COMPUTATIONAL METHODS IN STATISTICS

Anuj Srivastava
Department of Statistics
Florida State University
Tallahassee, FL 32306

August 24, 2009

PREFACE

Statistics plays an important role in scientific studies, in a wide variety of applications. From the core statistical elements (observe, analyze and infer) to complicated design of experiments, statistics is finding newer roles in new areas. We believe that the most important reason behind advancement of statistics, and its applications in important and challenging problems, is the growth in computing power. The phenomenal advances in architectures supporting faster processing, and the simultaneous decline in cost of desktop computing have enabled diverse practitioners to use sophisticated statistical procedures. Applications with massive datasets (of the order of terrabytes) or real-time applications (requiring inferences in fractions of milliseconds) or applications with complicated probability models, can all employ reasonable scientific frameworks and still provide satisfactory results. In this course, we focus on the basic issues and standard practices in application of computational techniques in standard statistical analysis. The following outlines the basic philosophy behind this course:

1. The main focus is on ability to convert ideas into algorithms and practice their computer implementations. In this sense, this course complements the knowledge of statistical theory acquired in other mathematics/statistics courses.

2. Theoretical results, although important, are not emphasized here and instead the emphasis is on taking computational solutions and implementing them into algorithms and programs.

3. Through implementations and examples, we will learn, verify and appreciate various assumptions that are needed in deriving some theoretical results. One example is to establish the conditions under which Markov chains generate samples from given probability distributions.

4. This course is designed to be self-contained. Each chapter will start with the notations, definitions and the theory needed to set up computational goals. The rest of the chapter will deal with deriving algorithms and practice programs.

5. The main programming environment used here is matlab.

We will start with the representation of real number numbers of a digital storage device of a computer, called the floating point representation. This leads to the study of errors introduced in the computations due to floating point arithmetic. Understanding of these errors will guide us in selecting proper algorithms for computations of sample statistics (mean, variance) given the availability of memory, processing and storage resources.

The next topic is solving for (linear) regression between the observed vector and a set of predictor vectors. In Chapter 2, we will start by re-visiting some basic definitions and rules from linear algebra. Then, we will pose the problem of multiple linear regression and present a solution based on orthogonal transformations of underlying coordinate system, resulting in a simpler structure which is solvable using backward substitution. Linear representations also play an important role in analyzing high-dimensional systems. Problems in image analysis, meterology, atmospheric sciences, and fluid mechanics often undergo linear dimension reductions before any statistical approach is applied. We will investigate the use of principal component analysis and Fisher's discriminant analysis for such linear projections.

Even though linear solutions are important tools, most practical problems are characterized by non-linear relationships between the variables; the solutions correspond to optimal points under some chosen cost function that is non-quadratic. Chapter 3 is devoted to the techniques for finding the roots of score-functions, or equivalently, finding the optimal points of non-linear cost functions, through numerical procedures. There is a tremendous variety in non-linear optimization techniques, we will restrict our studies to a representative set. Applications of these ideas in studies include maximum likelihood estimation (MLE) and maximum a-posterior (MAP) estimation. We will consider an interesting case of finding MLE using only partial observations and expectation-maximization (EM) algorithm.

Several estimation and many other inference problems can be posed as integrals of appropriate functions. It should be emphasized that integration is basic to almost all statistical inference procedures. Very often, the integrands are far too complicated to seek analytical solutions; hence, numerical approximations are pursued, and Monte Carlo techniques have become prominent for this task. To understand and appreciate the role of Monte Carlo approaches, one should be familiar with the classical (deterministic) methods for numerical integration. In Chapter 4 we will study a set of numerical integration techniques for integrating functions of single and multiple variables.

One exciting area in statistics is computations through simulations. To calculate quantities associated with random variables, such as mean, variances etc., one can simulate those random variables and utilize the simulated values to approximate the quantities sought. With the advent of high-speed computing it is possible to perform sophisticated simulations in reasonable times. In Chapter 5 we will start by studying the simulation techniques for random variables, both discrete and continuous, following it up with some examples of simulating stochastic processes.

Chapter 6 applies random sampling techniques in estimating $\int g(x)f(x)dx$ where f is a probability density function, and g is an arbitrary (integrable) function. We will start with the classical formulation of Monte Carlo approach, using i.i.d samples of $X \sim f(x)$. Next, we describe several improvements in the classical approach using variance (error) reduction techniques.

In cases where it is not possible to exactly sample $X \sim f(x)$, one resorts to approximate sampling, and an important idea in that area is Markov chain based Monte Carlo (MCMC) approximation. In Chapter 7, we present a basic overview of MCMC techniques and, in particular, study the two classical techniques for constructing Markov chains: Gibb's sampler and the Metropolis algorithm. We study some asymptotic properties of the resulting Markov chains and investigate their applications in some estimation problems.

A problem that is exact opposite of sampling is that of density estimation. Given a number of samples $x_i \sim f$, the goal is to estimate f. In low-dimensional spaces, $\mathbb{R}^1$ or $\mathbb{R}^2$, there have been significant number of techniques that provide acceptable solutions, and in Chapter 8 we survey some of these techniques.

Chapter 7 deals with sampling from static or fixed distributions but many applications deal with analysis of dynamical systems where the probability distributions are changing in time. Consider, for example, the problem of tracking a system x_t that evolves in time according to some (known or unknown) law. Instead of observing x_t directly, we get to measure a related quantity y_t, and our goal is to estimate x_t using measurements at each observation time. In a Monte Carlo framework, this problem requires sampling from probability distribution of the type $f(x_t|y_1, y_2, \ldots, y_t)$ which changes at every t. Efficient, recursive techniques for sampling from such time-varying distributions is described in Chapter 10. Here we start with the classical Kalman filter and generalize to a nonlinear filtering approach using sequential Monte Carlo apparatus.

An important contribution of computational approaches in statistical analysis has been the invention of bootstrap methods. Bootstrap methods are very useful in situations when one wants to evaluate a statistical procedure or an estimator, an no explicit evaluation formula is available. Bootstrap is essentially a technique of resampling the observed data to generate enough replicates of the given estimator and use these replicates in computing standard error, or bias, or a confidence interval. These techniques are described in Chapter 11.

The final chapter includes a collection of topics that are important to any discussion on computational statistics. These topics, such as bootstrapping, perfect sampling, and dynamic programming, have been covered summarily in Chapter 12.

(This is a work in progress with regular updates, corrections, and additions. Please beware of a large number of unintended mistakes that are present in this document).

Contents

List of Figures

Chapter 1

NUMERICAL ANALYSIS

1.1 Introduction

There are many references suitable for this topic, we will follow the notation and discussion presented in [5]. In this chapter, we start the process of deriving computer algorithms and developing mathematical procedures to achieve desired mathematical results. This requires a basic understanding to how the numbers are represented on a digital machine, how the arithmetic operations are performed, what errors are possible and how to minimize errors in computations, and so on.

1.2 Numerical Methods

We start by making precise our usage of terms such as algorithms and numerical procedures.

Definition 1 *An algorithm is a step by step procedure that produces a solution to a problem in a finite number of steps.*

So it can be a sequence of statements resulting in, for example, an approximation of an integral or a precise exponential of a real number. There is not a precise syntax for writing algorithms. Any symbolic representation of the steps which clearly conveys the steps is sufficient. Having understood the mathematical theory behind a desired output generally one writes an algorithm which can later be translated into a computer program. Some of the famous algorithms that we will study include Newton-Raphson, Metropolis, Viterbi, and Householder algorithms.

Definition 2 *An algorithm for solving a problem whose solution consists of one or more numerical values is called a* **numerical procedure***.*

Consider the following example of a numerical procedure:

Example 1 *Finding $f'(x)$ if one just has sampled values of $f(x)$. A numerical procedure to approximate $f'(x)$ is as follows:*

```
Get $x$, the value at which $f'(x)$ is desired.
Initialize $h$, the variable step-size.
Get $r$, ratio of step-size in successive iteration.
Get NumDec, desired number of decimal accuracy.
$TOL \leftarrow 0.5 * 10^{-NumDec}$
Initialize $PrevDQ$
```

```
REPEAT
            $DQ \leftarrow [f(x+h) - f(x)]/h$
            $DelDQ \leftarrow DQ - PrevDQ$
            $PrevDQ \leftarrow DQ$
            $h \leftarrow h/r$
UNTIL $|DelDQ| < TOL$
```

Why should one use a program for numerical methods, instead of just using a calculating instrument of some kind?

1. **Ease of Implementation**: Once the program is written and debugged it can be run with a different numerical input every time. Such a computation can be unsupervised or automated. For example, in the example above replace $f(x)$ by a new function $g(x)$ to approximate its derivative, or replace x by y to approximate $f'(y)$. Furthermore, very often there are steps or sets of steps which are common to many different numerical procedures, so they need to be programmed only once.

2. **Flexibility**: Once a numerical procedure is programmed it is flexible enough to be modified or re-used. A simple re-organization of modules can result in a variety of numerical procedures. Smaller algorithms can be combined to generate larger and more complicated procedures. In fact, the strength of computational approaches lies in their heirarchical nature; building sophisticated programs using simple building blocks.

3. **Speed and Reliability**: Computer programs are much faster than a human being entering values on a calculator and performing arithmetic.

Numerical analysis is a study of numerical methods. The computations performed on digital computers are not exact: they produce side effects. Numerical analysis is a study of these side effects and constructing algorithms which minimize them.

1.3 Floating Point Calculus

Storage areas on digital computers have finite sizes, i.e. they can store the numbers only to some finite precision. Hence, not all real numbers are representable on digital computers, only a finite subset can be.

Definition 3 *The particular set of real numbers representable on a digital computer are called floating point numbers.*

The set of *floating-point* numbers plus the set of allowable arithmetic operations is called the *floating-point system.* Floating point system is not same for every computer, but it conforms to a set of standards approved by ANSI. A floating point number can be written as

$$(-1)^s (d_0.d_1 d_2 \ldots d_{t-1}) \times \beta^e \tag{1.1}$$

where s is the arithmetic sign, e is the exponent, $d_0, \ldots, d_{t-1}$ form the mantissa (also called fraction), and β is the base. s is either 0 or 1 while $0 \le d_i \le \beta - 1$ and $E_{min} \le e \le E_{max}$. The value of t dictates the precision to which a number can be stored on a computer. For a binary system $\beta = 2$ while for the decimal system $\beta = 10$. Additional numbers such as ∞, $-\infty$, and NaN (not a number) may also be represented. The smallest magnitude in this system is $x = 0.00\ldots1 \times \beta^{E_{min}}$. Another system often used to represent floating point numbers is

$$(-1)^s(.d_0d_1\ldots d_{t-1}) \times \beta^e) \ . \tag{1.2}$$

All the significant digits are after the decimal in this system.

It is clear that the same number can have multiple representations, for example 0.2008×10^4 and 2.0080×10^3 are exactly the same. To avoid multiple representations, a *normalized representation* is defined by forcing the first significant digit (d_0) should non-zero, i.e. $0 < d_0 \leq \beta - 1$. The numbers $\{x : |x| < \beta^{E_{min}-1}\}$ (called *subnormals*) can not be normalized and, are sometimes excluded from the system. If an operation results in a subnormal number *underflow* is said to have occurred. Similarly if an operation results in a number with the exponent $e > E_{max}$ *overflow* is said to have occurred. For a normalized floating point number we will use the notation

$$(\pm d_0d_1\ldots d_{t-1}, e) \tag{1.3}$$

to imply $(-1)^s(.d_0d_1\ldots d_{t-1}) \times \beta^e$. In short, it is called a β, t system. For example, in a $10, 3$ system 19 is represented by $(190, 2)$ and 0.003 is represented by $(300, -2)$. On a n-digit computer, a typical storage of a floating point number is as follows:

s	d_0	d_1	$\ldots$	d_{t-1}	e_1	$\ldots$	e_r

where $n = t + r + 1$.

Example 2 *Add the numbers 105.0, 3.14, and 2.47 in the* (β, t) *system with* $\beta = 10$ *and* $t = 3$. *For floating point operations, we often use double accuracy (*$t = 6$ *in this case) for computing and storing intermediate results.*

The representations with equal exponents are

$$105 \Rightarrow (105000, 3),\ \ 3.14 \Rightarrow (003140, 3),\ \ 2.47 \Rightarrow (002470, 3) \ .$$

First addition results in

$$(105000, 3) + (003140, 3) = (108140, 3) \ ,$$

which in $(10, 3)$ *system is* $(108, 3)$. *And the second addition is*

$$(108000, 3) + (002470, 3) = (110470, 3)$$

which in $(10, 3)$ *system is (110,3).*

1.4 Error Analysis

Knowing that only the floating point numbers can be represented on a digital computer, what are its consequences? Furthermore, their operations can result in non-floating point numbers which have to be represented by the nearest floating point number. This results in creation and propagation of errors in a numerical procedure.

There are two ways to generate a floating point approximation from a given real number: (i) truncation, only finite number of bits are kept to make it a floating point number, and (ii) round-off, the number is rounded-off to the nearest floating point number in the Euclidean sense. A real-valued number x is then represented by its floating point representation x^* such that

$$x^* = x(1+\epsilon) \ ,$$

where ϵ is the relative error. Based on the choice of β and t, we can put an upper bound on the absolute value of ϵ. Let x be denoted by $(.d_0d_1d_2\ldots d_{t-1}d_t\ldots)\times\beta^e$. Under (β,t) system its floating point representation becomes $(.d_0d_1\ldots d_{t-1})\times\beta^e$. Therefore, the error is

$$x - x^* = (.00\ldots 0d_td_{t+1}\ldots)\times\beta^e$$

And the absolute relative error becomes

$$\begin{aligned} |\frac{x-x^*}{x}| &= \frac{(.00\ldots 0d_td_{t+1}\ldots)\times\beta^e}{(.d_0d_1d_2\ldots d_{t-1}d_t\ldots)\times\beta^e} \\ &\leq \frac{1.0\times\beta^{-t}}{0.1} \\ &= \beta^{1-t} \end{aligned}$$

Similarly, in case of the roundoff error the absolute relative error can be shown to be upper bounded by $\frac{1}{2}\beta^{1-t}$. We will denote these quantities by $\mathbf{u}$ which represents the precision of a computer in representing real numbers.

$$\mathbf{u} = \begin{cases} \beta^{1-t} & \text{truncation} \\ \frac{1}{2}\beta^{1-t} & \text{roundoff} \end{cases} \qquad (1.4)$$

The errors in representing real numbers accumulate further under the floating point arithmetic. There are four main sources of error in floating point arithmetic: let $fl(x\circ y)$ represent a floating point operation involving the two floating point number x and y.

1. **Creeping roundoff/truncation**: This is the situation when $fl(x\circ y)$ is not a floating point number and it is approximated via round-off or truncation. Let x^* and y^* be the floating point representations of x and y, respectively. Then, their floating point addition is given by

$$\begin{aligned} (x^*+y^*)^* &= (x^*+y^*)(1+\epsilon_3) \\ &= (x(1+\epsilon_1)+y(1+\epsilon_2))(1+\epsilon_3) \\ &= (x+y)+(x+y)\epsilon_3+(x\epsilon_1+y\epsilon_2)+(x\epsilon_1+y\epsilon_2)\epsilon_3 \\ &\leq (x+y)+2(|x|+|y|)\mathbf{u}+ \ \text{higher order terms} \end{aligned}$$

   So floating point addition of x and y results in cumulative errors resulting from their individual representations and the floating point operation performed on them.

2. **Negligible Addition**: In case the magnitudes of x and y are very different, their sum gets rounded off to the larger of x and y. For example, if we add the numbers 1.23×10^4 and 0.1 in $\beta=10$ and $t=3$ system, we get: $(123,5)+(100,0)=(123,5)$ even if intermediate values are stored in $\beta=10$ and $t=6$ format.

3. **Error Magnification**: Let $\circ$ denote the multiplication, and if there is an error incurred in representation of x, that error will be magnified by the value of y.

4. **Subtractive Cancellation**: If the two numbers x and y have large number of significant digits and their difference has a relatively smaller number of significant digits, then the difference can lose significant digits in floating point computation. For example, let $x = 122.9572$ and $y = 123.1498$, their floating point representations are, for $t = 4$, $\beta = 10$, $(1230, 3)$ and $(1231, 3)$. The resulting difference is -0.1 while the actual answer is -0.1926. Similarly, in the calculation of e^{-x} by the partial sums, for large x's the successive terms can have subtractive cancellation.

Some of these errors are unavoidable. Sometimes they can be minimized. There are some general guidelines for minimizing some specific errors in floating point arithmetic:

1. In case of integral exponents such as x^n with n small, it advisable to compute this value through iterative multiplication rather than the exponential evaluation. That is, to calculate y^n, for n integer, use $u = y * y$ and then $v = u * u$, and so on.

2. Very often one can change expressions in the form which requires fewer floating point operations, specially in trigonometric expressions. Furthermore, some expressions are more stable as compared to others even though they are algebraically equivalent. For example, evaluating the expression $1 + \cos(x)$ will result in subtractive cancellation for the values closer to π. Instead, one can evaluate the equivalent expression $2(\cos(\frac{x}{2}))^2$ which avoids the issue of subtractive cancellation.

3. Use nested multiplication to compute polynomials. For example, to compute $y = a_1 x^n + a_1 x^{n-1} + \ldots + a_n x + a_{n+1}$ use the loop:
```
$y = 0$;
for $i = 1 : n + 1$
    $y = y \cdot x + a_i$;
end
```
The total number of floating point operations performed in this evaluation is $2n$, an addition and a multiplication in each iteration.

4. Sometimes precision can be improved significantly by storing the intermediate values in extended precision. For a $float$ evaluation of a function the intermediate values can be stored using *double* precision. This double precision register model is widely used in computers.

Based on this discussion, evaluating $f(x)$ for a real-valued number will have two types of errors:

$$f^*(x^*) - f(x) = (f(x^*) - f(x)) + (f^*(x^*) - f(x^*)) \ .$$

The first error is due to the representation of x while the second term is the error in representing $f(x^*)$ in a floating point system. Irrespective of the function and its arguments, the relative error in the second term is bounded the machine precision $\mathbf{u}$ while the first term can be significantly high. Some functions will have the first term much larger than others. In fact, the same function can have the first term much larger at some values of x compared to some other values. This motivates a study of the *stability* in evaluating a function at a given point.

Definition 4 *A function is called ill-conditioned at a given point x if a small perturbation in x causes a large perturbation in $f(x)$. Its called well-conditioned otherwise.*

As an example consider the function $f(x) = \frac{1}{x}$ being evaluated at a point $x \neq 0$. Then,

$$\begin{aligned} f^*(x^*) &= f^*(x(1+\epsilon_1)) \\ &= \frac{1}{x(1+\epsilon_1)}(1+\epsilon_2) \\ &= \frac{1}{x}\frac{(1+\epsilon_1)}{(1+\epsilon_2)} \end{aligned}$$

For small values of ϵ_1 and ϵ_2 the factor $\frac{(1+\epsilon_1)}{(1+\epsilon_2)}$ is nearly 1 and the function is well-conditioned. Similar discussion suggests that for $x \sim 0$ ($x \neq 0$) this function is ill-conditioned. On the other hand consider the function

$$f(x) = \frac{1.01+x}{1.01-x} .$$

In this case

$$f^*(x^*) = \frac{1.01+x+x\epsilon_1}{1.01-x-x\epsilon_1}(1+\epsilon_2) .$$

For $x = 1$,

$$f^*(x^*) = \frac{2.01+\epsilon_1}{0.01-\epsilon_1}(1+\epsilon_2) .$$

Even if $\epsilon_2 = 0$, for $\epsilon_1 = 0$ $f^*(x^*) = 201$ and for $\epsilon_1 = 0.005$, $f^*(x^*) = 403$. Hence, f is ill-conditioned near $x = 1$.

To quantify the stability of a function being evaluated at a given point we utilize the notion of a **condition number**.

Definition 5 *$C(x)$ is called the condition number of a function f at a given point x if it satisfies the condition*

$$|\textit{relative change in } f(x)| \sim C(x)|\textit{relative change in } x| . \tag{1.5}$$

For a large value of $C(x)$, more accurate representation of x is needed to get the fixed accuracy in representing $f(x)$. Clearly, $C(x)$ relates to the slope of f at x. Large values of C imply an ill-conditioned function and vice-versa. Let f be differentiable at x, then for $|dx| << |x|$,

$$f(x+dx) \sim f(x) + f'(x)dx$$

Rearranging the terms, we obtain, for $x,\ f(x) \neq 0$,

$$\frac{f(x+dx)-f(x)}{f(x)} \sim \frac{xf'(x)}{f(x)}\frac{dx}{x} .$$

From the definition in Eqn. 1.5, the condition number is given by,

$$C(x) = |\frac{xf'(x)}{f(x)}| .$$

For the function $f(x) = \exp(x)$, the condition number is given by:

$$C(x) = |\frac{x\exp(x)}{\exp(x)}| = |x| ,$$

and hence this function is ill-conditioned for x with large magnitude. One should be careful before computing exponential of large numbers.

1.5 Moment Computations

A very frequent requirement in statistics is the computation of sample moments based on a finite observation set. In this section we investigate the error propagation resulting in a floating point system. Let $x_1, x_2, \ldots, x_n$ are the observations then denote the j-th sample moment by

$$M_j = \frac{1}{n}\sum_{i=1}^{n}(x_i)^j \ . \tag{1.6}$$

The mean is given by $\bar{x} = M_1$ and the central moments are given by $m_j = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^j$. m_2 is the variance and $\sqrt{m_2}$ is the standard deviation.

1.5.1 Computing Sample Mean

First we analyze the error incurred in evaluating means, the common algorithm is:

Algorithm 1 (Sample Mean)

```
$s = 0;$
$for\ j = 1 : n$
        $s = s + x(j);$
$end$
$m = s/n;$
```

Let $n = 3$ and compute $f = x_1 + x_2 + x_3$ where x_1, x_2, x_3 are floating point numbers. Each addition in the algorithm can result in a number which may not a floating point. Its floating point representation results in an error denoted by multiple $(1+\epsilon)$ to that number. That is,

$s_1 = (x_1 + 0)^* = (x_1 + 0)(1 + \epsilon_1)$
$s_2 = (s_1 + x_2)^* = (s_1 + x_2)(1 + \epsilon_2)$
$s_3 = (s_2 + x_3)^* = (s_2 + x_3)(1 + \epsilon_3)$
$\sum^* = (x_1 + x_2 + x_3) + x_1(\epsilon_1 + \epsilon_2 + \epsilon_3) + x_2(\epsilon_2 + \epsilon_3) + x_3(\epsilon_3) +$ higher order terms.

Therefore, the cumulative error is given by

$$\begin{aligned} |\overset{*}{\sum} - \sum| &\sim |x_1(\epsilon_1 + \epsilon_2 + \epsilon_3) + x_2(\epsilon_2 + \epsilon_3) + x_3(\epsilon_3)| \\ &\leq |x|(|\epsilon_1| + |\epsilon_2| + |\epsilon_3|) + |y|(|\epsilon_2| + |\epsilon_3|) + |z||\epsilon_3| \\ &\leq (3|x_1| + 2|x_2| + |x_3|)\mathbf{u} \end{aligned}$$

where $\mathbf{u}$ is the machine precision. This result shows that the error can be more if x_1 is the largest of the three as opposed to the case when x_1 is the smallest of the three. This suggests that one should sort the numbers according to their magnitudes in increasing order before computing their mean. Another advantage of sorting the numbers before the summation is that it helps avoid negligible addition. It is possible to have two numbers say x_1 and x_2 which differ a lot in magnitude. Their addition will result in loss of significant digits associated with the smaller number. But through sorting it is possible that the smaller numbers accumulate first and become more comparable to larger numbers before their addition. Obviously, there is extra time spent in sorting the numbers ($O(n\log(n))$ operations) so there is a trade-off between the speed of implementation and the resulting accuracy.

Extending this calculation to an arbitrary n, it can be shown that

$$|\sum_i^* x_i - \sum_i x_i| \leq (\sum_{i=1}^n (n+1-i)|x_i|)\mathbf{u} \tag{1.7}$$

Dividing both sides by n, we have

$$\begin{aligned}|f^* - f| &\leq \frac{1}{n}(\sum_{i=1}^n (n+1-i)|x_i|)\mathbf{u} + \bar{x}\epsilon \\ &\leq |x_{max}|\mathbf{u}\frac{1}{n}\sum_{i=1}^n (n+1-i) + |x_{max}|\mathbf{u} \\ &\leq \frac{1}{2}(n+3)|x_{max}|\mathbf{u}\end{aligned}$$

This is the worst case error that can be introduced in the process of computing mean of n floating point numbers. Note that this error grows linearly in n.

Neely proposed an algorithm to reduce this error. Here is the algorithm

Algorithm 2 (Neely's Sample Mean)

```
$S = 0$;
for $j = 1 : n$
        $S = S + x(j)$;
end
$\bar{x} = \frac{S}{n}$;
$S = 0$;
for $j = 1 : n$
        $S = S + (x(j) - \bar{x})$;
end
$\bar{X} = \bar{x} + \frac{S}{n}$;
```

Basically, the first *for* loop computes the mean as earlier but the second run through the loop does finer corrections in computing the mean. In ideal calculation S should be 0, but for floating point system S is of the order of the linear term in summation we calculated earlier, that is

$$S \sim \left(\sum_{i=1}^n (n+1-i)x_i\right)\mathbf{u}\ .$$

1.5.2 Computing Variance

Variance can be calculated in many ways, two of them according to the formulae,

$$\frac{1}{n}\sum_{i=1}^n (x_i - \bar{x})^2 = \frac{1}{n}\sum_{i=1}^n x_i^2 - (\frac{1}{n}\sum_{i=1}^n x_i)^2\ .$$

These two lead to the following two algorithms for computing the variance. The natural implementation of the variance formula (left side of the equation) results in a **two-pass algorithm**.

Algorithm 3 (Two-Pass Variance)

```
$s = 0;$
$for\ i = 1 : n$
        $s = s + x(i);$
$\bar{x} = s/n;$
$end$
$s = 0;$
$for\ i = 1 : n$
        $s = s + (x(j) - \bar{x})^2;$
$end$
$var = s/n;$
```

The other algorithm, utilizing only **one pass** through the data is based on the right side of the equation:

Algorithm 4 (One-Pass Variance)

```
$s = 0;$
$s2 = 0;$
$for\ j = 1 : n$
        $s = s + x(j);$
        $s2 = s2 + x(j) * x(j);$
$end$
$var = s2/n - (s/n)^2;$
```

It can be seen that the algorithm 1 is better. Why? The reason is that the cancellation of significant digits takes place when we take the difference of two large numbers. For example, consider computing the variance of the dataset $X = (19, 20, 21)$. The mean is $\bar{x} = 20$ and the variance is $2/3$. The computations according to Algorithm 1, in a $\beta = 10$ and $t = 3$ system, will result in:

$$\begin{aligned}
(-100, 1)^2 &= (100, 1) \\
(000, 1)^2 &= (000, 1) \\
(100, 1)^2 &= (100, 1) \\
&= (200, 1)
\end{aligned}$$

and the variance becomes $(200, 1)/3 = (667, 0)$. On the other hand, the second algorithm results in:

$$\begin{aligned}
(190, 2)^2 &= (361, 3)\ [+(000, 1) = (361, 3)] \\
(200, 2)^2 &= (400, 3)\ [+(361, 3) = (761, 3)] \\
(210, 2)^2 &= (441, 3)\ [+(761, 3) = (1202, 4)] = (120, 4)\ .
\end{aligned}$$

The first term is given by $(120, 4)/3 = (400, 3)$. The second terms is given by $(200, 2)^2 = (400, 3)$. The variance is given by their difference,

$$(400, 3) - (400, 3) = 0\ .$$

This algorithm demonstrates that significant digits may be lost in using the second algorithm. Why is the two-pass algorithm better? In that case we avoided subtracting cancellation, i.e. subtracting close, large numbers, by centering them, and hence, the improvement in the performance.

To analyze the errors incurred in computing the variance according to Algorithm 2, define the coefficient of variation (CV), according to

$$CV = \sqrt{\frac{S}{n\bar{x}^2}} = \sqrt{\frac{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2}{n\bar{x}^2}} = \frac{\text{standard deviation}}{\text{mean}} \ .$$

It can be shown that if $(CV)^2$ is less than the machine precision then all the significant digits will be lost. Another quantity which is used to analyze the algorithmic performance is the condition number κ. It bounds the amount by which the relative errors introduced into the data set are magnified in the variance computations. If κ is large the data set is ill-posed. If the error in number representation is less than $\mathbf{u}$, then the error in variance calculation is lass than κu. Also,

$$\kappa = \sqrt{1 + (CV)^{-2}} \sim CV^{-1} \ .$$

Analyzing the two algorithms, the errors are bounded as follows. In algorithm 1, the error in S are bounded by $nu + n^2\kappa^2\mathbf{u}^2$, while in Algorithm 2 the error is bounded by $n\kappa^2\mathbf{u}$. If we use $y_i = x_i - \bar{x}$ instead of x_i then the error in computing S is bounded by $n\mathbf{u} + \kappa^2(n\mathbf{u})^3$. If the addition in computing the mean is performed pairwise, the replace n by $log_2 n$ in these bounds.

Provisional Centering

For any constant g, the variance is given by

$$\sum_{i=1}^{n}(x_i - \bar{x})^2 = \sum_{i=1}^{n}(x_i - g)^2 - n(\bar{x} - g)^2 \ . \tag{1.8}$$

One can utilize this formula to compute the variance more accurately and still have only one pass through the data. This technique is called *provisional centering*. It retains the speed of one pass algorithm and still seeks the accuracy of the two-pass algorithm. What should be the value of g? g should be a measure of central tendency in the data, ideally the mean $\bar{x}$ or the median. In the notation of Eqn. 1.8, the one-pass algorithm implements the case $g = 0$, and the two pass algorithm implements the case $g = \bar{x}$. But computing $\bar{x}$ requires a pass through the data and the procedure becomes identical to Algorithm 1. A simple choice is to select any x_i, say x_1, and use it as g. But this g may not be a good representative of the data. A better technique is to do **adaptive centering**. In this technique, one starts with an initial value of g but changes it in the loop as and when certain conditions are met. At every step k, the quantity to checked for the condition

$$|\sum_{i=1}^{k}(x_i - g)| >> 0 \ ,$$

or, change g when $|\sum_{i=1}^{k}(x_i - g)|$ exceeds a certain predefine quantity E. At that moment, the current value of g gets substituted by the current sample mean $\tilde{g} = \frac{1}{k}\sum_{i=1}^{k} x_i$. This substitution will also require a correction term to account for changing g in the middle of the pass. Let g gets replaced by $\tilde{g}$ at the kth step of the loop. Then, the current sum of squares gets corrected according to the equation,

$$\sum_{i=1}^{k}(x_i - \tilde{g}) \ = \ \sum_{i=1}^{k}(x_i - g) + k(g - \tilde{g}) \tag{1.9}$$

$$\sum_{i=1}^{k}(x_i-\tilde{g})^2 \quad = \quad \sum_{i=1}^{k}(x_i-g)^2 + k(\tilde{g}^2-g^2) - 2(\tilde{g}-g)\left((\sum_{i=1}^{k}(x_i-g)) + kg\right) . \tag{1.10}$$

An algorithm to implement this strategy is as follows:

Algorithm 5 (Adaptive Centering Variance)

```
$s1 = 0;$
$s2 = 0;$
$g = x(1);$
$i = 0;$
while $(i < n)$
    while $(i < n$ & $abs(s1) < E)$
        $i = i + 1;$
        $s1 = s1 + (x(i) - g);$
        $s2 = s2 + (x(i) - g) * (x(i) - g);$
    end
    if $i < n$
        $gn = (s1/i) + g;$
        $s2 = s2 + i * (gn^2 - g^2) + 2 * (gn - g) * (s1 + i * g);$
        $s1 = 0;$
        $g = gn;$
    end
end
$m = (s1/n) + g;$
$var = s2/n - (s1/n)^2;$
```

Note that if $\tilde{g}$ is the current mean, then the new value of $s1$ is zero!

1.6 Inner Products

Computing inner products in finite dimensional vector spaces is given by

$$< a, b >= \sum_{i=1}^{n} a_i b_i$$

The possibility of cancellation error is large in these computations. The common techniques to avoid these errors are: (i) use double precision, and (ii) use provisional centering.

1.7 Problems

1. Let $f(x) = \frac{(\sum_{i=1}^{n} X_i)}{n}$. Let f^* be the floating point implementation of this equation. Show that
$$|f^* - f| \le \left(\sum_{i=0}^{n-1} (1 - \frac{i}{n}) |X_{i+1}| + |\bar{X}| \right) \mathbf{u} \, .$$

2. Prove that
$$|f^* - f| \le \left(\sum_{i=0}^{n-1} \frac{1}{2} (n+3) |X_{max}| \right) \mathbf{u} \, .$$

3. Consider the following three functions:
$$f_1(x) = \frac{1 - cos(x)}{x^2}, \; f_2(x) = \frac{(sin(x)/x)^2}{1 + cos(x)}, \; f_3(x) = 2(\frac{sin(x/2)}{x})^2 \, .$$

   (a) Start with $x = 0.1$ and calculate each function for that x, then choose $x = 0.01$ and so on.

   (b) Now start with $x = \pi + 0.01$ and calculate each function for that x, then choose $x = \pi + 0.001$ and so on.

   Tabulate your results, underline the digits which are wrong.

4. For our computers $\beta = 2$ but t can differ. Here is an algorithm to find t:
   Algorithm 6 $x = 1.5; u = 1.0; t = 0; \alpha = 1.0;$
```
while   $x > \alpha$
    $u = u/2;$
    $x = \alpha + u;$
    $t = t + 1;$
end
```

   Find t for your computer. Why is α chosen to be 1.0 ?

5. Using $\beta = 10$ and $t = 4$, find the floating point representation of the following numbers using truncation. Also calculate their relative errors.
$$122.9572, \quad 123.1498, \quad 0.0014973, \quad 457932$$

6. For the following algorithm:
   Algorithm 7 $s = 0;$
```
for   $j = 1 : n$
    $s = s + x(j);$
end
$m = s/n;$
$s = 0;$
for   $j = 1 : n$
    $s = s + (x(j) - m);$
end
$m = m + s/n;$
```

Show that algebraically m is equal to the mean of $x(j)$'s.

7. For $\beta = 10$ and $t = 4$, and for $x = (5298, 0)$, $y = (8012, 0)$ and $z = (6024, 0)$ show that

$$((x + y)^* + z)^* \neq (x + (y + z)^*)^* \ .$$

That is, through this example show that the floating point addition may not be associative.

8. Write a matlab program which sorts a given set of numbers in an increasing order using the following algorithm:

```
for $i = 1 : n - 1$
    for $j = 1 : n - 1$
        if $(x(j) > x(j + 1))$
            $t = x(j);$
            $x(j) = x(j + 1);$
            $x(j + 1) = t;$
        end
    end
end
```

9. For the following data:

3078.14 0.0034 12.45 4.0586 245.45 0.234 987.1 56.0987 7.031 100.02

44.156 2.679 962.0162 96.0 108.0123 1209.04

(a) Write a matlab program to calculate the mean using the regular algorithm.

(b) Write a matlab program which first sorts the numbers and then calculates their mean using the regular algorithm.

(c) Write a program to calculate the mean using pairwise addition.

10. For a quadratic equation $ax^2 + bx + c = 0$, the two roots are given by:

$$r_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \ .$$

In situations where $b^2 >> 4ac$, the discriminant becomes $\sqrt{b^2 - 4ac} \sim \sqrt{b^2} = |b|$. Hence, if b is positive $-b + |b|$ can have subtractive cancellation. If b is negative then $-b - |b|$ can be erroneous. To avoid this situation, an alternative algorithm is:

```
$b_1 = \frac{-b}{2a};$
$c_1 = \frac{c}{a};$
$d = b_1^2 - c_1;$
$r_1 = abs(b_1) + sqrt(d);$
if $b_1 < 0$
        $r_1 = -r_1;$
end
if $r_1 \neq 0$
        $r_2 = \frac{c_1}{r_1};$
end
```

Write a matlab program using this algorithm to find the roots of $x^2 + 1000.001x + 1 = 0$.

11. Write a matlab program to implement the adpative one-pass strategy for computing the variance of a given sequence of numbers. Test your program on a random sequence of 1000 numbers.

1.8 References

The material presented in this chapter is taken mainly from the texts [5, 9, 2].

Chapter 2

NUMERICAL LINEAR ALGEBRA

2.1 Introduction

Using observations either from controlled or observational experimats, to discover relationships between variables of interest, is an important component of applied statistics. For instance, we may be interested in studying the effect of a number of treatments on a response variable. As an example, in a meterological experiment we may be interested in the effect of the moisture level and wind velocity (independent variables) on the probability of rain (dependent variable)? Or in a medical study, we may be interested in the effect of a particular drug treatment on some cancer patients? Clearly, in most practical situations these relationships are quite complicated. To understand the problem formulation and the solution, one starts by simplifying to the stage where the response variables are related to dependent variables (or the treatments) in a linear fashion. Once the tools to infer linear relationships are derived then these techniques can be expanded to more interesting non-linear situations. Clearly, such techniques will apply to a variety of areas ranging from meteorology to mechanical control systems to medical diagnosis.

Let $x_1, x_2, \ldots, x_n$ be a set of independent variables and y be a variable of interest which depends upon the values of x_i's. For example, x_i's may correspond to temperature, atmospheric pressure, wind velocities, etc. in a region and y may correspond to the amount of rainfall in that region. Furthermore, assume that the relationship between x_i's and y is linear, that is,

$$y = \sum_{i=1}^{n} b_i x_i + \epsilon$$

where b_i's are scalar constants and ϵ represents the residual error in the model. Very often ϵ is assumed as a random variable and, in particular, a Gaussian random variable. The coefficients b_i's are unknown and they have to be estimated using the observed values of x_i's and y. One can design an experiment in which several sets of measurements of the variables x_i's and y are taken, and then analyzed to estimate the coefficients b_i. For example, the observations may belong to different sample times $t_1, t_2, \ldots, t_m$ such that

$$y(t_j) = \sum_{i=1}^{n} b_i x_i(t_j) + \epsilon(t_j) \tag{2.1}$$

In a matrix form these equations can be restated as

$$\mathbf{y} = Xb + \epsilon, \quad \mathbf{y} = [y(t_1)\ y(t_2)\ \ldots y(t_m)]^T$$

where $\mathbf{y} \in I\!R^m$, $X \in I\!R^{m \times n}$, $b \in I\!R^n$, and ϵ is a random vector of size m. In this notation, the problem can be stated as follows: given X and $\mathbf{y}$ what is the best estimate of b and how to compute it? The study of this equation and its solutions is called *multiple regression analysis.*

2.2 Multiple Regression Analysis

Solving a linear system of equations is common to many problems, including multiple regression analysis as stated earlier. We take a least squares approach to understanding multiple regression analysis. Restating the problem, let $\mathbf{y}$ be a set of observations and X be a matrix of independent variables of predictors, the goal is to find the weight vector b such that

$$(y_i - \hat{y}_i)^2$$

is minimized (or such that $\hat{\mathbf{y}} = Xb$). b is also called the vector of regression coefficients. A similar problem formulation arises in the maximum likelihood estimation of b based on the observation model

$$\mathbf{y} = Xb + \epsilon \tag{2.2}$$

where b is deterministic but unknown and $\epsilon \sim N(0, \sigma^2 I)$ can be thought of as measurement errors. Given X, and having observed $\mathbf{y}$ the maximum likelihood estimate of b is

$$\hat{b}_{ML} = \operatorname*{argmax}_b \frac{1}{(2\sigma^2)^{m/2}} \exp\{\frac{-1}{2\sigma^2}\|\mathbf{y} - Xb\|^2\}$$

In case $m = n$ and X is invertible, the least square solution (or maximum likelihood estimation) is relatively straightforward. $\hat{b}$ is given by

$$\hat{b} = X^{-1}\mathbf{y} \ .$$

In case $m \neq n$ and $X^T X$ is non-singular, the solution is given by

$$\hat{b} = (X^T X)^{-1} X^T \mathbf{y} \ .$$

The term $(X^T X)^{-1} X^T$ is called the **pseudo-inverse** of X. Computing this pseudo-inverse involves inner-products, and therefore, the error analysis for inner products is applicable. Furthermore, computing inverse of a matrix is order $O(n^3)$ operation which is computationally expensive. We seek another approach to solve this problem without resorting to compute the inverse of a matrix. This approach is based on transforming a given system of equations in to a relatively simple form and then solving the simpler system, as described in the next section.

In addition to finding a least squares estimate of the vector b, we want to quantify how good the linear model is. That is, how good is the assumption that $\mathbf{y}$ and X are related through the linear system stated in Eqn. 2.2. To quantify the goodness of fit of the regression model, the following quantities are used:

1. the residual sum of squares (RSS), alos called the sum of squres of error (SSE), is given by:
$$SSE = \|\mathbf{y} - X\hat{b}\|^2 \ .$$
2. For $m > n$, the residual variance, an estimate of σ^2, given by:
$$\text{residual variance } (s^2) = \frac{SSE}{(m-n)} \ .$$
Recall that σ^2 is the variance of the elements of ϵ in the regression model.

3. an estimate of the variance of $\hat{b}$ given by

$$\text{variance matrix } V(\hat{b}) = s^2(X^TX)^{-1} ,$$

assuming that X^TX is invertible.

2.2.1 Orthogonal Transformations

We are interested in solving the problem

$$\hat{b} = \underset{b}{\operatorname{argmin}} \|\mathbf{y} - Xb\|^2 \tag{2.3}$$

Let Q be an $m \times m$ orthogonal matrix (Q is called orthogonal if $Q^TQ = I_m$, see appendix for details), and

$$\mathbf{y}^* = Q\mathbf{y} = QXb + Q\epsilon = X^*b + \epsilon*$$

It must be noted that multiplication by an orthogonal matrix does not change the 2-norm of a vector. It results only in the rotation of the vector in $I\!R^m$ and, hence, its length is unchanged. Q being an orthogonal matrix

$$\|\mathbf{y} - Xb\|^2 = \|\mathbf{y}^* - X^*b\|^2$$

for any given b. Therefore,

$$\hat{b} = \underset{b}{\operatorname{argmin}} \|\mathbf{y} - Xb\|^2 = \underset{b}{\operatorname{argmin}} \|\mathbf{y}^* - X^*b\|^2 \tag{2.4}$$

That is, the estimate $\hat{b}$ is same for the original system of equations and the rotated system of equations. If we can select a Q in such a way that X^* is an upper triangular matrix (zeros below the diagonal elements). Due to that additional structure

$$\begin{pmatrix} \mathbf{y}_1^* \\ \mathbf{y}_2^* \end{pmatrix} = \begin{pmatrix} X_1^* \\ 0 \end{pmatrix} b + \begin{pmatrix} \epsilon_1^* \\ \epsilon_2^* \end{pmatrix} .$$

Therefore,

$$\|\mathbf{y}^* - X^*b\|^2 = \|\mathbf{y}_1^* - X_1^*b\|^2 + \|\mathbf{y}_2^*\|^2$$

and, therefore,

$$\hat{b} = \operatorname{argmin} \|\mathbf{y}_1^* - X_1^*b\|^2$$

And now, X_1^* being upper triangular $\hat{b}$ can be found by the backward substitution and $RSS = \|\mathbf{y}_2^*\|^2$.

Backward Substitution: This is a technique to solve for the vector $b \in I\!R^n$ in an equation $\mathbf{y}_1^* = X_1^*b$ where $\mathbf{y}_1^* \in I\!R^n$ and $X_1^* \in I\!R^{n\times n}$ is an upper-trianguler matrix. That is,

$$\begin{bmatrix} y_1^* \\ y_2^* \\ \dots \\ y_n^* \end{bmatrix} = \begin{bmatrix} x_{11}^* & x_{12}^* & \dots & x_{1n}^* \\ 0 & x_{22}^* & \dots & x_{2n}^* \\ \dots & & & \\ 0 & 0 & \dots & x_{nn}^* \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix} .$$

First consider the last equation in this system of equations. All the elements in the last row of X_1^* are zero except the last element and therefore one can solve easily for b_n according to

$$\hat{b}_n = \frac{y_n}{x^*_{nn}} \ .$$

Once we have $\hat{b}_n$, then we can solve the second last equation finding $\hat{b}_{n-1}$ according to

$$\hat{b}_{n-1} = \frac{y_{n-1} - x^*_{n-1,n} b_n}{x^*_{n-1,n-1}} \ .$$

Continuing this way we can solve for all the elements of the estimate $\hat{b}$ according to the general formula,

$$\hat{b}_j = \frac{y_j - \sum_{i=j+1}^{n} x^*_{j,i} b_i}{x^*_{j,j}} \ , \ j = n-1, n-2, \dots, 1 \tag{2.5}$$

Algorithm: Given an upper triangular matrix $X \in I\!R^{m\times n}$ and a vector $\mathbf{y} \in I\!R^{m}$, find the least squares estimate of b.

Algorithm 8 (Backward Substitution)

```
function $b$ = backsub($X$,$\mathbf{y}$)
$l = size(X)$;
$n = l(2)$;
$b(n,1) = \mathbf{y}(n,1)/X(n,n)$;
for $j = n-1 : -1 : 1$
        $b(j,1) = (\mathbf{y}(j,1) - X(j, j+1 : n) * b(j+1 : n, 1))/X(j,j)$;
end
```

The next question is: *how to find O efficiently such that X^* is upper triangular?* Most reliable solutions to linear regression are based on a reduction of X to some canonical form via orthogonal transformations. There are two popular techniques to find such orthogonal transformations: Householder transformations and Givens rotations. These are based on modifying the column vectors of matrix X using the reflections or the rotations. By selection of appropriate plane of reflections or appropriate angles of rotation, we can introduce zeros in appropriate places in X.

2.2.2 Basic Rotations & Reflections

We wish to transform vectors and matrices through multiplication by orthogonal matrices in such a way that some of their elements can be selectively made zero. In our case, for a $m \times n$ matrix X (with $m > n$) we want to zero out the elements below the diagonal elements.

Definition 6 *An $n \times n$ matrix O is called an orthogonal matrix if $O^T O = I_n$, where I_n is a $n \times n$ identity matrix.*

There are two types of orthogonal matrices: (i) rotations and (ii) reflections. Since for an orthogonal matrix O, $O^T O = I$, we have $\det(O) = \pm 1$. In case the determinant is $+1$, the matrix O is called a rotation matrix, and for determinant -1, it is called a reflection matrix. Therefore, the task of making X upper triangular can be performed using either the rotation and reflection transformations. For the case of $n = 2$, the two types of orthogonal matrices can be specified as follows.

1. In case of $n = 2$, the rotation matrices always take the form,

$$O_{rot} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

   For an $x \in I\!\!R^2$, the vector Ox is nothing but a rotated version of x, rotated in the counterclock direction by an angle θ, as shown in Fig 2.1. To zero out the second component

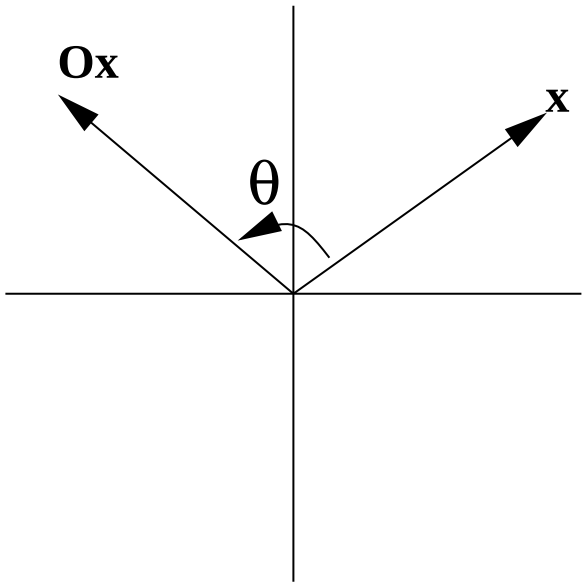


Figure 2.1: A vector x when multiplied by a special orthogonal matrix O results in a positive rotation by angle θ.

   of x, that is x_2, we can rotate x such that it ends up on the x-axis. So for a given $x \in I\!\!R^2$, the goal is to find an angle θ such Ox is aligned with the x-axis.

   The same explanation extends in case of vectors in higher dimensions.

2. Instead, a matrix of the following type

$$O_{ref} = \begin{bmatrix} cos(\theta) & sin(\theta) \\ sin(\theta) & -cos(\theta) \end{bmatrix}$$

   is called a reflection matrix. Multiplication of a vector by this matrix results in reflection of this vector in a mirror positioned on line $\frac{\theta}{2}$, as shown in Figure 2.2. The advantage of rotation and reflections is that they can be used to introduce zeros in a vector by appropriately choosing the rotation angle and the reflection plane, while preserving the norm of that vector.

2.3 Householder Reflection Transformations

First we look at the technique of transforming data through reflection, according to a process called Householder reflections. Let S be a vector subspace (please refer to [4] for a definition of vector spaces). Then, a $m \times m$ matrix P is called the orthogonal projection onto S if

1. $range(P) = S$,

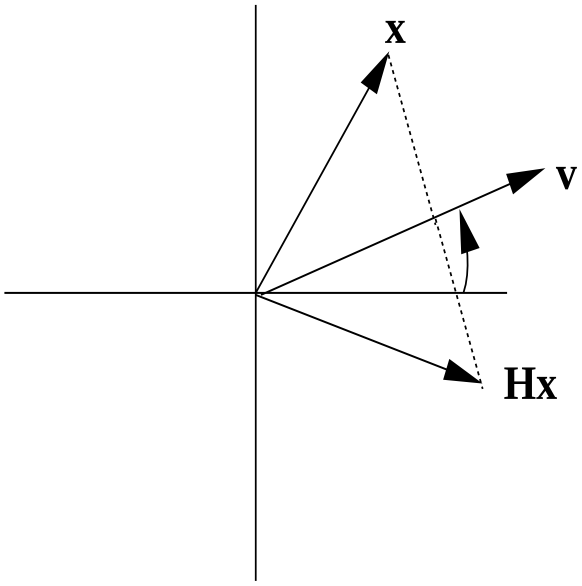


Figure 2.2: A vector x when multiplied by a reflection matrix O results in a vector which is the mirror reflection of x with the mirror located at angle $\frac{\theta}{2}$. θ.

2. $P^2 = P$ (P is idempotent), and

3. $P^T = P$ (P is symmetric)

If $v = [v_1\ v_2, \ldots, v_m]$ forms an orthogonal basis of S then the matrix $P = \frac{vv^T}{v^Tv}$ is the unique orthogonal projection matrix on to S. Therefore, for a vector $v \in I\!R^m$, and $v \neq 0$, $P = \frac{vv^T}{v^Tv}$ is the orthogonal projection onto the subspace spanned by v, $S = span(v)$.

Definition 7 *For a vector $v \in I\!R^m$, a $m \times m$ matrix H of the form*

$$H = I - 2vv^T/v^Tv$$

is called a Householder reflection matrix (or just Householder matrix or Householder transformation).

v is called the Householder vector and H is an orthogonal matrix. For any $x \in I\!R^m$, Hx is the vector obtained by reflecting x is the hyperplane $span(v)^\perp$, as shown in Figure 2.3. A simple calculation shows that $Hv = -v$. Also, it should be noted that the scale (length) of v does not affect H. We can specify v within a scalar constant and still have a unique definition of H. H is a rank-1 modification of the identity and can be used to zero out the selected members of a vector while preserving its norm.

How to find the appropriate reflection matrix? Let's say that for a given vector $x \in I\!R^m$, we want to form an H, a householder matrix, in such a way that Hx has all but the first entry as zeros. In other words Hx lies in the span of e_1, that is, $Hx = \alpha e_1$ for some constant α. Since H is an orthogonal matrix, the only possible values for α are $\pm\|x\|_2$ (why?). First we try with $Hx = \|x\|_2 e_1$. Since,

$$Hx = (I - 2vv^T/v^Tv)x = x - (\frac{2v^Tx}{v^Tv})v \ ,$$

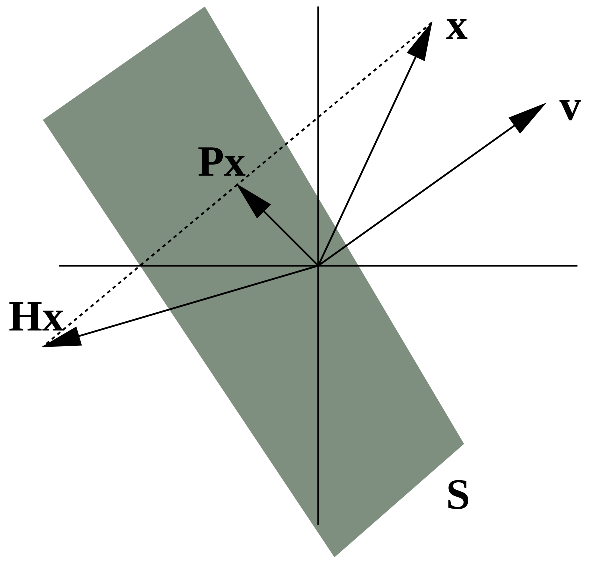


Figure 2.3: $S = span(v)^{\perp}$ is a plane perpendicular to v. Px is the projection of x onto the plane S. Hx is the mirror reflection of x with the mirror located in the plane S.

and $Hx = \|x\|_2 e_1$ imply that $v = \alpha x + \beta e_1$ for some constants α and β. However, as shown later, we need to find v only within a constant and therefore one of the constants can be ignored giving $v = x + \beta e_1$. Now, $v^T x = x^T x + \beta x_1$, and $v^T v = x^T x + 2\beta x_1 + \beta^2$. So,

$$Hx = (1 - 2\frac{x^T x + \beta x_1}{x^T x + 2\beta x_1 + \beta^2})x - 2\beta\frac{v^T x}{v^T v}e_1 \ .$$

We will have the desired value of Hx if the first term on the right side is zero and the second term equals that desired value. That is,

$$1 = 2\frac{x^T x + \beta x_1}{x^T x + 2\beta x_1 + \beta^2}$$

which implies that $\beta = \pm\|x\|_2$. The householder vector is given by $v = x \pm \|x\|_2$ and the reflected vector is $Hx = \mp\|x\|_2 e_1$. The next issue is: what sign should we choose for v, plus or minus? From the considerations of numerical analysis, to avoid subtractive cancellation, it is recommended to use the same sign as x_1.

$$v = x + sign(x_1)\|x\|_2 e_1 \ .$$

Notice that multiplying v by a constant scalar does not change the value of the resulting H. There is one extra degree of freedom in defining v according to the above equation. To remove that, we normalize v such that $v(1) = 1$. The resulting v always has $v(1) = 1$ so we need to store only its last $(m-1)$ entries. Similarly, Hx always has the last $(m-1)$ entries as zero, therefore we don't need to store these values. In total, only m storage places are needed to store both v and Hx. Hence, the algorithm is designed to perform compute v and Hx and store them in place of the original vector x. The vectors take the following form:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ v_2 \\ \cdots \\ v_m \end{bmatrix}, \quad Hx = \begin{bmatrix} \|x\|_2 e_1 \\ 0 \\ \cdots \\ 0 \end{bmatrix}$$

We are ready to write an algorithm to form a householder matrix for a given $x \in I\!R^m$.
Algorithm: Given an m-vector x compute an m-vector v such that $v(1) = 1$ and $(I - 2\frac{vv^T}{v^Tv})x = \beta e_1$ for a non-zero β. The matlab function to implement this algorithm is:

Algorithm 9 (Householder Vector)

```
function v = house(x);
m = length(x);
$\mu$ = norm(x,2);
v = x;
if $\mu \neq 0$
        $\beta = x(1) + sign(x(1))\mu$;
        $v(2:m,1) = v(2:m,1)/\beta$;
end
$v(1) = 1$;
```

The total number of operations is approximately $3m$ floating point operations. We can exploit the structure of H to maximally utilize the memory in the following way. Let $X \in I\!R^{m \times n}$ and let H be the householder matrix

$$H = I - 2\frac{vv^T}{v^Tv}$$

then $HX = X - 2\frac{vv^TX}{v^Tv}$. For $\beta = \frac{-2}{v^Tv}$ and $w = \beta X^T v$, $HX = X + vw^T$ which implies a rank-one update of X. That is, any matrix multiplied with a householder matrix results in its rank one update, which is implemented via the vectors w and v. This avoids explicit calculation of the householder matrix and is accomplished by, for example, the following algorithm.
Algorithm: Given an $m \times n$ matrix $(n < m)$ X and a m-vector v overwrite X with HX where

$$H = I - 2\frac{vv^T}{v^Tv} \ .$$

The matlab function is:

Algorithm 10 (Householder Multiplication - Row)

```
function X = rowhouse(X,v)
$\beta = \frac{-2}{v^Tv}$;
$w = \beta X^T v$;
$X = X + vw^T$ ;
```

Instead if we want XH then the following algorithm applies

Algorithm 11 (Householder Multiplication - Column)

```
function X = colhouse(X,v)
$\beta = -2v^Tv$;
$w = \beta Xv$;
$X = X + wv^T$;
```

The number of floating point operations in these algorithms is $4mn$.

We are interested in transforming a given matrix X through multiplication by orthogonal matrices in such a way that it results in an upper triangular matrix, X^*. Under the Householder technique, we apply a householder transformation to each of the n columns in an iterative way so that for each column the entries below the diagonal are converted to zero. If B is an intermediate result after the first j-transformations then $B(j+1:m,j)=0$, for $1 \leq j \leq n$. As discussed earlier, the lower diagonal entries in B can be used to store the the householder vector v for each corresponding column. The following algorithm accomplishes that.

Algorithm: Given a $m \times n$ matrix X convert it into an upper triangular matrix (**in place**) using Householder transformations, and store householder vectors v below diagonal in the transformed X.

Algorithm 12 (Householder Transformation)

```
function $X$ = householder($X$)
$[m,n] = size(X)$;
$v = zeros(m,1)$;
for $j = 1:n$
        $v(j:m,1) = house(X(j:m,j))$;
        $X(j:m,j:n) = rowhouse(X(j:m,j:n), v(j:m,1))$;
        $X(j+1:m,j) = v(j+1:m,1)$;
end
```

In each step of the *for*-loop we apply an order $m-(j-1)$ Householder matrix

$$\tilde{H}_j = I - 2\frac{\tilde{v}\tilde{v}^T}{\tilde{v}^T\tilde{v}}$$

to the $(m-j+1)\times(m-j+1)$ submatrix on the lower-right of X. Let H_j be the $m \times m$ matrix generated as:

$$H_j = \begin{bmatrix} I_{j-1} & 0 \\ 0 & \tilde{H}_j \end{bmatrix}.$$

To understand this procedure let X be the original $m \times n$ matrix, $X^{(1)}$ be the result after the first transformation ($j=1$), $X^{(2)}$ after the second transformation ($j=2$) and so on. That is, $X^{(1)} = H_1X$, $X^{(2)} = H_2X^{(1)}$. Let

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ & \dots & & \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}$$

Then

$$X^{(1)} = H_1X = \begin{bmatrix} x^1_{11} & x^1_{12} & \dots & x^1_{1n} \\ 0 & x^1_{22} & \dots & x^1_{2n} \\ & \dots & & \\ 0 & x^1_{m2} & \dots & x^1_{mn} \end{bmatrix}$$

and

$$X^{(2)} = H_2X^{(1)} = \begin{bmatrix} x^1_{11} & x^1_{12} & \dots & x^1_{1n} \\ 0 & x^2_{22} & \dots & x^2_{2n} \\ 0 & 0 & \dots & \\ 0 & 0 & \dots & x^2_{mn} \end{bmatrix}$$

and finally

$$X^{(n)} = H_n H_{n-1} \dots H_1 X^{(1)} = \begin{bmatrix} x_{11}^1 & x_{12}^1 & \dots & x_{1n}^1 \\ 0 & x_{22}^2 & \dots & x_{2n}^2 \\ 0 & 0 & \dots & \\ 0 & 0 & \dots & x_{nn}^n \\ 0 & 0 & \dots & 0 \\ \dots & & & \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

In the computation of X^* the following points are worth noting:

1. The composite Householder matrix $O = H_n H_{n-1} \dots H_1$ is never calculated directly.
2. Not even the individual transformation matrices H_j's are calculated explicitly.
3. Multiplication of $X^{(j-1)}$ by H_j does not affect the first $j-1$ columns and the first $j-1$ rows of $X^{(j-1)}$.
4. Application of j-th transformation (H_j) to the j-th column modifies the diagonal entry in that column and all the other entries below become zero.

Householder reflections introduce zeros on a grand scale, that is, all the elements below the diagonal become zero. For a more selective zeroing of elements in a matrix another technique, based on rotation matrices, can be used.

2.4 Givens Rotation Transformations

In Figure 2.1, a vector $x \in I\!R^2$ is rotated to a vector $y = Ox$ by multiplying it with a 2×2 rotation matrix

$$O = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix} .$$

Suppose that the angle θ is such that the resulting y is aligned with the horizontal axis. In other words, the second coordinate of y, namely y_2, has been zeroed out through rotation by an appropriate O. This idea is utilized in case of the Givens transformations for selectively zeroing the entries in a sparse matrix. Any element below diagonal can combine with the diagonal element (of its column) to form a 2-vector which can then be rotated by choosing appropriate O so that it becomes zero.

A Givens rotation matrix is basically a transformation of the form

$$G(i,k,\theta) = \begin{bmatrix} 1 & \dots & 0 & \dots & 0 & \dots 0 \\ 0 & \dots & & & & \\ \dots & & & & & \\ 0 & \dots & c & \dots & s & \dots 0 \\ \dots & & & & & \\ 0 & \dots & -s & \dots & c & \dots 0 \\ 0 & \dots & 0 & \dots & 0 & \dots 1 \end{bmatrix} ,$$

where $c = cos(\theta)$ and $s = sin(\theta)$ and the 2×2 submatrix formed by c and s resides in i, k rows and columns of the bigger matrix G. G is an orthogonal matrix and for any $x \in I\!R^m$ and

$y = G(i,k,\theta)^T x$

$$y_j = \{ \begin{array}{cc} cx_i - sx_k & j = i \\ sx_i + cx_k & j = k \\ x_j & \text{otherwise} \end{array} .$$

So if the desired value for y_k is 0 then have

$$c = \frac{x_i}{\sqrt{x_i^2 + x_k^2}}, \quad s = \frac{-x_k}{\sqrt{x_i^2 + x_k^2}} .$$

One should guard against the possibility of overflow, which occurs when one of these numbers is much larger than other. The following algorithm computes c and s and checks against the possibility of overflow:

Algorithm: Given scalars a and b compute the numbers $x = cos(\theta)$ and $s = sin(\theta)$ such that

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} .$$

Following subroutine accomplishes this task.

Algorithm 13 (Givens Rotation)

```
function $[c,s] = givens(a,b)$
if $b = 0$
        $c = 1; s = 0;$
else
        if $|b| > |a|$
                $\tau = \frac{-a}{b}; s = \frac{1}{\sqrt{1+\tau^2}}; c = s\tau;$
        else
                $\tau = \frac{-b}{a}; c = \frac{1}{\sqrt{1+\tau^2}}; s = c\tau;$
        end
end
```

Notice that this algorithm does not compute the rotation θ and also does not calculate the inverse trigonometric functions.

Example 3 *Let $x = [1\ 2\ 3\ 4]^T$ and we want to have a zero in the last location by multiplying x with an orthogonal matrix. That is, find G such that $y = Gx = [y_1\ y_2\ y_3 \ldots\ 0]^T$. We will choose $x_2 = 2$ as the i^{th} element and $x_4 = 4$ becomes the k^{th} element. In this case $a = 2$ and $b = 4$, and*

$$\tau = \frac{-1}{2}, s = \frac{2}{\sqrt{5}}, c = \frac{-1}{\sqrt{5}}\ l.$$

It can be checked that

$$G(2,4,\theta)x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{-1}{\sqrt{5}} & 0 & \frac{-2}{\sqrt{5}} \\ 0 & 0 & 1 & 0 \\ 0 & \frac{2}{\sqrt{5}} & 0 & \frac{-1}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ -\sqrt{20} \\ 3 \\ 0 \end{bmatrix} .$$

Next consider a sparse matrix $X \in I\!R^{m\times n}$ and its transformation by a Givens rotation matrix G. Notice that, in the left multiplication of X by $G(i,k,\theta)$ only the rows i and k are affected. Therefore, if this multiplication is implemented through a function, one needs to send it only these two rows. Following is an algorithm which implements the Givens rotation for the selected two rows of a matrix.

Algorithm: Given $X \in I\!R^{2\times n}$ and $c = cos(\theta)$ and $s = sin(\theta)$, write $\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T X$ over X. The matlab function to perform this task is:

Algorithm 14 (Givens Rotations - Row)

```
function $X = rowrot(X, c, s)$
$n = cols(X)$;
for $j = 1 : n$
        $\tau_1 = X(1,j); \tau_2 = X(2,j)$;
        $X(1,j) = c\tau_1 - s\tau_2$;
        $X(2,j) = s\tau_1 + c\tau_2$;
end
```

Notation: We will use the matlab notation of $X([i,k];:)$ to represent the submatrix

$$\begin{pmatrix} x_{i1} & x_{i2} & \dots & x_{in} \\ x_{k1} & x_{k2} & \dots & x_{kn} \end{pmatrix} .$$

In this notation, we can use the following function call

$$X([i,k],:) = rowrot(X([i,k],:), c, s) \ ,$$

to perform $X \leftarrow G(i,k,\theta)^T X$. To illustrate the use of Givens rotation in multiple linear regression, consider a $m \times n$ matrix X whose certain entries we want to zero out using orthogonal transformation. To zero out say $X(k,j)$ (assuming $k > j$), the transformation will modify the rows j and k of X. The algorithm which performs this task is as follows:
Algorithm: Given an $m \times n$ matrix X perform a Givens transformation to zero out its element $X(k,j)$ where $k > j$.

Algorithm 15 (Givens Transformation)

```
function $X = giventrans(X, k, j)$
$[c, s] = givens(X(j,j), X(k,j))$;
$X([j,k], j:n) = rowrot(X([j,k], j:n, c, s)$;
```

Error Analysis:
The numerical stability of Givens rotation is same as that of Householder reflections. The errors introduced in the computing c and s are bounded as follows:

$$\begin{aligned} \hat{c} &= c(1+\epsilon_c), \quad |\epsilon_c| \le \mathbf{u} \\ \hat{s} &= s(1+\epsilon_s), \quad |\epsilon_s| \le \mathbf{u} \end{aligned}$$

Therefore, for the Givens transformation of the matrix X

$$fl(\hat{G}(i,k,\theta)^T X) = G(i,k,\theta)^T (X+E)$$

where $\|E\|_2 \sim \mathbf{u}\|X\|_2$.

Considering the composite transformation of X through application of multiple Givens rotations, $O = G_t G_{t-1} \dots G_1$, it is more economical (computationally) to keep O in the factored form. The technique is to associate a single floating point number, call it ρ, with each 2×2 rotation denoted by

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}, \quad c^2 + s^2 = 1 \ .$$

Define ρ by the following algorithm

Algorithm 16 (Find Givens Rotation)

```
function $\rho = findrho(c,s)$
if $c = 0$
        $\rho = 1$;
else if $|s| < |c|$
        $\rho = sign(c).\frac{s}{2}$;
else
        $\rho = 2\frac{sign(s)}{c}$;
end
```

The basic idea is to store $\frac{s}{2}$ if s is smaller or store $\frac{2}{c}$ if c is smaller. From $\rho \in I\!R$, one can reconstruct c and s according to:

Algorithm 17 (Givens Rotation)

```
function $[c,s] = findcs(\rho)$
if $\rho = 1$
        $c = 0;\ s = 1$;
else if $|\rho| < 1$
        $s = 2\rho;\ c = \sqrt{1-s^2}$;
else
        $c = \frac{2}{\rho}; s = \sqrt{1-c^2}$;
end
```

From memory considerations, the number ρ associated with each transformation can be stored in place of the number being zeroed out.

2.5 Multiple Linear Regression Algorithms

Summarizing the problem set up, we are interested in the following minimization problem. For a given $\mathbf{y} \in I\!R^n$ and $X \in I\!R^{n \times p}$ find

$$\hat{b} = \underset{b}{\operatorname{argmin}} \|\mathbf{y} - Xb\|^2 \ .$$

Through orthogonal transformations we simplify the problem to finding

$$\hat{b} = \underset{b}{\operatorname{argmin}} \|\mathbf{y}^* - X^* b\|^2 \ ,$$

where $\mathbf{y}^* = O\mathbf{y}$ and $X^* = OX$ for an $n \times n$ orthogonal matrix O. O can be obtained through Householder transformations or the Givens transformations, as described earlier. Once O is found, then we can solve for $\hat{b}$ through **backward substitution**. That is, for $j = 1, \ldots, p$,

$$\hat{b}_j = \frac{1}{x^*_{jj}} (\mathbf{y}^*_j - \sum_{i=j+1}^{p} x^*_{ji} \hat{b}_i) \ .$$

The residual sum of squares (RSS) is given by

$$RSS = \sum_{i=p+1}^{n} \|y^*_i|^2 \ .$$

In the equation $\mathbf{y} = Xb$, we multiply both sides by orthogonal transformation O in such a way that $X^* = OX$ is upper triangular. For concreteness consider the Householder case. The left side results in y^*, which is computed as follows. Let $\mathbf{y}_j$ be the result of $j-1$ Householder multiplications to the original $\mathbf{y}$. At the j^{th} iterative stage, H_j is multiplied to the vector y_{j-1} to obtain $\mathbf{y}_j$.

$$\mathbf{y}_j = H_j \mathbf{y}_{j-1} = (I_m - 2\frac{vv^T}{v^T v})\mathbf{y}_{j-1} \ .$$

Therefore,

$$\mathbf{y}_j = \mathbf{y}_{j-1} + \beta v, \quad \text{where} \quad \beta = \frac{-2v^T \mathbf{y}_{j-1}}{v^T v} \ .$$

This formula can be utilized to write the complete regression function as follows:
Algorithm: Given a matrix $X \in I\!R^{m \times n}$ and a vector $\mathbf{y} \in I\!R^m$, find a vector $\hat{b} \in I\!R^n$ such that $\hat{b}$ satisfies Eqn. 2.3.

Algorithm 18 (Multiple Linear Regression)

```
function b = multlinreg(X,y)
[m,n] = size(X);
for j=1:n
        v(1:m) = zeros(m,1);
        v(j:m) = house(X(j:m,j));
        X(j:m,j:n) = rowhouse(X(j:m,j:n),v(j:m));
        beta = -2*(v(j:m)'*y(j:m))/(v(j:m)'*v(j:m);
        y(j:m) = y(j:m) + beta*v(j:m);
end
b = backsub(X,y);
```

2.5.1 Performance Analysis

Any statistical estimation procedure should be accompanied by a performance analysis. The main quantity for analyzing regression performance is the error variance. The residual sum of squares is given by :

$$\begin{aligned} \text{RSS} &= \sum_{i=1}^{m} (y_i - \sum_{j=1}^{n} X_{ij}\hat{b}_j)^2 \\ &= \sum_{i=n+1}^{m} (y_i^*)^2 \ . \end{aligned}$$

An estimate of the error variance (also called the mean squared error) is given by:

$$\text{MSE}(s^2) = \text{RSS}/(m-n) \ .$$

To compute the regression performance, one compares the RSS with the sum of squares for the dependent variable, or total sum of squares, denoted by SSY, defined as:

$$\text{SSY} = \sum_{i=1}^{m} y_i^2 - \frac{1}{m}(\sum_{i=1}^{m} y_i)^2 \ .$$

The quantity $SSR = SSY - RSS$ is also called the sum of squares due to regression. The comparison is made using the *coefficient of determination*

$$0 \leq \quad \text{COD} = \frac{\text{SSY} - \text{RSS}}{\text{SSY}} \quad \leq 1 \ .$$

COD takes values between 0 and 1, and a large value of COD implies good regression performance. Another useful indicator of the regression performance is the *coefficient of correlation*, denoted by r, which is given by the square root of the COD:

$$r = \sqrt{\text{COD}} \ .$$

Under the assumption that the error realizations ϵ_i are iid normal with mean zero and variance σ^2, it can be shown that the residuals $y_{n+1}^*, y_{n+2}^*, \ldots, y_m^*$ are iid normal with mean zero and variance σ^2. Therefore, the MSE is given by $\sigma^2\chi_{m-n}^2$, where χ_{m-n}^2 is a chi-squared random variable with $m-n$ degrees of freedom. Furthermore, MSE and $\hat{b}$ are statistically independent.

2.6 General Regression Problems

Although we have looked at the simplest type of linear regression problem, several generalizations are possible. Some of them are:

1. Weighted Least Squares (WLS): This criterion for solving a linear regression problem is useful when one wants to emphasize some observations more than others. For instance, in cases where some observations are taken in more error prone circumstances relative to other observations, we would like to de-emphasize their role in estimating b. This can be accomplished by introducing a diagonal matrix D whose entries are non-negative. The regression coefficients are now obtained by solving for:

$$\hat{b} = \underset{b}{\text{argmin}} (\mathbf{y} - Xb)^T D^{-1} (\mathbf{y} - Xb) \ . \tag{2.6}$$

   How to solve for $\hat{b}$ using our earlier solution? Simply define $A = (D^{\frac{1}{2}})^{-1}$ and solve for

$$\hat{b} = \underset{b}{\text{argmin}} \ \|\mathbf{y}^* - X^*b\|^2 \ ,$$

   where $\mathbf{y}^* = A\mathbf{y}$ and $X^* = AX$. Now the function multilinreg can be applied directly.

2. Generalized Least Squares: In cases where the observation noise ϵ has covariance other than a diagonal matrix, i.e. the variance of ϵ is now given by K, a symmetric positive-definite matrix, then the regression problem solves the following minimization:

$$\hat{b} = \underset{b}{\operatorname{argmin}} (\mathbf{y} - Xb)^T K^{-1} (\mathbf{y} - Xb) \ . \tag{2.7}$$

As earlier, the solution is found by choosing $A = (K^{\frac{1}{2}})^{-1}$, and following the same steps as in WLS. $K^{\frac{1}{2}}$ can be found in several ways including choleski decomposition.

3. Generalized Linear (Interactive) Models: So far we have restricted to only the linear relationships between the dependent and independent variables. But this solution for linear system of equations can sometimes be applied to nonlinear situations where some (nonlinear) function of $\mathbf{y}$ is now linearly related to the X. That is, for some function $f : I\!R \mapsto I\!R$, we solve for the regression coefficients according to:

$$\hat{b} = \underset{b}{\operatorname{argmin}} \|\mathbf{f}(\mathbf{y}) - Xb\|^2 \ . \tag{2.8}$$

f is called the link function. As an example, let y be a Bernoulli random variable with probability p and we want to relate p to the predictor variables. Instead of directly relating p, one relates the variable $\log(\frac{p}{1-p})$ with the predictors. This is also called a *logistic regression* models.

2.7 QR-Factorization

An important technique in analyzing matrices is through a factorization called QR-factorization.

Definition 8 *For a matrix $X \in I\!R^{m \times n}$, $X = QR$ is called the QR-factorization of X if $Q \in I\!R^{m \times m}$ is an orthogonal matrix and $R \in I\!R^{m \times n}$ is an upper-triangular matrix.*

We will assume that the rank of X is n, that is, X is full column ranked. The usefulness of QR-factorization comes from the following result.

Theorem 1 *If $X = QR$ is a QR-factorization of a full column rank matrix $X \in I\!R^{m \times n}$ and $X = [x_1 \ x_2 \ \ldots x_n]$, $Q = [q_1 \ q_2 \ \ldots q_m]$, then*

$$span(x_1, x_2, \ldots, x_n) = span(q_1, q_2, \ldots, q_n) \ .$$

In particular, if $Q_1 = Q(1 : m, 1 : n)$ and $Q_2 = Q(1 : m, n + 1 : m)$ then

$$range(X) = range(Q_1), \quad range(X)^{\perp} = range(Q_2) \ .$$

It must be noted that both Q_1 and Q_2 have orthonormal columns. Therefore, Q_1 and Q_2 provide orthonormal basis for the range space of X and its perpendicular subspace. Such bases are useful in analyzing the fundamental variability associated with the observed data. The uniqueness of such a factorization comes from the following result.

Proposition 1 *Suppose $X \in I\!R^{m \times n}$ has full column rank. Then, the factorization $X = QR$ is unique where $Q \in I\!R^{m \times m}$ has orthonormal columns and $R \in I\!R^{m \times n}$ is upper triangular with positive diagonal entries.*

We claim that using the tools developed so far in this chapter, computing QR-factorization is straight-forward. Recalling that earlier in this chapter we have computed an orthogonal transformation (using Householder and Givens transformations) O such that $OX = X^*$, where X^* is an upper-triangular matrix. For orthogonal matrix O, $O^{-1} = O^T$, we have

$$X = O^T X^* \ .$$

Labeling O^T as Q and X^* as R, this expression is nothing but the QR-factorization of X. So this factorization can be accomplished by our Householder or Givens algorithms. We will only do the Householder QR, the treatment of Givens transformation being similar.
Algorithm: (Householder QR)
Given $X \in \mathbb{R}^{m\times n}$ $(m > n)$ find Householder matrices $H_1,\ H_2,\ \dots, H_p$ such that if $O = H_1 H_2 \dots H_n$ then $OX = R$ is upper triangular. To write such an algorithm we have already done most of the work. That is, we have found transformations which convert X into X^* which is the upper triangular matrix R here. The only additional requirement is to explicitly calculate and store O. This can be done by adding the following steps to the existing programs:

Algorithm 19 (Householder QR Decomposition)

```
function [Q,R] = householderQR(X)
O = eye(m);
for $j = 1 : n$
        $v = zeros(m, 1)$;
        $v(j : m) = house(X(j : m, j))$;
        $X(j : m, j : n) = rowhouse(X(j : m, j : n), v(j : m))$;
        $H = eye(m) - 2\frac{v*v^T}{v^T*v}$;
        O = HO;
end
Q = O'; R = X;
```

In addition to Householder and Givens, there are other methods also by which the QR-factorization can be accomplished.

2.8 Singular Value Decomposition

An important tool to analyze properties of a given matrix is through its singular value decomposition.

Theorem 2 *For any $X \in \mathbb{R}^{m\times n}$ there exist orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ such that*

$$U^T XV = \Sigma, \ \text{where}\ \ \Sigma = diag(\sigma_1, \sigma_2, \dots, \sigma_n) \in \mathbb{R}^{m\times n} \ ,$$

and where $\sigma_1 \geq \sigma_2 \geq \dots \sigma_n \geq 0$.

σ_i's are called the singular values of X and the columns of U and V are called the singular vectors of X. Since $XV = U\Sigma$ and

$$Xv_i = \sigma_i u_i, \quad X^T u_i = \sigma_i v_i$$

For this reason, u_i's are called the left singular vectors and v_i's are called the right singular vectors.

Singular valued decomposition of a matrix reveals a great deal about the matrix. Let the singular values of the matrix be such that

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > \sigma_{r+1} = \ldots = \sigma_n = 0$$

then $rank(X) = r$. Also,

$$null(X) = span(v_{r+1}, v_{r+2}, \ldots, v_n)$$

and

$$range(X) = span(u_1, u_2, \ldots, u_r) \ .$$

The frobenious norm of a matrix is given by the square root of the sum of its singular values, i.e.

$$\|X\|_F^2 = \sum_{i=1}^{n} \sigma_i^2 \ ,$$

and $\|X\|_2 = \sigma_1$. If X happens to be a symmetric matrix then the two orthogonal matrices U and V are same. Based on these results we can analyze the sensitivity of our solution to a linear system equations to small perturbations in the observations.

Sensitivity of a square system: Due to the errors in floating point representation, the system of equation one solves through multiple linear regression changes to

$$(\mathbf{y} + \epsilon f) = (X + \epsilon F)b_1 \ ,$$

instead of the Equation 2.2. The question is: how different is b_1 from the ideal solution b with respect to the relative changes in $\mathbf{y}$ and X? For the purpose of this discussion assume that $m = n$, that is, X is a square matrix and its SVD is

$$X = \sum_{i=1}^{n} \sigma_i u_i v_i^T = U \Sigma V^T \ .$$

For $Xb = \mathbf{y}$ and for X to be invertible

$$b = X^{-1}\mathbf{y} = (U \Sigma V^T)^{-1}\mathbf{y} = \sum_{i=1}^{n} (\frac{u_i^T \mathbf{y}}{\sigma_i}) v_i \ .$$

Errors in representing X and $\mathbf{y}$ are magnified to errors in b when scaled by the singular values σ_i's. The exact dependence can be derived as follows. Let the new system of equations, in view of the floating point errors, be

$$(X + \epsilon F)b(\epsilon) = \mathbf{y} + \epsilon f \ , \tag{2.9}$$

where $b(0) = b$ (the ideal answer), f is an n-vector and F is an $n \times n$ matrix. We are interested in relating the error norm $\frac{\|b(\epsilon) - b\|}{\|b\|}$ to the terms $\epsilon \frac{\|f\|}{\|\mathbf{y}\|}$ and $\epsilon \frac{\|F\|}{\|X\|}$.

It can be seen that if X is invertible, then $b(\epsilon)$ is differentiable in a neighborhood near 0. Therefore, using Taylor's expansion, we have

$$b(\epsilon) = b + \epsilon \dot{b}(0) + O(\epsilon^2) \ .$$

To obtain $\dot{b}(0)$, we differentiate both sides in Eqn. 2.9 with respect to ϵ and set $\epsilon = 0$ to obtain:

$$\dot{b}(0) = X^{-1}(f - Fb) \ .$$

Substituting and analzying:

$$\begin{aligned} \|b(\epsilon) - b\| &= \epsilon\|\dot{b}(0)\| + O(\epsilon^2) \\ &= \epsilon\|X^{-1}(f - Fb)\| + O(\epsilon^2) \\ &\leq \epsilon\|X^{-1}\|\|f - Fb\| + O(\epsilon^2) \\ &\leq \epsilon\|X^{-1}\|(\|f\| + \|Fb\|) + O(\epsilon^2) \\ &\leq \epsilon\|X^{-1}\|(\|f\| + \|F\|\|b\|) + O(\epsilon^2) \end{aligned}$$

Therefore,

$$\frac{\|b(\epsilon) - b\|}{\|b\|} \leq \epsilon\|X^{-1}\|(\frac{\|f\|}{\|b\|} + \|F\|) + O(\epsilon^2) \ .$$

Using the fact that $\mathbf{y} = Xb$, we obtain that $\|\mathbf{y}\| \leq \|X\|\|b\|$, of $\|b\| \geq \|X\|^{-1}\|\mathbf{y}\|$, and therefore,

$$\begin{aligned} \frac{\|b(\epsilon) - b\|}{\|b\|} &\leq \epsilon\|X^{-1}\|(\frac{\|f\|}{\|X\|^{-1}\|\mathbf{y}\|} + \|F\|) + O(\epsilon^2) \\ &= \epsilon\|X^{-1}\|\|X\|(\frac{\|f\|}{\|\mathbf{y}\|} + \frac{\|F\|}{\|X\|}) + O(\epsilon^2) \ . \end{aligned}$$

The number $\kappa(X) = \|X^{-1}\|\|X\|$ is called the condition number of the matrix X. If $\|\cdot\|$ stands for the two-norm of a matrix, then $\kappa(X)$ equals the ratio of the largest singular value to the smallest singular value of X.

This relationship shows that if the condition number of X is large then even small errors in representing X, $\mathbf{y}$ can cause large errors in the estimated value of b. If the condition number is small then the system is more stable with respect to the representation errors.

2.8.1 Dimension Reduction Using Linear Projections

In analysis of systems with large number of components, it is common to reduce dimensions before any statistical analysis. The basic idea to treat observations as elements of a large vector space, and use projections to transform them into smaller vectors. In view of their convenience, linear projections have become popular. If the original observation space is $\mathbb{R}^n$, then the problem reduces to finding an appropriate projection that takes elements of $\mathbb{R}^n$ to elements of $\mathbb{R}^k$ in a linear fashion. Depending on the nature of the required projections, several solutions exists and are discussed next.

2.8.2 Principal Component Analysis

For applications involving a large number of correlated random variables, it is benefitial to try to reduce the the dimensions, without sacrificing too much of the information. One idea is to reduce dimensions by exploiting the correlation between the variables. This idea, called **principal component analysis**, rotates the coordinate axes in order to have the new coordinates with certain optimal variance properties. The original variables when rotated to the new coordinate system are termed the principal components. The principal components are uncorelated to each other which makes them attractive for analysis. In addition, if the number of uncorelated variables is less than the number of original variables, then a reduction in dimension results.

Let x be a vector of n random variables with mean zero and covariance Σ. If the original x has nonzero mean, then use $x - E[x]$ instead to make it zero mean. Let B be an $n \times n$ orthogonal matrix such that the elements of vector Bx are uncorrelated.

Definition 9 *The uncorrelated elements of the vector $z = Bx$ are called the principal components of x.*

According to this definition, the covariance matrix of z is a diagonal matrix, call it Λ. Therefore, $\Lambda = B\Sigma B^T$ implies that $\Sigma = B^T \Lambda B$ for a diagonal matrix Λ. Comparing this relation with the singular value decomposition of a covariance matrix ($\Sigma = U\Lambda U^T$), we can substitute the matrix of singular vectors in place of B^T. Hence the principal components of x are given by $z = U^T x$ where $U\Lambda U^T$ is the SVD of the covariance Σ. There are some interesting properties associated with z.

1. Let the SVD (of Σ) be such that the elements of Λ are non-increasing from top-left to bottom-right. Then, $\Lambda_{1,1}$ is the largest singular value of Σ and $z_1 = u_1^T x$ is the first principal component of x. Here, u_1 is the first column of U. In other words, z_1 has the largest variance, namely $\Lambda_{1,1}$, that can be obtained by projecting x on a unit length vector. Following this argument, z_2 is a random variable that is uncorrelated to z_1 and has the second largest variance namely $\Lambda_{2,2}$. Similarly, z_i is uncorrelated to the first $i-1$ principal components, and has the next highest variance.u_i's denote the directions of principal variations of x. The contribution of z_i towards the variation of x is given by:

$$\Lambda_{i,i} = \sum_{j=1}^{n} \Lambda_{j,j} \ .$$

2. Note that, instead of SVD, one can also use eigen decomposition to find the principal components and principal directions. Since the covariance matrices are symmetric and non-negative definite, both the SVD and eigen-decomposition provide the same results.

3. To motivate the utility of this component analysis, consider the following example. Define a vector $\tilde{x}_k$ as the linear combinations

$$\tilde{x}_k = \sum_{i=1}^{k} a_i u_i, \quad a_i \in I\!R \ , \tag{2.10}$$

where a_i's are i.i.d random variables that are normally distributed with mean zero and variance $\Lambda_{i,i}$. If for some $k < n$, the $\Lambda_{i,i}$'s are such that

$$\Lambda_{1,1} \geq \Lambda_{2,2} \geq \ldots \geq \Lambda_{k,k} > \Lambda_{k+1,k+1} = \ldots = \Lambda_{n,n} = 0 \ ,$$

then $\tilde{x}_k$ has the same mean and variance as the original x. More importantly, if x is multivariate normal then it is completely specified by k variables: $a_1, a_2, \ldots, a_k$. For k significantly less than n, there will be an important speed up in the computational procedures analyzing x. In general, $\Lambda_{k+1,k+1}, \ldots, \Lambda_{n,n}$ may not be zero but rather small for some value of k and can still provide computational simplification. To determine such a k, compute the ratio $\frac{\sum_{j=1}^{k} \Lambda_{j,j}}{\sum_{j=1}^{n} \Lambda_{j,j}}$ for increasing k and stop when this ratio exceeds some prescribed cutoff, say 0.95.

Also, in case x is not multivariate normal, the newer distributions are sought for a_i's keeping the same mean and variances.

Sample Principal Components:
So far we have defined the components given the values of Σ. If Σ is not known, then it is estimated using the samples of x, and the principal components are based on estimated valued of Σ. Let $x_1, x_2, \ldots, x_n$ be the sampled values of the vector x and

$$\hat{\Sigma} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^T \ ,$$

where $\bar{x}$ is the sample mean of x. Then, for $\hat{\Sigma} = \hat{U}\hat{\Lambda}\hat{U}^T$, the SVD of $\hat{\Sigma}$, the vector $\hat{U}^T x$ contains the **sample** principal components of x. The observations of sample components can be obtained as $\hat{U}^T x_i$. As the sample size increases, the estimated values of the components converge to their true values according to the following results.

Proposition 2 *For the sample size m getting larger:*

1. *The estimated values of λ's converge to their true values according to*

$$\sqrt{m}\begin{bmatrix} \hat{\lambda}_1 - \lambda_1 \\ \hat{\lambda}_2 - \lambda_2 \\ \vdots \\ \hat{\lambda}_n - \lambda_n \end{bmatrix} \rightarrow N(\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 2\lambda_1^2 & 0 & \ldots & 0 \\ 0 & 2\lambda_2^2 & \ldots & 0 \\ \vdots & & & \\ 0 & 0 & \ldots & 2\lambda_n^2 \end{bmatrix}) \ .$$

   λ_i's are asymptotically independent of $\hat{U}$.

2. *The estimated values of U converges to its true value according to*

$$\sqrt{m}(vec(\hat{U}) - vec(U)) \rightarrow N(0, V) \ ,$$

   where V is a $n^2 \times n^2$ matrix that depends upon the values of λ's and the matrix U.

The following algorithm generates observations of principal components of x given a large set of its observations.

Algorithm 20 (PCA of Given Data) *Let X be the $m \times n$ matrix where each row denotes an independent observation vector for the random vector x.*

1. *Find the sample covariance matrix $C \in \mathbb{R}^{n \times n}$ of the elements of X,*
2. *Compute the singular value decomposition (SVD) of C to obtain the orthogonal matrix $U \in \mathbb{R}^{n \times n}$,*
3. *Set U_1 to be the first k columns of U, and,*
4. *define $Z = XU_1 \in \mathbb{R}^{m \times k}$.*

2.8.3 Linear Discriminant Analysis

In case the goal is to separate and categorize the observations after their projection, then the choice of projection takes into account this categorization. In other words, we can try to choose a projection that minimizes the spread of projected vectors within the same class, and maximizes the separation of vectors across different classes. In this section we derive an optimal projection using such a criterion.

Let C_1, $C_2, \ldots, C_m$ be m sets that partition $\mathbb{R}^n$ into m classes. We are given N_j observations for class labeled by C_j, i.e. given $X_i^j \in C_j$, $i = 1, \ldots, N_j$, for $j = 1, \ldots, m$. Each X_i^j is a element of $\mathbb{R}^n$. The goal is to find a projection, denoted by the basis U, such that the coefficients from the same class cluster closer and the classes separate away maximally. We perform this analysis by introducing a number of useful quantities. Let μ_j be the mean of observations in class C_j:

$$\mu_j = \frac{1}{N_j} \sum_{i=1}^{N_j} X_i^j \quad \in \mathbb{R}^n \ .$$

A matrix that captures the separation between the classes is the *between-class* scatter matrix:

$$S_B = \sum_{j=1}^{m} (\mu_j - \mu)(\mu_j - \mu)^T \quad \in \mathbb{R}^{n \times n}, \quad \text{where} \mu = \frac{1}{m} \sum_{j=1}^{m} \mu_j \ .$$

Similarly, the average separation between elements within the same class is captured by the *within-class* scatter matrix:

$$S_W = \sum_{j=1}^{m} \left(\sum_{i=1}^{N_j} (X_i^j - \mu_j)(X_i^j - \mu_j)^T \right) \quad \in \mathbb{R}^{n \times n} \ .$$

In case $Z = U^T X \quad \in \mathbb{R}^k$ is the projection of X onto the space spanned by the columns of U, then the scatter matrices for the projections are derived similarly. The corresponding scatter matrices are given by: $S_B^z = U^T S_B U$, and $S_W^z = U^T S_W U$. The goal is to choose U that maximizes the following functions:

$$f(U) = \frac{\det(S_B^z)}{\det(S_W^z)} = \frac{\det(U^T S_B U)}{\det(U^T S_W U)} \ .$$

The optimal projection is given by: $\hat{U} = \text{argmax}_{U \in \mathbb{R}^{n \times k},\ U^T U = I_k} f(U)$. It can be shown that the solution $\hat{U}$ can be solved as the generalized eigen value problem. In other words, the columns of $\hat{U}$ satisfy the equation:

$$S_B \hat{u}_i = \lambda_1 S_W \hat{u}_i \ .$$

In matlab, this can be achieved using the *eig* function for solving the generalized eigen-value problem.

2.9 Problems

1. Let $O \in SO(3)$ be a 3×3 rotation matrix, i.e. Ox is the vector x rotated by an angle θ around an axis v. What is the eigen structure of O, how does it relate to θ and v?

2. Solve the following system of equations using backward substitution:

$$\begin{bmatrix} 12 & 10 & 13 & 10 \\ 0 & 7 & 1 & 4 \\ 0 & 0 & 3 & 2 \\ 0 & 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

3. Let H be a householder matrix given by

$$H = I_n - \frac{2vv^T}{v^Tv} \ , \quad \text{for any} \quad v \in I\!R^n \ .$$

   (a) Show that $H = H^T$ and $H^TH = HH^T = I_n$.

   (b) Show that for any $x \in I\!R^n$, $\|Hx\|_2 = \|x\|_2$.

4. Write a program to implement *house* and execute it with $x = (1, 7, 2, 3, -1)^T$.

5. Write program to implement *rowhouse* and execute it for

$$X = \begin{pmatrix} 1.0 & -5 & 0.5 & 2.0 \\ 7.0 & 1.2 & 3.0 & -1.0 \\ 2.0 & -1.0 & 3.0 & 2.5 \\ 3.0 & -0.9 & -4.0 & 1.0 \\ -1.0 & 3.5 & 2.2 & 4.0 \end{pmatrix}$$

   and v generated from the previous problem.

6. Write a matlab program *BackSub* to solve for the coefficients via backward substitution. That is, for a given vector $y \in I\!R^m$, and matrix $X \in I\!R^{m \times n}$ (upper triangular with non-zero diagonals) solve for $b \in I\!R^n$ such that

$$\mathbf{y} = Xb \ .$$

   Assume $m \geq n$. Test your program on Problem 1 in Assignment #3.

7. Using the functions *householder*, *house*, and *rowhouse* (as defined in the class) write program to convert a given $m \times n$ matrix into an upper triangular matrix. Test your program on

$$X = \begin{bmatrix} 9 & 1 & 2 & 3 \\ 9 & 4 & 2 & 2 \\ 4 & 8 & 2 & 1 \\ 8 & 0 & 6 & 7 \end{bmatrix} .$$

8. Combine the previous two programs to solve for the following linear regression problem: Find $\hat{b} \in I\!R^4$ such that

$$\hat{b} = \underset{b}{\operatorname{argmin}} \|\mathbf{y} - Xb\|^2$$

where $\mathbf{y} = [2\ 4\ 1\ 5]^T$ and

$$X = \begin{bmatrix} 9 & 1 & 2 & 3 \\ 9 & 4 & 2 & 2 \\ 4 & 8 & 2 & 1 \\ 8 & 0 & 6 & 7 \end{bmatrix} .$$

9. Use the above programs to solve for the inverse of X.

10. Let x and y be two non-zero vectors in $I\!R^n$. Give an algorithm for determining a Householder matrix H such that $Hx = y$.

11. For the linear system $\mathbf{y} = Xb$, $X \in I\!R^{m\times n}$, $\mathbf{y} \in I\!R^m$, $b \in I\!R^n$, where X has full column-rank ($rank(X) = n$) the matrix $C = (X^TX)^{-1}$ is known as *variance-covariance* matrix. Assume that the factorization $X = QR$ is available.

   (a) Show that $C = (R^TR)^{-1}$.

   (b) If $R = \begin{bmatrix} \alpha & v^T \\ 0 & S \end{bmatrix}$, then show that

   $$C = \begin{bmatrix} (1 + v^TC_1v)/\alpha^2 & -v^TC_1/\alpha \\ -C_1v/\alpha & C_1 \end{bmatrix} ,$$

   where $C_1 = (S^TS)^{-1}$.

   (c) Give an algorithm that overwrites the upper triangular portion of R with the upper triangular portion of C.

12. Consider a rigid object moving in three-dimensional space. Let v_1 and v_2 be the translational velocities (in $I\!R^3$) of the object separated by a unit time interval. If the driving force is considered random, then the two related according to,

$$v_2 = (I_3 - X)v_1 + n , \tag{2.11}$$

where X is a 3×3 skew-symmetric matrix of the angular velocities, $\begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$ and n is the additive noise (assume $n \sim N(0, \sigma^2I_3)$). Take $\sigma = 0.05$, $a_1 = 0.1$, $a_2 = 0.1$, $a_3 = 0.1$, and $v_1 = [111]^T$;

   (a) Generate a sample of n and evaluate v_2.

   (b) Now consider v_2 to be a given observation from the linear model (Eqn 2.11) and determine the maximum likelihood estimate of v_1, $\hat{v_1}$, using Householder-QR.

   (c) Study the norm of $\|v_1 - \hat{v_1}\|$ versus the noise standard deviation σ.

13. Let $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$. What are the entries in the Givens rotation matrix $G(1, 2, \theta)$ which converts a_{21} to zero.

14. Let x and y be two unit-norm vectors in $I\!R^n$. Use Givens transformations to find a Q such that such that $Q^Tx = y$.

15. For a $m \times n$ matrix X $(m \geq n)$ it can be shown that the singular values $\sigma_1 \geq \sigma_2 \ldots \geq \sigma_n \geq 0$ satisfy the equation

$$\sigma_k = \max_{dim(S)=k} \min_{v \in S, \|v\|=1} \|Xv\| = \max_{dim(T)=k} \min_{u \in T, \|u\|=1} \|u^T X\| ,$$

where S, T are meant to range over all subspaces of $I\!R^n$. Let $w \in I\!R^m$ and define $B = \begin{bmatrix} X & w \end{bmatrix}$. Show that (i) $\sigma_1(B) \geq \sigma_1(X)$, and (ii) $\sigma_{n+1}(B) \leq \sigma_n(X)$. Thus the condition number grows if a column is added to a matrix !

16. Using the programs from previous exercises to write a matlab program to perform (householder) QR factorization of a given matrix. Test your program on

$$X = \begin{bmatrix} 9 & 1 & 2 & 3 \\ 9 & 4 & 2 & 2 \\ 4 & 8 & 2 & 1 \\ 8 & 0 & 6 & 7 \end{bmatrix} .$$

17. Write a matlab program, based on Givens rotations, to zero out the element $(4, 2)$ (4-th row and 2nd column) in the matrix:

$$X = \begin{bmatrix} 9 & 1 & 2 & 3 \\ 0 & 4 & 2 & 2 \\ 0 & 0 & 2 & 1 \\ 0 & 6 & 0 & 7 \end{bmatrix} .$$

18. Let D be a diagonal matrix denoting the covariance matrix of the (zero mean) error vector ϵ, in the equation:

$$\mathbf{y} = Xb + \epsilon \in I\!R^m .$$

Write a matlab program to solve for the weighted least squares estimates of b, given X, y and D. Test your program using:

$$X = \begin{bmatrix} 3 & 3 & 3 & 1 \\ 2 & 1 & 2 & 1 \\ 1 & 3 & 3 & 1 \\ 3 & 2 & 2 & 3 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 52 \\ 28 \\ 44 \\ 48 \end{bmatrix}, \quad \text{and} \quad D = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

19. **Sample Principal Components**: In this problem, you have to write a matlab program to compute principal components from the sampled data. Let $x \in I\!R^4$ be a vector of random variables and its ten observations are given below:

0.9501	0.2311	0.6068	0.4860
0.8913	0.7621	0.4565	0.0185
0.8214	0.4447	0.6154	0.7919
0.9218	0.7382	0.1763	0.4057
0.9355	0.9169	0.4103	0.8936
0.0579	0.3529	0.8132	0.0099
0.1389	0.2028	0.1987	0.6038
0.2722	0.1988	0.0153	0.7468
0.4451	0.9318	0.4660	0.4186
0.8462	0.5252	0.2026	0.6721

Find the principal directions of variation of x, and their corresponding contributions to the total variation.

20. Consider a linear regression problem where $y \in I\!R^m$ and $X \in I\!R^{m \times n}$ are given, and we have to solve for the coefficients $b \in I\!R^n$ such that $\|y - Xb\|^2$ is minimized. In case n is too large to handle, we can use principal component analysis to reduce n to k and then solve for the coefficients.

    (a) For the data provided on the website, first compute a matrix $X_1 \in I\!R^{m \times k}$ as follows: (i) Find the sample covariance matrix $C \in I\!R^{n \times n}$ of the elements of X, (ii) Compute the singular value decomposition (SVD) of C to obtain the orthogonal matrix $U \in I\!R^{n \times n}$, (iii) Set U_1 to be the first k columns of U, and (iv) define $X_1 = XU_1 \in I\!R^{m \times k}$.

    (b) Now solve for the coefficients $\hat{b}_1 \in I\!R^k$ by minimizing $\|y - X_1 b_1\|^2$.

    (c) Compute the SSE $= \|y - X_1 \hat{b}_1\|^2$.

    For the dataset provided $m = 100$, $n = 100$, and use $k = 10$. Save the file on your computer and use "load fname" command to load the data in matlab. Compute and plot the SSE for values of k ranging from 10 to 100 in the steps of 5.

21. Consider a labeled data set X with the following properties: there are $m = 5$ classes, each class has $k = 10$ observations, and each observation is a vector of size $n = 3$. Therefore, X can be thought of as three-dimensional array with dimensions $3 \times 5 \times 10$. In matlab, $X(:, i, j)$ denotes the j^{th} observation vector of i^{th} class.

    Given this data, you have perform a linear discriminant analysis of the data for $d = 1$, and find the projection $U \in \mathbb{R}^{n \times d}$ that is optimal for separating observed classes. You can use the *eig* function in matlab to perform generalized eigen decomposition. For the resulting U:

    (a) State U.

    (b) Project the data X into z, and plot the observations of z.

    (c) Show 3D scatter plot of data X.

    Useful commands in matlab *eig*, $plot(\cdot, \cdot, 'r*')$, *plot3*. Download X from the class website from the file HA4_b_dat.mat.

2.10 References

The material presented in this chapter is taken mainly from the classic text by Golub and Vanloan [2].

Chapter 3

NON-LINEAR STATISTICAL METHODS

3.1 Introduction

(References, [5, 9, 2]). This chapter is devoted to solving for the optimal points of given functions. In the previous chapter, we have restricted ourselves to minimizing quadratic functions ($\|y - Xb\|^2$) but a large number of problems require analyzing non-quadratic functions. The solutions in quadratic case can be determined by solving for the zeros of the gradient function, which in this case defines a system of linear equations. However, in non-quadratic functions, the gradients are nonlinear and the previous methods do not apply. We need to specify more general techniques that optimize given functions, linear or nonlinear.

An instance of nonlinear optimization problems in statistics occurs in cases of **maximum likelihood** estimation. As an example consider the following observation model:

$$y = F(x, b) + \epsilon \ , \tag{3.1}$$

where x and y are random variables of interest, b is a constant vector of parameters, and ϵ is a random vector of measurement or sampling error with the density function given by f_ϵ. F is a functional form relating y, x and b, e.g. $F(x, b) = xb$ or $F(x, b) = (x - b)^2$ etc. In many practical situations, we have the sampled values of y, x, and our goal is to estimate the b according to some criterion. One criterion is to choose that value of b which maximizes the following quantity:

$$\hat{b} = \operatorname*{argmax}_b \prod_{i=1}^{n} f_\epsilon(y_i - F(x_i, b)) \ . \tag{3.2}$$

In this setup we are assuming that the error values in different measurements are independent of each other. The quantity on the right side of Eqn. 3.2 is called the **likelihood function** and $\hat{b}$ is called the **maximum likelihood estimate** (MLE) of b. We now consider some special situations:

1. The linear regression solution we have considered in the previous chapter can be be treated as a maximum likelihood estimate for $F(x, b) = x^T b$ ($y \in I\!R$, $x \in I\!R^n$) and where ϵ is a normal random variable with mean zero and variance $\sigma^2 I$. The likelihood function is given by

$$\frac{1}{(2\pi\sigma^2)^{n/2}} e^{-\frac{1}{2\sigma^2}\|\mathbf{y} - Xb\|^2} \ ,$$

where $\mathbf{y}$ is the vector of measurements of y, and X is the matrix of measurements of x. The log-likelihood function is proportional to

$$-\frac{1}{2}\|\mathbf{y} - Xb\|^2$$

and therefore

$$\hat{b} = \underset{b}{\operatorname{argmin}} \|\mathbf{y} - Xb\|^2$$

which is precisely the vector of regression coefficients found earlier in Chapter 2.

2. It is possible to have the sampling error ϵ that is a normal random variable ($N(0, \sigma^2 I)$ as earlier, but the function F is now a non-linear function of X and b. In this case, the maximum likelihood estimate of b is given by

$$\hat{b} = \underset{b}{\operatorname{argmin}} \|\mathbf{y} - F(X, b)\|^2 \ ,$$

and the solution is obtained by solving this minimization problem.

3. Another possibility is that the function F is linear but the addition noise is no longer normal, i.e.

$$\mathbf{y} = Xb + \epsilon$$

where ϵ is a random vector with joint density function f_ϵ. In that case, the maximum likelihood of b is obtained by solving the following maximization problem:

$$\hat{b} = \underset{b}{\operatorname{argmax}} f_\epsilon(\mathbf{y} - Xb) \ .$$

A value of b which maximizes the likelihood function can be found out by seeking the roots of the first derivative of the likelihood function (also called the score function). So an important ingredient in maximum likelihood estimation is finding the roots of the score function. We start our discussion of non-linear statistics with an overview of some popular techniques in numerical root finding. We will assume that we have a score function f as a function of some argument x and our goal is to find the roots of f.

3.2 Numerical Root Finding: Scalar Case

Start by assuming x to be an unknown deterministic (non-random) parameter, and let x be a scalar. We are given a real-valued function $f(x)$ where $x \in I\!R$ and the goal is to find the roots of f, that is find $x^* \in I\!R$ such that $f(x^*) = 0$.

Due to the non-linearity of the cost function its root are found in an iterative way: an initial estimate is chosen and is modified iteratively until some stopping criterion is met. There are many different methods for root-finding. They all have the following three essential elements:

1. Some of determining the starting value, x_0.

2. Given the i^{th} iterate some formula for calculating the $(i+1)^{th}$ iterate.

3. Some stopping criterion.

For each of these algorithms we are interested in finding their rate of convergence towards the roots of the function. The rate of convergence is defined by a quantity called the **order of convergence**.

Definition 10 *For a converging sequence $\{x_i\}$, the order of convergence is defined by β if,*

$$\lim_{i\to\infty} E_{i+1} = KE_i^{\beta} \ , \quad \textit{where} \ \ E_i = |x_i - x_{i-1}| \ . \tag{3.3}$$

3.2.1 Simple Iterations

Instead of solving for the roots of f we solve an equivalent problem of finding the fixed point of another function g. g is found in such a way that

$$f(x^*) = 0 \Leftrightarrow g(x^*) = x^* \ .$$

In other words, g is found such that a fixed point of g is also a root of f. Sometimes there are many choices of g possible for a single function f. The selection depends on which g is easier to handle and solve for a fixed point. One choice which is always valid is

$$g(x) = x + f(x) \ .$$

Let x_0 be some starting value for the iterative search of the roots of f. At the i^{th} stage of the algorithm, the next iterate is given by the formula:

$$x_{i+1} = g(x_i) = x_i + f(x_i) \ . \tag{3.4}$$

The algorithm for implementing the simple iteration is given by:
Algorithm: Given a function $f(x)$ find its roots using simple iterations by defining $g(x) = x + f(x)$, and:

Algorithm 21 (Simple Iterations)

```
$x(1) = x0;$
$gx = g(x(1));$
$i = 1;$
$while\ (abs(x(i) - gx) > \epsilon)$
    $gx = x(i);$
    $x(i+1) = g(x(i));$
    $i = i + 1;$
$end$
```

The natural question is ask is: under what conditions does this algorithm converges to a root of f or a fixed-point of g. The conditions are defined by the following theorem. Before that, we define the concept of Lipschitz continuity.

Definition 11 *A function $g(x)$, defined on an interval $[a,b]$, is said to be Lipschitz continuous for a given constant L if for any $s,t \in [a,b]$*

$$|g(s) - g(t)| \leq L|s - t| \ .$$

Now we state the theorem.

Theorem 3 *Let $g(x)$ be a continuous function on an interval $[a, b]$ such that $g(x) \in [a, b]$ whenever $x \in [a, b]$ and g satisfies Lipschitz condition with $L < 1$. Then, for any $x_0 \in [a, b]$, the sequence*

$$x_{i+1} = g(x_i) \ ,$$

converges to a fixed point of g and the solution is unique.

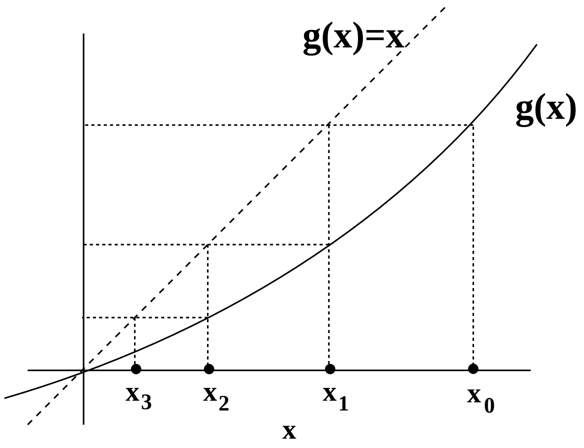


Figure 3.1: Pictorial illustration of convergence via simple iterations. The value of g at the previous iterate becomes the next iterate.

It must be noted that the order of convergence in simple iterations is 1.0, or the convergence is linear.

A necessary and sufficient for this algorithm to converge is that g should be Lipschitz continuous with $L < 1$. But what if g does not satisfy that property. Perhaps g can be scaled such that it satisfies that property.

3.2.2 Scaled Simple Iterations

This is a technique where g can be scaled in such a way that its scaled version satisfies the requirements of Theorem 3. Assume that g is given by

$$g(x) = x + f(x) \ .$$

Notice that for x^* to be the root of f implies that x^* is also the fixed point of g. If g is modified to

$$g(x) = x + \alpha f(x) \ ,$$

where α is a constant scalar, even then the fixed point of g is also the root of f.

Theorem 3 requires that g be Lipschitz continuous with $L < 1$. This can be achieved if $|g'(x)| < 1$ for all $x \in [a, b]$. That is, $|1 + f'(x)| < 1$, for all $x \in [a, b]$. If not, one can choose α in such a way that

$$|1 + \alpha f'(x)| < 1 \ .$$

If $f'(x)$ has the same sign over the domain, then one way is to find $f_0 = \text{argmax}_{x \in [a,b]} |f'(x)|$, and then select any $\alpha < -\frac{1}{f_0}$. The algorithm for implementing the scaled simple iteration is given by:

Algorithm: Given a function $f(x)$ find its roots using scaled simple iterations:

Algorithm 22 (Scaled Simple Iterations)

```
$x(1) = x0;$
$i = 1;$
$gx = x(i) + \alpha * f(x(1));$
$while\ (abs(x(i) - gx) > \epsilon)$
    $gx = x(i);$
    $x(i+1) = x(i) + \alpha * f(x(i));$
    $i = i + 1;$
$end$
```

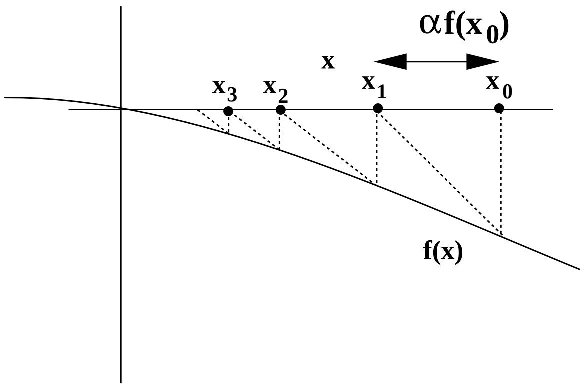


Figure 3.2: Pictorial illustration of convergence via scaled-simple iterations. The value of f, scaled by α, is the increment to find the next iterate.

The advantages of using scaled simple iterations is that it is simple and relatively stable algorithm. The drawback is that it is a slow algorithm.

3.2.3 Newton-Raphson's Method

The slow convergence of the simple iterations and scaled simple iterations is avoided by using the next method. It is one of the most popular techniques used in numerical root finding or fixed point estimation. There are stricter requirements for Newton's method to be applicable to any given function.

For Newton's method to apply, f should be twice continuously differentiable and $f'(x) \neq 0$ near the solution x^*. Again, we will derive an iterative procedure for computing x_{i+1} given x_i. Let x_i be close to x^* ($f(x^*) = 0$). Using Taylor's expansion,

$$f(x) = f(x_i) + (x - x_i)f'(x_i) + \frac{(x - x_i)^2}{2} f''(x_t) \ ,$$

where $x_t \in (x, x_i)$. For $x = x^*$,

$$0 = f(x^*) = f(x_i) + (x^* - x_i)f'(x_i) + \frac{(x^* - x_i)^2}{2} f''(x_t) \ ,$$

Dropping the second order term and solving for x^*, we obtain,

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \ , \quad (f'(x_i) \neq 0) \ . \tag{3.5}$$

An algorithm to implement this equation iteratively is given by:
Algorithm: Given a function $f(x)$ find its roots using Newton-Raphson's method:

Algorithm 23 (Newton-Raphson Method)

```
$x(1) = x0;$
$i = 1;$
$gx = x(i) - f(x(i))/f'(x(i));$
$while\ (abs(x(i) - gx) > \epsilon)$
    $gx = x(i);$
    $x(i+1) = x(i) - f(x(i))/f'(x(i));$
    $i = i + 1;$
$end$
```

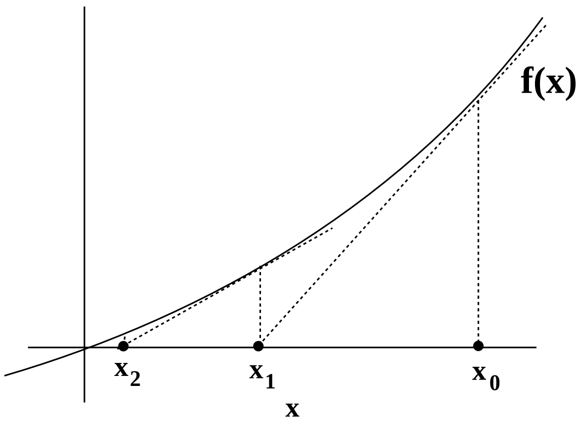


Figure 3.3: Pictorial illustration of convergence via Newton-Raphson's method. The root of the tangent vector at x_i becomes the next iterate, x_{i+1}.

As shown in Figure 3.3, the basic idea is to approximate the function at a given point by a first order polynomial (or a straight line). In this case, the line is given by the line tangent to f at that point. Its root becomes the next iterate x_{i+1}. Newton-Raphson is one of the fastest known algorithms for root finding in general situations. Its order of convergence is $\beta = 2$. The trade-off for speed is of course the restrictions imposed on its applicability: f should be twice continuously differentiable and $f'(x) \neq 0$ near the solution x^*.

There are situations where $f'(x)$ is to difficult to compute or, perhaps, is just not known. Then, the tangent vector at a given point must be approximated numerically. One way is to approximate this derivative function using finite differences, as described in the next section.

3.2.4 Secant Method

From the definition of the derivative, we can attempt to approximate $f'(x)$ using finite differences according to,

$$f'(x) \sim \frac{f(x+h) - f(x-h)}{2h} \ ,$$

or for any two points x_1 and x_2, substitute

$$\frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

in place of $f'(x)$ in the Newton-Raphson iteration. The new formula obtained is called the Secant's method for root finding. the iteration is given by:

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

An algorithm to implement this method iteratively is given by:
Algorithm: Given a function $f(x)$ find its roots using the Secant method:

Algorithm 24 (Secant Method)

```
$x(1) = x0;$
$x(2) = x1;$
$i = 2;$
$gx = x(i) - f(x(i))(x(i) - x(i-1))/(f(x(i)) - f(x(i-1)));$
$while\ (abs(x(i) - gx) > \epsilon)$
        $gx = x(i);$
$x(i+1) = x(i) - f(x(i))(x(i) - x(i-1))/(f(x(i)) - f(x(i-1)));$
        $i = i + 1;$
$end$
```

The basic idea behind the secant method is the following: given two starting points, x_{-1} and x_0, draw a straight line passing through them. Find the root of that line, i.e.the point where it intersects the x-axis. This point becomes the next iterate, x_1. Perform these steps again, now with x_0 and x_1.

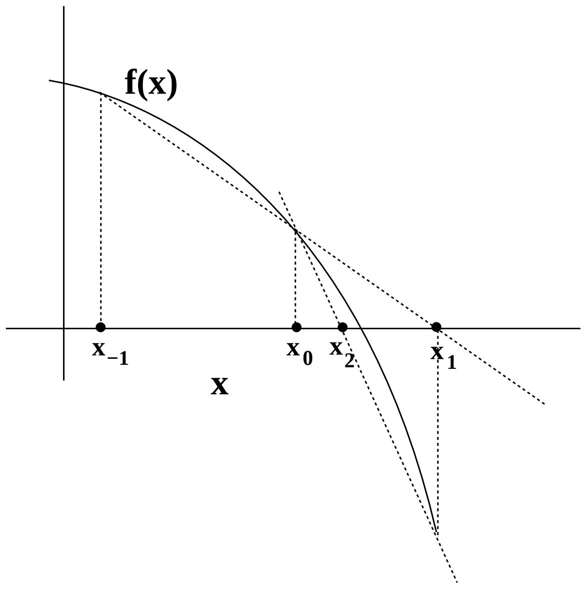


Figure 3.4: Pictorial illustration of convergence via Secant's method. Two points are used to approximate the curve by a secant line whose root becomes the next iterate.

The disadvantage of using secant method is that it requires two starting values, and the convergence is sensitive to the starting values.

One can extend this idea of approximating the curve locally by low-order polynomials. In the secant method the approximation is by a first order polynomial or a straight line. One can approximate the function by a second order polynomial given three starting values. The next iterate is then the root of the second order polynomial approximating the function in the previous step. This method is called **Muller's method**. The order of convergence of the secant method is given by 1.618 and for Muller's method the order of convergence is 1.839.

3.2.5 Bracketing Methods

There is another family of methods which do not rely on the derivatives or their approximations for root finding. This technique basically relies on the intermediate value theorem.

Theorem 4 (Intermediate Value Theorem) *Let f be a continuous function on $[a, b]$ then for all $\gamma \in [f(a), f(b)]$ there exists at least one $c \in [a, b]$ such that $f(c) = \gamma$.*

Assume that there are two points x_0, x_1 with $x_0 < x_1$ such that $f(x_0)f(x_1) < 0$ then 0 is contained in the interval having $f(x_0)$ and $f(x_1)$ as the boundaries. Therefore, by the intermediate value theorem the root of f is contained in $[x_0, x_1]$. Bracketing methods seek this root by successively decreasing the interval while keeping the root in between the end-points. Different bracketing methods differ in the way they reduce the intervals and the associated speeds of convergence. Following are some of the bracketing method commonly used for root finding:

1. **Bisection Method**: This method relies in dividing the interval in half at every step till the resulting interval size is with in the desired accuracy. The algorithm is as follows:
   Algorithm: Given x_0 and x_1 ($x_0 < x_1$) such that $f(x_0)f(x_1) < 0$ find a root of f with in an accuracy given by TOL.

   Algorithm 25 (Bisection Method)

```
function $r = bisection(x_0, x_1)$
$l = x_0;$
$u = x_1;$
$i = 1;$
while $((u - l) > TOL)$
    $x(i + 1) = (u + l)/2;$
    if $f(x(i + 1)f(l) < 0)$
        $u = x(i + 1);$
    else
        $l = x(i + 1);$
    end
    $i = i + 1;$
end
$r = x(i);$
```

   As shown in Figure 3.5, the bisection method has the advantage that it is stable, there is no chance of algorithm diverging. In order to reach an accuracy of ϵ, it takes at most $O(\log_2(1/\epsilon))$ number of steps. To gain some speed one can use the next two methods while still keeping the algorithms stable.

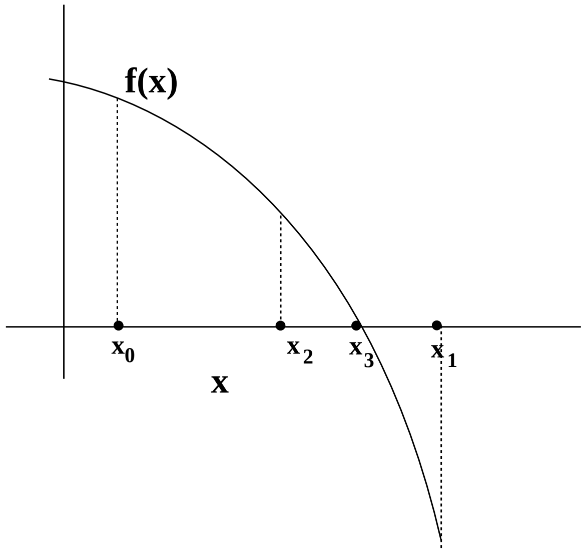


Figure 3.5: Pictorial illustration of convergence via bisection iterations. The mid-point of the interval becomes the next iterate.

2. **Secant Method**: In this case, at any iteration the current end points are used to form a straight line and the root of this line is gives the next iterate. The algorithm is given by: **Algorithm**: Given x_0 and x_1 ($x_0 < x_1$) such that $f(x_0)f(x_1) < 0$ find a root of f with in an accuracy given by ϵ.

Algorithm 26 (Secant Bracketing Method)

```
function $r = secant(x_0, x_1)$
$l = x_0$;
$u = x_1$;
$i = 1$;
while $((u - l) > \epsilon)$
        $x(i+1) = \frac{lf(u)-uf(l)}{f(u)-f(l)}$;
        if $f(x(i+1)f(l) < 0)$
                $u = x(i+1)$;
        else
                $l = x(i+1)$;
        end
        $i = i + 1$;
end
$r = x(i)$;
```

As shown in Figure 3.6, the secant method is faster than the bisection but it has the following drawback: when the function is convex or concave one of the end-points becomes fixed and only the other one changes at every iteration. this results in a slower convergence. To avoid this possibility one can use the Illinois method.

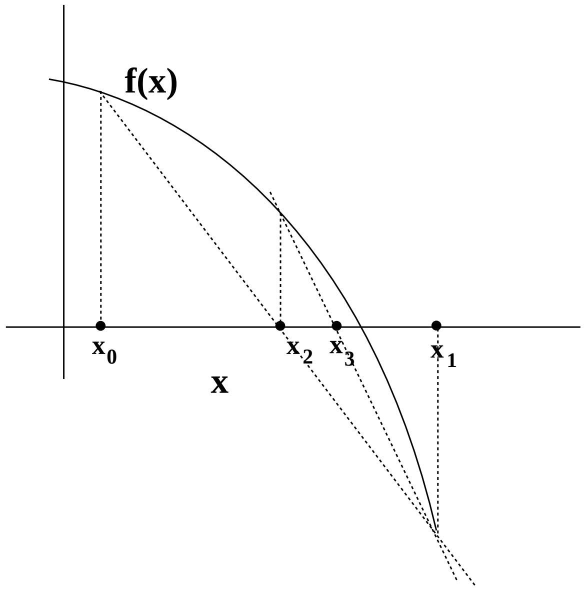


Figure 3.6: Pictorial illustration of convergence via bracketed secant iterations. The root of the line joining two points becomes the next iterate.

3. **Illinois Method**: If one of the end-points is same as previous iteration, then the function value can be replaced by half its value at that point. The algorithm is given by:
   Algorithm: Given x_0 and x_1 ($x_0 < x_1$) such that $f(x_0)f(x_1) < 0$ find a root of f with in an accuracy given by ϵ.

Algorithm 27 (Illinois Bracketing Method)

```
$function\ r = secant(x_0, x_1)$
$l = x_0; fl = f(l);$
$u = x_1; fu = f(u);$
$i = 1;$
$while\ ((u - l) > \epsilon)$
    $x(i+1) = \frac{lfu - ufl}{fu - fl};$
    $if\ f(x(i+1)fl < 0)$
        $u = x(i+1); fu = f(u);$
        $if\ f(x(i))f(x(i+1)) > 0$
            $fl = fl/2;$
        $end$
    $else$
        $l = x(i+1); fl = f(l);$
        $if\ f(x(i))f(x(i+1)) > 0$
            $fu = fu/2;$
        $end$
    $end$
    $i = i + 1;$
$end$
$r = x(i);$
```

To compare the convergence rates, the order of convergence of bisection method is 1, for secant-bracket method is also 1, and for Illinois method is 1.442.

3.3 Starting Values & Stopping Criteria

In root finding if the function is not linear, then a general technique is to start with some initial guess, improve upon the guess iteratively and then stop when certain stopping criterion is satisfied. In the algorithms discussed earlier, various techniques for improving upon the iterates have been established. Aside from the iterative rules, one needs to specify how to find the initial guess and what should be the stopping criteria?

First we discuss the starting values, or the initial guess. Unfortunately, there is not much theory to finding starting values in general. It basically depends on the prior knowledge about the function. One way is to approximate the given function with another function with know roots. For example, a continuous function can be approximated by a polynomial of an appropriate order, and the roots of this polynomial can form the starting value for iterative methods on the original function. Another technique is to plot roughly the given function by computing some sample points and connecting the sampled points to seek an approximate root. This root can then be the starting point for the iterative methods.

The next issue is to define a criterion for stopping the iterative procedure. We need to ascertain when the iteration has sufficiently converged to provide the desired accuracy in the final result. The following quantities can be chosen to measure the convergence:

1. The absolute change in the values of the successive iterates:
$$|x_{i+1} - x_i| < \epsilon \ .$$
2. The absolute relative change in the values of the successive iterates:
$$\frac{|x_{i+1} - x_i|}{|x_i|} < \epsilon \ .$$

The first criterion is a better choice if the root is close to 0, but the second criterion is better if the root is some value away from 0. sometimes it is a good idea to include $|f(x_i)$ in the stopping criterion because near the root this value should be $|f(x_i)| \sim 0$. Another suggestion is to put a hard limit on the iteration count just in case the algorithm has a possibility of diverging.

3.4 Numerical Root Finding: Vector Case

Now we consider the case when the function to be optimized has multiple arguments, i.e. the likelihood function is a function of n variables. In this case, the derivative function, $f(x)$, is a function from $I\!R^n$ to $I\!R^n$, and $x^* \in I\!R^n$ is its root when $f(x^*)$ is a vector of zeros. In this section we will study techniques for finding the vector roots of vector valued functions. Let f be a function from $\mathbf{x} \in I\!R^n$ to $f(\mathbf{x}) \in I\!R^n$ whose root we are interested in finding. Its first derivative with respect to $\mathbf{x}$ will be a matrix, each column consisting of the gradient of each of its components with respect to $\mathbf{x}$. Similarly, its second derivative will be a three-dimensional array of size $n \times n \times n$, each component being a scalar function of $\mathbf{x}$.

The vector root finding can be accomplished by extending the methods described in the previous sections. Additionally, there are some interesting ideas that go beyond the simple extensions of the scalar cases. We consider the methods next:

1. **Simple Iterations**:
   As in the scalar case, the iteration for the simple iteration is given by
   $$\mathbf{x}_{i+1} = \mathbf{x}_i + f(\mathbf{x}_i) \ . \tag{3.6}$$
   Where $g(\mathbf{x}) = \mathbf{x} + f(\mathbf{x})$ satisfies certain properties: its range is a subset of its domain, and it is Lipschitz continuous with parameter less than one.

2. **Scaled Simple Iterations**: If g does not satisfy the requirement for Lipschitz continuity then f can be scaled in order to satisfy that condition. That is, find a matrix J such that
   $$\rho(I - J^{-1}f'(\mathbf{x})) < 1 \ , \quad \text{for all} \ \ \mathbf{x} \text{ near the root} \ ,$$
   where ρ is the spectral radius of its matrix argument. Since it is not straightforward to find a J which satisfies this property, some simple of choices for J are often made. For example $J = \alpha I_n$, for some constant α small enough, is a common choice. This J implies having the same step size in all the dimensions in the iterative procedure for root finding. Very often it is clear from the function involved that the step-sizes for different components should be different. If the choices for individual step sizes is obvious, they can be incorporated using $J = \text{diag}(\alpha_1, \ \alpha_2, \ldots, \alpha_n)$.

3. **Newton-Raphson's Method**: As described in the scalar case the choice of matrix J is analytically precise if certain assumptions are made about f. If the second derivative of f is continuous and $f'(\mathbf{x})$ is not zero in the neighborhood of the root, then using the Taylor's series:
   $$f(\mathbf{x}) = f(\mathbf{x}_i) + f'(\mathbf{x}_i)(\mathbf{x} - \mathbf{x}_i) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_i)^T f''(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}_i) \ ,$$
   for some $\mathbf{x}^* \in \{t\mathbf{x} + (1-t)\mathbf{x}_1 : 0 \le t \le 1\}$. Substituting for $\mathbf{x} = \mathbf{s}$, the root of f, and solving for s, we obtain the next iterate as:
   $$\mathbf{x}_{i+1} = \mathbf{x}_i - \left(f'(\mathbf{x}_i)\right)^{-1} f(\mathbf{x}_i) \ . \tag{3.7}$$
   Computing the next iterate requires inverting an $n \times n$ matrix $f'(\mathbf{x}_i)$. Following the techniques learnt in Chapter 2 we can solve for $\mathbf{x}_{i+1}$ as the solution of the equation
   $$f(\mathbf{x}_i) = -f'(\mathbf{x}_i)(\mathbf{x}_{i+1} - \mathbf{x}_i) \ .$$
   Householder factorization is one way to solve this linear system of equations.

   The advantage of Newton-Raphson is its faster speed of convergence. The disadvantages are that it requires the continuity of $f''(x)$ and the starting point has to be in a small neighborhood of the root.

4. **Secant Method**: In situations where $f'(\mathbf{x})$ is either not known or difficult to calculate, then the secant method is applicable. Extrapolating the secant technique discussed for the scalar case, now we need $(n+1)$ starting points. Each component of f, say f_i, is evaluated at these $n+1$-points and a hyperplane is fitted to pass through the samples of f_i at these points. That is, solve for the coefficients $a_1, \ a_2, \ldots, \ a_n$, in the equations
   $$f(\mathbf{x}_i) = a_1\mathbf{x}_i(1) + a_2\mathbf{x}_i(2) + \ldots + a_n\mathbf{x}_i(n) \ , \quad i = -n, -n+1, \ldots, -1 \ .$$
   The intersection of hyperplanes corresponding to different components of f provides the next iterate, $\mathbf{x}_{i+1}$.

5. **Non-Linear Gauss-Seidal Iteration**: This is a interesting idea for solving n-dimensional root-finding using multiple applications of scalar-root finding. The basic framework is: the i^{th} component of f is considered to be a function only of the i^{th} component of $\mathbf{x}$, other components are assumed fixed. Using scalar techniques, once can solve for the root of f_i as a function of x_i. Perform this step for each $i = 1, 2, \ldots, n$ and combining these individual roots the next iterate for f is ready. If Newton's method is used of scalar root-finding, then the resulting method is called **Gauss-Seidal-Newton**'s method.

   Theorem 5 *Non-linear Gauss-Seidal iteration converges globally to a unique solution in $\mathbb{R}^n$ provided that $f'(\mathbf{x})$ is symmetric positive definite and that the smallest characteristic root of $f'(\mathbf{x})$ is bounded away from zero.*

3.5 Numerical Optimization

so far we have discussed numerical root finding which is only a necessary condition (not sufficient condition) for finding the optimal point (maxima or minima) of a given function. A general optimization problem can be stated as follows:

> Let F be a real-valued function on $\mathbb{R}^n$ and let S be a given connected subset of $\mathbb{R}^n$, then find $\mathbf{x} \in S$ at which F attains an optimum (maximum or minimum) value.

F is called the objective function and S is called the constraint set. Roots of the gradient function are good candidates for the optimal points but additional steps are needed to check for their optimality. In general, the task of optimization is a little different from the root-finding procedure because of the following reasons:

1. The root of the derivative of a function may be a maximum, a minimum or a saddle point. If we are looking for a maximum of f the second derivative $F''(\mathbf{x}^*)$ called the Hessian matrix, should be positive definite. Similarly, if the Hessian matrix is negative definite at $\mathbf{x}^*$ then it implies the maximal point. In other cases, $\mathbf{x}^*$ may be a saddle point.

2. Even if $F'(\mathbf{x}^*) = 0$ and $F''(\mathbf{x}^*)$ is positive definite, $\mathbf{x}^*$ may only be a maximum value in a small neighborhood of $\mathbf{x}^*$. In other words, $\mathbf{x}^*$ is only a local maximum and while considering the whole set S, there may be other global maxima.

3. Sometimes an optimal point is such that $F'(\mathbf{x}) \neq 0$ for that point. For example, consider a linear function F over a rectangular set S. The optimal points of this function lie on the boundary of the constraint set and $F'(\mathbf{x}) \neq 0$ for these points.

4. In certain situation, we may not have a closed form expression for F but have only its sampled values. Again, the search can not be based on $F'(\mathbf{x})$ as is done in root finding.

In case the earlier methods do not apply, there are some additional ideas that can be tried for optimization. These ideas lead to large computational costs, but can still be useful if the earlier methods do not work.

1. **Grid Search**: This method, also called **raw search**, involves partitioning the domain S into numerous subsets and sampling the function f at a point in each subset. Based on the sampled values, the maximal or minimal value can be selected. Obviously, finer

partitioning can result in more accurate results depending upon the continuity of the function. Let $\mathbb{Z}^n$ be the integer grid and define the set of sample points,

$$L = \{\mathbf{x} | \mathbf{x} = \mathbf{x}_0 + M\mathbf{a},\ \mathbf{a} \in \mathbb{Z}^n\} \ .$$

L is called the lattice or grid with origin $\mathbf{x}_0$. M is a full-ranked matrix called the Minkowski's basis of L.

2. **Sequential Search**: Another strategy to search for optimal points relies on solving for each component at a time. Fix all but one components and use the grid method to find the optimal point in that dimension. Then, using the new value fix that component and search for other components one by one.

3. **Linear Search Strategies**: This idea for searching one component at a time can be generalized as follows. Let $\mathbf{x}$ be the a starting point for search in S and let $\mathbf{d} \in \mathbb{R}^n$ be a direction vector. Define a scalar function,

$$g(t) = F(\mathbf{x} + t\mathbf{d}), \quad t \in \mathbb{R} \ .$$

$\mathbf{d}$ is called an admissible if g is a decreasing function at $t = 0$. Assuming that one can solve for the minimizer of g, the minimum t^* provides the next iterate in the search. That is, $\mathbf{x} + t^*\mathbf{d}$ for the new starting point and the process is repeated. It must be noted that the admissible direction $\mathbf{d}$ varies from point to point. So the question is, given a point $\mathbf{x}$ how to find the admissible direction? There are at least three ways of selecting an appropriate $\mathbf{d}$.

   (a) Newton's Method: For $\mathbf{x} \in \mathbb{R}^n$, the admissible direction is calculated as

$$\mathbf{d}_N = -\left(F''(\mathbf{x})\right)^{-1} F'(\mathbf{x}) \ ,$$

   where $F''(\mathbf{x})$ is a positive definite matrix.

   (b) Steepest-Descent: In this case the admissible direction is given by the negative of the gradient vector

$$\mathbf{d}_S = -F'(\mathbf{x}) \ .$$

   (c) Levenberg-Marquardt Adjustment: A combination of these two methods can be formulated by using the direction

$$\mathbf{d}_{LM} = -\left(F''(\mathbf{x}) + \lambda I\right)^{-1} F'(\mathbf{x}) \ ,$$

   where $\lambda > 0$ is a constant greater than the smallest eigen-value of $F''(\mathbf{x})$. In case $F''(\mathbf{x})$ is positive definite then λ can be chosen to be 0 to imply Newton's method ($d_{LM} = d_N$). In case λ is large and dominates $F''(\mathbf{x})$ term, then d_{LM} is quite close to the steepest descent direction, d_S.

3.6 Conjugate Gradient Technique

Another commonly used technique for optimization which depends upon the local definition on an admissible direction at each point is the conjugate gradient technique. We illustrate the basic idea behind conjugate gradients using a simple example. Let a function $F : I\!R^n \to I\!R$ be given by

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T b \tag{3.8}$$

for some fixed $b \in I\!R^n$. Also, assume that A is a $n \times n$ symmetric, positive definite matrix. We already know that the minimizer of F is given by $\mathbf{x} = A^{-1}b$ and the minimum value of F is $\frac{-b^T A^{-1} b}{2}$. That is, if A is symmetric, positive definite, then minimizing F is same as solving for $\mathbf{x} = A^{-1}b$. As earlier, we seek an iterative numerical technique which avoids computing the inverse of A.

We will use the linear search strategy, that is, at any point we select a direction and solve for the optimal point along that direction (by solving a one-dimensional optimization problem). The question is, how to find that search direction at any arbitrary point. As we have seen before, the method of *steepest descent* suggests using $\nabla F(\mathbf{x})$ as the search direction at $\mathbf{x} \in I\!R^n$. Let $\mathbf{x}_c$ be the current point. In this case, $-\nabla F(\mathbf{x}_c) = b - A\mathbf{x}_c$. Let this vector be called the residual vector r_c. If $r_c \neq 0$, then there exists a positive $\alpha \in I\!R$ such that $F(\mathbf{x}_c + \alpha r_c) < F(\mathbf{x}_c)$. The α which minimizes the function $f(\alpha) = F(\mathbf{x}_c + \alpha r_c)$ can be found by the usual calculus. It is given by

$$\hat{\alpha} = \frac{r_c^T r_C}{r_c^T A r_C} \ .$$

Then, $F(\mathbf{x}_c + \alpha r_c) = F(\mathbf{x}_c)\alpha r_c^T r_c + \frac{1}{2}\alpha^2 r_c^T A r_C$. This can be implemented by the following algorithm:

Algorithm 28 *Let $\mathbf{x}_0$ be the initial guess and set $r_0 = b - A\mathbf{x}_0$ and $k = 0$.*

```
while ($r_k \neq 0$)
        $k = k + 1$;
        $\alpha_k = r_{k-1}^T r_{k-1} / r_{k-1}^T A r_{k-1}$;
        $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k r_{k-1}$;
        $r_k = b - A\mathbf{x}_k$;
end
```

It can be shown that

$$(F(\mathbf{x}_k) + \frac{1}{b^T A^{-1} b}) \leq (1 - \frac{1}{\kappa_2(A)})(F(\mathbf{x}_{k-1}) + \frac{1}{b^T A^{-1} b}) \ .$$

where $\kappa_2(A)$ is the condition number of the matrix A (given by the ratio of the largest eigen value to the smallest eigen value of A). If the condition number of A is large, then the progression of the algorithm is slow. The geometric interpretation of this condition is that the level curves of F are elongated hyper-ellipsoids on the gradient direction (which is locally the steepest direction) is not a good choice in the global sense.

Conjugate Search Directions

Now we will look for search directions $p_1, p_2 \ldots$ which are different from the residual $r_1, r_2, \ldots$. If p_k is a search direction then it can be shown that the α which minimizes F along that direction is given by

$$\hat{\alpha}_k = \operatorname*{argmin}_{\alpha} F(\mathbf{x}_{k-1} + \alpha p_k) = \frac{p_k^T r_{k-1}}{p_k^T A p_k} \ ,$$

and $F(\mathbf{x}_{k-1}+\hat{\alpha}_k p_k) = F(\mathbf{x}_{k-1}) - \frac{1}{2}\frac{(p_k^T r_{k-1})^2}{p_k^T A p_k}$. Therefore, p_k should not be orthogonal to r_{k-1}, in addition to the fact that it is not going to be completely along r_{k-1}.

If the search directions $\{p_1, p_2, \ldots\}$ are linearly independent of each other and if

$$\mathbf{x}_k = \underset{\mathbf{x}\in\mathbf{x}_0+span\{p_1,p_2,\ldots,p_k\}}{\text{argmin}} F(\mathbf{x}), \quad k = 1, 2\ldots, n \ ,$$

then the convergence of this iterative procedure it guaranteed in n steps. It is because $span\{p_1, p_2, \ldots, p_k\} = I\!R^n$. Following this strategy, we should select p_k which makes the evaluation $\mathbf{x}_k$ feasible. Let $P_{k-1} = [p_1\ p_2\ \ldots p_{k-1}] \in I\!R^{n\times(k-1)}$. Then $\mathbf{x}_k$ is given by

$$\mathbf{x}_k = \mathbf{x}_0 + P_{k-1}y + \alpha p_k$$

for some $y \in I\!R^{k-1}$ and $\alpha \in I\!R$. And

$$F(\mathbf{x}_k) = F(\mathbf{x}_0 + P_{k-1}y) + \alpha y^T P_{k-1}^T A p_k + \frac{\alpha^2}{2} p_k^T A p_k - \alpha p_k^T r_0 \ .$$

In this function the term $\alpha y^T P_{k-1}^T A p_k$ relates the two elements α and y. We want to select a p_k which makes it easier to find α and y which minimize this cost function. So if the cross-term is zero the optimization over these two quantities separates into two different problems. Set the cross-term to zero (by choosing p_k such that $P_{k-1}^T A p_k = 0$) and let y_{k-1} be the vector which minimizes the first term. Then for $\mathbf{x}_{k-1} = \mathbf{x}_0 + P_{k-1}y_{k-1}$,

$$\mathbf{x}_{k-1} = \underset{\mathbf{x}\in\mathbf{x}_0+span\{p_1,p_2,\ldots,p_{k-1}\}}{\text{argmin}} F(x) \ .$$

Given p_k, $\alpha_k = \frac{p_k^T r_0}{p_k^T A p_k}$ minimizes the other terms. It is convenient to notices that

$$p_k^T r_{k-1} = p_k^T(b - A(\mathbf{x}_0 + P_{k-1}y_{k-1})) = p_k^T r_0 \ .$$

The algorithm to implement this technique is given by:

Algorithm 29 *Let* $\mathbf{x}_0$ *be the initial guess and set* $r_0 = b - A\mathbf{x}_0$ *and* $k = 0$.

```
while ($r_k \neq 0$)
        $k = k + 1$;
        choose $p_k \in span\{Ap_1, Ap_2, \ldots, Ap_k\}^\perp$ so that $p_k^T r_{k-1} \neq 0$,
        $\alpha_k = p_k^T r_{k-1}/p_k^T A p_k$;
        $\mathbf{x}{=}\mathbf{x}_{k-1} + \alpha_k p_k$;
        $r_k = b - A\mathbf{x}_k$;
end
```

Since $p_i^T A p_j = 0$ for all $i \neq j$ the search directions are called *A-conjugate.* From construction, if $P_k = [p_1\ p_2\ \ldots p_k]$, then

$$P_k^T A P_k = diag(p_1^A p_1, p_2^T A p_2, \ldots, pk^T A p_k) \ .$$

Since A is a positive definite matrix $P_k^T A P_k$ is a non-singular matrix, that is, it has full column rank k. Therefore, the convergence is guaranteed in n-steps because P_n is a full ranked $n \times n$ matrix and $\mathbf{x}_n$ lies in its span.

Still, we haven't completely specified how to find p_k, though we have found some conditions which p_k must satisfy. p_k is A-conjugate to p_1, p_2,..,p_{k-1}, p_k is not exactly along r_{k-1} and p_k is not orthogonal to r_{k-1}. There are several strategies suggested in the literature to find p_k. One of them is to compute p_k recursively according to

$$p_k = r_{k-1} + \beta_k p_{k-1} \ , \quad \beta_k = -\frac{p_{k-1}^T A r_{k-1}}{p_{k-1}^T A p_{k-1}} \ .$$

3.7 Expectation Maximization (EM) Algorithm

So far we have looked at maximizing likelihood for parameter θ given an observation x and a likelihood function $f(x|\theta)$. In such cases it is assumed that the maximizer exists, either locally or globally, and the techniques described earlier can solve of it. However, in some problems the full observation may not be provided and that makes the maximization a difficult, or even ill-posed, problem. Let the data vector x be made up of two components $x = [x_o \ x_m]$ where x_o denotes the observed part of x and x_m stands for the missing part. Assume that given the full data x, it is possible to solve for the maximum likelihood estimate of θ but goal now is to solve for the maximizer:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} f(x_o|\theta) \ . \tag{3.9}$$

In fact, EM is beneficial mostly where $f(x|\theta)$ is easier to maximize compare to maximizing $f(x_o|\theta)$. A note of caution here in that our notation is to use f for all density functions, including the conditional density function of x_o given θ, and the arguments of f should make clear the actual function it refers to.

Example 4 (Mixture of Gaussians)
Let Y be a real-valued random variable such that $Y = Y^{(j)}$ with probability α_j, where $Y^{(j)} \sim N(\mu_j, \sigma_j^2)$ and $\sum_j \alpha_j = 1$. The goal is to use independent observations of Y, say $Y_1, Y_2, \dots, Y_n$, to estimate the parameters $\alpha_j, \mu_j, \sigma_j^2$, for all j. To simplify this discussion let the number of densities in the mixture be two, and Y becomes a mixture of two Gaussian random variables. We seek a maximum likelihood estimate of the unknowns $\theta = (\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \alpha_1, \alpha_2)$. The likelihood function is given by:

$$\begin{aligned} f(Y|\theta) &= \prod_{i=1}^{n} f(Y_i|\theta) \\ &= \prod_{i=1}^{n} \left(\alpha_1 f_1(Y_i|\mu_1, \sigma_1^2) + \alpha_2 f_2(Y_i|\mu_2, \sigma_2^2) \right) \end{aligned}$$

where f_1 and f_2 are the two normal density functions with appropriate parameters. The log-likelihood function is given by:

$$\log(f(Y|\theta)) = \sum_{i=1}^{n} \log \left(\alpha_1 f_1(Y_i|\mu_1, \sigma_1^2) + \alpha_2 f_2(Y_i|\mu_2, \sigma_2^2) \right) \ .$$

Solving for $\hat{\theta} = \operatorname{argmax}_\theta \log(f(Y|\theta))$ is difficult because of the summation inside the log function.

Consider a different situation where in addition to Y_is one also observes a label l_i that equals one if $Y_i \sim f_1$ and two if $Y_i \sim f_2$. In other words, l_i tells us what density Y_i came from. Form a larger observation (Y, l), where $l = (l_1, l_2, \ldots, l_n)$, and derive the log-likelihood function:

$$\begin{aligned}\log(f(Y, l|\theta)) &= \sum_{i=1}^{n} \log(f(Y_i, l_i|\theta)) \\ &= \sum_{i=1}^{n} \log(f(Y_i|l_i, \theta)P(l_i|\theta)) \\ &= \sum_{i:l_i=1} \log(\alpha_{l_i} f_1(Y_i|l_i, \mu_1, \sigma_1^2)) + \sum_{i:l_i=2} \log(\alpha_{l_i} f_2(Y_i|l_i, \mu_2, \sigma_2^2)) \qquad (3.10)\end{aligned}$$

Here, we have used the fact that $P(l_i|\theta) = \alpha_i$. Now, the data breaks into two groups: one from f_1 and the other from f_2, and one can use this separate data sets to estimate parameters for those respective densities. So, this example illustrates a situation where the log-likelihood function for the observed data $\log(f(Y|\theta))$ is rather difficult to maximize, while the same function for a complete data, assuming additional data in form of the labels l, is much easier to maximize. The EM algorithm is applied in such situations where the additional data, called the missing data, is not available but could have greatly simplified the optimization problem. In EM algorithm, one does not have the missing data, but tries to integrate out this unknown using a density function conditioned on the observed data.

3.7.1 Derivation of EM Algorithm

Our goal is to construct a sequence $\{\theta_i\}$ such that: (i) $f(x_o|\theta_{i+1}) \geq f(x_o|\theta_i)$, and (ii) with some additional conditions this sequence converges to the estimator $\hat{\theta}$. Rearrange the equation: $f(x|\theta) = f(x_o|\theta)f(x_m|x_o, \theta$ to write:

$$f(x_o|\theta) = \frac{f(x|\theta)}{f(x_m|x_o, \theta)} .$$

Taking log on both sides, we get

$$\log(f(x_o|\theta)) = \log(f(x|\theta)) - \log(f(x_m|x_o, \theta)) .$$

Next, take expectation on both sides with respect to the density function $f(x_m|\theta_0, x_o)$, for some θ_0. Given x_0, the left hand side is a constant and remains same. On the right side, we get

$$\begin{aligned}&E[\log(f(x_o, x_m|\theta))|\theta_0, x_o] - E[\log(f(x_m|\theta, x_o))|\theta_0, x_o] \\ =\ & Q(\theta|\theta_0, x_o) - H(\theta|\theta_0, x_o)\end{aligned}$$

where Q and H are defined by above equations. Q is the expected value of the complete data (log) likelihood conditioned on the given values of θ_0 and x_o. Considering the second term first, we focus on the difference:

$$\begin{aligned}H(\theta|\theta_0, x_o) - H(\theta_0|\theta_0, x_o) &= E[\log(f(x_m|\theta, x_o))|\theta_0, x_o] - E[\log(f(x_m|\theta_0, x_o))|\theta_0, x_o] \\ &= E[\log(\frac{f(x_m|\theta, x_o)}{f(x_m|\theta_0, x_o)})|\theta_0, x_o] \\ &\leq \log(E[\frac{f(x_m|\theta, x_o)}{f(x_m|\theta_0, x_o)}|\theta_0, x_o]) \\ &= \log(\int \frac{f(x_m|\theta, x_o)}{f(x_m|\theta_0, x_o)} f(x_m|\theta_0, x_o) dx_m) = 0\end{aligned}$$

The inequality comes from the Jensen's inequality which says that $E[\log(Y)] \le \log(E[Y])$ since log is a convex function. This implies that $H(\theta|\theta_0, x_o) \le H(\theta_0|\theta_0, x_o)$ for any θ_0.

Now consider a sequence $\theta_1, \theta_2, \ldots, \theta_m$ where the iteration is given by some function l, i.e. $\theta_{m+1} = l(\theta_m)$. Substituting $\theta_0 = \theta_m$ we get:

$$\begin{aligned}\log(f(x_o|\theta_{m+1})) - \log(f(x_o|\theta_m)) &= \\ &Q(\theta_{m+1}|\theta_m, x_o) - Q(\theta_m|\theta_m, x_p) - (H(\theta_{m+1}|\theta_m, x_o) - H(\theta_m|\theta_m, x_o)) \ .\end{aligned}$$

We set

$$\theta_{m+1} = \underset{\theta}{\operatorname{argmax}}\, Q(\theta|\theta_m, x_o) \ , \tag{3.11}$$

i.e. the maximizer of the first term involving the complete data likelihood. Then, the first term by definition is non-negative, and we have already shown that the second term is non-negative. Together, these two conditions imply that $\log(f(x_o|\theta_{m+1})) \ge \log(f(x_o|\theta_m))$.

We can summarize the EM algorithm as follows:

Algorithm 30 (EM Algorithm) *Choose an initial value for θ_0 and set $m = 0$.*

1. **Expectation Step**: *Compute*

$$Q(\theta|\theta_m, x_o) = E[\log(f(x_o, x_m|\theta))|\theta_m, x_o] \ .$$

2. **Maximization Step**: *Set*

$$\theta_{m+1} = \underset{\theta}{\operatorname{argmax}}\, Q(\theta|\theta_m, x_o) \ .$$

3. *Check convergence. If not converged, set $m = m + 1$ and go to Step 1.*

Properties of the resulting sequence $\{\theta_m\}$ can be characterized by the following theorem.

Theorem 6 1. *For EM algorithm where θ_{m+1} is given by Eqn. 3.11, we have*

$$\log(f(x_o|\theta_{m+1})) \ge \log(f(x_o|\theta_m)) \ ,$$

with equality if and only if

$$Q(\theta_{m+1}|\theta_m, x_o) = Q(\theta_m|\theta_m, x_o) \ .$$

2. *If $Q(\theta|\theta_0, x_o)$ is continuous in both θ and θ_0, then every limit point of an EM sequence is a stationary point of the (incomplete data) likelihood function $f(x_o|\theta)$, and $f(x_o|\theta_m)$ converges monotonically to $f(x_o|\hat{\theta})$ for a stationary point $\hat{\theta}$.*

3.7.2 Some Examples of EM Algorithm

There are some well known examples of using EM algorithms for finding maximum likelihood estimates of parameters. In this section, we discuss two of them, namely, estimation of multinomial parameter(s) and estimation of Gaussian mixture parameters.

1. **Multinomial Case**: Consider a multinomial distribution parameterized by θ according to:

$$(x_1, x_2, x_3, x_4) \sim \mathcal{M}(n; 0.5 + 0.25\theta, 0.25(1-\theta), 0.25(1-\theta), 0.25\theta) \ .$$

The likelihood function of θ given the vector x is:

$$f(x|\theta) = \left(\begin{array}{c} n \\ x_1\ x_2\ x_3\ x_4 \end{array} \right) (0.5 + 0.25\theta)^{x_1} (0.25(1-\theta))^{x_2+x_3} (0.25\theta)^{x_4} \ .$$

Hence, the log-likelihood function is:

$$\log(f(x|\theta)) \propto x_1 \log(2+\theta) + (x_2 + x_3)\log(1-\theta) + x_4 \log(\theta) \ .$$

Define two new random variables y_1 and y_2 such that $x_1 = y + 1 + y_2$, where $(y_1, y_2) \sim \mathcal{M}(x_1; 0.5, 0.25\theta)$. Set the complete data to be $y = (y_1, y_2, x_2, x_3, x_4)$. The vector y has multinomial distribution according to:

$$y \sim \mathcal{M}(n; 0.5, 0.25\theta, 0.25(1-\theta), 0.25(1-\theta), 0.25\theta) \ .$$

The complete data log-likelihood is proportional to:

$$(y_2 + x_4)\log(\theta) + (x_2 + x_3)\log(1-\theta) \ .$$

   (a) **E-Step**: Its expectation with respect to the density function $f(y_1, y_2|\theta_0, x_1, x_2, x_3, x_4)$ is given by:

$$Q(\theta|\theta_0, y) = (\frac{\theta_0}{2+\theta_0} x_1 + x_4)\log(\theta) + (x_2 + x_3)\log(1-\theta) \ .$$

   Here, we have used the fact that $E[y_2|\theta_0, x_1] = \frac{\theta_0}{2+\theta_0} x_1$. Recall that

$$(y_1, y_2) \sim \mathit{binomial}(x_1; 0.5/(0.5 + 0.25\theta), 0.25\theta/(0.5 + 0.25\theta))$$

   and the conditional mean of y_2 is $\frac{x_1(0.25\theta)}{0.5+0.25\theta}$.

   (b) **M-Step**: To perform the maximization step, we maximize $Q(\theta|\theta_0, y)$ by taking its derivative and setting it equal to zero. Solving for θ_{m+1}, we obtain:

$$\theta_{m+1} = \frac{\frac{\theta_m x_1}{2+\theta_m} + x_4}{\frac{\theta_m x_1}{2+\theta_m} + x_2 + x_3 + x_4} \ .$$

   This results in the EM algorithm for finding MLE of θ.

2. **Gaussian mixture case**: Remember that the observed data in this case is $x_0 = Y$, the observations of mixture variable, and the missing data is $x_m = l$, the set of labels associated with elements of Y. Also, note that the missing data is a discrete random variable and hence we use a probability mass function instead of a probability density function. The complete data log-likelihood function for this case is given in Eqn. 3.10. Using that form, we proceed to deriving the EM algorithm:

(a) **E Step**: This step involves computing the expectation of the log-likelihood function with respect to the probability $P(l|Y,\theta)$, where l is an n-vector of labels. The resulting expectation is given by:

$$\begin{aligned} Q(\theta|\theta_m, Y) &= E[\log(f((l,Y)|\theta))|Y,\theta_m] \\ &= \sum_l \log(f((l,Y)|\theta))P(l|Y,\theta_m) \\ &= \sum_l \left(\sum_{i=1}^n \log(f((l_i,Y_i)|\theta))P(l_i|Y_i,\theta_m) \right) \end{aligned}$$

Switching the two summations, we get:

$$\begin{aligned} Q(\theta|\theta_m, Y) &= \sum_{i=1}^n \left(\sum_{l_i=1}^2 \log(f((l_i,Y_i)|\theta))P(l_i|Y_i,\theta_m) \right) \\ &= \sum_{i=1}^n \left(\sum_{j=1}^2 \log(\alpha_{j,m} f_j(Y_i|\mu_j,\sigma_j^2))P(j|Y_i,\theta_m) \right) \qquad (3.12) \end{aligned}$$

(b) **M Step**: Maximizing the expected log-likelihood function $Q(\theta|\theta_m)$, we obtain the following updates:

$$\alpha_{j,m+1} = \frac{1}{n}\sum_{i=1}^n P(j|Y_i,\theta_m) \qquad (3.13)$$

$$\mu_{j,m+1} = \frac{\sum_{i=1}^n Y_i P(j|Y_i,\theta_m)}{\sum_{i=1}^n P(j|Y_i,\theta_m)} \qquad (3.14)$$

$$\sigma_{j,m+1} = \sqrt{\frac{\sum_{i=1}^n (Y_i - \mu_{j,m+1})^2 P(j|Y_i,\theta_m)}{\sum_{i=1}^n P(j|Y_i,\theta_m)}} \qquad (3.15)$$

where

$$P(j|Y_i,\theta_m) = \frac{\alpha_{j,m} f_j(Y_i|\mu_{j,m},\sigma_{j,m}^2)}{\sum_{j=1}^2 \alpha j, m f_j(Y_i|\mu_{j,m},\sigma_{j,m}^2)}$$

Additional examples of use of EM algorithm are provided through the exercises at the end of this chapter.

3.8 Miscellaneous Topics

3.8.1 Bayesian Inference

So far we have discussed computing the maximum likelihood estimate of an unknown parameter b either by seeking a root of the first derivative of the likelihood function (the score function) or by direct optimization techniques. Another situation which arises frequently in statistics is Bayesian inference. The parameter is assumed to be random with some known distribution and its estimate ends up being an expected value under the posterior distribution.

If b is a random parameter with some known distribution, then according to the Bayes' rule, the posterior distribution is given by

$$\begin{aligned} P(b|\mathbf{y}) &= \frac{P(\mathbf{y}|b)P(b)}{P(\mathbf{y})} \\ &\propto P(\mathbf{y}|b)P(b) \ , \end{aligned}$$

where $\mathbf{y}$ is the dependent random variable. Given an observation $\mathbf{y}$, the posterior is proportional to the product of the data likelihood and the prior probability on the parameter b. The next question is: what should be the criterion for estimating b from observed $\mathbf{y}$? Under the posterior probability, there are several cost functions that can be minimized, resulting in different values of the estimate. We briefly mention a few commonly used cost functions. Let $\| \cdot \|$ denote the Euclidean distance function on $\mathbb{R}^n$.

1. The maximizer of the posterior density is called the maximum a-posterior (MAP) estimate:
$$\hat{b}_{MAP} = \underset{b}{\operatorname{argmax}}\, P(b|\mathbf{y}) \ .$$
The MAP estimate can be found using techniques which were described to find the maximum-likelihood estimate, except the posterior density is maximized instead of the likelihood function.

2. Let b be the value of the estimator. Then the expected squared error is given by
$$\int_{b_1} \|b - b_1\|^2 P(b_1|\mathbf{y}) db_1 \ .$$
The value of b which minimizes this expected squared error is called the minimum mean squared error estimator (MMSE).
$$\hat{b}_{MMSE} = \underset{b}{\operatorname{argmin}}\, \text{Expected Squared Error} \ .$$
It can be shown that for Euclidean parameters, the MMSE estimator is given by the mean under the posterior (conditional) distribution. That is,
$$\hat{b}_{MMSE} = \int_b bP(b|\mathbf{y}) db \ . \tag{3.16}$$
This estimator involves computing integral of the posterior density. In case the analytical solutions are difficult to evaluate numerical techniques for integration can be used. In the next chapter (Chapter 4) we will describe some commonly used numerical techniques for numerical integrations.

3. The disadvantage of minimizing expected squared error is that it results in an averager: the solution is the average of high posterior probability points. Even the individual points may be high probability, their average itself can be a low-probability point and hence a bad estimate. Therefore, very often it is the expected error which is minimized instead of the expected squared error:
$$\int_{b_1} \|b - b_1\| P(b_1|\mathbf{y}) db_1 \ .$$
For Euclidean parameters the minimum expected absolute error criterion results in the median of the posterior probability.

Several of these solution require solving optimization problems, and the techniques discussed in this chapter are applicable. Others, such as computation of conditional mean in Eqn. 3.16, require the use of sampling algorithms covered later in Chapters 6-7.

3.9 Problems

1. Consider Problem 12 from Chapter 2.

   (a) The linear estimator can be expressed as $\hat{v_1} = Bv_2$ where B is the inverse of $(I_3 - A)$. Verify that the estimator $\hat{v_1}$ is unbiased.

   (b) Derive the expression for Cramer-Rao lower bounds on the expected error $E((\hat{v_1} - v_1)(\hat{v_1} - v_1)^T)$. Compare this value with your computed value of the error. Remember that you computed $\|\hat{v_1} - v_1\|$ and not the whole matrix.

2. Write a matlab program to compute a root of the following function using simple iterations:

$$f(x) = sin(x) - x \ .$$

   Choose $x = \pi/4$ as the starting value for x. Write a matlab program to solve the same problem using Newton-Raphson method.

3. For finding the roots of the function $f(x) = e^{-x} - cos(x)$,

   (a) Write a matlab program to implement Newton's method Run this program twice with the starting values $+1$ and -1.

   (b) Write a matlab program to implement Secant method. Run this program with the starting values $x_{-1} = 1$ and $x_0 = 1.5$.

   Plot the convergence of the sequence to the root in each of these cases.

4. Let y be a random variable defined by the linear equation,

$$y = 5 * b + w$$

   where w is a normal random variable with mean 4 and variance 2, that is $w \sim N(4, 2)$.

   (a) **Observe** y: Generate a realization of w in matlab using

$$w = 4 + sqrt(2) * randn; \ .$$

   Use this value of w to calculate the value of y according to the above linear equation.

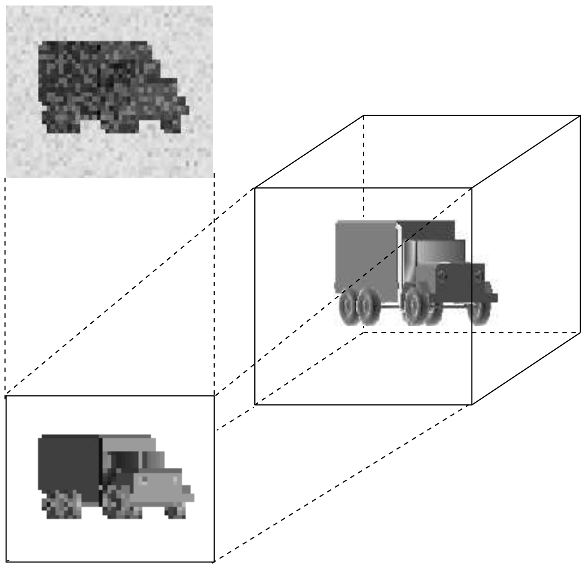
   (b) **Estimate** b: Based on this value of y, calculate the maximum likelihood estimate of b. (First write down the likelihood function of b and then find that value of b which maximizes it.) Remember that for a Gaussian random variable with mean μ and standard deviation σ, the probability density function is given by

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(\frac{-1}{2\sigma^2}(x - \mu)^2) \ .$$

5. Use the bracketing method to find the roots of the polynomial

$$x^3 - 12x^2 + 48x - 64 \ .$$

   Use $x_1 = 1.5$ and $x_2 = 5.0$ as the starting boundary points.



6. Assume a chair being pictured by a remote camera. Chair position, camera position and other nuisance parameters are assumed known with the only variable being the chair orientation angle. The model for camera output is given by the equation

   $$Y = F(\theta) + N, \quad Y, N \in \mathbb{R}^{64\times 64}$$

   where θ is the relative orientation between the camera and the chair, and F is the orthographic projection capturing the target profile (as shown in Figure). Assume that F is twice continuously differentiable and N is additive Gaussian noise with i.i.d elements $\sim N(0, \sigma^2)$. F is not available analytically but sample images $F(\theta_i)$ are given for $\theta_i = 2\pi i/360$ $i = 1, 2, \ldots 360$.

   Given an observation Y find the maximum-likelihood estimate of θ using: (i) rescaled simple iterations, and (ii) Newton's method. Try the following initial conditions: (i)0, (ii) $\pi/4$, (iii) $3\pi/4$, and (iv) $\pi/2$.

7. Assume a chair being pictured by a remote camera. Chair position, camera position and other nuisance parameters are assumed known with the only variable being the chair orientation angle. The model for camera output is given by the equation

   $$Y = F(\theta) + N, \quad Y, N \in \mathbb{R}^{64\times 64}$$

   where θ is the relative orientation between the camera and the chair, and F is the orthographic projection capturing the target profile. Assume that N is additive Gaussian noise with i.i.d elements $\sim N(0, \sigma^2)$. F is not available analytically but sample images $F(\theta_i)$ are given for $\theta_i = 2\pi i/M$ $i = 1, 2, \ldots M$, $M = 360$.

   Given an observation Y, following is an algorithm to calculate the Hilbert-Schmidt (H-S) estimate (minimum-mean squared error estimate) of the chair angle.

Algorithm 31 *(a) Calculate* $p_i = e^{-\|Y - F(\theta_i)\|^2/2\sigma^2}$, $i = 1, 2, \ldots, M$.

(b) Normalize them in to probabilities, $\tilde{p}_i = \frac{p_i}{\sum_{i=1}^M p_i}$.

(c) Compute the average matrix, $A = \sum_i^M O_i \tilde{p}_i$, *where* $O_i = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) \\ sin(\theta_i) & cos(\theta_i) \end{bmatrix}$.

(d) *The Hilbert-Schmidt estimate is given by*

$$\hat{O}_{HS} = \{ \begin{array}{cc} \frac{1}{det(A)} A & if \;\; det(A) \neq 0 \\ any \;\; O \in SO(2) & o.w. \end{array} .$$

(e) *Calculate* $\hat{\theta}_{HS}$ *from* $\hat{O}_{HS}$.

Prob: Generate a noisy image Y for $\theta_t = 60$deg. Use this algorithm to compute the H-S estimate, $\hat{\theta}_{HS}$, from Y.
Remark: If $p_i = 1/M$ for all i, then $\hat{\theta}_{HS}$ represents the "mean" of θ_i's with respect to the Hilbert-Schmidt metric.

8. Consider an extension of Problem 12 in Chapter 2 as follows. Let $v(n) \in \mathbb{R}^3$ be the state variables and $y(n)$ be the observations, satisfying the equations,

$$\begin{aligned} v(n) &= (I - A)v(n-1) + W_1 \\ y(n) &= Bv(n) + W_2 \end{aligned}$$

where W_1, W_2 are the Gaussian vectors with zero mean and $\sigma_1^2 I$, $\sigma_2^2 I$ covariances, respectively. A is a skew-symmetric matrix with entries $(+or-)0.1$. Let v_1 be Gaussian with mean $[1\ 1\ 1]^T$ and covariance $\sigma_1^2 I$. Choose $\sigma_1 = \sigma_2 = 0.1$. Matrix B is given by

$$B = \begin{bmatrix} 1 & 0.1 & 0.5 \\ -0.3 & 2 & 1.0 \\ 0.01 & 0.5 & 2.5 \end{bmatrix} .$$

(a) For $n = 2, \dots, 5$ generate a sequence of $v(n)$'s.

(b) Using $v(n)$'s generate the observations $y(n)$'s.

(c) Derive the discrete Kalman filter formulas for conditional mean and conditional covariance estimates. Use these formulas to estimate the conditional mean and covariances for $v(5)$.

9. Write a matlab program to find the **fixed-point(s)** of the following function using simple iterations:

$$g_p(x) = \frac{1}{p}\left((p-1)x + \frac{78.8}{x}\right) ,$$

for (i) p = 1.5, and (ii) p = 2.0; Use $x_1 = 9$ for the first case and $x_1 = 12$ for the second case. For both cases tabulate your results according to:

iter (k)	x_k	$\Delta x_k = x_{k+1} - x_k$	$\frac{\Delta x_k}{\Delta x_{k-1}}$	$\frac{\Delta x_k}{(\Delta x_{k-1})^2}$
1				
2	...			

Also, plot the values $g_p(x_k)$ versus k. Based on these results state the order of convergence for the two cases.

Run your program form $p = 0.5$ and verify that it does not converge.

10. In the case of Newton-Raphson and Secant methods for root finding, the sequences converge to a root only linearly if that root has a multiplicity greater than one. For roots with multiplicity greater than one, Newton-Raphson satisfies the equation

$$E_{i+1} \sim KE_i, \quad E_i = |x_i - x_{i-1}| \text{ and } \quad K = \frac{m-1}{m} \ ,$$

when m is the multiplicity of the root. Write a matlab program to find the multiplicity of root $x = -0.5$ of the polynomial

$$x^5 - 4.5x^4 + 4.55x^3 + 2.675x^2 - 3.3x - 1.4375 \ .$$

11. Use the bracketing methods (Bisection and Secant) to find the roots of the functions:

    (a) $x^2 - 78.8$. Use 0 and 10 as the starting points.

    (b) $x^3 - 12x^2 + 48x - 64$. Use $x_1 = 1.5$ and $x_2 = 5.0$ as the starting boundary points.

    Plot the sequences for all cases.

12. Using Newton-Raphson's method solve for the minimizer of the cost function $F(x) = \|y - Ax\|_2^2$, where

$$A = \begin{bmatrix} 3 & 3 & 3 & 1 \\ 2 & 1 & 2 & 1 \\ 1 & 3 & 3 & 1 \\ 3 & 2 & 2 & 3 \end{bmatrix}, \quad y = \begin{bmatrix} 52 \\ 28 \\ 44 \\ 48 \end{bmatrix} \ .$$

13. Solve the above problem using Gauss-Seidal-Newton's method.

14. Use Newton-Raphson's method to solve for the roots of the function

$$f(x, y) = [\begin{matrix} ye^{-x} - 2 \\ x^2 + y - 4 \end{matrix}] \ .$$

    Choose $[x \ y] = [1 \ 1]$ as the starting point.

15. Solve for the maximizer of F using the sequential grid-search where

$$F(x_1, x_2, x_3) = \exp(-\sum_{i=1}^{3}(x_i - \frac{1}{i})^2) \ .$$

16. Let $X_1, X_2, \ldots, X_n$ be independent and identically distributed samples from a logistic distribution with the probability density function

$$f(x|\theta) = \frac{\exp(-(x - \theta))}{(1 + \exp(-(x - \theta)))^2} \ .$$

    Given the values of $X_1, X_2, \ldots, X_n$, our goal is to find the maximum likelihood estimate (MLE) of θ, using the following steps:

    (a) Derive an expression for the log likelihood function

$$l(\theta) = \sum_{i=1}^{n} \log(f(X_i|\theta)) \ ,$$

    such that the MLE is given by $\hat{\theta} = \text{argmax}_\theta \, l(\theta)$.

(b) Find the expression for $\dot{l}(\theta)$, the derivative of l with respect to θ. Use Newton-Raphson method to find a root of $\dot{l}(\theta)$. Call it $\hat{\theta}$.

(c) Verify that $\ddot{l}(\hat{\theta}) < 0$.

Download a sample of X from the class website.

17. Let the random variables $X_1, X_2, \ldots, X_n$ be independent and identically distributed according to the probability density function:

$$f(x) = \frac{1}{\Gamma(p)} \exp(-x) x^{p-1}, \quad x \geq 0 \ .$$

Our goal is to estimate the unknown value p given the observed values of $X_1, \ldots, X_n$, using the maximum likelihood criterion. The likelihood of p is given by:

$$\begin{aligned} f(x_1, x_2, \ldots, x_n | p) &= \prod_{i=1}^{n} f(x_i | p) \\ &= \frac{1}{\Gamma(p)^n} \exp(-\sum_{i=1}^{n} x_i)(x_1 x_2 \ldots x_n)^{p-1} \\ &= \frac{1}{\Gamma(p)^n} \exp(-a) b^{p-1} \ , \quad \text{where } a = \sum_{i=1}^{n} x_i, \ b = \prod_{i=1}^{n} x_i \ . \end{aligned}$$

So the maximum likelihood estimate (MLE) of p is given by:

$$\hat{p} = \underset{p}{\operatorname{argmax}} \left(\frac{1}{\Gamma(p)^n} \exp(-a) b^{p-1} \right) = \underset{p}{\operatorname{argmax}} (-n \log(\Gamma(p)) + (p-1) \log(b)) \ .$$

Find the MLE of p for the following data:

$$n = 10, \ X = [2.06 \ 2.83 \ 4.84 \ 3.89 \ 8.74 \ 6.32 \ 3.67 \ 7.03 \ 7.16 \ 3.57] \ .$$

18. For the multinomial example studied in the class, estimate the parameter θ using the EM algorithm for the observation $(x_1, x_2, x_3, x_4) = (125, 18, 20, 30)$.

19. Consider the model where X_i's are independently distributed as

$$X_i \sim \theta g(x) + (1 - \theta) h(x) \ ,$$

where g and h are known probability densities. Our goal is to find MLE of θ given the observations $(x_1, x_2, \ldots, x_n)$. Derive an EM algorithm using the following steps:

(a) Introduce the variables $Z_1, Z_2, \ldots, Z_n$, where Z_i indicates from which density X_i has been drawn. (That is, $Z_i = 1$ or 0. If $Z_i = 1$, then $X_i \sim g(x)$ and if $Z_i = 0$ then $X_i \sim h(x)$).

(b) Show that the complete data likelihood can be written as:

$$\prod_{i=1}^{n} (z_i g(x_i) + (1 - z_i) h(x_i)) \theta^{z_i} (1 - \theta)^{(1 - z_i)} \ .$$

(c) Show that the conditional expectation of Z_i is:

$$E[Z_i|\theta_0, x_i] = \frac{\theta_0 g(x_i)}{\theta_0 g(x_i) + (1-\theta_0)h(x_i)} .$$

(d) Write down the EM algorithm for estimation of θ.

Implement this algorithm for g being $N(0,1)$ and h being normal $N(5,1)$, and the following observations:

$$(0.95, 0.23, 5.45, 0.60, 5.44, 0.48, 0.49) .$$

20. **Mixture of Gaussians**: Let Y be a continuous random variable with probability density function:

$$Y \quad \sim \quad \alpha_1 f_1(y; \mu_1, \sigma_1^2) + \alpha_2 f_2(y; \mu_2, \sigma_2^2) ,$$

where f_1 and f_2 are two Gaussian density functions with means μ_1, μ_2 and variances σ_1^2, σ_2^2, respectively. Also, $0 \leq \alpha_1, \alpha_2 \leq 1$, such that $\alpha_1 + \alpha_2 = 1$. Given n observations of Y, our goal is to find maximum likelihood estimate of $\theta = (\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \alpha_1, \alpha_2)$.

We will use the EM algorithm for this estimation. Let $\theta^(m)$ be the current values of the unknown. Then, the update for $\theta^{(m+1)}$ is given by:

$$\begin{aligned}
\alpha_l^{(m+1)} &= \frac{1}{n}\sum_{i=1}^{n} P(l|\theta^{(m)}, Y_i) \\
\mu_l^{(m+1)} &= \frac{\sum_{i=1}^{n} Y_i P(l|\theta^{(m)}, Y_i)}{\sum_{i=1}^{n} P(l|\theta^{(m)}, Y_i)} \\
\sigma_l^{(m+1)} &= \sqrt{\frac{\sum_{i=1}^{n} (Y_i - \mu_l^{(m+1)})^2 P(l|\theta^{(m)}, Y_i)}{\sum_{i=1}^{n} P(l|\theta^{(m)}, Y_i)}}
\end{aligned}$$

where

$$P(l|\theta^{(m)}, Y_i) = \frac{\alpha_l^{(m)} f_l(Y_i; \mu_l^{(m)}, (\sigma_l^{(m)})^2)}{\sum_{l=1}^{2} \alpha_l^{(m)} f_l(Y_i; \mu_l^{(m)}, (\sigma_l^{(m)})^2)} .$$

Download two datasets from the class website to apply to this problem. For each data:

(a) Plot a histogram of the data using the *hist* function in matlab.

(b) Using some initial values guessed from the histogram, apply EM algorithm to estimate the unknown parameters.

(c) Plot the evolution of the observed data log-likelihood function versus the iteration index. The observed data log-likelihood function is:

$$\sum_{i}^{n} \log\left(\alpha_1 f_1(Y_i; \mu_1^{(m)}, (\sigma_1^{(m)})^2) + \alpha_2 f_2(Y_i; \mu_2^{(m)}, (\sigma_2^{(m)})^2)\right) .$$

3.10 References

The material presented in this chapter is taken from the texts [9, 5].

Chapter 4

NUMERICAL INTEGRATION

4.1 Introduction

Integration is fundamental to statistical inference. Evaluation of probabilities, means, variances, and mean squared error can all be thought of as integrals. Very often it is not feasible to solve for the integral of a given function via analytical techniques and alternative methods are adapted. In situations where the approximate answers are acceptable, the numerical techniques can be applied. In this chapter we will study some techniques for numerically approximating integrals of real valued functions. These numerical techniques for integration are also often referred to as *quadrature* methods.

To understand these numerical techniques, we start with a discussion of Riemann integral. Consider the definition of Riemann Integral of a function $f(x)$ on an interval $[a, b]$. Divide $[a, b]$ into n-disjoint intervals Δx_i, such that $\cup_i \Delta x_i = [a, b]$ and $\{\Delta x_i\}$ is called a *partition* P of $[a, b]$. The mesh of this partition is defined to be the largest size of sub-intervals, $mesh(P) = max_i |\Delta x_i|$. Define a finite sum,

$$S_n = \sum_{i=1}^{n} f(x_i)\Delta x_i \ ,$$

where $x_i \in \Delta x_i$ is any point. If the quantity $\lim_{meshP \downarrow 0} S_n$ exists, then it is called the **integral of** f on $[a, b]$ and is denoted by $\int_a^b f(x)dx$. This construction demonstrates that any numerical approximation of $\int_a^b f(x)dx$ will have two features: (i) some selection of samples points which partition the interval, and (ii) a finite number of function evaluations on these sample points.

4.2 Quadrature Integration

Our goal is to approximate $\int_a^b f(x)dx$ by using evaluations of f at some sample points, in such a way that more samples imply more accuracy. To simplify the discussion, in this chapter we will restrict to scalar-valued functions of scalar variables.

The underlying idea is to approximate f by a simple function which can be easily integrated on $[a, b]$ and which agrees with f on the sampled points. The technique of finding a smooth curve passing through a set of points is also called *curve fitting.* The most common technique is to sample $n + 1$ points and find an order-n polynomial passing through those points. Then, the integral of f over that region (containing $n+1$-points) can be approximated by the integral of the polynomial over the same region. Given $n+1$ sample points there is a unique polynomial passing through these points, though there are several methods to obtain it. We will use the **Lagrange's method** to find this polynomial, which we will call Lagrange's interpolating polynomial.

4.2.1 Basic Formulation

If we try to fit a polynomial to the sample points over the whole interval $[a, b]$ we may end up with a high order polynomial which itself might be difficult to integrate. Therefore, to start with, we focus on a smaller region in $[a, b]$, lets say $[x_k, x_{k+n}]$, containing the points $x_k,\ x_{k+1},\ \ldots, x_{k+n}$. Let $P_{k+i} \equiv (x_{k+i}, f(x_{k+i}))$ be the pairs of the sampled points and the function values; they are called *knots*. Let $p_{k,k+n}(x)$ denote the polynomial of degree less than or equal to n that interpolates $P_k, P_{k+1}, \ldots, P_{k+n}$. With this notation, our goal is the following

Given $P_k, \ldots, P_{k+n}$ find the polynomial $p_{k,k+n}(x)$ such that

$$p_{k,k+n}(x_{k+i}) = f(x_{k+i}), \quad 0 \le i \le n \ .$$

To understand the construction of $p_{k,k+n(x)}$ first we look at some examples for small values of n.

1. In case of $n = 0$, that is, given only one point P_k, the only choice is a constant function, $p_{k,k}(x) = f(x_k)$.

2. In case of $n = 1$, we are given P_k and P_{k+1}, $p_{k,k+1}(x)$ is a first order polynomial (or a straight line) passing through these two points. The equation of a straight line passing through P_k and P_{k+1} is given by:

$$\begin{aligned} p_{k,k+1}(x) &= f(x_k) + \frac{f(x_{k+1}) - f(x_k)}{x_{k+1} - x_k}(x - x_k) \\ &= f(x_k)\left(\frac{x - x_{k+1}}{x_k - x_{k+1}}\right) + f(x_{k+1})\left(\frac{x - x_k}{x_{k+1} - x_k}\right) \end{aligned}$$

3. Extending these ideas to an arbitrary value of n, the polynomial takes the form

$$p_{k,k+n}(x) = \sum_{i=0}^{n} f(x_{k+i}) L_{k+i}(x), \quad \text{where} \quad L_{k+i}(x) = \prod_{j=0, j\neq i}^{n} \frac{(x - x_{k+j})}{(x_{k+i} - x_{k+j})} \ .$$

   Notice that $L_{k+i}(x) = \begin{cases} 1, & x = x_{k+i} \\ 0, & x = x_{k+j}, j \neq i \\ \textit{else}, & \textit{in between} \end{cases}$. Therefore, $p_{k,k+n}(x)$ satisfies the requirement that it should pass through the $n+1$ knots and is an order-n polynomial.

The basic idea is to select n-adjacent points and interpolate them with n-the order polynomial. $f(x)$, over the region $[x_k, x_{k+n}]$, is now approximated by $p_{k,k+n}(x)$ for the purpose of numerical integration.

$$f(x) \sim p_n(x) = f(x_k)L_k(x) + f(x_{k+1})L_{k+1}(x) + \ldots + f(x_{k+n})L_{k+n}(x) \ ,$$

where $L_{k+i}(x) = \prod_{j=0,j\neq i}^{n} \frac{(x - x_{k+j})}{(x_{k+i} - x_{k+j})}$. We will now approximate $\int_{x_k}^{x_{k+n}} f(x)dx$ by the quantity $\int_{x_k}^{x_{k+n}} p_{k,k+n}(x)dx$.

$$\int_{x_k}^{x_{k+n}} f(x)dx \quad \sim \quad \int_{x_k}^{x_{k+n}} p_{k,k+n}(x)dx$$

$$\begin{aligned} &= f(x_k)\int_{x_k}^{x_{k+n}} L_k(x)dx + f(x_{k+1})\int_{x_k}^{x_{k+n}} L_{k+1}(x)dx + \ldots + f(x_{k+n})\int_{x_k}^{x_{k+n}} L_{k+n}(x)dx \\ &= w_k f(x_k) + w_{k+1}f(x_{k+1}) + \ldots + w_{k+n}f(x_{k+n}) \ , \end{aligned}$$

where the weights are obtained by integrating the Lagrange's polynomials according to

$$w_{k+i} = \int_{x_k}^{x_{k+n}} L_{k+i}(x)dx \ .$$

We will calculate these weights to derive a family of numerical integration techniques but in a simplified setup. We will assume that the sample points $x_k,\ x_{k+1}, \ldots,$ are uniformly spaced with the spacing $h > 0$. Any point $x \in [x_k, x_{k+n}]$ can now be represented by $x = x_k + sh$, where s takes values $1,\ 2,\ 3, \ldots, n$ at the sample points and other values in between. With this assumption the Lagrange polynomials and the evaluation of the weights simplify:

$$L_{k+i}(x_k + sh) = \prod_{j=0, j\neq i}^{n} \frac{(s-j)}{(i-j)}$$

Changing the variable of integration from x to s using $x = x_k + sh$ and $dx = hds$, we obtain

$$\int_{x_k}^{x_{k+n}} f(x)dx \sim \sum_{i=0}^{n} f(x_{k+i}) \int_0^n L_{k+i}(x_k + sh)hds \ .$$

So the weights are now given by

$$w_{k+i} = h\int_0^n L_{k+i}(x_k + hs)ds \ . \tag{4.1}$$

The polynomials $L_{k+i}(s)$ can be evaluated and integrated before-hand resulting in some well-known integration rules such as trapezoidal, Simpson's etc. The basic difference between these methods is the order of the polynomial chosen (or the number of knots grouped together) and the corresponding region of integration. Following is a table for the interpolating polynomials:

n	$L_{k+1}(x_k + hs)$
1	$L_k = s$
	$L_{k+1} = 1 - s$
2	$L_{k+0} = \frac{1}{2}(s^2 - 3s + 2)$
	$L_{k+1} = 1(s^2 - 2s)$
	$L_{k+2} = \frac{1}{2}(s^2 - s)$
3	$L_{k+0} = \frac{-1}{6}(s^3 - 6s^2 + 11s - 6)$
	$L_{k+1} = \frac{1}{2}(s^3 - 5s^2 + 6s)$
	$L_{k+2} = \frac{-1}{2}(s^3 - 4s^2 + 3s)$
	$L_{k+3} = \frac{1}{6}(s^3 - 3s^2 + 2s)$

Lagrange polynomials for uniform spacing and different values of n.

Given these polynomials, we can choose different values of n and compute the weights for numerical integration.

1. **For** $n = 0$, the weight is

$$w_k = h\int_0^0 L_k(s)ds = 0 \ .$$

2. **For** $n = 1$, the weights are given by

$$\begin{aligned} w_k &= h\int_0^1 L_k(s)ds \\ &= h\int_0^1 sds = \frac{h}{2} \end{aligned}$$

Similarly,

$$\begin{aligned} w_{k+1} &= h\int_0^1 L_{k+1}(s)ds \\ &= h\int_0^1 (1-s)ds = \frac{h}{2} \end{aligned}$$

The integral value is given by

$$\int_{x_k}^{x_{k+1}} f(x)dx \sim \frac{h}{2}(f(x_k) + f(x_{k+1}))\ .$$

This approximation is called the **Trapezoidal rule**, because the integral is equal to the area of the trapezoid formed by the two knots.

3. **For** $n = 2$, the weights are given by

$$\begin{aligned} w_k &= h\int_0^2 L_k(s)ds \\ &= h\int_0^2 \frac{1}{2}(s^2 - 3s + 2)ds = \frac{h}{3} \end{aligned}$$

$$\begin{aligned} w_{k+1} &= h\int_0^2 L_{k+1}(s)ds \\ &= h\int_0^2 (s^2 - 2s)ds = \frac{4h}{3} \end{aligned}$$

$$\begin{aligned} w_{k+2} &= h\int_0^2 L_{k+2}(s)ds \\ &= h\int_0^2 \frac{1}{2}(s^2 - s)ds = \frac{h}{3} \end{aligned}$$

The integral value is given by

$$\int_{x_k}^{x_{k+1}} f(x)dx \sim \frac{h}{3}(f(x_k) + 4f(x_{k+1}) + f(x_{k+2}))\ .$$

This rule is called the **Simpson's** $1/3$ **rule**.

4. **For** $n = 3$, we obtain Simpson's-3/8 rule given by

$$\int_{x_k}^{x_{k+3}} f(x)dx \sim \frac{3h}{8}\left(f(x_k) + 3(f(x_{k+1}) + f(x_{k+2})) + f(x_{k+3})\right) \ . \tag{4.2}$$

A variation of this technique results in many different formulas. Instead of selecting some samples and integrating over the corresponding region, some sample points from the neighboring intervals can also be utilized to improve accuracy. For example, to integrate over $[x_k, x_{k+1}]$, sample x_{k-1} and x_{k+2} are utilized. This results in **central cubic rule**:

$$\int_{x_k}^{x_{k+1}} f(x)dx \sim \frac{h}{24}\left(-f(x_{k-1}) + 13(f(x_k) + f(x_{k+1}) - f(x_{k+2})\right) \ . \tag{4.3}$$

4.2.2 Composite Rules

So far we have derived integration rules for sub-intervals of $[a, b]$, the interval which we are interested in. Considering the whole space, we will divide it into n sub-intervals of equal width. We will utilize a sliding window on $[a, b]$ by including only a small number of these sub-intervals at a time. That is,

$$\int_a^b f(x)dx = \sum_{k=0}^{N/n} \int_{x_k}^{x_{k+n-1}} f(x)dx \ , \tag{4.4}$$

where each of the integrals on the right side can be approximated using the basic rules derived earlier. The summation of basic rules over sub-intervals to obtain an approximation over $[a, b]$ gives rise to **composite rules**. We will discuss two such composite rules for a uniform spacing of h between the sampled points, $x_0 = a,\ x_2,\ x_2, \ldots,\ x_n = b$:

1. **Composite Trapezoidal Rule:** The numerical approximation formula is given by:

$$\int_a^b f(x)dx \sim T[h] = h\left(\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\right) \ . \tag{4.5}$$

   The error is this approximation is given by

$$\frac{-1}{12}h^2 f''(\xi)(b-a), \quad \text{for} \ \ \xi \in (a, b) \ .$$

2. **Composite Simpson's-1/3 Rule:** To obtain a composite Simpson's-1/3 rule the number of samples $n + 1$ should be odd, or the number of intervals should be even. The integral approximation is given by

$$\int_a^b f(x)dx \sim S_{1/3}[h] = \frac{h}{3}\left(f(a) + f(b) + 4\sum_{i \text{ odd}} f(x_i) + 2\sum_{i \text{ even}} f(x_i)\right) \ . \tag{4.6}$$

   The error associated with this approximation is given by

$$\frac{-1}{90} f^{(iv)}(\xi)(b-a)h^4, \quad \text{for} \ \ \xi \in (a, b) \ .$$

4.2.3 Richardson's Improvement Formula

We have derived an approximation to the integral $\int_a^b f(x)dx$ base on some number of samples of f taken h-distance apart. Let

$$Q = \int_a^b f(x)dx \ .$$

and $F[h]$ be an approximation computed using h-spacing. Then,

$$Q = F[h] + Ch^n + O(h^m) \ , \tag{4.7}$$

where C is a constant and $m > n$. The term $O(h)$ roughly implies that $\frac{O(h)}{h}$ is bounded and, hence, $O(h)$ goes to zero at least linearly as h goes to zero. Clearly, $O(h^m)$ goes to zero faster than $O(h^n)$ as h goes to zero. In other words, $O(h^n)$ is a bigger error term than $O(h^m)$. Richardson's formula is useful in the situations where there is a separation between n and m in Eqn. 4.7. It eliminates the error term Ch^n by evaluating Eqn. 4.7 for two different values of h and mixing the results appropriately.

Assume that Eqn. 4.7 is evaluated for two values of h: h_1 and h_2. Let $h_2 > h_1$ and $\frac{h_2}{h_1} = r$ where $r > 1$. For the sample spacing given by h_2 or rh,

$$Q = F[rh_1] + Cr^n h_1^n + r^m O(h_1^m) F[rh_1] + Cr^n h_1^n + O(h_1^m) \tag{4.8}$$

because since r is constant $r^m O(h^m)$ is still $O(h^m)$. Multiplying Eqn. 4.7 by r^n and subtracting from it the Eqn. 4.8, we obtain,

$$r^n Q - Q = r^n F[h_1] - F[rh_1] + O(h_1^m) \ .$$

Rearranging,

$$Q = \frac{r^n F[h] - F[rh]}{r^n - 1} + O(h^m) \ . \tag{4.9}$$

The first term on the right can now be used as an approximation for Q with the error term given by $O(h^m)$. This removal to Ch^n from the error using two evaluations of $f[h]$ at two different values of h is called **Richardson's Improvement Formula**. This result when applied to numerical integration is called **Romberg's Integration**.

We will analyze Romberg's Integration in the context of composite trapezoidal rule. Remember that the error term in the composite trapezoidal rule is given by

$$C_1 h^2 + C_2 h^4 + C_3 h^6 + \ldots \ .$$

For a h-spaced sampling of the interval $[a, b]$, the composite trapezoidal approximation is

$$T[h] = h\left(\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\right) \ .$$

Let $h_2 = 2h$ be the other spacing for applying Richardson's improvement formula, i.e. $r = 2$. Then, according to the improvement formula, the result, denoted by $T_1[h]$, is given by

$$T_1[h] = \frac{4T[h] - T[2h]}{3} \ . \tag{4.10}$$

It can be shown that the resulting expression for $T_1[h]$ is identical to the composite rule for Simpson's-1/3 rule. $T[2h]$ can be calculated first and then the more accurate evaluation is computed recursively using,

$$T[h] = \frac{1}{2}(T[2h] + 2h \sum_{i \text{ odd}} f(x_i)) \ . \tag{4.11}$$

For the first improvement formula, $T_1[h]$, the error is a polynomial with the leading term h^4. This idea of utilizing improvement to remove the leading term in the error can be carried further by evaluating $T_1[h]$ for two different values of h and applying the improvement formula again. In general, for the i^{th}-column the formula involves two entries from the previous column according to

$$T_i[h] = \frac{4^i T_{i-1}[h] - T_{i-1}[2h]}{4^i - 1} \ .$$

This results in the Romberg's integration table:

h	$T[h]$	$T_1[h]$	$T_2[h]$	$T_3[h]$
h_0	$T[h_0]$			
$h_0/2$	$T[h_0/2]$	$T_1[h_0/2]$		
$h_0/4$	$T[h_0/4]$	$T_1[h_0/4]$	$T_2[h_0/4]$	
$h_0/8$	$T[h_0/8]$	$T_1[h_0/8]$	$T_2[h_0/8]$	$T_3[h_0/8]$
	$O(h^2)$	$O(h^4)$	$O(h^6)$	$O(h^8)$

The accuracy associated with these approximations increases as we move from left to right. Romberg's table is suitable for numerically approximating an integral when the desired accuracy is specified. In each row, the spacing is reduced by half so that the approximation can be obtained in a recursive way from the previous row via Eqn. 4.11. Once the first column is computed, the other columns utilize those values directly without requiring to compute the function samples any more.

4.2.4 Non-Uniform Spacing

So far the integration formulas we have derived assumed that the function samples are obtained at equal spacing. If the function behaves differently in different regions of $[a, b]$ it seems reasonable to choose appropriately different spacings in those regions. There are types of formulas which involve non-uniform spacing: (i) in which the spacing is non-uniform but fixed independent of the function to be integrated, and (ii) in which the spacing changes according to the behavior of the function over sub-regions of $[a, b]$. The first set of methods are called **Gauss Quadrature methods**, while the second set comes under the **Adaptive Quadrature methods**.

We start with a discussion on Gauss quadrature methods. At first, we consider the integration of a given function f on the interval $[-1, 1]$. The goal is to determine n-sample points $(x_1,\ x_2, \ldots, x_n)$ and n weights $(w_1,\ w_2,\ \ldots,\ w_n)$ such that the integral value is approximated by

$$\int_a^b f(x)dx \sim w_1 f(x_1) + w_2 f(x_2) + \ldots + w_n f(x_n) \ . \tag{4.12}$$

Since there are $2n$ unknowns, we need $2n$ equations to solve for them. We obtain these $2n$ equations by imposing a constraint that the sample points and the weights should be chosen in such a way that the result in Eqn. 4.12 is exact for f which are polynomials of order $2n - 1$ or less. In other words, Eqn. 4.12 should be exact for the polynomials $f = 1,\ x,\ x^2,\ \ldots,\ x^{2n-1}$, and their linear combinations. Substituting for these $f = x^i$, $0 \le i \le 2n-1$, we obtain $2n$ equations of the kind

$$w_1 x_1^i + w_2 x_2^i + \ldots + w_n x_n^i = \int_{-1}^{1} x^i dx \ .$$

This is a system of $2n$ non-linear equations involving $2n$ unknowns.

To illustrate the solution of these non-linear equations consider an example for $n = 2$. In this case there are four equations involving four unknowns $(x_1,\ x_2,\ w_1,\ w_2)$:

$$\begin{aligned} w_1 + w_2 &= 2 \\ w_1 x_1 + w_2 x_2 &= 0 \\ w_1 x_1^2 + w_2 x_2^2 &= \frac{2}{3} \\ w_1 x_1^3 + w_2 x_2^3 &= 0 \end{aligned}$$

To solve these equations, we argue that the two points x_1 and x_2 must be symmetrically located in the interval $[-1, 1]$, since this formula will apply to a general function f. We have preference of one side over another. Similarly, if x_1 and x_2 are symmetric with respect to zero, the weights associated with them w_1 and w_2 must also be equal. With the assumptions, we can easily solve for the unknowns obtaining:

$$x_1 = -\frac{1}{\sqrt{3}}, x_2 = \frac{1}{\sqrt{3}}, w_1 = w_2 = 1 \ .$$

Hence the 2-point Gauss quadrature formula is

$$\int_{-1}^{1} f(x)dx = f(-\frac{1}{\sqrt{3}}) + f(\frac{1}{\sqrt{3}}) \ . \tag{4.13}$$

This formula is exact for polynomials of the order three or less.

Similarly, the 3-point Gauss quadrature formula is given by

$$\int_{-1}^{1} f(x)dx = \frac{1}{9}\left(5f(-\sqrt{0.6}) + 8f(0) + 5f(\sqrt{0.6in})\right) \ . \tag{4.14}$$

This formula is exact for the polynomials of the order five or less. The sample points for $n = 4$ are given by ± 0.861136 and ± 0.339981 with the weights given by 0.347854 and 0.652145, respectively.

Remarks:

1. The sample points, x_i's, are located symmetrically around zero in $[-1, 1]$.

2. The density of Gaussian sample points is greater near the end points -1 and 1, as compared to the the middle.

3. The weights w_i's are all positive.

4. The sample points for the n-point rule, $\{x_i\}_{i=1}^{n}$, are not a subset of the sample points for the $(n+1)$-point rule, $\{y_i\}_{i=1}^{n+1}$.

To extend the Gauss quadrature formulas to arbitrary intervals $[a, b]$, we use the transformation

$$y = a + \frac{b-a}{2}(x+1) \ ,$$

which transforms $[-1, 1]$ to $[a, b]$ and $dy = \frac{b-a}{2}dx$. Hence,

$$\int_a^b f(y)dy \ = \ \int_{-1}^1 f(a + \frac{(b-a)}{2}(x+1))\frac{b-a}{2}dx = \frac{b-a}{2}\int_{-1}^1 f(a + \frac{b-a}{2}(x+1))dx$$

Using Gauss quadrature formula, this becomes

$$\frac{b-a}{2}\left(w_1 f(y_1) + w_2 f(y_2) + \ldots + w_n f(y_n)\right) \ , \tag{4.15}$$

where $y_i = a + \frac{b-a}{2}(x_i + 1)$, for x_i's as derived earlier.

4.2.5 Choosing a Method

We have discussed a number of techniques for approximating an integral on a compact interval $[a, b]$. Given a specific situation of requiring numerical integration which technique is most suitable? The answer to that question depends upon two things: in what form the information is provided to us and what is the ultimate concern in approximation. Based on these points an appropriate technique can be selected as summarized below:

1. If the number of sample points (given by n) is specified but their locations (values) can be chosen by the user, then Gauss quadrature formulas as most accurate. This can be a situation where the user is limited by the number of function samples but has choice to selecting the sample points. For the same number of samples, Gauss formula is more accurate than the composite rules because they insist on uniform spacing.

2. If the function samples are given for a specified set of sample points then Gauss quadrature may not apply. The user is supplied with a set of sample points and these may not coincide with the Gauss sample points. In this situation there are two possibilities:

   (a) If the given sample points are uniformly spaced then the composite rules (Trapezoidal, Simpson's, or their combinations) apply.

   (b) If the samples are non-uniform then using Lagrange's interpolating polynomial one can derive the weights for the weighted sum approximation of the integral.

3. Sometimes the ultimate concern is the resulting accuracy and there is no hard limit on the number of sample points and their values. In this case Gauss quadrature may apply but it has the limitation that increasing n renders the old samples useless. The sample points for different n are completely different in Gauss formulas. Therefore, Romberg's table is a preferred method since it uses all the previous function samples and calculates only the other half at any value of n. So one can construct a Romberg's table increasing the number of rows and columns till the desired accuracy is achieved.

4.3 Problems

1. (a) Derive Simpson's $\frac{3}{8}$-rule. Recall that it is a three panel, 4-point algorithm. The Lagrange's interpolating polynomials for $n = 3$ are given in class.

   (b) Evaluate $\int_0^3 x^3 dx$ using Simpson's $\frac{3}{8}$-rule. Comment on the precision of numerical integration.

2. Consider the following two composite rules for proper integrals on $[a, b]$:

   (a) Trapezoidal Rule:

   $$\int_a^b f(x)dx \sim T[h] = h\{\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\}$$

   (b) Simpson's Rule:

   $$\int_a^b f(x)dx \sim S_{1/3}[h] = \frac{h}{3}\{f(a) + f(b) + 4\sum_{odd\ i} f(x_i) + 2\sum_{even\ i} f(x_i)\}$$

   Show that if $T[2h]$ is used in Richardson's formula to improve $T[h]$, i.e. $T_1[h]$, the result is $S_{1/3}[h]$.

3. Form the Romberg Table for $\int_{2.0}^{6.8} e^x dx$ starting from $h_0 = 1.6$ and going up to $h_0/8$. Compare with the exact value.

4. Use 3-point Gauss quadrature formula to evaluate $\int_1^{2.2} ln(x)dx$. Also, evaluate $S_{1/3}[0.3]$ and compare the two results.

5. Previously, the integral $\int_{SO(2)} Op(O|Y)dO$ was evaluated using composite Trapezoidal rule. Now use the composite Simpson's rule to approximate the integral and compare with earlier results.

6. Let $f(x) = e^x$. Evaluate the integral $\int_0^1 f(x)dx$ numerically using Monte-Carlo technique. Plot the error versus the number of samples used.

7. Find a cubic spline passing through the knots $(-2, -8), (0, 0), (1, 1)$ and $(2, 8)$. Choose the end-point conditions as

   (a) $s''(-2) = -12$ and $s''(2) = 12$.

   (b) Assume $\sigma_0 = \sigma_1$ and $\sigma_3 = \sigma_2$.

   (c) Assume $s'(-2) = s'(2) = 12$.

   Plot your results.

8. Estimate $I = \int_0^1 \int_1^{e^x} (x + \frac{1}{y})dydx$. Use 3-point Gauss quadrature for the x interval and 2-point Gauss quadrature for the y interval.

4.4 References

The material presented in this chapter is taken mainly from the textbook by Thisted [9].

Chapter 5

SIMULATING PROBABILITY DISTRIBUTIONS

5.1 Introduction

Simulation of random variables and random processes using computers is among the fastest growing areas of computational statistics. Many statistical techniques rely on simulating random variables. One traditional area is the use of random numbers to sample from a population. More recent applications include simulation of high-dimensional, complex stochastic systems that are beyond analytical studies. In many practical situations the probability distributions are far too complicated to analyze and often it is easier to simulate these distributions on computers and the resulting samples can be analyzed instead. Given the faster computing and visualization environments of recent times more and more distributions are becoming accessible through simulations than ever before. In fact, the study of a random variable through simulations is becoming a powerful tool in the hands of the statisticians.

In view of this importance, we spend this chapter learning techniques to simulate from some well known distributions and summarize the emerging techniques in simulating from distributions which are continuous, discrete and perhaps a mixture of the two. It will be observed that in most cases the generation of random variables from any distribution involves first generating samples from a uniform density function. In fact, in most approaches there are two steps to simulating a given distribution:

1. Generate samples from a standard uniform probability, say between $[0, 1]$, and

2. Transform those samples appropriately to simulate the desired distribution.

We will study these two steps in the next sections.

5.2 Simulating Uniform Random Variable

How can one simulate a uniform random variable? One possibility is to tap a physical process (temperature, wind velocity etc.) and modify the measurements to simulate randomness. Even though the real life physical processes simulate randomness well, a drawback in this method is the lack of knowledge about the underlying probability. Since we are not sure of the **exact** probability model governing the physical process, it makes it hard for us to use it as a ranodm number generator. The other idea is to use algebraic methods to generate sequences of numbers that mimic the behavior of a uniform random variable. These numbers are called **pseudorandom** numbers and we will look at some common techniques for generating them.

Definition 12 *A uniform pseudorandom number generator is a mapping f that, starting from an initial value x_0, generates a sequence*

$$x_0,\ f(x_0),\ f(f(x_0)),\ f(f(f(x_0))), \dots\ .$$

Since f is computed on a computer (without the use of random number generator!!), it is a deterministic mapping. That is, given x_0 the remaining sequence is fixed everytime the sequence is computed. Interestingly, a deterministic sequence is being utilized to simulate a random variable. The elements of such a sequence should have the following properties:

1. The patterns between the numbers the appearing in a sequence should be minimized.
2. The correlation between the neighboring elements should be reasonably small.
3. The values should be distributed nearly uniformly over the whole the range of possible values.
4. The sequences should have large periods, where a period is later defined to be duration after which a sequence repeats itself.
5. There exist a set of goodness of fit tests for testing the probability distributions associated with the obsered random variables. The elements of a pseudorandom sequence should provide a reasonable performace in these goodness of fite tests.

Next, we study a few techniques to generate pesudorandom sequences on digital computers.

1. **Congruential Generators**:
   A common idea in pesudorandom methods utilizes modular arithmetic to formulate the congruential generators. First we introduce the notation for modular arithmatic. For a positive integer m and a given number b, a is called the residue of b modulus m if

   $$a = b - [b/m]m\ ,$$

   where $[\cdot]$ denotes the largest integer less than the argument. For example, if $m = 5$ and $b = 13$, then the residue of b modulus m is 3.0. We will use the notation that a is b mod m. It can be checked that a large number of numbers have the same residue. In fact, if we define two numbers to be equivalent if they have the same residue, then this relationship establishes an equivalence class on the space of numbers. We will retrict ourselves to non-negative integers, and the equivalence class for this set can be denoted by

   $$\{0, 1, 2, 3, \dots, m-1\}\ .$$

   Modular arithmatic can be used to generate sequences of pseudorandom numbers in the following way. Define the next element of the sequence is given by

   $$x_{i+1} = (ax_i + c)\mathrm{mod}m\ ,$$

   where a is called the multiplier, m is the base and c is the additive constant. The generators based on this general formula are called **linear congruential generators**. If $c = 0$, then the resulting generator is called a **multiplicative** generator. The choice of a and m determines the properties of the resulting sequence. We start with an initial number, $x_0 \in \{1, 2, \ldots, m-1\}$, called the seed of the number generator. The remaining sequence is determined by the relation

$$x_{i+1} = (ax_i)\text{mod}m \ .$$

It should be noted that that this relation is completely deterministic, i.e. if a number repeats in the sequence then the following sequence also repeats. Furthermore, since the elements of the sequence lie in the set $\{1, 2, \ldots, m-1\}$, the numbers repeat themselves if we generate a sequence long enough. The leads to the definition of a period.

Definition 13 *The period of a sequence is defined as the number of elements in the sequence before the sequence repeats itself.*

For the base m, the maximum period of a sequence is $m-1$. For the same m, but different multipliers a's, the resulting sequences can have very different periods. For example, for $m = 31$, and for $a = 12$, and for $a = 9$, we have .... therefore, the choice of a becomes important.

Also, for some choices of a we may have some unwanted structure in the sequence. Either the correlations between the successive numbers can be high, or there can be some non-apparent relationship between the numbers which makes the resulting sequence not appropriate for use in random number generators,

2. **Shift-Register Geneators**: This class of generators rely on high-speed implementation of a finite-memory device, that utilizes a weighted sum of the last n elements of the series to compute the next element. A general form can be described by the equation:

   $$x_{i+1} = (w_1 x_i + w_2 x_{i-1} + \ldots + w_n x_{i-n+1} \ .$$

   The advantage of this method is that it can be implemented rather fast in hardware.

3. **Lagged Fibonacci Generators**

   The general form of these generators is given by:

   $$x_i = x_{i-r} \cdot x_{i-s} \ ,$$

   where $\cdot$ is a binary operation defined on the set in which the x_i's take values.

We will denote a continuous uniform random variable taking values between 0 and 1 by $U[0,1]$.

5.3 Simulating Other Random Variables

We will start with a discussion on simulating discrete random variables followed by a discussion on sampling from the continuous distributions. In this chapter we will assume that we already have a uniform random number generator available to us. We will denote a uniform random variable taking values between 0 and 1 by $U[0,1]$. Essentially, this chapter will be concerned with transforming the numbers obtained from simulating $U[0,1]$ appropriately to obtain a desired random variable.

5.3.1 Discrete Random Variables

A discrete random variable takes only a countable number of values with pre-defined probabilities. A discrete random variable is characterized by its probability mass function defined as

$$\begin{aligned} P(x_1) &= p_1 \\ P(x_2) &= p_2 \\ &\dots \\ P(x_n) &= p_n \\ &\dots \end{aligned}$$

such that for all i, $0 \leq p_i \leq 1$, and $\sum_{i=1} p_i = 1$. Commonly used discrete random variables are binomial, Poisson, geometric and negative-binomial. As an example, the probability mass function of a Poisson random variable with parameter λ is given by:

$$p_i = \frac{\exp(-\lambda)\lambda^i}{i!}, \quad i = 0, 1, 2, \dots \ .$$

Similarly, for a binomial random variable, with parameters (n, p), the probability mass function is

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

Before we study various algorithms for simulating discrete random variables, we make the definition of simulation precise.

Definition 14 *For a given random variable, with a specified probability mass function $\{(x_i, p_i),\ i = 0, 1, 2, \dots\}$, the process of selecting a value x_i with probability p_i is called Simulation. If this selection is performed many times, generating a sequence $\{X_j\}$, then*

$$\frac{1}{n} \sum_{j=1}^{n} I_{X_j}(\{x_i\}) \to p_i \ . \tag{5.1}$$

In other words, one selects from the set of the allowed values with the associated probability. For discrete random variables, there are at least three standard simulation techniques.

1. **Inverse Transform Method**:
   Let X be a discrete random variable with a given probability mass function. We have to utilize numbers generated using $U[0,1]$ to simulate values of X. This can be accomplished using the following algorithm:

   (a) Generate U according to $U[0,1]$.

   (b)
```
If $U < p_1$
            Set $x = x_1$
else if $U < p_2 + p_1$
            Set $x = x_2$
...
else if $U < \sum_{i=1}^{n} p_i$
            Set $x = x_n$
...
```

As shown in Figure 5.1, the random variable U will take value between 0 and 1 on the vertical axis. That value is projected to the right till it meets with a vertical bar and the corresponding value is selected. It can be shown that the value x_j is selected with probability p_j. To verify that X generated according to these steps has the desired probability

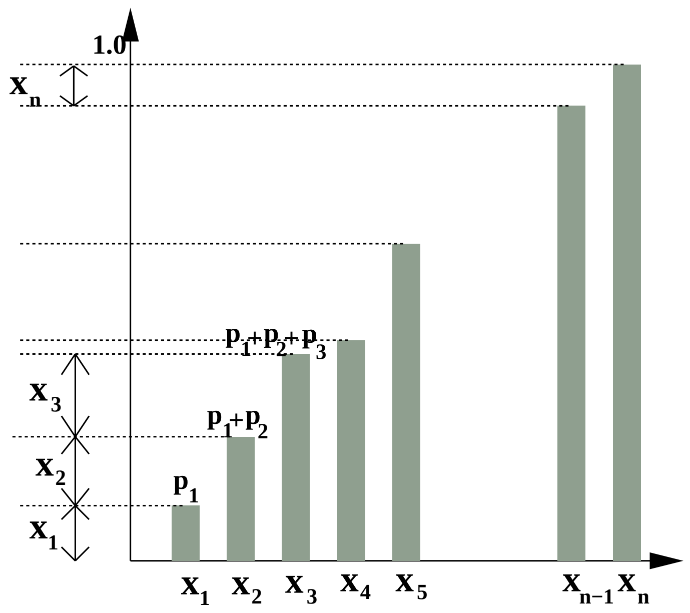


Figure 5.1: Random sampling from a discrete probability distribution.

distribution:

$$\begin{aligned} P(X = x_j) &= P(\sum_{i=1}^{j-1} P(x_i) \leq U \leq \sum_{i=1}^{j} P(x_i)) = p_j \\ &= \sum_{i=1}^{j} P(x_i) - \sum_{i=1}^{j-1} P(x_i) = P(x_j) \end{aligned}$$

This is a general technique for sampling discrete random variables. We take some examples.

(a) **Geometric random variable**

A random variable X is called geometric if its probability density function is given by

$$P\{X = i\} = p(1-p)^{i-1}, \quad i = 1, 2, \dots \ .$$

Consider an experiment involving independent and identical trials, each resulting in either success with probability p or failure with probability $(1-p)$. X denotes the number of trials performed to reach the first success.

$$P\{X \leq j-1\} = \sum_{i=1}^{j-1} p(1-p)^{i-1} = 1 - (1-p)^{j-1} \ .$$

Let U be distributed according to $U[0, 1]$. The algorithm sets $X = j$ if

$$\sum_{i=1}^{j-1} P\{X = i\} \quad \leq \quad U < \sum_{i=1}^{j} P\{X = i\}$$

$$\begin{aligned} 1-(1-p)^{j-1} &\leq U < 1-(1-p)^j \\ (1-p)^{j-1} &\geq 1-U > (1-p)^j \end{aligned}$$

In other words, set $X = j$ if

$$\begin{aligned} X &= \min\{j : (1-p)^j < 1-U\} \\ &= \min\{j : j\log(1-p) < \log(1-U)\} \\ &= \min\{j : j > \frac{\log(1-U)}{\log(1-p)}\} \end{aligned}$$

Since U is the same random variable as $1-U$,

$$X = \text{ceil}(\frac{\log(U)}{\log(1-p)}) \ .$$

(b) **Poisson random variable**
X is a Poisson random variable with parameter λ if its probability mass function is given by

$$P\{X = i\} = \frac{\exp(-\lambda)\lambda^i}{i!} \ , \ i = 0, 1, 2, \ldots$$

Therefore,

$$\frac{P\{X = i+1\}}{P\{X = i\}} = \frac{\lambda}{i+1} \ .$$

Using this results, the cumulative distribution function can be iteratively calculated as

$$P\{X \leq j\} = \sum_{i=0}^{j} \frac{\lambda}{i+1} \ .$$

An algorithm to simulate X using inverse transform method is given by:

i. Generate U according to $U[0,1]$
ii. Set $i = 0$, $p = \exp(-\lambda)$, and $F = p$.
iii. If $U < F$, set $X = i$ and stop.
iv. Set $p = \frac{\lambda p}{i+1}$, $F = F + p$, $i = i + 1$
v. Go to Step (iii).

(c) **Binomial random variable**
X is said to be binomial with parameters (n, p) if its probability mass function is given by

$$P\{X = i\} = \binom{n}{i} p^i (1-p)^{n-i} \ .$$

As in the Poisson case,

$$\frac{P\{X = i+1\}}{P\{X = i\}} = \frac{n-i}{i+1}\frac{p}{1-p} \ .$$

An algorithm to simulate X using inverse transform method is given by:

i. Generate U according to $U[0,1]$

ii. Set $c = \frac{p}{1-p}$, $i = 0$, $pr = (1-p)^n$, and $F = pr$.
iii. If $U < F$, set $X = i$ and stop.
iv. Set $pr = c\frac{n-1}{i+1}pr$, $F = F + pr$, $i = i + 1$
v. Go to Step (iii).

Sometimes the random variable has additional structure that can be utilized to sample in a more direct fashion. As an example consider the poisson random variable.

(d) **Poisson Random Variable**: This is a discrete random variable which takes values in the set of non-negative integers with the probability given by

$$P(z = k) = \frac{\exp(-\lambda)\lambda^k}{k!} \ ,$$

where λ is called the rate of z.

Let $u_i, i = 1, 2, \ldots$ be a sequence of i.i.d. random variables each uniform over the interval $[0, 1]$. Let z be a random variable generated by the following procedure:

i. set $i = 0, T = 1, z = -1$.
ii. $i = i + 1$, $z = z + 1$.
iii. generate u_i.
iv. $T = Tu_i$.
v. if $T > e^{-\lambda}$ go to step (b), else stop.

It can be shown that the random variable z produced with this procedure is Poisson with parameter λ.

2. **Acceptance/Rejection Method**:
This method is useful in situations where we want to simulate a random variable Y and we have a technique of simulating another random variable X. Let the probability density functions of X and Y be given by

$$P\{X = i\} = p_i, \ i = 0, 1, 2, \ldots, \quad P\{Y = j\} = q_j, \ j = 0, 1, 2, \ldots \ .$$

We will assume that there exists a constant c such that

$$\frac{p_j}{q_j} \leq c, \quad \text{for all} \ \ j \ \ \text{for which } p_j > 0 \ .$$

The general algorithm for accetancerejection is given by:

(a) Simulate a value of Y.

(b) Generate U according to $U[0, 1]$.

(c) If $U < \frac{p_Y}{q_Y}$, then set $X = Y$,
Else, return to (i)

As a first step, we have to prove that this procedure simulates X. That is, for an X generated according to this procedure $P\{X = j\} = p_j$.

3. **Composition Method**:
   Consider n discrete random variables $X_1, X_2, \ldots, X_i, \ldots, X_n$, with probability mass functions $P_j^{(i)}$, such that $\sum_j P_j^{(i)} = 1$, for all i. Let a random variable X has a probability distribution given by:

   $$Pr\{X = j\} = \sum_{i=1}^{n} \alpha_i P_j^{(i)} \ ,$$

   where $0 \leq \alpha_i \leq 1$ and $\sum_{i=1}^{n} \alpha_i = 1$.

   Given algorithms for simulating each of the random variables $X_1, X_2, \ldots, X_n$, how can we simulate the composite variable X. Since $\{\alpha_i\}$ forms a probability mass function on $\{1, 2, \ldots, n\}$, one can simulate a random value from this set according to α, call it Y. Then, generate a value of X according to the probability mass function $P^{(Y)}$.

   Simulation of a composite random variable is simplified because one can perform this process sequentially. At first, one of the individual random variable is selected according to the weights (α) and then, conditional probability of X is identical to that of the selected random variable, making it simple to simulate.

5.3.2 Continuous Random Variables

Continuous random variables can take uncountable number of values according a pre-defined probability density function. There are two commonly used techniques to simulate a given continuous random variable:

1. **Inverse Transform Method**: The usefulness of this method comes from the following result:

   Proposition 3 *Let U be uniform in $[0, 1]$. If for any random variable with continuous distribution function F we define a random variable Y according to*

   $$Y = F^{-1}(U) \ ,$$

   then the random variable Y has distribution function F.

   Proof:

   The probability distribution function of Y is given by

   $$F_Y(a) = P\{Y \leq a\} = P\{F^{-1}(U) \leq a\} \ .$$

   Since F is a monotonically increasing function

   $$F^{-1}(U) \leq a \Leftrightarrow U \leq F(a)$$

   and, therefore, $F_Y(a) = P\{U \leq F(a)\} = F(a)$ since U is uniform in $[0, 1]$.

   QED

We can utilize the inverse transform method to generate a variety of random variables. We start with the example of the **exponential random variable** with mean one. Its density and distribution functions are given by

$$f(x) = \exp(-x),\ x \geq 0, \quad F(x) = 1 - \exp(-x),\ x \geq 0\ .$$

To find $F^{-1}(U)$, let $U = F(x)$ and solving for x provides

$$x = F^{-1}(U) = -\log(1-U)\ .$$

From Proposition 3, if U is uniform in $[0,1]$, then this x is exponentially distributed with mean one. It should be noted that if U is uniform $[0,1]$ then $1-U$ is also a uniform $[0,1]$ random variable. Therefore, to reduce computations $1-U$ can be replaced by U with no change in the results. Hence, $x = -\log(U)$ is an exponential random variable with mean one. To generate an **exponential random variable** with mean λ utilize $x = -\lambda \log(U)$.

A **Gamma random variable**, which is the sum of n independent exponential random variables each with rate λ, can be generated as follows: let $U_1,\ U_2, \ldots,\ U_n$ be independent realizations of uniform $[0,1]$, the define

$$x = -\sum_{i=1}^{n} \frac{1}{\lambda} \log(U_i) = \frac{-1}{\lambda} \log(\prod_{i=1}^{n} U_i)\ .$$

x is a realization of from the gamma density function.

2. **Acceptance/Rejection Method**: In this case we assume that we have a method for simulating from some density function g and our task is utilize samples from g to simulate from a given density function f. g can be fairly arbitrary except for one condition mentioned below. The basic idea is to simulate from g and accept the samples with probability proportional to the ratio f/g. Let C be a constant such that

$$\frac{f(Y)}{g(Y)} \leq C, \quad \text{for all } Y\ .$$

Then, the simulation procedure is:

(a) Simulate Y from the density g and simulate U from uniform $[0,1]$.

(b) If $U \leq \frac{f(Y)}{Cg(Y)}$ then $X = Y$ else go to step 1.

Proposition 4 *X is random variable with density f.*

Proof:

Let X be the value obtained and n be the number of iterations required to reach this value. Then,

$$\begin{aligned} P(X \leq x) &= P(Y_n \leq x) \\ &= P(Y \leq x | U \leq \frac{f(Y_n)}{Cg(Y_n)}) \\ &= \frac{P(Y \leq x, U \leq \frac{f(Y_n)}{Cg(Y_n)})}{K}\ , \end{aligned}$$

where $K = P(U \leq \frac{f(Y_n)}{Cg(Y_n)})$. Since Y and U are independent random variables, their joint density function is the product of the marginals

$$g(y) \times 1 \ ,$$

since U is uniform in $[0,1]$. Therefore,

$$\begin{aligned}
P(X \leq x) &= \frac{1}{K} \int_{\frac{U \leq f(Y)}{Cg(Y)}} \int_{Y \leq x} g(Y) dY dU \\
&= \frac{1}{K} \int_{-\infty}^{x} (\int_{0}^{\frac{f(Y)}{Cg(Y)}} dU) g(Y) dY \\
&= \frac{1}{K} \int_{-\infty}^{x} \frac{f(Y)}{Cg(Y)} g(Y) dY \\
&= \frac{1}{CK} \int_{-\infty}^{x} f(Y) dY
\end{aligned}$$

For $x \to \infty$, the left side goes to 1 and the integral on the right side also goes to 1. Therefore, $CK = 1$ and

$$P(X \leq x) = \int_{-\infty}^{x} f(Y) dY \ .$$

Therefore, X is random with probability density f.

QED

For a given value of Y we accept Y by generating a uniform U and comparing U with $\frac{f(Y)}{Cg(Y)}$. This is also called accepting Y with probability $\frac{f(Y)}{Cg(Y)}$. Each iteration in the loop involves independent realizations, we can compute the probability of accepting Y as X according to

$$P(U \leq \frac{f(Y)}{Cg(Y)}) = K = \frac{1}{C} \ .$$

If C is large then the process, of generating samples from f using this method, will be slow.

Next we illustrate the use of acceptance/rejection method by generating sample from **standard normal** density function. As a first step we will simulate from the density function given by

$$f(x) = \frac{2}{\sqrt{2\pi}} \exp(\frac{-x^2}{2}), \ x \geq 0 \ .$$

Notice, this is the density function associated with $x = |z|$, where z is the standard normal random variable. Also, we will assume that we have tools to sample from the standard exponential density function which becomes our g for the above discussion ($g(x) = \exp(-x)$). To bound the ratio of f to g with a constant,

$$\begin{aligned}
\frac{f(x)}{g(x)} &= \sqrt{2\pi} \exp(-\frac{(x^2 - 2x)}{2}) \\
&= \sqrt{\frac{2e}{\pi}} \exp(-\frac{(x-1)^2}{2}) \\
&\leq \sqrt{\frac{2e}{\pi}} = C
\end{aligned}$$

and

$$\frac{f(x)}{Cg(x)} = \exp(-\frac{(x-1)^2}{2}) \ .$$

To generate a random variable with density f the following algorithm is used:

1. Generate Y, an exponential random variable with mean 1, and U, a uniform $[0,1]$ random variable.
2. If $U \leq \exp(\frac{-(Y-1)^2}{2})$ set $X = Y$, otherwise return to (a).

Now,

$$\begin{aligned} U &\leq \exp(-\frac{(Y-1)^2}{2}) \\ -\log(U) &\geq \frac{(Y-1)^2}{2} \end{aligned}$$

Since $-\log(U)$ is exponential with rate 1 we can re-formulate as follows:

1. Generate Y_1 and Y_2, two samples from exponential random variable with mean 1.
2. If $Y_2 \geq \frac{(Y_1-1)^2}{2}$ set $X = Y_1$, otherwise return to (a).

Having generated a random variable which is the absolute value of a standard normal, we can generate sample from standard normal according to the following algorithm.

1. Generate U a uniform random variable between $[0,1]$ and generate X according to the algorithm described above.
2. If $U \in (0, \frac{1}{2}]$ set $Z = X$, else set $Z = -X$.

Polar Method: There is another method used popularly to generate samples from standard normal density. It is also called Box-Muller method. The principle behind this method is the following:

if X and Y are independent and standard normal random variables then for

$$\theta = tan^{-1}(\frac{Y}{X}), \quad R = \sqrt{(X^2+Y^2)}$$

θ is uniform in $[0, 2\pi]$ and R^2 is exponential with mean 2.

To reverse this result, if U_1 and U_2 are uniform in $[0,1]$ then for

$$R = (-2\log(U_1))^{1/2}, \quad \theta = 2\pi U_2 \ ,$$

and

$$X = R\cos(\theta), \quad Y = R\sin(\theta)$$

X and Y are independent samples from standard normal density. It has a drawback that it involves computing trigonometric functions which is always computationally expensive. This method can be modified in the following way to avoid computing sines and cosines.

If U_1 and U_2 are uniform in $[0,1]$ then

$$V_1 = 2U_1 - 1, \quad V_2 = 2U_2 - 1 \ ,$$

are uniform in $[-1, 1]$. They may or may not lie in the circle of radius 1 and centered at $(0, 0)$. Generate the pair (V_1, V_2) until it lies in this circle and let $(\bar{R}, \bar{\theta})$ be the polar coordinates of this pair. It can be shown that $\bar{R}^2$ is uniform in $[0, 1]$ and $\bar{\theta}$ is uniform in $[0, 2\pi]$. For these quantities

$$sin(\bar{\theta}) = \frac{V_2}{\bar{R}}, \quad cos(\bar{\theta}) = \frac{V_1}{\bar{R}} \ .$$

Therefore, if U is a uniform $[0, 1]$ random variable independent of $\bar{\theta}$, then

$$\begin{aligned} X &= (-2\log(U))^{1/2}\frac{V_1}{\bar{R}} \\ Y &= (-2\log(U))^{1/2}\frac{V_2}{\bar{R}} \end{aligned}$$

are independent sample from standard normal density. Since $\bar{R}^2$ is uniform in $[0, 1]$ and independent of $\bar{\theta}$ it can be used in place of U. Hence X and Y can be generated according to the equations,

$$\begin{aligned} X &= (-2\log(\bar{R}^2))^{1/2}\frac{V_1}{\bar{R}} \\ Y &= (-2\log(\bar{R}^2))^{1/2}\frac{V_2}{\bar{R}} \end{aligned}$$

An algorithm to implement these equations to generate independent samples from standard normal is:

1. Generate independent, uniform $[0, 1]$ random variables.
2. Set $V_1 = 2U_1 - 1$, $V_2 = 2U_2 - 1$ and $S = V_1^2 + V_2^2$.
3. If $S > 1$, return to (a).
   Else $T = \sqrt{\frac{-2\log(S)}{S}}$ and $X = TV_1$ and $Y = TV_2$.

To generate Y, a **normal random variable** with mean μ and standard deviation σ, generate X a standard normal random variable and set

$$Y = \sigma X + \mu \ .$$

5.4 Simulating Stochastic Processes

We can extend the ideas associated with simulating random variables to simulate stochastic processes. To start with, we briefly introduce the notion of stochastic process. So far we have dealt only with real-valued, i.e. scalar, random variables but now we extend to a collection of random variables.

Definition 15 *A stochastic process is an indexed collection of random variables*

Most often, the indexing variables is time. For example let X be a stochastic process, indexed by time, such that X_t is a real-valued random variable. For a fixed time t, X_t is just a random variable. Each realization, or sample, of X_t is now a function of time, also called a *sample path.* To simulate a stochastic process is same as to generate a sample path. In general, the indexing can be according to other variables, such as spatial location, frequency, wavelength, etc. At least in the start, we will restrict ourselves to time-indexed, scalar-valued stochastic processes.

In case the indexing variable t is continuous, the stochastic process called a *continuous-time* process; else, it is called a *discrete-time* process. Similarly, if the space in which X_t takes values is continuous, then the process is called a *continuous-valued* process. Otherwise, it is called a discrete-valued process. The ideas preserned in Chapter 1 should convince us that one can implement only discrete-time, discrete-valued processes on a digital computer. However, a study of continuous processes is important in many regards, including the development of properties of discrete processes. There are two main kinds of continuous stochastic processes: (i) ones with smooth (e.g. continuous) samples paths, and (ii) ones with discontinuities, or jumps, in their sample paths. In this section, we will study the two simplest, and most famous, examples of each category.

5.4.1 Random Walk and Wiener Process

A random walk is one of the simplest examples of a discrete time, discrete state stochastic process. The process transitions only at times separated by a unit time T, and takes values only in the set of integer multiples of s. At every transition time nT, for $n = 1, 2, \ldots,$, toss a coin and depending on the outcome, add $+s$ for a head or $-s$ for a tail to the current value of the process. It is rather easy to simulate this process, and next we look at a way of analyzing it.

Let us assume that we start with process at time zero with the process value being zero. At a transition time nT, the value of of the process is $X(nT) = ks - (n-k)s$ where the number of heads in n tosses is k. In other words,

$$X(nT) = ms, \quad \text{where} \quad m = (2k - n) \quad \text{or} \quad k = \frac{m+n}{2} \ .$$

Since $k \in \{0, 1, 2, \ldots, n\}$, the possible values for m are $\{-n, -n+1, \ldots, 0, \ldots, n\}$. The probability that $X(nT) = ms$ is given by

$$Pr\{X(nT) = ms\} = \left(\begin{array}{c} n \\ \frac{m+n}{2} \end{array} \right) (\frac{1}{2})^n \ ,$$

assuming a fair coin. Given this probability, one can analyze this random walk for any time nT. For example, one can easily compute the mean of $X(nT)$, or the variance of $X(nT)$, and so on.

Another interpretation of $X(nT)$ is given by the following:

$$X(nT) = X_1 + X_2 + \ldots + X_n \ , \tag{5.2}$$

where X_is are i.i.d random variables. Each X_i takes on the value either $+s$ or $-s$ with equal probabilities; it is easy to show that $E[X_i] = 0$ and $\text{var}(X_i) = s^2$. $X(nT)$ is a sum of independent and identically distributed random variables with mean zero and variance s^2. Therefore, $X(nT)$ is a random variable with mean zero and variance ns^2.

An interesting case results when we look at the following limiting situation. Let the transition times get closer to each other by setting $T \to 0$, and the transition sizes get smaller by setting $s \to 0$. We require that the transition size gets smaller at a rate slower than the rate of decrease in step size; this is done by setting $s = \alpha\sqrt{T}$ for some position number α. In the limit, the random walk becomes a continuous time, continuous state process with the following properties. Let $t = nT$ denote the time index of the process; t is a continuous random variable in the limiting case. Consider the random process at time t. $X(t)$, through its relation with $X(nT)$, has the following statistics:

$$\begin{aligned} E[X(t)] &= E[X(nT)] = 0 \\ \text{var}(X(t) &= \text{var}(X(nT)) = ns^2 = \frac{t}{T}\alpha^2 T = \alpha^2 t \ . \end{aligned}$$

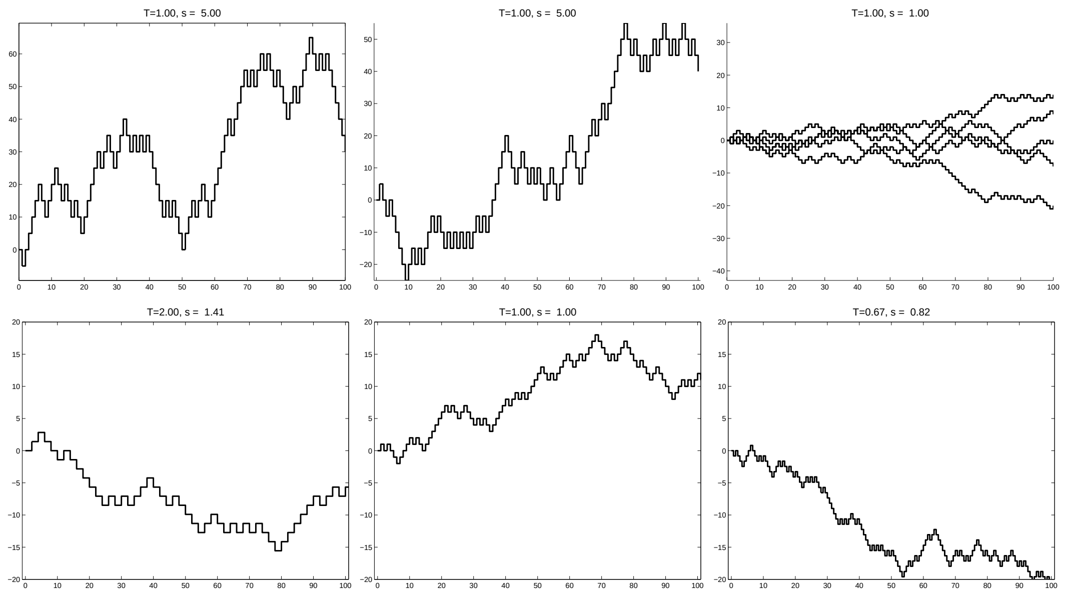


Figure 5.2: Simulations of Markov random walk for different parameters. The lower panels, plotted for $\alpha = 1$, show that as T gets smaller, the sample path looks increasingly continuous.

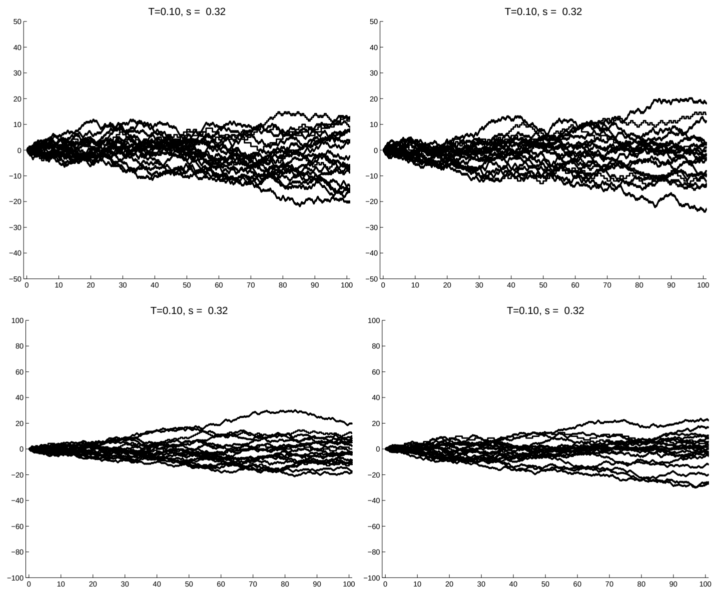


Figure 5.3: Pictorial illustration of increasing variances of X_t as t increases.

That is, $X(t)$ is a continuous random variable with mean zero and variance αt. What is the nature of this random variable. Applying central limit theorem in Eqn. 5.2, we can approximate $X(t)$ to be a Gaussian random variable for a large n which is the case in the limiting situation. Therefore, $X(t) \sim N(0, \alpha^2 t)$. This limiting process is called the *Wiener process*. For $\alpha = 1$, it is called the *standard Wiener process*. Some properties of a Wiener process can be established directly from its definition:

1. If (t_1, t_2) and (t_3, t_4) are non-overlapping intervals, then $X(t_2) - X(t_1)$ and $X(t_4) - X(t_3)$ are statistically independent random variables.

2. The increment $X(t_2) - X(t_1)$ is independent of the value $X(t_1)$ or any past value of the process. In fact, this increment is a Gaussian random variable with mean zero and variance $\alpha^2(t_2 - t_1)$.

3. For any two times $t_1 < t_2$, we have:

$$\begin{aligned} E[X(t_1)X(t_2)] &= E[X(t_1)(X(t_2) - X(t_1) + X(t_1))] \\ &= E(X(t_1)^2] + E[X(t_1)(X(t_2) - X(t_1))] \\ &= \alpha^2 t_1 + E[X(t_1)]E[X(t_2)] \\ &= \alpha^2(t_1) \ . \end{aligned}$$

It is interesting to note that the variance of a Wiener process grows linearly with time, although variance is a second order statistic. Shown in Figure 5.3 are some sample paths of a standard Wiener process, and a linear increase in the variance of $X(t)$ is apparent from these pictures.

There is a larger family of sample-path continuous stochastic processes whose simplest example is a Wiener process. These are called *diffusion processes*. A general diffusion process is given by:

$$dX(t) = \mu(X(t), t)dt + \sigma(X(t), t)dW(t) \ , \tag{5.3}$$

where μ and σ are called the infinitesimal means and variances, respectively, of the diffusion process. With appropriate smoothness conditions on μ and σ, one show that the solution $X(t)$, of this equation, is a sample-path continuous, Markov process with probability one. There are several interesting properties associated with diffusion processes, and they play an important role in statistical inferences. In particular, diffusion processes can used to generate samples from complicated probability distributions within the framework of Markov chain Monte Carlo methods, a topic studied later in Chapter 7.

5.4.2 One-Dimensional Poisson Process

While a Wiener process is a classical example of a sample path continuous process, a Poisson process represents the discrete-state processes. We start with a simple homogeneous Poisson process and later extend to heterogeneous processes.

Assume that we are interested in the occurrences of an event, and that event occurs at random times. This event, for example, can be an automobile accident at a particular crossing, or ringing of a particular telephone. We are interested in recording and modeling the times of occurrences, also called the *arrival times*. The time lapses between the arrival times are called *inter-arrival times*. Denote the arrival times by t_is and the inter-arrival times by τ_is. They are related according to:

$$t_i = t_{i-1} + \tau_{i-1} \ . \tag{5.4}$$

In case of a homogeneous Poisson process with intensity λ, these τ_is are assumed to be independent and identically distributed according to the exponential density with intensity λ, i.e. $\tau_i \sim \lambda \exp(-\lambda\tau)$. The resulting Poisson counting process can defined as follows:

$$N(t) = \sum_{i=1}^{\infty} 1_{[0,t)}(t_i) \ , \ N(0) = 0 \ .$$

$N(t)$ counts the number of arrivals, or the occurrences, till time t. It can be shown that on any interval (s_1, s_2), the number of arrival times, i.e. $N(s_2) - N(s_1)$, is given by a Poisson random variable with mean $\lambda(s_2 - s_1)$. That is:

$$Pr\{N(s_2) - N(s_1) = k\} = \frac{\exp(-\lambda)\lambda^k}{k} \ .$$

Some additional properties that can be established from the earlier definition are as follows.

1. If (s_1, s_2) and (s_3, s_4) are non-overlapping intervals, then $N(s_2) - N(s_1)$ and $N(s_4) - N(s_3)$ are statistically independent random variables.

2. The incremental count, given by the random variable $N(s_2) - N(s_1)$, is independent of the process value at s_1 or any other previous time.

3. Assuming $N(0) = 0$, the expected value of $N(t)$ is λt and its variance is also λt.

4. For any two times $s_1 < s_2$, we have:

$$\begin{aligned} E[N(s_1)N(s_2)] &= E[N(s_1)(N(s_2) - N(s_1) + N(s_1))] \\ &= E(N(s_1)^2] + E[N(s_1)(N(s_2) - N(s_1))] \\ &= \lambda s_1 + \lambda^2 s_1^2 + \lambda s_1 \lambda(s_2 - s_1) \\ &= \lambda s_1(1 + \lambda s_2) \end{aligned}$$

The simulation of a homogeneous Poisson process is straightforward. Generate τ_is from the exponential density, and use Eqn. 5.4 to compute the arrival times. Shown in Figure 5.4 are two examples of a homogeneous Poisson process with $\lambda = 1$. For each sample, the left figure shows a histogram of values of inter-arrival times τ_is, and the right figure shows the evolution of $N(t)$ versus t. Also shown on the time axis are the arrival times t_is.

A heterogenous Poisson process can be studied similarly. In this case, the intensity λ is not fixed but a given function of time $\lambda(t)$. Similar to the homogenous case, the number of arrival times in any interval (s_1, s_2) is a Poisson random variable with intensity given by $\int_{s_1}^{s_2} \lambda(t)dt$. Also, the inter-arrival times are exponentially distributed as earlier, except now the intensities of these exponential variables are given by $\int \lambda(t)dt$. The properties of independent increments and independent of arrival counts in disjoint intervals still hold. The next question is: How to simulate a heterogenous Poisson process with a given intensity function? There are methods to simulate such processes:

1. **Pruning**: For a fixed time interval $[0, T]$, find the maximum value of λ in that interval as

$$\lambda_{max} = \max_{t \in [0,T]} \lambda(t),$$

and simulate a homogeneous one-dimensional processes with intensity λ_{max}. Let $\{t_i\}$ be the collection of arrival times generate by this simulation. For each i, perform the

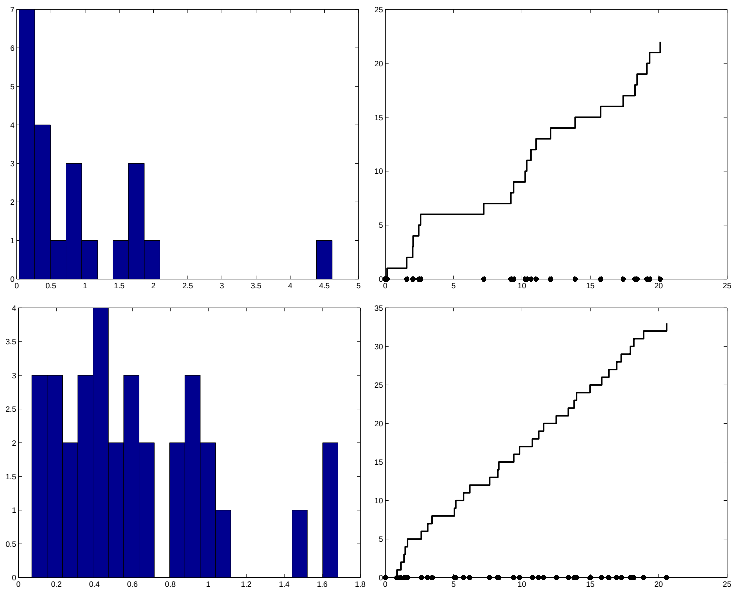


Figure 5.4: Sample paths illustrating a homogeneous Poisson process with $\lambda = 1$. Left panels show histograms of τ_is and the right panels show markers for t_is and plot $N(t)$ versus t.

following test: Keep t_i as an arrival time with probability $\lambda(t_i)/\lambda_{max}$, and reject it with probability $1 - \lambda(t_i)/\lambda_{max}$. The set of accepted arrival times forms a realization of the desired heterogeneous Poisson process.

2. **Directly Simulating Arrival Times**: Another possibility is to simulate the arrival times t_is directly using the given intensity function $\lambda(t)$. The basic idea is to accumulate intensity function over an interval till it exceeds an exponential random variable with mean one.

5.4.3 Spatial Point Processes

Similar to one-dimensional processes, one can study occurrences of random events in higher spaces using appropriate Poisson processes. In this section we consider a Poisson process in $\mathbb{R}^2$, also called a spatial point process. We start the discussion with a homogeneous point process with a constant intensity function λ. A point process consisting of randomly occurring points in the plane is said to be a two-dimensional Poisson process having rate λ, if

1. the number of points in any given region of area A is Poisson distributed with mean λA; and

2. the number of points in disjoint regions are independent.

Let O be the origin in $I\!R^2$ and R_i be the i-th nearest Poisson point to O, $i \geq 1$ ($R_0 = O$). Then, it can be shown that $(\pi R_i^2 - \pi R_{i-1}^2)$ are exponentially distributed with rate λ. By symmetry, the respective angles of the Poisson points are independent and uniform $[0, 2\pi]$. Then, the following algorithm simulates a two-dimensional Poisson process in a ball of radius r centered at O, $C(r)$.

Algorithm 32 *1. Generate independent exponentials $X_1, X_2, \ldots$, with rate 1, stopping at*

$$N = min\{n : \frac{X_1 + X_2 + \ldots X_n}{\lambda\pi} > r^2\}$$

2. *if $N = 1$, stop, there are no points in $C(r)$. Otherwise, for $i = 1, 2, \ldots, N-1$, set*

$$R_i = \sqrt{(X_1 + X_2 + \ldots + X_i)/\lambda\pi}$$

3. *Generate independent uniform $[0, 1]$ random variables $U_1, U_2, \ldots, U_{N-1}$.*

4. *Return the $N - 1$ Poisson points in $C(r)$ whose polar coordinates are $(R_i, 2\pi U_i), i = 1, \ldots, N-1$.*

5.5 Problems

1. Write a matlab program to implement a uniform random number generator using a multiplicative congruential method with $m = 2^{13} - 1$ and $a = 17$. Generate 500 numbers for the starting point $x_0 = 100$. Define the sequence $u_i = \frac{x_i}{m}$ and plot a histogram to display the results. Also, calculate the correlation of the pairs of successive number u_i and u_{i+1}.

2. Repeat Problem 2 with $a = 85$.

3. Write a matlab program to find the period of a random number generator for a given seed. Find the period of the sequence generated by a multiplicative congruential generator for $m = 31$, $a = 7$, and $x_0 = 19$. Find the period if a is changed to 3.

4. Form a combination sequence using two multiplicative congruential generators. Generate the first sequence of 30 numbers for $m = 31$, $a = 3$, and $x_0 = 9$. Generate the second sequence of 30 numbers using $a = 12$. Rearrange the order of the first sequence according to the numbers in the second sequence. Find the correlation of successive pairs in the first sequence before and after the rearrangement.

5. Simulate a two-dimensional Poisson process with $\lambda = 0.5$ on $C(4)$.

6. Suppose it is relatively easy to simulate from the distributions $F_i, i = 1, 2, \dots, n$. If n is small, how can we simulate from

$$F(x) = \sum_{i=1}^{n} P_i F_i(x),\ P_i \geq 0, \sum_i P_i = 1 \ .$$

   Give a method to simulate from the distribution

$$F(x) = \begin{cases} \frac{1-e^{-2x}+2x}{3} & 0 < x \leq 1 \\ \frac{3-e^{-2x}}{3} & 1 < x < \infty \end{cases} .$$

7. Suppose a function g is bounded in $[0, 1]$, there are at least two simulation techniques to approximate $\int_0^1 g(x)dx$: (i) Hit-Miss method covered in an earlier home-assignment, and (ii) generating independent uniform $[0, 1]$ realizations U_i and calculating $\frac{1}{N}\sum_i^N g(U_i)$. Show that though the two techniques converge to the same mean the variance in the second case is smaller.

8. **Binomial**: To generate a sample from a binomial random variable (n, p), generate n samples from a uniform random variable with values between 0 and 1. Count how many of them are less than p. This number is a sample from binomial (n, p). Generate 100 samples from binomial $(10, 0.6)$ and histogram the samples.

9. **Gaussian**: Generate N samples of a Gaussian random variable with mean 5 and standard deviation 2 using the polar method. Histogram these samples for $N = 10,\ 100,\ 1000$.

10. For poisson random variable, choose $\lambda = 1$. Generate N realizations of z and plot its histogram for $N = 100$ and $N = 500$.

11. **Monte-Carlo Integration**: Let $f(x) = e^x$. We will analyze a technique to approximate the integral $\int_0^1 f(x)dx$ numerically using Monte-Carlo technique.

(a) Let $S = 0$ and $i = 1$.

(b) Generate u_1 and u_2, two samples of a uniform random variable between 0 and 1.

(c) If $u_2 \leq f(u_1)$, then $S = S + 1$.

(d) $i = i + 1$. If $i < N$, then go to Step 2.

S/N is the approximate value of the integral. Plot this value as a function of N up to $N = 50000$.

12. Previously we have utilized the inverse transform method to sample from exponential densities. Utilizing sampling from exponential densities how can we sample from the following density:

$$f(x) = 0.2\exp(-x) + 0.4\exp(-2x) + 0.4\exp(-4x) \ .$$

f is a weighted sum of exponential densities. Following is an algorithm to generate samples from f:

Algorithm 33 *(a) Generate U, uniform random variable in* $[0, 1]$.

```
(b) If $U \leq 0.2$ then $\lambda = 1$
    else if $U \leq 0.6$ then $\lambda = 2$
    else $\lambda = 4$.
(c) Generate a sample from exponential density with parameter $\lambda$.
```

Generate $N = 1000$ samples from f and plot a histogram of the samples.

13. **Computing expectations via sampling**: For a given density function f, and certain smooth function g

$$\int_{\mathbb{R}} g(x)f(x)dx \ ,$$

is called the expectation of g with respect to f. A numerical technique to approximate this quantity is to generate samples from f, call them $x_1,\ x_2,\ \ldots, x_N$ and compute the sample mean,

$$\frac{1}{N}\sum_{i=1}^{N} g(x_i) \ .$$

For large values of N this sample mean approximates the expectation of g. For f given by Gaussian density function with mean 5 and variance 2, compute the following expectation via sampling for N=100,1000,10000:

$$\int_{\mathbb{R}} (x-5)^2 f(x)dx \ .$$

14. Estimate the value of π via random sampling using the the two variance reduction techniques simultaneously: (i) variance reduction by conditioning, and (ii) variance reduction by using antithetic variables. Use $N = 50000$.

15. If $\{p_i,\ i = 1, \ldots, n\}$ is the probability mass function of a discrete random variable and $v = [v_1,\ v_2, \ldots, v_n]$ is any vector, then show that

$$(\sum_{i=1}^{n} p_i v_i)^2 \leq \sum_{i=1}^{n} p_i v_i^2 \ .$$

16. Write a matlab program to sample k values from the probability mass function $p_i = \frac{i}{55}$, $i = 1, 2, \ldots, 10$. Plot the histogram of generated values for $k = 50,\ 500$, and 5000.

17. Write a matlab program to simulate k values of a binomial random variable with parameters $n = 10$ and $p = 0.15$. Plot the means and variances of these k values as functions of k and compare the convergence to np and $np(1-p)$.

18. Let p_j and q_j be the probability mass functions associated with two binomial random variables, with parameters $(20, 0.1)$ and $(20, 0.4)$, respectively. Write a matlab program to simulate a random variable X specified by the probability mass function

$$P\{X = j\} = 0.1p_j + 0.9q_j,\ j = 0, 1, \ldots, n \ .$$

Simulate 1000 values of X and plot a histogram.

19. Write a matlab program to simulate k values of a normal random variable with mean μ and variance σ^2. Use this program to generate values for $\mu = 5$ and $\sigma^2 = 4$. Plot a histogram of the samples generated for: (i) $k = 500$, and (ii) $k = 5000$.

20. Write a matlab program to simulate a random variable X with the density function

$$f(x) \begin{cases} 4x & 0 \le x < 0.5 \\ 4(1-x) & 0.5 \le x < 1 \\ 0 & \text{otherwise} \end{cases} .$$

First calculate the cumulative distribution function and then use the inverse transform method for simulating X. Generate 100 values from this simulator and plot a histogram.

21. Write a matlab program to simulate a Poisson random variable with mean 5. Generate 100 values and plot a histogram.

22. Write a matlab program to simulate k-values of a beta random variable with parameters b and c. Use this program to simulate values for $k = 100$, $b = 2$, and $c = 1$. Plot a histogram.

23. Given a procedure to simulate an exponential random variable with parameter λ, how can one simulate a random variable with double-exponential distribution. The density function of a double-exponential random variable is given by

$$f(x) = \frac{\lambda}{2} exp(-\lambda |x|) \ .$$

24. Write a matlab program, using polar method, to generate samples from a student's t-distribution with degrees of freedom 5. Generate 1000 samples and plot a histogram.

25. In this problem, you have to derive an acceptance/rejection procedure for generating a standard normal random variable. Assume that you have a program to generate a double-exponential random variable (from the previous assignment) with parameter 1 and density,

$$g(x) = \frac{1}{2} \exp(-|x|) \ .$$

First, find a constant c such that $f(x)/g(x) \le c$, for all x. Then, write an acceptance-rejection program using this values of c. Generate 1000 samples of a standard normal using this method. What fraction of the iterations resulted in acceptance? Compare it with $\frac{1}{c}$.

26. Using the classical Monte-Carlo method, estimate the value of the following integral:
$$I = \int_0^1 x(1-x)^3 dx.$$
Also, estimate the variance, $V(I)$, of this estimator. Plot both the estimates as functions of n, the number of samples generated.

27. Using a normal random generator, you have written previously, estimate the fourth-moment of X, where X is normal with mean 5 and variance 4.
$$E[(X-\mu)^4] = \int_{-\infty}^{\infty} (x-\mu)^4 f_X(x) dx \ .$$
Plot this estimate as a function of the sample size.

28. A pair of fair dice are continually rolled until all possible outcomes 2, 3, ..., 12 have occurred. Let n be the number of rolls needed before all outcomes have occurred. Write a matlab program to simulate a value of n. Generate 1000 random samples of n, estimate its mean and variance, and plot a histogram.

29. Let a discrete random variable X has a probability mass function $p_j = P\{X = j\}$. Define a new function
$$\lambda_n = P\{X = n | X > n-1\} = \frac{p_n}{1 - \sum_{j=1}^{n-1} p_j} \ .$$
The quantities λ_n, $n \geq 1$ are called discrete hazard rates since if we think of X as the lifetime of some item then λ_n represents the probability that an item has reached the age n will die before $n+1$. If we are given the mass function p, we can simulate X. Write a program to simulate X when only λ_ns are given. (Do not convert λs back into p_j to simulate X, use λs directly.)

    For the mass function $p_j = (0.9)^j/9$, first compute the hazard function λ_n for $n = 1, 2, \ldots, 100$. Then, use these λ_n's in your program to simulate 500 values of X. Plot a histogram of these simulated values.

30. Derive and implement a method to generate samples of a Weibull random variable whose probability distribution function is given by:
$$F(x) = 1 - \exp(-\alpha x^{\beta}), \quad 0 < x < \infty \ .$$
Run your program to simulate 1000 values of Weibull random variable with $\alpha = 1$ and $\beta = 0.5$.

31. Write a matlab program that uses the composition method to sample a random variable having the distribution function:
$$F(x) = \int_0^{\infty} x^y e^{-y} dy, \quad 0 \leq x \leq 1 \ .$$

32. Suppose it is easy to generate a random variable from any of the distributions F_i, $i = 1, 2, \ldots, n$. How can we generate a random variable from the distribution:
$$F(x) = \prod_{i=1}^{n} F_i(x) \ .$$
Hint: If X_is are random variables with distributions F_is, respectively, then what random variable X has the distribution F?

5.6 References

The material presented in this section is taken from the textbooks by Robert-Casella [6], Ross [7], and Gentle, and an article by George Marsaglia.

Chapter 6

MONTE CARLO METHODS

6.1 Introduction

As stated earlier, integration plays a very important role in statistical inference. Many inference problems can be written as integrals under some given probability measure. In many situations, this probability measure is too difficult to analytically integrate out, and hence, numerical apporximations are used. One technique for numerical approximation is via sampling, that is, to approximate the integral using samples generated from the given probability measure. This technique is called *Monte Carlo method* and is gaining wide acceptance among the statisticians. In this chapter we study the classifical formulation of the Monte Carlo methods, their improvisations, and applications.

In cases where it is not possible to directly sample from a distribution, one resorts to sampling in an "approximate" fashion. The idea is to sample from a distribution which converges to the desired distribution asymptotically. Later in this chapter, we present an important tool commonly used for this approximate sampling, namely the Markov Chain Monte Carlo technique.

6.2 Monte Carlo Method

The main goal in this technique is to estimate the quantity Θ, where

$$\Theta = \int_{\mathbb{R}} g(x)f(x)dx = E[g(X)] \ , \tag{6.1}$$

for a random variable X distributed according to the density function $f(x)$. $g(x)$ is any function on $\mathbb{R}$ such that Θ and $E[g(X)^2]$ are bounded.. Suppose that we have tools to simulate independent and identically distributed samples from $f(x)$, call them $X_1, X_2, \ldots, X_n$, then one can approximate Θ by the quantity:

$$\hat{\Theta}_n = \frac{1}{n}\sum_{i=1}^{n} g(X_i) \ . \tag{6.2}$$

$\hat{\Theta}_n$ is an estimator of Θ because:

$$E[\hat{\Theta}_n] = \frac{1}{n}\sum_{i=1}^{n} E[g(X_i)] = \Theta \ ,$$

the variance of $\hat{\Theta}_n$ is given by:

$$\begin{aligned}
\mathrm{var}(\hat{\Theta}_n) &= E[(\hat{\Theta}_n - \Theta)^2)] \\
&= E[\hat{\Theta}_n^2] - \Theta^2 \\
&= E[(\frac{1}{n}\sum_{i=1}^{n} g(X_i))^2] - \Theta^2 \\
&= \frac{1}{n}\left(E[g(X)^2] - E[g(X)]^2\right) \\
&= \frac{1}{n}\mathrm{var}(g(X))
\end{aligned}$$

Therefore, as n gets larger the variance of $\hat{\Theta}_n$ goes down to zero and it converges to the mean value Θ. This setup is called the classical Monte Carlo approach where the samples from $f(x)$ are generated in an i.i.d fashion.

For the estimator $\hat{\Theta}_n$ one can compute the expected error as follows. Since Θ is the mean of $\hat{\Theta}_n$, the expected squared error of the estimator, $E[(\hat{\Theta}_n - \Theta)^2]$ is also the variance of $\hat{\Theta}_n$. This suggests that it is possible to obtain better estimators, than the classifical estimator, by reducing the variance of the estimator. In the next section we describe a number of techniques to reduce the variance (or expected squared error).

6.3 Variance Reduction Techniques

Let X be a random variable with a known density function $f(x)$. Our goal is to calculate the expected value of a function g with respect to the density of X. Let

$$\Theta = E(g(X))$$

and we want to estimate Θ using numerical techniques. We will utilize random sampling for the purpose. That is, we will generate samples randomly from the density function, compute g of these samples and average the resulting values. Let

$$X_1, X_2, \ldots, X_n \ ,$$

be the samples obtained via random sampling and let

$$\hat{Y}_n = \frac{1}{n}\sum_{j=1}^{n} g(X_i) \ .$$

According to the central limit theorem, for a large value of n, $\hat{Y}_n$ is approximately normal with mean $E(\hat{Y}_n) = \Theta$ and variance $E(\hat{Y}_n - \Theta)^2)$. $\hat{Y}_n$ is our estimator of Θ, therefore, its variance represents the error in our estimation. This variance goes to zero and the number of samples goes to infinity, but for practical situations restricted to some finite number of samples additional techniques are need to reduce the variance. There are three widely used techniques to accomplish this task:

1. **Using Antithetic Variables**: This method depends on generating averages from the samples which have negative covariance between them, causing overall variance to go

down. Let Y_1 and Y_2 be two identically distributed random variables with mean Θ. Then,

$$\begin{aligned} var(\frac{Y_1+Y_2}{2}) &= \frac{1}{4}(var(Y_1)+var(Y_2)+2cov(Y_1,Y_2)) \\ &= \frac{var(Y_1)}{2}+\frac{cov(Y_1,Y_2)}{2} \end{aligned}$$

If Y_1 and Y_2 are independent, then the last term is zero; if they are positively correlated then the variance of $var((Y_1+Y_2)/2)$ is bigger than $var(Y_1/2)$. But if they are negatively correlated the resulting variance reduces and this case becomes useful for us. The next question is: how to obtain random variables Y_1 and Y_2 with identical distribution but negative correlation.

We will illustrate one such method in the following situation: let $X_1, X_2, \ldots, X_n$ be independent random variables with the distribution functions given by $F_1, F_2, \ldots, F_n$. Let g be a monotonous function. Using the inverse transform method, the X_i's can be generated according to

$$X_i = F_i^{-1}(U_i), \quad \text{for } U_i \text{ uniform } [0,1] \ .$$

Then define

$$Y_1 = g(F_1^{-1}(U_1), F_2^{-1}(U_2), \ldots, F_n^{-1}(U_n)) \ .$$

Since U and $1-U$ are identically distributed and negatively correlated random variables, if we define

$$Y_2 = g(F_1^{-1}(1-U_1), F_2^{-1}(1-U_2), \ldots, F_n^{-1}(1-U_n)$$

then for monotonic function g, Y_1 and Y_2 are negatively correlated. Utilizing negatively correlated functions not only reduces the resulting variance of the sample average but also reduces the computation time as only half the samples need to be generated from uniform $[0,1]$.

2. **Variance Reduction by Conditioning**: Let Y and Z be two random variables:

$$\begin{aligned} var(Y|Z) &= E((Y-E(Y|Z))^2|Y) \\ &= E(Y^2|Z)-(E(Y|Z))^2 \\ var(Y) &= E(Y^2)-E(Y)^2 \end{aligned}$$

Reorganizing these equations and using law of nested expectations, we obtain

$$\begin{aligned} E(var(Y|Z)) &= E(Y^2)-E(E(Y|Z)^2) \\ var(E(Y|Z)) &= E(E(Y|Z)^2)-E(Y^2) \end{aligned}$$

Adding the two equations

$$var(Y) = E(var(Y|Z)) + var(E(Y|Z)) \ .$$

Now compare the two random variables Y and $E(Y|Z)$, both have the same means but comparing the variances

$$var(Y) \geq var(E(Y|Z)) \ .$$

Therefore $E(Y|Z)$ is a better random variable to simulate and average to estimate Θ. Of course, an important issue is how to find an appropriate Z such that $E(Y|Z)$ has significantly lower variance than Y.

3. **Variance Reduction using Control Variates**:
Our goal is to estimate Θ, the expected value of a function g of random variables $X = (X_1, X_2, \ldots, X_n)$. Assume that we know the expected value of another function f of these random variables, call it μ. For any constant a, define a random variable W according to

$$W = g(X) + a(f(X) - \mu) \ .$$

We can utilize the sample averages of W to estimate Θ since $E(W) = \Theta$. Studying the variance of W

$$var(W) = var(g(X)) + a^2 var(f(X)) + 2a cov(g(X), f(X)) \ .$$

Considering $var(W)$ as a function of a, and looking for the minimizer we get

$$a = \frac{-cov(g(X), f(X))}{var(f(X))} \ .$$

The resulting variance of W is given by

$$var(W) = var(g(X)) - \frac{(cov(f(X), g(X)))^2}{var(f(X))} \ .$$

6.4 Importance Sampling

Another technique commonly used for reducing variance in Monte Carlo methods is importance sampling. Importance sampling is different from a classifical Monte Carlo method is that instead of sampling from $f(x)$ one samples from another density $h(x)$, and computes the estimate of Θ using averages of $\frac{g(x)f(x)}{h(x)}$ instead of $g(x)$ evaluated on those samples. Mathematically, we can rearrange the definition of Θ as follows:

$$\Theta = \int g(x)f(x)dx = \int \frac{g(x)f(x)}{h(x)} h(x)dx \ . \tag{6.3}$$

$h(x)$ can be any density function as long as the support of $h(x)$ contains the support of $f(x)$.

Generate samples $X_1, X_2, \ldots, X_n$ from the density $h(x)$ and compute the estimate:

$$\hat{\Theta}_n = \frac{1}{n} \sum_{i=1}^{n} \frac{g(X_i)f(X_i)}{h(X_i)} \ .$$

It can be seen that the mean of $\hat{\Theta}_n$ is Θ and its variance is:

$$\mathrm{var}(\hat{\Theta}_n) = \frac{1}{n} \left(E_h[(\frac{g(X)f(X)}{h(X)})^2] - \Theta^2 \right) \ .$$

Recall that the variance associated with the classical Monte Carlo estimator differs in the first term. In that case, the first term is given by $E_f[g(X)^2]$. It possible that a suitable choice of h can reduce the estimator variance below that of the classical Monte Carlo estimator. We motivate this point using the following example taken from Robert and Casella.

Example 5 *Let X be a Cauchy random variable with parameters (0,1), i.e. X is distributed according to the density function:*

$$f(x) = \frac{1}{\pi(1+x^2)} \ ,$$

and $g(x) = 1_{\{x>2\}}$ be an indicator function. We are interested in estimating:

$$\Theta = \int g(x)f(x)dx = Pr\{X > 2\} \ .$$

We will do so using the notion of importance sampling although it is not too difficult to compute the exact value of Θ analytically to be 0.15.

To start with, consider $X_1, X_2, \ldots, X_n$ to be i.i.d samples from the Cauchy density $f(x)$ and with $g(x)$ being the indicator function, $\hat{\Theta}_n$ is just the frequency of sampled values larger than 2:

$$\hat{\Theta}_n = \frac{1}{n}\sum_{i=1}^{n} 1_{\{X_i>2\}} \ .$$

Variance of this estimator is simply $\frac{\Theta(1-\Theta)}{n}$ or $\frac{0.127}{n}$.

As a second method, we can utitlize the fact that the density $f(x)$ is symmetric around 0, and Θ is just half of the probability $Pr\{|X| > 2\}$. Again, assuming X_i's to be i.i.d. Cauchy, once can estimate Θ using the estimator:

$$\hat{\Theta}_n = \frac{1}{2n}\sum_{i=1}^{n} 1_{\{|X_i|>2\}} \ ,$$

and the variance of this estimator is $\frac{\Theta(1-2\Theta)}{n}$ or $\frac{0.052}{n}$.

If we rewrite Θ as the following integral:

$$\Theta = \frac{1}{2} - \int_0^2 \frac{1}{\pi(1+x^2)}dx \ ,$$

we can obtain another Monte Carlo estimator of Θ. Let $X_1, X_2, \ldots, X_n$ be samples from a uniform random variable taking values between 0 and 2, and define an estimator:

$$\hat{\Theta}_n = \frac{1}{2} - \frac{1}{n}\frac{2}{\pi(1+X_i^2)} \ .$$

It is easy to show that this estimator also estimates Θ and its variance is given by: $\frac{0.0092}{n}$. If we rewrite Θ as the integral:

$$\Theta = \int_0^{\frac{1}{2}} \frac{x^{-2}}{\pi(1+x^{-2})}dx \ .$$

Using i.i.d samples from $U[0,\frac{1}{2}]$ and evaluating average of the function $g(x) = \frac{1}{2\pi(1+x^2)}$ once can futher reduce the estimator variance.

This example shows that a suitable split of the integrand between $f(x)$ and $g(x)$ can lead to a reduction in variance. The question is: what should be new density $h(x)$ that leads to an estimator with minimum variance?

Theoretically, it is easier to answer that question, i.e. it is easy to write down the optimal $h(x)$, but it may not be easy to sample from this optimal density. In practice, one can try many different densities and use experiments to select the better one.

6.5 Tilted Sampling

This is a specific case of importance sampling where the sampling distribution is simply a tilted version of the original density function. In cases where one is interested in estimating tail probabilities, and where these tails are negligible, it may be useful to tilt the density while raising the tail probability. Let $f(x)$ be the original probability density function. Then, the tilted density is given by:

$$f_t(x) = \frac{\exp(tx)f(x)}{M(t)}, \quad \text{where} \quad M_t = \int \exp(tx)f(x)dx \ .$$

The amount of tilt is given by the parameter $-\infty < t < \infty$. For $t > 0$, the positive values of x increase in probability while the negative values decrease in probability. Opposite happens for $t < 0$.

Consider the probability where we are interested in estimating the tail probability:

$$\theta = P\{X \geq a\} = \int_a^\infty f(x)dx \ ,$$

and let a be much larger than the mean of X, i.e. $a >> E[X]$. Using random sampling from the tilted density X_1, X_2,.., X_n from $f_t(x)$, and the estimator is given by:

$$\hat{\theta}_n = \frac{1}{n}\sum_{i=1}^n I_{\{X_i \geq a\}} M(t)\exp(-tX_i) \ .$$

To verify the estimator,

$$\begin{aligned} E[\hat{\theta}_n] &= \frac{1}{n}\sum_{i=1}^n E[I_{\{X_i \geq a\}} M(t)\exp(-tX_i)] \\ &= \frac{1}{n}\sum_{i=1}^n \int_a^\infty M(t)\exp(-tx_i)\exp(tX_i)\frac{f(x_i)}{M(t)}dx_i \\ &= \theta \end{aligned}$$

The variance of $\hat{\theta}_n$ is given by:

$$\text{Var}(\hat{\theta}_n) = \frac{1}{n}[\int I^2_{\{X_i \geq a\}} M(t)^2 \exp(-2tx_i)f_t(x_i)dx_i - \theta^2] \ .$$

The first term on the right side simplifies to:

$$\int_a^\infty M(t)^2\exp(-2tx_i)\exp(tx_i)\frac{f(x_i)}{M(t)}dx_i = M(t)\int_a^\infty \exp(-tx_i)f(x_i)dx_i \ .$$

In the interest of reducing estimating variance, we seek a value of t that minimizes this term. However, this minimization is not straightforward and leads to an indirect solution. Assuming $t > 0$, we can bound the this term by:

$$M(t)\int_a^\infty \exp(-tx_i)f(x_i)dx_i \leq M(t)\int_a^\infty \exp(-ta)f(x_i)dx_i \leq M(t)\exp(-ta) \ .$$

It is easier to minimize this bounding term as follows:

$$\frac{d}{dt}(M(t)\exp(-ta)) = \dot{M}(t)\exp(-ta) - aM(t)\exp(-ta) = 0 \ ,$$

which implies that t should be chosen such that $a = \frac{\dot{M}(t)}{M(t)}$. What does the ratio $\frac{\dot{M}(t)}{M(t)}$ stand for? Since $M(t) = \int \exp(tx)f(x)dx$, we have $\dot{M}(t) = \int x\exp(tx)f(x)$, and that the ratio:

$$\frac{\dot{M}(t)}{M(t)} = \int x\exp(tx)\frac{f(x)}{M(t)}dx = \int xf_t(x)dx \ .$$

In other words, t should be chosen such that a is the mean of the tilted density.
Example: For a standard normal random variable X, find the probability $\Pr\{X \geq a\}$ for $a >> 0$. The quantity to be estimated is given by:

$$\theta = \int_a^{\infty} f(x)dx \ .$$

Since $f(x) = \frac{1}{\sqrt{2\pi}}\exp(-x^2/2)$. The tilted version of $f(x)$ is given by:

$$f_t(x) = \frac{\exp(tx)f(x)}{M(t)} \ .$$

The normalizer can be calculated:

$$\begin{aligned} M(t) &= \int \exp(tx)f(x)dx \\ &= \frac{1}{\sqrt{2\pi}}\int \exp(tx)\exp(-x^2/2)dx \\ &= \frac{1}{\sqrt{2\pi}}\exp(t^2/2)\int \exp(\frac{-1}{2}(x-t)^2)dx = \exp(t^2/2) \ . \end{aligned}$$

In fact, the tilted density can also be written as:

$$f_t(x) = \frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}(x-t)^2) \ .$$

The tilted density is normal with mean t and variance 1. Choosing t such that the mean of the tilted density is a. That is, set the amount of tilt to be a.

Let X_is be sampled from the tilted density. Then, θ can be estimated as follows. Since the ratio

$$\frac{f(x)}{f_t(x)} = \exp(-tx)M(t) = \exp(-ax)\exp(a^2/2) \ ,$$

an estimator of θ is given by:

$$\hat{\theta}_n = \frac{1}{n}\sum_{i=1}^{n} I_{\{X_i \geq a\}}\exp(a^2/2 - aX_i) \ .$$

A better estimator is given by:

$$\hat{\theta}_n = \frac{\exp(a^2/2)}{2n}\sum_{i=1}^{n}\exp(-aY_i) \ , \quad \text{where} \ \ Y_i = a + |X_i - a| \ .$$

6.6 Problems

1. Write matlab programs to estimate the quantity $\theta = Pr\{X < 1\}$, where X is an exponential random variable with mean 1, using two different Monte Carlo estimators. In other words, the random samples used in estimators should have different probability densities. Plot both the estimates as functions of n, the number of samples generated.

2. Suppose we want to estimated θ, where

$$\theta = \int_0^1 \exp(x^2) dx \ .$$

   Implement a Monte Carlo estimator by sampling from the uniform and averaging $\exp(U_i^2)$'s. Also, utilize the idea of variance reduction using antithetic variables to propose another algorithm with less variance. Plot the convergence (estimates versus the sample size) of both the algorithms on the same plot.

3. Suppose that X is exponential with mean 1, and given that $X = x$, Y is exponential with mean x. Given an efficient way to estimate $P\{XY \leq 3\}$. Implement the algorithm and plot the convergence versus the sample size.

4. Suggest a technique to estimate the quantity using

$$\theta = \int_{-\infty}^{\infty} x^4 e^{-(\lambda |x|^\alpha)} dx \ ,$$

   for given values of λ and α. It is not necessary but the use of importance sampling is encouraged. Write a matlab program to implement your estimator and plot its convergence for $\lambda = 1.0$ and $\alpha = 0.1$.

5. Use the technique of importance sampling via tilted densities to estimate the quantity $\theta = Pr\{X > 3.0\}$ where X is a standard normal random variable.

6. Use importance sampling to estimate the quantity:

$$\theta = \int_{-\infty}^{\infty} x \frac{e^{(y-x)^2} e^{-3x}}{Z} dx \ ,$$

   where $Z = \int e^{(y-x)^2} e^{-3x} dx$ and $y = 0.5$. Plot the convergence of your estimator versus the sample size.

6.7 References

Chapter 7

MARKOV CHAIN MONTE CARLO METHODS

In this chapter, we focus on a particular family of discrete time stochastic processes that provide us with a tool to sample from probability distributions. Let X be a continuous random variable with a probability density function $g(x)$, our goal is to generate samples from g. In case g is too complicated to use one of the direct methods described in Chapter 5, one has to resort to approximate methods for sampling from g. One such method is the use of Markov chains to sample from complicated probability density functions, when the direct methods do not apply. The basic idea is to construct a Markov chain X_{t_n}, starting from almost arbitrary initial conditions, but satisfying a set of conditions relating to g, so that as time n gets longer the values of X_{t_n} start resembling the samples of f.

We start this chapter with a brief introduction to Markov chains, and then develop a framework for their use in sampling from given probability density functions. The material presented in this chapter is taken from [3, 6].

7.1 Introduction to Markov Chains

Denote a discrete time stochastic process as: $\{X_t,\ t = t_1, t_2, \ldots\}$. Such a process can be characterized by n^{th}-order joint probability density function, for any $n \geq 1$,

$$f_{X_{t_1},X_{t_2},\ldots,X_{t_n}}(x_1, x_2, \ldots, x_n) \ .$$

Using the rule for total probability, we can factor the joint dendity function as:

$$f_{X_{t_1},X_{t_2},\ldots,X_{t_n}}(x_1, x_2, \ldots, x_n) = f_{X_{t_1}}(x_1) f_{X_{t_2}|X_{t_1}}(x_2|x_1) \ldots f_{X_{t_n}|X_{t_{n-1}},\ldots,X_{t_1}}(x_n|x_{n-1}, \ldots, x_2, x_1) \ ,$$

where the first term on the right side is the marginal density of the process at time t_1, and the remaining terms are the conditional densities. The joint density function is called *translation invariant* if for any $c \geq 0$, we have

$$f_{X_{t_1},X_{t_2},\ldots,X_{t_n}}(x_1, x_2, \ldots, x_n) = f_{X_{t_1+c},X_{t_2+c},\ldots,X_{t_n+c}}(x_1, x_2, \ldots, x_n) \ . \tag{7.1}$$

Definition 16 *A stochastic process is called a Markov process if*

$$f_{X_{t_n}|X_{t_{n-1}},\ldots,X_{t_1}}(x_n|x_{n-1}, \ldots, x_2, x_1) = f_{X_{t_n}|X_{t_{n-1}}}(x_n|x_{n-1}) \ .$$

A Markov process is a stochastic process whose past $(X_{t_{n-2}}, \ldots X_{t_1})$ has no influence on the future (X_{t_n}) if its present $(X_{t_{n-1}})$ is specified. This definition implies that joint density function can be written as a product of one-step conditional densities as follows:

$$f_{X_{t_1},X_{t_2},\ldots,X_{t_n}}(x_1, x_2, \ldots, x_n) = f_{X_{t_1}}(x_1) f_{X_{t_2}|X_{t_1}}(x_2|x_1) \ldots f_{X_{t_n}|X_{t_{n-1}}}(x_n|x_{n-1}) \ . \tag{7.2}$$

Note that to specify a Markov process one needs only a marginal density $f_{X_{t_1}}$ and a sequence of one-step transition densities $f_{X_{t_n}|X_{t_{n-1}}}$. Therefore, the specification of a Markov process avoids the needs for specifying all possible joint density functions.

For the definition, one can establish the following properties associated with Markov processes:

1. The conditional expectation of the future, given the past and the present is equal to the conditional expectation of the future given only the present.

$$E[X_{t_n}|X_{t_{n-1}}, \ldots, X_{t_2}, X_{t_1}] = E[X_{t_n}|X_{t_{n-1}}] \ .$$

Proof: By definition,

$$\begin{aligned} E[X_{t_n}|X_{t_{n-1}}, X_{t_{n-2}}, \ldots, X_{t_1}] &= \int x_n f_{X_{t_n}|X_{t_{n-1}},\ldots,X_{t_1}}(x_n|x_{n-1}, x_{n-1}, \ldots, x_1) dx_n \\ &= \int x_n f_{X_{t_n}|X_{t_{n-1}}}(x_n|x_{n-1}) dx_n = E[X_{t_n}|X_{t_{n-1}}] \end{aligned}$$

2. A Markov process remains Markov if the time index is reversed. That is,

$$f_{X_{t_1}|X_{t_2},\ldots,X_{t_n}}(x_1|x_2, \ldots, x_n) = f_{X_{t_1}|X_{t_2}}(x_1|x_2) \ .$$

Proof: Left side is:

$$\begin{aligned} \frac{f_{X_{t_1},X_{t_2},\ldots,X_{t_n}}(x_1, x_2, \ldots, x_n)}{f_{X_{t_2},\ldots,X_{t_n}}(x_2, \ldots, x_n)} &= \frac{f_{X_{t_1}}(x_1) f_{X_{t_2}|X_{t_1}}(x_2|x_1) \ldots f_{X_{t_n}|X_{t_{n-1}}}(x_n|x_{n-1})}{f_{X_{t_2}}(x_2) f_{X_{t_3}|X_{t_2}}(x_3|x_2) \ldots f_{X_{t_n}|X_{t_{n-1}}}(x_n|x_{n-1})} \\ &= \frac{f_{X_{t_1}}(x_1) f_{X_{t_2}|X_{t_1}}(x_2|x_1)}{f_{X_{t_2}}(x_2)} = f_{X_{t_1}|X_{t_2}}(x_1|x_2) \ . \end{aligned}$$

3. Conditioned on a given value of the present, the past and the future are statistically independent of each other. That is, for $t_1 < t_2 < t_3$ we have

$$f_{X_{t_1},X_{t_3}|X_{t_2}}(x_1, x_3|x_3) = f_{X_{t_1}|X_{t_2}}(x_1|x_2) f_{X_{t_3}|X_{t_2}}(x_3|x_2) \ .$$

Proof:

$$\begin{aligned} f_{X_{t_1},X_{t_3}|X_{t_2}}(x_1, x_3|x_2) &= \frac{f_{X_{t_1},X_{t_3},X_{t_2}}(x_1, x_3, x_2)}{f_{X_{t_2}}(x_2)} \\ &= f_{X_{t_3}|X_{t_2}}(x_3|x_2) f_{X_{t_1}|X_{t_2}}(x_1|x_2) \end{aligned}$$

4. A Markov process satisfies a condition, called Chapman-Kolmogoroff condition, that will be useful later. For a Markov process,

$$f_{X_{t_3}|X_{t_1}}(x_3|x_1) = \int f_{X_{t_2}|X_{t_1}}(x_3|x_2) f_{X_{t_2}|X_{t_1}}(x_2|x_1) dx_2 \tag{7.3}$$

Here are some example of the Markov processes. All stochastic processes with independent increments are Markov processes. For $t_1 < t_2$, the increment $X_{t_2} - X_{t_1}$ is independent of X_{t_1} implies that the variable X_{t_2} does not dependent on the past $(t < t_1)$ given X_{t_1}. As another example, a random walk, as described next, is a Markov processes. For a fixed $\Delta > 0$ and $s > 0$, define the process at $X(i\Delta) = X((i-1)\Delta) \pm s$ with probabilities 0.5 each. Since this defines a process with independent increments, it is a Markov process. Among continuous time processes, a Wiener process is a Markov process.

The concept of stationary stochastic processes is very important in development of MCMC methods.

Definition 17 *A stochastic process is called stationary if its n^{th}-order joint density function is translation invariance, for all $n \geq 1$. In other words, for any collection of times $\{t_1, t_2, \ldots, t_n\}$, we have*

$$f_{X_{t_1}, X_{t_2}, \ldots, X_{t_n}}(x_1, x_2, \ldots, x_n) = f_{X_{t_1+c}, X_{t_2+c}, \ldots, X_{t_n+c}}(x_1, x_2, \ldots, x_n) \ , \tag{7.4}$$

for all $n > 0$ and c.

In particular, for a stationary process the marginal density associated the process does not depend upon the time t,

$$f_{X_t}(x) = f_{X_{t+c}}, \quad \forall c \geq 0 \ ,$$

and for all t. Similarly, the conditional probability density is also invariant in time:

$$f_{X_{t_n}|X_{t_{n-1}}}(x_n|x_{n-1}) = f_{X_{t_2}|X_{t_1}}(x_n|x_{n-1}) \ ,$$

for all n. This invariance of probabilities across times makes stationary processes very important for sampling. For example, in case we are interested in sampling a probability density function g, we can potentially do so by constructing a stationary stochastic process such that the marginal density function at time time is g, i.e. $f_{X_t} = g$. Then, along a sample path, the value X_t, for any t, is a sample from g. Although this scheme seems simple, the problem comes in simulating a stationary stochastic process such that its marginal density function matches the given g. The difficulty of simulating such a stochastic process is same as simulating directly from the original density g. Since we do not know how to simulate directly from g, we can not simulate this stochastic process. However, there is a possibility of simulating another type of process, called a *homogeneous Markov* process, that is not stationary to start with but becomes stationary as $t \to \infty$. Furthermore, it is relatively easy to simulate this process. This idea is the underlying principle behind MCMC methods. Next, we study the construction and properties of homogeneous Markov processes.

Definition 18 *A Markov process is called homogeneous if the conditional density is invariant to a time shift. That is,*

$$f_{X_{t_n}|X_{t_{n-1}}}(x_n|x_{n-1}) = f_{X_{t_2}|X_{t_1}}(x_n|x_{n-1}), \quad \textit{for all} \ \ n \ . \tag{7.5}$$

It is important to note that the marginal density may be, and often is, dependent upon the time shift, i.e.

$$f_{X_{t_1}}(x) \neq f_{X_{t_n}}(x) \ , \quad \text{for some} \ \ n \ .$$

This implies that a homogeneous process, in general, is not a stationary process. When is a homogeneous Markov process stationary? If there exists a density function $g(x)$ such that:

$$g(y) = \int_{\mathbb{R}} f_{X_{t_2}|X_{t_1}}(y|x) g(x) dx \ , \tag{7.6}$$

then the resulting Markov chain is a stationary process. The density function g is called the *stationary probability density* of that Markov chain. An interesting thing to note here is that g is an eigenfunction of the transition function $f_{X_{t_2}|X_{t_1}}$, with an eigenvalue one! This interpretation helps in establishing existing of a stationary probability using results from eigen analysis of transition functions.

Our goal is to construct such homogeneous Markov processes that are not stationary to start with, but converge to stationary processes as the process is followed for a long time. As stated earlier, if the marginal density of this stationary process matches the desired probability density g, then the sample paths of this process sample from g.

7.2 Markov Chains for Sampling from Probabilities

In the next few sections, we develop a framework for using Markov chains to sample from given probability distributions. We start with the simplest case where the given probability distribution P is restricted to a finite set. Later, we allow an infinite but countable set and end by considering the uncountable case. This analysis can be broken into two distinct issues:

1. When does a given homogeneous Markov chain, with a given transition function, converge to a stationary process?
2. For a given probability distribution, how to construct a homogeneous Markov chain that samples from that distribution asymptotically?

7.2.1 Finite-State Space Case

To start with we will consider a discrete time Markov chain that takes values only in a finite set. That is, for any time $X_{t_i} \in \{a_1, a_2, \ldots, a_m\}$. For this finite state setup, the n^{th} order probability mass function is given by:

$$P\{X_{t_n} = a_n, X_{t_{n-1}} = a_{n-1}, \ldots, X_{t_1} = a_1\} \ ,$$

and the Markov property implies:

$$P\{X_{t_n} = a_n | X_{t_{n-1}} = a_{n-1}, \ldots, X_{t_1} = a_1\} = P\{X_{t_n} = a_n | X_{t_{n-1}} = a_{n-1}\} \ .$$

Additionally, we will assume that the Markov chain is homogeneous, i.e. its one-step transition probability distribution does not change in time. This transition probability is denoted by $\Pi_{n-1,n} = \Pi$, implying its independent of n, and the matrix of these elements generates one-step transition probability matrix $\Pi = \{\Pi_{i,j}\}$. Since the state space has cardinality of m, the transition matrix is an $m \times m$ matrix of non-negative elements such that each row adds up to one. The matrix Π is used for compact statements about probabilities of transitions between the state elements. The probability of transition from a_i to a_j in one step is given by $\Pi_{i,j}$; in two steps is given by $\sum_{k=1}^{m} \Pi_{i,k}\Pi_{k,j}$ or $(\Pi * \Pi)_{i,j}$, and in n steps is given by $(\Pi^n)_{i,j}$. Let $P[1]$ be the probability of being in different states at time $t = 1$ ($P[1]$ is an m-element probability mass function), and if the transitions are made according to Π, then the probability vector associated with time $t = n$ is given by

$$P[n] = P[n-1]\Pi = \ldots = P[1]\Pi^{n-1} \ .$$

A special case arises when $P[1] = P$ such that $P\Pi = P$, i.e. P is the eigenvector of the matrix Π with the corresponding eigenvalue given by one. In this situation, $P[n] = P$ for all

n and the resulting Markov process is not only homogeneous but also stationary. P is called the *stationary* probability distribution associated with the Markov chain X_{t_n}. If $P[1] \neq P$, i.e. the initial probability is not an eigenvector of Π with eigenvalue one, then the resulting chain is not stationary, and $P[n] \neq P$ for all n. But it is possible to construct a Markov process, with a transition matrix Π, in such a way that $P[n] \to P$ as $n \to \infty$, even if $P[1] \neq P$. Some additional conditions are required on Π for this situation to hold, and we will study these conditions next. The main question that we address is: Under what conditions on Π does the resulting Markov chain converge to a stationary process? Alternatively, under what conditions on Π does the probability $P[n]$ converge to a stationary probability P? There are two ways of addressing these questions: (i) using Peron-Frobenious theorem, and (ii) by characterizing the resulting Markov chain.

Peron-Frobenius Theorem

There are many ways of establishing the convergence of a resulting Markov chain if the transition matric Π satisfies certain properties. In case of finite state space, the simplest way to state the conditions is using the Peron-Frobenious theorem.

Theorem 7 (Peron-Frobenius) *If* $\Pi^n >> 0$ *for some non-negative integer* n*, then*

1. *there exists an* $X >> 0$ *such that* $X\Pi = X$*, and*
2. *if* λ *is any other eigenvalue of* Π*, then* $|\lambda| < 1$*.*

The symbol $A >> 0$ implies that all elements of that array (vector or matrix) are strictly positive. This theorem states that if there exists an n such that we can go from any state to any other state in n steps with position probability, then the resulting Markov chain has a unique stationary probability vector. This stationary probability is obtained by normalizing X into a probability vector, i.e. $P = X/(\sum_{i=1}^{m} X_i)$. Furthermore, it states that irrespective of the starting condition, the resulting Markov chain converges to a stationary process whose stationary probability is P.

So now we have a way of characterizing the condition under which a Markov chain will sample from a given distribution P. According to Peron-Frobenious theorem, if the transition matrix Π^n has all positive elements for some $n > 0$ and if $P\Pi = P$, then the Markov chain samples from P for $t \to \infty$. For a large $T > 0$, the process at times $t > T$ approximately sample from P. The degree of approximation, or how to find T for a given level of approximation, are discussed in a later section.

Although the condition required in Peron-Frobenious theorem is easy to state, it is difficult to establish for a given Π. Therefore, from a practical point of view, we seek another way of characterizing the convergence of a homogeneous Markov chain with conditions that can be checked easily.

Characterizing Markov Chains

Another way of characterizing the convergence of a homogeneous Markov chain is using a set of properties introduced next. In case of finite state space, Π needs to satisfy only two properties for the Markov chain to converge. These two properties are: ireducibility and aperiodicity. Irreducibility implies that it is possible to visit from any state to any state in a finite number of steps. Aperiodicity implies that the Markov chain does not cycle around in the states with a finite period. We introduce them formally next.

Let $a_1, a_2, \ldots, a_m$ be the states in a finite state space and Π be an $m \times m$ transition matrix. We start with a few definitions.

Definition 19 *Two states are said to communicate if it is possible to go from one to another in a finite number of steps. In other words, a_i and a_j are said to communicate if there exists a positive integer n such that $\Pi^n_{i,j} > 0$.*

For example, for the transition matrix on the left, all states communicate with each other, while, for the matrix on the right, the states a_1 and a_3 do not communicate.

$$\begin{bmatrix} 0.1 & 0.3 & 0.4 & 0.2 \\ 0.2 & 0.4 & 0.0 & 0.4 \\ 0.0 & 0.3 & 0.5 & 0.2 \\ 0.5 & 0.3 & 0.2 & 0.0 \end{bmatrix} \quad \begin{bmatrix} 0.5 & 0.5 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.5 & 0.5 \end{bmatrix} . \tag{7.7}$$

Definition 20 *A Markov chain is said to be irreducible if all states communicate with each other for the corresponding transition matrix Π.*

For example, in Eqn. 7.7 the Markov chain resulting from the first Π will be irreducible while the chain resulting from the second matrix will be reducible into two clusters: one including states a_1 and a_2, and the other including the states a_3 and a_4. Next we define the aperiodicity.

Definition 21 *For a state a_i and a given transition matrix Π, define the period of a_i as:*

$$d(a_i) = GCD\{n : \Pi^n_{i,i} > 0\} ,$$

where GCD implies the greatest common divisor.

In case of the following transition matrix:

$$\begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} ,$$

the period of state a_1 is:

$$d(a_1) = GCD\{2, 4, 6, 8, \ldots, \} = 2 .$$

For the left transition matrix given in Eqn. 7.7, the period of state a_1 is one.

Proposition 5 *If two states communicate, then their periods are same.*

Therefore, in an irreducible Markov chain where all the states communicate, they all have the same period. This is called the *period* of an irreducible Markov chain.

Definition 22 *An irreducible Markov chain is called aperiodic if its period is one.*

Irreducibility and aperiodicity are sufficient to show that the Markov chain converges to a stationary process and it samples from a (unique) stationary probability as we let the chain run for long time.

Theorem 8 *An irreducible, aperiodic, homogeneous Markov chain on a finite state space has the property that $\Pi^n >> 0$ for some $n > 0$. Furthermore, this Markov chain has a unique probability distribution P such that $P\Pi = P$. For any arbitrary starting condition, this Markov chain converges to a stationary process and generates samples from P as the time t goes to infinity.*

In summary, given a probability distribution P on a finite state space, we construct a Markov using a transition matrix Π that satisfies three conditions:

1. The resulting Markov chain is irreducible.
2. The resulting Markov chain is aperiodic.
3. P is a stationary probability of that Markov chain, i.e. $P\Pi = P$.

How to find such a transition matrix is described later using some well-known algorithms.

7.2.2 Countable State Space

In case of infinite state space, the Markov chain convergence requires an additional property – recurrence – to ensure that the chain has a unique stationary probability. To define and analyze recurrence, we need some additional notation.

Let $\mathcal{X} = \{a_1, a_2, \dots, a_i, \dots,\}$ be a countable state space. For a state $a_i \in \mathcal{X}$, let f_{ii}^n be the probability that a homogeneous Markov chain revisits a_i for the first time in n steps. In other words,

$$f_{ii}^n = Pr\{X_{t_n} = a_i, X_{t_{n-1}} \neq a_i, \dots X_{t_1} \neq a_i | X_{t_0} = a_i\} \ .$$

It is easy to see that $f_{ii}^1 = \Pi_{ii}^1$ and we adopt a convention that $f_{ii}^0 = 0$. Let $E_{k,i}$ be an event that a Markov chain, starting from a_i at t_0, reaches a_i in n transitions although its first revisit to a_i takes place at time t_k, for $1 \leq k \leq n$. That is,

$$E_{k,i} = \{X_{t_n} = a_i, X_{t_k} = a_i, X_{t_{k-1}} \neq a_i, \dots X_{t_1} \neq a_i | X_{t_0} = a_i\} \ .$$

The probability of this event is given by:

$$\begin{aligned} Pr(E_{k,i}) &= Pr\{X_{t_n} = a_i | X_{t_k} = a_i\} Pr\{X_{t_k} = a_i, X_{t_{k-1}} \neq a_i, \dots X_{t_1} \neq a_i | X_{t_0} = a_i\} \\ &= \Pi_{ii}^{n-k} f_{ii}^k \end{aligned}$$

Next consider the probability of going from a_i to itself in n transitions, Π_{ii}^n. For $n > 0$, it is given by:

$$\begin{aligned} \Pi_{ii}^n &= \sum_{k=1}^{n} Pr(E_{k,i}) \\ &= \sum_{k=1}^{n} \Pi_{ii}^{n-k} f_{ii}^k = \sum_{k=0}^{n} \Pi_{ii}^{n-k} f_{ii}^k \ . \end{aligned} \tag{7.8}$$

In the last equality, we have simply added a term that equals zero. It is important to note that this equation is not valid for $n = 0$. Extending these ideas to go from a state a_i to another state a_i, we define the quantity:

$$f_{ij}^n = Pr\{X_{t_n} = a_j, X_{t_{n-1}} \neq a_j, \dots X_{t_1} \neq a_j | X_{t_0} = a_i\} \ ,$$

and obtain a relation

$$\Pi_{ij}^n = \sum_{k=0}^{n} f_{ij}^k \Pi_{jj}^{n-k} \ . \tag{7.9}$$

To study and analyze recurrence of a Markov chain, we will need to relate the properties of f_{ii}^n and Π_{ii}^n. In addition to using Eqn. 7.8, we will use the following ideas from the theory of sequences.

Definition 23 *For a sequence $\{a_1, a_2, \ldots, a_n, \ldots\}$, we define the function*

$$A(s) = \sum_{n=0}^{\infty} a_n s^n, \quad |s| < 1 \ ,$$

as the generating function of that sequence.

As two examples of generating functions, let $P_{ii}(s) = \sum_{n=0}^{\infty} \Pi_{ii}^n s^n$, and $F_{ii}(s) = \sum_{n=0}^{\infty} f_{ii}^n s^n$. What happens when we multiply two generating functions and seeking the corresponding resulting sequence?

Proposition 6 *Let $A(s) = \sum_{n=0}^{\infty} a_n s^n$ and $B(s) = \sum_{n=0}^{\infty} b_n s^n$ be two generating functions. Then, the product function has a corresponding sequence given by:*

$$A(s)B(s) = \sum_{n=0}^{\infty} c_n s^n, \qquad \textit{where} \ \ c_n = \sum_{k=0}^{n} a_k b_{n-k} \ . \tag{7.10}$$

In particular, $c_0 = a_0 b_0$.

Applying this idea to the production $P_{ii}(s)F_{ii}(s)$, we obtain:

$$P_{ii}(s)F_{ii}(s) = \sum_{n=0}^{\infty} c_n s^n, \quad \text{where} \ \left\{ \begin{array}{ll} c_n = \sum_{k=0}^{n} f_{ii}^k \Pi_{ii}^{n-k}, & \text{for} \ \ n > 0 \\ c_0 = f_{ii}^0 \Pi_{ii}^0 = 0, & \text{for} \ \ n = 0 \end{array} \right. .$$

From Eqn. 7.8, the coefficients $c_n = \Pi_{ii}^n$ for $n > 0$. Therefore, we have

$$P_{ii}(s)F_{ii}(s) = \sum_{n=1}^{\infty} \Pi_{ii}^n s^n = P_{ii}(s) - P_{ii}^0 s^0 = P_{ii}(s) - 1 \ ,$$

and that gives the relation:

$$P_{ii}(s) = \frac{1}{1 - F_{ii}(s)} \ . \tag{7.11}$$

Using similar ideas and Eqn. 7.9, we can show that

$$P_{ij}(s) = F_{ij}(s)P_{jj}(s) \ . \tag{7.12}$$

Now we are ready state the definition of recurrence and to analyze its implications.

Definition 24 (Recurrence) *We say that a state state $a_i \in \mathcal{X}$ is recurrent under a Markov chain if and only if $\sum_{n=0}^{\infty} f_{ii}^n = 1$.*

Before we proceed further, we first try to understand this statement. f_{ii}^n is the probability of first revisit to a_i in n steps. Since f_{ii}^n and f_{ii}^{n-1} are probabilities of two independent events, their sum provides the probability of that either one occurs. Similarly, $\sum_{n=0}^{N} f_{ii}^n$ is the probability that the process revisits a_i for the first time in first N steps. By the definition of convergence, we have that for any $\epsilon > 0$, there exists an N such that for any $M > N$, we have $\sum_{n=0}^{M} f_{ii}^n > 1 - \epsilon$. In other words, the probability of first revisit being in first M steps is larger than $1 - \epsilon$. Since ϵ is chosen arbitrarily, this implies that the probability of revisit in a finite number of steps occurs with probability arbitrarily close to one.

A state that is not recurrent is said to be *transient*. Given a transition matrix Π, it is not easy to check whether a state is transient or recurrent. This task requires calculating f_{ii}^n for all n, and then checking the condition stated in Def. 24. Since the calculation of f_{ii}^n is difficult, we seek alternate ways of establishing recurrence of states under a Markov chain. This requires the following lemma.

Lemma 1 *1. If $\sum_{n=0}^{\infty} a_n$ converges to a value a, then*

$$\lim_{s \to 1-} A(s) = a \ .$$

2. If $a_n \geq 0$ for all n and $\lim_{s \to 1-} A(s) = a$, then

$$\sum_{n=0}^{\infty} a_n = a \ .$$

Based on this lemma we can establish the relation between Π_{ii}^n and recurrence as follows.

Theorem 9 *A state a_i is recurrent if and only if $\sum_{n=1}^{\infty} \Pi_{ii}^n = \infty$.*

Proof: First, assume that the state a_i is recurrent. According to Defn. 24, this implies that $\sum_{n=0}^{\infty} f_{ii}^n = 1$. According to Lemma 1(1), this further implies that $\lim_{s \to 1-} F_{ii}(s) = 1$. Therefore,

$$\lim_{s \to 1-} P_{ii}(s) = \lim_{s \to 1-} (\frac{1}{1 - F_{ii}(s)}) = \infty \ .$$

Now, using Lemma 1(2),

$$\sum_{n=1}^{\infty} \Pi_{ii}^n = \lim_{s \to 1-} P_{ii}(s) = \infty \ .$$

So, we have shown that if a_i is recurrent, then $\sum_{n=1}^{\infty} \Pi_{ii}^n = \infty$. Next, we show the inverse, i.e. if $\sum_{n=1}^{\infty} \Pi_{ii}^n = \infty$, then a_i is recurrent. We do so by assuming otherwise, that is, $\sum_{n=1}^{\infty} \Pi_{ii}^n = \infty$ by the state a_i is transient, hoping to reach a contradiction. The state a_i being transient implies that $\sum_{n=0}^{\infty} f_{ii}^n < 1$, and therefore, we have $\lim_{s \to 1-} F_{ii}(s) < 1$, using Lemma 1(1). This implies that

$$\lim_{s \to 1-} P_{ii}(s) = \lim_{s \to 1-} (\frac{1}{1 - F_{ii}(s)}) = M < \infty \ .$$

and

$$\sum_{n=0}^{\infty} \Pi_{ii}^n = \lim_{s \to 1-} P_{ii}(s) = M < \infty \ .$$

The last statement is a contradiction since we have assumed that $\sum_{n=1}^{\infty} \Pi_{ii}^n = \infty$. Thus, we have shown that if a_i is transient, then $\sum_{n=1}^{\infty} \Pi_{ii}^n < \infty$. Therefore, we have also shown that $\sum_{n=1}^{\infty} \Pi_{ii}^n = \infty$ implies that a_i is recurrent.

Q. E. D.

So far we have studied recurrence of individual states. It turns out that like aperiodicity, recurrence is also a class property. This is made precise by the following theorem.

Theorem 10 *If two states $a_i, a_j \in \mathcal{X}$ communicate, and a_i is recurrent, then a_j is also recurrent.*

Proof: Since a_i and a_j communicate, there exist n and m such that $\Pi_{ij}^n > 0$ and $\Pi_{ji}^m > 0$. For any $k > 0$, we have $\Pi_{jj}^{n+k+m} \geq \Pi_{ij}^n \Pi_{ii}^k \Pi_{ij}^m$. Therefore,

$$\sum_{k=0}^{\infty} \Pi_{jj}^{n+k+m} \geq \sum_{k=0}^{\infty} (\Pi_{ij}^n \Pi_{ii}^k \Pi_{ij}^m) = \Pi_{ij}^n \Pi_{ij}^m \sum_{k=0}^{\infty} \Pi_{ii}^k \ .$$

Since a_i is recurrent, the last term in ∞, and therefore, $\sum_{k=0}^{\infty} \Pi_{jj}^{n+k+m} = \sum_{k=0}^{\infty} \Pi_{jj}^k = \infty$. Hence, a_j is also recurrent.

Q. E. D.

Based on this result, for an irreducible Markov chain, all states are recurrent if any one state is recurrent. In that case, the Markov chain itself is called recurrent.

A useful property of transience can be the following. If a state a_j is transient, then for all i, we have $\sum_{n=0}^{\infty} \Pi_{ij}^n < \infty$. The proof is as follows: Since a_j is transient, we have from Theorem 9 that $\sum_{n=0}^{\infty} \Pi_{jj}^n < \infty$. Using Lemma 1(1), this implies that $\lim_{s \to 1-} P_{ij}(s) < \infty$. Consider the function: for $|s| < 1$,

$$F_{ij}(s) = \sum_{n=0}^{\infty} f_{ij}^n s^n \leq \sum_{n=0}^{\infty} f_{ij}^n \leq 1 \ .$$

The first inequality comes from the fact that $s^n << \leq 1$ in the region $|s| < 1$, and the second from the fact that $\sum_{n=0}^{\infty} f_{ij}^n$ is a sum of probabilities of mutually exclusive events, i.e. it cannot exceed one. Now, using Eqn. 7.12, we have:

$$P_{ij}(s) = F_{ij}(s) P_{jj}(s) \leq P_{jj}(s) \leq \infty \ .$$

Finally, using Lemma 1(2), we have $\sum_{n=0}^{\infty} \Pi_{ij}^n < \infty$.

Example 6 *Consider the random walk, induced by the transition matrix:*

$$\Pi = \begin{bmatrix} \cdots & & & & \\ p & 0 & q & 0 & \cdots \\ 0 & p & 0 & q & \cdots \\ \vdots & & & & \end{bmatrix} , 0 \leq p, q \leq 1, \quad p + q = 1 \ .$$

Lets try to judge if a state a_i is recurrent or not. The probability of returning to a_i in odd number of transitions is zero, while the probability for even transitions can be stated precisely,

$$\Pi_{ii}^{2n+1} = 0, \quad \Pi_{ii}^{2n} = \begin{pmatrix} 2n \\ n \end{pmatrix} p^n q^n = \frac{2n!}{n!n!} p^n q^n \ .$$

To approximate this probability, we use the Stirling's formula: $n! \approx n^{n+0.5} \exp(-n)\sqrt{2\pi}$. We get

$$\frac{2n!}{n!n!} \approx \frac{(2n)^{2n+0.5} \exp(-2n)\sqrt{2\pi}}{n^{2n+1} \exp(-2n) 2\pi} = 4^n / \sqrt{\pi n} \ .$$

Therefore,

$$\Pi_{ii}^{2n} = (4pq)^n / \sqrt{\pi n} \quad \text{and} \quad \sum_{n=0}^{\infty} \Pi_{ii}^{2n} = \sum_{n=0}^{\infty} (4pq)^n / \sqrt{\pi n} \ .$$

There are two possible cases:

1. **Case 1**: *$p = q = 1/2$. In this case, $4pq = 1$, and*

$$\sum_{n=0}^{\infty} \Pi_{ii}^{2n} = \sum_{n=0}^{\infty} \frac{1}{\sqrt{\pi n}} = \infty \ .$$

This implies that the state a_i is recurrent. Additionally, since Π is irreducible, the resulting Markov chain is also recurrent.

2. **Case 1**: *$4pq < 1$. Let $c = 4pq$, then*

$$\sum_{n=0}^{\infty} \Pi_{ii}^{2n} = \sum_{n=0}^{\infty} \frac{c^n}{\sqrt{\pi n}} = M < \infty \ .$$

   This implies that the state a_i is transient.

To further understand the property of recurrence, we consider another interpretation. We will show that if a state a_i is recurrent, then the Markov will visit it infinitely often with probability one.

Let $Q_{ii} = Pr\{$the chain starting from a_i returns to a_i infinitely often$\}$.

Theorem 11 *The state a_i is recurrent if and only if $Q_{ii} = 1$.*

Proof: Let $Q_{ii}^N = Pr\{$the chain starting from a_i returns to a_i at least N times$\}$. Then,

$$Q_{ii}^N = \sum_{k=1}^{\infty} f_{ii}^k Q_{ii}^{N-1} = Q_{ii}^{N-1} \sum_{k=1}^{\infty} f_{ii}^k \ .$$

Define $f_{ii}^* = \sum_{k=1}^{\infty} f_{ii}^k$. Therefore,

$$Q_{ii}^N = f_{ii}^* Q_{ii}^{N-1} = \ldots = (f_{ii}^*)^{N-1} Q_{ii}^1 \ .$$

For $N = 1$, we have $Q_{ii}^1 = \sum_{k=1}^{\infty} f_{ii}^k = f_{ii}^*$, and therefore, we obtain $Q_{ii}^N = (f_{ii}^*)^N$. Taking the limit,

$$Q_{ii} = \lim_{N\to\infty} Q_{ii}^N = \lim_{N\to\infty} f_{ii}^N = \left\{ \begin{array}{ll} 1 & \text{if } f_{ii}^* = 1 \\ 0 & \text{if } f_{ii}^* < 1 \end{array} \right. \ .$$

By definition, a_i is recurrent if and only if $\sum_{k=1}^{\infty} f_{ii}^k = 1$, or when $f_{ii}^* = 1$. Hence, the state a_i is recurrent if and only if $Q_{ii} = 1$.

Q.E.D.

Similar results can be stated for visiting another state a_j.

Theorem 12 *If two states a_i and a_j communicate with each other, and the class containing them is recurrent, then*

$$f_{ij}^* = \sum_{k=1}^{\infty} f_{ij}^k = 1 \ .$$

Let Q_{ij} be the probability that the Markov chain starts from a_i and visits a_j infinitely often.

Corollary 1 *If the two states a_i and s_j communicate with each other, and the class containing them is recurrent, then $Q_{ij} = 1$.*

Returning to the topic of characterizing the convergence of a homogeneous Markov chain, on a countable state space, we state the following important theorem.

Theorem 13 (Basic Limit Theorem of Markov Chains) 1. *Consider a recurrent, irreducible, aperiodic Markov chain. Let Π_{ii}^n be the probability of the Markov chain entering* *the state a_i in n transitions, given that it starts at a_i. Let f_{ii}^n be the probability of first revisit to a_i be at the n^{th} step. Thus,*

$$\Pi_{ii}^n - \sum_{k=0}^{n} f_{ii}^k \Pi_{ii}^{n-k} = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if } n > 0 \end{cases} .$$

Then,

$$\lim_{n\to\infty} \Pi_{ii}^n = \frac{1}{\sum_{n=0}^{\infty} n f_{ii}^n} . \tag{7.13}$$

2. *Under the same conditions,*

$$\lim_{n\to\infty} \Pi_{ji}^n = \lim_{n\to\infty} \Pi_{ii}^n . \tag{7.14}$$

The quantity $\sum_{n=0}^{\infty} n f_{ii}^n$ is the average number of steps taken by the Markov chain for its first revisit to the state a_i. It is also called the *mean recurrence time.*

Let $P_i = \lim_{n\to\infty} \Pi_{ii}^n$. If $P_i > 0$ for the state a_i in an aperiodic, recurrent class, then $P_j = \lim_{n\to\infty} \Pi_{ij}^n > 0$ for all the states a_j in that class. This class is called *positive recurrent.* A Markov chain is called *positive recurrent* if it is irreducible, aperiodic, recurrent, and $P_i > 0$ for at least one state in the class. If $P_i = 0$ is a recurrent class, then that class is called *null recurrent.* The important result on convergence of such a chain comes in form of the next theorem.

Theorem 14 *For an irreducible, aperiodic, positive recurrent Markov chain, on a countable state space, there exists a unique probability mass function $\{P_j\}$ such that*

$$\sum_i P_i \Pi_{ij} = P_j \quad , \sum_j P_j = 1, \quad P_j > 0, \ \forall j .$$

In other words, the resulting Markov chain has a unique stationary probability function. With this result, we have established the conditions for a homogeneous chain to converge to a stationary process. Compared to the conditions we derived for the finite state space in Theorem 8, we have an additional condition for the countable case – positive recurrence – for the Markov chain to work. The remaining case of uncountable state space has a similar condition.

7.2.3 Uncountable State Space

In this case one has to modify the definition of irreducibility, recurrence, and positive recurrence, to account for the fact that now we have a continuous random variable. The earlier definition is not valid since the probability of visiting any specific value is zero, even we have a perfect sampler from a density. Hence, the notion of revisit, or visiting infinitely often, is not applicable here. The definition of recurrence is now related to sets of positive measure. Some books in Markov chain has also called this recurrence as *Harris recurrence.*

For a set $A \subset \mathcal{X}$, Let $\tau_A = \inf\{n \geq 1 | X_n \in A\}$. In other words, τ_A is the time of first revisit to A. As a convention, $\tau_A = \infty$, if the chain never returns to A.

Definition 25 *A Markov chain is ϕ-irreducible if there exists a non-zero measure ϕ on $\mathcal{X}$ such that $Pr\{\tau_A < \infty | X_0 = x\} > 0$ for all $x \in \mathcal{A}$ and $A \subset \mathcal{X}$ that satisfy $\phi(A) > 0$.*

Definition 26 *A ϕ-irreducible Markov chain with a stationary probability density $f(x)$ is Harris recurrent if for all $A \subset \mathcal{X}$ with $f(A) > 0$ and for all $x \in \mathcal{X}$, we have $Pr\{\tau_A < \infty | X_0 = x\} = 1$.*

Theorem 15 *For a ϕ-irreducible, aperiodic, Harris recurrent Markov chain, with a stationary density f, the stationary density is unique. In other words, f is the only probability density function that satisfies:*

$$\int_x f(x)\Pi(x,y)dx = f(y) \quad , \int_x f(x) = 1, \quad f(x) \geq 0, \ \forall x \ .$$

7.3 Metropolis-Hastings Algorithm

One of the most popular MCMC technique used in approximate sampling from complicated distributions is the Metropolis-Hastings algorithm. The setup is as earlier: we are interested in generating samples of a random variable X distributed according to the density $f(x)$. In addition to $f(x)$, we will assume having a density $q(y|x)$ that satisfies the following properties:

1. It is easy to sample from $q(\cdot|x)$ for all x.
2. The support of q contains the support of $f(x)$.
3. The functional form of $q(y|x)$ is known or $q(y|x)$ is symmetric in y and x. As shown later, it is not necessary to know the normalizing constant in $q(y|x)$ as long as it does not depend upon x.

Given $f(x)$ and a choice of $q(y|x)$, that satisfies the above mentioned properties, the Metropolis-Hastings algorithm can be stated as follows:

Algorithm 34 (Metropolis-Hastings Algorithm) *Choose an initial condition X_0 in the support of $f(x)$. The Markov chain $X_1, X_2, \ldots, X_n$ is constructed iteratively according to the steps:*

1. *Generate a candidate $Y \sim q(y|X_t)$.*
2. *Update the state to X_{t+1} accroding to:*

$$X_{t+1} = \begin{cases} Y & \text{with probability } \rho(X_t, Y) \\ X_t & \text{with probability } 1 - \rho(X_t, Y) \end{cases} ,$$

where $\rho(x,y) = \min\{\frac{f(y)q(x|y)}{f(x)q(y|x)}, 1\}$.

$q(y|x)$ is called the *proposal density* and $\rho(x,y)$ is called the *acceptance-rejection* function.

In order to analyze this algorithm, consider first the case where the ratio $\frac{f(Y)q(X_t|Y)}{f(X_t)q(Y|X_t)}$ values more than one and hence the acceptance-rejection function takes the value one. In this case, we set $X_{t+1} = Y$ with probability one. In other words, whenever this ratio exceeds once we change the state to the candidate. In case this ratio goes below one, we set X_{t+1} to Y with probability $\rho(X_t, Y)$. Higher the value of $\rho(X_t, Y)$ is, higher are the chances of accepting Y as the new state. Also, note that the normalizing constants in the two densities f and q cancel out and hence are not explicitly needed. However, if the normalizing constant for $q(y|x)$ depends upon x, then it does not cancel out and is neeed in the expression for ρ. In the algorithm, one generates samples from the proposal q at every step independently but the elements of the chain are not independent of each other. In fact, many times it is possible to have X_t and X_{t+1} be identical.

There are some special cases that occur often in practice:

1. In case, $q(y|x)$ is symmetric in the two arguments, the expression for the acceptance-rejection function simplies:
$$\rho(x, y) = \min\{\frac{f(y)}{f(x)}, 1\} \ ,$$
where $\frac{f(y)}{f(x)}$ is often called the *likelihood ratio.*

2. **Independent M-H**: In cases where the proposal density is independent of the current state, i.e. $q(y|x) = q(y)$, then the algorithm is called Independent Metropolis- Hastings algorithm. In this case, the acceptance-rejection function becomes:
$$\rho(x, y) = \min\{\frac{f(y)q(x)}{f(x)q(y)}, 1\} \ .$$

3. **Random Walk M-H**: In some applications, it is useful to generate proposals using a random walk. That is, the proposal is obtained using the equation:
$$Y = X_t + \epsilon \ ,$$
where ϵ has density $g(\epsilon)$ that is symmetric and unimodal at zero. The proposal density is symmetric:
$$q(y|x) = g(y - x) \ ,$$
and the algorithm simplifies.

For a broad set of proposal densities, it can be shown that the M-H algorithm generates a Markov chain that asymptotically becomes stationary and generates samples from the target density $f(x)$. To study the proof, we need to define the detailed balance condition.

Definition 27 *A homogenous Markov chain with transition density function* Π *is said to satisfy the detailed balance condition if there exists a function* f *such that:*
$$f(x)\Pi(x, y) = f(y)\Pi(y, x) \ , \tag{7.15}$$
for every pair (x, y).

We anticipate the target density $f(x)$ to be the function that makes a M-H Markov chain has detailed balance.

Theorem 16 *Suppose that a Markov chain with the transition function* Π *satisfies detailed balance condition with* f *being a probability density function. Then,* f *is a stationary probability density of that chain.*

Proof: We want to show that, for all $y \in \mathbb{R}$,
$$\int_x f(x)\Pi(x, y)dx = f(y) \ .$$

In other words, we want to show that, for any measurable subset $B \subset \mathbb{R}$, we have:
$$\int_{y\in B}(\int_x f(x)\Pi(x, y)dx)dy = \int_{y\in B} f(y)dy \ .$$

We start with the left side:

$$\begin{aligned}\int_{y \in B} (\int_x f(x)\Pi(x,y)dx)dy &= \int_{y \in B} (\int_x f(y)\Pi(y,x)dx)dy \\ &= \int_{y \in B} f(y)dy(\int_x \Pi(y,x)dx) \\ &= \int_{y \in B} f(y)dy \ .\end{aligned}$$

Q.E.D.

Returning to the Markov chain generated by the M-H algorithm, we state the following result.

Theorem 17 *For any proposal density that satisfies the three conditions listed earlier, f is a stationary probability density of the Markov chain produced by M-H algorithm.*

Proof: We just need to that the transition function Π of the M-H Markov chain satisfies the detailed balance condition with the given $f(x)$. The transition kernel for the Markov chain generated by M-H algorithm is:

$$\Pi(x,y) = \rho(x,y)q(y|x) + (1 - r(x))\delta_x(y) \ , \tag{7.16}$$

where $r(x) = \int_y \rho(x,y)q(y|x)dy$. We need to show that Eqn. 7.15 is satisfied. Once we substitute Π in that equation, we obtain two terms on each side. We will show that respective terms balance each other on either side:

$$\rho(x,y)q(y|x)f(x) + (1 - r(x))\delta_x(y)f(x) = \rho(y,x)q(x|y)f(y) + (1 - r(y))\delta_y(x)f(y) \ . \tag{7.17}$$

Consider the case when x and y are such that $f(y)q(x|y) < f(x)q(y|x)$. In this case,

$$\rho(x,y) = \frac{f(y)q(x|y)}{f(x)q(y|x)}, \quad \rho(y,x) = 1 \ .$$

Substituting then in Eqn. 7.17, the first term on left side becomes:

$$\frac{f(y)q(x|y)}{f(x)q(y|x)} f(x)q(y|x) = f(y)q(x|y) \ ,$$

which is what the first term on the right side is. Similar equality occurs when $f(y)q(x|y) > f(x)q(y|x)$, and similarly one can match the second terms on both sides. Hence, this Markov chain satisfies the detailed balance condition, and therefore $f(x)$ is a stationary probability density associated with the Markov chain.

Lemma 2 *If a Metropolis-Hastings Markov chain is f-irreducible, then it is Harris recurrent.*

One implication of this lemma is that if we have f-irreducibility of a Markov chain, along with aperiodicity, then the stationary probability density is unique and the Markov chain converges to that probability density asymptotically. f-irreducibility of a M-H Markov chain comes from the fact that $q(y|x) > 0$ whenever $f(y) > 0$. A sufficient condition for M-H Markov chain to be aperiodic is that the algorithm allows events such as $\{X_{t+1} = X_t\}$.

Theorem 18 *For a given density function $f(x)$, and proposal density $q(y|x)$ that satisfies the positivity condition. Let X_t be a Markov chain generated by Metropolis-Hastings algorithm for this setup. The, for a function $h(x)$ that satisfies*

$$\int h(x)f(x)dx < \infty \ .$$

Then, we have

$$\lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} h(X_t) = \int h(x)f(x)dx \ .$$

Example 7 *Let X be a gamma random variable with shape parameter $\alpha > 0$ and scale parameter $\beta > 0$. Its density function $f(x)$ is given by:*

$$f(x|\alpha,\beta) = \frac{1}{\beta\Gamma(\alpha)} \exp(-\frac{x}{\beta})(\frac{x}{\beta})^{\alpha-1} \ .$$

We want to use M-H algorithm to generate samples from $f(x)$. Recall that if α is a positive integer, then X is the sum of α independent random variables, each being exponentially distributed with mean β. Let n be a positive integer closest to α. Then, a random variable Y with density $f(y|n,\beta)$ can be simulated using n independent exponential random variables. We will use Y as a proposal in an independent M-H algorithm.

Algorithm 35 *1. Generate $Y_t \sim f(y|n,\beta)$.*

2. Update the state to X_{t+1} according to:

$$X_{t+1} = \begin{cases} Y & \text{with probability } \rho(X_t,Y) \\ X_t & \text{with probability } 1-\rho(X_t,Y) \end{cases} \ ,$$

where

$$\rho(x,y) = \min\{\frac{f(y)q(x)}{f(x)q(y)}, 1\} = \frac{y}{x}^{\alpha-n}$$

7.4 Gibbs Sampler

Gibbs sampler is another commonly used tool for generating Markov chains with suitable asymptotic properties. By construction, Gibb's sampler applies only to the problem of sampling from multivariate densities. Let $X = (X_1, X_2, \ldots, X_p) \in \mathbb{R}^p$ be a vector of random variables with the joint density function given by $f(x_1, x_2, \ldots, x_p)$. Our goal is to generate samples from f and we will do so by constructing a Markov chain on $\mathbb{R}^p$. In order to use Gibbs sampling we make the following assumption: we know the conditional densities

$$f_i(x_i|x_j, \ j \neq i), \ i = 1, 2, \ldots, p \ ,$$

and have method(s) to sample from each of these. These conditional densities are called the *full conditionals* and have the simplicty of univariate densities.

An algorithm for Gibbs sampler is as follows:

Algorithm 36 *Let $X^{(t)} = [X_1^{(t)} \ X_2^{(t)} \ldots X_p^{(t)}] \in \mathbb{R}^p$ be the value of Markov chain at time t. Following steps describe an update from $X^{(t)}$ to $X^{(t+1)}$.*

1. *Generate* $X_1^{(t+1)} \sim f_1(x_1|X_2^{(t)}, X_3^{(t)}, \ldots, X_p^{(t)})$.

2. *Generate* $X_2^{(t+1)} \sim f_2(x_2|X_1^{(t+1)}, X_2^{(t)}, \ldots, X_p^{(t)})$. $\vdots$

3. *Generate* $X_p^{(t+1)} \sim f_p(x_p|X_1^{(t+1)}, X_2^{(t+1)}, \ldots, X_{p-1}^{(t+1)})$.

An important property of the Gibbs sampler is that even for large values of p, one samples from a univariate density at each step. This makes Gibbs sampler very attractive for large dimensional problems such as image analysis.

A specific case of Gibbs sampler for $p = 2$ is the *bivariate* Gibbs sampler. Let X and Y be two scalar random variables with the joint density function $f(x, y)$ and the full conditionals:

$$f_1(x|y) \text{ and } f_2(y|x) .$$

Gibbs sampler can be constructed as follows: start with some initial condition (x_0, y_0) and iterate according to:

Algorithm 37 (Bivariate Gibbs Sampler) 1. *Generate* $x_{t+1} \sim f_1(x|y_t)$.

2. *Generate* $y_{t+1} \sim f_2(y|x_{t+1})$.

3. *Set* $t = t + 1$ *and go to Step 1.*

To illustrate this special case, consider the bivariate normal density:

$$(X, Y) \sim N(0, \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}) ,$$

with the full conditionals:

$$f_1(x|y) = N(\rho y, 1 - \rho^2), \quad f_2(y|x) = N(\rho x, 1 - \rho^2) .$$

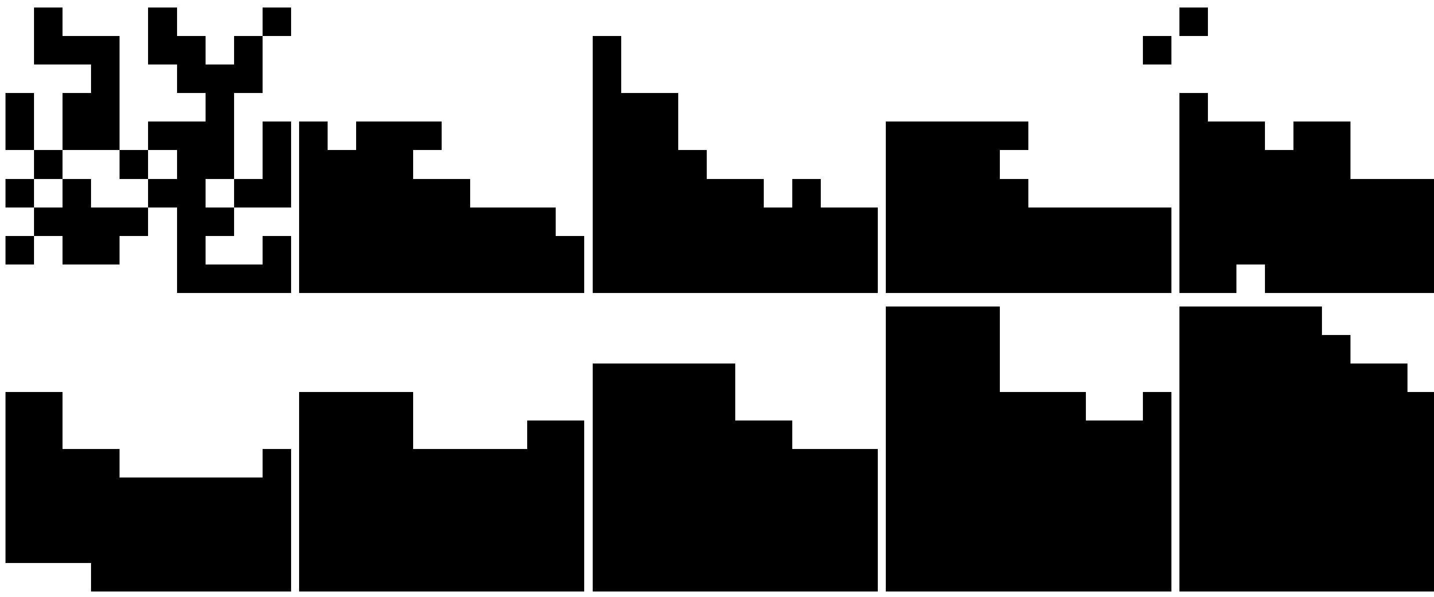
There are many interesting properties of Gibbs samplers. A Gibbs sampler can be considered a special case of the M-H algorithm with the proposal density given by the full conditionals. Also, in Gibbs sampler one always accepts the proposed state as opposed to the acceptance/rejection of the M-H algorithm.

Gibbs sampler by construction apllies only to multivariate densities. What if we are interested in sampling from a univariate density $f(x)$ with $p = 1$.

7.4.1 Markov Random Fields

An important applications of Gibbs sampler is in generating samples from Markov random field models. We begin by introducing Markov randon fields.

Let S be a collection of indexed locations, e.g. indices on a uniform lattice, in a plane. For each site $s \in S$, we assign a random variable $X(s) \in \mathbb{R}$, and the collection $X \equiv \{X(s), s \in S\}$ as a field of random variables. Elements of S are often called the pixel locations and the ransom variable $X(s)$ is called the pixel value at s. Before we introduce specific models, we introduce some additional notation. We will consider the sites geographically close to $s \in S$ as neighbors of s, denoted by $N_s \subset S$. For example, in a uniform, square lattice, we can choose the sites immediately north, south, east and west to s can form a neighborhood of s. Alternatively, one can include the diagonal neighbors also. Depending on the application, different neighborhoods can be chosen for a random field. With this notation, a MRF is defined as follows.



Definition 28 (Markov Random Field) *If the conditional probability density of a random variable $X(s)$, given the remaining variables $\{X(r), r \neq s\}$, is the same as the conditional probability density of $X(s)$ given only its neighbors $\{X(s)|X(r), r \in N_s\}$, for all s, then X is called a Markov random field. We have, for all $s \in S$,*

$$f(X(s)|X(r), r \neq s) = f(X(s)|X(r), r \in N_s) \ . \tag{7.18}$$

A simple example of MRF is the Ising model where the pixel values $X(s)$ are allowed only two values 1 or -1.

Example 8 (Ising Model) *For positive constants H and J, the joint probability distribution of all pixel values in an Ising model is given by: for $X \equiv \{X(s) \in \{-1, 1\}, s \in S\}$,*

$$P(X) = \frac{1}{Z} e^{H \sum_s X(s) + J \sum_s X(s) (\sum_{r \in N_s} X(r))} \ ,$$

where Z is the normalizing constant, often called the partition function. The conditional probability distribution is given by:

$$P(X(s)|X(r), r \in N_s) = \frac{e^{(H + J(\sum_{r \in N_s} X(r)))(1 + X(s))}}{1 + e^{2(H + J(\sum_{r \in N_s} X(r)))}} \ .$$

The Gibbs sampler algorithm for sampling from this Ising model is: for any fixed ordering of indices, generate a sample from $P(X(s)|X(r), r \in N_s)$ and use it to update $X(s)$.

7.5 Continuous Time Markov Processes

7.6 Problems

1. Write a matlab program to simulate a discrete-time, finite-state Markov process with the following transition matrix:

$$\Pi = \begin{bmatrix} 0.2 & 0.2 & 0.1 & 0.5 \\ 0.1 & 0.3 & 0.4 & 0.2 \\ 0.3 & 0.2 & 0.3 & 0.2 \\ 0.1 & 0.3 & 0.1 & 0.5 \end{bmatrix} .$$

Generate ONE sample path of this Markov chain, and plot the relative frequencies (versus i) with which the path visits the fours states.

2. Repeat the previous Problem for a Markov chain having the transition matrix:

$$\Pi = \begin{bmatrix} 0.5 & 0.5 & 0.0 & 0.0 \\ 0.1 & 0.9 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.3 & 0.7 \\ 0.0 & 0.0 & 0.2 & 0.8 \end{bmatrix} .$$

3. Let X_t be a Markov chain generated using some initial probability $P[1]$ and the transition matrix Π,

$$\Pi = \begin{bmatrix} 0.1 & 0.3 & 0.4 & 0.2 \\ 0.2 & 0.1 & 0.3 & 0.4 \\ 0.4 & 0.2 & 0.1 & 0.3 \\ 0.3 & 0.4 & 0.2 & 0.1 \end{bmatrix} .$$

First verify if X_t is (i) irreducible, and (ii) aperiodic, and then find the stationary probability vector for X_t. Using your program from Problem 1 verify that the averages along a sample path converge to the stationary probability. If $f : \{a_1, a_2, a_3, a_4\} \to I\!R$ as as follows:

$$f(a_1) = 2.0,\ f(a_2) = 1.0,\ f(a_3) = 2.5,\ f(a_4) = -1.0 ,$$

show through simulation that

$$\frac{1}{n}\sum_{i=1}^{n} f(X_i) \overset{n\to\infty}{\to} \sum_{j=1}^{4} f(a_j)P(a_j) ,$$

where P is the stationary probability.

4. Repeat Problem 3 for the transition matrix

$$\Pi = \begin{bmatrix} 0.1 & 0.3 & 0.4 & 0.2 \\ 0.2 & 0.4 & 0.0 & 0.4 \\ 0.0 & 0.3 & 0.5 & 0.2 \\ 0.5 & 0.3 & 0.2 & 0.0 \end{bmatrix} .$$

5. Consider a Markov chain generated by the transition matrix:

$$\Pi = \begin{bmatrix} p_1 & (1-p_1) & 0 & 0 & \dots \\ p_2 & 0 & (1-p_2) & 0 & \dots \\ \vdots & & & & \\ p_m & \dots & & (1-p_m) & \dots \\ \vdots & & & & \end{bmatrix} ,$$

where $0 < p_m < 1$ for all m. Establish the following conditions for the Markov chain:

(a) Show that the Markov chain is irreducible.

(b) Show that the Markov chain is aperiodic.

(c) Study the recurrence of this chain using the following steps: the recurrence of a chain has been defined using the condition $\sum_{n=1}^{\infty} f_{ii}^n = 1$.

i. First show that $f_{11}^n = p_n \prod_{m=1}^{n-1}(1 - p_m)$.

ii. Second, show that we can rewrite $f_{11}^n = u_{n-1} - u_n$, where

$$u_n = \prod_{m=1}^{n} (1 - p_m) \ .$$

iii. Next, show that $u_n \to 0$ as $n \to \infty$ if and only if $\sum_{m=1}^{\infty} p_m = \infty$.

iv. State the necessary and sufficient condition for the chain to be recurrent.

Write a matlab program to simulate this Markov chain for $p_m = \frac{1}{m+1}$. Show 10 sample paths for this chain on a plot.

6. Show that a finite, irreducible Markov chain whose transition matric is doubly stochastic (all the rows and the columns each add to 1) has a unique stationary measure given by the uniform probability vector.

7. Consider a finite state (say m-state) random walk with the following transition matrix:

$$\Pi = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots \\ p & 0 & (1-p) & 0 & \dots \\ 0 & p & 0 & (1-p) & \dots \\ \vdots & & & & \\ 0 & \dots & & 0 & 1 \end{bmatrix} ,$$

where $0 < p < 1$. Define

$$d(k) = E[\text{time to reach states } a_1 \text{ or } a_m \mid \text{initial state is } a_k] \ .$$

For $p = 0.5$ and $m = 10$, estimate $d(k)$ via simulation and plot the estimated values of $d(k)$ versus k.

8. Write a matlab program implementing the Metropolis-Hastings algorithm to sample from the density

$$f(x) = \frac{x^2 |sin(\pi x)| e^{-x/2}}{\int_{I\!R_+} x^2 |sin(\pi x)| e^{-x/2} dx} , \quad x > 0 \ .$$

You have to decide what q (proposal density) you want to use. Choose positive numbers to start the Markov chain. Show several sample paths of this Markov chain for $t = 1$ to $t = 10000$. Estimate the value of $E[X]$ and $var(X)$.

9. Consider a finite state, irreducible Markov chain whose transition matrix is idempotent, i.e. $\Pi^2 = \Pi$. Show that the chain is aperiodic.

10. We revisit the problem of sampling from a posterior density when it is easy to sample from the prior density. (Earlier, we solved the problem using importance sampling.) For two continuous random variables X and Y, the Bayes' rule can be stated as:

$$f_{X|Y}(x|y) = \frac{f_{Y|X}(y|x) f_X(x)}{f_Y(y)} \ .$$

$f_{X|Y}(x|y)$ is the *posterior density*, $f_X(x)$ is the prior density, $f_{Y|X}(y|x)$ is called the likelihood function (as a function of x), and $f_Y(y)$ is a normalizer (which is fixed for a given y). Let X be a normal random variable with mean 5 and variance 4, and the conditional density of Y given $X = x$ be normal with mean $2x$ and variance 9. That is,

$$f_X(x) = \frac{1}{\sqrt{2\pi\ 4}} \exp(-\frac{1}{2\ 4}(x-5)^2) \ .$$

and

$$f_{Y|X}(y|x) = \frac{1}{\sqrt{2\pi\ 9}} \exp(-\frac{1}{2\ 9}(y-2x)^2) \ .$$

Choose the prior $f_X(x)$ as the proposal density and the posterior $f_{X|Y}(x|y)$ as the target density. Write a matlab program implementing Metropolis-Hastings algorithm to sample from the posterior for $y = 6$. Show several sample paths of this Markov chain for $t = 1$ to $t = 10000$ and estimate the posterior mean.

11. A gamma random variable X with parameters (α, β) has the following density function

$$f(x) = \frac{\beta e^{-\beta x} (\beta x)^{\alpha-1}}{(\alpha-1)!} \ .$$

If α is a positive integer, then X is just the sum of α independent exponential random variables each with parameter β. Therefore, we know how to simulate X for α being a positive integer. When $\beta = 1$, this simulation can be used to generate samples from gamma density with non-integer α, using Metropolis-Hastings algorithm as follows. Let $[\alpha]$ denote the largest integer below α.

Gamma Metropolis-Hastings

(a) Generate $Y_t \sim gamma([\alpha], [\alpha]/\alpha)$.

(b) Take

$$X_{t+1} = \begin{cases} Y_t & \text{with probability } \rho_t \\ x_t & \text{otherwise} \end{cases}$$

where

$$\rho_t = min\{(\frac{Y_t}{x_t} \exp(\frac{x_t - Y_t}{\alpha}))^{\alpha-[\alpha]}, 1\} \ .$$

(c) Set $t = t + 1$, go to Step 1.

Write a matlab program to generate samples from $X \sim gamma(2.5, 1)$. Show several sample paths of this Markov chain for $t = 1$ to $t = 10000$ and estimate the mean of X.

12. Write a matlab program implementing Langevin's stochastic difference equation to simulate from a given density:

$$f(x) = \frac{x^2 |sin(\pi x)| e^{-x/2}}{\int_{I\!R_+} x^2 |sin(\pi x)| e^{-x/2} dx} \ , \quad x > 0 \ .$$

Show a few sample paths for arbitrary initial conditions and estimate the mean of f.

13. Use a Metropolis-adjusted Langevin's algorithm to simulate from the same f. Show a few sample paths and estimate the mean of this density.

14. **Dynamic Monte Carlo Method** Consider the following filtering situation. The state of a system evolves according to the equation"

$$x_t = x_{t-1}^2 sin(x_{t-1}) + n_t \ ,$$

where $n_t \sim N(0,1)$. Assume that x_1 is known to be 1.0 (you do not have to estimate it. The state is observed according to the observation model:

$$y_t = x_t^2 + m_t \ , \quad \text{where} \quad m_t \sim N(0,1)$$

(a) Generate a sample path of the sequence x_t for $t = 1, 2, \ldots, 20$.

(b) Generate a sample path of the observation process, y_t, for $t = 1, 2, \ldots, 20$.

(c) Using these observations, write a program to implement the dynamic Monte Carlo algorithm for estimating x_t's. Use $M = 5000$ for the sample size at each time t. Plot the estimation error $\|x_t - \hat{x}_t\|^2$ versus t.

7.7 References

Chapter 8

SMOOTHING & DENSITY ESTIMATION

8.1 Introduction

A fundamental problem in statistical analysis of data is to find the probability distribution that generated the data in the first place. This problem is the exact opposite of the simulation problem studied in Chapter 5. There we are interested in generating samples x_i from a given probability model f while here we are interested in estimating f given a collection of samples x_is from that probability.

How can one estimate f from its samples? There are two broad categories of approaches that are followed. One is to assume a form, or a structure, on f and then find the f in that form that "best" fits the data. An important example is to assume that f belongs to a parametric family, i.e. $f \equiv f(x;\theta)$, f is a probability density function in x parameterized by θ such that the search for best f reduces to that for best θ. Density estimation now reduces to parameter estimation and techniques for maximum likelihood estimation, described in Chapter 3 can be applied. The other, and more general, idea is not to assume any particular form on f and let the data dictate estimation of f. This approach called *nonparametric* density estimation is more challenging and is introduced here in this chapter. Please refer to the text [8] for a detailed analysis of density estimation and smoothing techniques.

8.2 Common Techniques for Density Estimation

1. **Histogram**
2. **Naive Density Estimation**
3. **Kernel Estimators**:

8.3 Problems

1. Consider a real-valued random variable, X, having the density function $\frac{e^{-H(x)}}{Z}$ where $H(x) = \frac{1}{2\sigma^2}(x-m)^2$. For $m = 5$ and $\sigma = 0.5$, generate samples of X using discrete version of Langevin's equation: select $x(0) = 0$, $\Delta = 0.1$ and let

$$x(k) = -\frac{\partial H}{\partial x}(x_{k-1})\Delta + \sqrt{2\Delta}w$$

where $w \sim N(0,1)$. Let $\{x(k)\}$, $k = 1, \ldots, 1000$ be the samples generated. It may be a good idea to arrange the samples in increasing or decreasing order. Also, use the same sample set for each of the following methods.

(a) Plot the histogram of these samples, for window sizes 0.5 and 0.1.

(b) Calculate and plot the density estimate using the "naive" estimator for $h = 0.5$ and $h = 0.1$. The estimate is given by,

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}w(\frac{x - x(i)}{h}) ,$$

where $w(x) = \begin{cases} 1 & |x| \leq 1 \\ 0 & o.w \end{cases}$.

(c) Calculate and plot the nearest neighbor density estimate for $k = 5$ and 10. The estimate is given by,

$$\hat{f}(x) = \frac{1}{nd_k(x)}\sum_{i=1}^{n} w(\frac{x - x(i)}{d_k(x)}) ,$$

where $d_k(x)$ is the minimum half-width of an interval around x containing k points.

(d) Calculate and plot the density estimate using the kernel estimator assuming Biweight kernel function and $h = 0.1$. The estimate is given by

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}K(\frac{x - x(i)}{h}) ,$$

where $K(x) = \frac{15}{16}((1 - x^2)_+)^2$.

2. Consider a logistic distribution with the probability density function

$$f(x|\theta) = \frac{\exp(-(x - \theta))}{(1 + \exp(-(x - \theta)))^2}$$

and the distribution function:

$$F(x|\theta) = \frac{1}{1 + \exp(-(x - \theta))} .$$

The goal in this problem is to: (i) generate samples from this distribution and (ii) estimate the density function from the generated samples. Follow these steps:

(a) **Sampling**: For $\theta = 5$, generate $n = 100$ samples from this distribution using the inverse transform method.

(b) **Non-parametric Density Estimation**: Write a matlab program that uses these samples and a kernel based estimator to estimate the underlying density function. Plot the estimated density function for three different kernels: (a) triangular kernel, (b) biweight kernel, and (c) Gaussian kernel. For each case, use two different bandwidths, say $h = 0.5$ and $h = 1.0$. Compare each of these estimated densities with the true density function $f(x|5)$ by plotting it on the same graph.

3. In this problem the goal is to generate samples from a multivariate distribution and then use these sample to estimate the joint density function. Let X_1 and X_2 be two real values random variables that are jointly normal with a given mean and variance. For the vector $X = [X_1 \; X_2]^T$, the joint density is given by:

$$f(X) = \frac{1}{2\pi \det(K)} \exp(-\frac{1}{2}(X-\mu)^T) \; K^{-1}(X-\mu)) \; ,$$

where det stands for the determinant. Here $\mu \in I\!R^2$ is the vector of means and $K \in I\!R^{2\times 2}$ is the matrix of covariance.

   (a) **Sampling**: For $\mu = [2 \; 5]^T$ and

$$K = \begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix} \; ,$$

   generate $n = 100$ samples of the pairs (X_1, X_2). Let A be a matrix such that $A*A = K$ (you can use *sqrtm* command in matlab to compute A). Then, samples from the joint density are given by:

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \mu + A \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} \; ,$$

   where Z_1 and Z_2 are independent, standard normals (*randn* in matlab).

   (b) **Non-parametric Density Estimation**: Write a matlab program that uses sample pairs and a Gaussian kernel to non-parametrically estimate the joint density function. The kernel is given by:

$$K(x_1, x_2) = \frac{1}{2\pi} \exp(-(x_1^2 + x_2^2)/2) \; .$$

   Generate estimates for several values of $h = 0.5, 1.0, ...$

   (c) **Display**: Show mesh plots of both the true density and the estimated density functions.

Chapter 9

STATISTICAL PATTERN RECOGNITION

9.1 Introduction

1. Classification
2. Clustering

9.2 Classification

Consider the problem of classing an observation $X \in \mathbb{R}^n$ into one of the following classes: $C_1, C_2, \ldots, C_k$. One can view as the problem of (disjointly) partitioning the observation space $\mathbb{R}^n$ into k regions; if an observation falls into region i, we declare it to be of class C_i. Another viewpoint is that we assume there are k probability models- $f_1, f_2, \ldots, f_k$– associated with the different classes. The observation X is assumed to be a random sample from one of these probabilities, and the goal is to find f_i it came from. One can do this either in a likelihood framework or a Bayesian framework, as described later in this chapter. The problem of classification boils to designing the partitions or the probability models, from the past data, so that future data can be classified automatically.

9.2.1 Preliminaries

1. **Feature Vector**: Since n is too large in practice, it is practically impossible to analyze the observations directly. In most cases, one uses a mapping from $\mathbb{R}^n$ to $\mathbb{R}^d$ for a $d << n$ to map observations into a more manageable space. Let $g : \mathbb{R}^n \mapsto \mathbb{R}^d$ be such a map. For an observation $X \in \mathbb{R}^n$, the vector $x = g(X)$ is called a *feature vector* or a *representation* of X. The choice of g is not simple and has important bearing on the resulting pattern recognition. Once can broadly classify such dimension-reducing maps into two categories:

   (a) **Linear Dimension Reduction**: Let $U \in \mathbb{R}^{n \times d}$ be an orthogonal matrix, i.e. $U^T U = I_d$, and define $g(X) = U^T X$. This g is a linear projection to a subspace spanned by the columns of U matrix. An example of this linear projection is U obtained through principal component analysis (PCA), as described earlier in Chapter 2. More examples will be presented later.

(b) **Nonlinear Dimension Reduction**: In case the mapping g from $\mathbb{R}^n$ to $\mathbb{R}^d$ is not linear, it cannot be stated in the matrix form as earlier. An example of such a mapping is as follows. Form a histogram of the elements of X using d pre-determined, equally-spaced bins. The bins remain fixed irrespective of the values of X. Then, left x be the frequency vector associated with these bins, i.e. $x(1)$ is the number of elements in the first bin, $x(2)$ is the number of elements in the second bin, and so on. This define a mapping $x = g(X) = hist(X)$ that is a nonlinear mapping from $\mathbb{R}^n$ to $\mathbb{R}^d$.

Feature Space: The space of all feature vectors is called the *feature space*. This is the space in which a statistical analysis is performed to obtain pattern classification.

As described later, several methods utilize a metric on this space to perform classification. In case the feature space is Euclidean, $\mathbb{R}^d$, one can use the p-norm:

$$\|x_1 - x_2\|_p = \left(\sum_{l=1}^{d} (|x_1(l) - x_2(l)|)^p\right)^{1/p} .$$

$p = 2$ provides the usual Euclidean norm and is use most often in practice. This metric is also used in cases where PCA-based linear dimension reduction has been used to obtain the feature space. For $p = 1$, the resulting metric is also called *manhattan* or *city-block* metric.

In cases where the feature vectors are histograms, obtained from a nonlinear projection, the p-norm may not be very interesting. Since a histogram has an additional constraint that it always sums up to the number of observations, some more suitable metrics have been proposed. For example, one can use the normalized χ^2-statistic:

$$d(x_1, x_2) = \sum_{l=1}^{d} \frac{(x_1(l) - x_2(l))^2}{(x_1(l) + x_2(l))} ,$$

or the Kullback-Leibler divergence:

$$\sum_{l=1}^{d} x_1(l) \log(\frac{x_1(l)}{x_2(l)} \quad \text{or} \quad \sum_{l=1}^{d} x_2(l) \log(\frac{x_2(l)}{x_1(l)} ,$$

or their sum to make it symmetric. It should be kept in mind that these are not proper distances on the feature space, but still are useful because they provide a quantification of differences between any two histograms.

2. **Training Data**: We will assume that we have some number of observations available for each class. Let $X_i^j \in \mathbb{R}^n$ denote the j^{th} observation for i^{th} class, with $j = 1, 2, \ldots, n_i$ and $i = 1, 2, \ldots, k$. This data is mostly labelled, i.e. with each observation we are provided the class to which it belongs. The primary use of this data is to develop or derive classifiers, that can then be used in future classifications. Examples of use of training data in development of classifiers are given later in this chapter.

3. **Test Data**: Let $Y \in \mathbb{R}^n$ denote a new observation that we wish to classify. In general we are not given the underlying (true) class of Y, i.e. the test data is unlabelled. The test data serves as a set of future observations to which a classifier is applied and tested. This data is not used in the development of a classifier but is often used in evaluation of a classifier. One applies a classifier to the test data and measures the classification performance to evaluate that classifier.

4. **Validation Data**: In addition to the training and the test data, some methods use another data set called the *validation data*. This labelled set is used in cases where the development of a classifier is feedback-based, i.e. the classifier design is based on its performance on a dataset different from the training data. For instance, say one designs a classifier using the training data, evaluates its performance on the validation data, and learns of ways to improve the classifier. After improvement this process is repeated until no more improvement is possible.

9.2.2 Classification

In this section, we formally define the process of classification and provide simple examples.

Definition 29 *A classifier is a mapping from the feature space to the set of all classes. That is,*

$$\phi : \mathbb{R}^d \mapsto \{C_1, C_2, \ldots, C_k\} \ , \tag{9.1}$$

is a classifier. For any test observation Y, and its feature vector $y = g(Y)$, if $\phi(y) = C_i$, then Y is designated to the i^{th} class.

There is a large variety of classifiers that used in applications, but they can be divided into two broad categories:

1. **Metric-Based Classifiers**: These classifiers assume that there exists a metric in the feature space to quantify difference between any two elements of $\mathbb{R}^d$. Given such a metric, called it $d(\cdot, \cdot)$, one can define a number of classifiers.

   Examples of metric-based classifiers:

   - **Nearest Neighbor (NN) Classifier**: In this classifier, one computes the distance between $y = g(Y)$ and all possible training vectors $g(X_i^j)$, and assigns that class to Y that has the nearest element to y. That is,

     $$\hat{i} = \operatorname*{argmin}_{i=1}^{k} \left(\operatorname*{argmin}_{j} d(g(Y), g(X_i^j)) \right) \ .$$

     This is one of the simplest classifiers, although it has the disadvantage of being slow since it requires computing distances between the test data and all the training data. Another disadvantage is that it is vulnerable to noise. If the noise moves a training vector from the wrong class closest to y, then the classification will be wrong.

   - **k-Nearest Neighbor (kNN) Classifier**: In order to make the classifier more immune to observation noise, one uses the k-nearest neighbor classifier. Instead of the nearest training data, here one finds k nearest training data vectors, and uses the majority class amongst them to classify Y. In order to avoid ties, it is often useful to take k to be odd.

   Advantages and disadvantages: Makes no assumption about the probability model (implicit assumptions present sometimes). Can be slow if the training data size is large.

2. Probability Based

   Examples of probability-based classifiers:

- **Maximum-Likelihood Classifier**:
  Example: Assume a two class classification problem. For each class $f(x|C_i)$ is multivariate normal with mean μ_i and a fixed covariance Σ.
- **Bayesian Classifier**:

9.3 Boosting

9.4 Clustering

9.4.1 Hierarchical Clustering

9.4.2 Partitional Clustering

1. **Nearest-Neighbor Clustering**
2. **Mutual Neighborhood Clustering**

9.5 Kernel Trick

Chapter 10

ANALYSIS OF DYNAMIC SYSTEMS

In Chapter 6 we studied the problem of sampling from a probability distribution, using a homogeneous Markov with certain properties. Now we consider a similar problem of sampling with a difference that the probability distribution is changing in time. In other words, we need to sample from an indexed family of probability distributions, indexed by time. Assuming that these probability distribution evolve in time according to some specific transitions, then this task can be simplified greatly. Let $\{f_t\}$ be sequence of probability densities that we want to sample from, for each t, and let S_t be a collection of samples from f_t. A major simplification in the sampling process can be obtained if one can use samples S_t of f_t to obtain samples S_{t+1} of f_{t+1}, rather than sampling from the start. This is the topic of study in this chapter.

This topic relates closely to a a broader class of problems where one uses observations from a time-series to estimate the underlying process. Let $x_1, x_2, \ldots, x_t, x_{t+1}, \ldots,$ form a process of interest, and instead of measuring x_ts directly, one observes variables $y_1, y_2, \ldots, y_t, y_{t+1}, \ldots$. The goal is to use the observations, and joint probability models of x and y to estimate the unknown x_ts. Let $f(x_t|y_1, y_2, \ldots, y_\tau)$ be the posterior density function of x_t given a set of observations $y_1, \ldots, y_\tau)$. The estimation can be one of the following three kinds:

1. **Smoothing**: In this case the goal is to estimate a state x_t using observations up to a time $\tau > t$. In other words,
$$\hat{x}_t = \underset{x_t}{\text{argmax}}\, f(x_t|y_1, y_2, \ldots, y_t, \ldots, y_\tau) \ .$$

2. **Filtering**: In this case the goal is to estimate a state x_t using observations up to t. In other words,
$$\hat{x}_t = \underset{x_t}{\text{argmax}}\, f(x_t|y_1, y_2, \ldots, y_t) \ .$$

3. **Prediction**: In this case the goal is to estimate a state x_t using observations up to a time $\tau < t$. In other words,
$$\hat{x}_t = \underset{x_t}{\text{argmax}}\, f(x_t|y_1, y_2, \ldots, y_\tau) \ .$$

Some simplifying assumptions that are often made are: (i) the process x_t forms a Markov process, and (ii) for all t, the observation y_t is conditionally independent of x_τ, for $\tau \neq t$, given the state x_t. This set up is also known as a *hidden Markov model* (HMM) where $\{x_t\}$

is considered the hidden process and $\{y_t\}$ is used to discover or estimate it. This model has been found very useful in many applications including speech analysis, steering and robotics, engineering design. A particular case of HMM arises when the posterior density $f(x_t|y_1, \ldots, y_t)$ is Gaussian for all t. In that case, it is easy to summarize the posterior, since only mean and covariance at each time t are needed, and there is no need to generate samples. In this case, one focuses on the problem of estimating mean and covariance at time $t+1$, given these quantities at time t and the new observation y_{t+1}. An algorithm for performing this update was proposed by Robert Kalman and this algorithm forms one of the most important results in statistical time-series analysis.

10.1 Nonlinear Filtering Problem

Let the state vector x_t be an element of $\mathbb{R}^m$ and the observation vector y_t take value in $\mathbb{R}^n$. Assume that state vectors and observation vectors satisfy the following equations:

$$x_{t+1} = F(x_t) + \Gamma u_t \tag{10.1}$$

$$y_t = G(x_t) + w_t \tag{10.2}$$

Eqn. 10.1 is called the *state equation* and it models the evolution of state in time. Here, $F : \mathbb{R}^m \mapsto \mathbb{R}^m$ is a given function, Γ is a given $m \times m$ matrix and $u_t \in \mathbb{R}^m$ is a vector of noise or *perturbation* variables. Assuming that u_ts are statistically independent, the resulting process x_t is a Markov process. That is:

$$f(x_t|x_t, x_2, \ldots, x_{t-1}) = f(x_t|x_{t-1}) \ . \tag{10.3}$$

Eqn. 10.2 is called the *observation equation* and it relates the underlying state (x_t) with the observation y_t. Here, $G : \mathbb{R}^m \mapsto \mathbb{R}^n$ is a given function and $w_t \in \mathbb{R}^n$ is the vector of observation noise. It is assumed that the observation noise w_t is statistically independent across times, i.e. w_t is independent of w_s for $t \neq s$, and it is also independent of x_t and u_t. Therefore,

$$f(y_t|y_1, y_2, \ldots, y_{t-1}, x_t) = f(y_t|x_t) \ . \tag{10.4}$$

Combining these assumptions, one can write the full posterior of x_t, as

$$\begin{aligned}
f(x_t|y_1, y_2, \ldots, y_t) &= \frac{f(x_t, y_1, y_2, \ldots, y_t)}{f(y_1, y_2, \ldots, y_t)} \\
&= \frac{f(y_t|x_t, y_1, \ldots, y_{t-1}) f(x_t|y_1, \ldots, y_{t-1}) f(y_1, y_2, \ldots, y_{t-1})}{f(y_t|y_1, y_2, \ldots, y_{t-1}) f(y_1, y_2, \ldots, y_{t-1})} \\
&= \frac{f(y_t|x_t) f(x_t|y_1, \ldots, y_{t-1})}{f(y_t|y_1, y_2, \ldots, y_{t-1})}
\end{aligned}$$

Investigating the right term in the numerator, we obtain:

$$f(x_t|y_1, \ldots, y_{t-1}) = \int_{x_{t-1}} f(x_t, x_{t-1}|y_1, \ldots, y_{t-1}) dx_{t-1} = \int_{x_{t-1}} f(x_t|x_{t-1}) f(x_{t-1}|y_1, \ldots, y_{t-1}) dx_{t-1} \ .$$

This leads to the two equations of nonlinear filtering:

$$\boxed{\begin{aligned} f(x_t|y_1,\ldots,y_{t-1}) &= \int_{x_{t-1}} f(x_t|x_{t-1})f(x_{t-1}|y_1,\ldots,y_{t-1})dx_{t-1} && (10.5)\\ f(x_t|y_1,y_2,\ldots,y_t) &= \frac{f(y_t|x_t)f(x_t|y_1,\ldots,y_{t-1})}{f(y_t|y_1,y_2,\ldots,y_{t-1})} && (10.6)\end{aligned}}$$

The first equation, Eqn. 10.5, is called the *prediction equation*, since it uses the past data $y_1, y_2, \ldots, y_{t-1}$, to predict a future state x_t. Note that this equation involves two densities: (i) the posterior density of x_{t-1} given all the past data, and (ii) one step prediction of x_t given x_{t-1}. The data at that time, y_t, does not play a role in the prediction equation. This data enters into the second equation, called the *update equation*, that involves two densities: (i) the prediction posterior density $f(x_t|y_1, y_2, \ldots, y_{t-1})$ obtained from the prediction equation, and (ii) the likelihood function for data at time t, given by $f(y_t|x_t)$.

The problem of Bayesian nonlinear filtering is to solve for estimates from the posterior density at time t. There are several estimators that can be designed:

1. The maximum a-posterior (MAP) estimate of x_t given all the data upto time t. That is,
$$\hat{x}_t^{map} = \operatorname*{argmax}_{x_t} f(x_t|y_1,\ldots,y_t)\ . \tag{10.7}$$

2. The posterior mean estimator, or
$$\hat{x}_t^{\mu} = \int_{x_t} x_t f(x_t|y_1,\ldots,y_t)dx_t\ .$$

3. Other estimators, such as posterior median, are also useful in some situations.

In addition to the estimation of x_t, using an estimator $\hat{x}_t$, one is also interested in the quality of that estimator. This often results in the problem of estimating variance of x_t under the posterior density, $f(x_t|y_1,\ldots,y_t)$.

10.2 Kalman Filter

A special case arises when the functions $F(\cdot)$ and $G(\cdot)$ are linear maps, i.e. $F(x) = Ax$ for an $m \times m$ matrix A and $G(x) = Bx$ for an $n \times m$ matrix B. Additionally, if we assume that the vectors u_t and w_t are multivariate Gaussians, and the initial state x_0 is Gaussian, then the resulting posterior density $f(x_t|y_1,\ldots,y_t)$ is also Gaussian for all t. Therefore, one can specify this posterior using a mean vector and a covariance matrix, and there is no need to generate samples. This was realized by Kalman in his celebrated paper in [], resulting in a major tool in area of time series analysis, called the *Kalman filter*. Next, we derive the two basic equations that form a Kalman filter iteration.

Assuming the linearity of functions F and G, the state and observation equations are given by: for $t > 1$,

$$x_{t+1} = A_t x_t + \Gamma u_t, \quad u_t \sim N(0, Q) \tag{10.8}$$
$$y_t = B_t x_t + w_t, \quad w_t \sim N(0, \Lambda) \tag{10.9}$$

We will also assume that the initial state $x_0 \sim N(\mu_0, P_0)$ where $P_0 \in \mathbb{R}^{m\times m}$ covariance matrix. As a first we will show that $f(x_t|y_1,\ldots,y_t)$ is normal and we will specify its mean and variance, for all $t \geq 1$.

Theorem 19 *Assuming that: (i) $x_0 \sim N(\mu_0, P_0)$, (ii) x_t, y_t satisfy Eqns. 10.8-10.9, and (iii) u_t, w_t are independent as assumed earlier, we have that $f(x_{t+1}|y_1, \ldots, y_t, y_{t+1}) \equiv N(\mu_{t+1}, P_{t+1|t+1})$, where*

$$\begin{aligned}
\mu_{t+1} &= A_t\mu_t + P_{t+1|t+1}B_t^T\Lambda^{-1}(y_{t+1} - B_tA_t\mu_t) \\
P_{t+1|t+1} &= (P_{t+1|t}^{-1} + B_t^T\Lambda^{-1}B_t)^{-1}, \quad P_{t+1|t} = A_tP_{t|t}A_t^T + \Gamma Q\Gamma^T
\end{aligned}$$

Note that $P_{0|0} = P_0$.
Proof: We will prove this result by induction. Assume that it is true for some t, and prove it for $t + 1$. We start with $t = 1$.

1. Since there is no data at time $t = 0$, the posterior density at $f(x_0|y_0$ is same as the prior $f(x_0)$, which by assumption, is Gaussian with mean μ_0 and variance P_0.

2. Fix any $t > 0$, let the theorem hold for that t. That is, the posterior density $f(x_t|y_1, \ldots, y_t)$ is Gaussian with mean μ_t and covariance $P_{t|t}$. That is,

$$x_t|(y_1, \ldots, y_t) \sim N(\mu_t, P_{t|t}) \ . \tag{10.10}$$

We will prove this result for $t + 1$. As a first step, consider the probably density of x_{t+1} given $(y_1, \ldots, y_t)$. Using Eqns. 10.8 and 10.10, the conditional probability density of x_{t+1}, given $(y_1, y_2, \ldots, y_t)$, is

$$x_{t+1}|(y_1 \ldots, y_t) \sim N(A\mu_t, A_tP_{t|t}A_t^T + \Gamma Q\Gamma^t) \ .$$

For later convenience, let $M_t = A_tP_{t|t}A_t^T + \Gamma Q\Gamma^t$; $M_t \in \mathbb{R}^{m\times m}$ is a covariance matrix. Using Eqn. 10.9, the conditional probably density of y_{t+1} given x_{t+1} is:

$$y_{t+1}|x_{t+1} \sim N(Bx_{t+1}, \Lambda) \ .$$

Our goal is to find the posterior density $f(x_{t+1}|y_1, \ldots, y_{t+1})$, we do so using the Bayes rule,

$$\begin{aligned}
f(x_{t+1}|y_1, \ldots, y_{t+1}) &= \frac{f(y_{t+1}|x_{t+1})f(x_{t+1}|y_1, \ldots, y_t)}{f(y_{t+1}|y_1, \ldots, y_{t+1})} \\
&\propto \frac{1}{(\sqrt{2\pi\det(\Lambda)})^n}\exp(-\frac{1}{2}(y_{t+1} - B_{t+1}x_{t+1})^T\Lambda^{-1}(y_{t+1} - B_{t+1}x_{t+1})) \\
&\quad \frac{1}{(\sqrt{2\pi\det(M_t)})^n}\exp(-\frac{1}{2}(x_{t+1} - A_t\mu_t)^TM_t^{-1}(x_{t+1} - A_t\mu_t))
\end{aligned}$$

Collecting the terms in the exponent, we have:

$$\begin{aligned}
&-\frac{1}{2}(y_{t+1}\Lambda^{-1}y_{t+1} - x_{t+1}^TB^T\lambda^{-1}y_{t+1} - y_{t+1}^T\Lambda^{-1}Bx_{t+1} + x_{t+1}^TB^T\Lambda^{-1}Bx_{t+1} \\
&x_{t+1}^TM_t^{-1}x_{t+1} - \mu_t^TA_t^TM_t^{-1}x_{t+1} - x_{t+1}^TM_t^{-1}A\mu_t + \mu_t^TA_t^TM_t^{-1}A_t\mu_t) \ .
\end{aligned}$$

Combining the terms, we obtain,

$$\begin{aligned}
&-\frac{1}{2}(x_{t+1}^T(B_t^T\Lambda^{-1}B_t + M_t^{-1})x_{t+1} - x_{t+1}^T(B_t^T\Lambda^{-1}y_{t+1} \\
&+M_t^{-1}A_t\mu_t) - (y_{t+1}^T\Lambda^{-1}B_t + \mu_t^TA_t^TM_t^{-1})x_{t+1} + y_{t+1}^T\Lambda^{-1}y_{t+1} + \mu_t^TA_t^TM_t^{-1}A_t\mu_t)
\end{aligned}$$

Set $P^{-1}_{t+1|t+1} = M_t^{-1} + B_t^T\Lambda^{-1}B_t$ and $M_t^{-1} = P^{-1}_{t+1|t+1} - B_t^T\Lambda^{-1}B_t$. The exponent changes to,

$$\begin{aligned}
&= -\frac{1}{2}(x^T_{t+1}P^{-1}_{t+1|t+1}x_{t+1} - x^T_{t+1}(B_t^T\Lambda^{-1}y_{t+1} + (P^{-1}_{t+1|t+1} - B_t^T\Lambda^{-1}B_t)A_t\mu_t) \\
&\quad -y^T_{t+1}\Lambda^{-1}B_t + \mu_t^T A_t^T(P^{-1}_{t+1|t+1} - B_t^T\Lambda^{-1}B_t)x_{t+1} + y^T_{t+1}\Lambda^{-1}y_{t+1} \\
&\quad +\mu_t^T A_t^T(P^{-1}_{t+1|t+1} - B_t^T\Lambda^{-1}B_t)A_t\mu_t) \\
&= -\frac{1}{2}(x^T_{t+1}P^{-1}_{t+1|t+1}x_{t+1} - x^T_{t+1}(B_t^T\Lambda^{-1}(y_{t+1} - B_tA_t\mu_t)) - x^T_{t+1}P^{-1}_{t+1|t+1}A_t\mu_t \\
&\quad -((y^T_{t+1} - \mu_t^T A_t^T B_t^T)\Lambda^{-1}B_t)x_{t+1} - \mu_t^T A_t^T P^{-1}_{t+1|t+1}x_{t+1} + y^T_{t+1}\Lambda^{-1}y_{t+1} \\
&\quad +\mu_t^T A_t^T(P^{-1}_{t+1|t+1} - B_t^T\Lambda^{-1}B_t)A_t\mu_t) \\
&= -\frac{1}{2}(x^T_{t+1}P^{-1}_{t+1|t+1}x_{t+1} - x^T_{t+1}P^{-1}_{t+1|t+1}(A_t\mu_t + P_{t+1|t+1}B_t^T\Lambda^{-1}(y_{t+1} - B_tA_t\mu_t)) \\
&\quad -(\mu_t^T A_t^T + (y^T_{t+1} - \mu_t A_t^T B_t^T)\Lambda^{-1}B_tP_{t+1|t+1})P^{-1}_{t+1|t+1}\mu^T_{t+1} \\
&\quad +y^T_{t+1}\Lambda^{-1}y_{t+1} + \mu_t^T A_t^T(P^{-1}_{t+1|t+1} - B_t^T\Lambda^{-1}B_t)A_t\mu_t) \\
&= -\frac{1}{2}((x_{t+1} - (A_t\mu_t + P_{t+1|t+1}B_t^T\Lambda^{-1}(y_{t+1} - B_tA_t\mu_t)))^T P^{-1}_{t+1|t+1} \\
&\quad (x_{t+1} - (A_t\mu_t + P_{t+1|t+1}B_t^T\Lambda^{-1}(y_{t+1} - B_tA_t\mu_t))))
\end{aligned}$$

Since this is a perfect quadratic form, the posterior density at $t+1$ is Gaussian with mean

$$\mu_{t+1} = A_t\mu_t + P_{t+1|t+1}B_t^T\Lambda^{-1}(y_{t+1} - B_tA_t\mu_t)\ , \tag{10.11}$$

and covariance $P_{t+1|t+1} = (M_t^{-1} + B_t\Lambda^{-1}B_t)^{-1}$.

This leads to a preliminary formulation of the Kalman filter for updating mean and covariance from time t to $t+1$:

Algorithm 38 (Kalman Filter – Classical) *At time t we have the mean μ_t and the covariance $P_{t|t}$.*

1. **Prediction Step**: *The first equation of nonlinear filtering leads to prediction of these parameters:*

$$\begin{aligned}
\mu_{t+1|t} &= A_t\mu_t & (10.12)\\
P_{t+1|t} &\equiv M_t = A_tP_{t|t}A_t^T + \Gamma Q\Gamma & (10.13)
\end{aligned}$$

Note that this step does not involve the data y_{t+1} yet. It uses past data to predict the mean and the covariance at time $t+1$.

2. **Correction Step**: *This step uses y_{t+1} to update the predicted values of mean and covariance:*

$$\begin{aligned}
\mu_{t+1|t+1} &= \mu_{t+1|t} + P_{t+1|t+1}B_t^T\Lambda^{-1}(y_{t+1} - B_t\mu_{t+1|t}) & (10.14)\\
P_{t+1|t+1} &= (P^{-1}_{t+1|t} + B_t^T\Lambda^{-1}B_t)^{-1} & (10.15)
\end{aligned}$$

In situations where the observation space has smaller dimension that the state space, i.e. $n < m$, it is possible to simplify this calculation using the Sherman-Morrison-Woodbury formula:

$$(A^{-1} + VC^{-1}V^T)^{-1} = A - \left(AV(C + V^T AV)^{-1}V^T A\right) \ . \tag{10.16}$$

This formula provides the inverse of an $m \times m$ matrix (A^{-1}) with a rank n update ($VC^{-1}V^T$).

Define a matrix $C = P_{t+1|t} - P_{t+1|t+1}$. Using Sherman-Morrison-Woodbury formula, it is given by:

$$\begin{aligned} C &= P_{t+1|t} - (P_{t+1|t}^{-1} + B_t^T \Lambda^{-1} B_t)^{-1} \\ &= P_{t+1|t} - P_{t+1|t} + \left(P_{t+1|t} B_t^T (\Lambda + B_t P_{t+1|t} B_t^T)^{-1} B_t P_{t+1|t}\right) \\ &= P_{t+1|t} B_t^T (\Lambda + B_t P_{t+1|t} B_t^T)^{-1} B_t P_{t+1|t} \end{aligned}$$

Seeking to simplify the expression $P_{t+1|t+1} B_t^T \Lambda^{-1}$, we copmpute:

$$\begin{aligned} P_{t+1|t+1} B_t^T \Lambda^{-1} &= ((P_{t+1|t} - C) B_t^T \Lambda^{-1} \\ &= ((P_{t+1|t} - P_{t+1|t} B_t^T (\Lambda + B_t P_{t+1|t} B_t^T)^{-1} B_t P_{t+1|t}) B_t^T \Lambda^{-1} \\ &= P_{t+1|t} B_t^T (I - (\Lambda + B_t P_{t+1|t} B_t^T)^{-1} B_t P_{t+1|t} B_t^T) \Lambda^{-1} \\ &= P_{t+1|t} B_t^T (\Lambda + B_t P_{t+1|t} B_t^T)^{-1} (\Lambda + B_t P_{t+1|t} B_t^T + B_t P_{t+1|t} B_t^T) \Lambda^{-1} \\ &= P_{t+1|t} B_t^T (\Lambda + B_t P_{t+1|t} B_t^T)^{-1} \ . \end{aligned}$$

With this simplification, the Kalman filter can be modified as:

Algorithm 39 (Kalman Filter) *At time t we have the mean μ_t and the covariance $P_{t|t}$.*

1. **Prediction Step**: *The first equation of nonlinear filtering leads to prediction of these parameters:*

$$\begin{aligned} \mu_{t+1|t} &= A_t \mu_t & (10.17) \\ P_{t+1|t} &\equiv M_t = A_t P_{t|t} A_t^T + \Gamma Q \Gamma & (10.18) \end{aligned}$$

2. **Correction Step**: *This step uses y_{t+1} to update the predicted values of mean and covariance:*

$$\begin{aligned} \mu_{t+1|t+1} &= \mu_{t+1|t} + P_{t+1|t} B_t^T (\Lambda + B_t P_{t+1|t} B_t^T)^{-1} (y_{t+1} - B_t \mu_{t+1|t}) & (10.19) \\ P_{t+1|t+1} &= P_{t+1|t} - P_{t+1|t} B_t^T (\Lambda + B_t P_{t+1|t} B_t^T)^{-1} B_t P_{t+1|t} & (10.20) \end{aligned}$$

What is the advantage of using the modified update step in this version of Kalman filter. In the earlier version, to compute $P_{t+1|t+1}$ we need to invert two $m \times m$ matrices in $(P_{t+1|t}^{-1} + B_t^T \Lambda^{-1} B_t)^{-1}$. In the modified version, we need to compute only one matrix inverse $(\Lambda + B_t P_{t+1|t} B_t^T)^{-1}$ involving an $n \times n$ matrix. Since computing inverse is both computationally expensive and error-prone operation, the modified version (assuming $n < m$) is computationally preferable.

10.3 Extended Kalman Filters

In some situations it is possible to approximate the state and the observation equations in such a way that they result in a linear Gaussian system. For example, if in Eqns. 10.5-10.6, the additive noise u_t and v_t are Gaussian, and if it is possible to approximate well the functions $F(\cdot)$ and $G(\cdot)$ by their linear approximations, then one can apply Kalman filter to this approximation. Such a situation is termed as an extended Kalman filter.

In general, however, when linear approximations of state and observation equations are not valid, or when the additive noise u_t, v_t are not Gaussian, one has to address the non-Gaussianity of the resulting posterior density directly. The next section describes a Monte Carlo approach to filtering in situations where the posterior density is non-Gaussian.

10.4 Sequential Monte Carlo Methods

In many problems it is not possible to assume linearity of system evolution and observation maps, and the linearization does not yield good results. Such problems require tools to handle nonlinearity of the underlying systems. An important consequence of nonlinear models is that the posterior density at any time may not be Gaussian, and hence its representation using a mean and a covariance is no longer sufficient. In such cases one resorts to Monte Carlo idea and generates a large number of samples from the posterior, at every time t, and uses those samples to estimate system state, the estimation error, or any other parameter of interest. Since the posterior changes at every observation time, one would have to generate samples at every time. However, using the relationship between evolving posteriors, one seeks an efficient algorithm for generating samples from posterior at time $t+1$, using samples from the posterior at time t, in a recursive fashion. In this section, we describe the method of sequential Monte Carlo to accomplish this task.

Let $f(x_t|y_1, \ldots, y_t)$ be the posterior density function at time t, and let x_t^i be a sample from this posterior. Define the sample set

$$S_t = \{x_t^i,\ i = 1, 2, \ldots, n\} \ .$$

Our goal is to use elements of S_t, and the new observation y_{t+1} and the system-observation models, to generate the sample set S_{t+1}. Using the nonlinear filtering equations (Eqns. 10.5-10.6), this task is performed in two steps (analogous to Kalman Filtering). The first step generates prediction and the second step updates, or corrects, these predictions using the data y_{y+1}.

1. **Prediction Step**: This step uses samples from the posterior density at time t to generate samples from the prediction density $f(x_{t+1}|y_1, \ldots, y_t)$. The basic idea is to interpret Eqn. 10.5, reproduced here, as a mixture density.

   $$f(x_{t+1}|y_1, \ldots, y_t) = \int_{x_t} f(x_{t+1}|x_t) f(x_t|y_1, \ldots, y_t) dx_t$$

   Since x_t^i is a sample from $f(x_t|y_1, \ldots, y_t)$, one can use a sample from the conditional density $f(x_{t+1}|x_t^i)$. As described in 5, the value $\tilde{x}_t^i \sim f(x_{t+1}|x_t^i)$ can be shown to be a sample from $f(x_{t+1}|y_1, \ldots, y_t)$. We will call the set

   $$\tilde{S}_{t+1} = \{\tilde{x}_t^1, \tilde{x}_t^2, \ldots, \tilde{x}_t^n\} \ ,$$

as the *prediction set*. If needed, one can estimate the mean, covariance and other moments, under the prediction density, using sample averages over this prediction set. For example,

$$\hat{\mu}_{t+1|t} = \frac{1}{n}\sum_{i=1}^{n} \tilde{x}_t^i \ .$$

Note that this step does not use the new data y_{t+1}, and success of this step depends upon our ability to sample from one-step-prior $f(x_{t+1}|x_t^i)$. As stated in Eqn. 10.1, this one-step-prior comes from a state equation involving an update function $F(\cdot)$ and a random term Γu_t. In this situation, sampling from this prior is straightforward, according to:

$$\tilde{x}_{t+1}^i = F(x_t^i) + \Gamma u_t^i \ ,$$

where u_t^i is a random sample from its given density function. In examples where u_t is multivariate normal this simulation is rather simple.

2. **Update Step**: This step uses Eqn. 10.6 and the prediction set $\tilde{S}_{t+1}$ to generate samples from the posterior density $f(x_{t+1}|y_1,\ldots,y_{t+1})$. In addition, we are interested in estimating a parameter

$$\theta_{t+1} = \int g(x_{t+1}) f(x_{t+1}|y_1,\ldots,y_{t+1}) dx_{t+1} \ ,$$

for a given function g. In case of minimum mean square estimation, the function is g is identity and we are interested in estimating the posterior mean as an estimate of the state x_{t+1}. However, under other estimation criteria, the function g can be different but is always known beforehand. There two goals are accomplished using **important sampling** and **resampling** as described next.

First, consider the goal of estimating θ_{t+1}. Using ideas from importance sampling we can rewrite the definition of θ_{t+1} as follows:

$$\begin{aligned}
\theta_{t+1} &= \int g(x_{t+1}) f(x_{t+1}|y_1,\ldots,y_{t+1}) dx_{t+1} \\
&= \int g(x_{t+1}) \frac{f(y_{t+1}|x_{t+1}) f(x_{t+1}|y_1,\ldots,y_t)}{f(y_{t+1}|y_1,y_2,\ldots,y_t)} dx_{t+1} \\
&= \int \left(\frac{g(x_{t+1}) f(y_{t+1}|x_{t+1})}{f(y_{t+1}|y_1,y_2,\ldots,y_t)} \right) f(x_{t+1}|y_1,\ldots,y_t) dx_{t+1} \qquad (10.21)
\end{aligned}$$

The second term comes directly from applying Eqn. 10.6 to this situation. Now that we have samples from $f(x_{t+1}|y_1,\ldots,y_t)$, namely the elements of the set $\tilde{S}_{t+1}$, we can use them to estimate θ_{t+1}. This requires evaluating the expression inside $(\cdot)$ on these samples and taking an average over n samples. However, notice that this expression involves the quantity $f(y_{t+1}|y_1,y_2,\ldots,y_t)$ that is unknown. So, we have to estimate this quantity also, in order to estimate θ_{t+1} using importance sampling. This is done by considering:

$$f(y_{t+1}|y_1,y_2,\ldots,y_t) = \int f(y_{t+1}|x_{t+1}) f(x_{t+1}|y_1,\ldots,y_t) dx_{t+1} \ .$$

Once again, since we have samples from $f(x_{t+1}|y_1,\ldots,y_t)$, we can use them, and the likelihood function $f(y_{t+1}|x_{t+1})$ to estimate the left side. Define the weights

$$w_{t+1}^i = f(y_{t+1}|\tilde{x}_{t+1}^i) \ ,$$

and estimate $f(y_{t+1}|y_1, y_2, \ldots, y_t)$ using $\frac{1}{n}\sum_{i=1}^{n} w_{t+1}^i$. This is nothing but the classical Monte Carlo idea. Now that we have an estimate for $f(y_{t+1}|y_1, y_2, \ldots, y_t)$, we return to the estimation of θ_t using Eqn. 10.21, according to:

$$\hat{\theta}_{t+1} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{g(\tilde{x}_{t+1}^i)w_{t+1}^i}{\frac{1}{n}\sum_{i=1}^{n} w_{t+1}^i}\right) .$$

Define the normalized weights $\tilde{w}_{t+1}^i = \frac{w_{t+1}^i}{\sum_{j=1}^{n} w_{t+1}^j}$, we can restate the estimator of θ_{t+1} as:

$$\hat{\theta}_{t+1} = \sum_{i=1}^{n} g(\tilde{x}_{t+1}^i)\tilde{w}_{t+1}^i . \tag{10.22}$$

Next, we return to the task of generating samples from the posterior $f(x_{t+1}|y_1, \ldots, y_{t+1})$. It can be shown that if we resample from the set $\tilde{S}_{t+1}$ with probabilities given by

$$\{\tilde{w}_{t+1}^1, \tilde{w}_{t+1}^2, \ldots, \tilde{w}_{t+1}^n\} ,$$

then, the resulting values are samples from the posterior. Generating n such resamples, and calling them x_{t+1}^i, we obtain the sample set

$$S_{t+1} = \{x_{t+1}^1, x_{t+1}^2, \ldots, x_{t+1}^n\} .$$

This is the basic form of a sequential Monte Carlo algorithm.

Algorithm 40 (Classical SMC) *1. Generate n samples $x_0^i \sim f(x_0)$. Set t=0.*

2. **Prediction***: Generate the prediction set using: $\tilde{x}_{t+1}^i \sim f(x_{t+1}|x_t^i)$, $i = 1, 2, \ldots, n$.*

3. **Update***: Compute the weights $w_{t+1}^i = f(y_{t+1}|\tilde{x}_{t+1})$, and normalize them using $\tilde{w}_{t+1}^i = \frac{w_{t+1}^i}{\sum_{j=1}^{n} w_{t+1}^j}$.*

   (a) Estimate θ_{t+1} using $\hat{\theta}_{t+1} = \sum_{i=1}^{n} g(\tilde{x}_{t+1}^i)\tilde{w}_{t+1}^i$.

   (b) Resample from the set $\{\tilde{x}_{t+1}^1, \tilde{x}_{t+1}^2, \ldots, \tilde{x}_{t+1}^n\}$ with probabilities $\{\tilde{w}_{t+1}^1, \tilde{w}_{t+1}^2, \ldots, \tilde{w}_{t+1}^n\}$, n times to obtain the samples x_{t+1}^i, $i = 1, 2, \ldots, n$.

4. Set $t = t + 1$, and return to Step 2.

In some cases it is possible that resampling leads to only a small number of distinct values in the sample set S_{t+1}. Even though each one of these values is a sample from the posterior $f(x_{t+1}|y_1, \ldots, y_{t+1})$, lack of variety in the elements of S_{t+1} may lead to a bad estimator of θ_{t+1}. To avoid this situation, one can use a few steps of MCMC algorithms, such a M-H algorithm, starting from x_{t+1}^i, to generate a better variety of values. In that case, just replace x_{t+1}^i by the new value resulting from a few steps of M-H algorithm.

10.5 Problems

1. Write a matlab program to simulate two processes, $\{x_t\}$ and $\{y_t\}$ defined as follows:

$$\begin{aligned} x_{t+1} &= Ax_t + \Gamma u_t \quad \in I\!R^3 \\ y_{t+1} &= Bx_{t+1} + w_t \quad I\!R^2 \ , \end{aligned}$$

where

$$A = \begin{bmatrix} 1 & 0.1 & 0.2 \\ 0.3 & 1 & 0.1 \\ 0.2 & 0.1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0.5 & 0.2 \\ 0.5 & 1 & 0.1 \end{bmatrix}, \quad \Gamma = \begin{bmatrix} 0.1 & 0.0 & 0.0 \\ 0.0 & 0.1 & 0.0 \\ 0.0 & 0.0 & 0.1 \end{bmatrix}.$$

Also, u_t and w_t are vectors of Gaussian random variables with mean zeros and variances I_3 and $0.2I_2$, respectively. Use $x_0 = rand(3,1)$ as the initial condition. Show the plots of $\|x_t - x_1\|$ and $\|y_t - y_1\|$ versus t for three simulated processes.

2. **Kalman Filtering**: Consider the dynamic system specified in the last home work Problem 2.
   (a) Using the program from the last homework, generate a sequence of $\{x_t\}$ and $\{y_t\}$ for $t = 1, \ldots, 10$.
   (b) Write a matlab program to implement the Kalman filter for estimating the mean and the covariance under the posterior $f(x_t|y_1, \ldots, y_t)$. Use $\{y_t\}$ from part 1 to compute the estimates. Plot the estimation error $\|x_t - \hat{x}_{t|t}\|$ versus t.

3. **Nonlinear Systems**: Consider the nonlinear dynamic system described by the equations:

$$\begin{aligned} x_t &= x_2^2 + u_t \\ y_t &= \sqrt{x_t} + v_t \end{aligned}$$

where $x_0 \sim N(0,1)$, $u_t \sim N(0,1)$, and $v_t \sim N(0,0.1)$.
   (a) Write a matlab program to simulate the system and the observation process for $t = 1, \ldots, 20$.
   (b) Write a matlab program to implement sequential Monte Carlo algorithm for $n = 1000$.
   (c) Plot the histograms of the samples from the posterior at times $t = 1, 5, 10, 15, 20$.

4. **Sequential Monte Carlo Technique**: Consider the nonlinear dynamic system described by the equations:

$$\begin{aligned} x_t &= \frac{1}{2}x_{t-1} + \frac{25x_{t-1}}{1 + x_{t-1}^2} + 8\cos(1.2t) + u_t \\ y_t &= \frac{x_t^2}{20} + v_t \end{aligned}$$

where $x_0 \sim N(0,10)$, $u_t \sim N(0,10)$, and $v_t \sim N(0,1)$.
   (a) Write a matlab program to simulate the system and the observation process for $t = 1, \ldots, 20$.
   (b) Write a matlab program to implement sequential Monte Carlo algorithm for $n = 1000$.
   (c) Plot the histograms of the samples from the posterior at times $t = 1, 5, 10, 15, 20$.
   (d) Estimate the posterior means for all t and plot the error $\|x_t - \hat{x}_{t|t}\|$ versus t.

10.6 References

Chapter 11

BOOTSTRAP METHODS

11.1 Bootstrapping Methods

Very often in statistical analysis, involving an estimate of an unknown quantity, one is interested in determining the quality of this estimator. Since an estimator is a random quantity, due to the fact that data is considered random and an estimator is most often a function of the data, it is difficult to specify a single quantity that can completely measure the estimator quality. However, a number of quantities, including standard deviation of the estimator, also called the *standard error*, or its bias, or other moments, can all be used towards this purpose. For some estimators, mostly very simple, it is possible to analytically compute their standard deviations and biases. In the remaining cases, one looks a computational tool to estimate this quantities.

Definition 30 *Bootstrap is a technique that provides a general purpose tool for estimating standard errors and biases of given estimators, and for assigning measures of accuracy to those estimators.*

Once the technique is described, it becomes clear the central role that sampling and computation plays in bootstrap methodology. To motivate the basic idea behind bootstrap, we start with a simple example from [1]. In this example we take a sample mean estimator to estimate population mean. Sixteen mice were randomly selected for a study: seven assigned to the treatment group and the remaining nice to the control group. Shown below are their survival times, in days, following a test surgery. In order to estimate the mean survival time for the treatment population, we compute the sample mean:

$$\bar{x} = \sum_i x_i/7 = 86.86 \ .$$

Group	Symbol	Data		
Treatment	x_i	94	197	16
		38	99	141
		23		
Control	y_i	52	104	146
		10	51	30
		40	27	46

The question is: How good is this estimator? Once can compute the bias of this estimator, and that turns out to be zero. Furthermore, one can compute its standard error using the formula:

$$\sqrt{\frac{s^2}{7}} \text{ where } s^2 = \frac{1}{7-1}\sum_i (x_i - \bar{x})^2 \ .$$

This quantity turns out to be 25.24. Similarly, for the control class the standard error is 14.14. In case one is interested in overall standard error, it is $\sqrt{(25.24)^2 + (14.14)^2} = 28.93$. This standard error may, for example, be used in deciding if the means associated with treatment and control classes are equal. The difference between sample means is 30.63 and the overall standard error is 28.93 giving the ratio 30.63/28.93 = 1.05. Since this statistic is not significant, one can reject the hypothesis that the two population means are equal. In situation where the standard error of an estimator is readily available, such hypothesis testing is straightforward. However, in situations where we do not have analytical expressions for the standard errors (of the estimators), we need to start estimating them from the data. As an example, let us consider the question: Are the two population medians equal to each other? One can estimate these medians using sample statistics, obtaining values 94 and 46, respectively, resulting in a difference 48. Is this difference statistically significant? To answer this question, one needs standard error of this estimator which is not available directly. We introduce the use of bootstrap methods to estimate this standard error and other quantities that can be used to evaluate a given estimator.

11.1.1 Bootstrap Procedure

Let $x = (x_1, x_2, \ldots, x_n)$ be observed samples from an underlying probability density function $f(x)$, or a probability distribution function $F(x) = \int_{-\infty}^{x} f(y)dy$, and we are interested in studying an estimator of θ, given by:

$$\hat{\theta} = s(x) = s(x_1, x_2, \ldots, x_n) \tag{11.1}$$

As the first example, we focus on estimating the standard error of $\hat{\theta}$. We introduce the following definitions:

Definition 31 (Bootstrap Sample) *Let $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ be a random vector of quantities drawn from the set x uniformly with replacement. This vector x^* is called a* **bootstrap sample**.

Elements of x^* are taken from the elements of x but can appear multiple times in x^*.

Definition 32 (Bootstrap Replicate) *The evaluation of estimator s on a bootstrap sample x^* is called a* **bootstrap replicate** *of $\hat{\theta}$. It is denoted by $\hat{\theta}^* = s(x^*)$.*

Let X^{*1}, x^{*2}, .., and x^{*B} be B independent bootstrap samples of x, and let $\theta^{*b} = s(x^{*b})$, $b = 1, 2, \ldots, B$, be the resulting bootstrap replicates of $\hat{\theta}$. Then, the bootstrap estimate of the standard error of $\hat{\theta}$ is the standard deviation of the bootstrap replicates of $\hat{\theta}$:

$$\hat{se}_B = \frac{1}{B-1}\sum_{b=1}^{B}(\theta^{*b} - \bar{\theta}^*)^2, \text{ where } \bar{\theta}^* = \frac{1}{B}\sum_{b=1}^{B}\theta^{*b} \tag{11.2}$$

This expression provides a numerical technique for estimating standard errors. In case where analytical solutions are known, the two solutions coincide as expected. For example, if $s(x) = \frac{1}{n}\sum_{i=1}^{n} x_i$, the sample mean estimator, it can be shown that as $B \to \infty$, the estimate:

$$\hat{se}_B \to \{\frac{1}{n^2}\sum_{i=1}^{n}(x_i - s(x))^2\}^{0.5} \ .$$

If the factor $\frac{1}{n^2}$ in the front is replaced by $\frac{1}{n(n-1)}$, then this limit will be identical to the standard error of $s(x)$ given by the analytical expression. Returning to the mouse data (treatment group), the bootstrap estimate of standard errors are given by the table:

B	50	100	250	500	1000	∞
Mean	19.72	23.63	22.32	23.79	23.02	23.36
Median	32.21	36.35	34.46	36.72	36.48	37.83

This method provides an ability to compute standard error associated with estimators like median for which we do not have analytical answers. The estimated standard errors for median estimator are: (i) treatment group 37.83 and (ii) control group (). Therefore, one can compute the standard error of the estimator: $\hat{x}_{med}^{treat}$ - $\hat{x}_{med}^{control}$, and decide if this difference is statistically significant.

11.2 Empirical Distribution and Plug-In Principal

In order to analyze results of bootstrap estimator, we introduce the concept of empirical distribution. We assume that we are studying samples of a random variable X with a probability density function $f(x)$ or with a cumulative distribution function $F(x) = \int_{-\infty}^{x} f(y)dy$. In this section we will denote the random behavior of X by the distribution function F, rather than the common technique of using the density function f.

Let $x_1, x_2, \ldots, x_n$ be independent samples from F.

Definition 33 (Empirical Distribution) *For given observations $x_1, \ldots, x_n$ of a random variable X, its empirical probability distribution function is a discrete distribution that puts the probability $\frac{1}{n}$ on each x_i. It is denoted by $\hat{F}$.*

Using Dirac delta function, one can write $\hat{F}$ as:

$$\hat{F}(dx) = \frac{1}{n}\sum_{i=1}^{n} \delta(x - x_i)dx \tag{11.3}$$

For any set A, the empirical probability of A is:

$$\hat{F}(A) = \int_A \hat{F}dx = \frac{1}{n}(\text{number of } x_i\text{s in } A) \ .$$

Definition 34 (Plug-In Principle) *This is a simple method for estimating parameters of X, or F, where the estimator is given by simply replacing F by $\hat{F}$ in the definition of that estimator. Let $\theta = t(F)$ is the parameter of interest where t is an appropriate functional, then the plug-in estimator of θ is given by $\hat{\theta} = t(\hat{F})$.*

As an example, if we are interested in population mean $\mu = \int xF(dx)$, then its plug-in estimator is given by:

$$\hat{\mu} = \int x\hat{F}(dx) = \frac{1}{n}\int x(\sum_{i=1}^{n} \delta(x - x_i))dx = \frac{1}{n}\sum_{i=1}^{n} x_i \ .$$

If all the information about X comes in form of samples $x_1, \ldots, x_n$, then the plug-in estimator is often the best estimator of θ. For an exception, consider the plug-in estimator of the variance of a population. In case some additional information is available, one can perhaps improve upon the plug-in estimator. Empirical distribution and plug-in estimator play an important role in analysis of bootstrap procedures, as described later.

11.3 Estimation of Standard Error

If F is assumed to be from a parametric family, the resulting bootstrap procedure is called *parametric bootstrap*; otherwise, it is called *non-parametric* bootstrap. We will start with a discussion of non-parametric bootstrap and will return to the parametric procedure later.

11.3.1 Non-Parametric Estimation

As earlier, let x_1, x_2, .., x_n are samples of a random variable X with an unknown probability distribution function F, and we are analyzing an estimator of the parameter $\theta = t(F)$, where t is a known functional. Let the estimator be given by:

$$\hat{\theta} = s(x_1, x_2, \ldots, x_n) \ .$$

Here s may be the plug-in estimator $t(\hat{F})$ or something else. We are studying the question: How accurate is $\hat{\theta}$. One can use one or more of the quantities – bias, standard error, confidence intervals – associated with $\hat{\theta}$, to answer this question. We will focus on the standard error first, returning to bias and confidence interval estimation later. For a given observation $x = (x_1, x_2, \ldots, x_n)$, let $\hat{F}$ be the empirical distribution. Then, elements of a bootstrap sample x^* can be considered as independent samples from $\hat{F}$, $x_i^* \sim \hat{F}$. A bootstrap replicate of $\hat{\theta}$ is given by $\hat{\theta}^* = s(x^*)$.

We are interested in standard error of $\hat{\theta}$, denoted by $se_F(\hat{\theta})$ to emphasize the source of randomness in $\hat{\theta}$ is F, and the bootstrap estimate is given by $se_{\hat{F}}(\hat{\theta}^*)$. The last term denotes the standard deviation of $\hat{\theta^*}$ with respect to the distribution $\hat{F}$. This is called the ideal bootstrap estimate of the standard error of $\hat{\theta}$. In some cases one can use a direct formula to compute $se_{\hat{F}}(\hat{\theta}^*)$; in most cases it is further estimated as follows:

Algorithm 41 (Bootstrap Estimation of Standard Error) *1. Select B independent bootstrap samples $x^{*1}, x^{*2}, \ldots, x^{*B}$, each consisting of n values drawn with replacement from $x = (x_1, x_2, \ldots, x_n)$.*

2. Evaluate the bootstrap replication for each bootstrap sample using $\hat{\theta}^{*b} = s(x^{*b})$, $b = 1, 2, \ldots, B$.

3. Compute the standard deviation of B replicates of the estimator using:

$$\hat{se}_B = \left(\frac{1}{B-1} \sum_{b=1}^{B-1} (\hat{\theta}^{*b} - \hat{\theta}^*)^2 \right)^{0.5} \quad , \quad \hat{\theta}^* = \frac{1}{B} \sum_{b=1}^{B} \hat{\theta}^{*b}$$

Through Monte Carlo reasoning we can conclude that the bootstrap error converges to the ideal bootstrap error as B gets larger, i.e. $\lim_{B \to \infty} \hat{se}_B = se_{\hat{F}}(\hat{\theta}^*)$.

One might be tempted to exhaustively evaluate all possible x^* rather than using random sampling. That brings the question: For a given x, how many distinct bootstrap samples can we obtain? For a sample size of n, we can have $\begin{pmatrix} 2n-1 \\ n \end{pmatrix}$ distinct samples, and that can be a very large number. For instance, $n = 15$ results in $\sim 10^7$ distinct samples!

Law School Data

Index	LSAT	GPA		Index	LSAT	GPA
1	622	3.23		42	575	2.92
2	542	2.83		43	573	2.85
3	579	3.24		44	644	3.38
4	653	3.12		45	545	2.76
5	606	3.09		46	645	3.27
6	576	3.39		47	651	3.36
7	620	3.10		48	562	3.19
8	615	3.40		49	609	3.17
9	553	2.97		50	555	3.00
10	607	2.91		51	586	3.11
11	558	3.11		52	580	3.07
12	596	3.24		53	594	2.96
13	635	3.30		54	594	3.05
14	581	3.22		55	560	2.93
15	661	3.43		56	641	3.28
16	547	2.91		57	512	3.01
17	599	3.23		58	631	3.21
18	646	3.47		59	597	3.32
19	622	3.15		60	621	3.24
20	611	3.33		61	617	3.03
21	546	2.99		62	637	3.33
22	614	3.19		63	572	3.08
23	628	3.03		64	610	3.13
24	575	3.01		65	562	3.01
25	662	3.39		66	635	3.30
26	627	3.41		67	614	3.15
27	608	3.04		68	546	2.82
28	632	3.29		69	598	3.20
29	587	3.16		70	666	3.44
30	581	3.17		71	570	3.01
31	605	3.13		72	570	2.92
32	704	3.36		73	605	3.45
33	477	2.57		74	565	3.15
34	591	3.02		75	686	3.50
35	578	3.03		76	608	3.16
36	572	2.88		77	595	3.19
37	615	3.37		78	590	3.15
38	606	3.20		79	558	2.81
39	603	3.23		80	611	3.16
40	535	2.98		81	564	3.03
41	595	3.11		82	575	2.74

To illustrate the basic ideas behind bootstrap estimation of standard error, we consider the following estimation problem. We record average LSAT and GPA scores of students in $N = 82$ law schools denoting the full population. To demonstrate sampling, we take a sample of $n = 15$ from this population and estimation the coefficient of correlation between the two variables. Then, using bootstrap procedure we will estimate the standard error of this estimate.

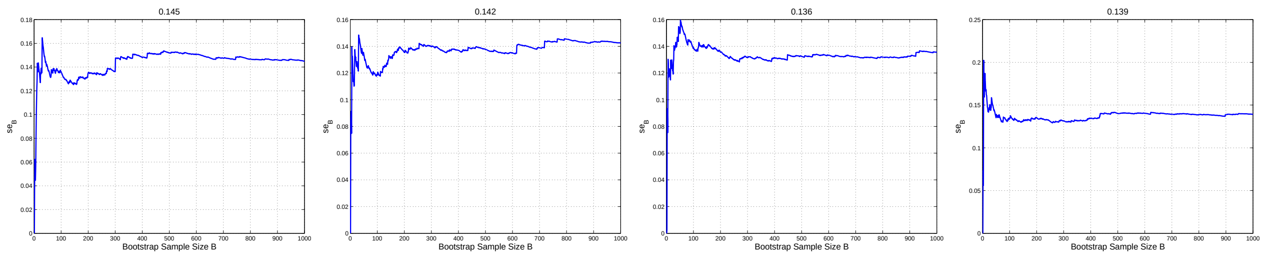


Figure 11.1: Evolution of $\hat{se}_B$ versus B for the sampled law school data, for four different runs of Algorithm 41.

We randomly select $n = 15$ samples from this population and obtain the following indices:

$$x = [72 \quad 14 \quad 80 \quad 22 \quad 73 \quad 82 \quad 29 \quad 20 \quad 57 \quad 51 \quad 70 \quad 81 \quad 23 \quad 28 \quad 60]$$

For these sample values we obtain an estimate of coefficient of correlation, $\hat{corr} = 0.6141$. In order to compute the standard error of this estimator, we perform the bootstrap procedure outlined in Algorithm 41. Shown in Figure 11.1 are four examples of evolution of the bootstrap error $\hat{se}_B$ versus B. The final value is shown at the top of each plot. The values are similar enough to imply convergence of $\hat{se}_B$ as B gets larger. In the problem of this scale, one can compute the exact standard error of $\hat{corr}$ and that is found out to be 0.131. Comparing it with the values in Figure 11.1, we find a reasonable similarity with $\hat{se}_B$.

What should be the value of B in an experiment?

Another experiment underscores the value of bootstrap in studying statistics of the estimator $\hat{corr}$. For a fixed x, we can generate bootstrap samples x^* and compute corresponding replicates $\hat{corr}^*$. A histogram of these replicates for $B = 3000$ is shown in Figure 11.2 top right panel. To compare this distribution, we have also generate a histogram of $\hat{corr}$ evaluated on x that are sampled from the original population using F; this histogram is shown in the top left panel. Similarity of the two histograms underlines the importance of bootstrap replication in studying statistics of $\hat{corr}$. To emphasize this point, we show two more histograms of $\hat{corr}^*$ under two different random samples of x.

11.3.2 Parametric Estimation

In this case one assumes that the unknown distribution F comes from a parametric family $F \equiv F_\theta$. Additionally, one assumes that it is possible to sample from F_θ for all θ. Using this structure, we can often improve the resampling part in Bootstrap procedure. Instead of using empirical distribution $\hat{F}$ to generate elements of x^*, we will now use the distribution $F_{\hat{\theta}}$, for $\hat{\theta}$ estimated from the data x, to generate bootstrap samples.

The parametric bootstrap procedure can be summarized as follows:

Algorithm 42 1. *Estimate the value of θ using the given estimator $\hat{\theta} = s(x) = s(x_1, x_2, \ldots, x_n)$.*

2. *Generate bootstrap samples x^{*b}, $b = 1, 2, \ldots, B$, where each element of x^{*b} is generated independently from the estimated distribution $F_{\hat{\theta}}$.*

3. *Evaluate bootstrap replicates of the estimator using $\hat{\theta}^{*b} = s(x^{*b})$.*

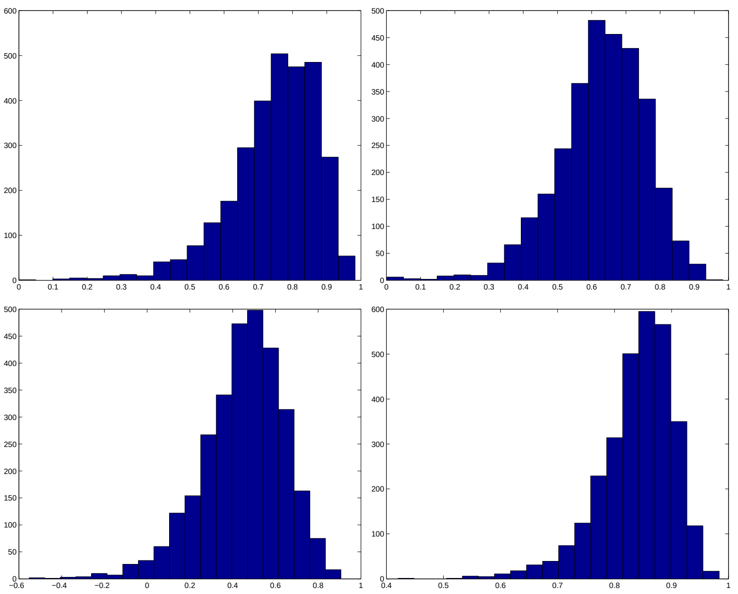


Figure 11.2: Histogram of random values of $\widehat{corr}$ under two distributions: (i) Top left is for x sampled from the population using F and $\widehat{corr}$ is computed for each x, and (ii) Top right is for $\widehat{corr}$ when x^* is sampled from $\hat{F}$ for a fixed x. Bottom are two more examples of bootstrap statistics of $\widehat{corr}^*$ for two different random samples of x.

4. Compute $\hat{se}_B$ as the standard deviation of $\hat{\theta}^{*b}$ values.

$$\hat{se}_B = \left(\frac{1}{B-1} \sum_{b=1}^{B-1} (\hat{\theta}^{*b} - \hat{\theta}^*)^2 \right)^{0.5} \quad , \quad \hat{\theta}^* = \frac{1}{B} \sum_{b=1}^{B} \hat{\theta}^{*b}$$

As an example, assume that the variables (LSAT and GPA scores) in the law school data are bivariate normal. We can estimate their means and covariances from the sample of size $n = 15$ taken earlier. Using Algorithm 42 we find a parametric estimate of the standard error of $\widehat{corr}$ to be 0.124, not too different from non-parametric estimates and the exact value stated earlier.

11.4 Estimation of Bias

So far we have considered only the standard error of an estimator for its evaluation but other properties such as bias also play an important role. In this section we focus on the estimation of bias of an estimator.

Continuing with the earlier setup, let $x = (x_1, x_2, \ldots, x_n)$ be the given observation from an unknown distribution F, and we are studying the given estimator $\hat{\theta} = s(x_1, x_2, \ldots, x_n)$. If θ takes the form $t(F)$, where t is a given functional, then $\hat{\theta}$ can take the form $t(\hat{F})$, where $\hat{F}$ is the empirical distribution based on the sample x.

The bias of an estimator $\hat{\theta}$ is defined to the difference between its expected value and the true value θ:

$$\text{bias}_F = E_F[\hat{\theta}] - \theta \quad \text{or} \quad E_F[s(x_1, x_2, \ldots, x_n)] - t(F) \ . \tag{11.4}$$

F is stated in this expression to emphasize that the expected values are obtained with respect to this distribution. If $\text{bias}_F = 0$ for an estimator, then it is called *unbiased*. Otherwise, it is a

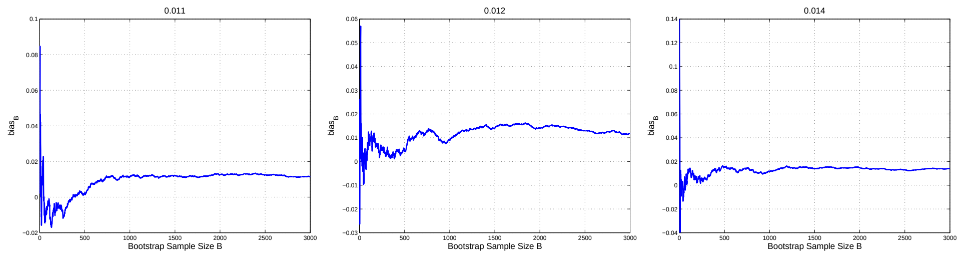


Figure 11.3: Evolution of $\hat{bias}_B$ versus B for three different runs of Algorithm 43 on the same sample x of law school data.

biased estimator. We want to estimate bias_F using a bootstrap procedure given the sample x and the estimator $s(x)$. The bootstrap estimator of the bias is given by simply replacing F in Eqn. 11.4 by the empirical distribution $\hat{F}$:

$$\text{bias}_{\hat{F}} = E_{\hat{F}}[s(x^*)] - t(\hat{F}) \ .$$

Note that the estimator $s(x)$ may be different from the plug-in estimator $t(\hat{F})$. $\text{bias}_{\hat{F}}$ is the plug-in estimate of bias_F whether or not $s(x)$ is a plug-in estimate of θ. Similar to the last section, $\text{bias}_{\hat{F}}$ is the ideal bootstrap estimate of the bias and is further estimated using B bootstrap samples. An algorithm for this computation is summarized next:

Algorithm 43 (Bootstrap Estimate of Bias)

1. *Generate bootstrap samples* $x^{*1}, x^{*2}, \ldots, x^{*B}$ *similar to Algorithms 41-42.*
2. *Compute bootstrap replicates of the estimator using* $\hat{\theta}^{*b} = s(x^{*b})$.
3. *Estimate the bias using:*

$$\hat{bias}_B = \frac{1}{B}\sum_{b=1}^{B} \hat{\theta}^{*b} - t(\hat{F}) \ .$$

Applying this algorithm the law school sample studied in the previous section, we obtain a $\hat{bias}_B \sim 0.011$. Shown in Figure 11.3 are three examples of evolutions of $\hat{bias}_B$ versus B on the same data sample x.

As another example, consider the following data. In an experiment, the blood harmone level, for a certain harmone, was measured under three different body patches. One patch is the one being used currently, called the old patch, another is the new patch being tried, and third represents a placebo. The parameter of interest is:

$$\theta = \frac{E[Y|\text{new patch}] - E[Y|\text{old patch}]}{E[Y|\text{old patch}] - E[Y|\text{placebo}]} \qquad (11.5)$$

where Y denotes the harmone level. The data was collected for eight different subjects:

#	placebo	old patch	new patch
1	9243	17649	16449
2	9671	12013	14614
...			
8	18806	29044	26325

Define two random variables:

$$\begin{aligned} Z_1 &= Y(\text{under new patch}) - Y(\text{under old patch}) \\ Z_1 &= Y(\text{under old patch}) - Y(\text{placebo}) \end{aligned}$$

and the parameter of interest becomes $\theta = \frac{E_F[Z_1]}{E_F[Z_2]}$. Using the plug-in principle, we obtain an estimate:

$$\hat{\theta} = \frac{E_{\hat{F}}[Z_1]}{E_{\hat{F}}[Z_2]} = \frac{\frac{1}{8}\sum_{i=1}^{8}(...)}{\frac{1}{8}\sum_{i=1}^{8}(...)} \ .$$

Substituting the observed values we get $\hat{\theta} = -0.0713$. Our goal is to estimate the bias of this estimator using bootstrap method. For $B = 400$, we obtain the average value of bootstrap replicates to be $\hat{\theta}^* = -0.0670$. Therefore, the bootstrap estimate of the bias is given by:

$$\hat{\text{bias}}_B = -0.0670 - (-0.0713) = 0.0043 \ .$$

To check if this value of estimated bias is significant, one can compare it with the standard error:

$$\hat{se}_B = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}(\hat{\theta}^{*b} - \hat{\theta}^*)^2} = 0.1049 \ .$$

Since $\frac{\hat{\text{bias}}_B}{\hat{se}_B} = 0.041$, one can safely decide that the bias is negligible.

Bias Correction: The process of estimation of bias can be used to correct and improve the estimator. The corrected estimator is given by:

$$\bar{\theta} = \hat{\theta} - \hat{\text{bias}} \ . \tag{11.6}$$

Since the estimated bias is $\hat{\text{bias}} = \hat{\theta}^* - \hat{\theta}$, we have $\bar{\theta} = 2\hat{\theta} - \hat{\theta}^*$. Repeated application of this correction will further improve the estimator.

To motivate this idea, consider an estimate that underestimates θ by 20%. If the true value of θ is 1.0, then $\hat{\theta} = 0.80$. Let us see what the value of estimated bias will be. The process of estimating bias using bootstrap involves resampling from the data (nonparametric approach), or from a data simulated from $\hat{\theta}$ (parametric approach). In either case, since the estimates under values by 80%, we expect the bootstrap average $\hat{\theta}^*$ to be $0.8 \times 0.8 = 0.64$. The estimated bias is.

11.5 Problems

1. Consider the law school data studied in this chapter. Write a matlab program to generate $n = 15$ sample pairs (LSAT and GPA) from the population of $N = 82$. Having obtained this sample write a matlab program to compute the following:

   (a) Estimate the means (μ_x and μ_y) and correlation coefficient from the sampled data.

   (b) Estimate the standard errors of each of these estimators for $B = 25$, 50, 100, 200, 500, 1000, and 2000. Either tabulate them or plot curves.

   (c) Plot a histogram of the bootstrap replicates of (estimated) correlation coefficients for the case $B = 2000$.

   To compare, generate $n = 15$ random samples from the original population and compute the estimated correlation coefficient. Generate 2000 such values by repeated sampling of the population to form a second histogram. Compare the two histograms.

2. Consider an artificial data set consisting of the 10 numbers:

$$1 \;\; 2 \;\; 3.5 \;\; 4 \;\; 7 \;\; 7.3 \;\; 8.6 \;\; 12.4 \;\; 13.8 \;\; 18.1 \; .$$

   Let $\hat{\theta}$ be the 25% trimmed mean, computed by deleting the smallest two numbers and the largest two numbers, and then taking the average of the remaining six numbers.

   (a) Calculate $\hat{se}_B$ for $B = 25,\ 100,\ 200,\ 500,\ 1000,\ 2000$.

   (b) Repeat part (a) using ten different random number seeds and assess the variability in the estimates. How large should we take B to provide reasonable accuracy?

3. Generate $n = 11$ samples of a normal random variable with mean zero and variance 3.0. Fix these values to be $x_1, \ldots, x_n$. Use your program to estimate the standard error of the estimator:

$$s(x_1, \ldots, x_n) = \text{median}(x_1, x_2, \ldots, x_n) \; .$$

   Plot its variation against B.

4. Generate n values from a standard normal random variable. Let $\theta = mean(X)$, and the estimator be the median of the observed values. Use bootstrap method to estimate the bias of this estimator for $n = 10,\ 20$ and 100.

Chapter 12

SPECIAL TOPICS

12.1 Dynamic Programming

Dynamic Programming has emerged as an important tool in finding optimal paths in a variety of situations. This tool is useful in finding an optimal trajectory in situations where the decisions are made in stages and decisions made in past stages have future consequences. In particular, the decisions are made at discrete indices. The cost associated with a trajectory is the sum of costs associated with these individual decisions.

We start with a general state equation for a discrete time system. Let $x_k \in \mathbb{R}^n$ be the state of a system at time k, and its evolution be governed by the equation:

$$x_{k+1} = f_k(x_k, u_k, w_k)\ ,\ k = 0, 1, \ldots, N-1 \tag{12.1}$$

where u_k is the decision selected at time k, w_k is a random disturbance (or noise), and f_k is a deterministic function that relates the next state x_{k+1} to x_k, u_k, and w_k. N is the considered the final time, or horizon. The cost associated with the system at time k is given by $g_k(x_k, u_k, w_k)$, so that the total cost becomes:

$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)\ , \tag{12.2}$$

where the first term is called the terminal cost. Since w_k is a random variable, for each k, the above cost involving w_ks is also a random quantity. Therefore, one is often interested in the expected cost given by:

$$E[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)] \tag{12.3}$$

where the expectation is computed with respect to the joint probability density of all the w_ks. Our goal is solve the problem: for $u = (u_0, u_1, \ldots, u_{N-1})$

$$\min_u E[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)], \quad \text{subject to appropriate constraints .} \tag{12.4}$$

This is referred to as the *basic problem* in dynamic programming.

To motivate dynamic programming, we introduce two examples. The first example of scheduling is a deterministic problem, while the second one involving inventory control is stochastic problem.

Example 9 (Scheduling) *To produce a certain product, four operations: A, B, C, and D, must be performed on a certain machine. The constraints in the manufacturing process are: B can be performed only when A has been performed. Similarly, D can be performed only when C has been performed. As an example, the sequence CDAB is allowed while the sequence CDBA is not.*

We are given costs associated with all the transitions: C_{AB} for following A by B, C_{CD} for following C by D, and so on. We are also given costs for starting with A and C, S_A and S_C, respectively. This way we can find the cumulative cost of any sequence. As an example, the cost of CDAB is $S_C + C_{CD} + C_{DA} + C_{AB}$. Our goal is to find a valid sequence that has the smallest cost. One is to exhaustively list all the paths, compute their cost, and select the optimal one. Instead, we will study a way that avoid enumerating all possible paths.

Example 10 (Inventory Control) *Consider the problem of ordering a certain product at each of the N periods to meet a (random) demand while minimizing the incurred expected cost. The decision relates to the quantity of this product ordered at each of the intervals, in such a way that the purchase cost and storage expenses are minimized. Additionally, there is a cost for having unmet demand. We set this problem up mathematically as follows.*

Let x_k be the stock (quantities) of that product available at the k^{th} period, u_k be the stock purchased (and available for consumption) during that period, and w_k be the random demand whose probability distribution is given. To simplify the problem, we make the following assumptions. First, we assume that demands at different times, w_0, w_1,...w_{N-1}, are independent of each other. Secondly, we assume that excess demand is backlogged and filled as soon as new inventory becomes available. In other words, any excess demand is not lost but we try to meet it in the next periods. The models for stock at time $k+1$ is:

$$x_{k+1} = x_k + u_k - w_k \ . \tag{12.5}$$

The cost associated with purchase of a unit is c, resulting in the total purchase cost cu_k at time k. The cost for storage (or excess demand) is given by a function $r(x_k + u_k - w_k)$; and for example, $r(x) = x^2$ penalizes both the excess demand and excess inventory quadratically. The total expected cost over N periods is given by:

$$E[\sum_{k=0}^{N-1}(cu_k + r(x_k + u_k - w_k))] \ , \tag{12.6}$$

and the goal is to minimize this cost by a proper choice of purchase orders u_0, u_1, ..,u_{N-1} subject to the constraint that $u_k \geq 0$ for all k.

As the inventory problem suggests, our goal is to come up with a formula that computes an optimal decision u_k depending upon the current state x_k. In other words, rather than seeking a fixed number, we seek and optimal policy that calculates the optimal number depending upon values of the current variables. Let this policy be denoted by the function $\mu_k(x_k)$; depending on the current stock x_k, we will order the quantity $u = \mu_k(x_k)$. The whole set of policies is denoted by $\pi = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$. The total expected cost resulting from a policy set π is given by:

$$J_\pi(x_0) = E[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k)] \ , \tag{12.7}$$

where the expectation is taken over the probability distribution of w_ks. For a given policy π, the total expected cost is simply a function of the initial stock x_0. In other words, we seek

$$\pi^*(x_0) = \underset{\pi}{\operatorname{argmin}}\, J_\pi(x_0) \ . \tag{12.8}$$

This is a restatement of the basic problem introduced earlier.

Theorem 20 (Principle of Optimality) *Let $\pi^* = (\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*)$ be an optimal policy for the basic problem. Assume that when using π^*, a state x_i occurs at time i with positive probability. Consider the subproblem of starting from x_i (at time i) and we wish to minimize the remaining cost:*

$$E[g_N(x_N) + \sum_{k=i}^{N-1} g_k(x_k, \mu_k(x_k), w_k)] \ .$$

Then, the truncated policy $\{\mu_i^*, \mu_{i+1}^*, \ldots, \mu_{N-1}^*\}$ is optimal for this subproblem.

A simple example of this principle is as follows: consider an optimal (under certain criterion) path going from Boston to Los Angeles, and assume that this path passes through Chicago. Then, the principle of optimality states that the optimal path from Chicago to Los Angeles will be the corresponding portion of the original optimal path.

How can this principle be used for solving problems involving path optimizations? We illustrate its use in solving the basic problem introduced in Eqn. 12.8. In this instance we will use a backward version of DP, although one can solve it identically using the forward version also. We will start with the very last time $k = N$ and work backwards till we reach the initial time $k = 0$.

1. For $k = N$, the cost associated with this stage is

$$J_N(x_N) = g_N(x_N) \ .$$

   This cost does not depend on any input and there is not optimization involved.

2. For $k = N - 1$, the input is based on the policy $\mu_{N-1}(X_{N-1})$. The cost associated with this policy is given by

$$J_{N-1}(x_{N-1}) = E_{w_{N-1}}[J_N(x_N) + g_{N-1}(x_{N-1}, \mu_{N-1}(x_{N-1}), w_{N-1})] \ ,$$

   where the term x_N is also influenced by μ_{N-1} and is thus a function of it. This influence can be written as $x_N = f_N(x_{N-1}, \mu_{N-1}(x_{N-1}), w_{N-1})$. Therefore, the optimal cost associated with this stage is obtained by solving:

$$J_{N-1}^*(x_{N-1}) = \min_{\mu_{N-1}(x_{N-1})} E_{w_{N-1}}[g_N(f_N(x_{N-1}, \mu_{N-1}(x_{N-1}), w_{N-1})) + g_{N-1}(x_{N-1}, \mu_{N-1}(x_{N-1}), w_{N-1})] \ .$$

   We have also used the fact that $J_N(x_N) = g_N(x_N)$. Solving this minimization results in the minimal cost J_{N-1}^* and the optimal policy μ_{N-1}^*, both as functions of the state X_{N-1}.

3. For $k = N - 2$, the input is based on the policy $\mu_{N-2}(X_{N-2})$. The cost associated with this policy is given by

$$J_{N-2}(x_{N-2}) = E_{w_{N-2}}[J_{N-1}^*(x_{N-1}) + g_{N-2}(x_{N-2}, \mu_{N-2}(x_{N-2}), w_{N-2})] \ .$$

The function $J^*_{N-1}(x_{N-1})$

$x_N = f_N(x_{N-1}, \mu_{N-1}(x_{N-1}), w_{N-1})$. Therefore, the optimal cost associated with this stage is obtained by solving:

$$J^*_{N-1}(x_{N-1}) = \min_{\mu_{N-1}(x_{N-1})} E_{w_{N-1}}[g_N(f_N(x_{N-1}, \mu_{N-1}(x_{N-1}), w_{N-1})) + g_{N-1}(x_{N-1}, \mu_{N-1}(x_{N-1}), w_{N-1})] .$$

We have also used the fact that $J_N(x_N) = g_N(x_N)$. Solving this minimization results in the minimal cost J^*_{N-1} and the optimal policy μ^*_{N-1}, both as functions of the state X_{N-1}.

12.2 Perfect Sampling

`To be added...`

12.3 Statistics of Directional Data

In applications where the variables of interest are *directions*, the ensuing statistical analysis takes on additional structure. Consider the problem of estimating direction of an incoming signal using noisy observations collected from acoustic sensors. If the system lies in a plane, then the direction lies on a unit circle $\mathbb{S}^1$, and the estimation problem is restricted to $\mathbb{S}^1$. For problems involving three-dimensional data collection, the space of directions is given by the unit sphere $\mathbb{S}^2$. Similarly, in applications involving geometries of component one is concerned with relative orientations of these components in forming a larger structure. For example, in determining protein structures one is interested in learning the relative angles at which amino acids and other elements bond each other forming primary, secondary, and tertiary chains. In such problems requiring inferences on spaces of directions, $\mathbb{S}^n$, with $n = 1, 2, \dots,$, the classical statistical methods do not directly apply. The reason is nonlinearity of $\mathbb{S}^n$; we can not use the calculus associated with vector spaces, such as $\mathbb{R}^n$, on nonlinear space $\mathbb{S}^n$. Using the geometry of $\mathbb{S}^n$, one can derive definitions and algorithms for computing statistics of directional data.

In this section, we describe ideas and techniques for computing statistics such as means, covariances, and even probability distributions on $\mathbb{S}^1$. We will define representations, metrics, probability models, and statistics for $\mathbb{S}^1$.

1. **Representations**: The first question we address is how to represent elements of $\mathbb{S}^1$. For instance, we can denote a point on a circle using its Euclidean coordinates (x_1, x_2) in $\mathbb{R}^2$, with the additional constraint that $x_1^2 + x_2^2 = 1$. Ad advantage of this representation is that each point is represented uniquely, although it uses two variables to represent an element of a one-dimensional space. A point on $\mathbb{S}^1$ can also be denoted by an angle $\theta \in [0, 2\pi)$ where 0 is identified with 2π. The representation is unique as along as 2π is not allowed in the space. An additional representation for planar rotation is given by the special orthogonal group:
$$SO(2) = \{O \in \mathbb{R}^{2\times 2} | O^T O = I_2, \det(O) = 1\} .$$
In words, $SO(2)$ is the set of all 2×2 orthogonal matrices with positive determinant. Elements of $SO(2)$ take the form:
$$O = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} .$$
Multiplying a vector in $\mathbb{R}^2$ by this O results in its counter-clockwise rotation by an angle θ around the origin.

2. **Distances**: Similar to different choices of representations of elements on $\mathbb{S}^1$, one can have different choices of distances between its elements. For the unit vector representation, we can choose the Euclidean distance in $\mathbb{R}^2$. If $x = (x_1, x_2)$ and $y = (y_1, y_2)$ are two points on a circle, then a distance between them can be $\|x - y\| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$; this is also called the *chord length.* If points on $\mathbb{S}^1$ is denoted as $x = [\cos(\theta_1) \quad \sin(\theta_1)]$ and $y = [\cos(\theta_2) \quad \sin(\theta_2)]$, then the chord length can also be expressed as:

$$\|x - y\| = \sqrt{2}\sqrt{1 - \cos(\theta_1 - \theta_2)} \ .$$

A similar distance is obtained by representing elements of $\mathbb{S}^1$ as 2×2 orthogonal matrices and by computing Euclidean distances between them in $\mathbb{R}^4$. This distance turns out to be:

$$\|O_1 - O_2\| = \sqrt{\sum_{i,j=1}^{2} (O_1(i,j) - O_2(i,j))^2} = 2\sqrt{1 - \cos(\theta_1 - \theta_2)} \ ,$$

$\sqrt{2}$ times the Euclidean distance in $\mathbb{R}^2$. Another distinct possibility is to use the *arc length* distance between angle representations of elements of $\mathbb{S}^1$. This distance is given by:

$$d(\theta_1, \theta_2) = \min\{|\theta_1 - \theta_2|, |\theta_1 + 2\pi - \theta_2|, |\theta_1 - 2\pi - \theta_2|\} \ .$$

This expression measures the length of shortest path between $\theta_1, \theta_2 \in \mathbb{S}^1$ on the circle. These two distances – chord length and arc length – provide two distinct measurements of separation between points and lead to quite distinct ways of computing statistics on $\mathbb{S}^1$.

3. **Probability Densities on** $\mathbb{S}^1$: Since $\mathbb{S}^1$ is a compact space with periodic boundaries, the classical probability models have to be modified. As an example, the commonly used normal density function is not applicable directly as a probability density on $\mathbb{S}^1$ as the normal density has infinite tails. So, the question is: What probability densities can be used for statistical modeling on $\mathbb{S}^1$? Of course, the simplest choice is the uniform density function: $f(\theta) = \frac{1}{2\pi}$. All elements of $\mathbb{S}^1$ occur with equal probability. However, analogous to the normal random variable on $\mathbb{R}^1$, one would like a density with a mean value and a specifiable variance around it. This role is performed by the **von-Mises density** function given by:

$$f(\theta; \mu, a) = \frac{1}{I_0(a)} \exp(a \cos(\theta - \mu)) \ , \tag{12.9}$$

where μ is the mean and a is a called the *dispersion* parameter. I_0 is the modified Bessel function of order zero, and serves as the normalizing constant for f. The parameter a behaves as the inverse of the variance; large value of a implies smaller variance and vice-versa. Shown in Figure 12.1 are a few examples von-Mises density for a fixed μ and increasing values of a.

4. **Sampling on** $\mathbb{S}^1$: Next, we consider the problem of simulating a random variable taking values on $\mathbb{S}^1$. Of the three methods studied in Chapter 5 – inverse transform method, acceptance-rejection method, and composition method – the second one is general enough to be useful here. Of course, while the strength of an acceptance-rejection method is its wide applicability, its main weakness is the inefficiency resulting from rejection of proposed values. Consider the problem of generating random values from a von-Mises density function with a given mean μ and dispersion a, and let the candidate probability

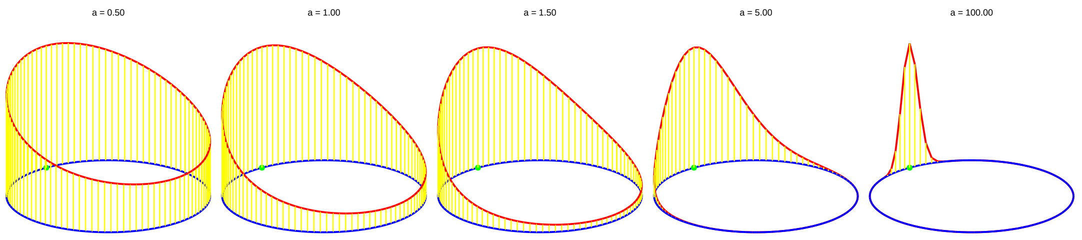


Figure 12.1: Examples of von-Mises density function on $\mathbb{S}^1$ for increasing values of dispersion parameter a from left to right.

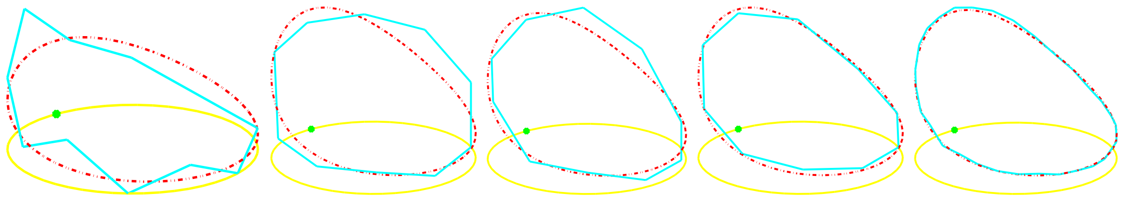


Figure 12.2: Samples from von-Mises density function on $\mathbb{S}^1$ for increasing values of accepted samples from left to right.

be the uniform density function. Then, the bounding coefficient is given by:

$$\frac{f(\theta;\mu,a)}{1/2\pi} = \frac{2\pi}{I_0(a)}\exp(a\cos(\theta-\mu)) \leq \frac{2\pi}{I_0(a)}\exp(a) \equiv c \ .$$

Hence, the ratio:

$$R(\theta) \equiv \frac{f(\theta;\mu,a)}{c/2\pi} = \exp(a(\cos(\theta-\mu)-1)) \ .$$

The acceptance-rejection sampling algorithm is given by:

Algorithm 44 (Acceptance-Rejection for Von-Mises Density)

```
1. Generate $U_1 \sim U[0,1]$ and set $Y = 2\pi U_1$.
2. Generate $U_2 \sim U[0,1]$.
3. If $U_2 < R(Y)$, then set $X = Y$.
   Else, return to Step 1.
```

Shown in Figure 12.2 are examples of this sampling procedures. Each panel shows histograms of samples generate using this algorithm; the number of samples increases from left to right. The solid line shows the observed histograms and the broken line shows the underlying von-Mises density function.

5. **Sample Statistics**: Now that we know how to sample from a given density, we address the question of computing sample statistics, such as mean and variance, from the observed samples. In view of the nonlinear geometry of $\mathbb{S}^1$, the definition of statistics is neither straightforward nor unique. In fact there are two distinct ways of defining statistics: extrinsic and intrinsic, and we we study them next.

(a) **Extrinsic Statistics**: The basic idea here is to embed the underlying space in a larger Euclidean space, and to compute statistics in that large space. Since the computed statistic does not lie in the underlying space, one can project the solution into the required space. We elaborate this procedure for $\mathbb{S}^1$, considered as an embedding in $\mathbb{R}^2$. For a set of sampled values $\{\theta_i,\ i = 1, 2, \dots, n\}$, find $\mathbb{R}^2$ coordinates of each according to $x_i = [\cos(\theta_i)\ \ \sin(\theta_i)] \in \mathbb{R}^2$, and find their mean according:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \ \ \mathbb{R}^2 \ .$$

Since $\bar{x}$ is not in $\mathbb{S}^1$, we can project it to $\mathbb{S}^1$ using $\bar{x} \mapsto \hat{\mu}_n \equiv \bar{x}/\|\bar{x}\|$. Of course, this projection is valid only if $\|\bar{x}\| \neq 0$. In case $\bar{x}$ coincides with the origin, then there is not preferred projection, and each point of $\mathbb{S}^1$ is a valid estimate of the mean $\hat{\mu}_n$. This point, or a set of points $\hat{\mu}_n$ is called the *extrinsic mean* of the observed set, and it depends on the chosen embedding. If we had chosen another way of embedding $\mathbb{S}^1$ in $\mathbb{R}^2$, or some other higher dimensional vector space, then the resulting extrinsic mean may take a different value.

With an eye towards the intrinsic analysis, we redefine the extrinsic mean as follows. The extrinsic mean can be defined as:

$$\hat{\mu}_n = \operatorname*{argmin}_{x \in \mathbb{S}^1} \sum_{i=1}^{n} \|x - x_i\|^2 \ , \tag{12.10}$$

where $\|\cdot\|$ denotes the Euclidean distance as usual. It can be shown that the solution can be computed in two steps as:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \in \mathbb{R}^2 \quad \text{and} \quad \hat{\mu}_n = \operatorname*{argmin}_{x \in \mathbb{S}^1} \|x - \bar{x}\| \ . \tag{12.11}$$

The last minimization seeks a point on $\mathbb{S}^1$ that is closest to $\bar{x}$ in the Euclidean distance, and that is obtained by taking the projection $\bar{x}/\|\bar{x}\|$ as stated earlier. For a probability density function f on $\mathbb{S}^1$, we can define the mean similarly. Let $x(\theta)$ be the embedding of θ in $\mathbb{R}^2$, and define a probability density g on $\mathbb{R}^2$ such that $f(\theta) = g(x(\theta))$ and $g(x) = 0$ for $x \notin \mathbb{S}^1$.

$$\mu = \operatorname*{argmin}_{x \in \mathbb{S}^1} \int_{\mathbb{S}^1} \|x - y\|^2 g(y)dy \ ,$$

and μ is computed in two steps according to: (i) define $\bar{x} = \int_{\mathbb{R}^2} y g(y)dy$, and (ii) form the projection $\mu = \bar{x}/\|\bar{x}\|$.

The variance associated with a density function is given by:

$$\sigma^2 = \min_{x \in \mathbb{S}^1} \int_{\mathbb{S}^1} \|x - y\|^2 g(y)dy \ ,$$

and the sample variance is given by:

$$s^2 = \operatorname*{argmin}_{x \in \mathbb{S}^1} \sum_{i=1}^{n} \|x - x_i\|^2 \ .$$

(b) **Intrinsic Statistics**: Similar to Eqn. 12.10, one can define a mean using the arc length distance on $\mathbb{S}^1$ and this approach

6. **Monte Carlo Estimation**: Let f be a probability density on $\mathbb{S}^1$ and for a given function $g : \mathbb{S}^1 \mapsto \mathbb{R}$, we want to estimate a quantity:

$$\phi = \int_{\mathbb{S}^1} g(\theta) f(\theta) d\theta \ .$$

In case g is a complicated function, this integral can not be performed analytically, and one can resort to a Monte Carlo approximation. The classical MC approach is to generate independent samples $\theta_i \sim f$, and use the approximation:

$$\hat{\phi}_n = \frac{1}{n} \sum_{i=1}^{n} g(\theta_i) \ .$$

Appendix A

Appendix

A.1 Summary of Matrix Analysis

Let A be an $m \times n$ matrix, represented by $A = (a_{ij})$ for

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & & & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

We will also denote the columns of A by a_i's, $A = [a_1 \ a_2 \dots \ a_n]$.

The vectors $\{a_1, a_2, \dots, a_n\}$ are linearly independent if $\sum_{j=1}^{n} \alpha_j a_j = 0$ implies that $\alpha_j = 0$ for all j. A subset of $I\!R^n$ that is also a vector space is called *subspace* of $I\!R^n$. The span of vectors $a_1, a_2, \dots, a_n$ is defined by

$$span\{a_1, a_2, \dots, a_n\} = \{\sum_{j=1}^{n} \beta_j a_j : \beta_j \in I\!R\}$$

If the vectors $\{a_1, a_2, \dots, a_n\}$ are linearly independent abd $b \in span\{a_1, a_2, \dots, a_n\}$ then the coefficients of b are unique. For a subspace $S \subset I\!R^n$, the vectors $\{a_1, a_2, \dots, a_n\}$ are its basis if

$$S \subset span\{a_1, a_2, \dots, a_n\}$$

For a $m \times n$ matrix A:

- the range space of A is given by

$$range(A) = \{r \in I\!R^m : y = Ax, \text{ for some } \ x \in I\!R^n\}$$

- the null space is given by

$$null(A) = \{x \in I\!R^n : Ax = 0\}$$

- the rank of A is defined to be $rank(A) = dimension(range(A))$.

If A is a $n \times n$ matrix,

- $rank(A) = rank(A^T)$.
- $dim(null(A)) + dim(range(A)) = n$.

Denote $e_i \in I\!R^n$ to be unit vector such that it has 1 in the i-th location and zero everywhere else. Let I_n denote an $ntimesn$ identity matrix. For a $n \times n$ matrix A, if $XA = I_n$ then X is the inverse of A, denote $inv(A)$ or A^{-1}. If A^{-1} exists then A is called non-singular.

Let A be an $n \times n$ matrix and $U, V \in I\!R^{n \times k}$ be rank-k matrices for $k \leq n$, then $A + UV^T$ is called the rank k update of A. Assuming that A and $I_n + V^T A^{-1} U$ are non-singular, the inverse of the update is given by

$$(A + UV^T)^{-1} = A^{-1} - A^{-1}U(I_n + V^T A^{-1} U)^{-1} V^T A^{-1} \ .$$

This implies that the rank-k correction to the matrix results in a rank-k correction to it's inverse. The determinant of a matrix is defined in a recursive fashion. It is given by

$$det(A) = \sum_{i=1}^{n} (-1)^{j+1} a_{ij} det(A_{ij})$$

where A_{ij} is $(n-1) \times (n-1)$ matrix obtained by deleting the i-th row and j-th column.

Bibliography

[1] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap.* Chapman & Hall, 1993.

[2] Gene H. Golub and C. F. Vanloan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, 1989.

[3] Samuel Karlin and Howard M. Taylor. *A Second Course in Stochastic Processes.* Academic Press, Inc., New York, 1981.

[4] D.G. Luenberger. *Optimization by Vector Space Methods.* John Wiley and Sons, New York, 1969.

[5] M. J. Maron. *Numerical Analysis: A Practical Approach.* Macmillan Publishing Compnay, 1982.

[6] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods.* Springer Text in Statistics, 1999.

[7] S. M. Ross. *Introduction to Probability Models.* Academic Press, Inc., 1985.

[8] B. W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman and Hall, 1985.

[9] Ronald A. Thisted. *Elements of Statistical Computing: Numerical Computing.* Chapman & Hall, 1996.