NUMERICAL RECIPES

Webnote No. 9, Rev. 1

Complete VEGAS Code

Here is the full listing of vegas and its accompanying utility routine rebin.

vegas.h

```
void vegas(VecDoub_I &regn, Doub fxn(VecDoub_I &, const Doub), const Int init,
    const Int ncall, const Int itmx, const Int nprn, Doub &tgral, Doub &sd,
    Doub &chi2a) {
```

Performs Monte Carlo integration of a user-supplied ndim-dimensional function fxn over 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 integration consists of itmx iterations, each with approximately ncall calls to the function. After each iteration the grid is refined; more than 5 or 10 iterations are rarely useful. The input flag init signals whether this call is a new start or a subsequent call for additional iterations (see comments in the code). The input flag nprn (normally 0) controls the amount of diagnostic output. Returned answers are tgral (the best estimate of the integral), sd (its standard deviation), and chi2a (χ^2 per degree of freedom, an indicator of whether consistent results are being obtained). See text for further details.

```
    static const Int NDMX=50, MXDIM=10, RANSEED=5330;
    static const Doub ALPH=1.5, TINY=1.0e-30;
    static Int i,it,j,k,mds,nd,ndo,ng,npg;
    static Doub calls,dv2g,dxg,f,f2,f2b,fb,rc,ti;
    static Doub tsi,wgt,xjac,xn,xnd,xo,schi,si,swgt;
    static VecInt ia(MXDIM),kg(MXDIM);
    static VecDoub dt(MXDIM),dx(MXDIM),r(NDMX),x(MXDIM),xin(NDMX);
    static MatDoub d(NDMX,MXDIM),di(NDMX,MXDIM),xi(MXDIM,NDMX);
```

Best make everything static, allowing restarts.

```
    static Ran ran_vegas(RANSEED);
```

Initialize a captive, static random number generator.

```
    Int ndim=regn.size()/2;
    if (init <= 0) {
        mds=ndo=1;
        for (j=0;j<ndim;j++) xi[j][0]=1.0;
    }
```

Normal entry. Enter here on a cold start.

Change to mds=0 to disable stratified sampling, i.e., use importance sampling only.

```
    if (init <= 1) si=swgt=schi=0.0;
```

Enter here to inherit the grid from a previous call, but not its answers.

```
    if (init <= 2) {
        nd=NDMX;
        ng=1;
        if (mds != 0) {
            ng=Int(pow(ncall/2.0+0.25,1.0/ndim));
            mds=1;
            if ((2*ng-NDMX) >= 0) {
                mds = -1;
                npg=ng/NDMX+1;
                nd=ng/npg;
                ng=npg*nd;
            }
        }
        for (k=1,i=0;i<ndim;i++) k *= ng;
        npg=MAX(Int(ncall/k),2);
        calls=Doub(npg)*Doub(k);
```

Enter here to inherit the previous grid and its answers.

Set up for stratification.

```
        dxg=1.0/ng;
        for (dv2g=1,i=0;i<ndim;i++) dv2g *= dxg;
        dv2g=SQR(calls*dv2g)/npg/npg/(npg-1.0);
        xnd=nd;
        dxg *= xnd;
        xjac=1.0/calls;
        for (j=0;j<ndim;j++) {
            dx[j]=regn[j+ndim]-regn[j];
            xjac *= dx[j];
        }
        if (nd != ndo) {                Do binning if necessary.
            for (i=0;i<MAX(nd,ndo);i++) r[i]=1.0;
            for (j=0;j<ndim;j++)
                rebin(ndo/xnd,nd,r,xin,xi,j);
            ndo=nd;
        }
        if (nprn >= 0) {
            cout << " Input parameters for vegas";
            cout << "  ndim= " << setw(4) << ndim;
            cout << "  ncall= " << setw(8) << calls << endl;
            cout << setw(34) << "  it=" << setw(5) << it;
            cout << "  itmx=" << setw(5) << itmx << endl;
            cout << setw(34) << "  nprn=" << setw(5) << nprn;
            cout << "  ALPH=" << setw(9) << ALPH << endl;
            cout << setw(34) << "  mds=" << setw(5) << mds;
            cout << "  nd=" << setw(5) << nd << endl;
            for (j=0;j<ndim;j++) {
                cout << setw(30) << " x1[" << setw(2) << j;
                cout << "]= " << setw(11) << regn[j] << " xu[";
                cout << setw(2) << j << "]= ";
                cout << setw(11) << regn[j+ndim] << endl;
            }
        }
    }
    for (it=0;it<itmx;it++) {
```

Main iteration loop. Can enter here (`init` ≥ 3) to do an additional `itmx` iterations with all other parameters unchanged.

```
        ti=tsi=0.0;
        for (j=0;j<ndim;j++) {
            kg[j]=1;
            for (i=0;i<nd;i++) d[i][j]=di[i][j]=0.0;
        }
        for (;;) {
            fb=f2b=0.0;
            for (k=0;k<npg;k++) {
                wgt=xjac;
                for (j=0;j<ndim;j++) {
                    xn=(kg[j]-ran_vegas.doub())*dxg+1.0;
                    ia[j]=MAX(MIN(Int(xn),NDMX),1);
                    if (ia[j] > 1) {
                        xo=xi[j][ia[j]-1]-xi[j][ia[j]-2];
                        rc=xi[j][ia[j]-2]+(xn-ia[j])*xo;
                    } else {
                        xo=xi[j][ia[j]-1];
                        rc=(xn-ia[j])*xo;
                    }
                    x[j]=regn[j]+rc*dx[j];
                    wgt *= xo*xnd;
                }
                f=wgt*fxn(x,wgt);
                f2=f*f;
                fb += f;
                f2b += f2;
                for (j=0;j<ndim;j++) {
```

```
            di[ia[j]-1][j] += f;
            if (mds >= 0) d[ia[j]-1][j] += f2;
        }
    }
    f2b=sqrt(f2b*npg);
    f2b=(f2b-fb)*(f2b+fb);
    if (f2b <= 0.0) f2b=TINY;
    ti += fb;
    tsi += f2b;
    if (mds < 0) {                  Use stratified sampling.
        for (j=0;j<ndim;j++) d[ia[j]-1][j] += f2b;
    }
    for (k=ndim-1;k>=0;k--) {
        kg[k] %= ng;
        if (++kg[k] != 1) break;
    }
    if (k < 0) break;
}
tsi *= dv2g;                        Compute final results for this iteration.
wgt=1.0/tsi;
si += wgt*ti;
schi += wgt*ti*ti;
swgt += wgt;
tgral=si/swgt;
chi2a=(schi-si*tgral)/(it+0.0001);
if (chi2a < 0.0) chi2a = 0.0;
sd=sqrt(1.0/swgt);
tsi=sqrt(tsi);
if (nprn >= 0) {
    cout << " iteration no. " << setw(3) << (it+1);
    cout << " : integral = " << setw(14) << ti;
    cout << " +/- " << setw(9) << tsi << endl;
    cout << " all iterations:  " << " integral =";
    cout << setw(14) << tgral << "+-" << setw(9) << sd;
    cout << " chi**2/IT n =" << setw(9) << chi2a << endl;
    if (nprn != 0) {
        for (j=0;j<ndim;j++) {
            cout << " DATA FOR axis  " << setw(2) << j << endl;
            cout << "     X      delta i          X      delta i";
            cout << "          X       deltai" << endl;
            for (i=nprn/2;i<nd-2;i += nprn+2) {
                cout << setw(8) << xi[j][i] << setw(12) << di[i][j];
                cout << setw(12) << xi[j][i+1] << setw(12) << di[i+1][j];
                cout << setw(12) << xi[j][i+2] << setw(12) << di[i+2][j];
                cout << endl;
            }
        }
    }
}
for (j=0;j<ndim;j++) {              Refine the grid. Consult references to understand
    xo=d[0][j];                         the subtlety of this procedure. The refine-
    xn=d[1][j];                         ment is damped, to avoid rapid, destabiliz-
    d[0][j]=(xo+xn)/2.0;                ing changes, and also compressed in range
    dt[j]=d[0][j];                      by the exponent ALPH.
    for (i=2;i<nd;i++) {
        rc=xo+xn;
        xo=xn;
        xn=d[i][j];
        d[i-1][j] = (rc+xn)/3.0;
        dt[j] += d[i-1][j];
    }
    d[nd-1][j]=(xo+xn)/2.0;
    dt[j] += d[nd-1][j];
}
```

```
        for (j=0;j<ndim;j++) {
            rc=0.0;
            for (i=0;i<nd;i++) {
                if (d[i][j] < TINY) d[i][j]=TINY;
                r[i]=pow((1.0-d[i][j]/dt[j])/
                    (log(dt[j])-log(d[i][j])),ALPH);
                rc += r[i];
            }
            rebin(rc/xnd,nd,r,xin,xi,j);
        }
    }
}
```

rebin.h

```
void rebin(const Doub rc, const Int nd, VecDoub_I &r, VecDoub_O &xin,
    MatDoub_IO &xi, const Int j) {
```

Utility routine used by vegas, to rebin a vector of densities contained in row j of xi into new bins defined by a vector r.

```
    Int i,k=0;
    Doub dr=0.0,xn=0.0,xo=0.0;

    for (i=0;i<nd-1;i++) {
        while (rc > dr)
            dr += r[(++k)-1];
        if (k > 1) xo=xi[j][k-2];
        xn=xi[j][k-1];
        dr -= rc;
        xin[i]=xn-(xn-xo)*dr/r[k-1];
    }
    for (i=0;i<nd-1;i++) xi[j][i]=xin[i];
    xi[j][nd-1]=1.0;
}
```