NUMERICAL RECIPES

Webnote No. 10, Rev. 1

Complete Miser Code

Here is the full listing of miser and its utility routine ranpt.

miser.h

```
void miser(Doub func(VecDoub_I &), VecDoub_I &regn, const Int npts,
    const Doub dith, Doub &ave, Doub &var) {
```

Monte Carlo samples a user-supplied ndim-dimensional function func in a rectangular volume specified by regn[0..2*ndim-1], a vector consisting of ndim "lower-left" coordinates of the region followed by ndim "upper-right" coordinates. The function is sampled a total of npts times, at locations determined by the method of recursive stratified sampling. The mean value of the function in the region is returned as ave; an estimate of the statistical uncertainty of ave (square of standard deviation) is returned as var. The input parameter dith should normally be set to zero, but can be set to (e.g.) 0.1 if func's active region falls on the boundary of a power-of-two subdivision of region.

```
    const Int MNPT=15, MNBS=60;
    const Doub PFAC=0.1, TINY=1.0e-30, BIG=1.0e30;
```

Here PFAC is the fraction of remaining function evaluations used *at each stage* to explore the variance of func. At least MNPT function evaluations are performed in any terminal subregion; a subregion is further bisected only if at least MNBS function evaluations are available. We take MNBS $= 4 *$ MNPT.

```
    static Int iran=0;
    Int j,jb,n,ndim,npre,nptl,nptr;
    Doub avel,varl,fracl,fval,rgl,rgm,rgr,s,sigl,siglb,sigr,sigrb;
    Doub sum,sumb,summ,summ2;

    ndim=regn.size()/2;
    VecDoub pt(ndim);
    if (npts < MNBS) {                          Too few points to bisect; do straight
        summ=summ2=0.0;                            Monte Carlo.
        for (n=0;n<npts;n++) {
            ranpt(pt,regn);
            fval=func(pt);
            summ += fval;
            summ2 += fval * fval;
        }
        ave=summ/npts;
        var=MAX(TINY,(summ2-summ*summ/npts)/(npts*npts));
    } else {                                    Do the preliminary (uniform) sampling.
        VecDoub rmid(ndim);
        npre=MAX(Int(npts*PFAC),Int(MNPT));
        VecDoub fmaxl(ndim),fmaxr(ndim),fminl(ndim),fminr(ndim);
        for (j=0;j<ndim;j++) {                  Initialize the left and right bounds for
            iran=(iran*2661+36979) % 175000;       each dimension.
            s=SIGN(dith,Doub(iran-87500));
            rmid[j]=(0.5+s)*regn[j]+(0.5-s)*regn[ndim+j];
            fminl[j]=fminr[j]=BIG;
            fmaxl[j]=fmaxr[j]=(-BIG);
        }
        for (n=0;n<npre;n++) {                  Loop over the points in the sample.
            ranpt(pt,regn);
            fval=func(pt);
            for (j=0;j<ndim;j++) {                  Find the left and right bounds for each
                if (pt[j]<=rmid[j]) {                   dimension.
                    fminl[j]=MIN(fminl[j],fval);
                    fmaxl[j]=MAX(fmaxl[j],fval);
                } else {
                    fminr[j]=MIN(fminr[j],fval);
                    fmaxr[j]=MAX(fmaxr[j],fval);
                }
            }
        }
        sumb=BIG;                                   Choose which dimension jb to bisect.
        jb= -1;
        siglb=sigrb=1.0;
        for (j=0;j<ndim;j++) {
            if (fmaxl[j] > fminl[j] && fmaxr[j] > fminr[j]) {
                sigl=MAX(TINY,pow(fmaxl[j]-fminl[j],2.0/3.0));
                sigr=MAX(TINY,pow(fmaxr[j]-fminr[j],2.0/3.0));
                sum=sigl+sigr;                      Equation (7.9.24), see text.
                if (sum<=sumb) {
                    sumb=sum;
                    jb=j;
                    siglb=sigl;
                    sigrb=sigr;
                }
            }
        }
        if (jb == -1) jb=(ndim*iran)/175000;  MNPT may be too small.
        rgl=regn[jb];                               Apportion the remaining points between
        rgm=rmid[jb];                                   left and right.
        rgr=regn[ndim+jb];
        fracl=abs((rgm-rgl)/(rgr-rgl));
        nptl=Int(MNPT+(npts-npre-2*MNPT)*fracl*siglb
            /(fracl*siglb+(1.0-fracl)*sigrb));                  Equation (7.9.23).
        nptr=npts-npre-nptl;
        VecDoub regn_temp(2*ndim);                  Now allocate and integrate the two sub-
        for (j=0;j<ndim;j++) {                          regions.
            regn_temp[j]=regn[j];
            regn_temp[ndim+j]=regn[ndim+j];
        }
        regn_temp[ndim+jb]=rmid[jb];
        miser(func,regn_temp,nptl,dith,avel,varl);
        regn_temp[jb]=rmid[jb];                     Dispatch recursive call; will return back
        regn_temp[ndim+jb]=regn[ndim+jb];               here eventually.
        miser(func,regn_temp,nptr,dith,ave,var);
        ave=fracl*avel+(1-fracl)*ave;
        var=fracl*fracl*varl+(1-fracl)*(1-fracl)*var;
        Combine left and right regions by equation (7.9.11) (1st line).
    }
}
```

ranpt.h

```
void ranpt(VecDoub_O &pt, VecDoub_I &regn) {
Returns a uniformly random point pt in an n-dimensional rectangular region. Used by miser.
    static const int RANSEED=5331;
    static Ran ran(RANSEED);
    Int j,n=pt.size();
    for (j=0;j<n;j++) pt[j]=regn[j]+(regn[n+j]-regn[j])*ran.doub();
}
```