

CHAPTER 2

Some Basic Algorithms

The Quicksort Algorithm

The problem of sorting (arranging a set of numbers or character variables $X(1), \dots, X(n)$ into non-decreasing order) occurs in many places in statistics. The basic operations in sorting are comparisons and swaps (exchanges) of elements of the array. Most naive sorting algorithms (such as the infamous bubble sort, see below) require proportional to n^2 operations on the average to sort a vector of n randomly generated distinct numbers.

Bubble Sort

Perhaps the most famous sorting algorithm is bubble sort since it is simple to code and has a catchy name. Unfortunately bubble sort is also a terribly slow method unless n is very small (less than 20 or so). The algorithm starts by defining a boundary $k = n$ and then keeps going through the data for $i = 1, \dots, k-1$, interchanging $X(i)$ and $X(i+1)$ if they are out of order and each time setting k to be the largest value of i for which a swap is made. The algorithm stops if a pass through the data is made with no swaps.

Quicksort

Quicksort starts by splitting (partitioning) $X(1), \dots, X(n)$ into three pieces: 1) one element (called the splitting element) which is placed in its proper place in the final sorted array, 2) a piece consisting of all of the elements in the array that are less than or equal to the splitting element (these are placed to the left of the splitting element), and 3) a piece consisting of all of the elements that are greater than the splitting element (these are placed to the right of the splitting element). All of this is done “in place”, that is, in the same array that we started with. Once this splitting is done, the algorithm continues by splitting each of the two pieces, and then splitting the resulting four pieces, etc. until there are no pieces left to split (we know this will happen as every split results in an element being in its proper place). The algorithm keeps track of what pieces need to be split by keeping a counter of how many there are as well as two arrays containing the left and right “boundaries” of the pieces. There are two other basic rules that are used in the algorithm: 1) pieces of size three or fewer are sorted, not split (it’s faster to sort them than to split them), and 2) the next piece to be split is the shorter of the two pieces obtained from the previous splitting (this makes it so that the “stack”, that is, the arrays containing the boundaries of the pieces still to be split, need only be of length k where k is the smallest integer such that $2^k > n$). Notice that it could happen that the splitting element is the largest of smallest element in a piece in which case the splitting only results in a single piece.

Choosing the Splitting Element and Doing the Splitting

Ideally, one would like the splitting element to be approximately the median of the piece to split as then the piece would be split approximately in half. There is no generally agreed upon method for choosing the splitting element, but one popular choice is as the median of the first, last, and “middle” (that is, the ones having indexes i , j , and $[(i+j)/2]$ where i and j are the boundaries of the piece. Another possibility is to just choose one of the elements of the piece at random.

The actual splitting process is difficult to describe. Consult the code below for one possible method.

Extensions

When data come in pairs $(X(i), Y(i))$, $i = 1, \dots, n$, it is often desired to sort one of the variables while preserving the pairing. One simple method for doing this is to pass both arrays to the algorithm and merely replace the SWAP routine by a SWAP2 routine that swaps the elements of the second array whenever elements of the first array are swapped. Note that QKSORT is written so that swapping is done only in the SWAP routine, which makes this extension simple.

Another common task is to find the ranks of the elements of an array. One way to do this is to do the paired version of the quicksort algorithm with an array of X 's together with an array of indices (which starts out with i th element being i). The resulting array of indices can then be used to easily find the ranks.

```

1      subroutine qsort(x,n)
2 c*****
3 c
4 c   Quicksort a real array x of length n.
5 c
6 c*****
7 c
8       double precision x(n)
9       integer ll(20),lr(20)
10 c
11 c   ns is # of pieces left to partition
12 c
13 c   ll and lr are left and right borders of pieces
14 c
15       ns=1
16       ll(1)=1
17       lr(1)=n
18 c
19 c   After splitting we come here. Stop when no pieces left.
20 c
21     5  if(ns.eq.0) go to 99
22 c
23       i=ll(ns)
24       j=lr(ns)
25       nl=j-i+1
26 c
27 c   Sort pieces of size 1 or 2:
28 c
29       if(nl.le.2) then
30         ns=ns-1
31         if(nl.eq.2.and.x(i).gt.x(j)) call swap(x(i),x(j))
32         go to 5
33       endif
34 c
35 c   Sort first, ‘middle’ and last elements:
36 c
37       nm=(i+j)/2
38       if(x(i).gt.x(nm)) call swap(x(i),x(nm))
39       if(x(nm).gt.x(j)) call swap(x(nm),x(j))
40       if(x(i).gt.x(nm)) call swap(x(i),x(nm))
41 c
42 c   If piece of size 3, it's now sorted
43 c
44       if(nl.eq.3) then
45         ns=ns-1
46         go to 5
47       endif
48 c
49 c   Put middle (target) into 1st element and keep a copy for comparisons
50 c

```

```

51      ax=x(nm)
52      call swap(x(i),x(nm))
53 c
54 c  Now we'll look for 1st one from left (starting with 2nd) > target
55 c  and first from right < target. If no such pair we end up at 20.
56 c
57      itemp=i
58      i=i+1
59 c
60 10  if(x(i).le.ax) then
61      if(i.eq.j) go to 20
62      i=i+1
63      go to 10
64  endif
65 c
66 15  if(x(j).ge.ax) then
67      if(i.eq.j) go to 20
68      j=j-1
69      go to 15
70  endif
71 c
72 c  Can only get to here if we found a pair to swap
73 c
74      call swap(x(i),x(j))
75 c
76 c  Only go back to look for another pair if there's a chance to get one
77 c
78      if(j-i.le.1) then
79          go to 20
80      else
81          i=i+1
82          j=j-1
83          go to 10
84  endif
85 c
86 c  Get to here when no more pairs to swap. i might be 1 space too far
87 c  to right
88 c
89 20  if(x(i).gt.ax) i=i-1
90      call swap(x(itemp),x(i))
91 c
92 c  Now put shorter piece at end of ll and lr (this guarantees small
93 c  number of pieces at any one time). A piece might be empty, but that
94 c  is taken care of above.
95 c
96      nleft=i-ll(ns)
97      nright=lr(ns)-i
98      ns=ns+1
99      if(nleft.le.nright) then
100         ll(ns)=ll(ns-1)
101         ll(ns-1)=i+1
102         lr(ns)=i-1
103     else
104         lr(ns)=lr(ns-1)
105         lr(ns-1)=i-1
106         ll(ns)=i+1
107     endif
108     go to 5
109 c
110 99  continue
111     return
112     end
113     subroutine swap(a,b)
114 c*****
115 c
116 c  Swap two reals.
117 c
118 c*****
119 c
120     double precision a,b,c
121     c=b
122     b=a
123     a=c
124 c
125     return
126     end

```

Vector Operations

The inner product of n -dimensional vectors x and y is given by

$$\langle x, y \rangle = x^T y = y^T x = \sum_{i=1}^n x_i y_i,$$

and x and y are said to be orthogonal (denoted by $y \perp x$) if $\langle x, y \rangle = 0$. By the operation of “making y orthogonal to x ” we mean the finding a vector $z = y - \alpha x$, where the number α is chosen so that $z \perp x$, that is, so that

$$(y - \alpha x)^T x = (y^T - \alpha x^T) x = y^T x - \alpha x^T x = 0,$$

that is $\alpha = \langle y, x \rangle / \langle x, x \rangle$.

Matrix Operations

Most matrix operations in statistical computing are concerned with either an $n \times m$ matrix X having columns x_1, \dots, x_m or with a symmetric positive semidefinite $m \times m$ matrix V . The case where $V = X^T X$ is particularly important. In almost every situation, n is bigger than m , that is, X has more columns than rows. In this case, the matrix X is said to be of full rank if all of its columns are linearly independent, that is, no column of X can be written as a linear combination of the remaining columns. This can be expressed succinctly by saying that for any m dimensional vector h whose elements are not all zero, then $Xh \neq 0$.

An $(m \times m)$ matrix V is said to be positive definite or positive semidefinite if for any nonzero m -dimensional nonzero vector h , we have $h^T V h > 0$ or $h^T V h \geq 0$ respectively. If $V = X^T X$, then its (i, j) th element is the inner product $\langle x_i, x_j \rangle = \langle x_j, x_i \rangle$ and thus it is symmetric. Further, for any nonzero vector h ,

$$h^T V h = h^T X^T X h = (Xh)^T (Xh) = z^T z = \langle z, z \rangle,$$

which is nonnegative for any X , and in fact, is positive if and only if X is of full rank.

In statistics it often necessary to find what is called the square root of a positive definite matrix.

Definition. The square root of a symmetric, positive definite $(m \times m)$ matrix V is an $(m \times m)$ matrix A satisfying $V = AA^T$. We denote such a matrix by $V^{1/2}$. The inverse square root of V is the inverse of $V^{1/2}$ and is denoted by $V^{-1/2}$. The transposes of V^{-1} and $V^{-1/2}$ are denoted by V^{-T} and $V^{-T/2}$ respectively.

In this section we consider a variety of operations on matrices like X and $X^T X$. We will answer a wide variety of questions, including

1. How do we check whether X is of full rank?
2. How do we check whether V is positive definite?
3. How do we find the square root of V ?
4. How do we find the determinant of V ?
5. How do we solve a system of equations $Vx = b$?

The Modified Cholesky Decomposition

The next theorem shows that there exists a unique matrix square root for any positive definite matrix and also shows how to find it. Note that a triangular matrix is called unit if it has ones on the main diagonal.

Theorem 1.	MODIFIED CHOLESKY DECOMPOSITION
-------------------	---------------------------------

Let V be a symmetric $(m \times m)$ matrix. Then

a) V is positive definite if and only if there exists a unique unit lower triangular $(m \times m)$ matrix L and a unique diagonal $(m \times m)$ matrix D having positive diagonal elements, such that

$$V = LDL^T.$$

b) This factorization is called the modified Cholesky decomposition (MCD) of V , and if the decomposition exists we can calculate the elements of L and D one row at a time by $D_{11} = V_{11}$ and for $i = 2, \dots, m$:

$$L_{ij} = \frac{V_{ij} - \sum_{l=1}^{j-1} L_{il} D_{ll} L_{jl}}{D_{jj}}, \quad j = 1, \dots, i-1$$

$$D_{ii} = V_{ii} - \sum_{l=1}^{i-1} D_{ll} L_{il}^2.$$

c) The MCD of V is nested; that is, if V_k , L_k , and D_k represent the upper left-hand $(k \times k)$ parts of V , L , and D , respectively, then

$$V_k = L_k D_k L_k^T, \quad k = 1, \dots, m,$$

and thus for any k greater than or equal to i and j , the (i, j) th elements of V_k , L_k , and D_k can be denoted by V_{ij} , L_{ij} , and D_{ij} , respectively.

d) The unique square root of a positive definite matrix V is given by $V^{1/2} = LD^{1/2}$ where $D^{1/2} = \text{Diag}(D_{11}^{1/2}, \dots, D_{nn}^{1/2})$.

e) The inverse square root of a positive definite matrix V is given by $V^{-1/2} = D^{-1/2}L^{-1}$ where $D^{-1/2} = \text{Diag}(D_{11}^{-1/2}, \dots, D_{nn}^{-1/2})$.

Solving Systems of Equations and Checking for Positive Definiteness

The Gram-Schmidt Decomposition

An important operation in many areas of mathematics and statistics is to find what is called an orthogonal basis for a matrix; that is, from one set of vectors, find a new set by some linear transformation such that the inner product of any two different new vectors is zero. We will use the Gram-Schmidt decomposition (see Clayton (1971)) to accomplish this aim.

Theorem 2	GRAM-SCHMIDT DECOMPOSITION
------------------	----------------------------

If X is an $(n \times m)$ matrix having full rank m , then there exists an $(n \times m)$ orthogonal matrix Q (that is, $Q^T Q$ is diagonal) and a unit upper triangular $(m \times m)$ matrix R such that

$$X = QR.$$

This decomposition of X is called its Gram-Schmidt decomposition (GSD).

Note that $X^T X = R^T Q^T Q R = R^T V R$ where R^T is unit lower triangular and V is diagonal with positive diagonal elements. Since the modified Cholesky decomposition $X^T X = LDL^T$ of $X^T X$ is unique, we have that $R^T = L$ and $V = D$. The Modified Gram-Schmidt decomposition algorithm for an $n \times m$ matrix X having columns x_1, \dots, x_m consists of a series of $m - 1$ steps. The nature of these steps is such that the algorithm can be done in place, that is, the matrix Q will be produced in the matrix X .

For $i = 1, \dots, m - 1$, the i th step consists of making the columns x_{i+1}, \dots, x_m orthogonal to x_i . As each new column is produced it replaces the original column in X . The coefficients that are produced by the orthogonalizations are placed in $R_{i,i+1}, \dots, R_{im}$.

The Sweep Operator

Many of the computational and theoretical results in regression, multivariate analysis, and time series can be expressed succinctly using what is called the sweep operator.

Definition. Let A be an $(n \times n)$ matrix. The process of sweeping A on its k th diagonal element, denoted by $B = \text{SWEEP}(k)A$, is the process of forming the matrix B by (assuming $A_{kk} \neq 0$):

$$\begin{aligned} B_{kk} &= \frac{1}{A_{kk}} \\ B_{ik} &= -\frac{A_{ik}}{A_{kk}}, \quad i \neq k \quad (\textit{kth column}) \\ B_{kj} &= \frac{A_{kj}}{A_{kk}}, \quad j \neq k \quad (\textit{jth row}) \\ B_{ij} &= A_{ij} - \frac{A_{ik}A_{kj}}{A_{kk}}, \quad i \neq k, j \neq k. \end{aligned}$$

If $B_1 = \text{SWEEP}(k_1)A$, $B_2 = \text{SWEEP}(k_2)B_1$, \dots , $B = \text{SWEEP}(k_r)B_{r-1}$, we write $B = \text{SWEEP}(k_1, \dots, k_r)A$, and say that B is the result of sweeping A on its diagonals k_1, \dots, k_r .

The sweep operator has many useful properties as we see in the next theorem.

Theorem 3	PROPERTIES OF THE SWEEP OPERATOR
------------------	----------------------------------

Let

$$A = \begin{bmatrix} B & C \\ D & E \end{bmatrix},$$

where B , C , D , and E are $(r \times r)$, $(r \times s)$, $(s \times r)$, and $(s \times s)$ respectively. Then

a) $\text{SWEEP}(k_1, \dots, k_p)A$ can be found by sweeping the diagonals in any order, for example,

$$\text{SWEEP}(k_1, k_2)A = \text{SWEEP}(k_2, k_1)A.$$

b) $\text{SWEEP}(k, k)A = A$; that is, sweeping a second time on a given diagonal undoes the effect of a previous sweeping on that diagonal.

$$c) \text{ SWEEP}(1, \dots, r)A = \begin{bmatrix} B^{-1} & B^{-1}C \\ -DB^{-1} & E - DB^{-1}C \end{bmatrix} \text{ if } B \text{ is nonsingular.}$$

$$d) \text{ SWEEP}(r + 1, \dots, r + s)A = \begin{bmatrix} B - CE^{-1}D & -CE^{-1} \\ E^{-1}D & E^{-1} \end{bmatrix} \text{ if } E \text{ is nonsingular.}$$

$$e) \text{ SWEEP}(1, \dots, r + s)A = A^{-1} \\ = \begin{bmatrix} B^{-1} + B^{-1}C(E - DB^{-1}C)^{-1}DB^{-1} & -B^{-1}C(E - DB^{-1}C)^{-1} \\ -(E - DB^{-1}C)^{-1}DB^{-1} & (E - DB^{-1}C)^{-1} \end{bmatrix}$$

if B , E , and $(E - DB^{-1}C)^{-1}$ are nonsingular.

The sweep operator can be used to solve many theoretical and computational problems. We consider two examples of its use. First, the basic quantities for the regression of y on X (where X is $(n \times m)$) can be found by applying property (c)

$$\text{SWEEP}(1, \dots, m) \begin{bmatrix} X^T X & X^T y \\ y^T X & y^T y \end{bmatrix} = \begin{bmatrix} (X^T X)^{-1} & \hat{\beta} \\ -\hat{\beta}^T & \text{RSS} \end{bmatrix}.$$

To illustrate the theoretical utility of sweep, notice for the matrix A in Theorem 3 that sweeping on all of its diagonals is the same as sweeping the matrix $\text{SWEEP}(r + 1, \dots, r + s)A$ on diagonals $1, \dots, r$. Thus the upper left-hand corner in part (e) is the inverse of the upper left-hand corner in part (d) by part (c). Doing a similar calculation with $\pm E^{-1}$ replacing E in A gives the well-known matrix inversion lemma (see Rao (1973), p. 33 for example).

Theorem 4 THE MATRIX INVERSION LEMMA

Let B and E be nonsingular $(r \times r)$ and $(s \times s)$ matrices and C and D be $(r \times s)$ and $(s \times r)$ matrices. Then

$$(B \pm CED)^{-1} = B^{-1} \mp B^{-1}C(E^{-1} \pm DB^{-1}C)^{-1}DB^{-1}.$$

This formula is used extensively in recursive regression, that is, when we consider adjusting estimators for the addition or deletion of observations. It can also be used to motivate the Kalman Filter Algorithm.

Finding Zeros of Functions

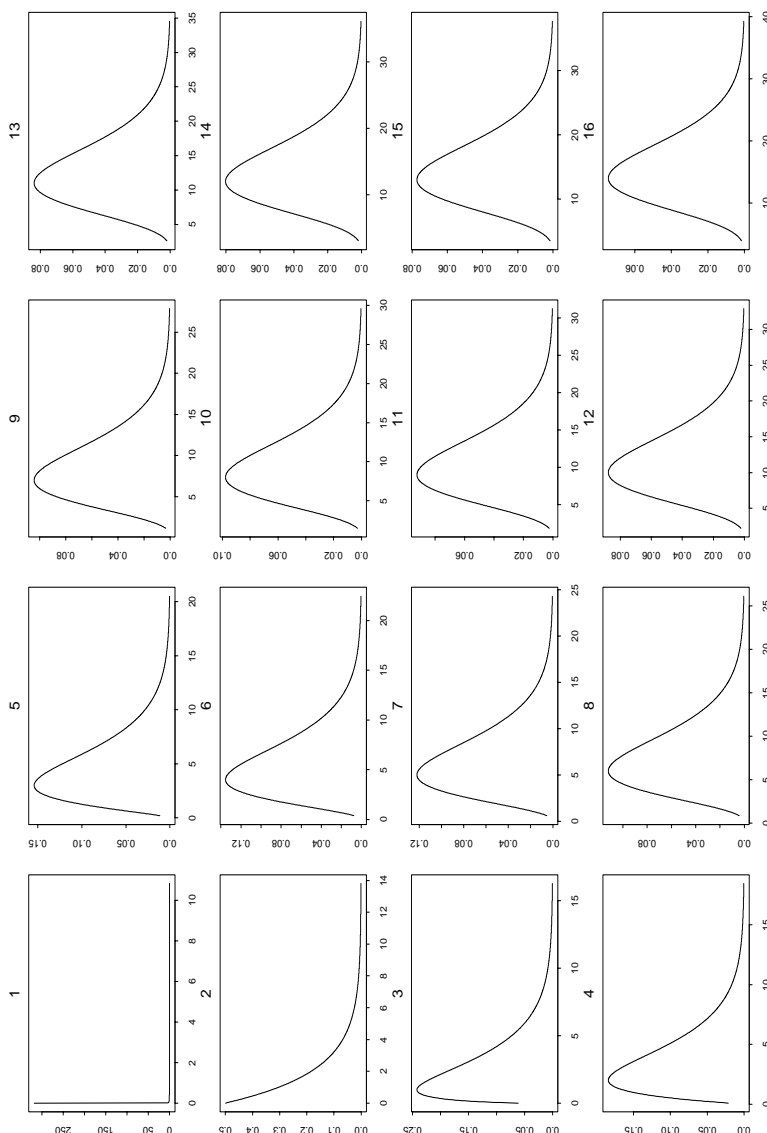
In this section we consider two standard methods for finding a zero of a function f . Note that these methods can be used to find a relative min or max of f by applying them to the derivative of f .

The first method is called Newton's method and consists of the the iterative process

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \quad i = 0, 1, 2, \dots,$$

given a starting value x_0 . In most cases, if the starting value is close to the zero, the convergence of the process is very rapid.

An alternative method called bisection is very effective if one can bracket a zero of f in an interval $[l, u]$. One starts by checking whether the zero is in $[l, m]$ or $[m, u]$, where $m = (l + u)/2$ is the midpoint of $[l, u]$



χ^2 pdfs for degrees of freedom $\nu = 1, 2, \dots, 16$.

(the zero is in the left or right half of the interval depending on whether the sign of $f(l)f(m)$ is negative or positive, respectively). If the zero is in the left (right) half of the interval, one sets $u = m$ ($l = m$) and repeats the process. This continues until the length of the bracketing interval becomes very close to zero.

An example that combines Newton's method and bisection is as follows. A χ^2_ν pdf for $\nu \geq 3$ is unimodal (see the set of graphs of the χ^2 pdf). For a given ν and probability $\alpha < 0.5$, we want to find the value f_α of the pdf

$$f_{\chi^2}(x; \nu) = \frac{1}{2^{\nu/2} \Gamma(\nu/2)} e^{-x/2} x^{(\nu/2)-1}, \quad x > 0,$$

(and the two values x_l and x_u of x that make the pdf equal to f_α) so that the total area under the pdf to the left of x_l and the to the right of x_u is α (this is used in finding the $100(1 - \alpha)\%$ confidence interval for the variance σ^2 of a normal population so that the interval has smallest width). Note that this process consists of raising and lowering a horizontal line cutting the pdf until the area in the tails is α . For a specified value of the pdf, we use Newton's method to find x_l and x_u using starting values equal to the $\alpha/2$ and $1 - (\alpha/2)$ quantiles of the distribution. We can bracket the value of the pdf by a small value that will work for all

values of ν (like 0.001) and its value at the 0.25 quantile of the distribution.

The Newton-Raphson Procedure

If we want to minimize the scalar function $L(\theta)$ of the r dimensional argument θ and we have an initial guess θ_0 for the value θ^* of θ maximizing L , then the Newton-Raphson algorithm calculates the vectors θ_1 , θ_2 , and so on until they converge to θ^* (or possibly don't converge) by

$$\theta_{i+1} = \theta_i - H_i^{-1}g_i, \quad i \geq 0,$$

where g_i and H_i are called the gradient vector and Hessian matrix of L and are the vector of first derivatives of L with respect to each of the r elements of θ and the $r \times r$ matrix of second derivatives, respectively. We can also write this process as

$$\theta_{i+1} = \theta_i - \delta_i, \quad \text{where } H_i\delta_i = g_i,$$

since typically it is better to solve a system of equations rather than finding an inverse and doing the multiplication.

The Newton-Raphson procedure is the multidimensional version of Newton's root finding algorithm and actually will find a critical point of L , that is, a value of θ for which the elements of the gradient vector are all zero. Such a point can be a maximum or a minimum and care must be taken to ensure that a critical point is in fact what one wants. We will consider minimization since if θ^* is a minimum, the matrices H_1 , H_2 , and so on can be expected to be positive definite and we can use the Modified Cholesky Decomposition method of solving the system $H_i\delta_i = g_i$.