NUMERICAL RECIPES

Webnote No. 11, Rev. 1

Code Listing for selip

Two minor additional tricks in the following routine, `selip`, are (i) augmenting the set of M random values by an $M+1$st, the arithmetic mean, and (ii) choosing the M random values "on the fly" in a pass through the data, by a method that makes later values no less likely to be chosen than earlier ones. (The underlying idea is to give element $m > M$ an M/m chance of being brought into the set. You can prove by induction that this yields the desired result.)

selip.h

```
Doub selip(const Int k, VecDoub_I &arr) {
Given k in [0..n-1] returns an array value from arr[0..n-1] such that k array values are less
than or equal to the one returned. The input array is not altered.
    const Int M=64;
    const Doub BIG=.99e99;
    Int i,j,jl,jm,ju,kk,mm,nlo,nxtmm,n=arr.size();
    Doub ahi,alo,sum;
    VecInt isel(M+2);
    VecDoub sel(M+2);
    if (k < 0 || k > n-1) throw("bad input to selip");
    kk=k;
    ahi=BIG;
    alo = -BIG;
    for (;;) {                          Main iteration loop, until desired ele-
        mm=nlo=0;                           ment is isolated.
        sum=0.0;
        nxtmm=M+1;
        for (i=0;i<n;i++) {             Make a pass through the whole array.
            if (arr[i] >= alo && arr[i] <= ahi) {
                Consider only elements in the current brackets.
                mm++;
                if (arr[i] == alo) nlo++;       In case of ties for low bracket.
                Now use statistical procedure for selecting m in-range elements with equal
                probability, even without knowing in advance how many there are!
                if (mm <= M) sel[mm-1]=arr[i];
                else if (mm == nxtmm) {
                    nxtmm=mm+mm/M;
                    sel[(i+2+mm+kk) % M]=arr[i];    The % operation provides a some-
                }                                       what random number.
                sum += arr[i];
            }
        }
        if (kk < nlo) {                 Desired element is tied for lower bound;
            return alo;                     return it.
        }
        else if (mm < M+1) {            All in-range elements were kept. So re-
            shell(sel,mm);                  turn answer by direct method.
            ahi = sel[kk];
            return ahi;
        }
```

```
        sel[M]=sum/mm;                          Augment selected set by mean value (fixes
        shell(sel,M+1);                              degeneracies), and sort it.
        sel[M+1]=ahi;
        for (j=0;j<M+2;j++) isel[j]=0;          Zero the count array.
        for (i=0;i<n;i++) {                     Make another pass through the array.
            if (arr[i] >= alo && arr[i] <= ahi) {      For each in-range element..
                jl=0;
                ju=M+2;
                while (ju-jl > 1) {             ...find its position in the selected set by
                    jm=(ju+jl)/2;                    bisection...
                    if (arr[i] >= sel[jm-1]) jl=jm;
                    else ju=jm;
                }
                isel[ju-1]++;                   ...and increment the counter.
            }
        }
        j=0;                                    Now we can narrow the bounds to just
        while (kk >= isel[j]) {                      one bin, that is, by a factor of order
            alo=sel[j];                              m.
            kk -= isel[j++];
        }
        ahi=sel[j];
    }
}
```

Approximate timings: `selip` is about 10 times slower than `select`. Indeed, for N in the range of $\sim 10^5$, `selip` is about 1.5 times slower than a full sort with `sort`, while `select` is about 6 times faster than `sort`. You should weigh time against memory and convenience carefully.