifft2(Btf);
subplot(2,2,4), imshow(uint8(Bf)), colormap(gray)
```

In this code, the **meshgrid** command is used to generate the two-dimensional wavenumbers in the x and y direction. This allows for the construction of the filter function that has a two-dimensional form. In particular, a Gaussian centered around $(k_x, k_y) = (301, 401)$ is created with $\sigma_x = \sigma_y = 0.0001$. The bottom two panels in Fig. 45 show the filtered spectrum followed by its inverse Fourier transform. The final image on the bottom right of this figure shows the denoised image produced by simple filtering. The filtering significantly improves the image quality.

One can also over filter and cut out substantial information concerning the figure. Figure 46 shows the log of the spectrum of the noisy image for different values of filters in comparison with the unfiltered image. For narrow filters,

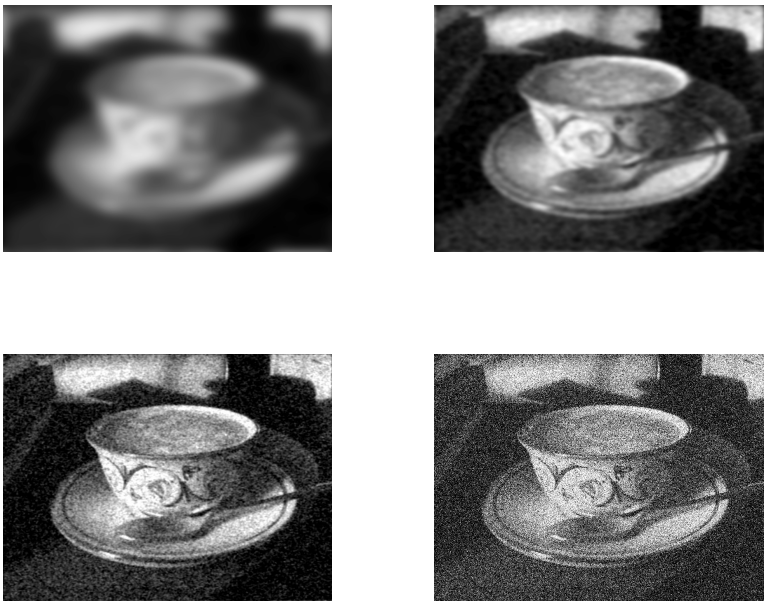


Figure 47: Comparison of the image quality for the three filter strengths considered in Fig. 46. A high degree of filtering blurs the image and no fine scale features are observed (top right). In contrast, moderate strength filtering produces exceptional improvement of the image quality (top right and bottom left). These denoised images should be compared to the unfiltered image (bottom right).

much of the image information is irretrievably lost along with the noise. Finding an optimal filter is part of the image processing agenda. In the image spatial domain, the filtering strength (or width) can be considered. Figure 47 shows a series of filtered images and their comparison to the unfiltered image. Strong filtering produces a smooth, yet blurry image. In this case, all fine scale features are lost. Moderate levels of filtering produce reasonable images with significant reduction of the noise in comparison to the non-filtered image. The following code was used to produce these last two figures.

```
% Gaussian filter
fs=[0.01 0.001 0.0001 0];
for j=1:4
  F=exp(-fs(j)*(Kx-401).^2-fs(j)*(Ky-301).^2);
  Btsf=Bts.*F; Btf=ifftshift(Btsf); Bf=ifft2(Btf);
  figure(4), subplot(2,2,j), pcolor(log(abs(Btsf)))
  shading interp,colormap(gray),set(gca,'Xtick',[],'Ytick',[])
```

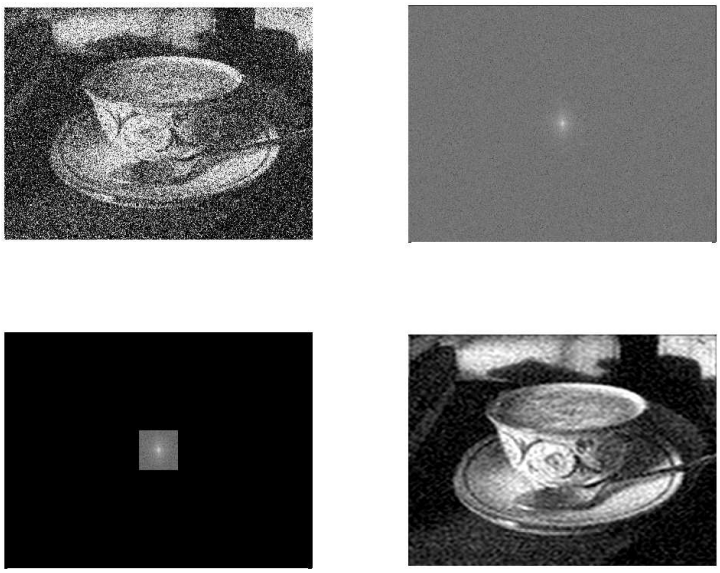


Figure 48: A noisy image and its Fourier transform are considered in the top panel. The ideal image is shown in Fig. 43. By applying a Shannon (step function) filter, the high-frequency noise components can be eliminated (bottom left) and the image quality is significantly improved (bottom right).

```
  figure(5), subplot(2,2,j), imshow(uint8(Bf)), colormap(gray)
end
```

Note that the processed data is converted back to a graphics image before plotting with the **imshow** command.

Gaussian filtering is only one type of filtering that can be applied. There are, just like in signal processing applications, myriads of filters that can be used to process a given image, including low-pass, high-pass, band-pass, etc. As a second example filter, the Shannon filter is considered. The Shannon filter is simply a step function with value of unity within the transmitted band and zero outside of it. In the example that follows, a Shannon filter is applied of width 50 pixels around the center frequency of the image.

```
width=50;
Fs=zeros(600,800);
Fs(301-width:1:301+width,401-width:1:401+width) ...
       =ones(1:2*width+1,1:2*width+1);
Btsf=Bts.*Fs;
```

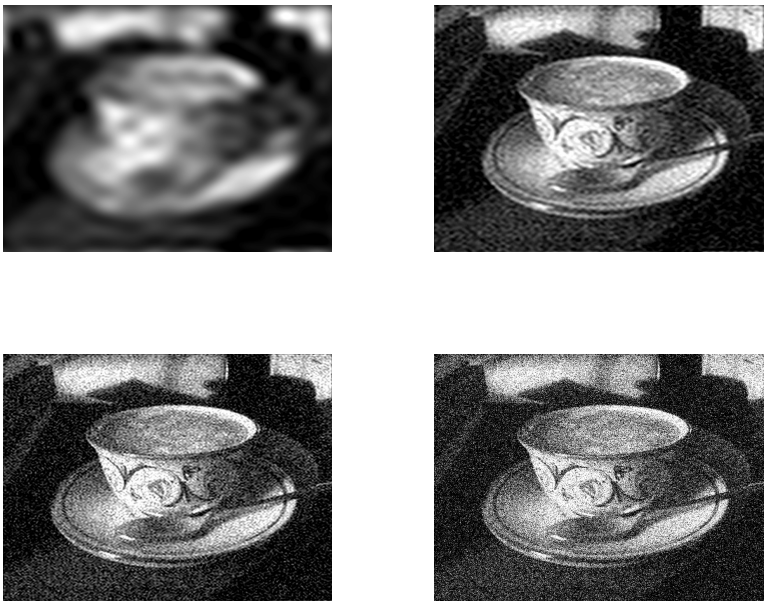


Figure 49: Comparison of the image quality for the three filter widths. A high degree of filtering blurs the image and no fine scale features are observed (top right). In contrast, moderate strength filtering produces exceptional improvement of the image quality (top right and bottom left). These denoised images should be compared to the unfiltered image (bottom right).

```
Btf=ifftshift(Btsf); Bf=ifft2(Btf);

subplot(2,2,3), pcolor(log(abs(Btsf))); shading interp
colormap(gray), set(gca,'Xtick',[],'Ytick',[])
subplot(2,2,4), imshow(uint8(Bf)), colormap(gray)
```

Figure 48 is a companion figure to Fig. 45. The only difference between them is the filter chosen for the image processing. The key difference is represented in the lower left panel of both figures. Note that the Shannon filter simply suppresses all frequencies outside of a given filter box. The performance difference between Gaussian filtering and Shannon filtering appears to be fairly marginal. However, there may be some applications where one or the other is more suitable. The image enhancement can also be explored as a function of the Shannon filter width. Figure 49 shows the image quality as the filter is widened along with the unfiltered image. Strong filtering again produces a blurred image while moderate filtering produces a greatly enhanced image quality. A code to

produce this image is given by the following:

```
fs=[10 50 100 200];
for j=1:4
  Fs=zeros(600,800);
  Fs(301-fs(j):1:301+fs(j),401-fs(j):1:401+fs(j)) ...
     =ones(1:2*fs(j)+1,1:2*fs(j)+1);
  Btsf=Bts.*Fs; Btf=ifftshift(Btsf); Bf=ifft2(Btf);
  figure(7), subplot(2,2,j), imshow(uint8(Bf)), colormap(gray)
end
```

The width is adjusted by considering the number of filter pixels around the center frequency with value unity. One might be able to argue that the Gaussian filter produces slightly better image results since the inverse Fourier transform of a step function filter produces sinc-like behavior.

3.3 Diffusion and image processing

Filtering is not the only way to denoise an image. Intimately related to filtering is the use of diffusion for image enhancement. Consider for the moment the simplest spatial diffusion process in two dimensions, i.e. the heat equation:

$$u_t = D\nabla^2 u \tag{3.3.58}$$

where $u(x, y)$ will represent a given image, $\nabla^2 = \partial_x^2 + \partial_y^2$, D is a diffusion coefficient, and some boundary conditions must be imposed. If for the moment we consider periodic boundary conditions, then the solution to the heat equation can be found from, for instance, the Fourier transform

$$\hat{u}_t = -D(k_x^2 + k_y^2)\hat{u} \quad \rightarrow \quad \hat{u} = \hat{u}_0 e^{-D(k_x^2+k_y^2)t} \tag{3.3.59}$$

where the $\hat{u}$ is the Fourier transform of $u(x, y)$ and $\hat{u}_0$ is the Fourier transform of the initial conditions, i.e. the original noisy image. The solution of the heat equation illustrates a key and critical concept: the wavenumbers (spatial frequencies) decay according to a Gaussian function. Thus linear filtering with a Gaussian is equivalent to a linear diffusion of the image for periodic boundary conditions.

The above argument establishes some equivalency between filtering and diffusion. However, the diffusion formalism provides a more general framework in which to consider image cleanup since the heat equation can be modified to

$$u_t = \nabla \cdot (D(x, y)\nabla u) \tag{3.3.60}$$

where $D(x, y)$ is now a spatial diffusion coefficient. In particular, the diffusion coefficient could be used to great advantage to target trouble spots on an image while leaving relatively noise free patches alone.

what if we do not want to filter our entire image? Applying a general filter will alter the entire image, but it is likely some parts of the image would be best if left be.

To solve the heat equation numerically, we discretize the spatial derivative with a second-order scheme (see Table **??**) so that

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{\Delta x^2}\left[u(x+\Delta x, y) - 2u(x,y) + u(x-\Delta x, y)\right] \quad (3.3.61a)$$

$$\frac{\partial^2 u}{\partial y^2} = \frac{1}{\Delta y^2}\left[u(x, y+\Delta y) - 2u(x,y) + u(x, y-\Delta y)\right]. \quad (3.3.61b)$$

This approximation reduces the partial differential equation to a system of ordinary differential equations. Once this is accomplished, then a variety of standard time-stepping schemes for differential equations can be applied to the resulting system.

The ODE system for 1D diffusion

Consider first diffusion in one-dimension so that $u(x,y) = u(x)$. The vector system for MATLAB can then be formulated. To define the system of ODEs, we discretize and define the values of the vector $\mathbf{u}$ in the following way.

$$\begin{aligned} &u(-L) = u_1 \\ &u(-L+\Delta x) = u_2 \\ &\vdots \\ &u(L-2\Delta x) = u_{n-1} \\ &u(L-\Delta x) = u_n \\ &u(L) = u_{n+1}\,. \end{aligned}$$

If periodic boundary conditions, for instance, are considered, then periodicity requires that $u_1 = u_{n+1}$. Thus the system of differential equations solves for the vector

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}. \quad (3.3.62)$$

The governing heat equation is then reformulated in discretized form as the differential equations system

$$\frac{d\mathbf{u}}{dt} = \frac{\kappa}{\Delta x^2}\mathbf{A}\mathbf{u}\,, \quad (3.3.63)$$

where $\mathbf{A}$ is given by the sparse matrix

$$\mathbf{A} = \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 & 1 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & & \\ \vdots & & & & & \vdots \\ & & & & & 0 \\ \vdots & \cdots & 0 & 1 & -2 & 1 \\ 1 & 0 & \cdots & 0 & 1 & -2 \end{bmatrix}, \tag{3.3.64}$$

and the values of one on the upper right and lower left of the matrix result from the periodic boundary conditions.

MATLAB implementation

The system of differential equations can now be easily solved with a standard time-stepping algorithm such as *ode23* or *ode45*. The basic algorithm would be as follows

1. Build the sparse matrix $\mathbf{A}$.

```
e1=ones(n,1);  % build a vector of ones
A=spdiags([e1 -2*e1 e1],[-1 0 1],n,n);  % diagonals
A(1,n)=1;  A(n,1)=1;  % periodic boundaries
```

2. Generate the desired initial condition vector $\mathbf{u} = \mathbf{u}_0$.

3. Call an ODE solver from the MATLAB suite. The matrix $\mathbf{A}$, the diffusion constant κ and spatial step Δx need to be passed into this routine.

```
[t,y]=ode45('rhs',tspan,u0,[],k,dx,A);
```

   The function *rhs.m* should be of the following form

```
function rhs=rhs(tspan,u,dummy,k,dx,A)
rhs=(k/dx^2)*A*u;
```

4. Plot the results as a function of time and space.

The algorithm is thus fairly routine and requires very little effort in programming since we can make use of the standard time-stepping algorithms already available in MATLAB.

2D MATLAB implementation

In the case of two dimensions, the calculation becomes slightly more difficult since the 2D data is represented by a matrix and the ODE solvers require a vector input for the initial data. For this case, the governing equation is

$$\frac{\partial u}{\partial t} = \kappa \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \tag{3.3.65}$$

Provided $\Delta x = \Delta y = \delta$ are the same, the system can be reduced to the linear system

$$\frac{d\mathbf{u}}{dt} = \frac{\kappa}{\delta^2} \mathbf{A}\mathbf{u}\,, \tag{3.3.66}$$

where we have arranged the vector $\mathbf{u}$ by stacking slices of the data in the second dimension y. This stacking procedure is required for implementation purposes of MATLAB. Thus be defining

$$u_{nm} = u(x_n, y_m) \tag{3.3.67}$$

the collection of image (pixel) points can be arranged as follows

$$\mathbf{u} = \begin{pmatrix} u_{11} \\ u_{12} \\ \vdots \\ u_{1n} \\ u_{21} \\ u_{22} \\ \vdots \\ u_{n(n-1)} \\ u_{nn} \end{pmatrix}, \tag{3.3.68}$$

The matrix $\mathbf{A}$ is a sparse matrix and so the sparse implementation of this matrix can be be used advantageously in MATLAB.

Again, the system of differential equations can now be easily solved with a standard time-stepping algorithm such as *ode23* or *ode45*. The basic algorithm follows the same course as the 1D case, but extra care is taken in arranging the 2D data into a vector.

subsubsection*Diffusion of an image

To provide a basic implementation of the above methods, consider the following MATLAB code which loads a given image file, converts it to double precision numbers, then diffuses the image in order to remove some of the noise content. The process begins once again with the loading of an ideal image on which noise will be projected.

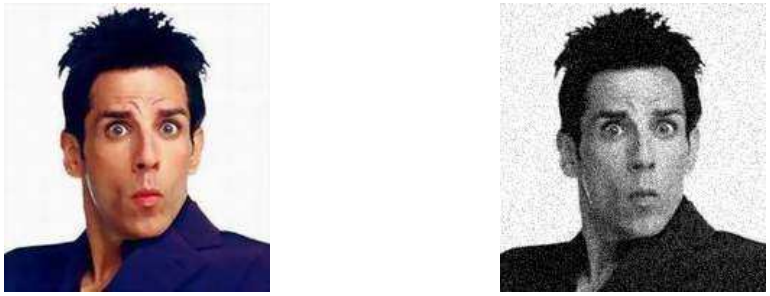


Figure 50: A ridiculously good looking image demonstrating the classic Zoolander *blue steel* look (left panel). Ugly protesters have added noise to the ideal image forcing us to use mathematics to cleanup his image (right panel).

```
clear all; close all; clc;

A=imread('blu','jpeg');
Abw=rgb2gray(A); Abw=double(Abw);
[nx,ny]=size(Abw);
u2=uint8(Abw+20*randn(nx,ny));

subplot(2,2,1), imshow(A)
subplot(2,2,2), imshow(u2)
```

As before, the black-and-white version of the image will be the object consideration for diagnostic purposes. Figure 50 shows the ideal image along with a noisy black-and-white version. Diffusion will be used to denoise this image.

In what follows, a constant diffusion coefficient will be considered. To make the Laplacian operator in two dimensions, the **kron** command is used. The following code first makes the x- and $y-$derivatives in one dimension. The **kron** command is much like the **meshgrid** in that it converts the operators into their two-dimensional versions.

```
x=linspace(0,1,nx); y=linspace(0,1,ny); dx=x(2)-x(1); dy=y(2)-y(1);
onex=ones(nx,1); oney=ones(ny,1);
Dx=(spdiags([onex -2*onex onex],[-1 0 1],nx,nx))/dx^2; Ix=eye(nx);
Dy=(spdiags([oney -2*oney oney],[-1 0 1],ny,ny))/dy^2; Iy=eye(ny);
L=kron(Dx,Iy)+kron(Ix,Dy);
```

The generated operator **L** takes two derivatives in x and y and is the Laplacian in two-dimensions. This operator is constructed to act upon data that has been stacked as in Eq. (3.3.68).

It simply remains to evolve the image through the diffusion equation. But now that the Laplacian operator has been constructed, it simply remains to

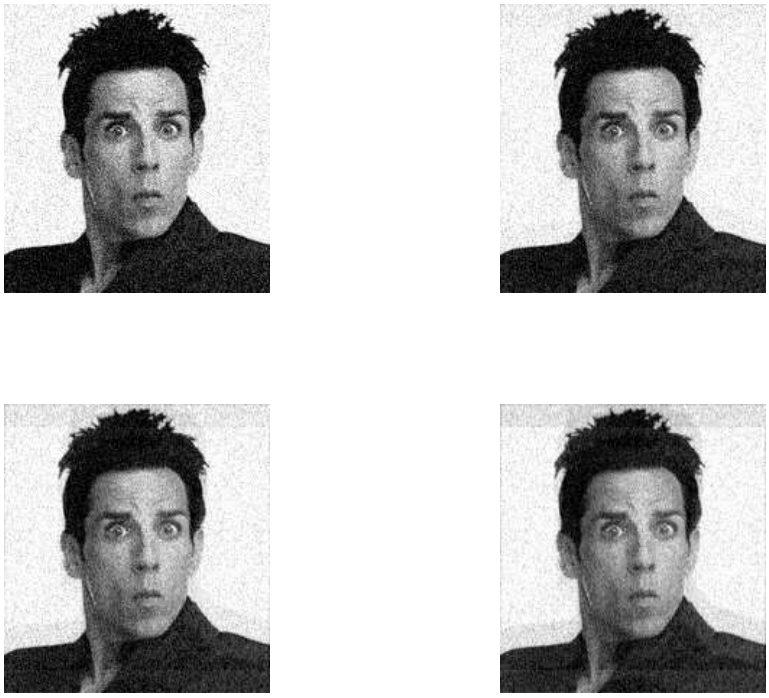


Figure 51: Image denoising process for a diffusion constant of $D = 0.0005$ as a function of time for $t = 0, 0.002, 0.004$ and 0.006. Perhaps the best image is for $t = 0.004$ where a moderate amount of diffusion has been applied. Further diffusion starts to degrade the image quality.

reshape the image, determine a time-span for evolving, and choose a diffusion coefficient. The following code performs the diffusion of the image and produces a plot that tracks the image from its initial state to the final diffused image.

```
tspan=[0 0.002 0.004 0.006]; D=0.0005;

u3=Abw+20*noise2;
u3_2=reshape(u3,nx*ny,1);
[t,usol]=ode113('image_rhs',tspan,u3_2,[],L,D);

for j=1:length(t)
  Abw_clean=uint8(reshape(usol(j,:),nx,ny));
  subplot(2,2,j), imshow(Abw_clean);
end
```

The above code calls on the function **image_rhs** which contains the diffusion

equation information. This function contains the following two lines of code:

```
function rhs=image_rhs(t,u,dummy,L,D)
rhs=D*L*u;
```

Figure 51 shows the evolution of the image through the diffusion process. Note that only a slight amount of diffusion is needed, i.e. a small diffusion constant as well as a short diffusion time, before the pixelation has been substantially reduced. Continued diffusion starts to degrade image quality. This corresponds to over-filtering the image.

Nonlinear filtering and diffusion

As mentioned previously, the linear diffusion process is equivalent to the application of a Gaussian filter. Thus it is not clear that the diffusion process is any better than simple filtering. However, the diffusion process has several distinct advantages: first, particular regions in the spatial figure can be targeted by modification of the diffusion coefficient $D(x, y)$. Second, nonlinear diffusion can be considered for enhancing certain features of the image. This corresponds to a nonlinear filtering process. In this case,

$$u_t = \nabla \cdot (D(u, \nabla u)\nabla u) \tag{3.3.69}$$

where the diffusion coefficient now depends on the image and its gradient. Nonlinear diffusion, with a properly constructed D above, can be used, for instance, as an effective method for extracting edges from an image [6]. Thus although this section has only considered a simple linear diffusion model, the ideas are easily generalized to account for more general concepts and image processing aims.

4 Linear Algebra and Singular Value Decomposition

Linear algebra plays a central role in almost every application area of mathematics in the physical, engineering and biological sciences. It is perhaps the most important theoretical framework to be familiar with as a student of mathematics. Thus it is no surprise that it also plays a key role in data analysis and computation. In what follows, emphasis will placed squarely on the *singular value decomposition* (SVD). This concept is often untouched in undergraduate courses in linear algebra, yet it forms one of the most powerful techniques for analyzing a myriad of applications areas.

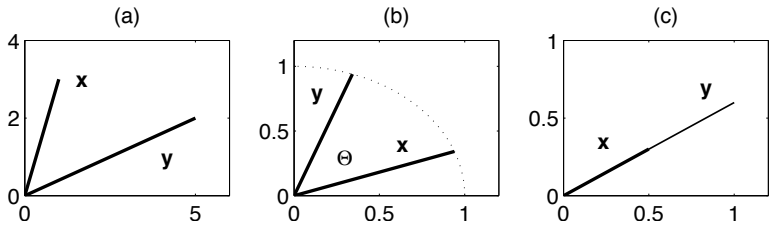


Figure 52: Transformation of a vector $\mathbf{x}$ under the action of multiplication by the matrix $\mathbf{A}$, i.e. $\mathbf{y} = \mathbf{A}\mathbf{x}$. (a) Generic rotation and stretching of the vector as given by Eq. (4.1.70). (b) rotation by 50^0 of a unit vector by the rotation matrix (4.1.71). (c) Stretching of a vector to double its length using (4.1.72) with $\alpha = 2$.

4.1 Basics of The Singular Value Decomposition (SVD)

a matrix multiply can be thought of as a series of rotations/ stretches

In even the earliest experience of linear algebra, the concept of a matrix transforming a vector via multiplication was defined. For instance, the vector $\mathbf{x}$ when multiplied by a matrix $\mathbf{A}$ produces a new vector $\mathbf{y}$ that is now aligned, generically, in a new direction with a new length. To be more specific, the following example illustrates a particular transformation:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 2 & 1 \\ -1 & 1 \end{bmatrix} \quad \rightarrow \quad \mathbf{y} = \mathbf{A}\mathbf{x} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}. \tag{4.1.70}$$

Figure 52(a) shows the vector $\mathbf{x}$ and is transformed version, $\mathbf{y}$, after application of the matrix $\mathbf{A}$. Thus generically, matrix multiplication will rotate and stretch (compress) a given vector as prescribed by the matrix $\mathbf{A}$ (See Fig. 52(a)).

The rotation and stretching of a transformation can be precisely controlled by proper construction of the matrix $\mathbf{A}$. In particular, it is well known that in a two-dimensional space, the rotation matrix

$$\mathbf{A} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{4.1.71}$$

takes a given vector $\mathbf{x}$ and rotates it by an angle θ to produce the vector $\mathbf{y}$. The transformation produced by $\mathbf{A}$ is known as a *unitary transformation* since the matrix inverse $\mathbf{A}^{-1} = \bar{A}^T$ where the bar denotes complex conjugation [7]. Thus rotation can be directly specified without the vector being scaled. To scale the vector in length, the matrix

$$\mathbf{A} = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix} \tag{4.1.72}$$

can be applied to the vector $\mathbf{x}$. This multiplies the length of the vector $\mathbf{x}$ by α. If $\alpha = 2$ (0.5), then the vector is twice (half) its original length. The

the defining coordinate system of S
v1, v2, are stretched by DIFFERENT
ammounts

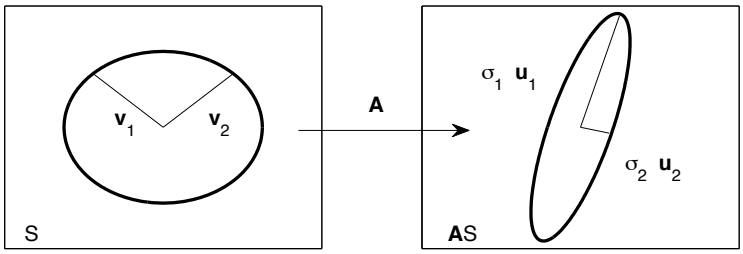


Figure 53: Image of a unit sphere S transformed into a hyperellipse $\mathbf{A}S$ in $\mathbb{R}^2$. The values of σ_1 and σ_2 are the singular values of the matrix $\mathbf{A}$ and represent the lengths of the semiaxes of the ellipse.

combination of the above two matrices gives arbitrary control of rotation and scaling in a two-dimentional vector space. Figure 52 demonstrates some of the various operations associated with the above matrix transformations.

A *singular value decomposition* (SVD) is a factorization of matrix into a number of constitutive components all of which have a specific meaning in applications. The SVD, much as illustrated in the preceding paragraph, is essentially a transformation that stretches/compresses and rotates a given set of vectors. In particular, the following geometric principle will guide our forthcoming discussion: the image of a unit sphere under any $m \times n$ matrix is a hyperellipse. A hyperellipse in $\mathbb{R}^m$ is defined by the surface obtained upon stretching a unit sphere in $\mathbb{R}^m$ by some factors $\sigma_1, \sigma_2, \cdots, \sigma_m$ in the orthogonal directions $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_m \in \mathbb{R}^m$. The stretchings σ_i can possibly be zero. For convenience, consider the $\mathbf{u}_j$ to be unit vectors so that $\|\mathbf{u}_j\|_2 = 1$. The quantity $\sigma_j \mathbf{u}_j$ is then the *principal semiaxes* of the hyperellipse with the length σ_j. Figure 53 demonstrates a particular hyperellipse created under the matrix transformation $\mathbf{A}$ in $\mathbb{R}^2$.

A few things are worth noting at this point. First, if $\mathbf{A}$ has rank r, exactly r of the lengths σ_j will be nonzero. And if the matrix $\mathbf{A}$ is an $m \times n$ matrix where $m > n$, at most n of the σ_j will be nonzero. Consider for the moment a full rank matrix $\mathbf{A}$. Then the n singular values of $\mathbf{A}$ are the lengths of the principal semiaxes $\mathbf{A}S$ as shown in Fig. 53. Convention assumes that the singular values are ordered with the largest first and then in descending order: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$.

On a more formal level, the transformation from the unit sphere to the hyperellipse can be more succinctly stated as follows:

$$\mathbf{A}\mathbf{v}_j = \sigma_j \mathbf{u}_j \quad 1 \leq j \leq n\,. \tag{4.1.73}$$

Thus in total, there are n vectors that are transformed under $\mathbf{A}$. A more com-

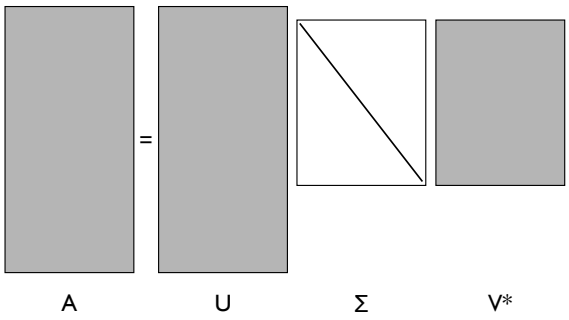


Figure 54: Graphical description of the reduced SVD decomposition.

pact way to write all of these equations simultaneously is with the representation

$$\begin{bmatrix} & \\ & \mathbf{A} \\ & \end{bmatrix} \begin{bmatrix} & & & \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ & & & \end{bmatrix} = \begin{bmatrix} & & & \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ & & & \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix} \tag{4.1.74}$$

so that in compact matrix notation this becomes

$$\mathbf{AV} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\,. \tag{4.1.75}$$

reduced SVD
matlab : svd(X, 'econ')

The matrix $\hat{\mathbf{\Sigma}}$ is an $n \times n$ diagonal matrix with positive entries provided the matrix $\mathbf{A}$ is of full rank. The matrix $\hat{\mathbf{U}}$ is an $m \times n$ matrix with orthonormal columns, and the matrix $\mathbf{V}$ is an $n \times n$ unitary matrix. Since $\mathbf{V}$ is unitary, the above equation can solved for $\mathbf{A}$ by multiplying on the right with $\mathbf{V}^*$ so that

$$\mathbf{A} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\mathbf{V}^*\,. \tag{4.1.76}$$

This factorization is known as the *reduced singular value decomposition*, or reduced SVD, of the matrix $\mathbf{A}$. Graphically, the factorization is represented in Fig. 54.

if you need U to be unitary (i.e. so its inverse can just be U*
for purposes of like PCA)
matlab: svd(X)

The reduced SVD is not the standard definition of the SVD used in the literature. What is typically done to augment the treatment above is to construct a matrix $\mathbf{U}$ from $\hat{\mathbf{U}}$ by adding an additional $m-n$ columns that are orthonormal to the already existing set in $\hat{\mathbf{U}}$. Thus the matrix $\mathbf{U}$ becomes an $m \times m$ unitary matrix. In order to make this procedure work, an additional $m-n$ rows of zeros is also added to the $\hat{\mathbf{\Sigma}}$ matrix. These "silent" columns of $\mathbf{U}$ and rows of $\mathbf{\Sigma}$ are shown graphically in Fig. 55. In performing this procedure, it becomes evident

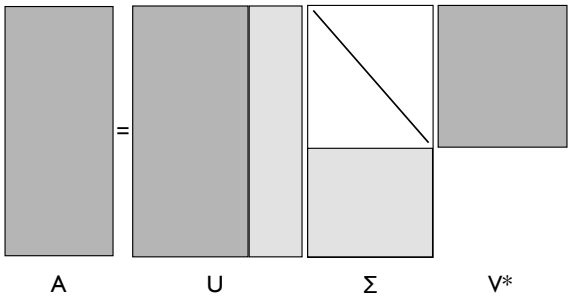


Figure 55: Graphical description of the full SVD decomposition where both $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices. The light shaded regions of $\mathbf{U}$ and $\mathbf{\Sigma}$ are the silent rows and columns that are extended from the reduced SVD.

that rank deficient matrices can easily be handled by the SVD decomposition. In particular, instead of $m-n$ silent rows and matrices, there are now simply $m-r$ silent rows and columns added to the decomposition. Thus the matrix $\mathbf{\Sigma}$ will have r positive diagonal entries, with the remaining $n-r$ being equal to zero.

The full SVD decomposition thus take the form

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \,. \tag{4.1.77}$$

with the following three matrices

$$\mathbf{U} \in \mathbb{C}^{m\times m} \text{ is unitary} \tag{4.1.78a}$$
$$\mathbf{V} \in \mathbb{C}^{n\times n} \text{ is unitary} \tag{4.1.78b}$$
$$\mathbf{\Sigma} \in \mathbb{R}^{m\times n} \text{ is diagonal} \tag{4.1.78c}$$

Additionally, it is assumed that the diagonal entries of Σ are nonnegative and ordered from largest to smallest so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ where $p = \min(m,n)$. The SVD decomposition of the matrix $\mathbf{A}$ thus shows that the matrix first applies a unitary transformation preserving the unit sphere via $\mathbf{V}^*$. This is followed by a stretching operation that creates an ellipse with principal semiaxes given by the matrix $\mathbf{\Sigma}$. Finally, the generated hyperellipse is rotated by the unitary transformation $\mathbf{U}$. Thus the statement: the image of a unit sphere under any $m \times n$ matrix is a hyperellipse, is shown to be true. The following is the primary theorem concerning the SVD:

Theorem: *Every matrix $\mathbf{A} \in \mathbb{C}^{m\times n}$ has a singular value decomposition (4.1.77). Furthermore, the singular values $\{\sigma_j\}$ are uniquely determined, and, if $\mathbf{A}$ is square and the σ_j distinct, the singular vectors $\{\mathbf{u}_j\}$ and $\{\mathbf{v}_j\}$ are uniquely determined up to complex signs (complex scalar factors of absolute value 1).*

Computing the SVD

The above theorem guarantees the existence of the SVD, but in practice, it still remains to be computed. This is a fairly straightforward process if one considers the following matrix products:

V* = V^-1, so we are left with the diagonalization (eigen decomposition) of A^TA!

$$\begin{aligned}\mathbf{A}^T\mathbf{A} &= (\mathbf{U\Sigma V}^*)^T(\mathbf{U\Sigma V}^*)\\ &= \mathbf{V\Sigma U}^*\mathbf{U\Sigma V}^*\\ &= \mathbf{V\Sigma}^2\mathbf{V}^* \end{aligned} \tag{4.1.79}$$

Thus the singular values sigma_j correspond to the squareroots of the eigen values of A^TA

and

$$\begin{aligned}\mathbf{A}\mathbf{A}^T &= (\mathbf{U\Sigma V}^*)(\mathbf{U\Sigma V}^*)^T\\ &= \mathbf{U\Sigma V}^*\mathbf{V\Sigma U}^*\\ &= \mathbf{U\Sigma}^2\mathbf{U}^*\,. \end{aligned} \tag{4.1.80}$$

Multiplying (4.1.79) and (4.1.80) on the right by $\mathbf{V}$ and $\mathbf{U}$ respectively gives the two self-consistent eigenvalue problems

sigma ^2 is just a diagonal matrix.

$$\mathbf{A}^T\mathbf{AV} = \mathbf{V\Sigma}^2 \tag{4.1.81a}$$

$$\mathbf{A}\mathbf{A}^T\mathbf{U} = \mathbf{U\Sigma}^2\,. \tag{4.1.81b}$$

Thus if the normalized eigenvectors are found for these two equations, then the orthonormal basis vectors are produced for $\mathbf{U}$ and $\mathbf{V}$. Likewise, the square root of the eigenvalues of these equations produces the singular values σ_j.

Example: Consider the SVD decomposition of

$$A = \begin{bmatrix} 3 & 0\\ 0 & -2 \end{bmatrix}\,. \tag{4.1.82}$$

The following quantities are computed:

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 3 & 0\\ 0 & -2 \end{bmatrix}\begin{bmatrix} 3 & 0\\ 0 & -2 \end{bmatrix} = \begin{bmatrix} 9 & 0\\ 0 & 4 \end{bmatrix} \tag{4.1.83a}$$

$$\mathbf{A}\mathbf{A}^T = \begin{bmatrix} 3 & 0\\ 0 & -2 \end{bmatrix}\begin{bmatrix} 3 & 0\\ 0 & -2 \end{bmatrix} = \begin{bmatrix} 9 & 0\\ 0 & 4 \end{bmatrix}\,. \tag{4.1.83b}$$

The eigenvalues are clearly $\lambda = \{9, 4\}$, giving singular values $\sigma_1 = 3$ and $\sigma_2 = 2$. The eigenvectors can similarly be constructed and the matrices $\mathbf{U}$ and $\mathbf{V}$ are given by

$$\begin{bmatrix} \pm 1 & 0\\ 0 & \pm 1 \end{bmatrix} \tag{4.1.84}$$

where there is an indeterminate sign in the eigenvectors. However, a self-consistent choice of signs must be made. One possible choice gives

$$\mathbf{A} = \mathbf{U\Sigma V}^* = \begin{bmatrix} 1 & 0\\ 0 & 1 \end{bmatrix}\begin{bmatrix} 3 & 0\\ 0 & 2 \end{bmatrix}\begin{bmatrix} 1 & 0\\ 0 & -1 \end{bmatrix} \tag{4.1.85}$$

The SVD can be easily computed in MATLAB with the following command:

```
[U,S,V]=svd(A);
```

svd(A, 'econ') gives the reduced form

where the $[U, V, D]$ correspond to $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}$ respectively. This decomposition is a critical tool for analyzing many data driven phenomena. But before proceeding to such applications, the connection of the SVD with standard and well-known techniques is elucidated.

4.2 The SVD in broader context

The SVD is an exceptionally important tool in many areas of applications. Part of this is due to its many mathematical properties and its guarantee of existence. In what follows, some of its more important theorems and relations to other standard ideas of linear algebra are considered with the aim of setting the mathematical framework for future applications.

Eigenvalues, eigenvectors and diagonalization

The concept of eigenvalues and eigenvectors is critical to understanding many areas of applications. One of the most important areas where it plays a role is in understanding differential equations. Consider, for instance, the system of differential equations:

$$\frac{d\mathbf{y}}{dt} = \mathbf{A}\mathbf{y} \tag{4.2.86}$$

for some vector $\mathbf{y}(t)$ representing a dynamical system of variables and where the matrix $\mathbf{A}$ determines the interaction among these variables. Assuming a solution of the form $\mathbf{y} = \mathbf{x}\exp(\lambda t)$ results in the eigenvalue problem:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}\,. \tag{4.2.87}$$

The questions remains: how are the eigenvalues and eigenvectors found? To consider this problem, we rewrite the eigenvalue problem as

$$\mathbf{A}\mathbf{x} - \lambda\mathbf{I}\mathbf{x} = (\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}. \tag{4.2.88}$$

Two possibilities now exist.

Option I: The determinant of the matrix $(\mathbf{A}-\lambda\mathbf{I})$ is not zero. If this is true, the matrix is *nonsingular* and its inverse, $(\mathbf{A} - \lambda\mathbf{I})^{-1}$, can be found. The solution to the eigenvalue problem (4.2.87) is then

$$\mathbf{x} = (\mathbf{A} - \lambda\mathbf{I})^{-1}\mathbf{0} \tag{4.2.89}$$

which implies that $\mathbf{x} = \mathbf{0}$. This trivial solution could have easily been guessed. However, it is not relevant as we require nontrivial solutions for $\mathbf{x}$.

Option II: The determinant of the matrix $(\mathbf{A} - \lambda\mathbf{I})$ is zero. If this is true, the matrix is *singular* and its inverse, $(\mathbf{A} - \lambda\mathbf{I})^{-1}$, cannot be found. Although there is no longer a guarantee that there is a solution, it is the only scenario which allows for the possibility of $\mathbf{x} \neq \mathbf{0}$. It is this condition which allows for the construction of eigenvalues and eigenvectors. Indeed, we choose the eigenvalues λ so that this condition holds and the matrix is singular.

Another important operation which can be performed with eigenvalue and eigenvectors is the evaluation of

$$\mathbf{A}^M \tag{4.2.90}$$

where M is a large integer. For large matrices $\mathbf{A}$, this operation is computationally expensive. However, knowing the eigenvalues and eigenvectors of $\mathbf{A}$ allows for a significant ease in computational expense. Assuming we have all the eigenvalues and eigenvectors of $\mathbf{A}$, then

$$\begin{aligned} \mathbf{A}\mathbf{x}_1 &= \lambda_1\mathbf{x}_1 \\ \mathbf{A}\mathbf{x}_2 &= \lambda_2\mathbf{x}_2 \\ &\vdots \\ \mathbf{A}\mathbf{x}_n &= \lambda_1\mathbf{x}_n \,. \end{aligned}$$

This collection of eigenvalues and eigenvectors gives the matrix system

$$\mathbf{AS} = \mathbf{S\Lambda} \tag{4.2.91}$$

where the columns of the matrix $\mathbf{S}$ are the eigenvectors of $\mathbf{A}$,

$$\mathbf{S} = (\mathbf{x}_1 \;\; \mathbf{x}_2 \;\; \cdots \;\; \mathbf{x}_n) \,, \tag{4.2.92}$$

and $\mathbf{\Lambda}$ is a matrix whose diagonals are the corresponding eigenvalues

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & & \cdots & 0 & \lambda_n \end{pmatrix} . \tag{4.2.93}$$

By multiplying (4.2.91) on the right by $\mathbf{S}^{-1}$, the matrix $\mathbf{A}$ can then be rewritten as (note the similarity between this expression and Eq. (4.1.77) for the SVD decomposition)

$$\mathbf{A} = \mathbf{S\Lambda S}^{-1} \,. \tag{4.2.94}$$

The final observation comes from

$$\mathbf{A}^2 = (\mathbf{S\Lambda S}^{-1})(\mathbf{S\Lambda S}^{-1}) = \mathbf{S\Lambda}^2\mathbf{S}^{-1} \,. \tag{4.2.95}$$

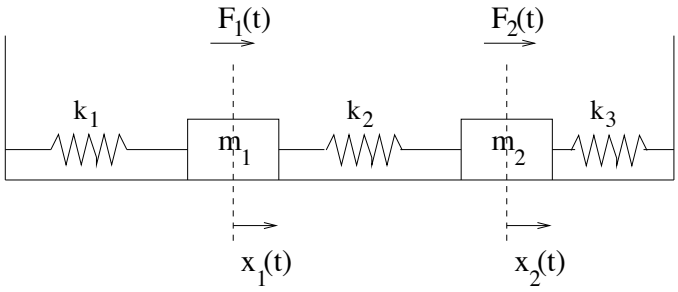


Figure 56: A two mass, three spring system. The fundamental behaviour of the system can be understood by decomposition the system via diagonalization. This reveals that all motion can be expressed as the sum of two fundamental motions: the masses oscillating in-phase, and the masses oscillating exactly out-of-phase.

This then generalizes to

$$\mathbf{A}^M = \mathbf{S}\mathbf{\Lambda}^M\mathbf{S}^{-1} \tag{4.2.96}$$

where the matrix $\mathbf{\Lambda}^M$ is easily calculated as

$$\mathbf{\Lambda}^M = \begin{pmatrix} \lambda_1^M & 0 & \cdots & & 0 \\ 0 & \lambda_2^M & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & & \cdots & 0 & \lambda_n^M \end{pmatrix}. \tag{4.2.97}$$

Since raising the diagonal terms to the M^{th} power is easily accomplished, the matrix $\mathbf{A}$ can then be easily calculated by multiplying the three matrices in (4.2.96)

Diagonalization can also recast a given problem so as to elucidate its more fundamental dynamics. A classic example of the use of diagonalization is a two mass-spring system where the masses m_1 and m_2 are acted on by forces $F_1(t)$ and $F_2(t)$. A schematic of this situation is depicted in Fig. **??**. For each mass, we can write down Newton's Law:

$$\sum F_1 = m_1\frac{d^2x_1}{dt^2} \qquad \text{and} \qquad \sum F_2 = m_2\frac{d^2x_2}{dt^2} \tag{4.2.98}$$

where $\sum F_1$ and $\sum F_2$ are the sum of the forces on m_1 and m_2 respectively. Note that the equations for $x_1(t)$ and $x_2(t)$ are coupled because of the spring with spring constant k_2. The resulting governing equations are then of the form:

$$m_1\frac{d^2x_1}{dt^2} = -k_1x_1 + k_2(x_2 - x_1) + F_1 = -(k_1 + k_2)x_1 + k_2x_2 + F_1 \tag{4.2.99a}$$

$$m_2\frac{d^2x_2}{dt^2} = -k_3x_2 - k_2(x_2 - x_1) + F_2 = -(k_2 + k_3)x_2 + k_2x_1 + F_2. \tag{4.2.99b}$$

This can be reduced further by assuming, for simplicity, $m = m_1 = m_2$, $K = k_1/m = k_2/m$ and $F_1 = F_2 = 0$. This results in the linear system which can be diagonalized via Eq. (4.2.94). The pairs of complex conjugate eigenvalues are produced $\lambda_1^{\pm} = \pm i(2K + \sqrt{2K})^{1/2}$ and $\lambda_2^{\pm} = \pm i(2K - \sqrt{2K})^{1/2}$. Upon diagonalization, the full system can be understood as simply a linear combination of oscillation of the masses that in-phase with each other or out-of-phase with each other.

Diagonalization via SVD

Like the eigenvalue and eigenvector diagonalization technique presented above, the SVD method also makes the following claim: *the SVD makes it possible for every matrix to be diagonal if the proper bases for the domain and range are used.* To consider this statement more formally, consider that since $\mathbf{U}$ and $\mathbf{V}$ are orthonormal bases in $\mathbb{C}^{m\times m}$ and $\mathbb{C}^{n\times n}$ respectively, then any vector in these spaces can be expanded in their bases. Specifically, consider a vector $\mathbf{b} \in \mathbb{C}^{m\times m}$ and $\mathbf{x} \in \mathbb{C}^{n\times n}$, then each can be expanded in the bases of $\mathbf{U}$ and $\mathbf{V}$ so that

Sigma is a diagonal matrix

map our input and output vectors into these two bases, U, V

$$\mathbf{b} = \mathbf{U}\hat{\mathbf{b}}, \quad \mathbf{x} = \mathbf{V}\hat{\mathbf{x}} \tag{4.2.100}$$

where the vectors $\hat{\mathbf{b}}$ and $\hat{\mathbf{x}}$ give the weightings for the orthonormal bases expansion. Now consider the simple equation:

$$\begin{aligned} \mathbf{A}\mathbf{x} = \mathbf{b} \quad \rightarrow \quad & \mathbf{U}^*\mathbf{b} = \mathbf{U}^*\mathbf{A}\mathbf{x} \\ & \mathbf{U}^*\mathbf{b} = \mathbf{U}^*\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*\mathbf{x} \\ & \hat{\mathbf{b}} = \mathbf{\Sigma}\hat{\mathbf{x}}\,. \end{aligned} \tag{4.2.101}$$

U gives the principal components of the columns, V gives the principal components of the rows?

Thus the last line shows $\mathbf{A}$ reduces to the diagonal matrix $\mathbf{\Sigma}$ when the range is expressed in terms of the of the basis vectors of $\mathbf{U}$ and the domain is expressed in terms of the basis vectors of $\mathbf{V}$.

Thus matrices can be diagonalized via either a eigenvalue decomposition or an SVD decomposition. However, there are three key differences in the diagonalization process.

- The SVD performs the diagonalization using two different bases, $\mathbf{U}$ and $\mathbf{V}$, while the eigenvalue method uses a single basis $\mathbf{X}$.
- The SVD method uses an orthonormal basis while the basis vectors in $\mathbf{X}$, while linearly independent, are not generally orthogonal.
- Finally, the SVD is guaranteed to exist for any matrix $\mathbf{A}$ while the same is not true, even for square matrices, for the eigenvalue decomposition.

Useful theorems of SVD

Having established the similarities and connections between eigenvalues and singular values, in what follows a number of important theorems are outlined concerning the SVD. These theorems are important for several reasons. First, the theorems play a key role in the numerical evaluation of many matrix properties via the SVD. Second, the theorems guarantee certain behaviors of the SVD that can be capitalized upon for future applications. Here is a list of important results. A more detailed account is given by Trefethen and Bau [7].

Theorem: *If the rank of* $\mathbf{A}$ *is* r*, then there are* r *nonzero singular values.*

The proof of this is based upon the fact that the rank of a diagonal matrix is equal to the number of its nonzero entries. And since the decomposition $\mathbf{A} = \mathbf{U\Sigma V}^*$ where $\mathbf{U}$ and $\mathbf{V}$ are full rank, then rank($\mathbf{A}$)=rank($\mathbf{\Sigma}$)=r. As a side note, the rank of a matrix can be found with the MATLAB command:

```
rank(A)
```

The standard algorithm used to compute the rank is based upon the SVD and the computation of the nonzero singular values. Thus the theorem is of quite useful in practice.

Theorem: *The* range($\mathbf{A}$)=$\langle \mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_r \rangle$ *and* null ($\mathbf{A}$)=$\langle \mathbf{v}_r, \mathbf{v}_{r+1}, \cdots, \mathbf{v}_n \rangle$.

Note that the range and null space come from the two expansion bases $\mathbf{U}$ and $\mathbf{V}$. The range and null space can be found in MATLAB with the commands:

```
range(A)
null(A)
```

This theorem also serves as the most accurate computation basis for determining the range and null space of a given matrix $\mathbf{A}$ via the SVD.

Theorem: *The norm* $\|\mathbf{A}\|_2 = \sigma_1$ *and* $\|\mathbf{A}\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2}$.

These norms are known as the 2-norm and the Frobenius-norm respectively. They essentially measure the *energy* of a matrix. Although it is difficult to conceptualize abstractly what this means, it will become much more clear in the context of given applications. This result is established by the fact that $\mathbf{U}$ and $\mathbf{V}$ are unitary operators so that their norm is unity. Thus with $\mathbf{A} = \mathbf{U\Sigma V}$, the norm $\|\mathbf{A}\|_2 = \|\mathbf{\Sigma}\|_2 = \max\{|\sigma_j|\} = \sigma_1$. A similar reasoning holds for the Frobenius norm definition. The norm can be calculated in MATLAB with

```
norm(A)
```

Notice that the Frobenius norm contains the total matrix energy while the 2-norm definition contains the energy of the largest singular value. The ratio $\|\mathbf{A}\|_2/\|\mathbf{A}\|_F$ effectively measures the portion of the energy in the semiaxis $\mathbf{u}_1$. This fact will be tremendously important for us. Furthermore, this theorem gives the standard way for computing matrix norms.

Theorem: *The nonzero singular values of* $\mathbf{A}$ *are the square roots of the nonzero eigenvalues of* $\mathbf{A}^*\mathbf{A}$ *or* $\mathbf{A}\mathbf{A}^*$. *(These matrices have the same nonzero eigenvalues).*

This has already been shown in the calculation for actually determining the SVD. In particular, this process is illustrated in Eqs. (4.1.80) and (4.1.79). This theorem is important for actually producing the unitary matrices $\mathbf{U}$ and $\mathbf{V}$.

Theorem: *If* $\mathbf{A} = \mathbf{A}^*$ *(self-adjoint), then the singular values of* $\mathbf{A}$ *are the absolute values of the eigenvalues of* $\mathbf{A}$.

As with most self-adjoint problems, there are very nice properties to the matrices, such as the above theorem. Eigenvalues can be computed with the command:

```
eig(A)
```

Alternatively, one can use the **eigs** to specify the number of eigenvalues desired and their specific ordering.

Theorem: *For* $\mathbf{A} \in \mathbb{C}^{m\times m}$, *the determinant is given by* $|\det(A)| = \prod_{j=1}^{m} \sigma_j$.

Again, due to the fact that the matrices $\mathbf{U}$ and $\mathbf{V}$ are unitary, their determinants are unity. Thus $|\det(A)| = |\det(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*)| = |\det(\mathbf{U})||\det(\mathbf{\Sigma})||\det(\mathbf{V}^*)| = |\det(\mathbf{\Sigma})| = \prod_{j=1}^{m} \sigma_j$. Thus even determinants can be computed via the SVD and its singular values.

Low dimensional reductions

Now comes the last, and most formative, property associated with the SVD: *low-dimensional approximations* to high-degree of freedom or complex systems. In linear algebra terms, this is also *low-rank approximations*. The interpretation of the theorems associated with these low-dimensional reductions are critical for the use and implementation of the SVD. Thus we consider the following:

Theorem: $\mathbf{A}$ *is the sum of* r *rank-one matrices*

$$\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^* \,. \tag{4.2.102}$$

Since sigma_j are decreasing, we can observe their trend and select only the greatest sigma values, ignoring some below a threshold. This allows us to approximate A

sigma_j weighs the importance of column j in U and V* in representing A, i.e. our data. As V* acts on the

SVD is the best low rank approximation available in the L2 sense.

There are a variety of ways to express the $m \times n$ matrix $\mathbf{A}$ as a sum of rank one matrices. The bottom line is this: *the Nth partial sum captures as much of the matrix $\mathbf{A}$ as possible.* Thus the partial sum of the rank one matrices is an important object to consider. This leads to the following theorem:

Theorem: *For any N so that $0 \leq N \leq r$, we can define the partial sum*

low rank approximation!

$$\mathbf{A}_N = \sum_{j=1}^{N} \sigma_j \mathbf{u}_j \mathbf{v}_j^* \,. \tag{4.2.103}$$

And if $N = \min\{m, n\}$, define $\sigma_{N+1} = 0$. Then

$$\|A - A_N\|_2 = \sigma_{N+1} \,. \tag{4.2.104}$$

Likewise, if using the Frobenius norm, then

$$\|A - A_N\|_N = \sqrt{\sigma_{N+1}^2 + \sigma_{N+2}^2 + \cdots + \sigma_r^2} \,. \tag{4.2.105}$$

Interpreting this theorem is critical. Geometrically, we can ask *what is the best approximation of a hyperellipsoid by a line segment? Simply take the line segment to be the longest axis, i.e. that associated with the singular value σ_1.* Continuing this idea, what is the best approximation by a two-dimensional ellipse? Take the longest and second longest axes, i.e. those associated with the singular values σ_1 and σ_2. After r steps, the total energy in $\mathbf{A}$ is completely captured. **Thus the SVD gives a type of least-square fitting algorithm, allowing us to project the matrix onto low-dimensional representations in a formal, algorithmic way.** Herein lies the ultimate power of the method.

4.3 Introduction to Principle Component Analysis (PCA)

To make explicit the concept of the SVD, a simple model example will be formulated that will illustrate all the key concepts associated with the SVD. The model to be considered will be a simple spring-mass system as illustrated in Fig. 57. Of course, this is a fairly easy problem to solve from basic concepts of $\mathbf{F} = m\mathbf{a}$. But for the moment, let's suppose we didn't know the governing equations. In fact, our aim in this lecture is to use a number of cameras (probes) to extract out data concerning the behavior of the system and then to try extract empirically the governing equations of motion.

given noisy or complex dynamics, what components are the most important?

SVD can give a pretty good answer

This prologue highlights one of the key applications of the SVD, or alternatively a *Principle Component Analysis* (PCA), *Proper Mode Decomposition* (POD), or *Karhunen-Loéve Decomposition* as it also known in the literature. Namely, from seemingly complex, perhaps random, data, can low-dimensional reductions of the dynamics and behavior be produced when the governing equations are not known. Such methods can be used to quantify low-dimensional

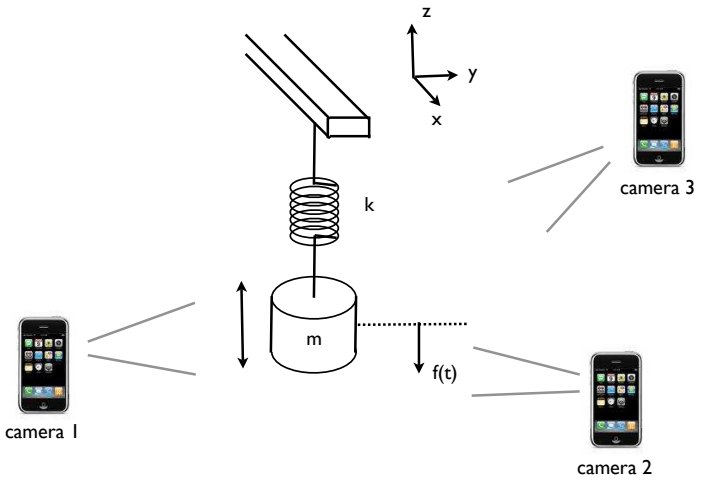


Figure 57: A prototypical example of how we might apply a principal component analysis, or SVD, is the simple mass-spring system exhibited here. The mass m is suspended with a spring with Hook's constant k. Three video cameras collect data about its motion in order to ascertain its governing equations.

SVD gives us the most relevant coordinate system for our data by highlighting the most important components ("directions")

dynamics arising in such areas as turbulent fluid flows [9], structural vibrations [10, 11], insect locomotion [12], damage detection [13], and neural decision making strategies [14] to name just a few areas of application. It will also become obvious as we move forward on this topic that the breadth of applications is staggering and includes image processing and signal analysis. Thus the perspective to be taken here is clearly one in which the data analysis of an unknown, but potentially low-dimensional system is to be analyzed.

Again we turn our attention to the simple experiment at hand: a mass suspended by a spring as depicted in Fig. 57. If the mass is perturbed or taken from equilibrium in the z-direction only, we know that the governing equations are simply

$$\frac{d^2 f(t)}{dt^2} = -\omega^2 f(t) \tag{4.3.106}$$

where the function $f(t)$ measures the displacement of the mass in the z-direction as a function of time. This has the well known solution (in amplitude-phase form)

$$f(t) = A\cos(\omega t + \omega_0) \tag{4.3.107}$$

where the values of A and ω_0 are determined from the initial state of the system. This essentially states that the state of the system can be described by a one-

degree of freedom system.

In the above analysis, there are many things we have ignored. Included in the list of things we have ignored is the possibility that the initial excitation of the system actually produces movement in the $x-y$ plane. Further, there is potentially noise in the data from, for instance, shaking of the cameras during the video capture. Moreover, from what we know of the solution, only a single camera is necessary to capture the underlying motion. In particular, a single camera in the $x-y$ plane at $z=0$ would be ideal. Thus we have over-sampled the data with three cameras and have produced redundant data sets. From all of these potential perturbations and problems, it is our goal to extract out the simple solution given by the simple harmonic motion.

This problem is a good example of what kind of processing is required to analyze a realistic data set. Indeed, one can imagine that most data will be quite noisy, perhaps redundant, and certainly not produced from an optimal viewpoint. But through the process of PCA, these can be circumvented in order to extract out the ideal or simplified behavior. Moreover, we may even learn how to transform the data into the optimal viewpoint for analyzing the data.

Data collection and ordering

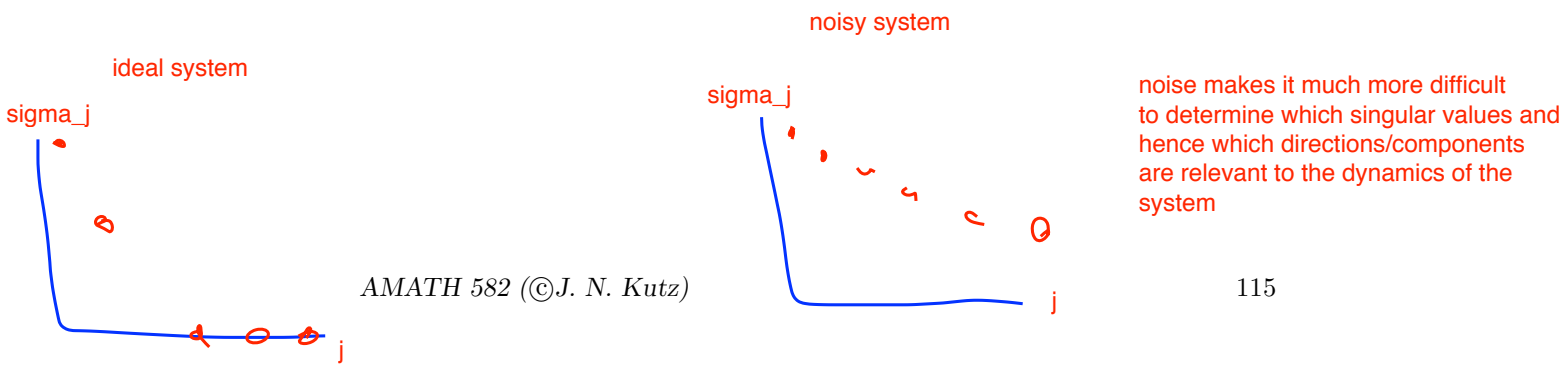
Assume now that we have started the mass in motion by applying a small perturbation in the z-direction only. Thus the mass will begin to oscillate. Three cameras are then used to record the motion. Each camera produces a two-dimensional representation of the data. If we denote the data from the three cameras with subscripts a, b and c, then the data collected are represented by the following:

$$\text{camera 1: } (\mathbf{x}_a, \mathbf{y}_a) \tag{4.3.108a}$$

$$\text{camera 2: } (\mathbf{x}_b, \mathbf{y}_b) \tag{4.3.108b}$$

$$\text{camera 3: } (\mathbf{x}_c, \mathbf{y}_c) \tag{4.3.108c}$$

where each set $(\mathbf{x}_j, \mathbf{y}_j)$ is data collected over time of the position in the $x-y$ plane of the camera. Note that this is not the same $x-y$ plane of the oscillating mass system as shown in Fig. 57. Indeed, we should pretend we don't know the correct $x-y-z$ coordinates of the system. Thus the camera positions and their relative $x-y$ planes are arbitrary. The length of each vector $\mathbf{x}_i$ and $\mathbf{y}_i$ depends on the data collection rate and the length of time the dynamics is observed. We denote the length of these vectors as n.



All the data collected can then be gathered into a single matrix:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_a \\ \mathbf{y}_a \\ \mathbf{x}_b \\ \mathbf{y}_b \\ \mathbf{x}_c \\ \mathbf{y}_c \end{bmatrix} \tag{4.3.109}$$

Thus the matrix $X \in \mathbb{R}^{m\times n}$ where m represents the number of measurement types and n is the number of data points taken from the camera over time.

Now that the data has been arranged, two issues must be addressed: noise and redundancy. Everyone has an intuitive concept that noise in your data can only deteriorate, or corrupt, your ability to extract the true dynamics. Just as in image processing, noise can alter an image beyond restoration. Thus there is also some idea that if the measured data is too noisy, fidelity of the underlying dynamics is compromised from a data analysis point of view. The key measure of this is the so-called signal-to-noise ratio: SNR$= \sigma^2_{\text{signal}}/\sigma^2_{\text{noise}}$, where the ratio is given as the ratio of variances of the signal and noise fields. A high SNR (much greater than unity) gives almost noiseless (high precision) data whereas a low SNR suggests the underlying signal is corrupted by the noise. The second issue to consider is redundancy. In the example of Fig. 57, the single degree of freedom is sampled by three cameras, each of which is really recording the same single degree of freedom. Thus the measurements should be rife with redundancy, suggesting that the different measurements are statistically dependent. Removing this redundancy is critical for data analysis.

The covariance matrix

An easy way to identify redundant data is by considering the covariance between data sets. Recall from our early lectures on probability and statistics that the covariance measures the statistical dependence/independence between two variables. Obviously, strongly statistically dependent variables can be considered as redundant observations of the system. Specifically, consider two sets of measurements with zero means expressed in row vector form:

$$\mathbf{a} = [a_1\ a_2\ \cdots\ a_n] \text{ and } \mathbf{b} = [b_1\ b_2\ \cdots\ b_n] \tag{4.3.110}$$

where the subscript denotes the sample number. The variances of $\mathbf{a}$ and $\mathbf{b}$ are given by

$$\sigma^2_{\mathbf{a}} = \frac{1}{n-1}\mathbf{a}\mathbf{a}^T \tag{4.3.111a}$$

$$\sigma^2_{\mathbf{b}} = \frac{1}{n-1}\mathbf{b}\mathbf{b}^T \tag{4.3.111b}$$

while the covariance between these two data sets is given by

$$\sigma^2_{\mathbf{ab}} = \frac{1}{n-1}\mathbf{a}\mathbf{b}^T \tag{4.3.112}$$

where the normalization constant of $1/(n-1)$ is for an unbiased estimator.

We don't just have two vectors, but potentially quite a number of experiments and data that would need to be correlated and checked for redundancy. In fact, the matrix in Eq. (4.3.109) is exactly what needs to be checked for covariance. The appropriate *covariance matrix* for this case is then

$$\mathbf{C_X} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T . \tag{4.3.113}$$

This is easily computed with MATLAB from the command line:

```
cov(X)
```

The covariance matrix $\mathbf{C_X}$ is a square, symmetric $m \times m$ matrix whose diagonal represents the variance of particular measurements. The off-diagonal terms are the covariances between measurement types. Thus $\mathbf{C_X}$ captures the correlations between all possible pairs of measurements. Redundancy is thus easily captured since if two data sets are identical (identically redundant), the off-diagonal term and diagonal term would be equal since $\sigma^2_{\mathbf{ab}} = \sigma^2_{\mathbf{a}} = \sigma^2_{\mathbf{b}}$ if $\mathbf{a} = \mathbf{b}$. Thus large diagonal terms correspond to redundancy while small diagonal terms suggest that the two measured quantities are close to statistically independent and have low redundancy. It should also be noted that large diagonal terms, or those with large variances, typically represent what we might consider *the dynamics of interest* since the large variance suggests strong fluctuations in that variable. Thus the covariance matrix is the key component to understanding the entire data analysis.

does he mean off diagonal terms? →

Achieving the goal: The insight given by the covariance matrix leads to our ultimate aim of

i) removing redundancy
ii) identifying those signals with maximal variance.

Thus in a mathematical sense, we are simply asking to represent $\mathbf{C_X}$ so that the diagonals are ordered from largest to smallest and the off-diagonals are zero, i.e. our task is to diagonalize the covariance matrix. This is *exactly* what the SVD does, thus allowing it to becomes the tool of choice for data analysis and dimensional reduction. In fact, the SVD diagonalizes and each singular directions captures as much energy as possible as measured by the singular values σ_j.

4.4 Principal Components, Diagonalization and SVD

The example presented in the previous section shows that the key to analyzing a given experiment is to consider the covariance matrix

$$\mathbf{C_X} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T. \tag{4.4.114}$$

where the matrix $\mathbf{X}$ contains the experimental data of the system. In particular, $\mathbf{X} \in \mathbb{C}^{m\times n}$ where m are the number of probes or measuring positions, and n is the number of experimental data points taken at each location.

In this setup, the following facts are highlighted:

- $\mathbf{C_X}$ is a square, symmetric $m \times m$ matrix.
- The diagonal terms of $\mathbf{C_X}$ are the measure variance for particular measurements. By assumption, large variances correspond to *dynamics of interest*, whereas low variances are assumed to correspond to *non-interesting dynamics*.
- The off-diagonal terms of $\mathbf{C_X}$ are the covariance between measurements. Indeed, the off-diagonals captures the correlations between all possible pairs of measurements. A large off-diagonal term represents two events that have a high-degree of redundancy, whereas a small off-diagonal coefficient means there is little redundancy in the data, i.e. they are statistically independent.

Diagonalization

The concept of diagonalization is critical for understanding the underpinnings of many physical systems. In this process of diagonalization, the correct coordinates, or basis functions, are revealed that reduce the given system to its low-dimensional essence. There is more than one way to diagonalize a matrix, and this is certainly true here as well since the constructed covariance matrix $\mathbf{C_X}$ is square and symmetric, both properties that are especially beneficial for standard eigenvalue/eigenvector expansion techniques.

The key idea behind the diagonalization is simply this: there exists an *ideal* basis in which the $\mathbf{C_X}$ can be written (diagonalized) so that in this basis, all redundancies have been removed, and the largest variances of particular measurements are ordered. In the language being developed here, this means that the system has been written in terms of its *principle components*, or in a *proper orthogonal decomposition*.

Eigenvectors and eigenvalues: The most straightforward way to diagonalize the covariance matrix is by making the observation that $\mathbf{X}\mathbf{X}^T$ is a square, symmetric $m \times m$ matrix, i.e. it is self-adjoint so that the m eigenvalues are real

and distinct. Linear algebra provides theorems which state the such a matrix can be rewritten as

$$\mathbf{X}\mathbf{X}^T = \mathbf{S}\mathbf{\Lambda}\mathbf{S}^{-1} \tag{4.4.115}$$

as stated in Eq. (4.2.94) where the matrix $\mathbf{S}$ is a matrix of the eigenvectors of $\mathbf{X}\mathbf{X}^T$ arranged in columns. Since it is a symmetric matrix, these eigenvector columns are orthogonal so that ultimately the $\mathbf{S}$ can be written as a unitary matrix with $\mathbf{S}^{-1} = \mathbf{S}^T$. Recall that the matrix $\mathbf{\Lambda}$ is a diagonal matrix whose entries correspond to the m distinct eigenvalue of $\mathbf{X}\mathbf{X}^T$.

This suggests that instead of working directly with the matrix $\mathbf{X}$, we consider working with the transformed variable, or in the principal component basis,

$$\mathbf{Y} = \mathbf{S}^T\mathbf{X}\,. \tag{4.4.116}$$

For this new basis, we can then consider its covariance

$$\begin{aligned}
\mathbf{C_Y} &= \frac{1}{n-1}\mathbf{Y}\mathbf{Y}^T \\
&= \frac{1}{n-1}(\mathbf{S}^T\mathbf{X})(\mathbf{S}^T\mathbf{X})^T \\
&= \frac{1}{n-1}\mathbf{S}^T(\mathbf{X}\mathbf{X}^T)\mathbf{S} \\
&= \frac{1}{n-1}\mathbf{S}^T\mathbf{S}\mathbf{\Lambda}\mathbf{S}\mathbf{S}^T \\
\mathbf{C_Y} &= \frac{1}{n-1}\mathbf{\Lambda}\,.
\end{aligned} \tag{4.4.117}$$

In this basis, the *principal components* are the eigenvectors of $\mathbf{X}\mathbf{X}^T$ with the interpretation that the jth diagonal value of $\mathbf{C_Y}$ is the variance of $\mathbf{X}$ along $\mathbf{x}_j$, the jth column of $\mathbf{S}$. The following codes of line produce the principal components of interest.

```
[m,n]=size(X);   %  compute data size
mn=mean(X,2); %  compute mean for each row
X=X-repmat(mn,1,n);  % subtract mean

Cx=(1/(n-1))*X*X';   % covariance
[V,D]=eig(Cx);      % eigenvectors(V)/eigenvalues(D)
lambda=diag(D);    % get eigenvalues

[dummy,m_arrange]=sort(-1*lambda);  % sort in decreasing order
lambda=lambda(m_arrange);
V=V(:,m_arrange);

Y=V'*X;  % produce the principal components projection
```

This simple code thus produces the eigenvalue decomposition and the projection of the original data onto the principal component basis.

Singular value decomposition: A second method for diagonalizing the covariance matrix is the SVD method. In this case, the SVD can diagonalize any matrix by working in the appropriate pair of bases $\mathbf{U}$ and $\mathbf{V}$ as outlined in the first lecture of this section. Thus by defining the transformed variable

$$\mathbf{Y} = \mathbf{U}^*\mathbf{X} \tag{4.4.118}$$

where $\mathbf{U}$ is the unitary transformation associated with the SVD: $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$. Just as in the eigenvalue/eigenvector formulation, we then compute the variance in $\mathbf{Y}$:

since the off diagonal entries of this covariance matrix are zero, we have generated completely independent vectors with Y = U*X. Thus, by finding the nonzero (or significant) diagonal elements of C_Y, we can find which directions are fundamental to the dynamics of our system.

$$\begin{aligned}
\mathbf{C_Y} &= \frac{1}{n-1}\mathbf{Y}\mathbf{Y}^T \\
&= \frac{1}{n-1}(\mathbf{U}^*\mathbf{X})(\mathbf{U}^*\mathbf{X})^T \\
&= \frac{1}{n-1}\mathbf{U}^*(\mathbf{X}\mathbf{X}^T)\mathbf{U} \\
&= \frac{1}{n-1}\mathbf{U}^*\mathbf{U}\mathbf{\Sigma}^2\mathbf{U}\mathbf{U}^* \\
\mathbf{C_Y} &= \frac{1}{n-1}\mathbf{\Sigma}^2 .
\end{aligned} \tag{4.4.119}$$

This makes explicit the connection between the SVD and the eigenvalue method, namely that $\mathbf{\Sigma}^2 = \mathbf{\Lambda}$. The following codes of line produce the principal components of interest using the SVD (Assume that the you have the first three lines from the previous MATLAB code).

```
[u,s,v]=svd(X'/sqrt(n-1));  % perform the SVD
lambda=diag(s).^2;  % produce diagonal variances
Y=u'*X;  % produce the principal components projection
```

This gives the SVD method for producing the principal components. Overall, the SVD method is the more robust method and should be used. However, the connection between the two methods becomes apparent in these calculations.

Spring Experiment

To illustrate this completely in practice, three experiments will be performed in class with the configuration of Fig. 57. The following experiments will attempt to illustrate various aspects of the PCA and its practical usefulness and the effects of noise on the PCA algorithms.

- **Ideal case:** Consider a small displacement of the mass in the z direction and the ensuing oscillations. In this case, the entire motion is in the z directions with simple harmonic motion being observed.
- **noisy case:** Repeat the ideal case experiment, but this time, introduce camera shake into the video recording. This should make it more difficult to extract the simple harmonic motion. But if the shake isn't too bad, the dynamics will still be extracted with the PCA algorithms.
- **horizontal displacement:** In this case, the mass is released off-center so as to produce motion in the $x-y$ plane as well as the z direction. Thus there is both a pendulum motion and a simple harmonic oscillations. See what the PCA tells us about the system.

In order to extract out the mass movement from the video frames, the following MATLAB code is needed. This code is a generic way to read in movie files to MATLAB for post-processing.

```
obj=mmreader('matlab_test.mov')

vidFrames = read(obj);
numFrames = get(obj,'numberOfFrames');

for k = 1 : numFrames
    mov(k).cdata = vidFrames(:,:,:,k);
    mov(k).colormap = [];
end

for j=1:numFrames
  X=frame2im(mov(j));
  imshow(X); drawnow
end
```

This command multi-media reader command **mmreader** simply characterizes the file type and its attributes. The bulk of time in this is the **read** command which actually uploads the movie frames into the MATLAB desktop for processing. Once the frames are extracted, they can again be converted to double precision numbers for mathematical processing. In this case, the position of the mass is to be determined from each frame. This basic shell of code is enough to begin the process of extracting the spring-mass system information.

4.5 Principal Components and Proper Orthogonal Models

Now that the basic framework of the principal component analysis and its relation to the SVD has been laid down, a few remaining issues need to be addressed. In particular, the principal component analysis seems to suggest that

we are simply expanding our solution in another *orthonormal* basis, one which can always diagonalize the underlying system.

Mathematically, we can consider a given function $f(x,t)$ over a domain of interest. In most applications, the variables x and t will refer to the standard space-time variables we are familiar with. The idea is to then expand this function in some basis representation so that

$$f(x,t) \approx \sum_{j=1}^{N} a_j(t)\phi_j(x) \tag{4.5.120}$$

where N is the total number of modes, $\phi_j(x)$, to be used. The remaining function $a_j(t)$ determines the weights of the spatial modes.

The expansion (4.5.120) is certainly not a new idea to us. Indeed, we have been using this concept extensively already with Fourier transforms, for instance. Specifically, here are some of the more common expansion bases used in practice:

$$\phi_j(x) = (x - x_0)^j \quad \text{Taylor expansion} \tag{4.5.121a}$$
$$\phi_j(x) = \cos(jx) \quad \text{Discrete cosine transform} \tag{4.5.121b}$$
$$\phi_j(x) = \sin(jx) \quad \text{Discrete sine transform} \tag{4.5.121c}$$
$$\phi_j(x) = \exp(jx) \quad \text{Fourier transform} \tag{4.5.121d}$$
$$\phi_j(x) = \psi_{a,b}(x) \quad \text{Wavelet transform} \tag{4.5.121e}$$
$$\phi_j(x) = \phi_{\lambda_j}(x) \quad \text{Eigenfunction expansion} \tag{4.5.121f}$$

where $\phi_{\lambda_j}(x)$ are eigenfunctions associated with the underlying system. This places the concept of a basis function expansion in familiar territory. Further, it shows that such a basis function expansion is not unique, but rather can potentially have an infinite number of different possibilities.

As for the weighting coefficients $a_j(t)$, they are simply determined from the standard inner product rules and the fact that the basis functions are orthonormal (Note that they are not written this way above):

$$\int \phi_j(x)\phi_n(x)dx = \begin{cases} 1 & j = n \\ 0 & j \neq n \end{cases} . \tag{4.5.122}$$

This then gives for the coefficients

$$a_j(t) = \int f(x,t)\phi_j(x)dx \tag{4.5.123}$$

and the basis expansion is completed.

Interestingly enough, the basis functions selected are chosen often for simplicity and/or their intuitive meaning for the system. For instance, the Fourier transform very clearly highlights the fact that the given function has a representation in terms of *frequency* components. This is fundamental for understanding

many physical problems as there is clear and intuitive meaning to the Fourier modes.

In contrast to selecting basis functions for simplicity or intuitive meaning, the broader question can be asked: what criteria should be used for selecting the functions $\phi_j(x)$. This is an interesting question given that any complete basis can approximate the function $f(x,t)$ to any desired accuracy given N large enough. But what is desired is the best basis functions such that, with N as small as possible, we achieve the desired accuracy. The goal is then the following: find a sequence of orthonormal basis functions $\phi_j(x)$ so that first two terms gives the best two-term approximation to $f(x,t)$, or the first five terms gives the best five-term approximation to $f(x,t)$. These special, ordered, orthonormal functions are called the *proper orthogonal modes* (POD) for the function $f(x,t)$. With these modes, the expansion (4.5.123) is called the POD of $f(x,t)$. The relation to the SVD will become obvious momentarily.

Example: Consider an approximation to the following surface

$$f(x,t) = e^{-|(x-0.5)(t-1)|} + \sin(xt) \quad x \in [0,1],\, t \in [0,2]. \tag{4.5.124}$$

As listed above, there are a number of methods available for approximating this function using various basis functions. And all of the ones listed are guaranteed to converge to the surface for a large enough N in (4.5.120).

In the POD method, the surface is first discretized and approximated by a finite number of points. In particular, the following discretization will be used:

```
x=linspace(0,1,25);
t=linspace(0,2,50);
```

where x has been discretized into 25 points and t is discretized into 50 points. The surface is represented in Fig. 58(a) over the domain of interest. The surface is constructed by using the **meshgrid** command as follows:

```
[X,T]=meshgrid(x,t);
f=exp(-abs((X-.5).*(T-1)))+sin(X.*T);
subplot(2,2,1)
surfl(X,T,f), shading interp, colormap(gray)
```

The **surfl** produces a lighted surface that in this case is represented in grayscale.

The basis functions are then computed using the SVD. Recall that the SVD produces ordered singular values and associated orthonormal basis functions that capture as much energy as possible. Thus an SVD can be performed on the matrix **f** that represents the surface.

```
[u,s,v]=svd(f);  % perform SVD
```

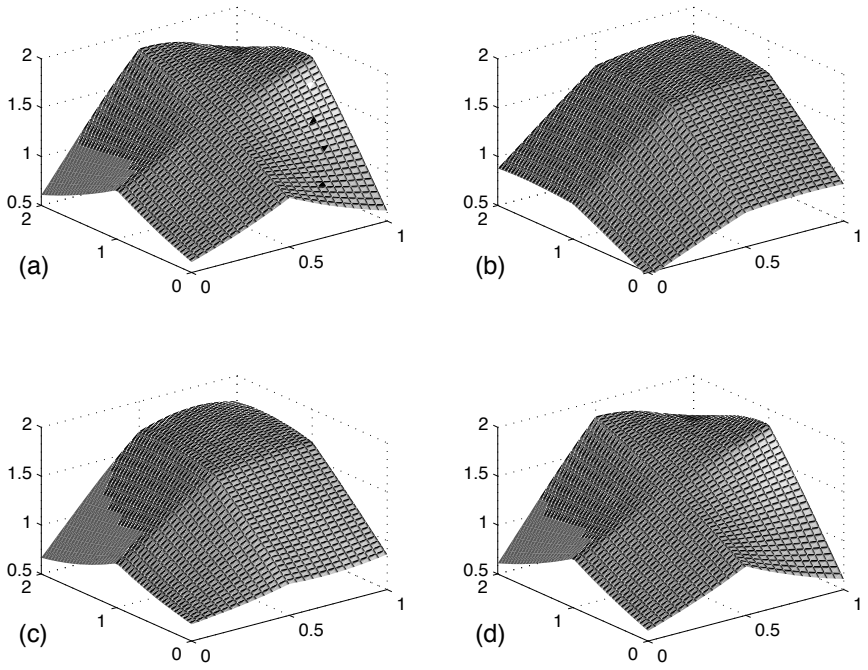


Figure 58: Representation of the surface (a) by a series of low-rank (low-dimensional) approximations with (b) one-mode, (c) two-modes and (d) three-modes. The energy captured in the first mode is approximate 92% of the total surface energy. The first three modes together capture 99.9% of the total surface energy, thus suggesting that the surface can be easily and accurately represented by a three-mode expansion.

```
for j=1:3
   ff=u(:,1:j)*s(1:j,1:j)*v(:,1:j)'; % modal projections
   subplot(2,2,j+1)
   surfl(X,T,ff), shading interp, colormap(gray)
   set(gca,'Zlim',[0.5 2])
end
subplot(2,2,1), text(-0.5,1,0.5,'(a)','Fontsize',[14])
subplot(2,2,2), text(-0.5,1,0.5,'(b)','Fontsize',[14])
subplot(2,2,3), text(-0.5,1,0.5,'(c)','Fontsize',[14])
subplot(2,2,4), text(-0.5,1,0.5,'(d)','Fontsize',[14])
```

The SVD command pulls out the diagonal matrix along with the two unitary matrices $\mathbf{U}$ and $\mathbf{V}$. The loop above processes the sum of the first, second and third modes. This gives the POD modes of interest. Figure 58 demonstrates the modal decomposition and the accuracy achieved with representing the surface

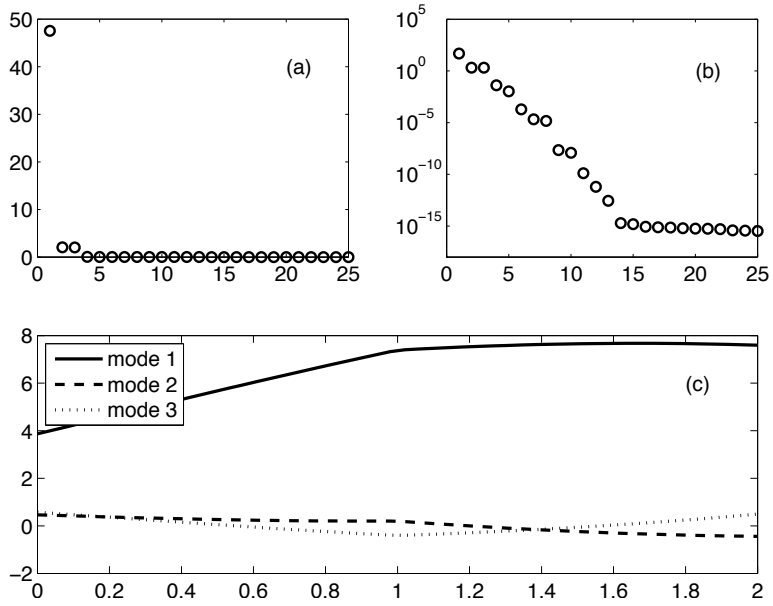


Figure 59: Singular values on a standard plot (a) and a log plot (b) showing the energy in each POD mode. For the surface considered in Fig. 58, the bulk of the energy is in the first POD mode. A three mode approximation produces 99.9% of the energy. The linear POD modes are illustrated in panel (c).

with one, two and three POD modes. The first mode alone captures 92% of the surface while three modes produce a staggering 99.9% of the energy. This should be contrasted with Fourier methods, for instance, which would require potentially hundreds of modes to achieve such accuracy. *As stated previously, these are the best one, two and three mode approximations of the function $f(x,t)$ that can be achieved.*

To be more precise about the nature of the POD, consider the *energy* in each mode. This can be easily computed, or already has been computed, in the SVD, i.e. they are the singular values σ_j. In MATLAB, the energy in the one mode and three mode approximation is computed from

```
energy1=sig(1)/sum(sig)
energy3=sum(sig(1:3))/sum(sig)
```

More generally, the entire *spectrum* of singular values can be plotted. Figure 59 shows the complete spectrum of singular values on a standard plot and a log plot. The clear dominance of a single mode is easily deduced. Additionally, the first three POD modes are illustrated. These constitute the orthonormal expansion basis of interest. The code for producing these plots is as follows:

```
sig=diag(s);
subplot(2,2,1), plot(sig,'ko','Linewidth',[1.5])
axis([0 25 0 50])
set(gca,'Fontsize',[13],'Xtick',[0 5 10 15 20 25])
text(20,40,'(a)','Fontsize',[13])
subplot(2,2,2), semilogy(sig,'ko','Linewidth',[1.5])
axis([0 25 10^(-18) 10^(5)])
set(gca,'Fontsize',[13],'Ytick',[10^(-15) 10^(-10) 10^(-5) 10^0 10^5],...
    'Xtick',[0 5 10 15 20 25]);
text(20,10^0,'(b)','Fontsize',[13])

v2=v(:,1:3);  % pull out modes
c=v2'*f';  % project the modes

subplot(2,1,2)
plot(t,-c(1,:),'k',t,c(2,:),'k--',t,c(3,:),'k:','Linewidth',[2])
set(gca,'Fontsize',[13])
legend('mode 1','mode 2','mode 3','Location','NorthWest')
text(1.8,6,'(c)','Fontsize',[13])
```

Note that the key part of the code is simply the calculation of the POD modes given by the matrix $\mathbf{c}$.

Summary and Limits of SVD/PCA/POD

[u, s, v] = svd(X)

Y = U*X

Y is the projection of our data onto the best possible coordinate system (a system where all the coordinates are independent; covariance matrix is diagonal; zero covariance between coordinates)

U contains the principal components (directions) of our data. i.e. the best possible coordinate system for our data. Sigma weights how important each direction is in our coordinate system. V* describes the weights of each of our data measurements in the new basis.

i.e. USigma is our basis and V* is the coordinate of our data X in that basis

The methods of PCA and POD, which are essentially related to the idea of diagonalization of any matrix via the SVD, are exceptionally powerful tools and methods for the evaluation data driven systems. Further, they provide the most precise method for performing low-dimensional reductions of a given system. A number of key steps are involved in applying the method to experimental or computational data sets.

SVD does BAD with translations!

Given two identical images, shifting one of them will eliminate all of their correlation

- Organize the data into a matrix $\mathbf{A} \in \mathbb{C}^{m\times n}$ where m is the number of measurement types (or probes) and n is the number of measurements (or trials) taken from each probe.
- Subtract off the mean for each measurement type or row $\mathbf{x}_j$.
  and divide by variance if measurements have different units?
- Compute the SVD and singular values of the covariance matrix to determine the principal components.

If considering a fitting algorithm, then the POD method is the appropriate technique to consider and the SVD can be applied directly to the $\mathbf{A} \in \mathbb{C}^{m\times n}$ matrix. This then produces the singular values as well as the POD modes of interest.

Although extremely powerful, and seemingly magical in its ability to produce insightful quantification of a given problem, these SVD methods are not without limits. Some fundamental assumptions have been made in producing the results thus far, the most important being the assumption of *linearity*. Recently some efforts to extend the method using nonlinearity prior to the SVD have been explored [16, 17, 18]. A second assumption is that larger variances (singular values) represent more important dynamics. Although generally this is believed to be true, there are certainly cases where this assumption can be quite misleading. Additionally, in constructing the covariance matrix, it is assumed that the mean and variance are sufficient statistics for capturing the underlying the dynamics. It should be noted that only exponential (Gaussian) probability distributions are completely characterized by the first two moments of the data. This will be considered further in the next section on independent component analysis. Finally, it is often the case that PCA may not produce the optimal results. For instance, consider a point on a propeller blade. Viewed from the PCA framework, two orthogonal components are produced to describe the circular motion. However, this is really a one-dimensional system that is solely determined by the phase variable. The PCA misses this fact unless use is made of the information already known, i.e. that a polar coordinate system with fixed radius is the appropriate context to frame the problem. Thus blindly applying the technique can also lead to non-optimal results that miss obvious facts.

i.e. the components of your system are linear…

5 Independent Component Analysis

The concept of principal components or of a proper orthogonal decomposition are fundamental to data analysis. Essentially, there is an assertion, or perhaps a hope, that underlying seemingly complex and unordered data, there is in fact, some characteristic, and perhaps low-dimensional ordering of the data. The singular value decomposition of a matrix, although a mathematical idea, was actually a formative tool for the interpretation and ordering of the data. In this section on *independent component analysis* (ICA), the ideas turn to data sets for which there are more than one statistically independent objects in the data. ICA provides a foundational mathematical method for extracting interleaved data sets in a fairly straightforward way. Its applications are wide and varied, but our primary focus will be application of ICA in the context of image analysis.

i.e. multiple conversations in a room that we are recording

5.1 The concept of independent components

Independent component analysis (ICA) is a powerful tool that extends the concepts of PCA, POD and SVD. Moreover, you are already intuitively familiar with the concept as some of the examples here will show. A simple way to motivate thinking about ICA is by considering the *cocktail party problem*. Thus consider two conversations in a room that are happening simultaneously. How

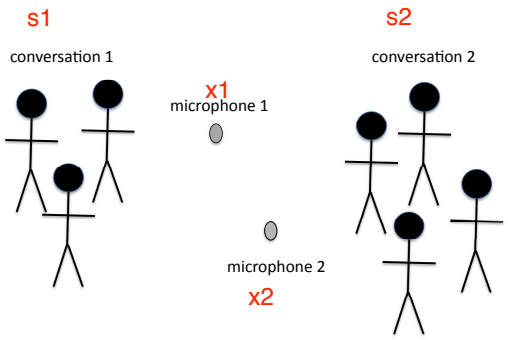


Figure 60: Envisioned cocktail party problem. Two groups of people are having separate conversations which are recorded from microphone one and two. The signals from the two conversations are mixed together at microphone one and two according to the placement relative to the conversations. From the two microphone recordings, the individual conversations can then be extracted.

is it that the two different acoustic signals of conversation one and two can be separated out? Figure 60 depicts a scenario where two groups are conversing. Two microphones are placed in the room at different spatial locations and from the two signals $s_1(t)$ and $s_2(t)$ a mathematical attempt is made to separate the signals that have been mixed at each of the microphone locations. Provided that the noise level is not too large or that the conversation volumes are sufficiently large, humans can perform this task with remarkable ease. In our case, the two microphones are our two ears. Indeed, this scenario and its mathematical foundations are foundational to the concept of eavesdropping on a conversation.

From a mathematical standpoint, this problem can be formulated with the following mixing equation

a = mixing coefficients

$$x_1(t) = a_{11}s_1 + a_{12}s_2 \tag{5.1.125a}$$

$$x_2(t) = a_{21}s_1 + a_{22}s_2 \tag{5.1.125b}$$

where $x_1(t)$ and $x_2(t)$ are the mixed, recorded signals at microphone one and two respectively. The coefficients a_{ij} are the mixing parameters that are determined by a variety of factors including the placement of the microphones in the room, the distance to the conversations, and the overall room acoustics. Note that we are omitting time-delay signals that may reflect off the walls of the room. This problem also resembles quite closely what may happen in a large number of applications. For instance, consider the following

Assume that the two signals are statistically independent!

- **Radar detection:** If there are numerous targets that are being tracked, then there is significant mixing in the scattered signal from all of the targets. Without a method for clear separation of the targets, the detector becomes useless for identifying location.

- **Electroencephalogram (EEG):** EEG readings are electrical recordings of brain activities. Typically these EEG readings are from multiple locations of the scalp. However, at each EEG reading location, all the brain activity signals are mixed, thus preventing a clear understanding of how many underlying signals are contained within. With a large number of EEG probes, ICA allows for the separation of the brain activity readings and a better assessment of the overall neural activity.

- **Terminator Salvation:** If you remember in the movie, John Connor and the resistance found a hidden signal (ICA) embedded on the normal signals in the communications sent between the terminators and skynet vehicles and ships. Although not mentioned in the movie, this was clearly somebody in the future who is taking AMATH 582 now. Somehow that bit of sweet math only made it to the cutting room floor. Regardless, one shouldn't under-restimate how awesome data analysis skills are in the real world of the future.

The ICA method is closely related to the *blind source separation* (BSS) (or blind signal separation) method. Here the sources refer to the original signals, or independent components, and blind refers to the fact that the mixing matrix coefficients a_{ij} are unknown.

A more formal mathematical framework for ICA can be established by considering the following: *Given N distinct linear combinations of N signals (or data sets), determine the original N images.* This is the succinct statement of the mathematical objective of ICA. Thus to generalize (5.1.125) to the N dimensional system we have

measurement is some linear combination of independent signals

$$x_j(t) = a_{j1}s_1 + a_{j2}s_2 + \cdots + a_{jN}s_N \quad 1 \le j \le N\,. \tag{5.1.126}$$

Expressed in matrix form, this results in

RHS is completely unknown

$$\vec{\mathbf{x}} = \mathbf{A}\mathbf{s} \tag{5.1.127}$$

where $\mathbf{x}$ are the mixed signal measurements, $\mathbf{s}$ are the original signals we are trying to determine, and $\mathbf{A}$ is the matrix of coefficients which determines how the signals are mixed as a function of the physical system of interest. At first, one might simply say that the solution to this is trivial and it is given by

$$\mathbf{s} = \mathbf{A}^{-1}\mathbf{x} \tag{5.1.128}$$

where we are assuming that the placement of our measurement devices are such that $\mathbf{A}$ is nonsingular and its inverse exists. Such a solution makes the following assumption: we know the matrix cofficients a_{ij}. The point is, we don't know them. Nor do we know the signal $\mathbf{s}$. Mathematically then, the aim of ICA is to approximate the coefficients of the matrix $\mathbf{A}$, and in turn the original signals $\mathbf{s}$, under as general assumptions as possible.

signals. It looks
for the underlying dynamics of a single

ICA makes the assumption that we have multiple linear systems interacting

SVD and PCA ICA gaussian distributed variables dont do well with ICA

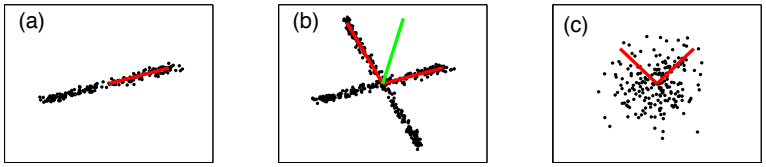


Figure 61: Illustration of the principals of PCA (a), ICA (b) and the failure of Gaussian distributions to distinguish principal components (c). The red vectors show the principal directions, while the green vector in the middle panel shows what would be the principal direction if a direct SVD where applied rather than an ICA. The principal directions in (c) are arbitrarily chosen since no principal directions can be distinguished.

The statistical model given by Eq. (5.1.127) is called *independent component analysis*. The ICA is a *generative model*, which means that is describes how the observed data are generated by a process of mixing in the components $s_j(t)$. The independent components are *latent variables*, meaning they are never directly observed. The key step in the ICA models is to assume the following: *the underlying signals* $s_j(t)$ *are* statistically independent *with probability distributions that are* not *Gaussian*. Statistical independence and higher-order moments of the probably distribution (thus requiring non-Gaussian distributions) are the critical mathematical tools that will allow us to recover the signals.

Image separation and SVD

To make explicit the mathematical methodology to be pursued here, a specific example of image separation will be used. Although there are a variety of mathematical alternatives for separating the independent components [19], the approach considered here will be based upon PCA and SVD. To illustrate the concept of ICA, consider the example data represented in Fig. 61. The three panels are fundamental to understanding the concept of ICA. In the left panel (a), measurements are taken of a given system and are shown to project nicely onto a dominant direction whose leading principal component is denoted by the red vector. This red vector would be the principal component with a length σ_1 determined by the largest singular value. It is clear that the singular value σ_2 corresponding to the orthogonal direction of the second principal component would be considerably smaller. The middle panel (b), the measurements indicate that there are two principal directions in the data fluctuations. If an SVD is applied directly to the data, then the dominant singular direction would be approximately the green vector. Yet it is clear that the green vector does not accurately represent the data. Rather, the data seems to be organized *indepedently* along two different directions. An SVD of the two independent data sets would produce two principal components, i.e. two *independent component*

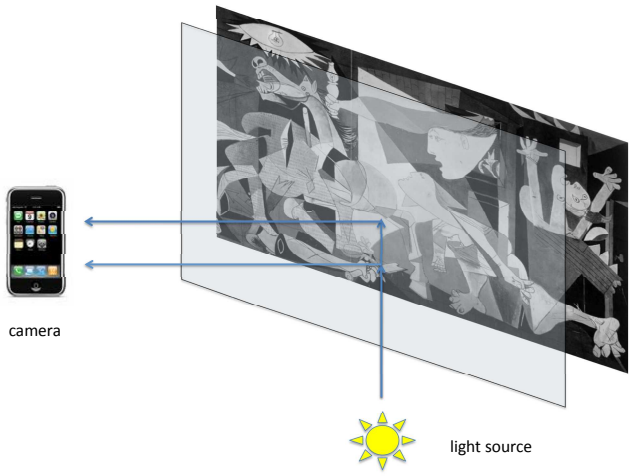


Figure 62: Illustration of image separation problem. A piece of glass reflects part of the background light source while some of the light penetrates to the image and is reflected to the camera. Thus the camera sees two images: the image (painting) and the reflection of the background light source.

analysis vectors. Thus it is critical to establish a method for separating out data of this sort into their independent directions. Finally, in the third panel (c), Gaussian distributed data is shown where no principal components can be identified with certainty. Indeed, there are an infinite number of possible orthogonal projections that can be made. Two arbitrary directions have been shown in panel (c). This graphic shows why Gaussian distributed random variables are disastrous for ICA, no projections onto the independent components can be accomplished.

We now turn our attention to the issue of image separation. Figure 62 demonstrates a prototypical example of what can occur when photographing an image behind glass. The glass encasing of the art work, or image, produces a reflection of the backlit source along with the image itself. Amazingly, the human eye can very easily compensate for this and *see through* much of the reflection to the original image. Alternatively, if we choose, we can also focus on the reflected image of the backlit source. Ultimately, there is some interest in training a computer to do this image processing automatically. Indeed, in artificial vision, algorithms such as these are important for mimicking natural human behaviors and abilities.

The light reflected off the glass is partially polarized and provides the crucial degree of freedom necessary for our purposes. Specifically, to separate the images, a photo is taken as illustrated in Fig. 62 but now with a linear polarizer

polarize light = rays of light move only in a specific direction

Figure 63: **The Judgement of Paris** by John Flaxman (1755-1826). This is a plate made by Flaxman for an illustrated version of Homer's Iliad. The picture was taken in my office and so you can see both a faint reflection of my books as well as strong signature of my window which overlooks Drumheller Fountain.

gives two independent measurements?

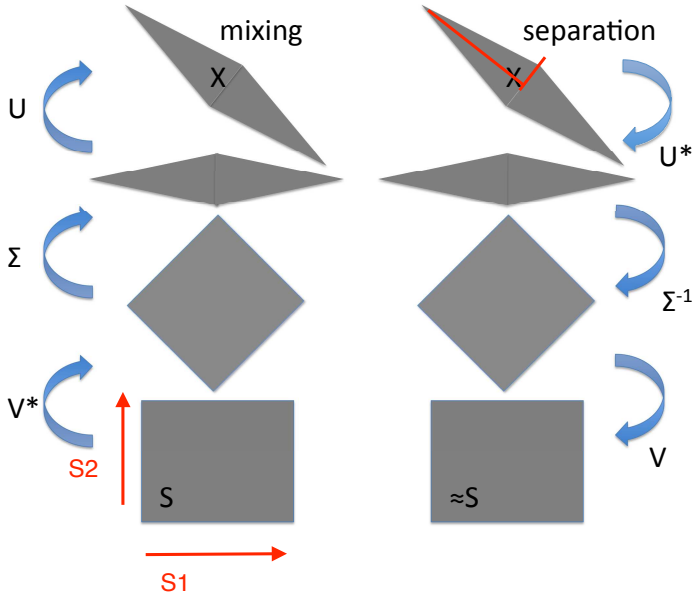
placed in front of the camera lens. Two photos are taken where the linear polarizer is rotated 90 degrees from each other. This is a simple home experiment essentially. But you must have some control over the focus of your camera in terms of focusing and flash. The picture to be considered is hanging in my office: **The Judgement of Paris** by John Flaxman (1755-1826). Flaxman was an English sculptor, draughtsman and illustrator who enjoyed a long and brilliant career that included a stint at Wedgwood in England. In fact, at the age of 19, Josiah Wedgwood hired Flaxman as a modeler of classic and domestic friezes, plagues and ornamental vessels and medallion portraits. You can still find many Wedgwood ornamental vessels today that have signature Flaxman works molded on their sides. It was during this period that the manufactures of that time period perfected the style and earned great reputations. But what gained Flaxman his general fame was not his work in sculpture proper, but his designs and engravings for the **Iliad**, **Odyssey**, and **Dante's Inferno**. All of his works on these mythological subjects where engraved by Piroli with considerable loss to Flaxman's own lines. The greatest collection of Flaxman's work is housed in the Flaxman Gallery at the University College Longon.

In the specific Flaxman example considered here, one of the most famous scenes of mythology is depicted: **The Judgement of Paris**. In this scene,

Figure 64: This is an identical picture to Fig. 63 with a linear polarizer rotated 90 degrees. The polarization state of the reflected field is effected by the linear polarizer, thus allowing us to separate the images.

Hermes watches as young Paris plays the part of judge. His task is to contemplate giving the golden apple with the inscription *To the Fairest* to either Venus, Athena, or Hera. This golden apple was placed at the wedding reception of Peleus and Thetis, father and mother of peerless Achilles, by Discord since she did not receive a wedding invitation. Each lovely goddess in turn promises him something for the apple: Athena promises that he will become the greatest warrior in history, Juno promises him to be greatest ruler in the land, ruling over a vast and unmatched empire. Last comes the lovely Venus who promises him that the most beautiful woman in the world will be his. Of course, to sweeten up her offer, she is wearing next to nothing as usual. Lo and behold, the hot chick wins! And as a result we have the Trojan War for which Helen was fought for. I could go on here, but what else is to be expected of a young 20-something year old male (an older guy would have taken the vast kingdom while a prepubescent boy would have chosen to be the great warrior). It is comforting to know that very little has changed since in some 4000 years. Figures 63 and 64 represent this Flaxman scene along with the reflected image of my office window. The two pictures are taken with a linear polarizer in front of a camera that has been rotated by 90 degrees between the photos.

S = V Sigma^-1 U* X



find where the variance is largest/smallest

x = As

A = U Sigma V*

singular values sigma_j are the variances of our components which we calculated above!

find s1 and s2 such that P(S1S2) ~ P(S1)P(S2)

4th distribution moment?

A is performing my mixing!

take the SVD of A… but we dont know A
to separate the signals, we would have to approximate the SVD inverses of A!

Figure 65: Graphical depiction of the SVD process of mixing the two images. The reconstruction of the images is accomplished by approximating the inversions of the SVD matrices so as to achieve a separable (statistically independent) probability distribution of the two images.

SVD method for ICA

assume S1 and S2 are not gaussian and that they are independent.

Given the physical problem, how is SVD used then to separate the two images observed in Figs. 63 and 64. Consider the action of the SVD on the image mixing matrix $\mathbf{A}$ of Eq. (5.1.127). In this case, there are two images $\mathbf{S}_1$ and $\mathbf{S}_2$ that we are working with. This gives us 6 unknowns $(\mathbf{S}_1, \mathbf{S}_2, a_{11}, a_{12}, a_{21}, a_{22})$ with only the two constraints from Eq. (5.1.127). Thus the system cannot be solved without further assumptions being made. The first assumption will be that the two images are statistically independent. If the pixel intensities are denoted by $\mathbf{S}_1$ and $\mathbf{S}_2$, then statistical independence is mathematically expressed as

S1 & S2 independent

$$P(\mathbf{S}_1, \mathbf{S}_2) = P(\mathbf{S}_1)P(\mathbf{S}_2) \tag{5.1.129}$$

which states that the joint probability density is separable. No specific form is placed upon the probability densities themselves aside from this: they cannot be Gaussian distributed. In practice, this is a reasonable constraint since natural images are rarely Gaussian. A second assumption is that the matrix $\mathbf{A}$ is full rank, thus assuming that the two measurements of the image indeed have been filtered with different polarization states. These assumptions yield an estimation problem which is the ICA.

Consider what happens under the SVD process to the matrix $\mathbf{A}$

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{5.1.130}$$

where again $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices that simply lead to rotation and $\mathbf{\Sigma}$ stretches (scales) an image as prescribed by the singular values. A graphical illustration of this process is shown in Fig. 65 for distributions that are uniform, i.e. they form a square. The mixing matrix $\mathbf{A}$ can be thought to first rotate the square via unitary matrix $\mathbf{V}^*$, then stretch it into a parallelogram via the diagonal matrix $\mathbf{\Sigma}$ and then rotate the parallelogram via the unitary matrix $\mathbf{U}$. This is now the mixed image $\mathbf{X}$. *The estimation, or ICA, of the independent images thus reduces to finding how to transform the rotated parallelogram back into a square. Or mathematically, transforming the mixed image back into a separable product of one-dimensional probability distributions.* This is the defining mathematical goal of the ICA image analysis problem, or any general ICA reduction technique.

5.2 Image separation problem

The method proposed here to separate two images relies on reversing the action of the SVD on the two statistically independent images. Again, the matrix $\mathbf{A}$ in (5.1.130) is not known, so a direct computation of the SVD cannot be performed. However, each of the individual matrices can be approximated by considering its net effect on the assumed uniformly distributed images.

only the first part is an approximation, but the other parts rely on it so the whole process isnt completely exact.

Three specific computations must be made: (i) the rotation of the parallelogram must be approximated, (ii) the scaling of the parallelogram according to the variances must be computed, and (iii) the final rotation back to a separable probability distribution must be obtained. Each step is outlined below.

Step 1: Rotation of the parallelogram

To begin, consider once again Fig. 65. Our first objective is to undo the rotation of the unitary matrix $\mathbf{V}^*$. Thus we will ultimately want to compute the inverse of this matrix which is simply $\mathbf{V}$. In a geometrical way of thinking, our objective is to align the long and short axes of the parallelogram with the primary axis as depicted in the two top right shaded boxes of Fig. 65. The angle of the parallelogram relative to the primary axes will be denoted by θ, and the long and short axes correspond to the axes of the maximal and minimal variance respectively. From the image data itself, then, the maximal and minimal variance directions will be extracted. Assuming mean-zero measurements, the variance at an arbitrary angle of orientation is given by

largest/smallest sigma will be given to the direction with the greatest/smallest variance (see PCA covariance matrix)

$$Var(\theta) = \sum_{j=1}^{N} \left\{ [x_1(j) \;\; x_2(j)] \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \right\}^2 . \tag{5.2.131}$$

where we have N total pixels (i.e. for black and white photos)

why only 2 x's? two images we had earlier, but how do we expand this to N images?

The maximal variance is determined by computing the angle θ that maximizes this function. It will be assumed that the corresponding angle of minimal variance will be orthogonal to this at $\theta - \pi/2$. These axes are essentially the principal component directions that would be computed if we actually knew components of the matrix $\mathbf{A}$.

The maximum of (5.2.131) with respect to θ can be found by differentiating $Var(\theta)$ and setting it equal to zero. To do this, Eq. (5.2.131) is rewritten as

$$\begin{aligned} Var(\theta) &= \sum_{j=1}^{N} \left\{ [x_1(j) \;\; x_2(j)] \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \right\}^2 \\ &= \sum_{j=1}^{N} [x_1(j)\cos\theta + x_2(j)\sin\theta]^2 \\ &= \sum_{j=1}^{N} x_1^2(j)\cos^2\theta + 2x_1(j)x_2(j)\cos\theta\sin\theta + x_2^2(j)\sin^2\theta. \end{aligned} \tag{5.2.132}$$

Differentiating with respect to θ then gives

$$\begin{aligned} \frac{d}{d\theta}Var(\theta) &= 2\sum_{j=1}^{N} -x_1^2(j)\sin\theta\cos\theta + x_1(j)x_2(j)\left[\cos^2\theta - \sin^2\theta\right] + x_2^2(j)\sin\theta\cos\theta \\ &= 2\sum_{j=1}^{N} \left[x_2^2(j) - x_1^2(j)\right]\sin\theta\cos\theta + x_1(j)x_2(j)\left[\cos^2\theta - \sin^2\theta\right] \\ &= \sum_{j=1}^{N} \left[x_2^2(j) - x_1^2(j)\right]\sin 2\theta + 2x_1(j)x_2(j)\cos 2\theta \end{aligned} \tag{5.2.133}$$

which upon setting this equal to zero gives the value of θ desired. Thus taking $d(Var(\theta))/d\theta = 0$ gives

$$\frac{\sin 2\theta}{\cos 2\theta} = \frac{-2\sum_{j=1}^{N} x_1(j)x_2(j)}{\sum_{j=1}^{N} x_2^2(j) - x_1^2(j)}, \tag{5.2.134}$$

or in terms of θ alone

maximal variance at theta_0
min = theta_0 - pi/2

$$\theta_0 = \frac{1}{2}\tan^{-1}\left[\frac{-2\sum_{j=1}^{N} x_1(j)x_2(j)}{\sum_{j=1}^{N} x_2^2(j) - x_1^2(j)}\right]. \tag{5.2.135}$$

In the polar coordinates $x_1(j) = r(j)\cos\psi(j)$ and $x_2(j) = r(j)\sin\psi(j)$, the expression reduces further to

$$\theta_0 = \frac{1}{2}\tan^{-1}\left[\frac{\sum_{j=1}^{N} r^2(j)\sin 2\psi(j)}{\sum_{j=1}^{N} r^2(j)\cos 2\psi(j)}\right]. \tag{5.2.136}$$

The rotation matrix, or unitary transformation, associated with the rotation of the parallelogram back to its aligned position is then

$$\mathbf{U}^* = \begin{bmatrix} \cos\theta_0 & \sin\theta_0 \\ -\sin\theta_0 & \cos\theta_0 \end{bmatrix} \tag{5.2.137}$$

with the angle θ_0 computed directly from the experimental data.

Step 2: Scaling of the parallelogram

The second task is to undo the principal component scaling achieved by the singular values of the SVD decomposition. This process is illustrated as the second step in the right column of Fig. 65. This task, however, is rendered straightforward now that the principal axes have been determined from step 1. In particular, the assumption was that along the direction θ_0, the maximal variance is achieved, while along $\theta_0 - \pi/2$, the minimal variance is achieved. Thus the components, or singular values, of the diagonal matrix $\mathbf{\Sigma}^{-1}$ can be computed.

we already calculated the variances in the previous step

The variance along the two principal component axes are given by

$$\sigma_1 = \sum_{j=1}^{N} \left\{ [x_1(j) \;\; x_2(j)] \begin{bmatrix} \cos\theta_0 \\ \sin\theta_0 \end{bmatrix} \right\}^2 \tag{5.2.138a}$$

$$\sigma_2 = \sum_{j=1}^{N} \left\{ [x_1(j) \;\; x_2(j)] \begin{bmatrix} \cos(\theta_0 - \pi/2) \\ \sin(\theta_0 - \pi/2) \end{bmatrix} \right\}^2 . \tag{5.2.138b}$$

This then gives the diagonal elements of the matrix $\mathbf{\Sigma}$. To undo this scaling, the inverse of $\mathbf{\Sigma}$ is constructed so that

$$\mathbf{\Sigma}^{-1} = \begin{bmatrix} 1/\sqrt{\sigma_1} & 0 \\ 0 & 1/\sqrt{\sigma_2} \end{bmatrix} . \tag{5.2.139}$$

This matrix, in combination with that of step 1, easily undoes the principal component direction rotation and its associated scaling. However, this process has only decorrelated the images, and a separable probability distribution has not yet been produced.

Step 3: Rotation to produce a separable probability distribution

The final rotation to separate the probability distributions is a more subtle and refined issue, but critical to producing nearly separable probability distributions. This separation process typically relies on the higher moments of the probability distribution. Since the mean has been assumed to be zero and there is no reason to believe that there is an asymmetry in the probability distributions, i.e. higher-order odd moments (such as the skewness) are negligible, the next

dominant statistical moment to consider is the fourth moment, or the kurtosis of the probability distribution. The goal will be to minimize this fourth-order moment, and by doing so we will determine the appropriate rotation angle. Note that the second moment has already been handled through steps and 1 and 2. Said in a different mathematical way, minimizing the kurtosis will be another step in trying to approximate the probability distribution of the images as separable functions so that $P(\mathbf{S}_1\mathbf{S}_2) \approx P(\mathbf{S}_1)P(\mathbf{S}_2)$.

minimize kurtosis of the data after U* and Sigma^-1 have been applied

Minimize fourth order moment of our data

The kurtosis of a probability distribution is given by

$$kurt(\phi) = K(\phi) = \sum_{j=1}^{N} \left\{ [\bar{x}_1(j) \quad \bar{x}_2(j)] \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix} \right\}^4 \tag{5.2.140}$$

where ϕ is the angle of rotation associated with the unitary matrix $\mathbf{U}$ and the variables $\bar{x}_1(j)$ and $\bar{x}_2(j)$ is the image that has undergone the two steps of transformation as outlined previously.

For analytic convenience, a normalized version of the above definition of kurtosis will be considered. Specifically, a normalized version is computed in practice. The expedience of the normalization will become clear through the algebraic manipulations. Thus consider

$$\bar{K}(\phi) = \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j) + \bar{x}_2^2(j)} \left\{ [\bar{x}_1(j) \quad \bar{x}_2(j)] \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix} \right\}^4 . \tag{5.2.141}$$

As in our calculation regarding the second moment, the kurtosis will be written in a more natural form for differentiation

$$\begin{aligned}
\bar{K}(\phi) &= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j) + \bar{x}_2^2(j)} \left\{ [\bar{x}_1(j) \quad \bar{x}_2(j)] \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix} \right\}^4 \\
&= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j) + \bar{x}_2^2(j)} [\bar{x}_1(j)\cos\phi + \bar{x}_2(j)\sin\phi]^4 \qquad (5.2.142) \\
&= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j) + \bar{x}_2^2(j)} \left[\bar{x}_1^4(j)\cos^4\phi + 4\bar{x}_1^3(j)\bar{x}_2(j)\cos^3\phi\sin\phi \right. \\
&\quad \left. +6\bar{x}_1^2(j)\bar{x}_2^2(j)\cos^2\phi\sin^2\phi + 4\bar{x}_1(j)\bar{x}_2^3(j)\cos\phi\sin^3\phi + \bar{x}_2^4(j)\sin^4\phi \right] \\
&= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j) + \bar{x}_2^2(j)} \left[\frac{1}{8}\bar{x}_1^4(j)(3 + 4\cos 2\phi + \cos 4\phi) \right. \\
&\quad +\bar{x}_1^3(j)\bar{x}_2(j)(\sin 2\phi + (1/2)\sin 4\phi) + \frac{3}{4}\bar{x}_1^2(j)\bar{x}_2^2(j)(1 - \cos 4\phi) \\
&\quad \left. +\bar{x}_1(j)\bar{x}_2^3(j)(\sin 2\phi - (1/2)\sin 4\phi) + \frac{1}{8}\bar{x}_2^4(j)(3 - 4\cos 2\phi + \cos 4\phi) \right] .
\end{aligned}$$

This quantity needs to be minimized with an appropriate choice of ϕ. Thus the derivative $d\bar{K}/d\phi$ must be computed and set to zero.

$$\begin{aligned}
\frac{d\bar{K}(\phi)}{d\phi} &= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j)+\bar{x}_2^2(j)} \left[\frac{1}{8}\bar{x}_1^4(j)(-8\sin 2\phi - 4\sin 4\phi) \right. \\
&\quad + \bar{x}_1^3(j)\bar{x}_2(j)(2\cos 2\phi + 2\cos 4\phi) + 3\bar{x}_1^2(j)\bar{x}_2^2(j)\sin 4\phi \\
&\quad \left. + \bar{x}_1(j)\bar{x}_2^3(j)(2\cos 2\phi - 2\cos 4\phi) + \frac{1}{8}\bar{x}_2^4(j)(8\sin 2\phi - 4\sin 4\phi) \right] \\
&= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j)+\bar{x}_2^2(j)} \left\{ [\bar{x}_2^4(j) - \bar{x}_1^4(j)]\sin 2\phi \right. \\
&\quad + [2\bar{x}_1(j)\bar{x}_2(j)^3 + 2\bar{x}_1^3(j)\bar{x}_2(j)]\cos 2\phi + [2\bar{x}_1^3(j)\bar{x}_2(j) - 2\bar{x}_1(j)\bar{x}_2^3(j)]\cos 4\psi \\
&\quad \left. + [3\bar{x}_1(j)^2\bar{x}_2^2(j) - (1/2)\bar{x}_1^4(j) - (1/2)\bar{x}_2^4(j)]\sin 4\psi \right\} \\
&= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j)+\bar{x}_2^2(j)} \left\{ [(\bar{x}_1^2(j)+\bar{x}_2^2(j))(\bar{x}_2^2(j) - \bar{x}_1^2(j))]\sin 2\phi \right. \\
&\quad + [2\bar{x}_1(j)\bar{x}_2(j)(\bar{x}_1^2(j) + \bar{x}_2^2(j))]\cos 2\phi + [2\bar{x}_1^3(j)\bar{x}_2(j) - 2\bar{x}_1(j)\bar{x}_2^3(j)]\cos 4\psi \\
&\quad \left. + [3\bar{x}_1(j)^2\bar{x}_2^2(j) - (1/2)\bar{x}_1^4(j) - (1/2)\bar{x}_2^4(j)]\sin 4\psi \right\} \\
&= \sum_{j=1}^{N} [\bar{x}_2^2(j) - \bar{x}_1^2(j)]\sin 2\phi + 2\bar{x}_1(j)\bar{x}_2(j)\cos 2\phi \\
&\quad + \frac{1}{\bar{x}_1^2(j)+\bar{x}_2^2(j)} \left\{ [2\bar{x}_1^3(j)\bar{x}_2(j) - 2\bar{x}_1(j)\bar{x}_2^3(j)]\cos 4\psi \right. \\
&\quad \left. + [3\bar{x}_1(j)^2\bar{x}_2^2(j) - (1/2)\bar{x}_1^4(j) - (1/2)\bar{x}_2^4(j)]\sin 4\psi \right\} \\
&= \sum_{j=1}^{N} \frac{1}{\bar{x}_1^2(j)+\bar{x}_2^2(j)} \left\{ [2\bar{x}_1^3(j)\bar{x}_2(j) - 2\bar{x}_1(j)\bar{x}_2^3(j)]\cos 4\psi \right. \\
&\quad \left. + [3\bar{x}_1(j)^2\bar{x}_2^2(j) - (1/2)\bar{x}_1^4(j) - (1/2)\bar{x}_2^4(j)]\sin 4\psi \right\} , \qquad (5.2.143)
\end{aligned}$$

where the terms proportional to $\sin 2\phi$ and $\cos 2\phi$ add to zero since they are minimized when considering the second moment or variance, i.e. we would like to minimize both the variance and the kurtosis, and this calculation does both. By setting this to zero, the angle ϕ_0 can be determined to be

$$\phi_0 = \frac{1}{4}\tan^{-1}\left[\frac{-\sum_{j=1}^{N}[2\bar{x}_1^3(j)\bar{x}_2(j) - 2\bar{x}_1(j)\bar{x}_2^3(j)]/[\bar{x}_1^2(j)+\bar{x}_2^2(j)]}{\sum_{j=1}^{N}[3\bar{x}_1(j)^2\bar{x}_2^2(j) - (1/2)\bar{x}_1^4(j) - (1/2)\bar{x}_2^4(j)]/[\bar{x}_1^2(j)+\bar{x}_2^2(j)]} \right] \qquad (5.2.144)$$

When converted to polar coordinates, this reduces very nicely to the following expression:

$$\phi_0 = \frac{1}{4}\tan^{-1}\left[\frac{\sum_{j=1}^{N} r^2 \sin 4\psi(j)}{\sum_{j=1}^{N} r^2 \cos 4\psi(j)} \right] \qquad (5.2.145)$$

angle of final rotation that V should perform to minimize fourth moment of data.

The rotation back to the approximately statistically independent square is then given by

$$\mathbf{V} = \begin{bmatrix} \cos\phi_0 & \sin\phi_0 \\ -\sin\phi_0 & \cos\phi_0 \end{bmatrix} \tag{5.2.146}$$

with the angle θ_0 computed directly from the experimental data. At this point, it is unknown wether the angle ϕ_0 is a minimum or maximum of the kurtosis and this should be checked. Sometimes this is a maximum, especially in cases when one of the image histograms has long tails relative to the other [20].

Completing the analysis

Recall that our purpose was to compute, through the statistics of the images (pixels) themselves, the SVD decomposition. The method relied on computing (approximating) the principal axis and scaling of the principal components. The final piece was to try and make the probability distribution as separable as possible by minimizing the kurtosis as a function of the final rotation angle.

To reconstruct the image, the following linear algebra operation is performed:

$$\begin{aligned} \mathbf{s} &= \mathbf{A}^{-1}\mathbf{x} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^*\mathbf{x} \\ &= \begin{bmatrix} \cos\phi_0 & \sin\phi_0 \\ -\sin\phi_0 & \cos\phi_0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{\sigma_1} & 0 \\ 0 & 1/\sqrt{\sigma_2} \end{bmatrix} \begin{bmatrix} \cos\theta_0 & \sin\theta_0 \\ -\sin\theta_0 & \cos\theta_0 \end{bmatrix} \mathbf{x} \end{aligned} \tag{5.2.147}$$

A few things should be noted about the inherent ambiguities in the recovery of the two images. The first is the ordering ambiguity. Namely, the two matrices are indistinguishable:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{21} & a_{22} \\ a_{11} & a_{12} \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \end{bmatrix} \tag{5.2.148}$$

Thus image one and two are arbitrary to some extent. In practice this does not matter since the aim was simply to separate, not label, the data measurements. There is also an ambiguity in the scaling since

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}/\alpha & a_{12}/\delta \\ a_{21}/\alpha & a_{22}/\delta \end{bmatrix} \begin{bmatrix} \alpha x_1 \\ \delta x_2 \end{bmatrix}. \tag{5.2.149}$$

Again this does not matter since the two separated images can then be rescaled to their full intensity range afterwards. Regardless, this shows the basic process that needs to be applied to the system data in order to construct a self-consistent algorithm capable of image separation.

5.3 Image separation and MATLAB

Using MATLAB, the previously discussed algorithm will be implemented. However, before proceeding to try and extract the images displayed in Figs. 63 and

Figure 66: Two images of Sicily: an ocean view from the hills of Erice and an interior view of the Capella Palatina in Palermo.

64, a more idealized case is considered. Consider the two ideal images of Sicily in Fig. 66. These two images will be mixed with two different weights to produce two mixed images. Our objective will be to use the algorithm outlined in the last section to separate out the images. Upon separation, the images can be compared with the ideal representation of Fig. 66. To load in the images and start the processing, the following MATLAB commands are used.

```
S1=imread('sicily1','jpeg');
S2=imread('sicily2','jpeg');

subplot(2,2,1), imshow(S1);
subplot(2,2,2), imshow(S2);
```

The two images are quite different with one overlooking the mediterranean from the medieval village of Erice on the Northwest corner, and the second is of the Capella Palatina in Palermo. Recall that what is required is statistical independence of the two images. Thus the pixel strengths and colors should be largely decorrolated throughout the image.

The two ideal images are now mixed with the matrix $\mathbf{A}$ in Eq. (5.1.127). Specifically, we will consider the arbitrarily chosen mixing of the form

randomly mix our data.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} 4/5 & \beta \\ 1/2 & 2/3 \end{bmatrix} \tag{5.3.150}$$

where in what follows we will consider the values $\beta = 1/5, 3/5$. This mixing produces two composite images with the strengths of image one and two altered between the two composites. This can be easily accomplished in MATLAB with the following commands.

```
A=[0.8 0.2; 1/2 2/3];  % mixing matrix
X1=double(A(1,1)*S1+A(1,2)*S2);
X2=double(A(2,1)*S1+A(2,2)*S2);
```

Figure 67: The top two figures show the mixed images produced from Eq. (5.3.150) with $\beta = 1/5$. The bottom two figures are the reconstructed images from the image separation process and approximation to the SVD. For the value of $\beta = 1/5$, the Capella Palatina has been exceptionally well reconstructed while the view from Erice has quite a bit of residual from the second image.

```
subplot(2,2,1), imshow(uint8(X1));
subplot(2,2,2), imshow(uint8(X2));
```

Note that the coefficients of the mixing matrix **A** will have a profound impact on our ability to extract one image from another. Indeed, just switching the parameter β from 1/5 to 3/5 will show the impact of a small change to the mixing matrix. It is these images that we wish to reconstruct by numerically computing and approximation to the SVD. The top row of Figs. 67 and 68 demonstrate the mixing that occurs with the two ideal images given the mixing matrix (5.3.150) with $\beta = 1/5$ (Fig. 67) and $\beta = 3/5$ (Fig. 68).

Image statistics and the SVD

The image separation method relies on the statistical properties of the image for reconstructing an approximation to the SVD decomposition. Three steps have been outlined in the last section that must be followed: first the rotation of the

Figure 68: The top two figures show the mixed images produced from Eq. (5.3.150) with $\beta = 3/5$. The bottom two figures are the reconstructed images from the image separation process and approximation to the SVD. For the value of $\beta = 3/5$, the view from Erice has been has been well reconstructed aside from the top right corner of the image while the Capella Palatina remains of poor quality.

parallelogram must be computed by finding the maximal and minimal directions of the variance of the data. Second, the scaling of the principal component directions are evaluated by calculating the variance, and third, the final rotation is computed by minimizing both the variance and kurtosis of the data. This yields an approximately separable probability distribution. The three steps are each handled in turn.

Step 1: maximal/minimal variance angle detection. This step is illustrated in the top right of Fig. 65. The actual calculation that must be performed for calculating the rotation angle is given by either Eq. (5.2.135) or (5.2.136). Once computed, the desired rotation matrix $\mathbf{U}^*$ given by Eq. (5.2.137) can be constructed. Recall the underlying assumption that a mean-zero distribution is considered. In MATLAB form, this computation takes the following form.

```
[m,n]=size(X1);
x1=reshape(X1,m*n,1);
x2=reshape(X2,m*n,1);
```

```
x1=x1-mean(x1); x2=x2-mean(x2);
```
dont forget to demean your data!

```
theta0=0.5*atan( -2*sum(x1.*x2) / sum(x1.^2-x2.^2) )
Us=[cos(theta0) sin(theta0); -sin(theta0) cos(theta0)];
```

This completes the first step in the SVD reconstruction process.

Step 2: scaling of the principal components. The second step is to undo the scaling/compression that has been performed by the singular values along the principal component directions θ_0 and $\theta_0 - \pi/2$. The rescaling matrix thus is computed as follows:

```
sig1=sum( (x1*cos(theta0)+x2*sin(theta0)).^2 )
sig2=sum( (x1*cos(theta0-pi/2)+x2*sin(theta0-pi/2)).^2 )
Sigma=[1/sqrt(sig1) 0; 0 1/sqrt(sig2)];
```

Note that the variable called sigma above is actually the inverse of the diagonal matrix constructed from the SVD process. This completes the second step in the SVD process.

Step 3: rotation to separability. The final rotation is aimed towards producing, as best as possible, a separable probability distribution. The analytic method used to do this is to minimize both the variance and kurtosis of the remaining distributions. The angle that accomplishes this task is computed analytically from either (5.2.144) or (5.2.145) and the associated rotation matrix $\mathbf{V}$ is given by (5.2.146). Before computing this final rotation, the image after steps 1 and 2 must be produced, i.e. we compute the $\bar{\mathbf{X}}_1$ and $\bar{\mathbf{X}}_2$ used in (5.2.144) and (5.2.145). The MATLAB code used to produce the final rotation matrix is then

```
X1bar= Sigma(1,1)*(Us(1,1)*X1+Us(1,2)*X2);
X2bar= Sigma(2,2)*(Us(2,1)*X1+Us(2,2)*X2);

x1bar=reshape(X1bar,m*n,1);
x2bar=reshape(X2bar,m*n,1);

phi0=0.25*atan( -sum(2*(x1bar.^3).*x2bar-2*x1bar.*(x2bar.^3)) ...
  / sum(3*(x1bar.^2).*(x2bar.^2)-0.5*(x1bar.^4)-0.5*(x2bar.^4)) )

V=[cos(phi0) sin(phi0); -sin(phi0) cos(phi0)];
S1bar=V(1,1)*X1bar+V(1,2)*X2bar;
S2bar=V(2,1)*X1bar+V(2,2)*X2bar;
```

This final rotation completes the three steps of computing the SVD matrix. However, the images needed to be rescaled back to the full image brilliance.

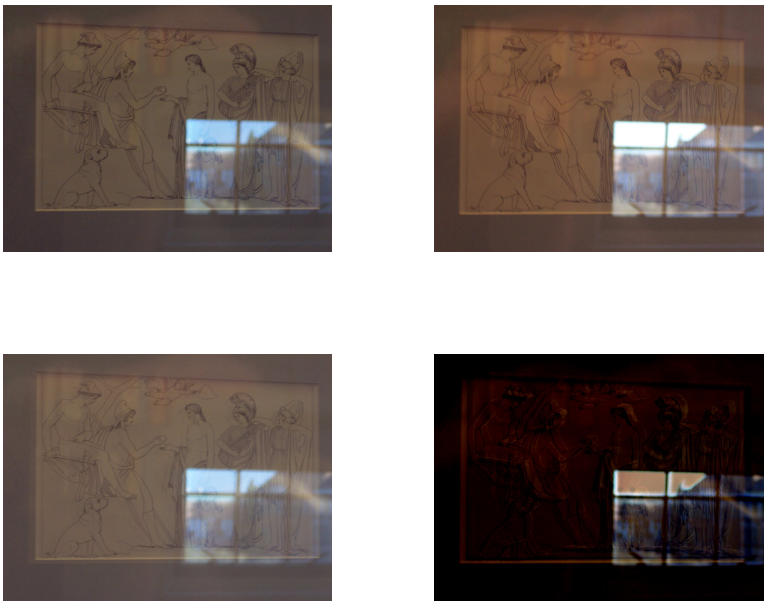


Figure 69: **The Judgement of Paris** by Flaxman with a reflection of my window which overlooks Drumheller Fountain. In the top row are the two original photos taken with linear polarizer rotated 90 degrees between shots. The bottom row represents the separated images. The brightness of the reflected image greatly effects the quality of the image separation.

Thus the data points need to be made positive and scaled back to values ranging from 1 (dim) to 255 (bright). This final bit of code rescales and plots the separated images.

```
min1=min(min(min(S1bar)));
S1bar=S1bar+min1;
max1=max(max(max(S1bar)));
S1bar=S1bar*(255/max1);

min2=min(min(min(S2bar)));
S2bar=S2bar+min2;
max2=max(max(max(S2bar)));
S2bar=S2bar*(255/max2);

subplot(2,2,3), imshow(uint8(S1bar))
subplot(2,2,4), imshow(uint8(S2bar))
```

Note that the successive use of the **min/max** command is due to the fact that

the color images are $m \times n \times 3$ matrices where the 3 levels give the color mapping information.

Image separation from reflection

The algorithm above can be applied to the original problem presented of the **Judgement of Paris** by Flaxman. The original images are again illustrated in the top row of Fig. 69. The difference in the two figures was the application of a polarizer whose angle was rotated 90 degrees between shots. The bottom two figures shows the extraction process that occurs for the two images. Ultimately, the technique applied here did not do such a great job. This is largely due to the fact that the reflected image was so significantly brighter than the original photographed image itself, thus creating a difficult image separation problem. Certainly experience tells us that we also have this same difficulty: when it is very bright and sunny out, reflected images are much more difficult for us to see through to the true image.

6 Image Recognition Text

6.1 Recognizing dogs and cats

6.2 Wavelets: edge detection and fur pattern

6.3 Implementing cat/dog recognition in MATLAB

7 Equation Free Modeling

7.1 Multi-scale physics and phenomena

7.2 The role of PCA and POD in multi-scale modeling

7.3 Algorithm development for multi-scale dynamics

References

[1] M. R. Spiegel, *Probability and Statistics*, (Schaum's Outline Series, McGraw-Hill, 1975)

[2] S. Ross, *Introduction to Probability Models*, 4th Ed. (Academic Press, 1989).

[3] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes*, 2nd Ed. (Cambridge, 1992).

[4] L. Debonath, *Wavelet Transforms and Their Applications* (Birkhäuser, 2002).

Image Recognition — Machine Learning and Classification

Feature space — set of defining characteristics of the objects we are trying to classify.

The main idea is trying to find a feature space and a defining separation algorithm that allows us to reliably classify objects into one group or another.
In a properly chosen feature space, objects of the same classification ought to be closer together and/or form some recognizable pattern so the real question is how do we divide the feature space?

Supervised learning — I give you the classes to sort items into and the classifications for the set of data beforehand (i.e. a training set) and learn how to divide the dataset (and future data points) into given different classes
Feature Domain is contained in the feature space of our training set

Unsupervised learning — I just give you the data, no classifications. You decide the boundaries between classes AND how many classes there are.
Feature Domain is contained in all of possible feature space

Use PCA to get defining characteristics of our images of class one (i.e. cats) and see how other classes project onto it (i.e. images of dogs). Which principal component provides the most separation between the object classes = important defining feature (doesnt work so well if images are shifted, from different angles, different sizes, etc… i.e. really bad)

Considerations
Transform data before trying to classify — i.e. use a wavelet transform to pick out edges…

Chapter 5

Clustering and Classification

Machine learning is based upon optimization techniques for data. The goal is to find both a low-rank subspace for optimally embedding the data, as well as regression methods for clustering and classification of different data types. Machine learning thus provides a principled set of mathematical methods for extracting meaningful features from data, i.e. data mining, as well as binning the data into distinct and meaningful patterns that can be exploited for decision making. Specifically, it learns from and makes predictions based on data. For business applications, this is often called *predictive analytics*, and it is at the forefront of modern data-driven decision making. In an integrated system, such as is found in autonomous robotics, various machine learning components (e.g., for processing visual and tactile stimulus) can be integrated to form what we now call *artificial intelligence* (AI). To be explicit: AI is built upon integrated machine learning algorithms, which in turn are fundamentally rooted in optimization.

There are two broad categories for machine learning: *supervised machine learning* and *unsupervised machine learning*. In the former, the algorithm is presented with labelled datasets. The training data, as outlined in the cross-validation method of the last chapter, is labeled by a teacher/expert. Thus examples of the input and output of a desired model are explicitly given, and regression methods are used to find the best model for the given labeled data, via optimization. This model is then used for prediction and classification using new data. There are important variants of supervised methods, including *semi-supervised learning* in which incomplete training is given so that some of the input/output relationships are missing, i.e. for some input data, the actual output is missing. *Active learning* is another common subclass of supervised methods whereby the algorithm can only obtain training labels for a limited set of instances, based on a budget, and also has to optimize its choice of objects to acquire labels for. In an interactive framework, these can be presented to the user for labeling. Finally, in *reinforcement learning*, rewards or punishments are the training labels that help shape the regression architecture in order to build the best model. In

contrast, no labels are given for *unsupervised learning* algorithms. Thus, they must find patterns in the data in a principled way in order to determine how to cluster data and generate labels for predicting and classifying new data. In unsupervised learning, the goal itself may be to discover patterns in the data embedded in the low-rank subspaces so that *feature engineering* or *feature extraction* can be used to build an appropriate model.

In this chapter, we will consider some of the most commonly used supervised and unsupervised machine learning methods. As will be seen, our goal is to highlight how data mining can produce important data features (feature engineering) for later use in model building. We will also show that the machine learning methods can be broadly used for clustering and classification, as well as for building regression models for prediction. Critical to all of this machine learning architecture is finding low-rank feature spaces that are informative and interpretable.

5.1 Feature selection and data mining

To exploit data for diagnostics, prediction and control, dominant features of the data must be extracted. In the opening chapter of this book, SVD and PCA were introduced as methods for determining the dominant correlated structures contained within a data set. In the eigenfaces example of Sec. 1.6, for instance, the dominant features of a large number of cropped face images were shown. These eigenfaces, which are ordered by their ability to account for commonality (correlation) across the data base of faces was guaranteed to give the best set of r features for reconstructing a given face in an ℓ_2 sense with a rank-r truncation. The eigenface modes gave clear and interpretable features for identifying faces, including highlighting the eyes, nose and mouth regions as might be expected. Importantly, instead of working with the high-dimensional measurement space, the feature space allows one to consider a significantly reduced subspace where diagnostics can be performed.

The goal of data mining and machine learning is to construct and exploit the intrinsic low-rank feature space of a given data set. The feature space can be found in an unsupervised fashion by an algorithm, or it can be explicitly constructed by expert knowledge and/or correlations among the data. For eigenfaces, the features are the PCA modes generated by the SVD. Thus each PCA mode is high-dimensional, but the only quantity of importance in feature space is the weight of that particular mode in representing a given face. If one performs an r-rank truncation, then any face needs only r features to represent it in feature space. This ultimately gives a low-rank embedding of the data in an interpretable set of r features that can be leveraged for diagnostics, prediction, reconstruction and/or control.

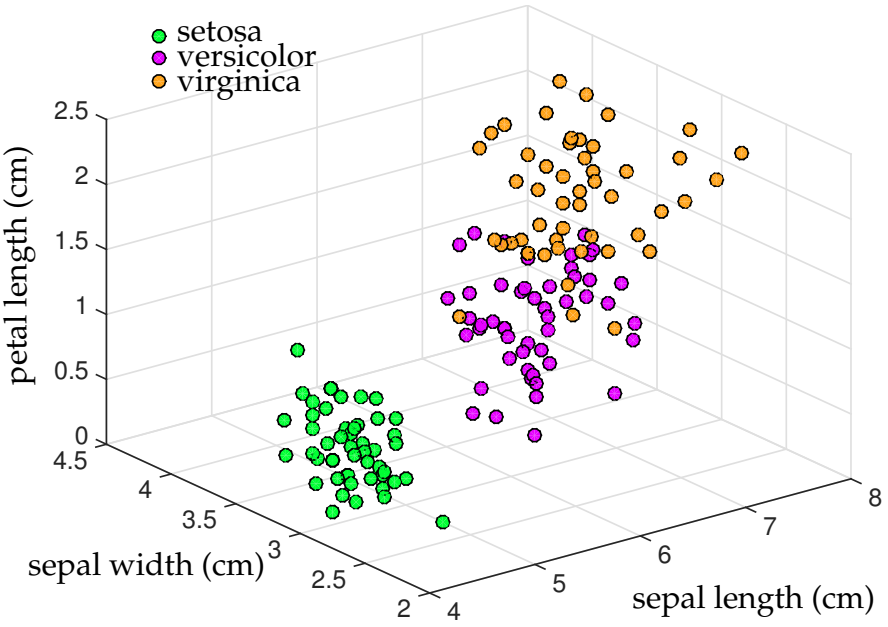


Figure 5.1: Fisher iris data set with 150 measurements over three varieties including 50 measurements each of setosa, versicolor and virginica. Each flower includes a measurement of sepal length, sepal width, petal length and petal width. The first three of these are illustrated here showing that these simple biological features are sufficient to show that the data has distinct, quantifiable differences between the species.

Several examples will be developed that illustrate how to generate a feature space, starting with a standard data set included with MATLAB. The Fisher iris data set includes measurements of 150 irises of three varieties: setosa, versicolor and virginica. The 50 samples of each flower include measurements in centimeters of the sepal length, sepal width, petal length and petal width. For this data set, the four features are already defined in terms of interpretable properties of the biology of the plants. For visualization purposes, Fig. 5.1 considers only the first three of these features. The following code accesses the Fisher iris data set:

Code 5.1: Features of the Fisher irises.

```
load fisheriris;
x1=meas(1:50,:);     % setosa
x2=meas(51:100,:);   % versicolor
x3=meas(101:150,:); % virginica

plot3(x1(:,1),x1(:,2),x1(:,4),'go'), hold on
plot3(x2(:,1),x2(:,2),x2(:,4),'mo')
plot3(x3(:,1),x3(:,2),x3(:,4),'ro')
```

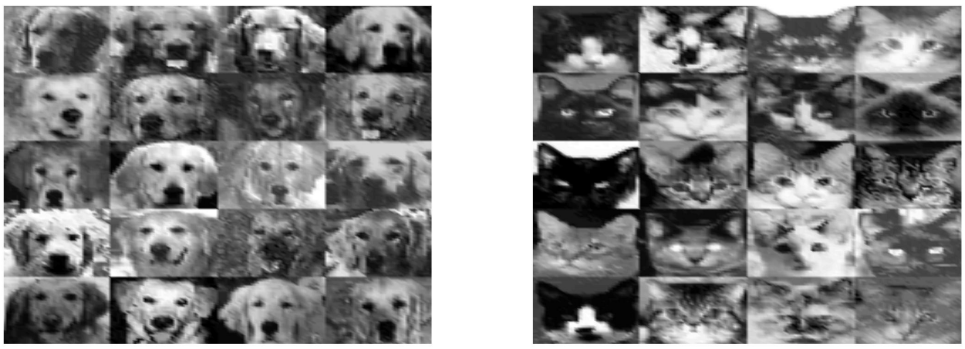


Figure 5.2: Example images of dogs (left) and cats (right). Our goal is to construct a feature space where automated classification of these images can be efficiently computed.

Figure 5.1 shows that the properties measured can be used as a good set of features for clustering and classification purposes. Specifically, the three iris varieties are well separated in this feature space. The setosa iris is most distinctive in its feature profile, while the versicolor and virginica have a small overlap among the samples taken. For this data set, machine learning is certainly not required to generate a good classification scheme. However, data generally does not so readily reduce down to simple two- and three-dimensional visual cues. Rather, decisions about clustering in feature space occur with many more variables, thus requiring the aid of computational methods to provide good classification schemes.

As a second example, we consider in Fig. 5.2 a selection from an image database of 80 dogs and 80 cats. A specific goal for this data set is to develop an automated classification method whereby the computer can distinguish between cats and dogs. In this case, the data for each cat and dog is the 64×64 pixel space of the image. Thus each image has 4096 measurements, in contrast to the 4 measurements for each example in the iris data set. Like eigenfaces, we will use the SVD to extract the dominant correlations among the images. The following code loads the data and performs a singular value decomposition on the data after the mean is subtracted. The SVD produces an ordered set of modes characterizing the correlation between all the dog and cat images. Figure 5.3 shows the first four SVD modes of the 160 images (80 dogs and 80 cats).

Code 5.2: Features of dogs and cats.

```
load dogData.mat
load catData.mat
CD=double([dog cat]);
[u,s,v]=svd(CD-mean(CD(:)),'econ');
```

The original image space, or pixel space, is only one potential set of data to work with. The data can be transformed into a wavelet representation where

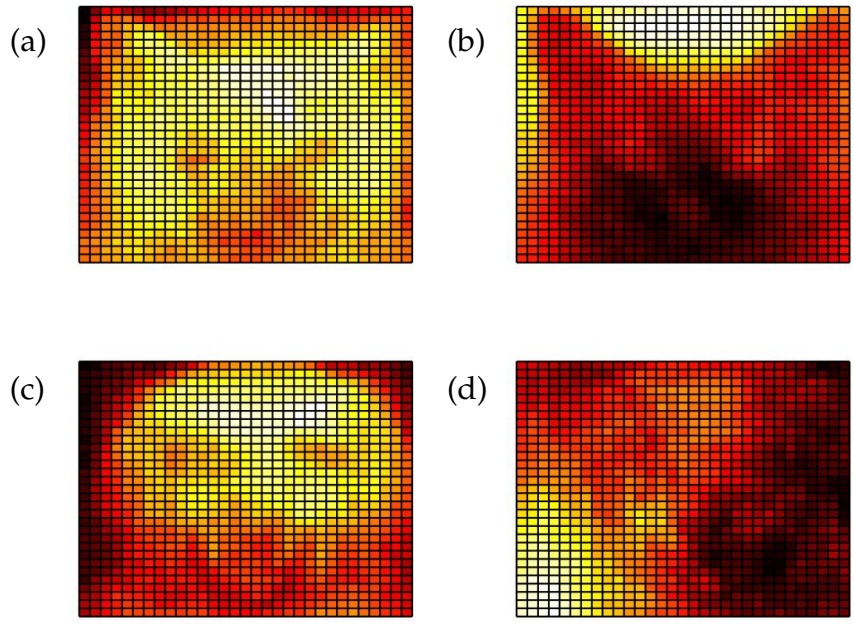


Figure 5.3: First four features (a)-(d) generated from the SVD of the 160 images of dogs and cats, i.e. these are the first four columns of the U matrix of the SVD. Typical cat and dog images are shown in Fig. 5.2. Note that the first two modes (a) and (b) show that the triangular ears are important features when images are correlated. This is certainly a distinguishing feature for cats, while dogs tend to lack this feature. Thus in feature space, cats generally add these two dominant modes to promote this feature while dogs tend to subtract these features to remove the triangular ears from their representation.

edges of the images are emphasized. The following code loads in the images in their wavelet representation and computes a new low-rank embedding space.

Code 5.3: Wavelet features of dogs and cats.

```
load catData_w.mat
load dogData_w.mat
CD2=[dog_wave cat_wave];
[u2,s2,v2]=svd(CD2-mean(CD2(:)),'econ');
```

The equivalent of Fig. 5.3 in wavelet space is shown in Fig. 5.4. Note that the wavelet representation helps emphasize many key features such as the eyes, nose, and ears, potentially making it easier to make a classification decision. Generating a feature space that enables classification is critical for constructing effective machine learning algorithms.

Whether using the image space directly or a wavelet representation, Figs. 5.3 and 5.4 respectively, the goal is to project the data onto the feature space generated by each. A good feature space helps find distinguishing features that

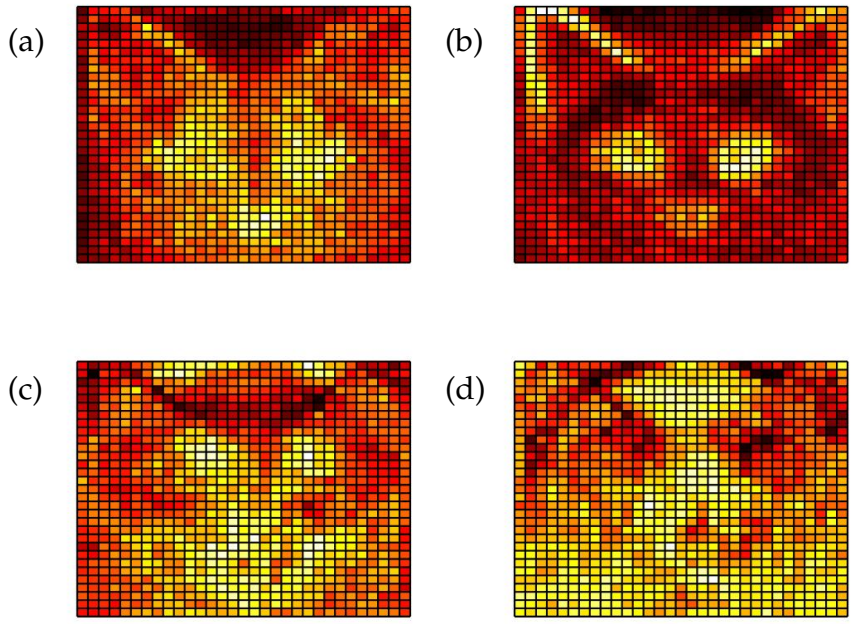


Figure 5.4: First four features (a)-(d) generated from the SVD of the 160 images of dogs and cats in the wavelet domain. As before, the first two modes (a) and (b) show that the triangular ears are important. This is an alternative representation of the dogs and cats that can help better classify dogs versus cats.

i.e. V gives the coordinates of each of our data measurements (row of data matrix X) in our new coordinate system given by the columns of U (i.e. our principal components). Our goal is to find a component that has very different weightings depending on if we have a dog or a cat (i.e. ideal would be the weighting is always negative for a cat and positive for a dog or vice versa, then this one feature would perfectly classify dog vs cat)

allow one to perform a variety of tasks that may include clustering, classification, and prediction. The importance of each feature to an individual image is given by the $\mathbf{V}$ matrix in the SVD. Specifically, each column of $\mathbf{V}$ determines the loading, or weighting, of each feature onto a specific image. Histograms of these loadings can then be used to visualize how distinguishable cats and dogs are from each other by each feature (See Fig. 5.5). The following code produces a histogram of the distribution of loadings for the dogs and the cats (first 80 images versus second 80 images respectively).

Code 5.4: Feature histograms of dogs and cats.

```
xbin=linspace(-0.25,0.25,20);
for j=1:4
    subplot(4,2,2*j-1)
    pdf1=hist(v(1:80,j),xbin)
    pdf2=hist(v(81:160,j),xbin)
    plot(xbin,pdf1,xbin,pdf2,'Linewidth',[2])
end
```

Figure 5.5 shows the distribution of loading scores for the first four modes for both the raw images as well as the wavelet transformed images. For both the sets of images, the distribution of loadings on the second mode clearly shows a strong separability between dogs and cats. The wavelet processed

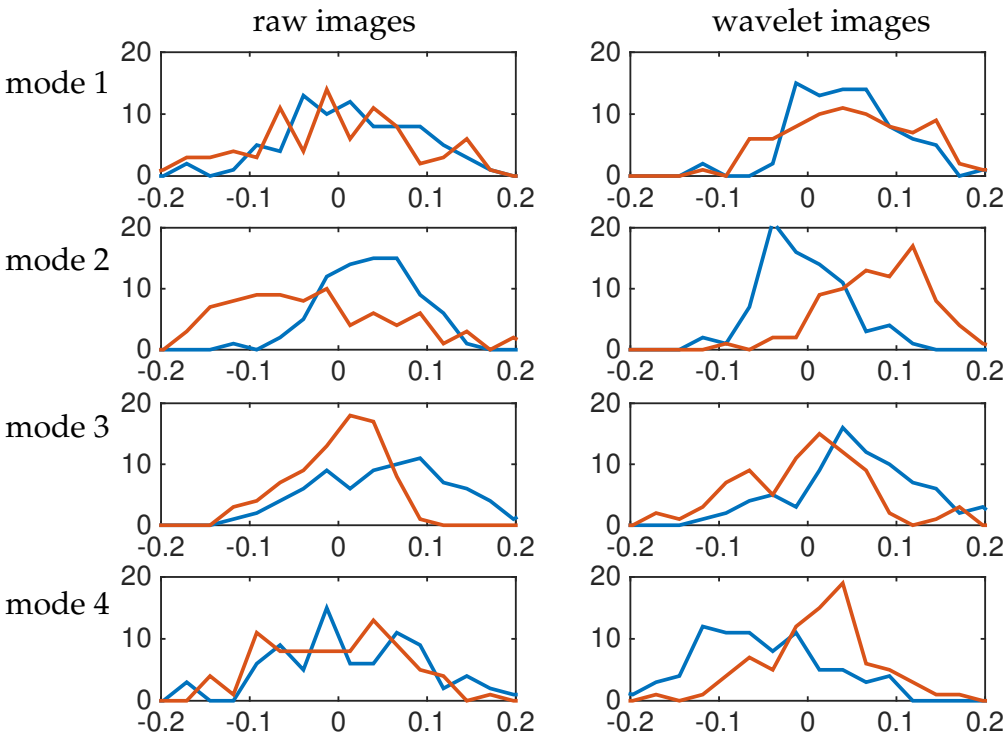


Figure 5.5: Histogram of the distribution of loadings for dogs (blue) and cats (red) on the first four dominant SVD modes. The left panel shows the distributions for the raw images (See Fig. 5.3) while the right panels show the distribution for wavelet transformed data (See Fig. 5.4. The loadings come from the columns of the $\mathbf{V}$ matrix of the SVD. Note the good separability between dogs and cats using the second mode.

images also show a nice separability on the fourth mode. Note that the first mode for both shows very little discrimination between the distributions and is thus not useful for classification and clustering objectives.

Features that provide strong separability between different types of data (e.g. dogs and cats) are typically exploited for machine learning tasks. This simple example shows that feature engineering is a process whereby an initial data exploration is used to help identify potential pre-processing methods. These features can then help the computer identify highly distinguishable features in a higher-dimensional space for accurate clustering, classification and prediction. As a final note, consider Fig. 5.6 which projects the dog and cat data onto the first three PCA modes (SVD modes) discovered from the raw images or their wavelet transformed counterparts. As will be seen later, the wavelet transformed images provide a higher degree of separability, and thus improved classification.

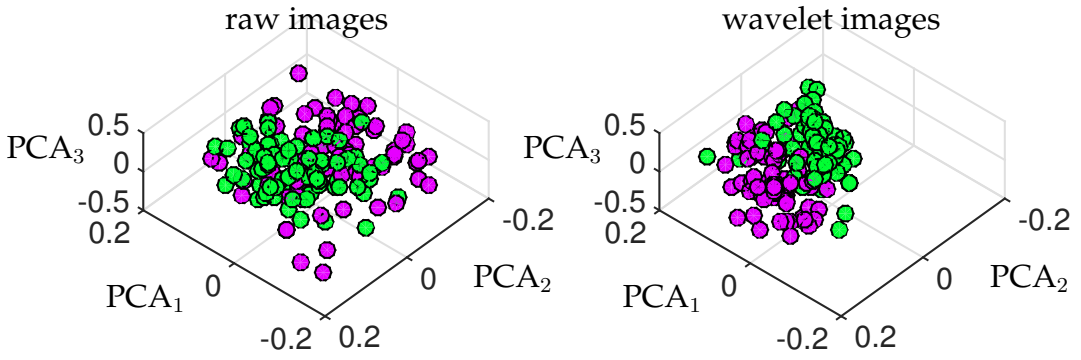


Figure 5.6: Projection of dogs (green) and cats (magenta) into feature space. Note that the raw images and their wavelet counterparts produce different embeddings of the data. Both exhibit clustering around their labeled states of dog and cat. This is exploited in the learning algorithms that follow. The wavelet images are especially good for clustering and classification as this feature space more easily separates the data.

5.2 Supervised versus unsupervised learning

As previously stated, the goal of data mining and machine learning is to construct and exploit the intrinsic low-rank feature space of a given data set. Good feature engineering and feature extraction algorithms can then be used to learn classifiers and predictors for the data. Two dominant paradigms exist for learning from data: *supervised methods* and *unsupervised methods*. Supervised data-mining algorithms are presented with labeled data sets, where the training data is labeled by a teacher/expert/supervisor. Thus examples of the input and output of a desired model are explicitly given, and regression methods are used to find the best model via optimization for the given labeled data. This model is then used for prediction and classification using new data. There are important variants of this basic architecture which include semi-supervised learning, active learning and reinforcement learning. For unsupervised learning algorithms, no training labels are given so that an algorithm must find patterns in the data in a principled way in order to determine how to cluster and classify new data. In unsupervised learning, the goal itself may be to discover patterns in the data embedded in the low-rank subspaces so that feature engineering or feature extraction can be used to build an appropriate model.

To illustrate the difference in supervised versus unsupervised learning, consider Fig. 5.7. This shows a scatter plot of two Gaussian distributions. In one case, the data is well separated so that their means are sufficiently far apart and two distinct clusters are observed. In the second case, the two distributions are brought close together so that separating the data is a challenging task. The goal of unsupervised learning is to discover clusters in the data. This is a trivial task by visual inspection, provided the two distributions are sufficiently sepa-

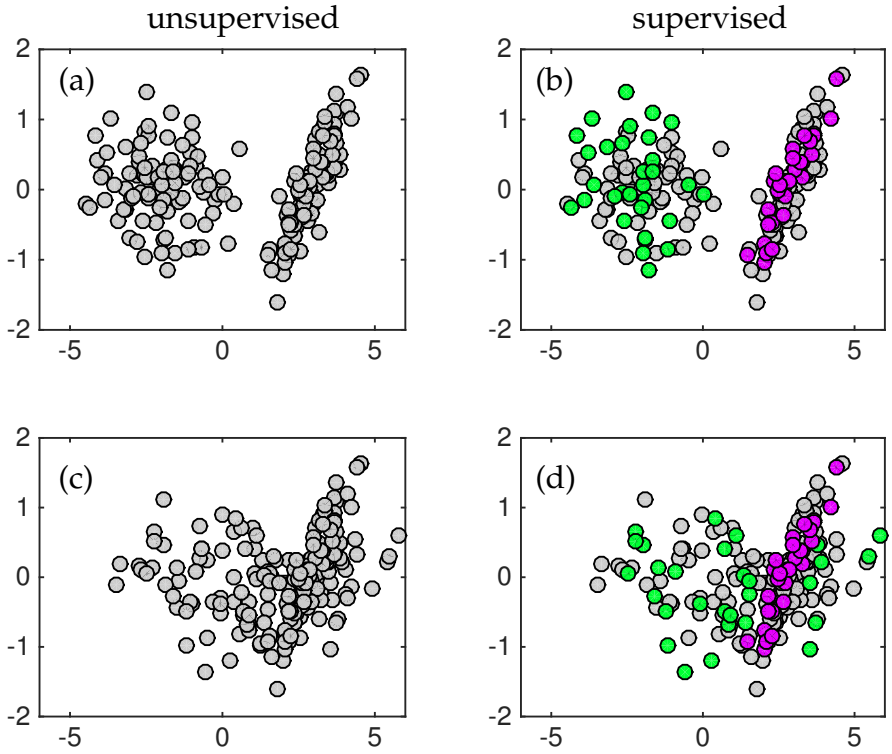


Figure 5.7: Illustration of unsupervised versus supervised learning. In the left panels (a) and (c), unsupervised learning attempts to find clusters for the data in order to classify them into two groups. For well separated data (a), the task is straightforward and labels can easily be produced. For overlapping data (c), it is a very difficult task for an unsupervised algorithm to accomplish. In the right panels (b) and (d), supervised learning provides a number of labels: green balls and magenta balls. The remaining unlabeled data is then classified as green or magenta. For well separated data (b), labeling data is easy, while overlapping data presents significant challenge.

rated. Otherwise, it becomes very difficult to distinguish clusters in the data. Supervised learning provides labels for some of the data. In this case, points are either labeled with green dots or magenta dots and the task is to classify the unlabeled data (grey dots) as either green or magenta. Much like the unsupervised architecture, if the statistical distributions that produced the data are well separated, then using the labels in combination with the data provides a simple way to classify all the unlabeled data points. Supervised algorithms also perform poorly if the data distributions have significant overlap.

Supervised and unsupervised learning can be stated mathematically. Let

$$\mathcal{D} \subset \mathbb{R}^n \tag{5.1}$$

so that $\mathcal{D}$ is an open bounded set of dimension n. Further, let

$$\mathcal{D}' \subset \mathcal{D}\,. \tag{5.2}$$

The goal of classification is to build a classifier labeling all data in $\mathcal{D}$ given data from $\mathcal{D}'$.

To make our problem statement more precise, consider a set of data points $\mathbf{x}_j \in \mathbb{R}^n$ and labels $\mathbf{y}_j$ for each point where $j = 1, 2, \cdots, m$. Labels for the data can come in many forms, from numeric values, including integer labels, to text strings. For simplicity, we will label the data in a binary way as either plus or minus one so that $\mathbf{y}_j \in \{\pm 1\}$.

For unsupervised learning, the following inputs and outputs are then associated with learning a classification task

Input

$$\text{data} \quad \{\mathbf{x}_j \in \mathbb{R}^n,\ j \in Z := \{1, 2, \cdots, m\}\} \tag{5.3a}$$

Output

$$\text{labels} \quad \{\mathbf{y}_j \in \{\pm 1\},\ j \in Z\}\,. \tag{5.3b}$$

Thus the mathematical framing of unsupervised learning is focused on producing labels $\mathbf{y}_j$ for all the data. Generally, the data $\mathbf{x}_j$ used for training the classifier is from $\mathcal{D}'$. The classifier is then more broadly applied, i.e. it generalizes, to the open bounded domain $\mathcal{D}$. If the data used to build a classifier only samples a small portion of the larger domain, then it is often the case that the classifier will not generalize well.

Supervised learning provides labels for the training stage. The inputs and outputs for this learning classification task can be stated as follows

Input

$$\text{data} \quad \{\mathbf{x}_j \in \mathbb{R}^n,\ j \in Z := \{1, 2, \cdots, m\}\} \tag{5.4a}$$

$$\text{labels} \quad \{\mathbf{y}_j \in \{\pm 1\},\ j \in Z' \subset Z\} \tag{5.4b}$$

Output

$$\text{labels} \quad \{\mathbf{y}_j \in \{\pm 1\},\ j \in Z\}\,. \tag{5.4c}$$

In this case, a subset of the data is labeled and the missing labels are provided for the remaining data. Technically speaking, this is a semi-supervised learning task since some of the training labels are missing. For supervised learning, all the labels are known in order to build the classifier on $\mathcal{D}'$. The classifier is then applied to $\mathcal{D}$. As with unsupervised learning, if the data used to build a classifier only samples a small portion of the larger domain, then it is often the case that the classifier will not generalize well.

For the data sets considered in our feature selection and data mining section, we can consider in more detail the key components required to build a

classification model: $\mathbf{x}_j$, $\mathbf{y}_j$, $\mathcal{D}$ and $\mathcal{D}'$. The Fisher iris data of Fig. 5.1 is a classic example for which we can detail these quantities. We begin with the data collected

$$\mathbf{x}_j = \{\text{sepal length, sepal width, petal length, petal width}\}\,. \tag{5.5}$$

Thus each iris measurement contains four data fields, or features, for our analysis. The labels can be one of the following

$$\mathbf{y}_j = \{\text{setosa, versicolor, virginica}\}\,. \tag{5.6}$$

In this case the labels are text strings, and there are three of them. Note that in our formulation of supervised and unsupervised learning, there were only two outputs (binary) which were labeled either ± 1. Generally, there can be many labels, and they are often text strings. Finally, there is the domain of the data. For this case

$$\mathcal{D}' \in \{\text{150 iris samples: 50 setosa, 50 versicolor, and 50 virginica}\} \tag{5.7}$$

and

$$\mathcal{D} \in \{\text{the universe of setosa, versicolor and virginica irises}\}. \tag{5.8}$$

We can similarly assess the dog and cat data as follows:

$$\mathbf{x}_j = \{64\times 64 \text{ image= 4096 pixels}\} \tag{5.9}$$

where each dog and cat is labeled as

$$\mathbf{y}_j = \{\text{dog, cat}\} = \{1, \text{-1}\}\,. \tag{5.10}$$

In this case the labels are text strings which can also be translated to numeric values. This is consistent with our formulation of supervised and unsupervised learning where there are only two outputs (binary) labeled either ± 1. Finally, there is the domain of the data which is

$$\mathcal{D}' \in \{\text{160 image samples: 80 dogs and 80 cats}\} \tag{5.11}$$

and

$$\mathcal{D} \in \{\text{the universe of dogs and cats}\}. \tag{5.12}$$

Supervised and unsupervised learning methods aim to either create algorithms for classification, clustering, or regression. The discussion above is a general strategy for classification. The previous chapter discusses regression architectures. For both tasks, the goal is to build a model from data on $\mathcal{D}'$ that can generalize to $\mathcal{D}$. As already shown in the preceding chapter on regression, generalization can be very difficult and cross-validation strategies are critical.

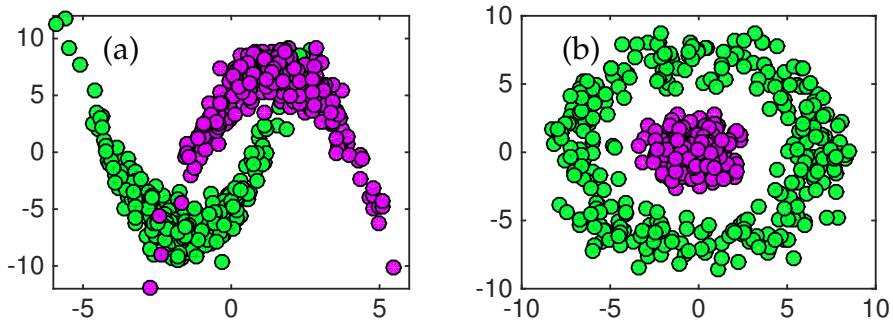


Figure 5.8: Classification and regression models for data can be difficult when the data have nonlinear functions which separate them. In this case, the function separating the green and magenta balls can be difficult to extract. Moreover, if only a small sample of the data $\mathcal{D}'$ is available, then a generalizable model may be impossible to construct for $\mathcal{D}$. The left data set (a) represents two half-moon shapes that are just superimposed while the concentric rings in (b) requires a circle as a separation boundary between the data. Both are challenging to produce.

Deep neural networks, which are state-of-the-art machine learning algorithms for regression and classification, often have difficulty generalizing. Creating strong generalization schemes is at the forefront of machine learning research.

Some of the difficulties in generalization can be illustrated in Fig. 5.8. These data sets, although easily classified and clustered through visual inspection can be difficulty for many regression and classification schemes. Essentially, the boundary between the data forms a nonlinear manifold that is often difficult to characterize. Moreover, if the sampling data $\mathcal{D}'$ only captures a portion of the manifold, then a classification or regression model will almost surely fail in characterizing $\mathcal{D}$. These are also only two-dimensional depictions of a classification problem. It is not difficult to imagine how complicated such data embeddings can be in higher dimensional space. Visualization in such cases is essentially impossible and one must rely on algorithms to extract the meaningful boundaries separating data. What follows in this chapter and the next are methods for classification and regression given data on $\mathcal{D}'$ that may or may not be labelled. There is quite a diversity of mathematical methods available for performing such tasks.

5.3 Unsupervised learning: k-means clustering

A variety of supervised and unsupervised algorithms will be highlighted in this chapter. We will start with one of the most prominent unsupervised algorithms in use today: k-means clustering. The k-means algorithm assumes one is given a set of vector valued data with the goal of partitioning m observations

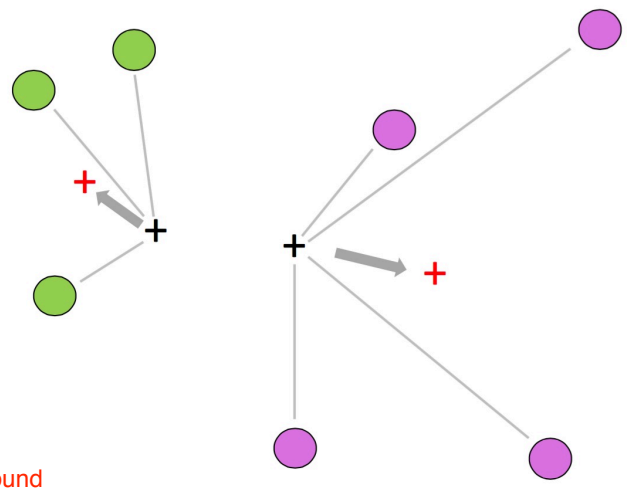
Take a bunch of data
pick k random points and
group based on which of these two points all other points are closest to

Find the mean/centroid of each cluster (NOT necessarily a data point)

Repeat process; regroup based on new points/centroids from last iteration

Continue until centroids converge
(do not change after a given iteration)

MATLAB: [labels, c] = Kmeans(data, k)
N data points
with k clusters and c are the dividing lines found
and labels is an N x 1 vector:
label/group for each data poiint



Issues with K means

How many clusters? This is unsupervised learning so going in we dont really know what our data is like

How do you pick the start points? Can effect our results.
Cross-validation: look for consistent results over repeated trials

High Performfance, cross validation
What happens if I choose more or less means? Lose performance or robustness trial to trial. A good solution performs well across many starting points

Figure 5.9: Illustration of the k-means algorithm for $k = 2$. Two initial starting values of the man are given (black $+$). Each point is labeled as belonging to one of the two means. The green balls are thus labeled as part of the cluster with the left $+$ and the magenta balls are labeled as part of the right $+$. Once labeled, the mean of the two clusters is recomputed (red $+$). The process is repeated until the means converge.

into k clusters. Each observation is labeled as belonging to a cluster with the nearest mean, which serves as a proxy (prototype) for that cluster. This results in a partitioning of the data space into Voronoi cells.

Although the number of observations and dimension of the system are known, the number of partitions k is generally unknown and must also be determined. Alternatively, the user simply chooses a number of clusters to extract from the data. The k-means algorithm is iterative, first assuming initial values for the mean of each cluster and then updating the means until the algorithm has converged. Figure 5.9 depicts the update rule of the k-means algorithm. The algorithm proceeds as follows: (i) given initial values for k distinct means, compute the distance of each observation $\mathbf{x}_j$ to each of the k means. (ii) Label each observation as belonging to the nearest mean. (iii) Once labeling is completed, find the *center-of-mass* (mean) for each group of labeled points. These new means are then used to start back at step (i) in the algorithm. This is a heuristic algorithm that was first proposed by Stuart Lloyd in 1957 [339], although it was not published until 1982.

The k-means objective can be stated formally in terms of an optimization problem. Specifically, the following minimization describes this process

$$\underset{\boldsymbol{\mu}_j}{\operatorname{argmin}} \sum_{j=1}^{k} \sum_{\mathbf{x}_j \in \mathcal{D}'_j} \|\mathbf{x}_j - \boldsymbol{\mu}_j\|^2 \tag{5.13}$$

where the $\boldsymbol{\mu}_j$ denote the mean of the jth cluster and $\mathcal{D}'_j$ denotes the subdomain of data associated with that cluster. This minimizes the within-cluster sum of

squares. In general, solving the optimization problem as stated is NP-hard, making it computationally intractable. However, there a number of heuristic algorithms that provide good performance despite not having a guarantee that they will converge to the globally optimal solution.

Cross-validation of the k-means algorithm, as well as any machine learning algorithm, is critical for determining its effectiveness. Without labels the cross validation procedure is more nuanced as there is no ground truth to compare with. The cross-validation methods of the last section, however, can still be used to test the robustness of the classifier to different sub-selections of the data through k-fold cross-validation. The following portions of code generate Lloyd's algorithm for k-means clustering. We first consider making two clusters of data and partitioning the data into a training and test set.

Code 5.5: k-means data generation.

```
% training & testing set sizes
n1=100;  % training set size
n2=50;   % test set size

% random ellipse 1 centered at (0,0)
x=randn(n1+n2,1); y=0.5*randn(n1+n2,1);

% random ellipse 2 centered at (1,-2) and rotated by theta
x2=randn(n1+n2,1)+1; y2=0.2*randn(n1+n2,1)-2; theta=pi/4;
A=[cos(theta) -sin(theta); sin(theta) cos(theta)];
x3=A(1,1)*x2+A(1,2)*y2; y3=A(2,1)*x2+A(2,2)*y2;
subplot(2,2,1)
plot(x(1:n1),y(1:n1),'ro'), hold on
plot(x3(1:n1),y3(1:n1),'bo')

% training set:  first 200 of 240 points
X1=[x3(1:n1) y3(1:n1)];
X2=[x(1:n1) y(1:n1)];

Y=[X1; X2]; Z=[ones(n1,1); 2*ones(n1,1)];

% test set:  remaining 40 points
x1test=[x3(n1+1:end) y3(n1+1:end)];
x2test=[x(n1+1:end) y(n1+1:end)];
```

Figure 5.11 shows the data generated from two distinct Gaussian distributions. In this case, we have ground truth data to check the k-means clustering against. In general, this is not the case. The Lloyd algorithm guesses the number of clusters and the initial cluster means and then proceeds to update them in an iterative fashion. k-means is sensitive to the initial guess and many modern versions of the algorithm also provide principled strategies for initialization.

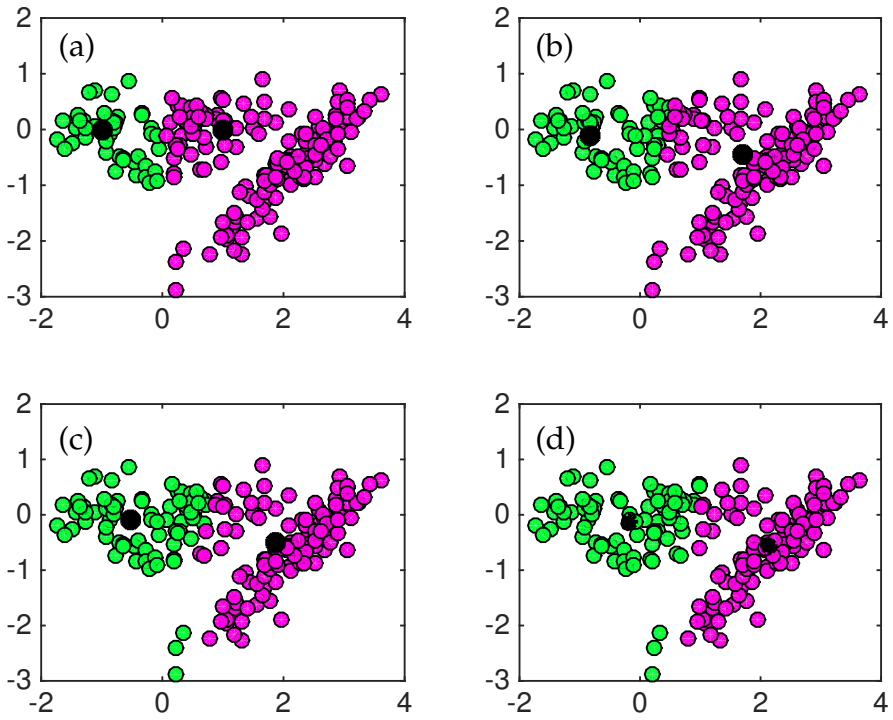


Figure 5.10: Illustration of the k-means iteration procedure based upon Lloyd's algorithm [339]. Two clusters are sought so that $k = 2$. The initial guesses (black circles in panel (a)) are used to initially label all the data according to their distance from each initial guess for the mean. The means are then updated by computing the means of the newly labeled data. This two-stage heuristic converges after approximately four iterations.

Code 5.6: Lloyd algorithm for k-means.

```
g1=[-1 0]; g2=[1 0]; %  Initial guess
for j=1:4
    class1=[]; class2=[];
    for jj=1:length(Y)
        d1=norm(g1-Y(jj,:));
        d2=norm(g2-Y(jj,:));
        if d1<d2
          class1=[class1; [Y(jj,1) Y(jj,2)]];
        else
          class2=[class2; [Y(jj,1) Y(jj,2)]];
        end
    end
    g1=[mean(class1(1:end,1)) mean(class1(1:end,2))];
    g2=[mean(class2(1:end,1)) mean(class2(1:end,2))];
end
```

Figure 5.10 shows the iterative procedure of the k-means clustering. The

two initial guesses are used to initially label all the data points (Fig. 5.10(a)). New means are computed and the data relabeled. After only four iterations, the clusters converge. This algorithm was explicitly developed here to show how the iteration procedure rapidly provides an unsupervised labeling of all of the data. MATLAB has a built in k-means algorithm that only requires a data matrix and the number of clusters desired. It is simple to use and provides a valuable diagnostic tool for data. The following code uses the MATLAB command **mean** and also extracts the *decision line* generated from the algorithm separating the two clusters.

Code 5.7: k-means using MATLAB.

```
% kmeans code
[ind,c]=kmeans(Y,2);
plot(c(1,1),c(1,2),'k*','Linewidth',[2])
plot(c(2,1),c(2,2),'k*','Linewidth',[2])

midx=(c(1,1)+c(2,1))/2; midy=(c(1,2)+c(2,2))/2;
slope=(c(2,2)-c(1,2))/(c(2,1)-c(1,1)); % rise/run
b=midy+(1/slope)*midx;
xsep=-1:0.1:2; ysep=-(1/slope)*xsep+b;

figure(1), subplot(2,2,1), hold on
plot(xsep,ysep,'k','Linewidth',[2]),axis([-2 4 -3 2])

% error on test data
figure(1), subplot(2,2,2)
plot(x(n1+1:end),y(n1+1:end),'ro'), hold on
plot(x3(n1+1:end),y3(n1+1:end),'bo')
plot(xsep,ysep,'k','Linewidth',[2]), axis([-2 4 -3 2])
```

Figure 5.11 shows the results of the k-means algorithm and depicts the decision line separating the data into two clusters. The green and magenta balls denote the true labels of the data, showing that the k-means line does not correctly extract the labels. Indeed, a supervised algorithm is more proficient in extracting the ground truth results, as will be shown later in this chapter. Regardless, the algorithm does get a majority of the data labeled correctly.

The success of k-means is based on two factors: (i) no supervision is required, and (ii) it is a fast heuristic algorithm. The example here shows that the method is not very accurate, but this is often the case in unsupervised methods as the algorithm has limited knowledge of the data. Cross-validation efforts, such as k-fold cross-validation, can help improve the model and make the unsupervised learning more accurate, but it will generally be less accurate than a supervised algorithm that has labeled data.

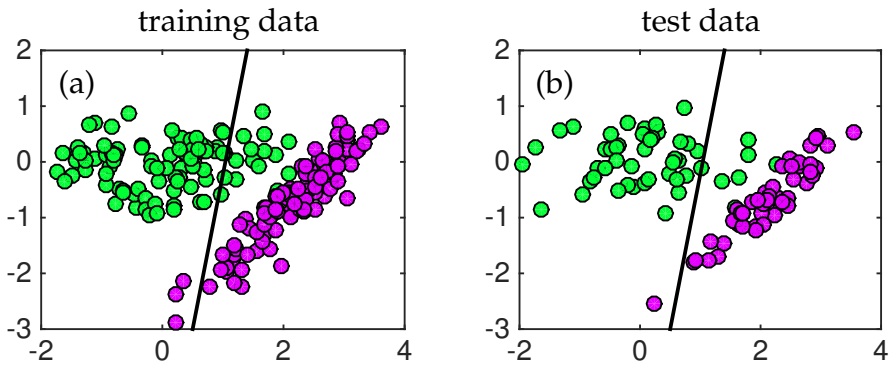


Figure 5.11: k-means clustering of the data using MATLAB's **means** command. Only the data and number of clusters need be specified. (a) The training data is used to produce a decision line (black line) separating the clusters. Note that the line is clearly not optimal. The classification line can then be used on withheld data to test the accuracy of the algorithm. For the test data, one (of 50) magenta ball would be mislabeled while six (of 50) green balls are mislabeled.

Start each data point as its own cluster

Join the closest pair of points, replacing them with a point at their mean.

Repeat until only a single point remains!

5.4 Unsupervised hierarchical clustering: Dendrogram

Another commonly used unsupervised algorithm for clustering data is a *dendrogram*. Like k-means clustering, dendrograms are created from a simple hierarchical algorithm, allowing one to efficiently visualize if data is clustered without any labeling or supervision. This hierarchical approach will be applied to the data illustrated in Fig. 5.12 where a ground truth is known. Hierarchical clustering methods are generated either from a top-down or a bottom-up approach. Specifically, they are one of two types:

Agglomerative: Each data point $\mathbf{x}_j$ is its own cluster initially. The data is merged in pairs as one creates a hierarchy of clusters. The merging of data eventually stops once all the data has been merged into a single über cluster. This is the bottom-up approach in hierarchical clustering.

Divisive: In this case, all the observations $\mathbf{x}_j$ are initially part of a single giant cluster. The data is then recursively split into smaller and smaller clusters. The splitting continues until the algorithm stops according to a user specified objective. The divisive method can split the data until each data point is its own node.

In general, the merging and splitting of data is accomplished with a heuristic, greedy algorithm which is easy to execute computationally. The results of hierarchical clustering are usually presented in a dendrogram.

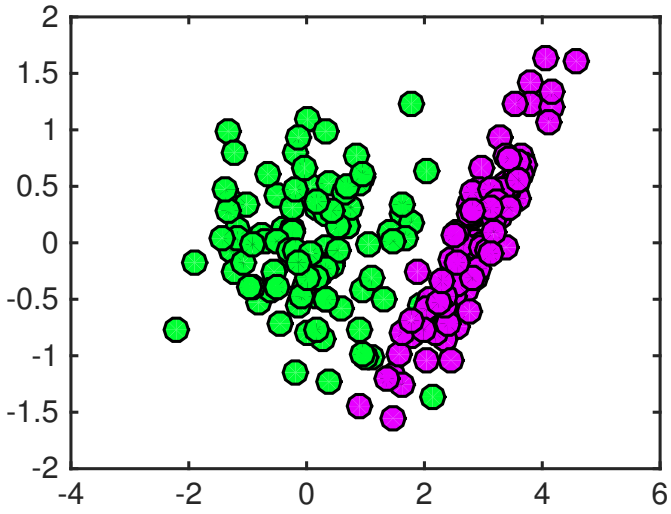


Figure 5.12: Example data used for construction of a dendrogram. The data is constructed from two Gaussian distributions (50 points each) that are easy to discern through a visual inspection. The dendrogram will produce a hierarchy that ideally would separate green balls from magenta balls.

In this section, we will focus on agglomerative hierarchical clustering and the dendrogram command from MATLAB. Like the Lloyd algorithm for k-means clustering, building the dendrogram proceeds from a simple algorithmic structure based on computing the distance between data points. Although we typically use a Euclidean distance, there are a number of important distance metrics one might consider for different types of data. Some typical distances are given as follows:

$$\text{Euclidean distance } \|\mathbf{x}_j - \mathbf{x}_k\|_2 \tag{5.14a}$$

$$\text{Squared Euclidean distance } \|\mathbf{x}_j - \mathbf{x}_k\|_2^2 \tag{5.14b}$$

$$\text{Manhattan distance } \|\mathbf{x}_j - \mathbf{x}_k\|_1 \tag{5.14c}$$

$$\text{Maximum distance } \|\mathbf{x}_j - \mathbf{x}_k\|_\infty \tag{5.14d}$$

$$\text{Mahalanobis distance } \sqrt{(\mathbf{x}_j - \mathbf{x}_k)^T \mathbf{C}^{-1} (\mathbf{x}_j - \mathbf{x}_k)} \tag{5.14e}$$

where $\mathbf{C}^{-1}$ is the covariance matrix. As already illustrated in the previous chapter, the choice of norm can make a tremendous difference for exposing patterns in the data that can be exploited for clustering and classification.

The dendrogram algorithm is shown in Fig. 5.13. The algorithm is as follows: (i) the distance between all m data points $\mathbf{x}_j$ is computed (the figure illustrates the use of a Euclidian distance), (ii) the closest two data points are merged into a single new data point midway between their original locations, and (iii) repeat the calculation with the new $m-1$ points. The algorithm continues until the data has been hierarchically merged into a single data point.

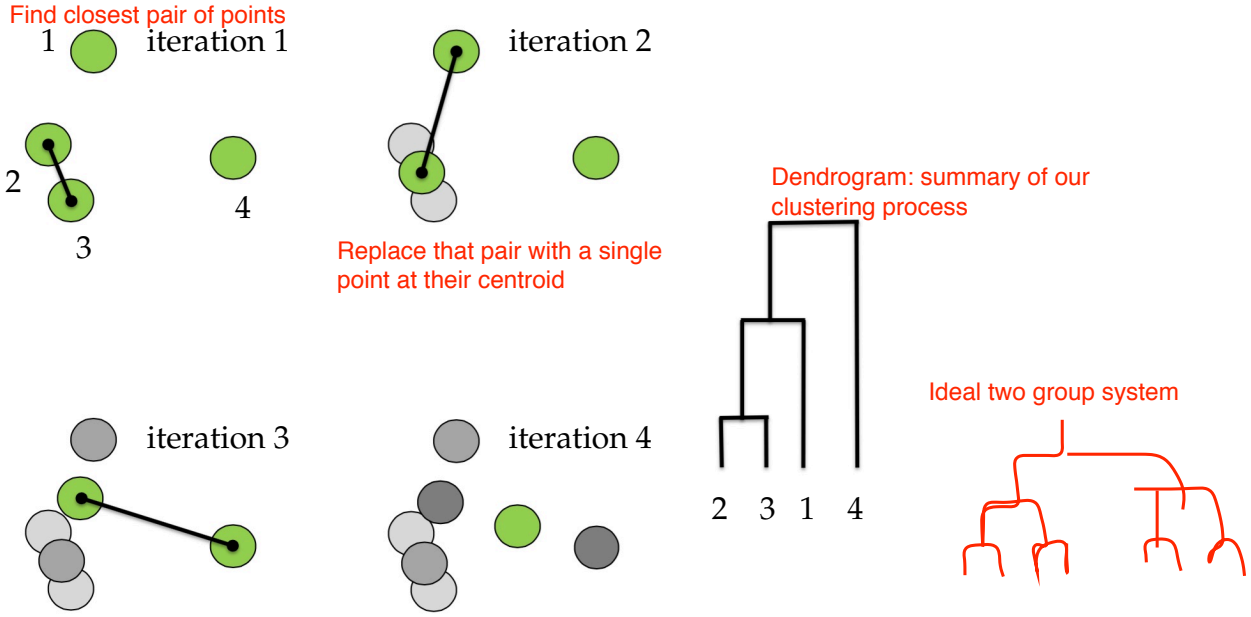


Figure 5.13: Illustration of the agglomerative hierarchical clustering scheme applied to four data points. In the algorithm, the distance between the four data points is computed. Initially the Euclidian distance between points 2 and 3 is closest. Points 2 and 3 are now merged into a point mid-way between them and the distances are once again computed. The dendrogram on the right shows how the process generates a summary (dendrogram) of the hierarchical clustering. Note that the length of the branches of the dendrogram tree are directly related to the distance between the merged points.

The following code performs a hierarchical clustering using the **dendrogram** command from MATLAB. The example we use is the same as that considered for k-means clustering. Figure 5.12 shows the data under consideration. Visual inspection shows two clear clusters that are easily discernible. As with k-means, our goal is to see how well a dendrogram can extract the two clusters.

Code 5.8: Dendrogram for unsupervised clustering.

```
Y3=[X1(1:50,:); X2(1:50,:)];
Y2 = pdist(Y3,'euclidean'); find the probability distribution based on euclidean distance?
Z = linkage(Y2,'average');
thresh=0.85*max(Z(:,3));
[H,T,O]=dendrogram(Z,100,'ColorThreshold',thresh);
```

Figure 5.14 shows the dendrogram associated with the data in Fig. 5.12. The structure of the algorithm shows which points are merged as well as the distance between points. The threshold command is important in labeling where each point belongs in the hierarchical scheme. By setting the threshold at different levels, there can be more or fewer clusters in the dendrogram. The following code uses the output of the dendrogram to show how the data was labeled. Recall that the first 50 data points are from the green cluster and the second 50

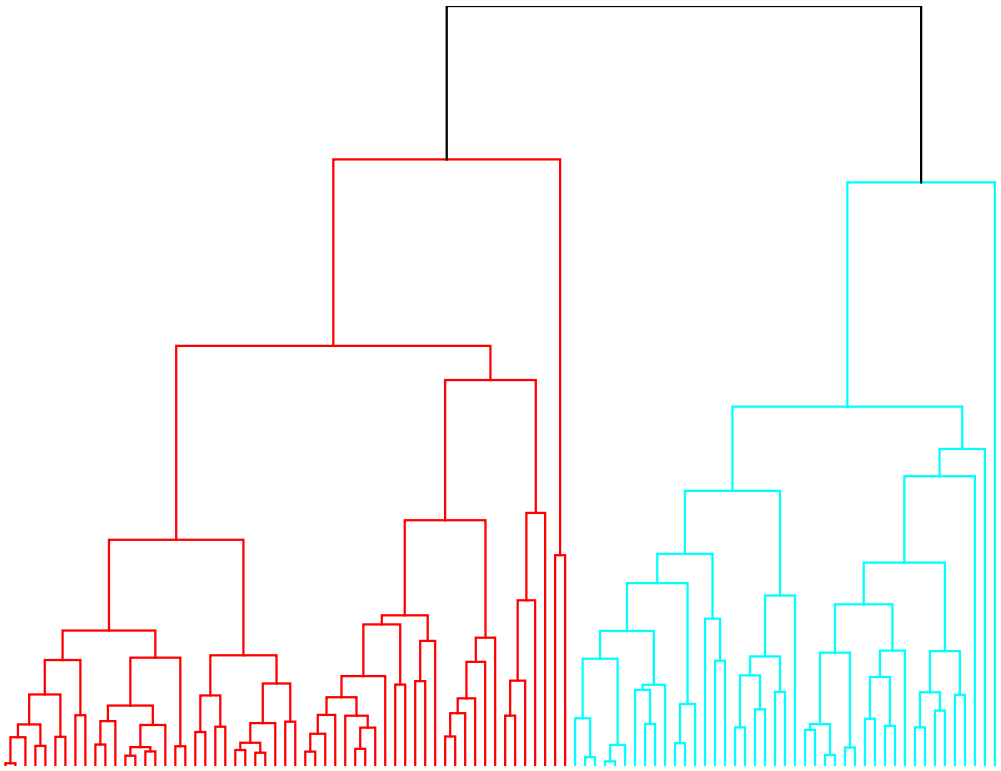


Figure 5.14: Dendrogram structure produced from the data in Fig. 5.12. The dendrogram shows which points are merged as well as the distance between points. Two clusters are generated for this level of threshold.

data points are from the magenta cluster.

Code 5.9: Dendrogram labels for cats and dogs.

```
bar(O), hold on
plot([0 100],[50 50],'r:','Linewidth',2)
plot([50.5 50.5],[0 100],'r:','Linewidth',2)
```

Figure 5.15 shows how the data was clustered in the dendrogram. If perfect clustering had been achieved, then the first 50 points would have been below the horizontal dotted red line while the second 50 points would have been above the horizontal dotted red line. The vertical dotted red line is the line separating the green dots on the left from the magenta dots on the right.

The following code shows how a greater number of clusters are generated by adjusting the threshold in the **dendrogram** command. This is equivalent to setting the number of clusters in k-means to something greater than two. Recall that one rarely has a ground truth to compare with when doing unsupervised clustering, so tuning the threshold becomes important.

```
thresh=0.25*max(Z(:,3));
[H,T,O]=dendrogram(Z,100,'ColorThreshold',thresh);
```

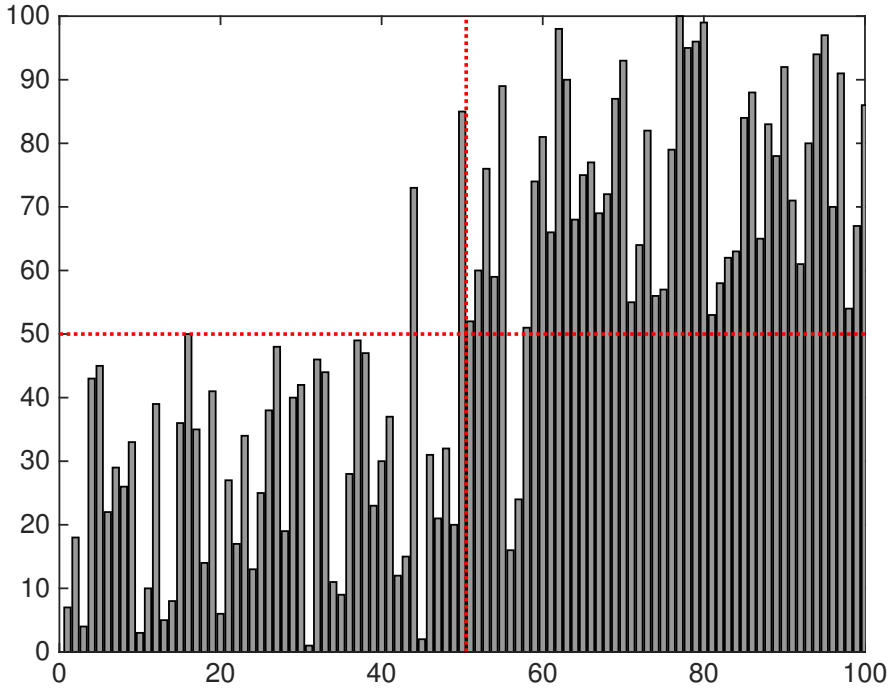


Figure 5.15: Clustering outcome from dendrogram routine. This is a summary of Fig. 5.14, showing how each of the points was clustered through the distance metric. The horizontal red dotted line shows where the ideal separation should occur. The first 50 points (green dots of Fig. 5.12) should be grouped so that they are below the red horizontal line in the lower left quadrant. The second 50 points (magenta dots of Fig. 5.12) should be grouped above the red horizontal line in the upper right quadrant. In summary, the dendrogram only misclassified two green points and two magenta points.

Figure 5.16 shows a new dendrogram with a different threshold. Note that in this case, the hierarchical clustering produces more than a dozen clusters. The tuning parameter can be seen to be critical for unsupervised clustering, much like choosing the number of clusters in k-means. In summary, both k-means and hierarchical clustering provide a method whereby data can be parsed automatically into clusters. This provides a starting point for interpretations and analysis in data mining.

5.5 Mixture models and the expectation-maximization algorithm

The third unsupervised method we consider is known as *finite mixture models*. Often the models are assumed to be Gaussian distributions in which case this method is known as *Gaussian mixture models* (GMM). The basic assumption in

parameterized by mean and variance; fully define distribution; If you dont know anything else about data, assume gaussian

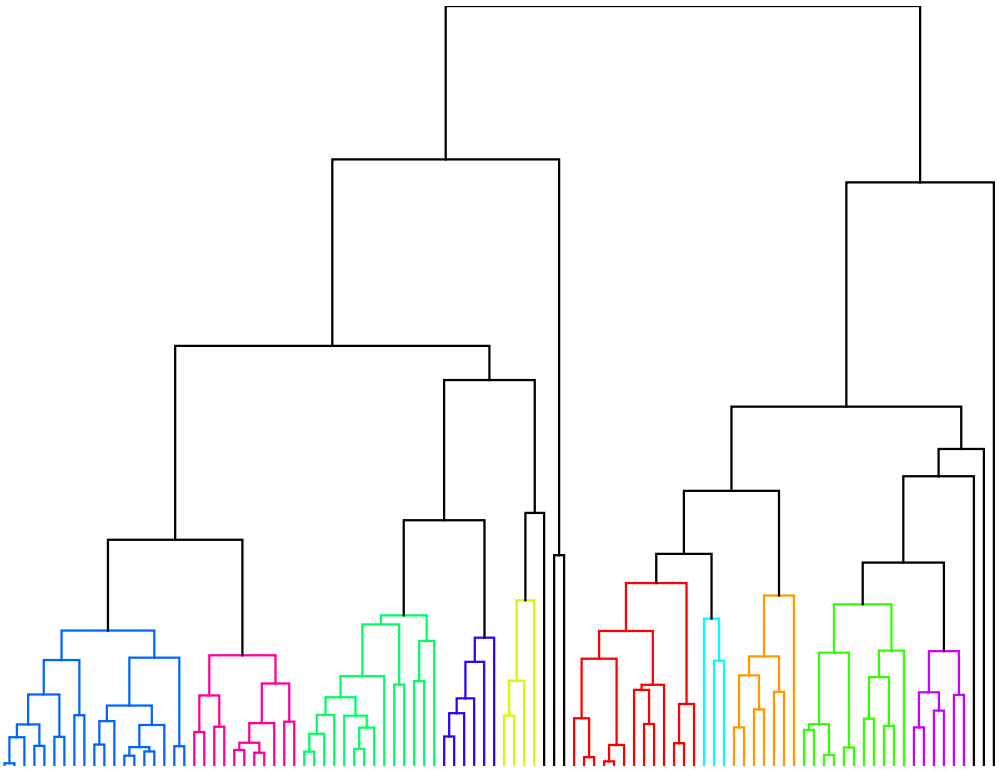


Figure 5.16: Dendrogram structure produced from the data in Fig. 5.12 with a different threshold used than in Fig. 5.14. The dendrogram shows which points are merged as well as the distance between points. In this case, more than a dozen clusters are generated.

this method is that data observations $\mathbf{x}_j$ are a mixture of a set of k processes that combine to form the measurement. Like k-means and hierarchical clustering, the GMM model we fit to the data requires that we specify the number of mixtures k and the individual statistical properties of each mixture that best fit the data. GMMs are especially useful since the assumption that each mixture model has a Gaussian distribution implies that it can be completely characterized by two parameters: the mean and the variance.

The algorithm that enables the GMM computes the maximum-likelihood using the famous *Expectation-Maximization* (EM) algorithm of Dempster, Laird and Rubin [148]. The EM algorithm is designed to find maximum likelihood parameters of statistical models. Generally, the iterative structure of the algorithm finds a local maximum-likelihood, which estimates the true parameters that cannot be directly solved for. As with most data, the observed data involves many latent or unmeasured variables and unknown parameters. Regardless, the alternating and iterative construction of the algorithm recursively estimates the best parameters possible from an initial guess. The EM algo-

rithm proceeds like the k-means algorithm in that initial guesses for the mean and variance are given for the assumed k-distributions. The algorithm then recursively updates the weights of the mixtures versus the parameters of each mixture. One alternates between these two until convergence is achieved.

In any such iteration scheme, it is not obvious that the solution will converge, or that the solution is good, since it typically falls into a local value of the maximum-likelihood. But it can be proven that in this context it does converge, and that the derivative of the likelihood is arbitrarily close to zero at that point, which in turn means that the point is either a maximum or a saddle point [561]. In general, multiple maxima may occur, with no guarantee that the global maximum will be found. Some likelihoods also have singularities, i.e., nonsensical maxima. For example, one of the solutions that may be found by EM in a mixture model involves setting one of the components to have zero variance and the mean equal to one of the data points. Cross-validation can often alleviate some of the common pitfalls that can occur by initializing the algorithm with some bad initial guesses.

The fundamental assumption of the mixture model is that the probability density function (PDF) for observations of data $\mathbf{x}_j$ is a weighted linear sum of a set of unknown distributions

Assume that our state function is the sum of some set of k probability distributions in this case, gaussian distributions

k clusters

$$f(\mathbf{x}_j, \boldsymbol{\Theta}) = \sum_{p=1}^{k} \alpha_p f_p(\mathbf{x}_j, \boldsymbol{\Theta}_p) \tag{5.15}$$

where $f(\cdot)$ is the measured PDF, $f_p(\cdot)$ is the PDF of the mixture j, and k is the total number of mixtures. Each of the PDFs $f_j(\cdot)$ is weighted by α_p ($\alpha_1 + \alpha_2 + \cdots + \alpha_k = 1$) and parametrized by an unknown vector of parameters $\boldsymbol{\Theta}_p$. To state the objective of mixture models more precisely then: *Given the observed PDF $f(\mathbf{x}_j, \boldsymbol{\Theta})$, estimate the mixture weights α_p and the parameters of the distribution $\boldsymbol{\Theta}_p$*. Note that $\boldsymbol{\Theta}$ is a vector containing all the parameters $\boldsymbol{\Theta}_p$. Making this task somewhat easier is the fact that we assume the form of the PDF distribution $f_p(\cdot)$.

For GMM, the parameters in the vector $\boldsymbol{\Theta}_p$ are known to include only two variables: the mean μ_p and variance σ_p. Moreover, the distribution $f_p(\cdot)$ is normally distributed so that (5.15) becomes

Assume that are distributions are Normal with a given mean and variance, weighted by alpha

GMModel = fitgmdist(data, k clusters)

$$f(\mathbf{x}_j, \boldsymbol{\Theta}) = \sum_{p=1}^{k} \alpha_p \mathcal{N}_p(\mathbf{x}_j, \mu_p, \sigma_p). \tag{5.16}$$

This gives a much more tractable framework since there are now a limited set of parameters. Thus once one assumes a number of mixtures k, then the task is to determine α_p along with μ_p and σ_p for each mixture. It should be noted that there are many other distributions besides Gaussian that can be imposed, but

GMM are common since without prior knowledge, an assumption of Gaussian distribution is typically assumed.

An estimate of the parameter vector $\mathbf{\Theta}$ can be computed using the *maximum likelihood estimate* (MLE) of Fisher. The MLE computes the value of $\mathbf{\Theta}$ from the roots of

$$\frac{\partial L(\mathbf{\Theta})}{\partial \mathbf{\Theta}} = 0 \tag{5.17}$$

where the log-likelihood function L is

$$L(\mathbf{\Theta}) = \sum_{j=1}^{n} \log f(\mathbf{x}_j|\mathbf{\Theta}) \tag{5.18}$$

and the sum is over all the n data vectors $\mathbf{x}_j$. The solution to this optimization problem, i.e. when the derivative is zero, produces a local maximizer. This maximizer can be computed using the EM algorithm since derivatives cannot be explicitly computed without an analytic form.

The EM algorithm starts by assuming an initial estimate (guess) of the parameter vector $\mathbf{\Theta}$. This estimate can be used to estimate

$$\tau_p(\mathbf{x}_j, \mathbf{\Theta}) = \frac{\alpha_p f_p(\mathbf{x}_j, \mathbf{\Theta}_p)}{f(\mathbf{x}_j, \mathbf{\Theta})} \tag{5.19}$$

which is the posterior probability of component membership of $\mathbf{x}_j$ in the pth distribution. In other words, does $\mathbf{x}_j$ belong to the pth mixture? The E-step of the EM algorithm uses this posterior to compute memberships. For GMM, the algorithm proceeds as follows: Given an initial parametrization of $\mathbf{\Theta}$ and α_p, compute

$$\tau_p^{(k)}(\mathbf{x}_j) = \frac{\alpha_p^{(k)} \mathcal{N}_p(\mathbf{x}_j, \mu_p^{(k)}, \sigma_p^{(k)})}{\mathcal{N}(\mathbf{x}_j, \mathbf{\Theta}^{(k)})}. \tag{5.20}$$

With an estimated posterior probability, the M-step of the algorithm then updates the parameters and mixture weights

$$\alpha_p^{(k+1)} = \frac{1}{n} \sum_{j=1}^{n} \tau_p^{(k)}(\mathbf{x}_j) \tag{5.21a}$$

$$\mu_p^{(k+1)} = \frac{\sum_{j=1}^{n} \mathbf{x}_j \tau_p^{(k)}(\mathbf{x}_j)}{\sum_{j=1}^{n} \tau_p^{(k)}(\mathbf{x}_j)} \tag{5.21b}$$

$$\Sigma_p^{(k+1)} = \frac{\sum_{j=1}^{n} \tau_p^{(k)}(\mathbf{x}_j) \left(\mathbf{x}_j - \mu_p^{(k+1)}\right) \left(\mathbf{x}_j - \mu_p^{(k+1)}\right)^T}{\sum_{j=1}^{n} \tau_p^{(k)}(\mathbf{x}_j)} \tag{5.21c}$$

where the matrix $\Sigma_p^{(k+1)}$ is the covariance matrix containing the variance parameters. The E- and M-steps are alternated until convergence within a specified

tolerance. Recall that to initialize the algorithm, the number of mixture models k must be specified and initial parametrization (guesses) of the distributions given. This is similar to the k-means algorithm where the number of clusters k is prescribed and an initial guess for the cluster centers is specified.

The GMM is popular since it simply fits k Gaussian distributions to data, which is reasonable for unsupervised learning. The GMM algorithm also has a stronger theoretical base than most unsupervised methods as both k-means and hierarchical clustering are simply defined as algorithms. The primary assumption in GMM is the number of clusters and the form of the distribution $f(\cdot)$.

The following code executes a GMM model on the second and fourth principal components of the dog and cat wavelet image data introduced previously in Figs. 5.4-5.6. Thus the features are the second and fourth columns of the right singular vector of the SVD. The **fitgmdist** command is used to extract the mixture model.

Code 5.10: Gaussian mixture model for cats versus dogs.

```
dogcat=v(:,2:2:4);
GMModel=fitgmdist(dogcat,2)
AIC= GMModel.AIC

subplot(2,2,1)
h=ezcontour(@(x1,x2)pdf(GMModel,[x1 x2]));
subplot(2,2,2)
h=ezmesh(@(x1,x2)pdf(GMModel,[x1 x2]));
```

The results of the algorithm can be plotted for visual inspection, and the parameters associated with each Gaussian are given. Specifically, the mixing proportion of each model along with the mean in each of the two dimensions of the feature space. The following is displayed to the screen.

```
Component 1:
Mixing proportion: 0.355535
Mean:   -0.0290   -0.0753

Component 2:
Mixing proportion: 0.644465
Mean:    0.0758    0.0076

AIC =

 -792.8105
```

The code can also produce an AIC score for how well the mixture of Gaussians explain the data. This gives a principled method for cross-validating in order to determine the number of mixtures required to describe the data.

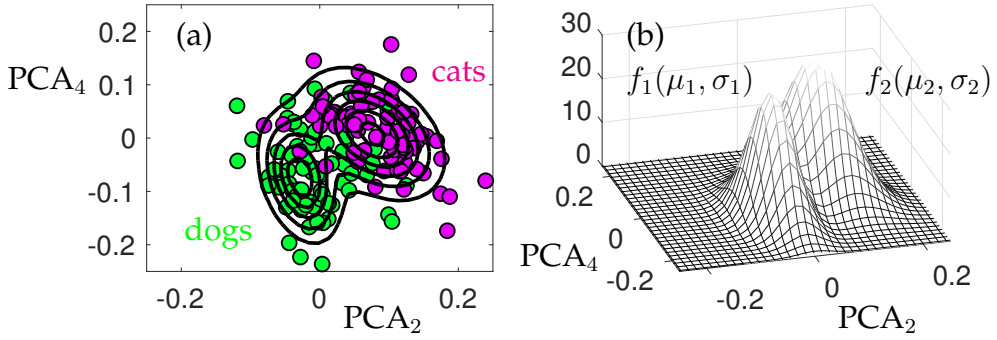


Figure 5.17: GMM fit of the second and fourth principal components of the dog and cat wavelet image data. The two Gaussians are well placed over the distinct dog and cat features as shown in (a). The PDF of the Gaussian models extracted are highlighted in (b) in arbitrary units.

Figure 5.17 shows the results of the GMM fitting procedure along with the original data of cats and dogs. The Gaussians produced from the fitting procedure are also illustrated. The **fitgmdist** command can also be used with **cluster** to label new data from the feature separation discovered by GMM.

using prior knowledge/ labeled training set to help improve classification. In this training set, we can develop a classification system that allows us to determine the classification of future data points that DONT have to be labeled

K -nearest neighbors (knn)

given a training dataset of labeled data, classify new data points based on the K nearest neighbors: which group is the closest?

Expensive computation when you have lots of variables and/or lots of data points.

5.6 Supervised learning and linear discriminants

We now turn our attention to supervised learning methods. One of the earliest supervised methods for classification of data was developed by Fisher in 1936 in the context of taxonomy [182]. His *linear discriminant analysis* (LDA) is still one of the standard techniques for classification. It was generalized by C. R. Rao for multi-class data in 1948 [446]. The goal of these algorithms is to find a linear combination of features that characterizes or separates two or more classes of objects or events in the data. Importantly, for this supervised technique we have labeled data which guides the classification algorithm. Figure 5.18 illustrates the concept of finding an optimal low-dimensional embedding of the data for classification. The LDA algorithm aims to solve an optimization problem to find a subspace whereby the different labeled data have clear separation between their distribution of points. This then makes classification easier because an optimal feature space has been selected.

LDA

The supervised learning architecture includes a training and withhold set of data. The withhold set is never used to train the classifier. However, the training data can be partitioned into k-folds, for instance, to help build a better classification model. The last chapter details how cross-validation should be appropriately used. The goal here is to train an algorithm that uses feature space to make a decision about how to classify data. Figure 5.18 gives a cartoon of the key idea involved in LDA. In our example, two data sets are considered

LDA:

Project our two dimensional data onto a single line. The idea behind LDA is to find the optimal projection that maximizes the separation between the data (via some optimization problem)

MATLAB

class = classify(data, training_data, label)

label : vector of same size as training_data that labels each piece of training data

data : rest of data to classify
class : labels for all of data inferred from training data

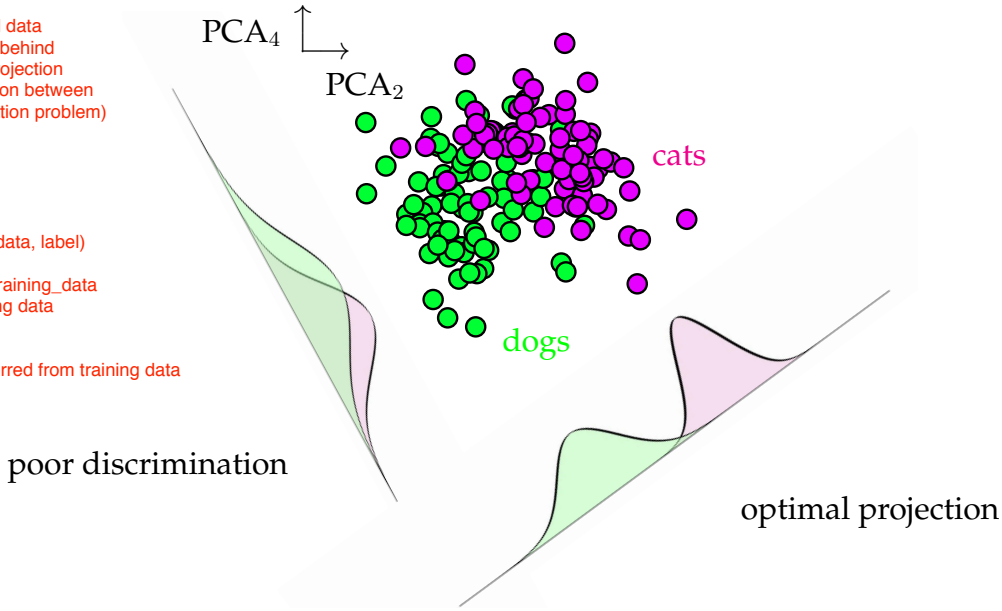


Figure 5.18: Illustration of linear discriminant analysis (LDA). The LDA optimization method produces an optimal dimensionality reduction to a decision line for classification. The figure illustrates the projection of data onto the second and fourth principal component modes of the dog and cat wavelet data considered in Fig. 5.4. Without optimization, a general projection can lead to very poor discrimination between the data. However, the LDA separates the probability distribution functions in an optimal way.

and projected onto new bases. In the left figure, the projection shows that the data is completely mixed, making it difficult to separate the data. In the right figure, which is the ideal charicature for LDA, the data are well separated with the means μ_1 and μ_2 being well apart when projected onto the chosen subspace. Thus the goal of LDA is two-fold: *find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data*.

For a two-class LDA, this results in the following mathematical formulation. Construct a projection $\mathbf{w}$ such that

$$\mathbf{w} = \arg\max_{\mathbf{w}} \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \tag{5.22}$$

where the scatter matrices for between-class $\mathbf{S}_B$ and within-class $\mathbf{S}_W$ data are given by

$$\mathbf{S}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{5.23}$$

$$\mathbf{S}_W = \sum_{j=1}^{2} \sum_{\mathbf{x}} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T . \tag{5.24}$$

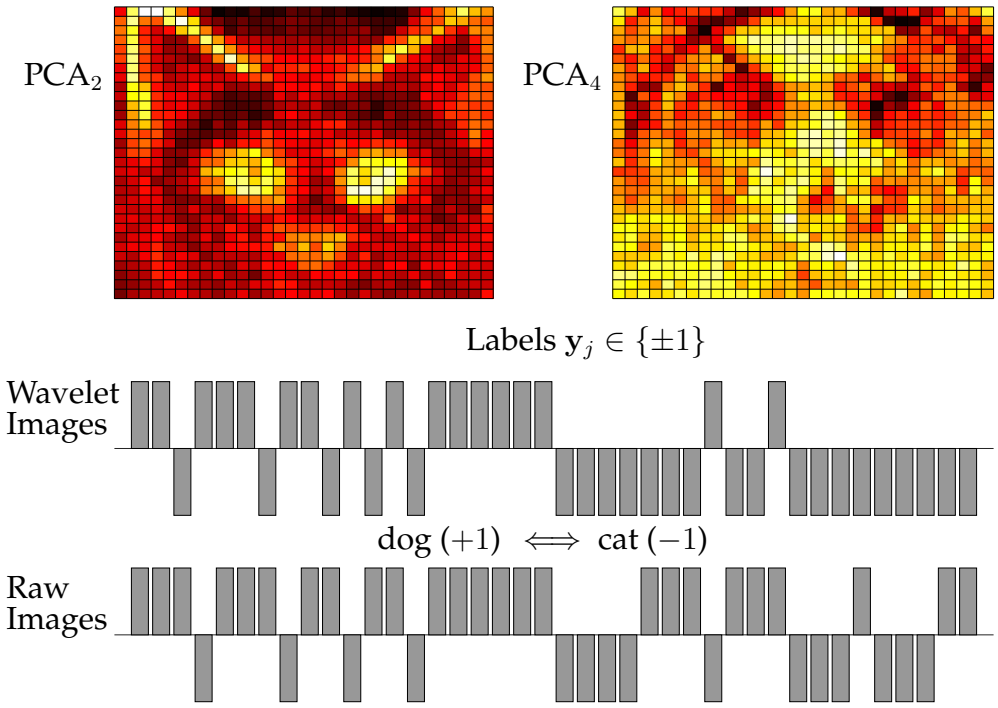


Figure 5.19: Depiction of the performance achieved for classification using the second and fourth principal component modes. The top two panels are PCA modes (features) used to build a classifier. The labels returned are either $\mathbf{y}_j \in \{\pm 1\}$. The ground truth answer in this case should produce a vector of 20 ones followed by 20 negative ones.

These quantities essentially measure the variance of the data sets as well as the variance of the difference in the means. The criterion in (5.22) is commonly known as the generalized Rayleigh quotient whose solution can be found via the generalized eigenvalue problem

$$\mathbf{S}_B\mathbf{w} = \lambda\mathbf{S}_W\mathbf{w} \tag{5.25}$$

where the maximum eigenvalue λ and its associated eigenvector gives the quantity of interest and the projection basis. Thus, once the scatter matrices are constructed, the generalized eigenvectors can be constructed with MATLAB.

Performing an LDA analysis in MATLAB is simple. One needs only to organize the data into a training set with labels, which can then be applied to a test data set. Given a set of data $\mathbf{x}_j$ for $j = 1, 2, \cdots, m$ with corresponding labels $\mathbf{y}_j$, the algorithm will find an optimal classification space as shown in Fig. 5.18. New data $\mathbf{x}_k$ with $k = m+1, m+2, \cdots, m+n$ can then be evaluated and labeled. We illustrate the classification of data using the dog and cat data set introduced in the feature section of this chapter. Specifically, we consider the dog and cat images in the wavelet domain and label them so that $\mathbf{y}_j \in \{\pm 1\}$ ($\mathbf{y}_j = 1$ is a dog

and $\mathbf{y}_j = -1$ is a cat). The following code trains on the first 60 images of dogs and cats, and then tests the classifier on the remaining 20 dog and cat images. For simplicity, we train on the second and fourth principal components as these show good discrimination between dogs and cats (See Fig. 5.5).

Code 5.11: LDA analysis of dogs versus cats.

```
load catData_w.mat
load dogData_w.mat
CD=[dog_wave cat_wave];
[u,s,v]=svd(CD-mean(CD(:)));

xtrain=[v(1:60,2:2:4); v(81:140,2:2:4)];
label=[ones(60,1); -1*ones(60,1)];
test=[v(61:80,2:2:4); v(141:160,2:2:4)];

class=classify(test,xtrain,label);
truth=[ones(20,1); -1*ones(20,1)];
E=100-sum(0.5*abs(class-truth))/40*100
```

Note that the **classify** command in MATLAB takes in the three matrices of interest: the training data, the test data, and the labels for the training data. What is produced are the labels for the test set. One can also extract from this command the decision line for online use. Figure 5.19 shows the results of the classification on the 40 test data samples. Recall that this classification is performed using only the second and fourth PCA modes which cluster as shown in Fig. 5.18. The returned labels are either ± 1 depending on whether a cat or dog is labeled. The ground truth labels for the test data should return a $+1$ (dogs) for the first 20 test sets and a -1 (cats) for the second test set. The accuracy of classification for this realization is 82.5% (2/20 cats are mislabeled while 5/20 dogs are mislabeled). Comparing the wavelet images to the raw images we see that the feature selection in the raw images is not as good. In particular, for the same two principal components, 9/20 cats are mislabeled and 4/20 dogs are mislabeled.

Of course, the data is fairly limited and cross-validation should always be performed to evaluate the classifier. The following code runs 100 trials of the **classify** command where 60 dog and cat images are randomly selected and tested against the remaining 20 images.

Code 5.12: Cross-validation of the LDA analysis.

```
for jj=1:100;
    r1=randperm(80); r2=randperm(80);
    ind1=r1(1:60); ind2=r2(1:60)+60;
    ind1t=r1(61:80); ind2t=r2(61:80)+60;
```

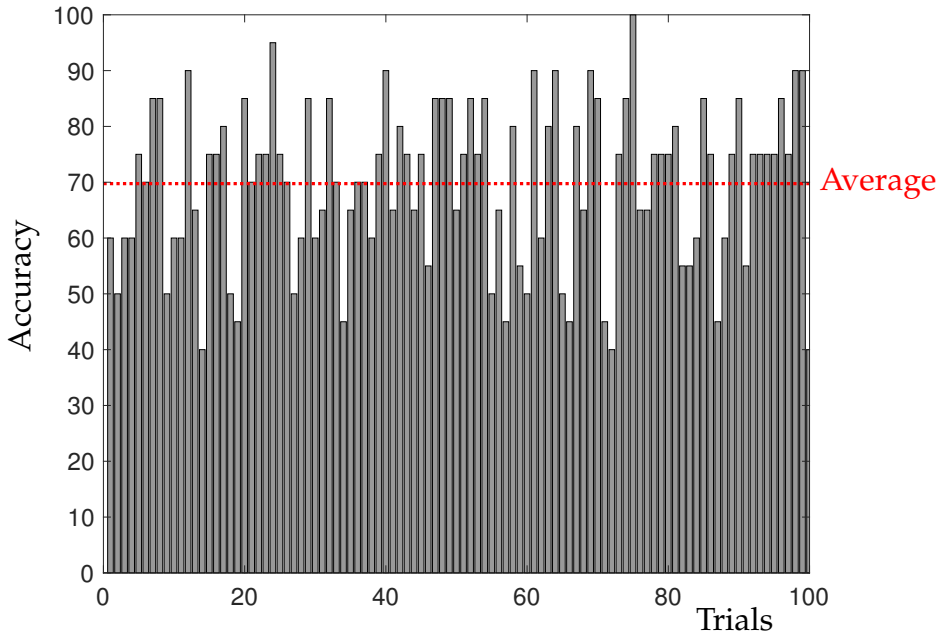


Figure 5.20: Performance of the LDA over 100 trials. Note the variability that can occur in the classifier depending on which data is selected for training and testing. This highlights the importance of cross-validation for building a robust classifier.

```
    xtrain=[v(ind1,2:2:4); v(ind2,2:2:4)];
    test=[v(ind1t,2:2:4); v(ind2t,2:2:4)];

    label=[ones(60,1); -1*ones(60,1)];
    truth=[ones(20,1); -1*ones(20,1)];
    class=classify(test,xtrain,label);
    E(jj)=sum(abs(class-truth))/40*100;
end
```

Figure 5.20 shows the results of the cross-validation over 100 trials. Note the variability that can occur from trial to trial. Specifically, the performance can achieve 100%, but can also be as low as 40%, which is worse than a coin flip. The average classification score (red dotted line) is around 70%. Cross-validation, as already highlighted in the regression chapter, is critical for testing and robustifying the model. Recall that the methods for producing a classifier are based on optimization and regression, so that all the cross-validation methods can be ported to the clustering and classification problem.

In addition to a linear discriminant line, a quadratic discriminant line can be found to separate the data. Indeed, the **classify** command in MATLAB allows one to not only produce the classifier, but also extract the line of separation between the data. The following commands are used to produce labels for new data as well as the discrimination line between the dogs and cats.

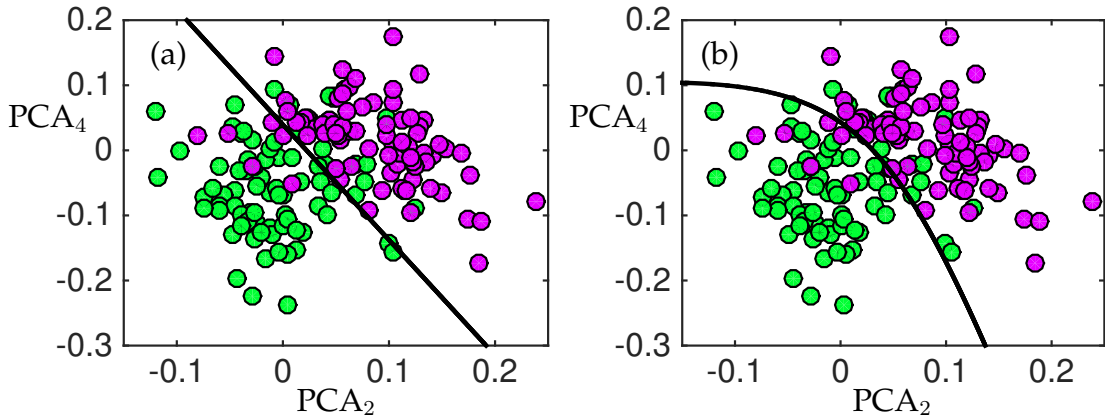


Figure 5.21: Classification line for (a) linear discriminant (LDA) and (b) quadratic discriminant (QDA) for dog (green dots) versus cat (magenta dots) data projected onto the second and fourth principal components. This two dimensional feature space allows for a good discrimination in the data. The two lines represent the best line and parabola for separating the data for a given training sample.

Code 5.13: Plotting the linear and quadratic discrimination lines.

```
subplot(2,2,1)
[class,~,~,~,coeff]=classify(test,xtrain,label);
K = coeff(1,2).const;
L = coeff(1,2).linear;
f = @(x,y) K + [x y]*L;
h2 = ezplot(f,[-.15 0.25 -.3 0.2]);
subplot(2,2,2)
[class,~,~,~,coeff]=classify(test,xtrain,label,'quadratic');
K = coeff(1,2).const;
L = coeff(1,2).linear;
Q = coeff(1,2).quadratic;
f = @(x,y) K + [x y]*L + sum(([x y]*Q) .* [x y], 2);
h2 = ezplot(f,[-.15 0.25 -.3 0.2]);
```

Figure 5.21 shows the dog and cat data along with the linear and quadratic lines separating them. This linear or quadratic fit is found in the structured variable **coeff** which is returned with classify. The quadratic line of separation can often offer a little more flexibility when trying to fit boundaries separating data. A major advantage of LDA based methods: they are easily interpretable and easy to compute. Thus, they are widely used across many branches of the sciences for classification of data.

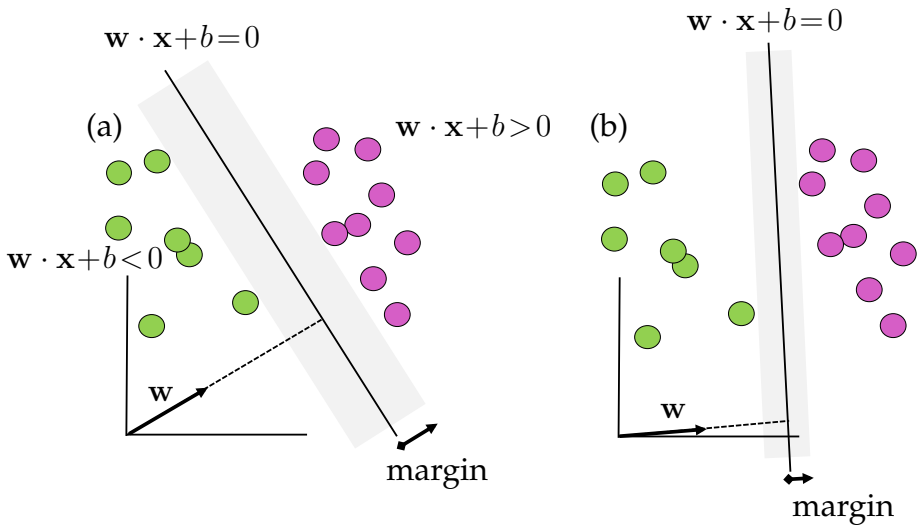
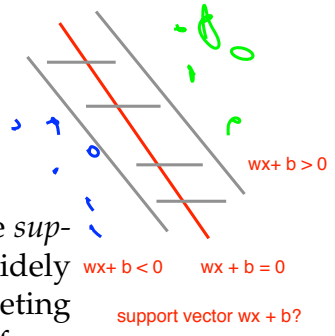


Figure 5.22: The SVM classification scheme constructs a hyperplane $\mathbf{w}\cdot\mathbf{x}+b=0$ that optimally separates the labeled data. The area of the margin separating the labeled data is maximal in (a) and much less in (b). Determining the vector $\mathbf{w}$ and parameter b is the goal of the SVM optimization. Note that for data to the right of the hyperplane $\mathbf{w}\cdot\mathbf{x}+b>0$, while for data to the left $\mathbf{w}\cdot\mathbf{x}+b<0$. Thus the classification labels $\mathbf{y}_j \in \{\pm 1\}$ for the data to the left or right of the hyperplane is given by $\mathbf{y}_j(\mathbf{w}\cdot\mathbf{x}_j+b) = \text{sign}(\mathbf{w}\cdot\mathbf{x}_j+b)$. So only the sign of $\mathbf{w}\cdot\mathbf{x}+b$ needs to be determined in order to label the data. The vectors touching the edge of the gray regions of are termed the *support vectors*.

5.7 Support vector machines (SVM)



Find a vector that separates the data and creates the most "space" around the line; penalize all points that lie on the wrong side of this line

One of the most successful data mining methods developed to date is the *support vector machine* (SVM). It is a core machine learning tool that is used widely in industry and science, often providing results that are better than competing methods. Along with the *random forest* algorithm, they have been pillars of machine learning in the last few decades. With enough training data, the SVM can now be replaced with deep neural nets. But otherwise, SVM and random forest are frequently used algorithms for applications where the best classification scores are required.

The original SVM algorithm by Vapnik and Chervonenkis evolved out of the statistical learning literature in 1963, where hyperplanes are optimized to split the data into distinct clusters. Nearly three decades later, Boser, Guyon and Vapnik created nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes [70]. The current standard incarnation (soft margin) was proposed by Cortes and Vapnik in the mid-1990s [138].

Linear SVM

The key idea of the linear SVM method is to construct a hyperplane

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{5.26}$$

where the vector $\mathbf{w}$ and constant b parametrize the hyperplane. Figure 5.22 shows two potential hyperplanes splitting a set of data. Each has a different value of $\mathbf{w}$ and constant b. The optimization problem associated with SVM is to not only optimize a decision line which makes the fewest labeling errors for the data, but also optimizes the largest margin between the data, shown in the gray region of Fig. 5.22. The vectors that determine the boundaries of the margin, i.e. the vectors touching the edge of the gray regions, are termed the *support vectors*. Given the hyperplane (5.26), a new data point $\mathbf{x}_j$ can be classified by simply computing the sign of $(\mathbf{w} \cdot \mathbf{x}_j + b)$. Specifically, for classification labels $\mathbf{y}_j \in \{\pm 1\}$, the data to the left or right of the hyperplane is given by

$$\mathbf{y}_j(\mathbf{w} \cdot \mathbf{x}_j + b) = \text{sign}(\mathbf{w} \cdot \mathbf{x}_j + b) = \begin{cases} +1 & \text{magenta ball} \\ -1 & \text{green ball.} \end{cases} \tag{5.27}$$

Thus the classifier $\mathbf{y}_j$ is explicitly dependent on the position of $\mathbf{x}_j$.

Critical to the success of the SVM is determining $\mathbf{w}$ and b in a principled way. As with all machine learning methods, an appropriate optimization must be formulated. The optimization is aimed at both minimizing the number of misclassified data points as well as creating the largest margin possible. To construct the optimization objective function, we define a loss function

$$\ell(\mathbf{y}_j, \bar{\mathbf{y}}_j) = \ell(\mathbf{y}_j, \text{sign}(\mathbf{w} \cdot \mathbf{x}_j + b)) = \begin{cases} 0 & \text{if } \mathbf{y}_j = \text{sign}(\mathbf{w} \cdot \mathbf{x}_j + b) \\ +1 & \text{if } \mathbf{y}_j \neq \text{sign}(\mathbf{w} \cdot \mathbf{x}_j + b) \end{cases}. \tag{5.28}$$

Stated more simply

$$\ell(\mathbf{y}_j, \bar{\mathbf{y}}_j) = \begin{cases} 0 & \text{if data is correctly labeled} \\ +1 & \text{if data is incorrectly labeled} \end{cases}. \tag{5.29}$$

Thus each mislabeled point produces a loss of unity. The training error over m data points is the sum of the loss functions $\ell(\mathbf{y}_j, \bar{\mathbf{y}}_j)$.

In addition to minimizing the loss function, the goal is also to make the margin as large as possible. We can then frame the linear SVM optimization problem as

$$\underset{\mathbf{w},b}{\text{argmin}} \sum_{j=1}^{m} \ell(\mathbf{y}_j, \bar{\mathbf{y}}_j) + \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad \min_j |\mathbf{x}_j \cdot \mathbf{w}| = 1. \tag{5.30}$$

Although this is a concise statement of the optimization problem, the fact that the loss function is discrete and constructed from ones and zeros makes it very

difficult to actually optimize. Most optimization algorithms are based on some form of gradient descent which requires smooth objective functions in order to compute derivatives or gradients to update the solution. A more common formulation then is given by

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \sum_{j=1}^{m} H(\mathbf{y}_j, \bar{\mathbf{y}}_j) + \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad \min_j |\mathbf{x}_j \cdot \mathbf{w}| = 1 \tag{5.31}$$

where α is the weighting of the loss function and $H(z) = \max(0, 1-z)$ is called a Hinge loss function. This is a smooth function that counts the number of errors in a linear way and that allows for piecewise differentiation so that standard optimization routines can be employed.

Nonlinear SVM

map data into a higher dimensional space and include some nonlinearities in some of the new variables to capture nonlinearities in the original variables and generate separation. This allows for a LINEAR separation in this new space and SVM works once more!

Although easily interpretable, linear classifiers are of limited value. They are simply too restrictive for data embedded in a high-dimensional space and which may have the structured separation as illustrated in Fig. 5.8. To build more sophisticated classification curves, the feature space for SVM must be enriched. SVM does this by included nonlinear features and then building hyperplanes in this new space. To do this, one simply maps the data into a nonlinear, higher-dimensional space

$$\mathbf{x} \mapsto \Phi(\mathbf{x}) \,. \tag{5.32}$$

We can call the $\Phi(\mathbf{x})$ new *observables* of the data. The SVM algorithm now learns the hyperplanes that optimally split the data into distinct clusters in a new space. Thus one now considers the hyperplane function

data LINEARLY separable in this higher dimensional space but can be expensive.

$$f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x}) + b \tag{5.33}$$

with corresponding labels $\mathbf{y}_j \in \{\pm 1\}$ for each point $f(\mathbf{x}_j)$.

This simple idea, of enriching feature space by defining new functions of the data $\mathbf{x}$, is exceptionally powerful for clustering and classification. As a simple example, consider two dimensional data $\mathbf{x} = (x_1, x_2)$. One can easily enrich the space by considering polynomials of the data.

watch out! This, like many data analysis techniques, may be susceptible to overfitting! Cross validate!

verification data sets!

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1, x_2, x_1^2 + x_2^2). \tag{5.34}$$

This gives a new set of polynomial coordinates in x_1 and x_2 that can be used to embed the data. This philosophy is simple: by embedding the data in a higher dimensional space, it is much more likely to be separable by hyperplanes. As a simple example, consider the data illustrated in Fig. 5.8(b). A linear classifier (or hyperplane) in the x_1-x_2 plane will clearly not be able to separate the data.

Text

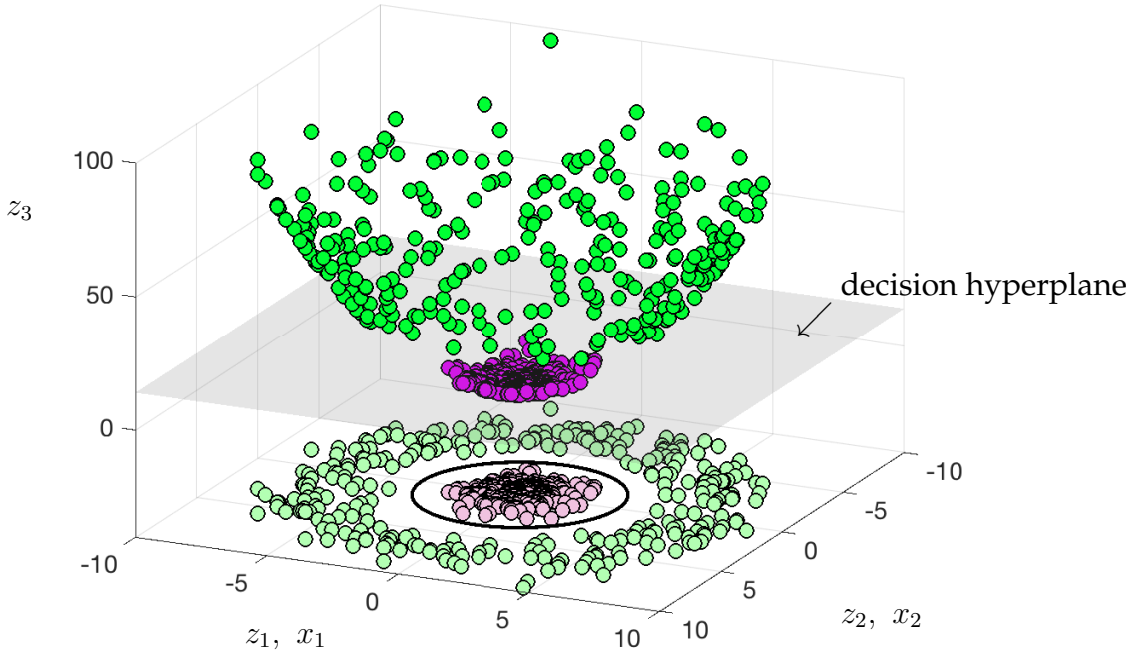


Figure 5.23: The nonlinear embedding of Fig. 5.8(b) using the variables $(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1, x_2, x_1^2 + x_2^2)$ in (5.34). A hyperplane can now easily separate the green from magenta balls, showing that linear classification can be accomplished simply be enriching the measurement space of the data. Visual inspection alone suggests that nearly optimal separation can be achieved with the plane $z_3 \approx 14$ (shaded gray plane). In the original coordinate system this gives a circular classification line (black line on the plane x_1 versus x_2) with radius $r = \sqrt{z_3} = \sqrt{x_1^2 + x_2^2} \approx \sqrt{14}$. This example makes it obvious how a hyperplane in higher-dimensions can produce curved classification lines in the original data space.

However, the embedding (5.34) projects into a three dimensional space which can be easily separated by a hyperplane as illustrated in Fig. 5.23.

The ability of SVM to embed in higher-dimensional nonlinear spaces makes it one of the most successful machine learning algorithms developed. The underlying optimization algorithm (5.31) remains unchanged, except that the previous labeling function $\bar{\mathbf{y}}_j = \text{sign}(\mathbf{w} \cdot \mathbf{x}_j + b)$ is now

$$\bar{\mathbf{y}}_j = \text{sign}(\mathbf{w} \cdot \Phi(\mathbf{x}_j) + b). \tag{5.35}$$

The function $\Phi(\mathbf{x})$ specifies the enriched space of observables. As a general rule, more features are better for classification.

Kernel methods for SVM

Despite its promise, the SVM method of building nonlinear classifiers by enriching in higher-dimensions leads to a computationally intractable optimization. Specifically, the large number of additional features leads to the *curse of dimensionality*. Thus computing the vectors $\mathbf{w}$ is prohibitively expensive and may not even be represented explicitly in memory. The *kernel trick* solves this problem. In this scenario, the $\mathbf{w}$ vector is represented as follows

$$\mathbf{w} = \sum_{j=1}^{m} \alpha_j \Phi(\mathbf{x}_j) \tag{5.36}$$

where α_j are parameters that weight the different nonlinear observable functions $\Phi(\mathbf{x}_j)$. Thus the vector $\mathbf{w}$ is expanded in the observable set of functions. We can then generalize (5.33) to the following

$$f(\mathbf{x}) = \sum_{j=1}^{m} \alpha_j \Phi(\mathbf{x}_j) \cdot \Phi(\mathbf{x}) + b. \tag{5.37}$$

The *kernel function* [479] is then defined as

$$K(\mathbf{x}_j, \mathbf{x}) = \Phi(\mathbf{x}_j) \cdot \Phi(\mathbf{x}) \,. \tag{5.38}$$

With this new definition of $\mathbf{w}$, the optimization problem (5.31) becomes

$$\underset{\boldsymbol{\alpha},b}{\operatorname{argmin}} \sum_{j=1}^{m} H(\mathbf{y}_j, \bar{\mathbf{y}}_j) + \frac{1}{2} \| \sum_{j=1}^{m} \alpha_j \Phi(\mathbf{x}_j) \|^2 \quad \text{subject to} \quad \min_j |\mathbf{x}_j \cdot \mathbf{w}| = 1 \tag{5.39}$$

where $\boldsymbol{\alpha}$ is the vector of α_j coefficients that must be determined in the minimization process. There are different conventions for representing the minimization. However, in this formulation, the minimization is now over $\boldsymbol{\alpha}$ instead of $\mathbf{w}$.

In this formulation, the kernel function $K(\mathbf{x}_j, \mathbf{x})$ essentially allows us to represent Taylor series expansions of a large (infinite) number of observables in a compact way [479]. The kernel function enables one to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between all pairs of data in the feature space. For instance, two of the most commonly used kernel functions are

think of taylor expansion of e^x… captures a lot of polynomials!

$$\text{Radial basis functions (RBF):} \quad K(\mathbf{x}_j, \mathbf{x}) = \exp\left(-\gamma \|\mathbf{x}_j - \mathbf{x}\|^2\right) \tag{5.40a}$$

$$\text{Polynomial kernel:} \quad K(\mathbf{x}_j, \mathbf{x}) = (\mathbf{x}_j \cdot \mathbf{x} + 1)^N \tag{5.40b}$$

where N is the degree of polynomials to be considered, which is exceptionally large to evaluate without using the kernel trick, and γ is the width of the Gaussian kernel measuring the distance between individual data points $\mathbf{x}_j$ and the classification line. These functions can be differentiated in order to optimize (5.39).

This represents the major theoretical underpinning of the SVM method. It allows us to construct higher-dimensional spaces using observables generated by kernel functions. Moreover, it results in a computationally tractable optimization. The following code shows the basic workings of the kernel method on the example of dog and cat classification data. In the first example, a standard linear SVM is used, while in the second, the RBF is executed as an option.

Code 5.14: SVM classification.

```
load catData_w.mat
load dogData_w.mat
CD=[dog_wave cat_wave];
[u,s,v]=svd(CD-mean(CD(:)));

features=1:20;
xtrain=[v(1:60,features); v(81:140,features)];
label=[ones(60,1); -1*ones(60,1)];
test=[v(61:80,features); v(141:160,features)];
truth=[ones(20,1); -1*ones(20,1)];

Mdl = fitcsvm(xtrain,label);
test_labels = predict(Mdl,test);

Mdl = fitcsvm(xtrain,label,'KernelFunction','RBF');
test_labels = predict(Mdl,test);
CMdl = crossval(Mdl);          % cross-validate the model
classLoss = kfoldLoss(CMdl)    % compute class loss
```

Note that in this code we have demonstrated some of the diagnostic features of the SVM method in MATLAB, including the cross-validation and class loss scores that are associated with training. This is a superficial treatment of the SVM. Overall, SVM is one of the most sophisticated machine learning tools in MATLAB and there are many options that can be executed in order to tune performance and extract accuracy/cross-validation metrics.

5.8 Classification trees and random forest

Decision trees are common in business. They establish an algorithmic flow chart for making decisions based on criteria that are deemed important and related to a desired outcome. Often the decision trees are constructed by experts

Sweep across all individual variables and see which one can be used to generate a single line that splits the data the best.
Continuously take the best separation variable until we have the number of clusters we want

with knowledge of the workflow involved in the decision making process. *Decision tree learning* provides a principled method based on data for creating a predictive model for classification and/or regression. Along with SVM, classification and regression trees are core machine learning and data mining algorithms used in industry given their demonstrated success. The work of Leo Breiman and co-workers [79] established many of the theoretical foundations exploited today for data mining.

The decision tree is a hierarchical construct that looks for optimal ways to split the data in order to provide a robust classification and regression. It is the opposite of the unsupervised dendrogram hierarchical clustering previously demonstrated. In this case, our goal is not to move from bottom up in the clustering process, but from top down in order to create the best splits possible for classification. The fact that it is a supervised algorithm, which uses labeled data, allows us to split the data accordingly.

There are significant advantages in developing decision trees for classification and regression: (i) they often produce interpretable results that can be graphically displayed, making them easy to interpret even for non-experts, (ii) they can handle numerical or categorical data equally well, (iii) they can be statistically validated so that the reliability of the model can be assessed, (iv) they perform well with large data sets at scale, and (v) the algorithms mirror human decision making, again making them more interpretable and useful.

As one might expect, the success of decision tree learning has produced a large number of innovations and algorithms for how to best split the data. The coverage here will be limited, but we will highlight the basic architecture for data splitting and tree construction. Recall that we have the following:

$$\text{data} \quad \{\mathbf{x}_j \in \mathbb{R}^n,\ j \in Z := \{1, 2, \cdots, m\}\} \tag{5.41a}$$

$$\text{labels} \quad \{\mathbf{y}_j \in \{\pm 1\},\ j \in Z' \subset Z\}\,. \tag{5.41b}$$

The basic decision tree algorithm is fairly simple: (i) scan through each component (feature) x_k $(k = 1, 2, \cdots, n)$ of the vector $\mathbf{x}_j$ to identify the value of x_j that gives the best labeling prediction for $\mathbf{y}_j$. (ii) Compare the prediction accuracy for each split on the feature x_j. The feature giving the best segmentation of the data is selected as the split for the tree. (iii) With the two new branches of the tree created, this process is repeated on each branch. The algorithm terminates once the each individual data point is a unique cluster, known as a *leaf*, on a new branch of the tree. This is essentially the inverse of the dendrogram.

As a specific example, consider the Fisher iris data set from Fig. 5.1. For this data, each flower had four features (petal width and length, sepal width and length), and three labels (setosa, versicolor and virginica). There were fifty flowers of each variety for a total of 150 data points. Thus for this data the

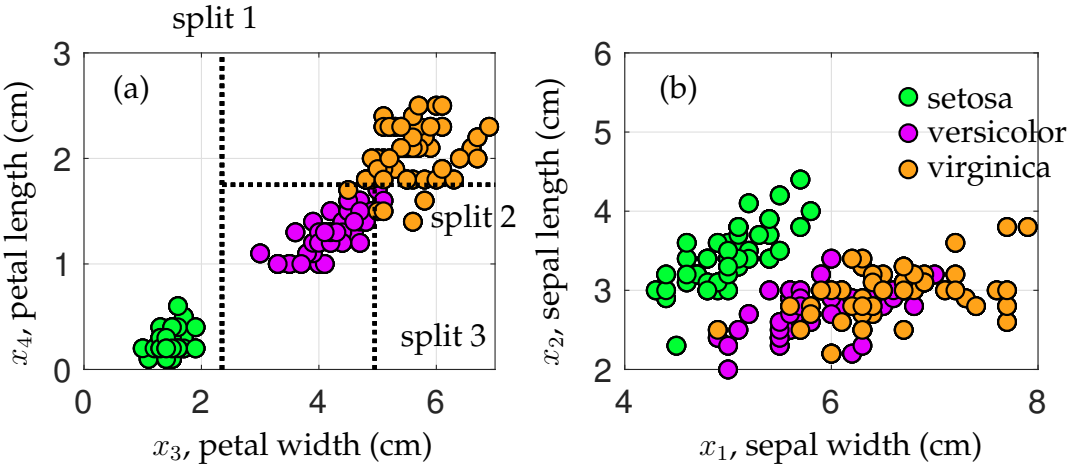


Figure 5.24: Illustration of the splitting procedure for decision tree learning performed on the Fisher iris data set. Each variable x_1 through x_4 is scanned over to determine the best split of data which retains the best correct classification of the labeled data in the split. The variable $x_3 = 2.35$ provides the first split in the data for building a classification tree. This is followed by a second split at $x_4 = 1.75$ and a third split at $x_3 = 4.95$. Only three splits are shown. The classification tree after three splits is shown in Fig. 5.25. Note that although the setosa data in the x_1 and x_2 direction seems to be well separated along a diagonal line, the decision tree can only split along horizontal and vertical lines.

vector $\mathbf{x}_j$ has the four components

$$x_1 = \text{sepal width} \tag{5.42a}$$
$$x_2 = \text{sepal length} \tag{5.42b}$$
$$x_3 = \text{petal width} \tag{5.42c}$$
$$x_4 = \text{petal length}. \tag{5.42d}$$

The decision tree algorithm scans over these four features in order to decide how to best split the data. Figure 5.24 shows the splitting process in the space of the four variables x_1 through x_4. Illustrated are two data planes containing x_1 versus x_2 (panel (b)) and x_3 versus x_4 (panel (a)). By visual inspection, one can see that the x_3 (petal length) variable maximally separates the data. In fact, the decision tree performs the first split of the data at $x_3 = 2.35$. No further splitting is required to predict setosa, as this first split is sufficient. The variable x_4 then provides the next most promising split at $x_4 = 1.75$. Finally, a third split is performed at $x_3 = 4.95$. Only three splits are shown. This process shows that the splitting procedure is has an intuitive appeal as the data splits optimally separating the data are clear visible. Moreover, the splitting does not occur on the x_1 and x_2 (width and length) variables as they do not provide a clear separation of the data. Figure 5.25 shows the tree used for Fig. 5.24.

The following code fits a tree to the Fisher iris data. Note that the `fitctree`

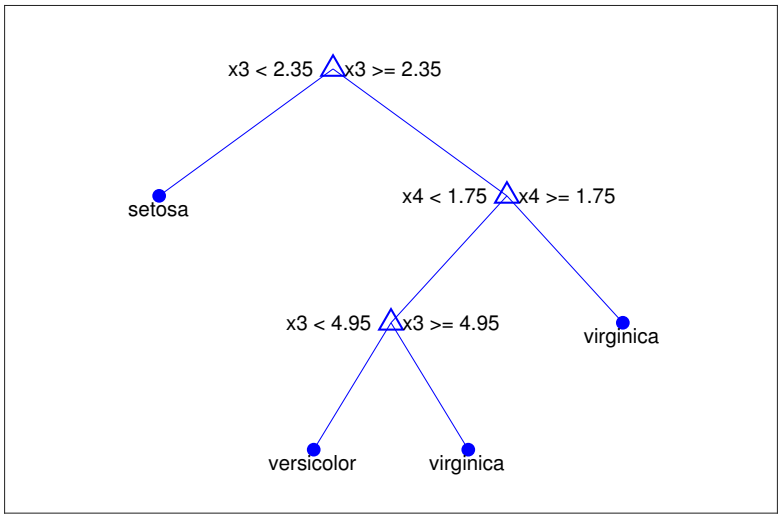


Figure 5.25: Tree structure generated by the MATLAB **fitctree** command. Note that only three splits are conducted, creating a classification tree that produces a class error of 4.67%

command allows for many options, including a cross-validation procedure (used in the code) and parameter tuning (not used in the code).

Code 5.15: Decision tree classification of Fisher iris data.

```
load fisheriris;
tree=fitctree(meas,species,'MaxNumSplits',3,'CrossVal','on')
view(tree.Trained{1},'Mode','graph');
classError = kfoldLoss(tree)

x1=meas(1:50,:);     % setosa
x2=meas(51:100,:);   % versicolor
x3=meas(101:150,:); % virginica
```

The results of the splitting procedure are demonstrated in Fig. 5.25. The **view** command generates an interactive window showing the tree structure. The tree can be pruned and other diagnostics are shown in this interactive graphic format. The class error achieved for the Fisher iris data is 4.67%.

As a second example, we construct a decision tree to the classify dogs versus cats using our previously considered wavelet images. The following code loads and splits the data.

Code 5.16: Decision tree classification of dogs versus cats.

```
load catData_w.mat
load dogData_w.mat
```

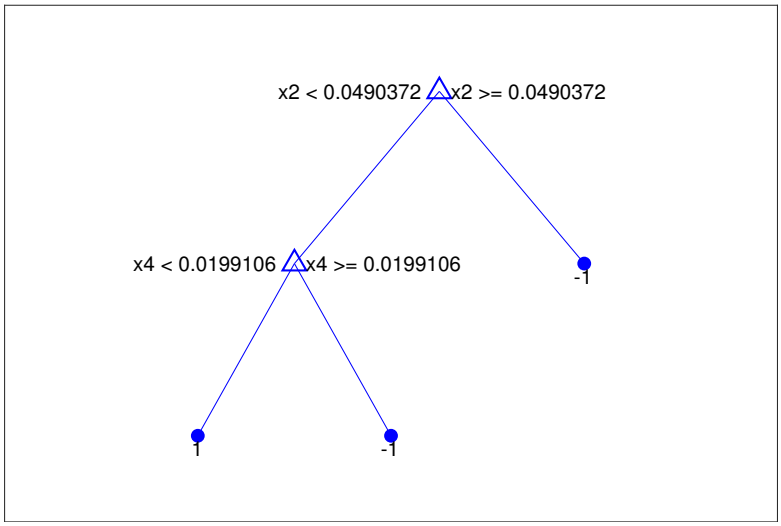


Figure 5.26: Tree structure generated by the MATLAB **fitctree** command for dog versus cat data. Note that only two splits are conducted, creating a classification tree that produces a class error of approximately 16%

```
CD=[dog_wave cat_wave];
[u,s,v]=svd(CD-mean(CD(:)));

features=1:20;
xtrain=[v(1:60,features); v(81:140,features)];
label=[ones(60,1); -1*ones(60,1)];
test=[v(61:80,features); v(141:160,features)];
truth=[ones(20,1); -1*ones(20,1)];

Mdl = fitctree(xtrain,label,'MaxNumSplits',2,'CrossVal','on'
    );
classError = kfoldLoss(Mdl)
view(Mdl.Trained{1},'Mode','graph');
classError = kfoldLoss(Mdl)
```

Figure 5.26 shows the resulting classification tree. Note that the decision tree learning algorithm identifies the first two splits as occurring along the x_2 and x_4 variables respectively. These two variables have been considered previously since their histograms show them to be more distinguishable than the other PCA components (See Fig. 5.5). For this splitting, which has been cross-validated, the class error achieved is approximately 16%, which can be compared with the 30% error of LDA.

As a final example, we consider census data that is included in MATLAB.

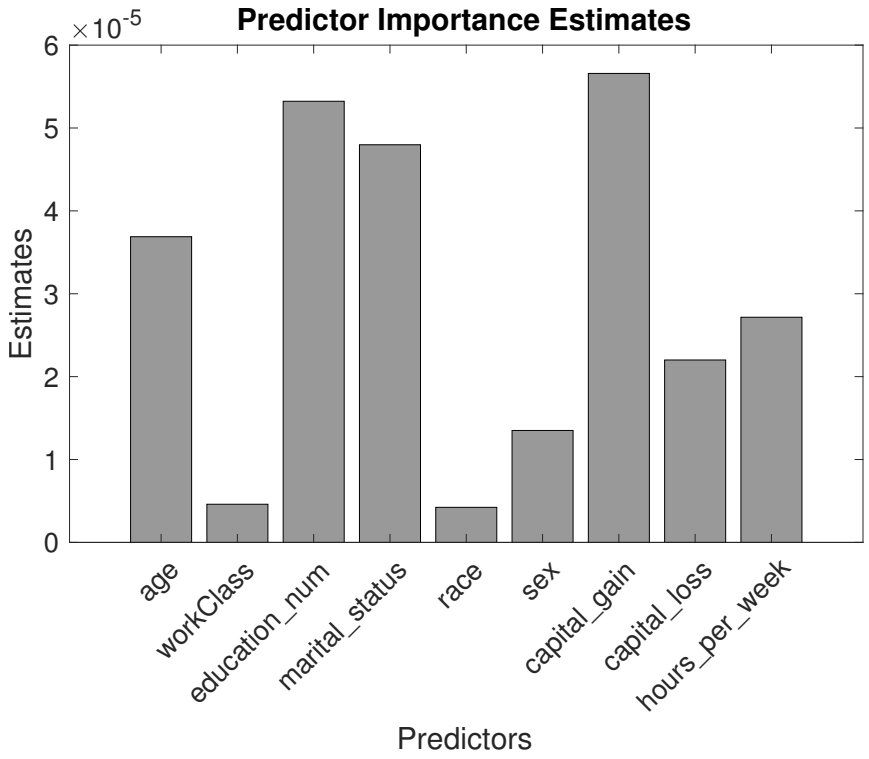


Figure 5.27: Importance of variables for prediction of salary data for the US census of 1994. The classification tree architecture allows for sophisticated treatment of data, including understanding how each variable contributes statistically to predicting a classification outcome.

The following code shows some important uses of the classification and regression tree architecture. In particular, the variables included can be used to make associations between relationships. In this case, the various data is used to predict the salary data. Thus, salary is the outcome of the classification. Moreover, the importance of each variable and its relation to salary can be computed, as shown in Fig. 5.27. The following code highlights some of the functionality of the tree architecture.

Code 5.17: Decision tree classification of census data.

```
load census1994
X = adultdata(:,{'age','workClass','education_num','
   marital_status','race','sex','capital_gain',...
    'capital_loss','hours_per_week','salary'});

Mdl = fitctree(X,'salary','PredictorSelection','curvature','
   Surrogate','on');

imp = predictorImportance(Mdl);
```

```
bar(imp,'FaceColor',[.6 .6 .6],'EdgeColor','k');
title('Predictor Importance Estimates');
ylabel('Estimates'); xlabel('Predictors'); h = gca;
h.XTickLabel = Mdl.PredictorNames;
h.XTickLabelRotation = 45;
```

As with the SVM algorithm, there exists a wide variety of tuning parameters for classification trees, and this is a superficial treatment. Overall, such trees are one of the most sophisticated machine learning tools in MATLAB and there are many options that can be executed to tune performance and extract accuracy/cross-validation metrics.

Take a random subset of my data and generate a tree; repeat this process many times; try joining these trees together?

Random forest algorithms

Before closing this section, it is important to mention Breiman's *random forest* [77] innovations for decision learning trees. Random forests, or random decision forests, are an ensemble learning method for classification and regression. This is an important innovation since the decision trees created by splitting are generally not robust to different samples of the data. Thus one can generate two significantly different classification trees with two subsamples of the data. This presents significant challenges for cross-validation. In ensemble learning, a multitude of decision trees are constructed in the training process. The random decision forests correct for a decision trees' habit of overfitting to their training set, thus providing a more robust framework for classification.

There are many variants of the random forest architecture, including variants with *boosting* and *bagging*. These will not be considered here except to mention that the MATLAB **figctree** exploits many of these techniques through its options. One way to think about ensemble learning is that it allows for robust classification trees. It often does this by focusing its training efforts on hard-to-classify data instead of easy-to-classify data. Random forests, bagging and boosting are all extensive subjects in their own right, but have already been incorporated into leading software which build decision learning trees.

5.9 Top 10 algorithms in data mining 2008

This chapter has illustrated the tremendous diversity of supervised and unsupervised methods available for the analysis of data. Although the algorithms are now easily accessible through many commercial and open-source software packages, the difficulty is now evaluating which method(s) should be used on a given problem. In December 2006, various machine learning experts attending the IEEE International Conference on Data Mining (ICDM) identified the top 10 algorithms for data mining [562]. The identified algorithms where the following: C4.5, k-Means, SVM, Apriori, EM, PageRank, AdaBoost, kNN, Naive

Bayes, and CART. These top 10 algorithms were identified at the time as being among the most influential data mining algorithms in the research community. In the summary article, each algorithm was briefly described along with its impact and potential future directions of research. The 10 algorithms covered classification, clustering, statistical learning, association analysis, and link mining, which are all among the most important topics in data mining research and development. Interestingly, deep learning and neural networks, which are the topic of the next chapter, are not mentioned in the article. The landscape of data science would change significantly in 2012 with the ImageNET data set, and deep convolutional neural networks began to dominate almost any meaningful metric for classification and regression accuracy.

In this section, we highlight their identified top 10 algorithms and the basic mathematical structure of each. Many of them have already been covered in this chapter. This list is not exhaustive, nor does it rank them beyond their inclusion in the top 10 list. Our objective is simply to highlight what was considered by the community as the state-of-the-art data mining tools in 2008. We begin with those algorithms already considered previously in this chapter.

k-means

This is one of the workhorse unsupervised algorithms. As already demonstrated, the goal of k-means is simply to cluster by proximity to a set of k points. By updating the locations of the k points according to the mean of the points closest to them, the algorithm iterates to the k-means. The structure of the MATLAB command is as follows

```
[labels,centers]=kmeans(X,k)
```

The **means** command takes in data $\mathbf{X}$ and the number of prescribed clusters k. It returns labels for each point **labels** along with their location **centers**.

EM (mixture models)

Mixture models are the second workhorse algorithm for unsupervised learning. The assumption underlying the mixture models is that the observed data is produced by a mixture of different probability distribution functions whose weightings are unknown. Moreover, the parameters must be estimated, thus requiring the Expectation-Maximization (EM) algorithm. The structure of the MATLAB command is as follows

parameters of our given gaussian function

```
Model=fitgmdist(X,k)
```

where the **fitgmdist** by default fits Gaussian mixtures to the data $\mathbf{X}$ in k clusters. The **Model** output is a structured variable containing information on the probability distributions (mean, variance, etc.) along with the goodness-of-fit.

Support vector machine (SVM)

One of the most powerful and flexible supervised learning algorithms used for most of the 90s and 2000s, the SVM is an exceptional off-the-shelf method for classification and regression. The main idea: project the data into higher dimensions and split the data with hyperplanes. Critical to making this work in practice was the kernel trick for efficiently evaluating inner products of functions in higher-dimensional space. The structure of the MATLAB command is as follows

```
Model = fitcsvm(xtrain,label);
test_labels = predict(Model,test);
```

where the **fitcsvm** command takes in labeled training data denoted by **train** and **label**, and it produces a structured output **Model**. The structured output can be used along with the **predict** command to take test data **test** and produce labels (**test_labels**). There exist many options and tuning parameters for **fitcsvm**, making it one of the best off-the-shelf methods.

CART (classification and regression tree)

This was the subject of the last section and was demonstrated to provide another powerful technique of supervised learning. The underlying idea was to split the data in a principled and informed way so as to produce an interpretable clustering of the data. The data splitting occurs along a single variable at a time to produce branches of the tree structure. The structure of the MATLAB command is as follows

```
tree = fitctree(xtrain,label);
```

where the **fitctree** command takes in labeled training data denoted by **train** and **label**, and it produces a structured output **tree**. There are many options and tuning parameters for **fitctree**, making it one of the best off-the-shelf methods.

k-nearest neighbors (kNN)

This is perhaps the simplest supervised algorithm to understand. It is highly interpretable and easy to execute. Given a new data point $\mathbf{x}_k$ which does not have a label, simply find the k nearest neighbors $\mathbf{x}_j$ with labels $\mathbf{y}_j$. The label of the new point $\mathbf{x}_k$ is determined by a majority vote of the kNN. Given a model for the data, the MATLAB command to execute the kNN search is the following

```
label = knnsearch(Mdl,test)
```

where the **knnsearch** uses the **Mdl** to label the test data **test**.

Naive Bayes

The Naive Bayes algorithm provides an intuitive framework for supervised learning. It is simple to construct and does not require any complicated parameter estimation, similar to SVM and/or classification trees. It further gives highly interpretable results that are remarkably good in practice. The method is based upon Bayes's theorem and the computation of conditional probabilities. Thus one can estimate the label of a new data point based on the prior probability distributions of the labeled data. The MATLAB command structure for constructing a Naive Bayes model is the following

fitcnb(xtrain, label) is the new command for this

```
Model = fitNaiveBayes(xtrain,label)
```

where the **fitcNativeBayes** command takes in labeled training data denoted by **train** and **label**, and it produces a structured output **Model**. The structured output can be used with the **predict** command to label test data **test**.

AdaBoost (ensemble learning and boosting)

AdaBoost is an example of an *ensemble learning* algorithm [188]. Broadly speaking, AdaBoost is a form of random forest [77] which takes into account an ensemble of decision tree models. The way all boosting algorithms work is to first consider an equal weighting for all training data $\mathbf{x}_j$. Boosting re-weights the importance of the data according to how difficult they are to classify. Thus the algorithm focuses on harder to classify data. Thus a family of weak learners can be trained to yield a strong learner by boosting the importance of hard to classify data [470]. This concept and its usefulness are based upon a seminal theoretical contribution by Kearns and Valiant [283]. The structure of the MATLAB command is as follows

```
ada = fitcensemble(xtrain,label,'Method','AdaBoostM1')
```

where the **fitcensemble** command is a general ensemble learner that can do many more things than AdaBoost, including robust boosting and gradient boosting. Gradient boosting is one of the most powerful techniques [189].

C4.5 (ensemble learning of decision trees)

This algorithm is another variant of decision tree learning developed by J. R. Quinlan [443, 444]. At its core, the algorithm splits the data according to an information entropy score. In its latest versions, it supports boosting as well as many other well known functionalities to improve performance. Broadly, we can think of this as a strong performing version of CART. The **fitcensemble** algorithm highlighted with AdaBoost gives a generic ensemble learning architecture that can incorporate decision trees, allowing for a C4.5-like algorithm.

Apriori algorithm

The last two methods highlighted here tend to focus on different aspects of data mining. In the Apriori algorithm, the goal is to find frequent itemsets from data. Although this may sound trivial, it is not since data sets tend to be very large and can easily produce NP-hard computations because of the combinatorial nature of the algorithms. The Apriori algorithm provides an efficient algorithm for finding frequent itemsets using a candidate generation architecture [4]. This algorithm can then be used for fast learning of associate rules in the data.

PageRank

The founding of Google by Sergey Brin and Larry Page revolved around the PageRank algorithm [82]. PageRank produces a static ranking of variables, such as web pages, by computing an off-line value for each variable that does not depend on search queries. The PageRank is associated with graph theory as it originally interpreted a hyperlink from one page to another as a vote. From this, and various modifications of the original algorithm, one can then compute an importance score for each variable and provide an ordered rank list. The number of enhancements for this algorithm is quite large. Producing accurate orderings of variables (web pages) and their importance remains an active topic of research.

Suggested reading

Texts

(1) Machine learning: a probabilistic perspective, by K. P. Murphy, 2012 [396].

(2) Pattern recognition and machine learning, by C. M. Bishop, 2006 [64].

(3) Pattern classification, by R. O. Duda, P. E. Hart, and D. G. Stork, 2000 [161].

(4) An introduction to statistical learning, by G. James, D. Witten, T. Hastie and R. Tibshirani, 2013 [264].

(5) Learning with kernels: support vector machines, regularization, optimization, and beyond, by B. Schölkopf and A. J. Smola, 2002 [479].

(6) Classification and regression trees, by L. Breiman, J. Friedman, C. J. Stone and R. A. Olshen, 1984 [79].

(7) Random forests, by L. Breiman, 2001 [77].

Papers and reviews

(1) Top 10 algorithms in data mining, by X. Wu et al., *Knowledge and information systems*, 2008 [562].

(2) The strength of weak learnability, by R. E. Schapire, *Machine Learning*, 1990 [470].

(3) Greedy function approximation: a gradient boosting machine, by J. H. Friedman, *Annals of Statistics*, 2001 [189].

Chapter 1

Dynamic Mode Decomposition: An Introduction

The data-driven modeling and control of complex systems is a rapidly evolving field with great potential to transform the engineering, biological, and physical sciences. There is unprecedented availability of high-fidelity measurements from historical records, numerical simulations, and experimental data, and although data is abundant, models often remain elusive. Modern systems of interest, such as a turbulent fluid, an epidemiological system, a network of neurons, financial markets, or the climate, may be characterized as high-dimensional, nonlinear dynamical systems that exhibit rich multi-scale phenomena in both space and time. However complex, many of these system evolve on a low-dimensional attractor that may be characterized by spatio-temporal coherent structures. In this chapter, we will introduce the topic of this book, the dynamic mode decomposition (DMD), which is a powerful new technique for the discovery of dynamical systems from high-dimensional data.

The DMD method originated in the fluid dynamics community as a method to decompose complex flows into a simple representation based on spatio-temporal coherent structures. Schmid and Sesterhenn [251] and Schmid [248] first defined the DMD algorithm and demonstrated its ability to provide insights from high-dimensional fluids data. The growing success of DMD stems from the fact that it is an *equation-free*, data-driven method capable of providing an accurate decomposition of a complex system into spatio-temporal coherent structures that may be used for short-time future state prediction and control. More broadly, DMD has quickly gained popularity since Mezić *et al.* [197, 195, 196]

and Rowley *et al.* [236] showed that it is connected to the underlying nonlinear dynamics through Koopman operator theory [164] and is readily interpretable using standard dynamical systems techniques.

The development of dynamic mode decomposition is timely due to the concurrent rise of data science, encompassing a broad range of techniques from machine learning and statistical regression to computer vision and compressed sensing. Improved algorithms, abundant data, vastly expanded computational resources and interconnectedness of data streams make this a fertile ground for rapid development. Throughout this chapter, we will introduce the core DMD algorithm, provide a broader historical context, and lay the mathematical foundation for future innovations and applications of DMD in the remaining chapters.

1.1 ▪ Dynamic mode decomposition

Modes of DMD does NOT have to be orthogonal

eigenvalues with positive real parts EXPLODE in time; drastically increase and this tends to make our DMD give poor predictions. This is a consequence of trying to use a linear approximation to an often nonlinear system

The dynamic mode decomposition is the featured method of this book. At its core, the method can be thought of as an ideal combination of spatial dimensionality-reduction techniques, such as the proper orthogonal decomposition (POD)[1], with Fourier transforms in time. Thus correlated spatial modes are also now associated with a given temporal frequency, possibly with a growth or decay rate. The method relies simply on collecting snapshots of data $\mathbf{x}_k$ from a dynamical system at a number of times t_k where $k = 1,2,3,\cdots,m$. As we will show, the DMD is algorithmically a regression of data onto locally linear dynamics $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k$ where $\mathbf{A}$ is chosen to minimize $||\mathbf{x}_{k+1} - \mathbf{A}\mathbf{x}_k||_2$ over the $k = 1,2,3,\cdots,m-1$ snapshots. The advantage of this method is that it is very simple to execute, and it makes almost no assumptions about the underlying system. The cost of the algorithm is a singular value decomposition (SVD) of the snapshot matrix constructed from the data $\mathbf{x}_k$.

DMD has a number of uses and interpretations. Specifically, the DMD algorithm can generally be thought to enable three primary tasks:

I. diagnostics: At its inception, the DMD algorithm was used as a diagnostic tool to characterize complex fluid flows [248, 236]. In particular, the algorithm extracts key low-rank spatio-temporal features of many high-dimensional systems, allowing for physically interpretable results in terms of spatial structures and their associated temporal re-

[1] The POD is often computed using the singular value decomposition (SVD). It is also known as the principal components analysis (PCA) in statistics [214, 149], empirical orthogonal functions (EOF) in climate and meteorology [183], the discrete Karhunen-Loeve transform, and the Hotelling transform [135, 136].

sponses. Interestingly, this is still perhaps the primary function of DMD in many application areas. The diagnostic nature of DMD allows for the data-driven discovery of fundamental, low-rank structures in complex systems analogous to POD analysis in fluid flows, plasma physics, atmospheric modeling, etc.

II. State Estimation and Future state prediction: A more sophisticated and challenging use of the DMD algorithm is associated with using the spatio-temporal structures that are dominant in the data to construct dynamical models of the underlying processes observed. This is a much more difficult task, especially as the DMD is limited to constructing the best fit (least-square) linear dynamical system to the nonlinear dynamical system generating the data. Thus unlike the diagnostic objective, the goal is to anticipate the state of the system in a regime where no measurements were made. Confounding the regressive nature of the DMD is the fact that the underlying dynamics can exhibit multi-scale dynamics in both time and space. Regardless, there are a number of key strategies, including intelligent sampling of the data and updating the regression, that allow the DMD to be effective for generating a useful linear dynamical model. This generative model approach can then be used for future state predictions of the dynamical systems and has been used with success in many application areas.

III. Control: Enabling viable and robust control strategies directly from data sampling is the ultimate, and most challenging, goal of the DMD algorithm. Given that we are using a linear dynamical model to predict the future of a nonlinear dynamical system, it is reasonable to expect that there is only a limited, perhaps short-time, window in the future where the two models will actually agree. The hope is that this accurate prediction window is long enough in duration to enable a control decision capable of influencing the future state of the system. The DMD algorithm in this case allows for a completely data-driven approach to control theory, thus providing a compelling mathematical framework for controlling complex dynamical systems whose governing equations are not known or are difficult to model computationally.

When using the DMD method, one should always be mindful of the intended use and the expected outcome. Further, although it seems quite obvious, the success of the DMD algorithm will depend largely on which of the above tasks one is attempting to achieve. Throughout this book, many of the chapters will address one, two or all of the above objectives. In some applications, it is only reasonable at this point in

time to use it as a diagnostic tool. In other applications, limited success has been achieved even for the task of control. Undoubtedly, many researchers are working hard to move emerging application areas through this task list towards achieving accurate modeling and control of dynamical systems in an equation-free framework.

1.2 ▪ Formulating the DMD architecture

In the DMD architecture, we typically consider data collected from a dynamical system

some function dictating the dynamics of our system

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t; \mu), \tag{1.1}$$

where $\mathbf{x}(t) \in \mathbb{R}^n$ is a vector representing the state of our dynamical system at time t, μ contains parameters of the system, and $\mathbf{f}(\cdot)$ represents the dynamics. Generally, a dynamical system is represented as a coupled system of ordinary differential equations that are often nonlinear. The state $\mathbf{x}$ is typically quite large, having dimension $n \gg 1$; this state may arise as the discretization of a partial differential equation at a number of discrete spatial locations. Finally, the continuous-time dynamics from (1.1) may also induce a corresponding discrete-time representation, in which we sample the system every Δt in time and denote the time as a subscript so that $\mathbf{x}_k = \mathbf{x}(k\Delta t)$. We denote the discrete-time *flow* map obtained by evolving (1.1) for Δt by $\mathbf{F}$:

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k). \tag{1.2}$$

Measurements of the system

$$\mathbf{y}_k = \mathbf{g}(\mathbf{x}_k), \tag{1.3}$$

are collected at times t_k from $k = 1, 2, \cdots, m$ for a total of m measurement times. In many applications, the measurements are simply the state, so that $\mathbf{y}_k = \mathbf{x}_k$. The data assimilation community often represents these measurements by the implicit expression $G(\mathbf{x}, t) = 0$. The initial conditions are prescribed as $\mathbf{x}(t_0) = \mathbf{x}_0$.

As already highlighted, $\mathbf{x}$ is an n-dimensional vector ($n \gg 1$) that arises from either discretization of a complex system, or in the case of applications such as video streams, it is the total number of pixels in a given frame. The governing equations and initial condition specify a well-posed initial value problem.

In general, it is not possible to construct a solution to the governing nonlinear evolution (1.1), and so numerical solutions are used to evolve to future states. The DMD framework takes the equation-free

perspective where the dynamics $\mathbf{f}(\mathbf{x}, t; \mu)$ may be unknown. Thus data measurements of the system alone are used to approximate the dynamics and predict the future state. The DMD procedure constructs the proxy, approximate locally linear dynamical system

A is a matrix with CONSTANT coefficients

$$\frac{d\mathbf{x}}{dt} = \mathcal{A}\mathbf{x} \tag{1.4}$$

with initial condition $\mathbf{x}(0)$ and whose well-known solution [33] is

i.e. an eigenfunction expansion

$$\mathbf{x}(t) = \sum_{k=1}^{n} \boldsymbol{\phi}_k \exp(\omega_k t) b_k = \boldsymbol{\Phi} \exp(\boldsymbol{\Omega} t)\mathbf{b} \tag{1.5}$$

w_k = ln(lambda_k) / dt?
see pg 10bottom below

where $\boldsymbol{\phi}_k$ and ω_k are the eigenvectors and eigenvalues of the matrix $\mathbf{A}$, and the coefficients b_k are the coordinates of $\mathbf{x}(0)$ in the eigenvector basis. → Phi * b = x(0)

Given continuous dynamics as in (1.4), it is always possible to describe an analogous discrete-time system sampled every Δt in time:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k, \tag{1.6}$$

where

$$\mathbf{A} = \exp(\mathcal{A}\Delta t). \tag{1.7}$$

$\mathcal{A}$ refers to the matrix in the continuous-time dynamics from (1.4). The solution to this system may be expressed simply in terms of the eigenvalues λ_k and eigenvectors $\boldsymbol{\phi}_k$ of the discrete-time map $\mathbf{A}$:

$$\mathbf{x}_k = \sum_{j=1}^{r} \boldsymbol{\phi}_j \lambda_j^k b_j = \boldsymbol{\Phi}\boldsymbol{\Lambda}^k \mathbf{b}. \tag{1.8}$$

As before, $\mathbf{b}$ are the coefficients of the initial condition $\mathbf{x}_1$ in the eigenvector basis, so that $\mathbf{x}_1 = \boldsymbol{\Phi}\mathbf{b}$. The DMD algorithm produces a low-rank eigen-decomposition (1.8) of the matrix $\mathbf{A}$ that optimally fits the measured trajectory $\mathbf{x}_k$ for $k = 1, 2, \cdots, m$ in a least square sense so that

DMD is an eigendecomposition of the dynamics matrix A where A maps states to future states.

$$||\mathbf{x}_{k+1} - \mathbf{A}\mathbf{x}_k||_2 \tag{1.9}$$

is minimized across all points for $k = 1, 2, \cdots, m-1$. The optimality of the approximation holds only over the sampling window where $\mathbf{A}$ is constructed, and the approximate solution can be used to not only make future state predictions, but also to decompose the dynamics into various time-scales since the λ_k are prescribed.

Arriving at the optimally constructed matrix $\mathbf{A}$ for the approximation (1.6) and the subsequent eigendecomposition in (1.8) is the focus

of the remainder of this introductory chapter. In subsequent chapters, both a deeper theoretical understanding will be pursued along with various demonstrations of the power of this methodology.

To minimize the approximation error (1.9) across all snapshots from $k = 1, 2, \cdots, m$, it is possible to arrange the m snapshots into two large data matrices:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{m-1} \\ | & | & & | \end{bmatrix} \tag{1.10a}$$

$$\mathbf{X}' = \begin{bmatrix} | & | & & | \\ \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_m \\ | & | & & | \end{bmatrix}. \tag{1.10b}$$

Recall that these data snapshots are likely sampled from a nonlinear dynamical system (1.1), although we are finding an optimal local linear approximation. The locally linear approximation (1.6) may be written in terms of these data matrices as:

Why do we define it like this rather than (X' - X) = AX as dx/dt = Ax
If our state is never changing, we would expect dx/dt = 0 —> A = 0 but equation 1.11 gives A = Identity

$$\mathbf{X}' \approx \mathbf{A}\mathbf{X}. \tag{1.11}$$

The best-fit $\mathbf{A}$ matrix is given by

$$\mathbf{A} = \mathbf{X}'\mathbf{X}^{\dagger}, \tag{1.12}$$

where † is the Moore-Penrose pseudo-inverse. This solution minimizes the error:

$$||\mathbf{X}' - \mathbf{A}\mathbf{X}||_F, \tag{1.13}$$

where $||\cdot||_F$ is the Frobenius norm, given by

$$||\mathbf{X}||_F = \sqrt{\sum_{j=1}^{n}\sum_{k=1}^{m} {X_{jk}}^2}. \tag{1.14}$$

It is important to note that (1.13) and the solution (1.12) may be thought of as a linear regression of data onto the dynamics given by $\mathbf{A}$. However, there is a key difference between DMD and alternative regression-based system identification and model reduction techniques. Importantly, we are assuming that the snapshots $\mathbf{x}_k$ in our data matrix $\mathbf{X}$ are high-dimensional so that the matrix is *tall-and-skinny*, meaning that the size of a snapshot n is larger than the number of snapshots $m - 1$. The matrix $\mathbf{A}$ may be high-dimensional; if $n = 10^6$, then $\mathbf{A}$ has 10^{12}

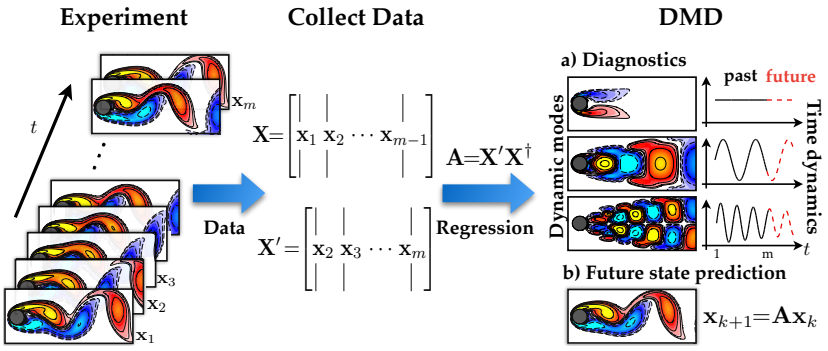


Figure 1.1. *Schematic overview of dynamic mode decomposition on a fluid flow example that will be explored further in Chapter 2. Note that the regression step does not typically construct* $\mathbf{A}$*, but instead constructs* $\tilde{\mathbf{A}}$ *which evolves the dynamics on a low-rank subspace, via proper orthogonal decomposition. The eigendecomposition of* $\tilde{\mathbf{A}}$ *is then used to approximate the eigendecomposition of the high-dimensional matrix* $\mathbf{A}$*.*

elements, so that it may be difficult to represent or decompose. However, the rank of $\mathbf{A}$ is at most $m-1$ since it is constructed as a linear combination of the $m-1$ columns of $\mathbf{X}'$. Instead of solving for $\mathbf{A}$ directly, we first project our data onto a low-rank subspace defined by at most $m-1$ POD modes, and then solve for a low-dimensional evolution $\tilde{\mathbf{A}}$ that evolves on these POD mode coefficients. The DMD algorithm then uses this low-dimensional operator $\tilde{\mathbf{A}}$ to reconstruct the leading non-zero eigenvalues and eigenvectors of the full-dimensional operator $\mathbf{A}$ without ever explicitly computing $\mathbf{A}$. This is discussed in § 1.3 below.

1.2.1 ▪ Defining DMD

The DMD method provides a spatio-temporal decomposition of data into a set of dynamic modes that are derived from snapshots or measurements of a given system in time. A schematic overview is provided in Fig. 1.1. The mathematics underlying the extraction of dynamic information from time-resolved snapshots is closely related to the idea of the Arnoldi algorithm [248], one of the workhorses of fast computational solvers. The data collection process involves two parameters:

$$n = \text{number of spatial points saved per time snapshot}$$
$$m = \text{number of snapshots taken}$$

The DMD algorithm was originally designed to collect data at regularly spaced intervals of time, as in (1.10). However, new innovations allow

for both sparse spatial [48] and temporal [290] collection of data as well as irregularly spaced collection times. Indeed, Tu *et al.* [291] provides the most modern definition of the DMD method and algorithm.

Definition: Dynamic Mode Decomposition (Tu *et al.* 2014 [291]): *Suppose we have a dynamical system (1.1) and two sets of data*

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{m-1} \\ | & | & & | \end{bmatrix} \tag{1.16a}$$

$$\mathbf{X}' = \begin{bmatrix} | & | & & | \\ \mathbf{x}'_1 & \mathbf{x}'_2 & \cdots & \mathbf{x}'_{m-1} \\ | & | & & | \end{bmatrix} \tag{1.16b}$$

so that $\mathbf{x}'_k = \mathbf{F}(\mathbf{x}_k)$ *where* $\mathbf{F}$ *is the map in* (1.2) *corresponding to the evolution of* (1.1) *for time* Δt. *DMD computes the leading eigendecomposition of the best-fit linear operator* $\mathbf{A}$ *relating the data* $\mathbf{X}' \approx \mathbf{A}\mathbf{X}$*:*

$$\mathbf{A} = \mathbf{X}'\mathbf{X}^{\dagger}. \tag{1.17}$$

The DMD modes, also called dynamic modes, are the eigenvectors of $\mathbf{A}$*, and each DMD mode corresponds to a particular eigenvalue of* $\mathbf{A}$.

If I know A, I can attempt to understand how to control the system dx/dt = Ax + Bu where Bu is some forcing term. I can update my A matrix after each data measurement xi to get more accurate predictions

1.2.2 ▪ DMD and the Koopman operator

The DMD method approximates the modes of the so-called *Koopman operator*. The Koopman operator is a linear, infinite-dimensional operator that represents the action of a nonlinear dynamical system on the Hilbert space of measurement functions of the state [236, 196]. It is important to note that the Koopman operator does not rely on linearization of the dynamics, but instead represents the flow of the dynamical system on measurement functions as an infinite dimensional operator.

The DMD method can be viewed as computing, from the experimental data, the eigenvalues and eigenvectors (low-dimensional modes) of a finite-dimensional linear model that approximates the infinite dimensional Koopman operator. Since the operator is linear, the decomposition gives the growth rates and frequencies associated with each mode. If the underlying dynamics in (1.1) are linear, then the DMD method recovers the leading eigenvalues and eigenvectors normally computed using standard solution methods for linear differential equations.

Mathematically, the Koopman operator $\mathscr{K}$ is an infinite dimensional linear operator that acts on the Hilbert space $\mathscr{H}$ containing *all* scalar-valued measurement functions $g : \mathbb{C}^n \to \mathbb{C}$ of the state $\mathbf{x}$. The action of

Koopman

y = g(x)

Y = [y1 y2 … ym]

dy/dt = Ay
can give AWESOME predictions to nonlinear systems. i.e. x may not be linear but choosing a good g(x) can make it well-approx by a linear system. Extends the valid range of our prediction window

Use our knowledge of the system to make a best guess of this function g(x)

the Koopman operator on a measurement function g equates to composition of the function g with the flow map $\mathbf{F}$:

$$\mathcal{K} g = g \circ \mathbf{F}, \tag{1.18a}$$

$$\implies \mathcal{K} g(\mathbf{x}_k) = g(\mathbf{F}(\mathbf{x}_k)) = g(\mathbf{x}_{k+1}). \tag{1.18b}$$

In other words, the Koopman operator, also known as the composition operator, advances measurements along with the flow $\mathbf{F}$. This is incredibly powerful and general, since this holds for all measurement functions g evaluated at any state vector $\mathbf{x}$.

The approximation of the Koopman operator is at the heart of the DMD methodology. As already stated, the mapping over Δt is linear even though the underlying dynamics that generated $\mathbf{x}_k$ may be nonlinear. It should be noted that this is fundamentally different than linearizing the dynamics. However, the quality of any finite-dimensional approximation to the Koopman operator depends on the measurements $\mathbf{y} = \mathbf{g}(\mathbf{x})$ chosen. This will be discussed in depth in Chapter 3.

1.3 ▪ The DMD algorithm

In practice, when the state dimension n is large, the matrix $\mathbf{A}$ may be intractable to analyze directly. Instead, DMD circumvents the eigendecomposition of $\mathbf{A}$ by considering a rank-reduced representation in terms of a POD-projected matrix $\tilde{\mathbf{A}}$. The DMD algorithm, also shown in Code 1.3 as a function, proceeds as follows [291]:

1. First, take the singular value decomposition (SVD) of $\mathbf{X}$ [284]:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*, \tag{1.19}$$

   where $*$ denotes the conjugate transpose, $\mathbf{U} \in \mathbb{C}^{n \times r}$, $\mathbf{\Sigma} \in \mathbb{C}^{r \times r}$ and $\mathbf{V} \in \mathbb{C}^{m \times r}$. Here r is the rank of the reduced SVD approximation to $\mathbf{X}$. The left singular vectors $\mathbf{U}$ are POD modes. The columns of $\mathbf{U}$ are orthonormal so $\mathbf{U}^*\mathbf{U} = \mathbf{I}$; similarly, $\mathbf{V}^*\mathbf{V} = \mathbf{I}$.

   The SVD reduction in (1.19) could also be exploited at this stage in the algorithm to perform a low-rank truncation of the data. Specifically, if low-dimensional structure is present in the data, the singular values of $\mathbf{\Sigma}$ will decrease sharply to zero with perhaps only a limited number of dominant modes. A principled way to truncate noisy data is given by the recent hard-thresholding algorithm of Gavish and Donoho [107], as discussed in § 8.2.

2. The matrix $\mathbf{A}$ from (1.17) may be obtained by using the pseudo-inverse of $\mathbf{X}$ obtained via the SVD:

$$\mathbf{A} = \mathbf{X}'\mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^*. \tag{1.20}$$

In practice, it is more efficient computationally to compute $\tilde{\mathbf{A}}$, the $r \times r$ projection of the full matrix $\mathbf{A}$ onto POD modes:

$$\tilde{\mathbf{A}} = \mathbf{U}^*\mathbf{A}\mathbf{U} = \mathbf{U}^*\mathbf{X}'\mathbf{V}\boldsymbol{\Sigma}^{-1}. \tag{1.21}$$

The matrix $\tilde{\mathbf{A}}$ defines a low-dimensional linear model of the dynamical system on POD coordinates:

$$\tilde{\mathbf{x}}_{k+1} = \tilde{\mathbf{A}}\tilde{\mathbf{x}}_k. \tag{1.22}$$

It is possible to reconstruct the high-dimensional state $\mathbf{x}_k = \mathbf{U}\tilde{\mathbf{x}}_k$.

3. Compute the eigendecomposition of $\tilde{\mathbf{A}}$:

$$\tilde{\mathbf{A}}\mathbf{W} = \mathbf{W}\boldsymbol{\Lambda}, \tag{1.23}$$

where columns of $\mathbf{W}$ are eigenvectors and $\boldsymbol{\Lambda}$ is a diagonal matrix containing the corresponding eigenvalues λ_k.

4. Finally, we may reconstruct the eigendecomposition of $\mathbf{A}$ from $\mathbf{W}$ and $\boldsymbol{\Lambda}$. In particular, the eigenvalues of $\mathbf{A}$ are given by $\boldsymbol{\Lambda}$ and the eigenvectors of $\mathbf{A}$ (DMD modes) are given by columns of $\boldsymbol{\Phi}$:

$$\boldsymbol{\Phi} = \mathbf{X}'\mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{W}. \tag{1.24}$$

Note that (1.24) from [291] differs from the formula $\boldsymbol{\Phi} = \mathbf{U}\mathbf{W}$ in [248], although these will tend to converge if $\mathbf{X}$ and $\mathbf{X}'$ have the same column spaces. The modes in (1.24) are often called *exact DMD modes*, because it was proven in Tu *et al.* [291] that these are exact eigenvectors of the matrix $\mathbf{A}$. The modes $\boldsymbol{\Phi} = \mathbf{U}\mathbf{W}$ are referred to as *projected DMD modes*. To find a dynamic mode of $\mathbf{A}$ associated with a zero eigenvalue $\lambda_k = 0$, the exact formulation in (1.24) may be used if $\phi = \mathbf{X}'\mathbf{V}\boldsymbol{\Sigma}^{-1}w \neq 0$. Otherwise, the projected DMD formulation $\phi = Uw$ should be utilized [291].

With the low-rank approximations of both the eigenvalues and eigenvectors in hand, the projected future solution can be constructed for all time in the future. By first rewriting for convenience $\omega_k = \ln(\lambda_k)/\Delta t$, then the approximate solution at all future times is given by

$$\mathbf{x}(t) \approx \sum_{k=1}^{r} \boldsymbol{\phi}_k \exp(\omega_k t) b_k = \boldsymbol{\Phi}\exp(\boldsymbol{\Omega}t)\mathbf{b} \tag{1.25}$$

where b_k is the initial amplitude of each mode, $\mathbf{\Phi}$ is the matrix whose columns are the DMD eigenvectors $\boldsymbol{\phi}_k$, and $\mathbf{\Omega} = \text{diag}(\omega)$ is a diagonal matrix whose entries are the eigenvalues ω_k. The eigenvectors $\boldsymbol{\phi}_k$ are the same size as the state $\mathbf{x}$, and $\mathbf{b}$ is a vector of the coefficients b_k.

As discussed earlier, it is possible to interpret (1.25) as the least-square fit, or regression, of a best-fit linear dynamical system $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k$ for the data sampled. The matrix $\mathbf{A}$ is constructed so that $||\mathbf{x}_{k+1} - \mathbf{A}\mathbf{x}_k||_2$ is minimized across all snapshots.

It only remains to compute the initial coefficient values b_k. If we consider the initial snapshot $\mathbf{x}_1$ at time $t_1 = 0$, then (1.25) gives $\mathbf{x}_1 = \mathbf{\Phi}\mathbf{b}$. The matrix of eigenvectors $\mathbf{\Phi}$ is generically not a square matrix so that the initial conditions

$$\mathbf{b} = \mathbf{\Phi}^{\dagger}\mathbf{x}_1 \tag{1.26}$$

can be found using a pseudo-inverse. Indeed, $\mathbf{\Phi}^{\dagger}$ denotes the Moore-Penrose pseudo-inverse that can be accessed in MATLAB via the `pinv` command. The pseudo-inverse is equivalent to finding the best-fit solution $\mathbf{b}$ the in the least-squares sense.

A MATLAB implementation of the DMD algorithm (`DMD.m`) is provided in Code 1.3 below. The DMD algorithm presented here takes advantage of low dimensionality in the data in order to make a low-rank approximation of the linear mapping that best approximates the nonlinear dynamics of the data collected for the system. Once this is done, a prediction of the future state of the system is achieved for all time. Unlike the POD-Galerkin method, which requires solving a low-rank set of dynamical quantities to predict the future state, no additional work is required for the future state prediction outside of plugging in the desired future time into (1.25). Thus the advantages of DMD revolve around the fact that (i) it is an equation-free architecture, and (ii) a future state prediction is possible to construct for any time t (of course, provided the DMD approximation holds).

A key benefit of the DMD framework is the simple formulation in terms of well-established techniques from linear algebra. This makes DMD amenable to a variety of powerful extensions, including: multi-resolution (Chapter 5), actuation and control (Chapter 6), time-delay coordinates and probabilistic formulations (Chapter 7), noise mitigation (Chapter 8), and sparse measurements via compressed sensing (Chapter 9). Another important advantage of DMD is that it is formulated *entirely* in terms of measurement data. For this reason, it may be applied to a broad range of applications, including: fluid dynamics (Chapter 2), video processing (Chapter 4), epidemiology (Chapter 11), neuroscience (Chapter 12), and finance (Chapter 13).

ALGORITHM 1.1. **DMD function (`DMD.m`).**

```
function [Phi,omega,lambda,b,Xdmd] = DMD(X1,X2,r,dt)
% function [Phi,omega,lambda,b,Xdmd] = DMD(X1,X2,r,dt)
% Computes the Dynamic Mode Decomposition of X1, X2
%
% INPUTS:
% X1 = X, data matrix
% X2 = X', shifted data matrix
% Columns of X1 and X2 are state snapshots
% r = target rank of SVD
% dt = time step advancing X1 to X2 (X to X')
%
% OUTPUTS:
% Phi, the DMD modes
% omega, the continuous-time DMD eigenvalues
% lambda, the discrete-time DMD eigenvalues
% b, a vector of magnitudes of modes Phi
% Xdmd, the data matrix reconstrcted by Phi, omega, b

%% DMD
[U, S, V] = svd(X1, 'econ');
r = min(r, size(U,2));

U_r = U(:, 1:r); % truncate to rank-r
S_r = S(1:r, 1:r);
V_r = V(:, 1:r);
Atilde = U_r' * X2 * V_r / S_r; % low-rank dynamics
[W_r, D] = eig(Atilde);
Phi = X2 * V_r / S_r * W_r; % DMD modes

lambda = diag(D); % discrete-time eigenvalues
omega = log(lambda)/dt; % continuous-time eigenvalues

%% Compute DMD mode amplitudes b
x1 = X1(:, 1);
b = Phi\x1;

%% DMD reconstruction
mm1 = size(X1, 2); % mm1 = m - 1
time_dynamics = zeros(r, mm1);
t = (0:mm1-1)*dt; % time vector
for iter = 1:mm1,
    time_dynamics(:,iter) = (b.*exp(omega*t(iter)));
end;
Xdmd = Phi * time_dynamics;
```

1.3.1 ▪ Historical perspective on DMD algorithm

Historically, the original DMD algorithm [248] was formulated in context of its connection to Krylov subspaces and the Arnoldi algorithm. In this context, it is assumed that the data is sampled from a single trajectory in phase space, although this assumption has subsequently been relaxed in the general definition presented in § 1.2.1. For completeness, as well as understanding this connection, we present the DMD algorithm as originally presented by Schmid [248]. To illustrate the algorithm, consider the regularly spaced sampling in time:

$$\text{data collection times}: \quad t_{k+1} = t_k + \Delta t \tag{1.27}$$

where the collection time starts at t_1 and ends at t_m, and the interval between data collection times is Δt.

As before, the data snapshots are then arranged into an $n \times (m)$ matrix

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}(t_1) & \mathbf{x}(t_2) & \cdots & \mathbf{x}(t_m) \\ | & | & & | \end{bmatrix} \tag{1.28}$$

where the vector $\mathbf{x}$ are the n measurements of the state variable of the system of interest at the data collection points. The objective is to mine the data matrix $\mathbf{X}$ for important dynamical information. For the purposes of the DMD method, the following matrix is also defined:

$$\mathbf{X}_j^k = \begin{bmatrix} | & | & & | \\ \mathbf{x}(t_j) & \mathbf{x}(t_{j+1}) & \cdots & \mathbf{x}(t_k) \\ | & | & & | \end{bmatrix} \tag{1.29}$$

Thus this matrix includes columns j through k of the original data matrix.

To construct the appropriate Koopman operator that best represents the data collected, the matrix $\mathbf{X}_1^{m+1}$ is considered:

$$\mathbf{X}_1^m = \begin{bmatrix} | & | & & | \\ \mathbf{x}(t_1) & \mathbf{x}(t_2) & \cdots & \mathbf{x}(t_m) \\ | & | & & | \end{bmatrix}. \tag{1.30}$$

Making use of (1.6), this matrix reduces to

$$\mathbf{X}_1^m = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{A}\mathbf{x}_1 & \cdots & \mathbf{A}^{m-1}\mathbf{x}_1 \\ | & | & & | \end{bmatrix}. \tag{1.31}$$

Here is where the DMD method connects to Krylov subspaces and the Arnoldi algorithm. Specifically, the columns of $\mathbf{X}_1^m$ are each elements in a Krylov space. This matrix attempts to fit the first $m-1$ data collection points using the matrix $\mathbf{A}$ that approximates the Koopman operator. In the DMD technique, the final data point $\mathbf{x}_m$ is represented, as best as possible, in terms of this Krylov basis, thus

$$\mathbf{x}_m = \sum_{k=1}^{m-1} b_k \mathbf{x}_k + \mathbf{r} \tag{1.32}$$

where the b_k are the coefficients of the Krylov space vectors and $\mathbf{r}$ is the residual (or error) that lies outside (orthogonal to) the Krylov space. Ultimately, this best fit to the data using this DMD procedure will be done in an ℓ_2 sense using a pseudo-inverse.

In this notation, we have the key result (compare to the definition (1.17)):

$$\mathbf{X}_2^m = \mathbf{A}\mathbf{X}_1^{m-1} \tag{1.33}$$

where the operator $\mathbf{A}$ is chosen to minimize the Frobenius norm of $||\mathbf{X}_2^m - \mathbf{A}\mathbf{X}_1^{m-1}||_F$. In other words, the operator $\mathbf{A}$ advances each snapshot column in $\mathbf{X}_1^{m-1}$ a single timestep, Δt, resulting in the future snapshot columns in $\mathbf{X}_2^m$. The DMD outlined previously can then be enacted. This definition of DMD is similar to the more modern definition with $\mathbf{X} \to \mathbf{X}_1^{m-1}$, $\mathbf{X}' \to \mathbf{X}_2^m$ and the formula $\mathbf{A}\mathbf{X}_1^{m-1} = \mathbf{X}_2^m$ equivalent to (1.17). However, this formulation can be thought of more broadly and does not necessarily need to relate directly to (1.1).

1.4 ▪ Example code and decomposition

To demonstrate the DMD algorithm, we consider a simple example of two mixed spatio-temporal signals. In particular, our objective is to demonstrate the ability of DMD to efficiently decompose the signal into its constituent parts. To build intuition, we also compare DMD against results from principal component analysis (PCA) and independent component analysis (ICA).

The two signals of interest are

$$\begin{aligned} f(x,t) &= f_1(x,t) + f_2(x,t) \\ &= \text{sech}(x+3)\exp(i2.3t) + 2\text{sech}(x)\tanh(x)\exp(i2.8t). \end{aligned} \tag{1.34}$$

The individual spatio-temporal signals $f_1(x,t)$ and $f_2(x,t)$ are illustrated in Fig. 1.2(a)–(b). The two frequencies present are $\omega_1 = 2.3$ and $\omega_2 = 2.8$ with distinct spatial structures. The mixed signal $f(x,t) = f_1(x,t) +$

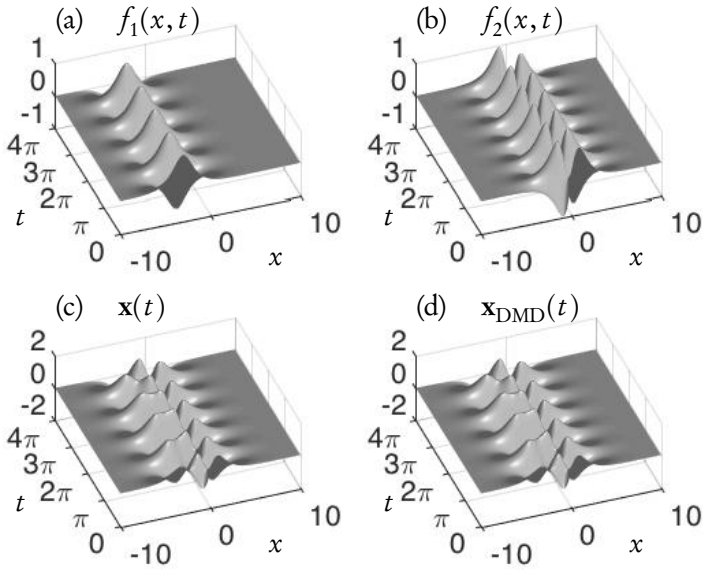


Figure 1.2. *An example of spatio-temporal dynamics of two signals (a) $f_1(x,t)$ and (b) $f_2(x,t)$ of (1.34) that are mixed in (c) $f(x,t) = f_1(x,t) + f_2(x,t)$. The function $f(x,t)$ can be represented instead by $\mathbf{x}(t)$. DMD of $\mathbf{x}$ was computed and a rank-2 approximate reconstruction (1.25) of the signal $\mathbf{x}_{DMD}(t)$ is shown in panel (d). The reconstruction is almost perfect, with the DMD modes and spectra closely matching those of the underlying the signals $f_1(x,t)$ and $f_2(x,t)$.*

$f_2(x,t)$ is illustrated in Fig. 1.2(c). The following code in MATLAB constructs the spatio-temporal signals.

ALGORITHM 1.2. **Mixing of two spatio-temporal signals.**

```
%% Define time and space discretizations
xi = linspace(-10,10,400);
t = linspace(0,4*pi,200);
dt = t(2) - t(1);
[Xgrid,T] = meshgrid(xi,t);

%% Create two spatio-temporal patterns
f1 = sech(Xgrid+3) .* (1*exp(1j*2.3*T));
f2 = (sech(Xgrid).*tanh(Xgrid)).*(2*exp(1j*2.8*T));

%% Combine signals and make data matrix
f = f1 + f2;
X = f.'; % Data Matrix
```

```
%% Visualize f1, f2, and f
figure;
subplot(2,2,1);
surfl(real(f1));
shading interp; colormap(gray); view(-20,60);
set(gca, 'YTick', numel(t)/4 * (0:4)),
set(gca, 'Yticklabel',{'0','\pi','2\pi','3\pi','4\pi'
    });
set(gca, 'XTick', linspace(1,numel(xi),3)),
set(gca, 'Xticklabel',{'-10', '0', '10'});

subplot(2,2,2);
surfl(real(f2));
shading interp; colormap(gray); view(-20,60);
set(gca, 'YTick', numel(t)/4 * (0:4)),
set(gca, 'Yticklabel',{'0','\pi','2\pi','3\pi','4\pi'
    });
set(gca, 'XTick', linspace(1,numel(xi),3)),
set(gca, 'Xticklabel',{'-10', '0', '10'});

subplot(2,2,3);
surfl(real(f));
shading interp; colormap(gray); view(-20,60);
set(gca, 'YTick', numel(t)/4 * (0:4)),
set(gca, 'Yticklabel',{'0','\pi','2\pi','3\pi','4\pi'
    });
set(gca, 'XTick', linspace(1,numel(xi),3)),
set(gca, 'Xticklabel',{'-10', '0', '10'});
```

This code produces the data matrix $\mathbf{X}$ of (1.10). $\mathbf{X}$ is the starting point for the decomposition algorithm of DMD, PCA and ICA. The following code implements DMD follows the steps outlined in the last subsection. In this particular case, we also perform a rank-2 decomposition to illustrate the low-dimensional nature of the decomposition.

ALGORITHM 1.3. **Perform DMD on data.**

```
%% Create DMD data matrices
X1 = X(:, 1:end-1);
X2 = X(:, 2:end);

%% SVD and rank-2 truncation
r = 2; % rank truncation
[U, S, V] = svd(X1, 'econ');
```

```
Ur = U(:, 1:r);
Sr = S(1:r, 1:r);
Vr = V(:, 1:r);

%% Build Atilde and DMD Modes
Atilde = Ur'*X2*Vr/Sr;
[W, D] = eig(Atilde);
Phi = X2*Vr/Sr*W;  % DMD Modes

%% DMD Spectra
lambda = diag(D);
omega = log(lambda)/dt;

%% Compute DMD Solution
x1 = X(:, 1);
b = Phi\x1;
time_dynamics = zeros(r,length(t));
for iter = 1:length(t),
    time_dynamics(:,iter) = (b.*exp(omega*t(iter)));
end;
X_dmd = Phi*time_dynamics;

subplot(2,2,4);
surfl(real(X_dmd'));
shading interp; colormap(gray); view(-20,60);
set(gca, 'YTick', numel(t)/4 * (0:4)),
set(gca, 'Yticklabel',{'0','\pi','2\pi','3\pi','4\pi'
    });
set(gca, 'XTick', linspace(1,numel(xi),3)),
set(gca, 'Xticklabel',{'-10', '0', '10'});
```

This code computes several important diagnostic features of the data, including the singular values of the data matrix $\mathbf{X}$, the DMD modes $\mathbf{\Phi}$, and the continuous-time DMD eigenvalues ω. These eigenvalues ω match the underlying frequencies exactly, where their imaginary components correspond to the frequencies of oscillation:

```
omega =
   0.0000 + 2.8000i
   0.0000 + 2.3000i
```

Following (1.25), a rank-2 DMD reconstruction $\mathbf{X}_{\text{DMD}}$ is possible, separating the data $\mathbf{X}$ into a sum of coupled spatio-temporal modes. This reconstruction is shown in Fig. 1.2(d). Fig. 1.3(a) shows the decay of singular values of $\mathbf{X}$, which reveals the data may be appropriately

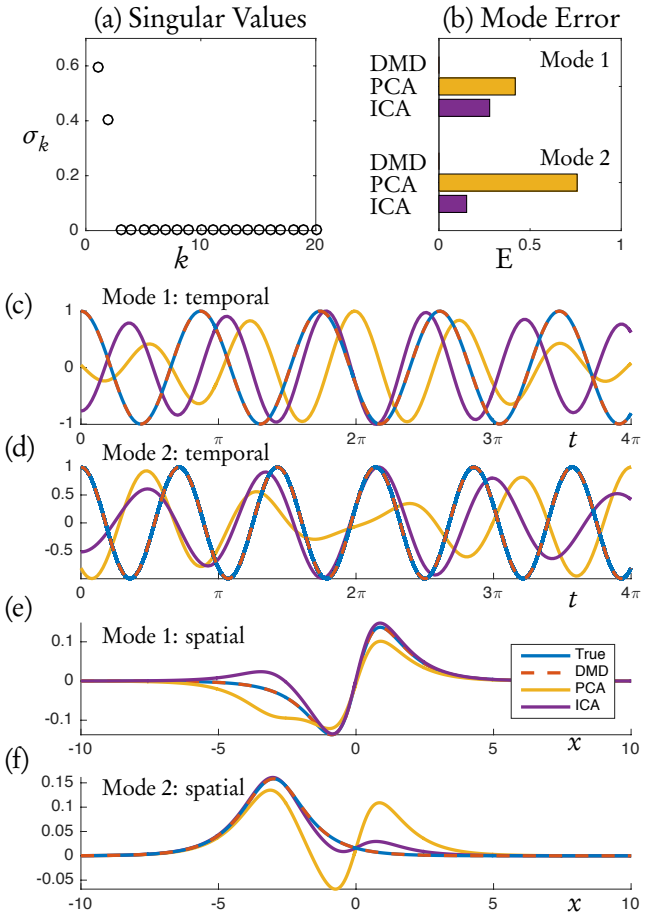


Figure 1.3. *A comparison of DMD with PCA and ICA as applied to example data generated with* (1.34)*. The singular values in (a) show that a rank-2 truncation is appropriate. Panels (c)–(f) compare the temporal and spatial aspects of the 2 modes, where the true modes are plotted along with modes extracted by DMD, PCA and ICA from the data. DMD produces results that are exactly (to numerical precision) aligned with the true solution. ICA modes approximate the true modes better than PCA does, but both deviate from the true solution. The ℓ_2-norm difference between the 2 true spatial modes and modes extracted by DMD, PCA and ICA are show in (b). DMD modes match the true modes exactly and have zero error..*

represented as rank $r = 2$. Fig. 1.3(a) compare the temporal and spatial modes as extracted by DMD with the true modes; these DMD modes almost exactly match the true solution modes $f_1(x,t)$ and $f_2(x,t)$.

To built intuition for DMD, we compare its against two commonly used modal decomposition techniques, PCA and ICA. The following code produces computes the PCA modes and their temporal evolution:

ALGORITHM 1.4. **PCA computed by SVD.**

```
[U, S, V] = svd(X);
pc1 = U(:, 1); % first PCA mode
pc2 = U(:, 2); % second PCA mode
time_pc1 = V(:, 1); % temporal evolution of pc1
time_pc2 = V(:, 2); % temporal evolution of pc2
```

In a similar fashion, we can also perform an ICA decomposition. There exists several implementations of ICA; here we use `fastica` from a MATLAB package [106].

ALGORITHM 1.5. **Fast ICA.**

```
[IC, ICt, ~] = fastica(real(X)');
ic1 = IC(1, :); % first ICA mode
ic2 = IC(2, :); % second ICA mode
time_ic1 = ICt(:, 1); % temporal evolution of ic1
time_ic2 = ICt(:, 2); % temporal evolution of ic2
```

The PCA and ICA extract modal structures contained in the data matrix $\mathbf{X}$; Fig. 1.3 shows these modes as compared to DMD modes and the true solution. PCA has no mechanism to separate the independent signals $f_1(x,t)$ and $f_2(x,t)$. It does, however, produce the correct rank-2 structure. The PCA modes produced in a rank-2 decomposition are chosen to maximize the variance in the data. They mix the two spatio-temporal modes as shown in Figs. 1.3(e) and (f). The time dynamics of the PCA modes are also shown in Fig. 1.3(c) and (d). The ICA results are much better than the PCA results since it attempts to account for the independent nature of the two signals [168]. The ICA modes and time dynamics are also shown in Figs. 1.3(c)–(f).

To make quantitative comparisons amongst DMD, PCA and ICA, Fig. 1.3(b) shows the ℓ_2-norm of the difference between the two true spatial modes and modes extracted by the three algorithms. The PCA modes have the largest error, while the DMD modes are accurate to nearly machine precision. The ICA error is less than half the error as-

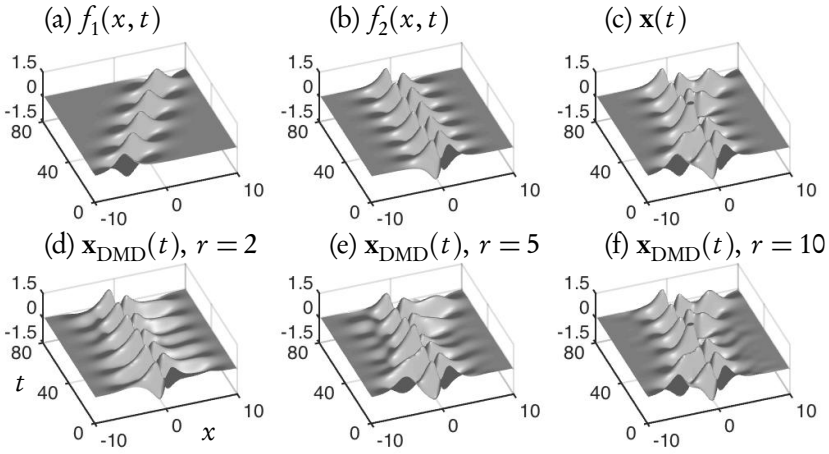


Figure 1.4. *An example of spatio-temporal dynamics with translation* (1.35) . *Panels (a)–(c) illustrate the real parts of the two signals and their mixture. Panels (d)–(f) show reconstructions of the signal* $\mathbf{x}_{DMD}(t)$ *using rank-2, rank-5 and rank-10 approximations. Although the dynamics are constructed from a two-mode interaction, the reconstruction now requires approximately 10 modes to get the right dynamics.*

sociated with PCA. This comparison shows the relative merits of DMD and its efficiency in decomposing spatio-temporal data.

1.5 ▪ Limitations of the DMD method

The DMD method, much like PCA, is based on an underlying singular value decomposition that extracts correlated patterns in the data. It is well known that a fundamental weakness of such SVD-based approaches is the inability to efficiently handle invariances in the data. Specifically, translational and/or rotational invariances of low-rank objects embedded in the data are not well captured. Moreover, transient time phenomena are also not well characterized by such methods. This will be illustrated in various examples that follow.

1.5.1 ▪ Translational and rotational invariances

To demonstrate the DMD algorithm and some of its limitations, we once again consider the simple example of two mixed spatio-temporal signals. In this case, however, one of the signals is translating at a con-

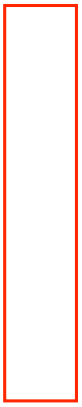
Solving Ax = b

In true data, we never have square matrices… Only:

short and fat matrices

infinity many solutions

tall and skinny matrices



No solution

1) add an extra constraint

min ll x ll2 such that Ax = b

Find the solution with the lowest L2 Norm!

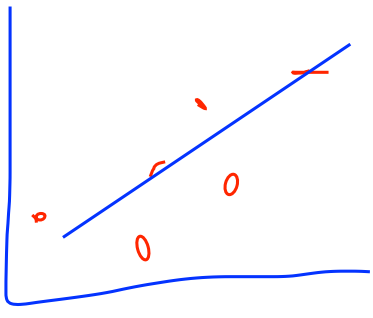
REGULARIZATION

1) min ll Ax - b ll + lambda f(x)

Some function which might capture another feature of some solution that we find desirable

---

Example: Curve fitting!



y = Ax + B

Least P-power solution:

Ep = (1/N sum(y(xi) - yi)^p) ^ 1/p

observed yvalue

predicted yvalue

p = 2 —> least squared (L2 norm)
p = 1 —> take absolute value = L1 Norm (city block/manhattan distances)

L2 Norms (and higher norms) are more susceptible to outliers than the L1 Norm. If I add a single outlier, it can drag the entire L2 fit line UP. Why does this matter? SVD and PCA rely on L2 Norms!

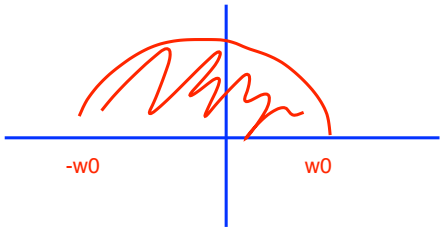
L1 gives us a concrete way of deciding how to remove outliers

CVX: Convex optimization package (for MATLAB; free to download!)

Many ways to solve Ax = b with infinitely many solution… what condition should we filter solutions by?

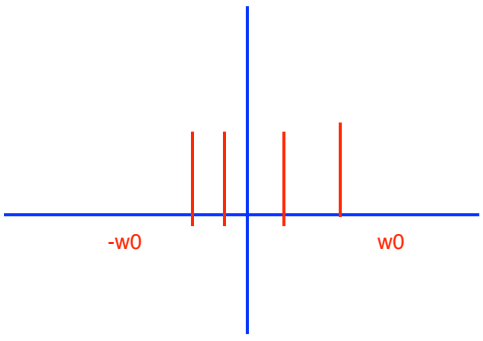
Compressive Sensing

Idea: To sense this frequency, won't we have to measure at twice the frequency of the highest frequency in the signal?
Assumes that our signal is CONTINUOUS



Continuous signals cover a range of frequencies

NOT TRUE FOR SPARSE SIGNALS!



Discrete signals ONLY occurring at a very limited number of frequencies.
This often occurs

DCT?

f = phi_c = sum(Cn phi_n)

i.e. we can expand our signal f as a sum of different fundamental signals from the basis Phi
When we have a sparse signal, we know that most of these coefficients Cn are ZERO!

Dont need to take a massive number of measurements if our system is low-dimensional!

b = M * f
= M Phi c

Where M is our “measurement matrix”; Each column is a row of the identity and tells us where we sample the true signal f. i.e. we only know the signal’s amplitude at a few number of points

Ax = b

Where most of our c’s are zero! What makes things sparse? Wavelets and Fourier Transforms!

our regularization (i.e. an L1 Norm)! The extra constraint we put on the solution to filter our solution space.

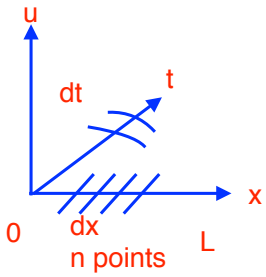
Important that sampling is RANDOM!

Reduced Order Models (ROMs)

Review: Solving a PDE

u_t = N(u, u_x, u_xx, …, x, t, Beta) —> nonlinear system that depends on both space and time; where _x = partial derivative wrt x

Discretize our system into n points; turn our system into a grid of ODEs?



uk = u(xk, t)

finite difference formulas for derivatives

u_x = (u(xk+1, t) - u(uk-1, t)) / 2dx

u_xx = (u(xk+1, t) -2u(xk,t) + (u(uk-1, t)) / dx^2

Local structure! All I care about is my neighbors around me. Easy to implement boundary conditions.

Another way to solve the PDE: Separation of Variables

u(x, t) = a(t) Phi(x) —> write our system as a function of space times a function of time separately.

u(x,t) = sum_k=1 to n [ak(t) Phik(x)]—> project into some basis where time and space are separate.

u_t = sum k=1 to n [d/dt ak(t) * Phik(x)] = N(u, u_x, u_xx, …, x,t, Beta)

< Phi_k, Phi_j> = integral from 0 to L of [Phi_k Phi_j*] dx —> = 1 if k=j and 0 otherwise

d/dt a(t)_k = <N, Phi_k> —> ODE! solve with ode45

Difference from above finite difference: global modes that span the whole space; how do these modes interact with each other. Spectral methods are often more accurate but harder to handle boundary conditions.

Issues? n is often very large if sufficient resolution is required: so how can I set up this problem to bring n down to a computationally reasonable level.

What if we use the ideal functional basis to expand our system? The best Phi_k(x); then I might not need as many modes (lower n) to accurately represent my system!

If we are lucky then we can describe our system in mostly linear terms

dot{u} = N(…) ~ Lu + epsilon N() with epsilon << 1 (weakly nonlinear)

Hopefully, L is strum-Liouville

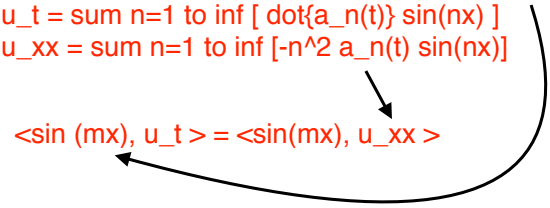
Other basis functions: FOurier, Hermile, Laguerre, Bessel Fn., Legendre, Spherical Harmonics

Example: heat equation!

u_t = u_xx with domain [0, 2pi] and boundary conditions u(0, t) = u(2pi, t) = 0

use basis expansion function Phi_n = sin (nx)

u = sum n=1 to inf [a_n(t) sin(nx)]
u_t = sum n=1 to inf [dot{a_n(t)} sin(nx)]
u_xx = sum n=1 to inf [-n^2 a_n(t) sin(nx)]



<sin (mx), u_t > = <sin(mx), u_xx >

all modes are zero exept when m = n

dot{a_m} = -m^2*a_m

ODE! easy to solve

a_m = ce^{-m^2t}

Each fourier modes is decaying in time by m^2t; Only the first few fourier modes matter as the others higher mode (higher m) modes decay so rapidly!

[5] A. Haar, "Zur Theorie der orthogonalen funktionen-systeme," Math. Ann. **69**, 331-371 (1910).

[6] T. F. Chan and J. Shen, *Image processing and analysis*, (SIAM, Philadelphia, 2005).

[7] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, (SIAM, Philadelphia, 1997).

[8] J. Shlens, "A tutorial on principal component analysis," (available on the web).

[9] P. Holmes, J. L. Lumley, and G. Berkooz, *Turbulence, Coherent Structures, Dynamical Systems, and Symmetry*, (Cambridge, 1996).

[10] J. P. Cusumano, M. T. Sharkady, and B. W. Kimble, "Dynamics of a flexible beam impact oscillator," Philos. Trans. R. Soc. London**347**, 421-438 (1994).

[11] B. F. Feeny and R. Kappagantu, "On the physical interpretation of proper orthogonal modes in vibrations," J. Sound Vibr. **211**, 607-616 (1998).

[12] T. Kubow, M. Garcia, W. Schwind, R. Full, R., and D. E. Koditschek, "A principal components analysis of cockroach steady state leg motion," (in preparation)

[13] R. Ruotolo and C. Surace, "Damage assessment of multiple cracked beams: numerical results and experimental validation." J. Sound Vibr. **206**, 567588 (1997).

[14] K. L. Briggman, H. D. I. Abarbanel, W. B. Kristan Jr. "Optical Imaging of Neuronal Populations During Decision-Making," Science **307**, 896 (2005).

[15] A. Chatterjee, "An introduction to the proper orthogonal decomposition," Current Science **78**, 808-817 (2000).

[16] J. Tenenbaum, V. de Silva and J. C. Langford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction," Science **290**, 2319-2323

[17] S. Roweis and L. K. Saul, "Linear EmbeddingNonlinear Dimensionality Reduction by Locally Linear Embedding," Science **290**, 2323-2326 (2000).

[18] K. Q. Weinberger and L. K. Saul, "Unsupervised Learning of Image Manifolds by Semidenite Programming," Int. J. Comp. Vision **70**, 77-90 (2006).

[19] A. Hyvärinen and E. Oja, "Independent Component Analysis: Algorithms and Applications," (available on the web)

[20] H. Farid and E. H. Adelson, "Separating reflections from images by use of independent component analysis," J. Opt. Soc. Am. A **16**, 2136-2145 (1999).