dominated by *one* of the design parameters, that parameter will be found with this sampling technique. On the other hand, if there is an important interaction among different design parameters, then the Latin hypercube gives no particular advantage. Use with care.

CITED REFERENCES AND FURTHER READING:

Halton, J.H. 1960, *Numerische Mathematik*, vol. 2, pp. 84–90. [1]

Bratley P., and Fox, B.L. 1988, *ACM Transactions on Mathematical Software*, vol. 14, pp. 88–100. [2]

Lambert, J.P. 1988, in *Numerical Mathematics – Singapore 1988*, ISNM vol. 86, R.P. Agarwal, Y.M. Chow, and S.J. Wilson, eds. (Basel: Birkhaüser), pp. 273–284.

Niederreiter, H. 1988, in *Numerical Integration III*, ISNM vol. 85, H. Brass and G. Hämmerlin, eds. (Basel: Birkhaüser), pp. 157–171.

Sobol', I.M. 1967, *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 4, pp. 86–112. [3]

Antonov, I.A., and Saleev, V.M 1979, *USSR Computational Mathematics and Mathematical Physics*, vol. 19, no. 1, pp. 252–256. [4]

Dunn, O.J., and Clark, V.A. 1974, *Applied Statistics: Analysis of Variance and Regression* (New York, Wiley) [discusses Latin Square].

## 7.8 Adaptive and Recursive Monte Carlo Methods

This section discusses more advanced techniques of Monte Carlo integration. As examples of the use of these techniques, we include two rather different, fairly sophisticated, multidimensional Monte Carlo codes: `vegas` [1,2], and `miser` [3]. The techniques that we discuss all fall under the general rubric of *reduction of variance* (§7.6), but are otherwise quite distinct.

### Importance Sampling

The use of *importance sampling* was already implicit in equations (7.6.6) and (7.6.7). We now return to it in a slightly more formal way. Suppose that an integrand $f$ can be written as the product of a function $h$ that is almost constant times another, positive, function $g$. Then its integral over a multidimensional volume $V$ is

$$\int f\,dV = \int (f/g)\,gdV = \int h\,gdV \qquad (7.8.1)$$

In equation (7.6.7) we interpreted equation (7.8.1) as suggesting a change of variable to $G$, the indefinite integral of $g$. That made $gdV$ a perfect differential. We then proceeded to use the basic theorem of Monte Carlo integration, equation (7.6.1). A more general interpretation of equation (7.8.1) is that we can integrate $f$ by instead sampling $h$ — not, however, with uniform probability density $dV$, but rather with nonuniform density $gdV$. In this second interpretation, the first interpretation follows as the special case, where the *means* of generating the nonuniform sampling of $gdV$ is via the transformation method, using the indefinite integral $G$ (see §7.2).

More directly, one can go back and generalize the basic theorem (7.6.1) to the case of nonuniform sampling: Suppose that points $x_i$ are chosen within the volume $V$ with a probability density $p$ satisfying

$$\int p\,dV = 1 \qquad (7.8.2)$$

The generalized fundamental theorem is that the integral of any function $f$ is estimated, using $N$ sample points $x_i, \dots, x_N$, by

$$I \equiv \int f\,dV = \int \frac{f}{p}\,pdV \approx \left\langle \frac{f}{p} \right\rangle \pm \sqrt{\frac{\langle f^2/p^2\rangle - \langle f/p\rangle^2}{N}} \tag{7.8.3}$$

where angle brackets denote arithmetic means over the $N$ points, exactly as in equation (7.6.2). As in equation (7.6.1), the "plus-or-minus" term is a one standard deviation error estimate. Notice that equation (7.6.1) is in fact the special case of equation (7.8.3), with $p = \text{constant} = 1/V$.

What is the best choice for the sampling density $p$? Intuitively, we have already seen that the idea is to make $h = f/p$ as close to constant as possible. We can be more rigorous by focusing on the numerator inside the square root in equation (7.8.3), which is the variance per sample point. Both angle brackets are themselves Monte Carlo estimators of integrals, so we can write

$$S \equiv \left\langle \frac{f^2}{p^2} \right\rangle - \left\langle \frac{f}{p} \right\rangle^2 \approx \int \frac{f^2}{p^2}\,pdV - \left[\int \frac{f}{p}\,pdV\right]^2 = \int \frac{f^2}{p}\,dV - \left[\int f\,dV\right]^2 \tag{7.8.4}$$

We now find the optimal $p$ subject to the constraint equation (7.8.2) by the functional variation

$$0 = \frac{\delta}{\delta p}\left(\int \frac{f^2}{p}\,dV - \left[\int f\,dV\right]^2 + \lambda \int p\,dV\right) \tag{7.8.5}$$

with $\lambda$ a Lagrange multiplier. Note that the middle term does not depend on $p$. The variation (which comes inside the integrals) gives $0 = -f^2/p^2 + \lambda$ or

$$p = \frac{|f|}{\sqrt{\lambda}} = \frac{|f|}{\int |f|\,dV} \tag{7.8.6}$$

where $\lambda$ has been chosen to enforce the constraint (7.8.2).

If $f$ has one sign in the region of integration, then we get the obvious result that the optimal choice of $p$ — if one can figure out a practical way of effecting the sampling — is that it be proportional to $|f|$. Then the variance is reduced to zero. Not so obvious, but seen to be true, is the fact that $p \propto |f|$ is optimal even if $f$ takes on both signs. In that case the variance per sample point (from equations 7.8.4 and 7.8.6) is

$$S = S_{\text{optimal}} = \left(\int |f|\,dV\right)^2 - \left(\int f\,dV\right)^2 \tag{7.8.7}$$

One curiosity is that one can add a constant to the integrand to make it all of one sign, since this changes the integral by a known amount, constant $\times\ V$. Then, the optimal choice of $p$ always gives zero variance, that is, a perfectly accurate integral! The resolution of this seeming paradox (already mentioned at the end of §7.6) is that perfect knowledge of $p$ in equation (7.8.6) requires perfect knowledge of $\int |f| dV$, which is tantamount to already knowing the integral you are trying to compute!

If your function $f$ takes on a known constant value in most of the volume $V$, it is certainly a good idea to add a constant so as to make that value zero. Having done that, the accuracy attainable by importance sampling depends in practice not on how small equation (7.8.7) is, but rather on how small is equation (7.8.4) for an *implementable* $p$, likely only a crude approximation to the ideal.

## Stratified Sampling

The idea of *stratified sampling* is quite different from importance sampling. Let us expand our notation slightly and let $\langle\langle f\rangle\rangle$ denote the true average of the function $f$ over the volume $V$ (namely the integral divided by $V$), while $\langle f\rangle$ denotes as before the simplest (uniformly sampled) Monte Carlo *estimator* of that average:

$$\langle\langle f\rangle\rangle \equiv \frac{1}{V}\int f\,dV \qquad \langle f\rangle \equiv \frac{1}{N}\sum_i f(x_i) \tag{7.8.8}$$

The variance of the estimator, $\text{Var}\,(\langle f\rangle)$, which measures the square of the error of the Monte Carlo integration, is asymptotically related to the variance of the function, $\text{Var}\,(f) \equiv \langle\langle f^2\rangle\rangle - \langle\langle f\rangle\rangle^2$, by the relation

$$\text{Var}\,(\langle f\rangle) = \frac{\text{Var}\,(f)}{N} \tag{7.8.9}$$

(compare equation 7.6.1).

Suppose we divide the volume $V$ into two equal, disjoint subvolumes, denoted $a$ and $b$, and sample $N/2$ points in each subvolume. Then another estimator for $\langle\langle f\rangle\rangle$, different from equation (7.8.8), which we denote $\langle f\rangle'$, is

$$\langle f\rangle' \equiv \frac{1}{2}\left(\langle f\rangle_a + \langle f\rangle_b\right) \tag{7.8.10}$$

in other words, the mean of the sample averages in the two half-regions. The variance of estimator (7.8.10) is given by

$$\begin{aligned}\text{Var}\left(\langle f\rangle'\right) &= \frac{1}{4}\left[\text{Var}\left(\langle f\rangle_a\right) + \text{Var}\left(\langle f\rangle_b\right)\right]\\ &= \frac{1}{4}\left[\frac{\text{Var}_a\,(f)}{N/2} + \frac{\text{Var}_b\,(f)}{N/2}\right]\\ &= \frac{1}{2N}\left[\text{Var}_a\,(f) + \text{Var}_b\,(f)\right]\end{aligned} \tag{7.8.11}$$

Here $\text{Var}_a\,(f)$ denotes the variance of $f$ in subregion $a$, that is, $\langle\langle f^2\rangle\rangle_a - \langle\langle f\rangle\rangle_a^2$, and correspondingly for $b$.

From the definitions already given, it is not difficult to prove the relation

$$\text{Var}\,(f) = \frac{1}{2}\left[\text{Var}_a\,(f) + \text{Var}_b\,(f)\right] + \frac{1}{4}\left(\langle\langle f\rangle\rangle_a - \langle\langle f\rangle\rangle_b\right)^2 \tag{7.8.12}$$

(In physics, this formula for combining second moments is the "parallel axis theorem.") Comparing equations (7.8.9), (7.8.11), and (7.8.12), one sees that the stratified (into two subvolumes) sampling gives a variance that is never larger than the simple Monte Carlo case — and smaller whenever the means of the stratified samples, $\langle\langle f\rangle\rangle_a$ and $\langle\langle f\rangle\rangle_b$, are different.

We have not yet exploited the possibility of sampling the two subvolumes with *different numbers* of points, say $N_a$ in subregion $a$ and $N_b \equiv N - N_a$ in subregion $b$. Let us do so now. Then the variance of the estimator is

$$\text{Var}\left(\langle f\rangle'\right) = \frac{1}{4}\left[\frac{\text{Var}_a\,(f)}{N_a} + \frac{\text{Var}_b\,(f)}{N - N_a}\right] \tag{7.8.13}$$

which is minimized (one can easily verify) when

$$\frac{N_a}{N} = \frac{\sigma_a}{\sigma_a + \sigma_b} \tag{7.8.14}$$

Here we have adopted the shorthand notation $\sigma_a \equiv \left[\text{Var}_a\,(f)\right]^{1/2}$, and correspondingly for $b$. If $N_a$ satisfies equation (7.8.14), then equation (7.8.13) reduces to

$$\text{Var}\left(\langle f\rangle'\right) = \frac{(\sigma_a + \sigma_b)^2}{4N} \tag{7.8.15}$$

Equation (7.8.15) reduces to equation (7.8.9) if $\text{Var}(f) = \text{Var}_a(f) = \text{Var}_b(f)$, in which case stratifying the sample makes no difference.

A standard way to generalize the above result is to consider the volume $V$ divided into more than two equal subregions. One can readily obtain the result that the optimal allocation of sample points among the regions is to have the number of points in each region $j$ proportional to $\sigma_j$ (that is, the square root of the variance of the function $f$ in that subregion). In spaces of high dimensionality (say $d \gtrsim 4$) this is not in practice very useful, however. Dividing a volume into $K$ segments along each dimension implies $K^d$ subvolumes, typically much too large a number when one contemplates estimating all the corresponding $\sigma_j$'s.

## Mixed Strategies

Importance sampling and stratified sampling seem, at first sight, inconsistent with each other. The former concentrates sample points where the magnitude of the integrand $|f|$ is largest, that latter where the variance of $f$ is largest. How can both be right?

The answer is that (like so much else in life) it all depends on what you know and how well you know it. Importance sampling depends on already knowing some approximation to your integral, so that you are able to generate random points $x_i$ with the desired probability density $p$. To the extent that your $p$ is not ideal, you are left with an error that decreases only as $N^{-1/2}$. Things are particularly bad if your $p$ is far from ideal in a region where the integrand $f$ is changing rapidly, since then the sampled function $h = f/p$ will have a large variance. Importance sampling works by smoothing the values of the sampled function $h$, and is effective only to the extent that you succeed in this.

Stratified sampling, by contrast, does not necessarily require that you know anything about $f$. Stratified sampling works by smoothing out the fluctuations of the *number* of points in subregions, not by smoothing the values of the points. The simplest stratified strategy, dividing $V$ into $N$ equal subregions and choosing one point randomly in each subregion, already gives a method whose error decreases asymptotically as $N^{-1}$, much faster than $N^{-1/2}$. (Note that quasi-random numbers, §7.7, are another way of smoothing fluctuations in the density of points, giving nearly as good a result as the "blind" stratification strategy.)

However, "asymptotically" is an important caveat: For example, if the integrand is negligible in all but a single subregion, then the resulting one-sample integration is all but useless. Information, even very crude, allowing importance sampling to put many points in the active subregion would be much better than blind stratified sampling.

Stratified sampling really comes into its own if you have some way of estimating the variances, so that you can put unequal numbers of points in different subregions, according to (7.8.14) or its generalizations, *and* if you can find a way of dividing a region into a practical number of subregions (notably *not* $K^d$ with large dimension $d$), while yet significantly reducing the variance of the function in each subregion compared to its variance in the full volume. Doing this requires a lot of knowledge about $f$, though different knowledge from what is required for importance sampling.

In practice, importance sampling and stratified sampling are not incompatible. In many, if not most, cases of interest, the integrand $f$ is small everywhere in $V$ except for a small fractional volume of "active regions." In these regions the magnitude of $|f|$ and the standard deviation $\sigma = [\text{Var}(f)]^{1/2}$ are comparable in size, so both techniques will give about the same concentration of points. In more sophisticated implementations, it is also possible to "nest" the two techniques, so that (e.g.) importance sampling on a crude grid is followed by stratification within each grid cell.

## Adaptive Monte Carlo: VEGAS

The VEGAS algorithm, invented by Peter Lepage [1,2], is widely used for multidimensional integrals that occur in elementary particle physics. VEGAS is primarily based on importance sampling, but it also does some stratified sampling if the dimension $d$ is small enough to avoid $K^d$ explosion (specifically, if $(K/2)^d < N/2$, with $N$ the number of sample points). The basic technique for importance sampling in VEGAS is to construct, adaptively, a multidimensional weight function $g$ that is *separable*,

$$p \propto g(x, y, z, \ldots) = g_x(x)g_y(y)g_z(z)\ldots \tag{7.8.16}$$

Such a function avoids the $K^d$ explosion in two ways: (i) It can be stored in the computer as $d$ separate one-dimensional functions, each defined by $K$ tabulated values, say — so that $K \times d$ replaces $K^d$. (ii) It can be sampled as a probability density by consecutively sampling the $d$ one-dimensional functions to obtain coordinate vector components $(x, y, z, \ldots)$.

The optimal separable weight function can be shown to be [1]

$$g_x(x) \propto \left[\int dy \int dz \ldots \frac{f^2(x, y, z, \ldots)}{g_y(y)g_z(z)\ldots}\right]^{1/2} \tag{7.8.17}$$

(and correspondingly for $y, z, \ldots$). Notice that this reduces to $g \propto |f|$ (7.8.6) in one dimension. Equation (7.8.17) immediately suggests VEGAS' adaptive strategy: Given a set of $g$-functions (initially all constant, say), one samples the function $f$, accumulating not only the overall estimator of the integral, but also the $Kd$ estimators ($K$ subdivisions of the independent variable in each of $d$ dimensions) of the right-hand side of equation (7.8.17). These then determine improved $g$ functions for the next iteration.

When the integrand $f$ is concentrated in one, or at most a few, regions in $d$-space, then the weight function $g$'s quickly become large at coordinate values that are the projections of these regions onto the coordinate axes. The accuracy of the Monte Carlo integration is then enormously enhanced over what simple Monte Carlo would give.

The weakness of VEGAS is the obvious one: To the extent that the projection of the function $f$ onto individual coordinate directions is uniform, VEGAS gives no concentration of sample points in those dimensions. The worst case for VEGAS, e.g., is an integrand that is concentrated close to a body diagonal line, e.g., one from $(0, 0, 0, \ldots)$ to $(1, 1, 1, \ldots)$. Since this geometry is completely nonseparable, VEGAS can give no advantage at all. More generally, VEGAS may not do well when the integrand is concentrated in one-dimensional (or higher) curved trajectories (or hypersurfaces), unless these happen to be oriented close to the coordinate directions.

The routine `vegas` that follows is essentially Lepage's standard version, minimally modified to conform to our conventions. (We thank Lepage for permission to reproduce the program here.) For consistency with other versions of the VEGAS algorithm in circulation, we have preserved original variable names. The parameter `NDMX` is what we have called $K$, the maximum number of increments along each axis; `MXDIM` is the maximum value of $d$; some other parameters are explained in the comments.

The `vegas` routine performs $m =$ `itmx` statistically independent evaluations of the desired integral, each with $N =$ `ncall` function evaluations. While statistically independent, these iterations do assist each other, since each one is used to refine the sampling grid for the next one. The results of all iterations are combined into a single best answer, and its estimated error, by the relations

$$I_{\rm best} = \sum_{i=1}^{m} \frac{I_i}{\sigma_i^2} \Bigg/ \sum_{i=1}^{m} \frac{1}{\sigma_i^2} \qquad \sigma_{\rm best} = \left(\sum_{i=1}^{m} \frac{1}{\sigma_i^2}\right)^{-1/2} \tag{7.8.18}$$

Also returned is the quantity

$$\chi^2/m \equiv \frac{1}{m-1} \sum_{i=1}^{m} \frac{(I_i - I_{\rm best})^2}{\sigma_i^2} \tag{7.8.19}$$

If this is significantly larger than 1, then the results of the iterations are statistically inconsistent, and the answers are suspect.

The input flag `init` can be used to advantage. One might have a call with `init=0`, `ncall=1000`, `itmx=5` immediately followed by a call with `init=1`, `ncall=100000`, `itmx=1`. The effect would be to develop a sampling grid over 5 iterations of a small number of samples, then to do a single high accuracy integration on the optimized grid.

Note that the user-supplied integrand function, `fxn`, has an argument `wgt` in addition to the expected evaluation point `x`. In most applications you ignore `wgt` inside the function. Occasionally, however, you may want to integrate some additional function or functions along with the principal function $f$. The integral of any such function $g$ can be estimated by

$$I_g = \sum_i w_i g(\mathbf{x}) \tag{7.8.20}$$

where the $w_i$'s and $\mathbf{x}$'s are the arguments `wgt` and `x`, respectively. It is straightforward to accumulate this sum inside your function `fxn`, and to pass the answer back to your main program via a common block. Of course, $g(\mathbf{x})$ had better resemble the principal function $f$ to some degree, since the sampling will be optimized for $f$.

```
      SUBROUTINE vegas(region,ndim,fxn,init,ncall,itmx,nprn,
     *     tgral,sd,chi2a)
      INTEGER init,itmx,ncall,ndim,nprn,NDMX,MXDIM
      REAL tgral,chi2a,sd,region(2*ndim),fxn,ALPH,TINY
      PARAMETER (ALPH=1.5,NDMX=50,MXDIM=10,TINY=1.e-30)
      EXTERNAL fxn
C     USES fxn,ran2,rebin
```

Performs Monte Carlo integration of a user-supplied `ndim`-dimensional function `fxn` over a rectangular volume specified by `region`, a 2×`ndim` 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 below). 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` ($\chi^2$ per degree of freedom, an indicator of whether consistent results are being obtained). See text for further details.

```
      INTEGER i,idum,it,j,k,mds,nd,ndo,ng,npg,ia(MXDIM),kg(MXDIM)
      REAL calls,dv2g,dxg,f,f2,f2b,fb,rc,ti,tsi,wgt,xjac,xn,xnd,xo,
     *     d(NDMX,MXDIM),di(NDMX,MXDIM),dt(MXDIM),dx(MXDIM),
     *     r(NDMX),x(MXDIM),xi(NDMX,MXDIM),xin(NDMX),ran2
      DOUBLE PRECISION schi,si,swgt
      COMMON /ranno/ idum          Means for random number initialization.
      SAVE                         Best make everything static, allowing restarts.
      if(init.le.0)then            Normal entry. Enter here on a cold start.
        mds=1                      Change to mds=0 to disable stratified sampling, i.e., use im-
        ndo=1                          portance sampling only.
        do 11 j=1,ndim
          xi(1,j)=1.
        enddo 11
      endif
      if (init.le.1)then           Enter here to inherit the grid from a previous call, but not its
        si=0.                          answers.
        swgt=0.
        schi=0.
      endif
      if (init.le.2)then           Enter here to inherit the previous grid and its answers.
        nd=NDMX
        ng=1
        if(mds.ne.0)then           Set up for stratification.
          ng=(ncall/2.+0.25)**(1./ndim)
          mds=1
          if((2*ng-NDMX).ge.0)then
            mds=-1
            npg=ng/NDMX+1
            nd=ng/npg
            ng=npg*nd
          endif
        endif
        k=ng**ndim
```

```
          npg=max(ncall/k,2)
          calls=float(npg)*float(k)
          dxg=1./ng
          dv2g=(calls*dxg**ndim)**2/npg/npg/(npg-1.)
          xnd=nd
          dxg=dxg*xnd
          xjac=1./calls
          do 12 j=1,ndim
              dx(j)=region(j+ndim)-region(j)
              xjac=xjac*dx(j)
          enddo 12
          if(nd.ne.ndo)then            Do binning if necessary.
              do 13 i=1,max(nd,ndo)
                  r(i)=1.
              enddo 13
              do 14 j=1,ndim
                  call rebin(ndo/xnd,nd,r,xin,xi(1,j))
              enddo 14
              ndo=nd
          endif
          if(nprn.ge.0) write(*,200) ndim,calls,it,itmx,nprn,
*              ALPH,mds,nd,(j,region(j),j,region(j+ndim),j=1,ndim)
      endif
      do 28 it=1,itmx
        Main iteration loop. Can enter here (init ≥ 3) to do an additional itmx iterations with all
        other parameters unchanged.
          ti=0.
          tsi=0.
          do 16 j=1,ndim
              kg(j)=1
              do 15 i=1,nd
                  d(i,j)=0.
                  di(i,j)=0.
              enddo 15
          enddo 16
10        continue
              fb=0.
              f2b=0.
              do 19 k=1,npg
                  wgt=xjac
                  do 17 j=1,ndim
                      xn=(kg(j)-ran2(idum))*dxg+1.
                      ia(j)=max(min(int(xn),NDMX),1)
                      if(ia(j).gt.1)then
                          xo=xi(ia(j),j)-xi(ia(j)-1,j)
                          rc=xi(ia(j)-1,j)+(xn-ia(j))*xo
                      else
                          xo=xi(ia(j),j)
                          rc=(xn-ia(j))*xo
                      endif
                      x(j)=region(j)+rc*dx(j)
                      wgt=wgt*xo*xnd
                  enddo 17
                  f=wgt*fxn(x,wgt)
                  f2=f*f
                  fb=fb+f
                  f2b=f2b+f2
                  do 18 j=1,ndim
                      di(ia(j),j)=di(ia(j),j)+f
                      if(mds.ge.0) d(ia(j),j)=d(ia(j),j)+f2
                  enddo 18
              enddo 19
              f2b=sqrt(f2b*npg)
              f2b=(f2b-fb)*(f2b+fb)
          if (f2b.le.0.) f2b=TINY
          ti=ti+fb
          tsi=tsi+f2b
          if(mds.lt.0)then      Use stratified sampling.
            do 21 j=1,ndim
              d(ia(j),j)=d(ia(j),j)+f2b
            enddo 21
          endif
        do 22 k=ndim,1,-1
          kg(k)=mod(kg(k),ng)+1
          if(kg(k).ne.1) goto 10
        enddo 22
        tsi=tsi*dv2g              Compute final results for this iteration.
        wgt=1./tsi
        si=si+dble(wgt)*dble(ti)
        schi=schi+dble(wgt)*dble(ti)**2
        swgt=swgt+dble(wgt)
        tgral=si/swgt
        chi2a=max((schi-si*tgral)/(it-.99d0),0.d0)
        sd=sqrt(1./swgt)
        tsi=sqrt(tsi)
        if(nprn.ge.0)then
          write(*,201) it,ti,tsi,tgral,sd,chi2a
          if(nprn.ne.0)then
            do 23 j=1,ndim
              write(*,202) j,(xi(i,j),di(i,j),
*                 i=1+nprn/2,nd,nprn)
            enddo 23
          endif
        endif
        do 25 j=1,ndim            Refine the grid. Consult references to understand the subtlety
          xo=d(1,j)                  of this procedure. The refinement is damped, to avoid
          xn=d(2,j)                  rapid, destabilizing changes, and also compressed in range
          d(1,j)=(xo+xn)/2.          by the exponent ALPH.
          dt(j)=d(1,j)
          do 24 i=2,nd-1
            rc=xo+xn
            xo=xn
            xn=d(i+1,j)
            d(i,j)=(rc+xn)/3.
            dt(j)=dt(j)+d(i,j)
          enddo 24
          d(nd,j)=(xo+xn)/2.
          dt(j)=dt(j)+d(nd,j)
        enddo 25
        do 27 j=1,ndim
          rc=0.
          do 26 i=1,nd
            if(d(i,j).lt.TINY) d(i,j)=TINY
            r(i)=((1.-d(i,j)/dt(j))/(log(dt(j))-log(d(i,j))))**ALPH
            rc=rc+r(i)
          enddo 26
          call rebin(rc/xnd,nd,r,xin,xi(1,j))
        enddo 27
      enddo 28
      return
200 FORMAT(/' input parameters for vegas: ndim=',i3,' ncall=',f8.0
*       /28x,' it=',i5,' itmx=',i5
*       /28x,' nprn=',i3,' alph=',f5.2/28x,' mds=',i3,' nd=',i4
*       /(30x,'xl(',i2,')= ',g11.4,' xu(',i2,')= ',g11.4))
201 FORMAT(/' iteration no.',I3,': ','integral =',g14.7,'+/- ',g9.2
*       /' all iterations: integral =',g14.7,'+/- ',g9.2,
*       ' chi**2/it''n =',g9.2)
202 FORMAT(/' data for axis ',I2/' X delta i ',
*        ' x delta i ',' x delta i ',
*        /(1x,f7.5,1x,g11.4,5x,f7.5,1x,g11.4,5x,f7.5,1x,g11.4))
      END
```

```
      SUBROUTINE rebin(rc,nd,r,xin,xi)
      INTEGER nd
      REAL rc,r(*),xi(*),xin(*)
         Utility routine used by vegas, to rebin a vector of densities xi into new bins defined by
         a vector r.
      INTEGER i,k
      REAL dr,xn,xo
      k=0
      xo=0.
      dr=0.
      do 11 i=1,nd-1
1         if(rc.gt.dr)then
              k=k+1
              dr=dr+r(k)
          goto 1
          endif
          if(k.gt.1) xo=xi(k-1)
          xn=xi(k)
          dr=dr-rc
          xin(i)=xn-(xn-xo)*dr/r(k)
      enddo 11
      do 12 i=1,nd-1
          xi(i)=xin(i)
      enddo 12
      xi(nd)=1.
      return
      END
```

## Recursive Stratified Sampling

The problem with stratified sampling, we have seen, is that it may not avoid the $K^d$ explosion inherent in the obvious, Cartesian, tesselation of a $d$-dimensional volume. A technique called *recursive stratified sampling* [3] attempts to do this by successive bisections of a volume, not along all $d$ dimensions, but rather along only one dimension at a time. The starting points are equations (7.8.10) and (7.8.13), applied to bisections of successively smaller subregions.

Suppose that we have a quota of $N$ evaluations of the function $f$, and want to evaluate $\langle f\rangle'$ in the rectangular parallelepiped region $R = (\mathbf{x}_a, \mathbf{x}_b)$. (We denote such a region by the two coordinate vectors of its diagonally opposite corners.) First, we allocate a fraction $p$ of $N$ towards exploring the variance of $f$ in $R$: We sample $pN$ function values uniformly in $R$ and accumulate the sums that will give the $d$ different pairs of variances corresponding to the $d$ different coordinate directions along which $R$ can be bisected. In other words, in $pN$ samples, we estimate Var$\,(f)$ in each of the regions resulting from a possible bisection of $R$,

$$\begin{aligned} R_{ai} &\equiv (\mathbf{x}_a, \mathbf{x}_b - \frac{1}{2}\mathbf{e}_i \cdot (\mathbf{x}_b - \mathbf{x}_a)\mathbf{e}_i) \\ R_{bi} &\equiv (\mathbf{x}_a + \frac{1}{2}\mathbf{e}_i \cdot (\mathbf{x}_b - \mathbf{x}_a)\mathbf{e}_i, \mathbf{x}_b) \end{aligned} \tag{7.8.21}$$

Here $\mathbf{e}_i$ is the unit vector in the $i$th coordinate direction, $i = 1, 2, \ldots, d$.

Second, we inspect the variances to find the most favorable dimension $i$ to bisect. By equation (7.8.15), we could, for example, choose that $i$ for which the sum of the square roots of the variance estimators in regions $R_{ai}$ and $R_{bi}$ is minimized. (Actually, as we will explain, we do something slightly different.)

Third, we allocate the remaining $(1-p)N$ function evaluations between the regions $R_{ai}$ and $R_{bi}$. If we used equation (7.8.15) to choose $i$, we should do this allocation according to equation (7.8.14).

We now have two parallelepipeds each with its own allocation of function evaluations for estimating the mean of $f$. Our "RSS" algorithm now shows itself to be *recursive*: To evaluate the mean in each region, we go back to the sentence beginning "First,..." in the paragraph above equation (7.8.21). (Of course, when the allocation of points to a region falls below some number, we resort to simple Monte Carlo rather than continue with the recursion.)

Finally, we combine the means, and also estimated variances of the two subvolumes, using equation (7.8.10) and the first line of equation (7.8.11).

This completes the RSS algorithm in its simplest form. Before we describe some additional tricks under the general rubric of "implementation details," we need to return briefly to equations (7.8.13)–(7.8.15) and derive the equations that we actually use instead of these. The right-hand side of equation (7.8.13) applies the familiar scaling law of equation (7.8.9) twice, once to $a$ and again to $b$. This would be correct if the estimates $\langle f\rangle_a$ and $\langle f\rangle_b$ were each made by simple Monte Carlo, with uniformly random sample points. However, the two estimates of the mean are in fact made recursively. Thus, there is no reason to expect equation (7.8.9) to hold. Rather, we might substitute for equation (7.8.13) the relation,

$$\mathrm{Var}\left(\langle f\rangle'\right)=\frac{1}{4}\left[\frac{\mathrm{Var}_a\left(f\right)}{N_a^\alpha}+\frac{\mathrm{Var}_b\left(f\right)}{(N-N_a)^\alpha}\right] \tag{7.8.22}$$

where $\alpha$ is an unknown constant $\geq 1$ (the case of equality corresponding to simple Monte Carlo). In that case, a short calculation shows that $\mathrm{Var}\left(\langle f\rangle'\right)$ is minimized when

$$\frac{N_a}{N}=\frac{\mathrm{Var}_a\left(f\right)^{1/(1+\alpha)}}{\mathrm{Var}_a\left(f\right)^{1/(1+\alpha)}+\mathrm{Var}_b\left(f\right)^{1/(1+\alpha)}} \tag{7.8.23}$$

and that its minimum value is

$$\mathrm{Var}\left(\langle f\rangle'\right)\propto\left[\mathrm{Var}_a\left(f\right)^{1/(1+\alpha)}+\mathrm{Var}_b\left(f\right)^{1/(1+\alpha)}\right]^{1+\alpha} \tag{7.8.24}$$

Equations (7.8.22)–(7.8.24) reduce to equations (7.8.13)–(7.8.15) when $\alpha=1$. Numerical experiments to find a self-consistent value for $\alpha$ find that $\alpha\approx 2$. That is, when equation (7.8.23) with $\alpha=2$ is used recursively to allocate sample opportunities, the observed variance of the RSS algorithm goes approximately as $N^{-2}$, while any other value of $\alpha$ in equation (7.8.23) gives a poorer fall-off. (The sensitivity to $\alpha$ is, however, not very great; it is not known whether $\alpha=2$ is an analytically justifiable result, or only a useful heuristic.)

Turn now to the routine, `miser`, which implements the RSS method. A bit of `FORTRAN` wizardry is its implementation of the required recursion. This is done by dimensioning an array `stack`, and a shorter "stack frame" `stf`; the latter has components that are equivalenced to variables that need to be preserved during the recursion, including a flag indicating where program control should return. A recursive call then consists of copying the stack frame onto the stack, incrementing the stack pointer `jstack`, and transferring control. A recursive return analogously pops the stack and transfers control to the saved location. Stack growth in `miser` is only logarithmic in $N$, since at each bifurcation one of the subvolumes can be processed immediately.

The principal difference between `miser`'s implementation and the algorithm as described thus far lies in how the variances on the right-hand side of equation (7.8.23) are estimated. We find empirically that it is somewhat more robust to use the square of the difference of maximum and minimum sampled function values, instead of the genuine second moment of the samples. This estimator is of course increasingly biased with increasing sample size; however, equation (7.8.23) uses it only to compare two subvolumes ($a$ and $b$) having approximately equal numbers of samples. The "max minus min" estimator proves its worth when the preliminary sampling yields only a single point, or small number of points, in active regions of the integrand. In many realistic cases, these are indicators of nearby regions of even greater importance, and it is useful to let them attract the greater sampling weight that "max minus min" provides.

A second modification embodied in the code is the introduction of a “dithering parameter,” dith, whose nonzero value causes subvolumes to be divided not exactly down the middle, but rather into fractions $0.5\pm$dith, with the sign of the $\pm$ randomly chosen by a built-in random number routine. Normally dith can be set to zero. However, there is a large advantage in taking dith to be nonzero if some special symmetry of the integrand puts the active region exactly at the midpoint of the region, or at the center of some power-of-two submultiple of the region. One wants to avoid the extreme case of the active region being evenly divided into $2^d$ abutting corners of a $d$-dimensional space. A typical nonzero value of dith, on those occasions when it is useful, might be $0.1$. Of course, when the dithering parameter is nonzero, we must take the differing sizes of the subvolumes into account; the code does this through the variable fracl.

One final feature in the code deserves mention. The RSS algorithm uses a single set of sample points to evaluate equation (7.8.23) in all $d$ directions. At bottom levels of the recursion, the number of sample points can be quite small. Although rare, it can happen that in one direction all the samples are in one half of the volume; in that case, that direction is ignored as a candidate for bifurcation. Even more rare is the possibility that all of the samples are in one half of the volume in *all* directions. In this case, a random direction is chosen. If this happens too often in your application, then you should increase MNPT (see line if (jb.eq.0)... in the code).

Note that miser, as given, returns as ave an estimate of the average function value $\langle\langle f\rangle\rangle$, not the integral of $f$ over the region. The routine vegas, adopting the other convention, returns as tgral the integral. The two conventions are of course trivially related, by equation (7.8.8), since the volume $V$ of the rectangular region is known.

```
      SUBROUTINE miser(func,region,ndim,npts,dith,ave,var)
      INTEGER ndim,npts,MNPT,MNBS,MAXD,NSTACK
      REAL ave,dith,var,region(2*ndim),func,TINY,BIG,PFAC
      PARAMETER (MNPT=15,MNBS=4*MNPT,MAXD=10,TINY=1.e-30,BIG=1.e30,
     *     NSTACK=1000,PFAC=0.1)
      EXTERNAL func
C     USES func,ranpt
         Monte Carlo samples a user-supplied ndim-dimensional function func in a rectangular
         volume specified by region, a 2×ndim 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.
         Parameters: 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. MAXD is the largest value of ndim. NSTACK is the total size of the stack.
      INTEGER iran,j,jb,jstack,n,naddr,np,npre,nptl,nptr,nptt
      REAL avel,fracl,fval,rgl,rgm,rgr,s,sigl,siglb,sigr,sigrb,
     *     sum,sumb,summ,summ2,varl,fmaxl(10),fmaxr(10),fminl(10),
     *     fminr(10),pt(10),rmid(10),stack(NSTACK),stf(9)
      EQUIVALENCE (stf(1),avel),(stf(2),varl),(stf(3),jb),
     *     (stf(4),nptr),(stf(5),naddr),(stf(6),rgl),(stf(7),rgm),
     *     (stf(8),rgr),(stf(9),fracl)
      SAVE iran
      DATA iran /0/
      jstack=0
      nptt=npts
1     continue
      if (nptt.lt.MNBS) then        Too few points to bisect; do straight Monte Carlo.
          np=abs(nptt)
          summ=0.
          summ2=0.
          do 11 n=1,np
              call ranpt(pt,region,ndim)
```

```
            fval=func(pt)
            summ=summ+fval
            summ2=summ2+fval**2
        enddo 11
        ave=summ/np
        var=max(TINY,(summ2-summ**2/np)/np**2)
    else                            Do the preliminary (uniform) sampling.
        npre=max(int(nptt*PFAC),MNPT)
        do 12 j=1,ndim              Initialize the left and right bounds for each dimension.
            iran=mod(iran*2661+36979,175000)
            s=sign(dith,float(iran-87500))
            rmid(j)=(0.5+s)*region(j)+(0.5-s)*region(j+ndim)
            fminl(j)=BIG
            fminr(j)=BIG
            fmaxl(j)=-BIG
            fmaxr(j)=-BIG
        enddo 12
        do 14 n=1,npre              Loop over the points in the sample.
            call ranpt(pt,region,ndim)
            fval=func(pt)
            do 13 j=1,ndim          Find the left and right bounds for each dimension.
                if(pt(j).le.rmid(j))then
                    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)
                endif
            enddo 13
        enddo 14
        sumb=BIG                    Choose which dimension jb to bisect.
        jb=0
        siglb=1.
        sigrb=1.
        do 15 j=1,ndim
            if(fmaxl(j).gt.fminl(j).and.fmaxr(j).gt.fminr(j))then
                sigl=max(TINY,(fmaxl(j)-fminl(j))**(2./3.))
                sigr=max(TINY,(fmaxr(j)-fminr(j))**(2./3.))
                sum=sigl+sigr       Equation (7.8.24), see text.
                if (sum.le.sumb) then
                    sumb=sum
                    jb=j
                    siglb=sigl
                    sigrb=sigr
                endif
            endif
        enddo 15
        if (jb.eq.0) jb=1+(ndim*iran)/175000       MNPT may be too small.
        rgl=region(jb)              Apportion the remaining points between left and right.
        rgm=rmid(jb)
        rgr=region(jb+ndim)
        fracl=abs((rgm-rgl)/(rgr-rgl))
        nptl=MNPT+(nptt-npre-2*MNPT)
*           *fracl*siglb/(fracl*siglb+(1.-fracl)*sigrb)         Equation (7.8.23).
        nptr=nptt-npre-nptl
        region(jb+ndim)=rgm         Set region to left.
        naddr=1                     Push the stack.
        do 16 j=1,9
            stack(jstack+j)=stf(j)
        enddo 16
        jstack=jstack+9
        nptt=nptl
        goto 1                      Dispatch recursive call; will return back here eventually.
10      continue
```

```
          avel=ave                    Save left estimates on stack variable.
          varl=var
          region(jb)=rgm              Set region to right.
          region(jb+ndim)=rgr
          naddr=2                     Push the stack.
          do 17 j=1,9
              stack(jstack+j)=stf(j)
          enddo 17
          jstack=jstack+9
          nptt=nptr
          goto 1                      Dispatch recursive call; will return back here eventually.
20        continue
          region(jb)=rgl                    Restore region to original value (so that we don't
          ave=fracl*avel+(1.-fracl)*ave       need to include it on the stack).
          var=fracl**2*varl+(1.-fracl)**2*var       Combine left and right regions by equa-
      endif                                           tion (7.8.11) (1st line).
      if (jstack.ne.0) then           Pop the stack.
          jstack=jstack-9
          do 18 j=1,9
              stf(j)=stack(jstack+j)
          enddo 18
          goto (10,20),naddr
          pause 'miser: never get here'
      endif
      return
      END
```

The miser routine calls a short subroutine ranpt to get a random point within a specified $d$-dimensional region. The following version of ranpt makes consecutive calls to a uniform random number generator and does the obvious scaling. One can easily modify ranpt to generate its points via the quasi-random routine sobseq (§7.7). We find that miser with sobseq can be considerably more accurate than miser with uniform random deviates. Since the use of RSS and the use of quasi-random numbers are completely separable, however, we have not made the code given here dependent on sobseq. A similar remark might be made regarding importance sampling, which could in principle be combined with RSS. (One could in principle combine vegas and miser, although the programming would be intricate.)

```
      SUBROUTINE ranpt(pt,region,n)
      INTEGER n,idum
      REAL pt(n),region(2*n)
      COMMON /ranno/ idum
      SAVE /ranno/
C     USES ran1
          Returns a uniformly random point pt in an n-dimensional rectangular region. Used by
          miser; calls ran1 for uniform deviates. Your main program should initialize idum, through
          the COMMON block /ranno/, to a negative seed integer.
      INTEGER j
      REAL ran1
      do 11 j=1,n
          pt(j)=region(j)+(region(j+n)-region(j))*ran1(idum)
      enddo 11
      return
      END
```

CITED REFERENCES AND FURTHER READING:

Hammersley, J.M. and Handscomb, D.C. 1964, *Monte Carlo Methods* (London: Methuen).

Kalos, M.H. and Whitlock, P.A. 1986, *Monte Carlo Methods* (New York: Wiley).

Bratley, P., Fox, B.L., and Schrage, E.L. 1983, *A Guide to Simulation* (New York: Springer-Verlag).

Lepage, G.P. 1978, *Journal of Computational Physics*, vol. 27, pp. 192–203. [1]

Lepage, G.P. 1980, “VEGAS: An Adaptive Multidimensional Integration Program,” Publication CLNS-80/447, Cornell University. [2]

Press, W.H., and Farrar, G.R. 1990, *Computers in Physics*, vol. 4, pp. 190–195. [3]