What is the Median of $\{1,2,3,5\}$?

Joe Dellinger

INTRODUCTION

Medians are used in data processing because of their insensitivity to the occasional data value which differs wildly from the mean. The usual definition is that X is the ***median*** of a set of numbers $\{x_i\}$ if the number of i for which $X > x_i$ is equal to the number of i for which $X < x_i$. This definition works well in practice, but mathematically leaves much to be desired. For one thing, it cannot guarantee that the median is unique. In fact, if the number of x_i is even, this will almost certainly not be the case. Another problem is that it does not easily generalize to the multi-dimensional case.

Francis Muir's definition of median does not have either of these problems. It guarantees a ***unique*** median and is easily generalized to higher dimensions. The unique median given by this definition is also a median according to the more often used definition given above. A sample program will also be given showing how Francis' definition may be used computationally.

FRANCIS MUIR'S DEFINITION AND ITS PROPERTIES

The Definition

Let

$$L_p(\{x_i\}) = (\frac{1}{N}\sum_i |x_i|^p)^{1/p},$$

where N is the number of elements in the set $\{x_i\}$. This is the L_p norm of $\{x_i\}$.

Then:
X is said to be the *median* of $\{x_i\}$ if it minimizes $L_p(\{X - x_i\})$ in the limit as $p \to 1^+$.

This definition is a bit tricky. For one thing, notice that the minimization precedes the taking of the limit. The order *cannot* be reversed, or the median will not be unique. Also, notice that p must approach 1 from the plus side. The two-sided limit may not even exist.

Given this definition, it can now be shown that the median defined by it exists and is unique for all sets $\{x_i\}$ and that this median is also one of the possible medians given by the definition of the introduction, if the median given by that definition exists.

Uniqueness

If $L_p(\{X - x_i\})$ has a continuous first derivative as a function of X, then the minimum must be assumed at a point where $\frac{d}{dX}L_p(\{X - x_i\}) = 0$. Now, this derivative is

$$(\frac{1}{N})^{\frac{1}{p}}(\sum_i |X - x_i|^p)^{\frac{1}{p}-1}(\sum_i |X - x_i|^{(p-1)}\mathrm{sgn}(X - x_i)).$$

The only element of the formula that is discontinuous is the sgn function, but the effect of this discontinuity is voided because it is always multiplied by the term $|X - x_i|^{p-1}$, which at the discontinuity has the value zero. (Remember that $p > 1$.) Thus the derivative is continuous.

All extrema can be found by setting this derivative to 0. The term $(\frac{1}{N})^{\frac{1}{p}}$ is a constant. The term $(\sum_i |X - x_i|^p)^{(\frac{1}{p}-1)}$ is always positive unless all of the x_i are equal. So if the derivative is to be 0 it must be because

$$\sum_i |X - x_i|^{p-1}\mathrm{sgn}(X - x_i) = 0. \qquad (1)$$

There is at least one zero, since for $X < \min(\{x_i\})$ the left hand side of equation (1) must be negative, and for $X > \max(\{x_i\})$ it must be positive. Since it is continuous, there must be some X for which equation (1) is satisfied. Furthermore, since each term of the summation is monotonically increasing, their sum is also monotonically increasing and so this solution is unique. Since the derivative is going from negative to positive, this solution must correspond to the unique global minimum that we want. Thus, the minimum exists and is unique for every p greater than 1. It still is left to be proven that the limit as $p \to 1^+$ indeed exists.

Existence

To prove that the median does indeed exist, it is worthwhile to look at two cases separately. In the first case, the sgn functions are the determining factor. Consider equation (1) as a function of p and X. Instead of forcing it to equal 0 we will put a Y for the right hand side of the equation. The limit of this equation as $p \to 1^+$ is

$$\sum_i \operatorname{sgn}(X - x_i) = Y.$$

If Y switches from being negative to positive at one well-defined point, then the corresponding value of X must be in fact the limiting value of X satisfying equation (1) as $p \to 1^+$. This is the case because as p tends nearer and nearer to 1 the $|X - x_i|^{p-1}$ terms can be made arbitrarily close to unity over the entire range of possible solutions. Therefore it is the discrete jumps from -1 to 1 of the sgn functions that ultimately dominate in controlling the limiting value of X.

In the second case there must be an interval in which

$$\sum_i \operatorname{sgn}(X - x_i) = 0 \tag{2}$$

is satisfied throughout. Within this interval the determining factor is not, then, the sgn functions, but instead the $|X - x_i|^{p-1}$ terms. If there is a limiting X it must be within this interval.

As an important aside, note that the first definition of median given can be restated as "for all X such that equation (2) is satisfied, X is a median of the set $\{x_i\}$".

Thus, if there is a limiting X satisfying Francis Muir's definition of the median, such an X must also be a median in the usual sense.

The best way to show that the limit in the second case does indeed exist is to find an expression for it. Let x_1 be the largest element in the set $\{x_i\}$. Then changing x_1 will not change the interval that X is constrained to lie in. For a given p, X and x_1 are then related through equation (1). As it will turn out, neither X nor x_1 will be a discontinuous function of the other either for a given p near 1 or in the limit. This is important because then we can consider X to be a function of x_i, find the limit as $p \to 1^+$, and then set x_1 to have its known value and so get an equation for the value of X in the limit.

Equation (1) is thus rewritten as:

$$x_1 = \lim_{\varepsilon \to 0^+} \Big[\sum_{x_i < X} (X - x_i)^\varepsilon - \sum_{x_i > X, i \neq 1} (x_i - X)^\varepsilon \Big]^{\frac{1}{\varepsilon}} + X. \tag{3}$$

Note that the sum inside the square brackets is 1 in the limit. Thus, taking the limit now involves only an easy 1^∞ sort of limit. This is solved by the standard trick of taking logs and using l'Hôpital's rule. This done, equation (3) becomes

$$x_1 = \frac{\prod_{x_i < X} (X - x_i)}{\prod_{x_i > X, i \neq 1} (x_i - X)} + X. \tag{4}$$

Since X is seen not to be a discontinuous function of x_1 nor vice-versa given the range x_1 was chosen to be in, our trick was valid and we have found an equation giving the limiting value of X. Stripping x_1 of its special status, equation (4) can be written symmetrically as

$$\prod_{x_i < X} (X - x_i) = \prod_{x_i > X} (x_i - X). \tag{5}$$

This is the equation giving Francis Muir's unique median in the interesting case that the median is not one of the x_i. Note that this equation may have more than one solution but has exactly one in the interval of interest.

An Example: The median of $\{1, 2, 3, 5\}$

The solution proceeds in a straightforward fashion:

$$(X - 1)(X - 2) = (3 - X)(5 - X)$$

to

$$X^2 - 3X + 2 = 15 - 8X + X^2$$

to

$$5X = 13$$

to

$$X = 2\frac{3}{5}.$$

This is the median of $\{1,2,3,5\}$.

A C PROGRAM FOR FINDING THE UNIQUE MEDIAN

It is easy to write a subroutine to return the unique median. It has to check which of the two cases is appropriate, and if the median is not equal to one of the elements in the set it must then find an approximate solution to equation (5). This is easy because it is guaranteed that there is only one solution in the search interval, and that within this interval both sides of equation (5) are monotonic. The program which follows does a straightforward binary search. At each step of the iteration the search interval is halved and the program determines which half to continue searching in. When the interval is small enough that the requested level of precision has been reached, the value found is returned.

The routine *median* calls the subroutine *quant* which was printed in SEP–10 on page 100. For convenience the routine is reprinted here, in slightly different form, and with some documentation to explain its algorithm, which the original was lacking.

Both these routines are given in an Appendix.

REFERENCES

Muir, F. 1984. personal communication

Canales, L. 1976. A quantile finding program, SEP–10, p.99-100

APPENDIX

median.c

```
/*
 * Median.c
 * This subroutine finds the unique median of a set of numbers.
 * The subroutine quant partially orders the array x to find the
 * interval to search for the median in.
 */

#include <math.h>
float   median (x, n, error)
int     n;                      /* Size of x array */
float   *x;                     /* Array of floats to take the median of */
float   error;                  /* Tolerance required. Must be positive. */
{
    int     k;
    float   lower, upper;/* Limits of interval being searched */
    float   X;                  /* Trial median value */
    float   left, right;/* Logs of left and right hand side of equation 5 */

    if (n == (2 * (int) (n / 2)))
    {
    /* n even */
        k = n / 2;              /* Find the interval to search in */
        quant (k, x, n);
        quant (k - 1, x, n);
        upper = x[k];
        lower = x[k - 1];
        if (upper == lower)/* easy case */
            return upper;
    /* Hard case. Do a binary search */
        while (upper - lower > error)/* Stop when you are within tolerance */
        {
            X = (upper + lower) / 2.;/* See which half contains the solution */
        /* Use logs for better numerical stability. */
            left = 0.;          /* Initialize before summing */
            right = 0.;
            for (k = 0; k < n / 2; k++)
            {
                left += (float) log ((double) (X - x[k]));/* Left side */
            }
            for (k = n / 2; k < n; k++)
            {
                right += (float) log ((double) (x[k] - X));/* Right side */
            }
            if (left > right)
                upper = X;  /* Keep lower half */
            else
                if (left < right)
                    lower = X;/* Keep upper half */
                else
                    return X;/* We got it exactly accidentally */
        }
    /* sufficient accuracy attained, stop */
        return (upper + lower) / 2.;
    }
    else
    {
    /* n odd */
    /* Easy case. Find the element that is the median */
        k = n / 2;
        quant (k, x, n);
        return x[k];
    }
}
```

quant.c

```
/*
 * Quant.c
 *
 *      This is a translation into C of a fortran program in SEP 10 p100.
 */
/*
 *      a is the input data.
 *      To find the k'th element in the completely ordered data set,
 *      without completely ordering it, call quant with
 *      n the total number of elements, and k the one you are
 *      interested in. Upon return, a[k] will be what you want.
 *      if you want several k, call again and again with k lower each
 *      time. a is reordered when quant returns, be warned.
 *      Note that k is thought of as in the C convention...
 *                  0 <= k <= n - 1
 */
/*
 * The basic idea is this: Pick an element. Subdivide the data into
 * two sets: those less than the chosen element, those more than the
 * chosen element. See which subset the element you want must be in.
 * repeat using that subset.
 *
 * The subdivision into two subsets is done by starting in from each end
 * looking for an element that is "out of place". When one is found, the
 * search continues on the other end. When you have one at each end, swap
 * them. Now those are both in their correct subset. Continue until your
 * two searches run into each other in the middle somewhere.
 *
 * This algorithm is somewhat dumb about values equal to your chosen value:
 * it always thinks they are in the wrong subset.
 *
 * It is also more efficient to do a search for the maximum or minimum of
 * the subset under consideration when you know that the element you want
 * is one of those two rather than to continue by the "divide and conquer"
 * method. But this program doesn't do that.
 */

quant (k, a, n)
int     k, n;
float   *a;
{
    int     low, hi, i, j;
    double      ak, aa;
    low = 0;
    hi = n - 1;
    while (low < hi)
    {
        ak = a[k];
        i = low;
        j = hi;
        do
        {
            while (a[i] < ak)
            {
                i++;
            }
            while (a[j] > ak)
            {
                j--;
            }
            if (i <= j)
            {
                aa = a[i];
                a[i] = a[j];
                a[j] = aa;
                i++;
                j--;
            }
        }
        while (i <= j);
        if (j < k)
        {
            low = i;
        }
        if (k < i)
        {
            hi = j;
        }
    }
}
```