arXiv:1009.4647v2 [hep-ph] 28 Oct 2010

Parameterized Adaptive Multidimensional Integration Routines (PAMIR) Localization by Repeated 2^p Subdivision

Stephen L. Adler[*]

Institute for Advanced Study, Einstein Drive, Princeton, NJ 08540, USA.

© 2010 Stephen L. Adler

Contents

[*]Electronic address: adler@ias.edu

I. INTRODUCTION

This book is concerned with numerical integration in general p dimensional spaces. To understand why special methods are needed, let us consider for the moment trapezoidal or center-of-bin integration on the unit interval in $p = 1$ dimension. Since these are both second order methods, to achieve an accuracy of one part in 10^4 one needs a division of the unit interval into roughly 100 subdivisions, with an evaluation of the integrand function at each. This poses no problem for numerical evaluation, but suppose instead we wish to integrate a function over a 9 dimensional region, achieving a similar accuracy of one part in 10^4. One then needs 100 divisions per axis, and $(100)^9 = 10^{18}$ function evaluations, which is a daunting task even for the fastest current computers. So a brute force extension of the trapezoidal rule (or similar higher order methods, such as Simpson's rule) is not a viable approach when the dimension of the space p is more than around four.

In consequence, methods for high dimensional spaces have focused on adaptive algorithms, in which function evaluations are concentrated in regions where the integrand is large and rapidly varying. Both Monte Carlo and deterministic algorithms have been proposed and widely used. Typically, they start from a base region, and then subdivide or refine on one to three or four sides along which the integrand is most rapidly varying. The process is then iterated, leading to finer subdivisions and an improved estimate of the integrand. Most authors, however, have considered it to be computationally prohibitive to proceed at each step by dividing the base region into 2^p subregions, so that the maximal length of *each* side is reduced by a factor of 2 at each step. Such a subdivision would allow localization of isolated integrand peaks in p dimensions, giving a method with the potential of achieving high accuracy for integrations in high dimensional spaces, and high resolution in applications such as template-based pattern recognition.

The motivation for this book is the observation that computer speed has dramatically increased in recent years, while the cost of memory has simultaneously dramatically decreased; our current laptop speeds, and memories, are characterized by "giga" rather than the "mega" of two decades ago. So it is now timely to address the problem of formulating practical high dimension integration routines that proceed by 2^p subdivision. We will develop methods for adaptive integration over both general simplexes, and axis-parallel hypercubes. Our simplex method is based on combining Moore's (1992) algorithm for 2^p subdivision of a general simplex, with new formulas for parameterized higher order integration over a general simplex that we derive using the centroid approach of Good and Gaskins (1969, 1971), to give a a fully localizable adaptive integration procedure for general dimension $p \geq 1$. In addition to giving a hypercube method based on partition into simplexes, we also give a simpler, direct method for integration over hypercubes, constructed by analogy with our methods for simplexes. We focus specifically on a few special base region geometries: the standard simplex (relevant for calculating Feynman parameter integrals in physics), the Kuhn simplex, which can be used to tile the p dimensional side 1 hypercube by symmetrization of the integrand, and the half-side 1 hypercube, which for which we give direct algorithms which are simpler than the simplex-based algorithms. By changes of variable, any multiple integral with fixed limits of integration in each dimension can be converted to an integration over the side 1 or half-side 1 hypercube. In the following sections we develop the theory behind our methods, and then give a suite of Fortran programs, for both serial and MPI parallel computation, implementing them.

II. ONE DIMENSIONAL ADAPTIVE INTEGRATION

As a simple example, let us sketch how to write an adaptive integration program in one dimension for the integral

$$I = \int_0^1 dx f(x) \quad . \tag{1}$$

A first estimate can be obtained by using the trapezoidal rule

$$I_a \simeq 0.5[f(0) + f(1)] \quad , \tag{2}$$

and a second estimate obtained by using the center-of-bin rule

$$I_b \simeq f(0.5) \quad . \tag{3}$$

These are both first order accurate methods, but since they are applied to the entire interval $(0, 1)$ there will be a significant error, unless $f(x)$ happens to be a linear function over the interval. If we want an evaluation of the integral with an estimated error ϵ, we test whether $|I_a - I_b| < \epsilon$. If this condition is satisfied, we output I_a and I_b as estimates of the integral. If the condition is not satisfied, we subdivide the interval $(0,1)$ into two half-sized intervals $(0,0.5)$ and $(0.5,1)$. In each subinterval we follow the same procedure. For a subinterval with upper limit x_U and lower limit x_L, and midpoint x_M, we now define

$$I_a \simeq 0.5[f(x_U) + f(x_L)] \quad , \tag{4}$$

and

$$I_b \simeq f(x_M) \quad . \tag{5}$$

For the two subintervals, we evaluate the trapezoidal and center-of-bin approximations to the integral, keeping $I_a(\text{subinterval})$ and $I_b(\text{subinterval})$ for the subinterval, multiplied by the subinterval width of $1/2$, as contributions to the answer if the "thinning" condition

$$|I_a(\text{subinterval}) - I_b(\text{subinterval})| < \epsilon \tag{6}$$

is met, and subdividing the interval by half again if this condition is not met. When, after a sequence of subdivisions, the condition is met for all subintervals, we have obtained good approximations to both a trapezoidal and center-of-bin evaluation of the integral,

$$\begin{aligned} I_a &\simeq \sum_{\text{subintervals}} L(\text{subinterval}) I_a(\text{subinterval}) \quad , \\ I_b &\simeq \sum_{\text{subintervals}} L(\text{subinterval}) I_b(\text{subinterval}) \quad . \end{aligned} \tag{7}$$

Here $L(\text{subinterval})$ is the subinterval length, and since the subintervals are a tiling of the interval $(0,1)$, we clearly have

$$\sum_{\text{subintervals}} L(\text{subinterval}) = 1 \quad . \tag{8}$$

From the difference of I_a and I_b we get an estimate of the error, given by

$$|\text{outdiff}| \equiv |I_a - I_b| \quad . \tag{9}$$

We can also compute the sum of the absolute values of the local subinterval errors,

$$\text{errsum} \equiv \sum_{\text{subintervals}} L(\text{subinterval}) |I_a(\text{subinterval}) - I_b(\text{subinterval})| \geq |\text{outdiff}| \quad . \tag{10}$$

When the condition $|I_a(\text{subinterval}) - I_b(\text{subinterval})| < \epsilon$ is met for all subintervals, then errsum reduces, using Eq. (8), to

$$\text{errsum} < \epsilon \quad , \tag{11}$$

and if all local subinterval errors have the same sign, then we have errsum = |outdiff|.

If the process of subdivision has to be stopped before the condition $|I_a(\text{subinterval}) - I_b(\text{subinterval})| < \epsilon$ is satisfied for all subintervals, with the remaining subregion contributions added to I_a and I_b before the program terminates, then errsum will typically be larger than ϵ. Such premature termination can happen for very irregular or singular functions, or if the parameter ϵ is made too small, or if one subdivides without imposing the thinning condition of Eq. (6). But for smooth functions $f(x)$ and thinning with attainable ϵ the subdivision process will terminate quite rapidly. The reason is that both the trapezoidal and center-of-bin methods are accurate to first order with a second order error, and so the difference $I_a(\text{subinterval}) - I_b(\text{subinterval})$ scales as $[L(\text{subinterval})]^2$ as the subinterval length $L(\text{subinterval})$ approaches zero.

The adaptive integration method just sketched is easily programmed, and works well. One does not have to keep track of the relative location of the various subintervals, only of their starting and ending x values. Thus, one maintains a list of active subintervals, stored in any convenient order; when a subinterval is divided the two resulting halves are added to the list of active subintervals, while if a subinterval obeys the thinning condition , its contributions to I_a, I_b, and errsum are added to an accumulation register, and the subinterval is removed from the list of active subintervals.

Even faster termination is obtained if Simpson's rule or an even higher order integration rule is used; see for example the Wikipedia article on the McKeeman (1962) adaptive Simpson rule, and references given there. The idea again is to compute two different evaluations of the integral over each subinterval, giving an error estimate that is used to determine whether to "harvest" the result at that level of subdivision, or to subdivide further. In generalizing to higher dimensional integrals, the same features persist: for each integration subregion, we evaluate a local thinning condition obtained from the difference of two alternative higher order integration rules. If the condition is obeyed, that subregion is "harvested" and deleted from the list of active subregions; if the condition is not obeyed, the subregion is further subdivided and the resulting smaller subregions are added to the active list.

III. GENERALIZING TO HIGHER DIMENSIONS: SIMPLEXES AND HYPERCUBES. REVIEW OF PRIOR WORK.

The first question to decide in generalizing to higher dimensions is the choice of base region geometry. There are two natural higher dimensional analogs of the one dimensional interval $(0, 1)$. The first is the side 1 hypercube $(0, 1) \otimes (0, 1) \otimes ... \otimes (0, 1)$, and the second is what we will term a

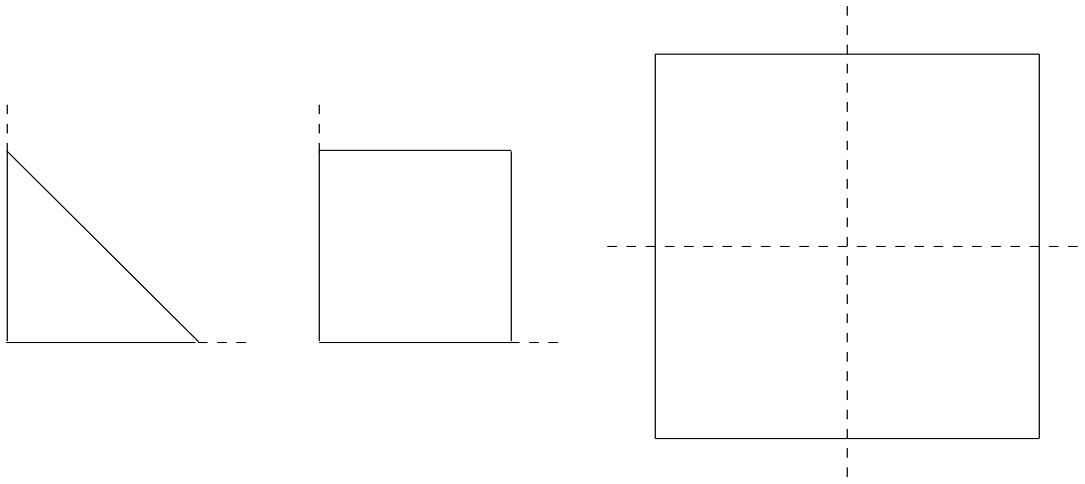

FIG. 1: From left to right, the unit standard simplex, the side 1 hypercube, and the half-side 1 hypercube, in 2 dimensions.

standard simplex with vertices $(0,0,...,0)$, $(0,1,0,0.....0)$, $(0,0,1,0,0....,0),....,(0,0,....,0,1)$. We will also make use of the half-side 1 hypercube, spanning $(-1,1)\otimes(-1,1)\otimes...\otimes(-1,1)$. These three basic regions are illustrated, in two dimensions, in Fig. 1.

Some simple geometric facts are important in setting a strategy. For a p dimensional simplex, the number of vertices is $p+1$ and the number of sides connecting vertices is $(p+1)p/2$, both of which have polynomial growth. Thus, the indexing problem of keeping track of vertices which define active regions is relatively simple. For a p dimensional hypercube, the number of m-dimensional hypercubes on the boundary (see, e.g., the Wikipedia article on hypercubes) is $2^{p-m}p\,!/(m\,!(p-m)\,!)$, and so the number of vertices ($m=0$) is 2^p, and the number of sides connecting vertices ($m=1$) is $p\,2^{p-1}$, both of which grow exponentially with p. Thus, if one labels hypercubes in terms of their vertices or sides, an exponentially growing index is required for large p. However, for the maximal boundary hypercube, with $m=p-1$, the number given by the above formula is just $2p$, which again has polynomial, in fact linear, growth. (For example, a square has $2\times2=4$ lines as sides, and a cube has $2\times3=6$ squares as faces.) So our direct method for hypercubes will use geometric features of the maximal boundary hypercubes for indexing, subdivision, and integration, closely following the methods that we develop for simplexes.

Before getting into further details, let us first give a very brief survey of adaptive methods for higher dimensional integration that are currently in the literature. A method that is widely used by physicists to evaluate Feynman parameter integrals is the VEGAS program of Lepage (1978), which uses a hypercube as the base geometry. This is a Monte Carlo method, in which random samplings of the integration volume are done with a separable probability density that is a product of one dimensional densities along each axis. This probability density is then iterated to give a more detailed sampling along axes on which the projection of the integrand is rapidly varying. A deterministic method of Genz and Cools (2003) is based on simplexes as the base regions. The algorithm picks the subregion with the largest estimated error, and subdivides it into up to four equal volume subregions by cutting edges along which the integrand is most rapidly varying. This, and related adaptive algorithms, are discussed in the survey of CUBPACK by Cools and Haegemans (2003). The CUBA set of algorithms described by Hahn (2005) includes both Monte Carlo methods and deterministic methods; the former include refinements of VEGAS and the latter proceed by bisection of the subregion with largest error. A survey of many types of high dimensional integration algorithms, including adaptive algorithms, is contained in the HIntLib Manual of Schürer (2008).

Most of the algorithms just described do not proceed directly to a 2^p subdivision of the base region (although the possibility of 2^p subdivision is sketched in “Algorithm 2” of Cools and Haegemans (2003)). An algorithm in the literature which makes use of a 2^p subdivision was given by Kahaner and Wells (1979). Unlike the algorithms which we develop below, which work directly from the vertex coordinates of a general simplex, the Kahaner Wells algorithm uses changes of variable for both simplex subdivision and integration. It also rank orders the errors for each subregion (as do most of the algorithms described in the preceding paragraph), and at each stage subdivides the subregion with the largest contribution to the total error. While this global method may result in efficiencies in reducing the number of subdivisions needed, it makes parallelization of the algorithm more complicated, since the computations for the different subregions are not independent of one another. Also, when many subregions have errors of similar size, which is often the case, the computational effort involved in rank ordering the errors may not be justified. In the algorithms developed below, as in the one-dimensional example given in Sec. 1, we use a local thinning condition for the subregions, making it easy to turn serial versions of the algorithm into parallel ones. We note, however, that the subdivision and integration methods that we use could also be incorporated into global adaptive algorithms.

IV. SIMPLEX PROPERTIES AND APPLICATIONS

Any set of $p + 1$ points in p dimensional space defines a p-simplex, and we will be concerned with integrations over the interior region defined this way. Thus, in 1 dimension, 2 points define a

1-simplex that is the line segment joining them, in 2 dimensions, 3 points define a 2-simplex that is a triangle, in three dimensions, 4 points define a 3-simplex that is a tetrahedron, and so forth. We will refer to the $p+1$ points, that each define a p-vector, as the vertices of the simplex, and our strategy will be to express all operations, both for the subdivision of simplexes and for calculating approximations to integrals over simplexes, directly in terms of these vertices. Our convention, both here in the text and in the programs, is that the $p+1$ vertices of a simplex are enumerated from 0 to p, and the p vector components of each vertex are enumerated from 1 to p. Both will be denoted by subscripts; it should be clear from context and from the range of the index whether an index is the label of a simplex vertex, as in $x_0, ..., x_p$, or the component index of a general point x, as in $x_1, ..., x_p$. In this notation, the ith component of the jth simplex vertex is denoted by a double subscript x_{ji}.

A. Simplex properties

A simplex forms a convex set. This means that for any integer $n \geq 1$ and any set of points $x_1, ..., x_n$ lying within (or on the boundary) of a simplex, and any set of non-negative numbers $\alpha_1, ..., \alpha_n$ which sum to unity,

$$\begin{aligned} \alpha_j \geq 0, \quad j = 1, ..., n \quad , \\ \sum_{j=1}^{n} \alpha_j = 1 \quad , \end{aligned} \tag{12}$$

the point

$$x = \sum_{j=1}^{n} \alpha_j x_j \tag{13}$$

also lies within (or on the boundary) of the simplex (see, e.g., Osborne (2001)).

In constructing integration rules for simplexes, we will be particularly interested in linear combinations of the form of Eq. (13) in which the points $x_1, ..., x_n$ are vertices of the simplex. For such sums, one can state a rule which determines precisely where the point x lies with respect to the boundaries of the simplex. Let $x_0, x_1, ..., x_p$ be the vertices of a simplex, and let x_c denote the centroid of the simplex,

$$x_c = \frac{1}{p+1} \sum_{j=0}^{p} x_j \quad . \tag{14}$$

Let us denote by $\tilde{x}_j$ the coordinates of the vertices with respect to the centroid as origin,

$$\tilde{x}_j = x_j - x_c \quad , \tag{15}$$

which obey the constraint following from Eq. (14),

$$\sum_{j=0}^{p} \tilde{x}_j = 0 \quad . \tag{16}$$

Correspondingly, let x denote a general point, and let $\tilde{x} = x - x_c$ denote the general point referred to the centroid as origin. Since we are assuming that the simplex is non-degenerate, the vectors $\tilde{x}_j$ span a linearly independent basis for the p-dimensional space, and so we can always expand $\tilde{x}$ as a linear combination of the $\tilde{x}_j$,

$$\tilde{x} = \sum_{j=0}^{p} \alpha_j \tilde{x}_j \quad . \tag{17}$$

This expansion is not unique, since by Eq. (16) we can add a constant a to all of the coefficients α_j, without changing the sum in Eq. (17). In particular, we can use this freedom to put the expansion of Eq. (17) in a standard form, which we will assume henceforth, in which the sum of the coefficients α_j is unity,

$$\sum_{j=0}^{p} \alpha_j = 1 \quad . \tag{18}$$

For coefficients (called barycentric coordinates) obeying this unit sum condition, we can use Eqs. (14) and (15) to also write

$$x = \sum_{j=0}^{p} \alpha_j x_j \quad . \tag{19}$$

In terms of the expansion of Eqs. (17) through (19) we can now state a rule $\big($see Pontryagin (1952) and the Wikipedia article on barycentric coordinates$\big)$ for determining where the point x lies with respect to the simplex: (1) If all of the α_j are strictly positive, the point lies inside the boundaries of the simplex; (2) If a coefficient α_j is zero, the point lies on the boundary plane opposite to the vertex x_j, and if several of the α_j vanish, the point lies on the intersection of the corresponding boundary planes; (3) If any coefficient α_j is negative, the point lies outside the simplex.

To derive this rule, we observe that a point x lies within the simplex only if it lies on the same side of each boundary plane of the simplex as the simplex vertex opposite that boundary. Let us focus on one particular vertex of the simplex, which we label x_p, so that the other p vertices are

$x_0, ..., x_{p-1}$. These p vertices span an affine hyperplane, which divides the p-dimensional space into two disjoint parts, and constitutes the simplex boundary hyperplane opposite the simplex vertex x_p. A general parameterization of this hyperplane takes the form

$$x = x_0 + \sum_{j=1}^{p-1} \beta_j (x_j - x_0) \quad , \tag{20}$$

that is, we take x_0 as a fiducial point on the hyperplane and add arbitrary multiples of a complete basis of vectors $x_j - x_0$ in the hyperplane. Rewriting Eq. (20) as

$$x = \sum_{j=0}^{p-1} \gamma_j x_j \quad , \tag{21}$$

with $\gamma_0 = 1 - \sum_{j=1}^{p-1} \beta_j$ and $\gamma_j = \beta_j, \quad j \geq 1$, we see that the p coefficients γ_j obey the condition

$$\sum_{j=0}^{p-1} \gamma_j = 1 \quad . \tag{22}$$

By virtue of this condition, we can also write the hyperplane parameterization of Eq. (21) in terms of coordinates with origin at the simplex centroid,

$$\tilde{x} = \sum_{j=0}^{p-1} \gamma_j \tilde{x}_j \quad . \tag{23}$$

We now wish to determine whether the general point $\tilde{x}$ lies on the same side of this hyperplane as the vertex $\tilde{x}_p$, or lies on the hyperplane, or lies on the opposite side from $\tilde{x}_p$, by using the expansion of Eqs. (17) and (18), which we rewrite in the form

$$\begin{aligned} \tilde{x} = & \sum_{j=0}^{p-1} \alpha_j \tilde{x}_j + \frac{\alpha_p}{p} \sum_{j=0}^{p-1} \tilde{x}_j \\ & - \frac{\alpha_p}{p} \sum_{j=0}^{p-1} \tilde{x}_j + \alpha_p \tilde{x}_p \quad . \end{aligned} \tag{24}$$

The first line on the right hand side of Eq. (24) has the form of the hyperplane parameterization of Eq. (23), since by construction the coefficients add up to unity, and so this part of the right hand side is a point on the boundary hyperplane opposite the vertex $\tilde{x}_p$. The second line on the right hand side of Eq. (24) can be rewritten, by using Eq. (16), as

$$\alpha_p \frac{p+1}{p} \tilde{x}_p \quad . \tag{25}$$

To appreciate the significance of this, we note that the centroid of the p points defining the boundary hyperplane is

$$\tilde{x}_{h;\,c} = \frac{1}{p}\sum_{j=0}^{p-1}\tilde{x}_j = -\frac{1}{p}\tilde{x}_p \quad , \tag{26}$$

where we have again used Eq. (16). Therefore the vector from the centroid of the points defining the hyperplane to the vertex $\tilde{x}_p$ is

$$\tilde{x}_p - \tilde{x}_{h;\,c} = \frac{p+1}{p}\tilde{x}_p \quad . \tag{27}$$

So Eq. (25) tells us that the point $\tilde{x}$ is displaced from the hyperplane by a vector parallel to that of Eq. (27), with its length rescaled by the factor α_p. Therefore, if $\alpha_p > 0$, the point $\tilde{x}$ lies on the same side of the boundary hyperplane as the opposite vertex $\tilde{x}_p$. If $\alpha_p = 0$, the point $\tilde{x}$ lies on the boundary hyperplane, and if $\alpha_p < 0$, the point $\tilde{x}$ lies on the opposite side of the boundary hyperplane from the vertex $\tilde{x}_p$. Applying this argument to all $p+1$ vertices in turn gives the rules stated above.

In constructing integration rules for simplexes, we will use the following elementary corollary of the result that we have just derived. Consider the sum

$$\tilde{X} = \sum_{i=1}^{N}\lambda_i\tilde{x}_i \quad , \tag{28}$$

with the coefficients λ_i obeying

$$\begin{aligned} &\lambda_i > 0, \quad i = 1, ..., N \\ &\sum_{i=1}^{N}\lambda_i < 1 \quad , \end{aligned} \tag{29}$$

with the points $\tilde{x}_i$ any vertices of a simplex. Some vertices may be omitted, and some used more than once, in the sum of Eq. (28). Then the point $\tilde{X}$ lies inside the simplex. To see this, we note that by adding a positive multiple of zero in the form of Eq. (16), the sum of Eq. (28) can be reduced to the form of Eqs. (17) and (18), with all expansion coefficients α_j strictly positive. By the rule stated above, this implies that the point $\tilde{X}$ lies within the simplex.

B. Simplex applications

Our programs for p-dimensional integration make special use of two kinds of simplexes, the unit "standard simplex" introduced above, and the unit Kuhn simplex. In this subsection, we discuss important applications of these two special types of simplexes.

To recapitulate, the unit standard simplex has vertices given by

$$\begin{aligned} x_0 =&(0,0,0,...,0) \quad , \\ x_1 =&(1,0,0,...,0) \quad , \\ x_2 =&(0,1,0,...,0) \quad , \\ x_3 =&(0,0,1,0,...,0) \quad , \\ &........... \\ x_{p-1} =&(0,0,0,...,0,1,0) \quad , \\ x_p =&(0,0,0,...,0,0,1) \quad . \end{aligned} \tag{30}$$

It is bounded by axis-parallel hyperplanes $x_j = 0, \quad j = 1,...,p$ and the diagonal hyperplane $1 = x_1 + x_2 + ... + x_p$. Thus, the integral of a function $f(x_1,...,x_p)$ over the standard simplex can be written as a multiple integral in the form

$$\begin{aligned} \int_{\text{standard simplex}} f(x_1,...,x_p)dx_1...dx_p = & \int_0^1 dx_1 \int_0^{1-x_1} dx_2 \int_0^{1-x_1-x_2} dx_3.... \\ & \times \int_0^{1-x_1-x_2-...-x_{p-2}} dx_{p-1} \int_0^{1-x_1-x_2-...-x_{p-1}} dx_p f(x_1,...,x_p) \quad . \end{aligned} \tag{31}$$

An important physics application of this formula is the Feynman-Schwinger formula for combining perturbation theory denominators,

$$\frac{1}{D_0 D_1 ... D_p} = p\,! \int_{\text{standard simplex}} \frac{1}{[(1-x_1-x_2-...-x_p)D_0 + x_1 D_1 + ... + x_p D_p]^{p+1}} \quad , \tag{32}$$

which can be proved inductively as follows. For $p = 1$, the Feynman-Schwinger formula reads

$$\frac{1}{D_0 D_1} = \int_0^1 dx_1 \frac{1}{[(1-x_1)D_0 + x_1 D_1]^2} \quad , \tag{33}$$

which is easily verified by carrying out the integral. Assume now that this formula holds for dimension p. For $p+1$, the formula asserts that

$$\begin{aligned} \frac{1}{D_0 D_1 ... D_p D_{p+1}} =&(p+1)\,! \int_0^1 dx_1 .... \int_0^{1-x_1-x_2-...-x_{p-1}} dx_p \int_0^{1-x_1-x_2-...-x_{p-1}-x_p} dx_{p+1} \\ & \times \frac{1}{[(1-x_1-x_2-...-x_p-x_{p+1})D_0 + x_1 D_1 + ... + x_p D_p + x_{p+1} D_{p+1}]^{p+2}} \quad . \end{aligned} \tag{34}$$

Carrying out the integral over x_{p+1}, we get

$$\frac{1}{D_0 D_1 ... D_p D_{p+1}} = p\,! \int_0^1 dx_1 .... \int_0^{1-x_1-x_2-...-x_{p-1}} dx_p \frac{1}{D_{p+1}-D_0}$$
$$\times \left[\frac{1}{[(1-\sum_{i=1}^p x_i)D_0 + x_1 D_1 + ... + x_p D_p]^{p+1}} - \frac{1}{[(1-\sum_{i=1}^p x_i)D_{p+1} + x_1 D_1 + ... + x_p D_p]^{p+1}} \right] \quad . \tag{35}$$

But applying the induction hypothesis for p dimensions, the right hand side of this equation reduces to

$$\frac{1}{D_{p+1}-D_0} \left[\frac{1}{D_0 D_1 .... D_p} - \frac{1}{D_{p+1} D_1 .... D_p} \right] = \frac{1}{D_0 D_1 ... D_p D_{p+1}} \quad , \tag{36}$$

which is the result to be proved. In the literature, numerical evaluation of Eq. (32) is usually accomplished by first making changes of variable to convert the simplex integral to an integral over a hypercube, and then using a hypercube-based program such as VEGAS. Using the methods developed below for direct evaluation of integrals over a standard simplex in arbitrary dimensions, the formula of Eq. (32) can also be integrated numerically in its original simplex form.

We next turn to the unit Kuhn (1960) simplex, which has the vertices given by

$$\begin{aligned} x_0 =& (0,0,0,...,0) \quad , \\ x_1 =& (1,0,0,...,0) \quad , \\ x_2 =& (1,1,0,...,0) \quad , \\ x_3 =& (1,1,1,0,...,0) \quad , \\ &............ \\ x_{p-1} =& (1,1,1,...,1,1,0) \quad , \\ x_p =& (1,1,1,...,1,1,1) \quad , \end{aligned} \tag{37}$$

and which defines a simplex in which $1 \geq x_1 \geq x_2 \geq x_3 .... \geq x_{p-1} \geq x_p$. The unit Kuhn simplex in two dimensions is illustrated in Fig. 2.

The integral of a function $f(x_1,...,x_p)$ over a unit Kuhn simplex can be written as a multiple integral in the form

$$\int_{\text{unit Kuhn simplex}} f(x_1,...,x_p) dx_1 ... dx_p = \int_0^1 dx_1 \int_0^{x_1} dx_2 \int_0^{x_2} dx_3 ....$$
$$\times \int_0^{x_{p-2}} dx_{p-1} \int_0^{x_{p-1}} dx_p f(x_1,...,x_p) \quad . \tag{38}$$

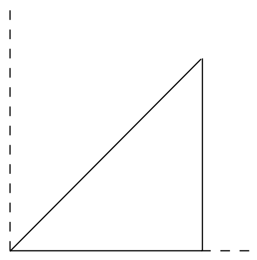

FIG. 2: The unit Kuhn simplex in 2 dimensions.

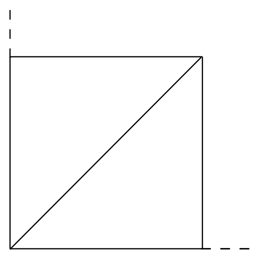

FIG. 3: Kuhn simplex tiling of a unit hypercube in 2 dimensions.

Consider now the integral of the function $f(x_1, ..., x_p)$ over the unit hypercube,

$$\int_0^1 dx_1 \int_0^1 dx_2 .... \int_0^1 dx_{p-1} \int_0^1 dx_p f(x_1, ..., x_p) \quad . \tag{39}$$

This hypercube can be partitioned into $p\,!$ regions, each congruent to the unit Kuhn simplex, by the requirement that in the region corresponding to the permutation P of the coordinate labels $1, ..., p$, the coordinates are ordered according to $x_{P(1)} \geq x_{P(2)} \geq x_{P(3)} .... \geq x_{P(p-1)} \geq x_{P(p)}$. This partitioning or tiling is illustrated for a square in Fig. 3, and for a cube in three dimensions in Fig. 1 of Plaza (2007).

Hence the integral of f over the unit hypercube is equal to the integral of the *symmetrized* function computed from f, integrated over the unit Kuhn simplex,

$$\begin{aligned} &\int_0^1 dx_1 \int_0^1 dx_2 .... \int_0^1 dx_{p-1} \int_0^1 dx_p f(x_1, ..., x_p) \\ &\qquad = \int_{\text{unit Kuhn simplex}} \sum_{p\,!\ \text{permutations}\ P} f(x_{P(1)}, ..., x_{P(p)}) dx_1 ... dx_p \quad . \end{aligned} \tag{40}$$

We will use this equivalence to construct adaptive programs for integration over a unit hypercube, based on first reducing it, by symmetrization, to an integral over a unit Kuhn simplex, and then adaptively subdividing the Kuhn simplex to reduce the integration error as needed.

V. SIMPLEX SUBDIVISION AND PROPERTIES

A. Simplex subdivision algorithms

Two very simple algorithms for subdividing simplexes have been given in the computer graphics literature by Moore (1992). Let us denote the vertices of the starting simplex by $x_0, ..., x_p$, each of which is a p-vector, and from these let us form the p-vectors $V(k_1, k_2)$ defined by

$$V(k_1, k_2) = \frac{1}{2}(x_{k_1} + x_{k_2}) \ , \quad k_1 \, , k_2 = 0, ..., p. \tag{41}$$

Thus, $V(0,0) = x_0$, $V(0,1) = (1/2)(x_0 + x_1)$ and so forth, so that the vectors $V(k_1, k_2)$ consist of the original simplex vertices, together with the midpoints of the original simplex edges. Let $k = 0, ..., 2^p - 1$ be an index which labels the 2^p subsimplexes into which the original simplex is divided. Moore then gives two algorithms, which he terms *recursive subdivision* and *symmetric subdivision*, for determining the vertices to be assigned to the subsimplex labelled with k. Both make use of the binary representation of k, and of a function determined by this representation, the bitcount function $b(k)$, which is the number of 1 bits appearing in the binary representation of k.

The recursive subdivision algorithm proceeds as follows. As the 0 vertex of the subsimplex labelled by k, take the vector $V(b(k), b(k))$, that is, $k_1 = k_2 = b(k)$. To get the other vertices, scan along the binary representation of k from right (the units digit) to left. For each 0 encountered, add 1 to k_2, and for each 1 encountered, subtract 1 from k_1. The sequence of vectors $V(k_1, k_2)$ obtained this way gives the desired $p + 1$ vertices of the kth subsimplex.

The symmetric subdivision algorithm proceeds as follows. As the 0 vertex of the subsimplex labelled by k, take the vector $V(0, b(k))$, that is, $k_1 = 0$, $k_2 = b(k)$. To get the other vertices, scan along the binary representation of k from right (the units digit) to left. For each 0 encountered, add 1 to k_2, and for each 1 encountered, add 1 to k_1. The sequence of vectors $V(k_1, k_2)$ obtained this way gives the desired $p + 1$ vertices of the kth subsimplex.

The application of these algorithms in the $p = 2$ case is illustrated in Tables I and II and Figs. 4–7, and in the $p = 3$ case is illustrated in Tables III and IV, where the notations $V^{(j)}(k_1, k_2)$ and $x^{(j)}$ both refer to the jth vertex, $j = 0, ..., p$, of the subdivided simplex labelled by the k in each row. After reviewing these tables, it should be easy to follow the Fortran program for the algorithms given later on. The standard Fortran library does not include a bitcount function, but it does include a function $IBITS(I, POS, LEN)$, which gives the value of the substring of bits of length LEN, starting at position POS, of the argument I. Thus, $IBITS(k, j, 1)$ gives the binary

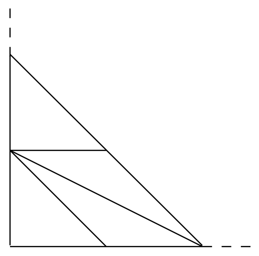

FIG. 4: Recursive subdivision of a standard simplex.

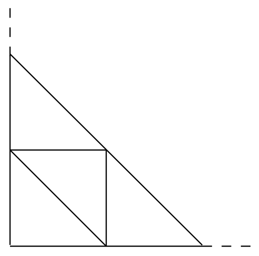

FIG. 5: Symmetric subdivision of a standard simplex.

digit (0 or 1) at position j in the binary representation of k, which is all the information needed for the algorithm.

B. Subdivision properties

This subdivision algorithm has a number of properties that will be useful in applying it to p dimensional integration.

1. As noted by Moore, the subdivided simplexes all have equal volume, equal to the initial simplex volume divided by 2^p. This follows from the fact that the general formula for the volume of a simplex with vertices $x_0, x_1, ..., x_p$ is

$$V = \frac{1}{p!} |\det(x_1 - x_0, x_2 - x_0, ..., x_p - x_0)| \quad . \tag{42}$$

   Applying this to the vertices for the subdivided simplexes in Tables I-IV verifies this statement for $p = 2, 3$, while a proof in the general case is given in Edelsbrunner and Grayson (2000).

2. Again as noted by Moore, both the recursive and symmetric algorithms subdivide Kuhn simplexes into Kuhn simplexes, which however do not all have the same orientation, as illustrated in Fig. 6 and Fig. 7. This follows from the fact that Kuhn simplexes are a tiling

TABLE I: Recursive subdivision of a triangle ($p = 2$)

k	$b(k)$	$V^{(0)}(k_1, k_2)$	$V^{(1)}(k_1, k_2)$	$V^{(2)}(k_1, k_2)$	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$
0=00	0	(0,0)	(0,1)	(0,2)	x_0	$\frac{1}{2}(x_0 + x_1)$	$\frac{1}{2}(x_0 + x_2)$
1=01	1	(1,1)	(0,1)	(0,2)	x_1	$\frac{1}{2}(x_0 + x_1)$	$\frac{1}{2}(x_0 + x_2)$
2=10	1	(1,1)	(1,2)	(0,2)	x_1	$\frac{1}{2}(x_1 + x_2)$	$\frac{1}{2}(x_0 + x_2)$
3=11	2	(2,2)	(1,2)	(0,2)	x_2	$\frac{1}{2}(x_1 + x_2)$	$\frac{1}{2}(x_0 + x_2)$

TABLE II: Symmetric subdivision of a triangle ($p = 2$)

k	$b(k)$	$V^{(0)}(k_1, k_2)$	$V^{(1)}(k_1, k_2)$	$V^{(2)}(k_1, k_2)$	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$
0=00	0	(0,0)	(0,1)	(0,2)	x_0	$\frac{1}{2}(x_0 + x_1)$	$\frac{1}{2}(x_0 + x_2)$
1=01	1	(0,1)	(1,1)	(1,2)	$\frac{1}{2}(x_0 + x_1)$	x_1	$\frac{1}{2}(x_1 + x_2)$
2=10	1	(0,1)	(0,2)	(1,2)	$\frac{1}{2}(x_0 + x_1)$	$\frac{1}{2}(x_0 + x_2)$	$\frac{1}{2}(x_1 + x_2)$
3=11	2	(0,2)	(1,2)	(2,2)	$\frac{1}{2}(x_0 + x_2)$	$\frac{1}{2}(x_1 + x_2)$	x_2

of hypercubes, which are divided into hypercubes by axis parallel planes that intersect the midpoints of the hypercube edges. Adding additional diagonal slices intersecting the midpoints of the hypercube edges gives Kuhn tilings of both the original and the subdivided hypercubes. However, as also noted by Moore, when the algorithms are applied to general simplexes, the resultant subdivided simplexes can have different shapes, and are not isomorphic. For $p = 2$, Fig. 4 shows that recursive subdivision applied to the standard simplex leads to subsimplexes of different shapes, while Fig. 5 shows that symmetric subdivision applied to the standard simplex leads to subsimplexes that are all standard simplexes with dimension reduced by half. However, an examination of the vertices in Table IV shows that already at $p = 3$, symmetric subdivision of a standard simplex does not lead to subsimplexes that are all half size standard simplexes. For example, for $k = 2$ in Table IV, there are vertices $\frac{1}{2}(x_0 + x_2)$ and $\frac{1}{2}(x_1 + x_3)$, the edge joining which has length $\sqrt{3}/2$, whereas the maximum side length of a half size $p = 3$ standard simplex is $\sqrt{2}/2$.

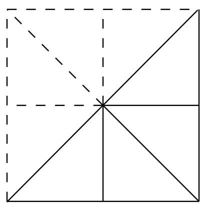

FIG. 6: Recursive subdivision of a Kuhn simplex.

TABLE III: Recursive subdivision of a tetrahedron ($p = 3$)

k	$b(k)$	$V^{(0)}(k_1,k_2)$	$V^{(1)}(k_1,k_2)$	$V^{(2)}(k_1,k_2)$	$V^{(3)}(k_1,k_2)$	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$
0=000	0	(0,0)	(0,1)	(0,2)	(0,3)	x_0	$\frac{1}{2}(x_0+x_1)$	$\frac{1}{2}(x_0+x_2)$	$\frac{1}{2}(x_0+x_3)$
1=001	1	(1,1)	(0,1)	(0,2)	(0,3)	x_1	$\frac{1}{2}(x_0+x_1)$	$\frac{1}{2}(x_0+x_2)$	$\frac{1}{2}(x_0+x_3)$
2=010	1	(1,1)	(1,2)	(0,2)	(0,3)	x_1	$\frac{1}{2}(x_1+x_2)$	$\frac{1}{2}(x_0+x_2)$	$\frac{1}{2}(x_0+x_3)$
3=011	2	(2,2)	(1,2)	(0,2)	(0,3)	x_2	$\frac{1}{2}(x_1+x_2)$	$\frac{1}{2}(x_0+x_2)$	$\frac{1}{2}(x_0+x_3)$
4=100	1	(1,1)	(1,2)	(1,3)	(0,3)	x_1	$\frac{1}{2}(x_1+x_2)$	$\frac{1}{2}(x_1+x_3)$	$\frac{1}{2}(x_0+x_3)$
5=101	2	(2,2)	(1,2)	(1,3)	(0,3)	x_2	$\frac{1}{2}(x_1+x_2)$	$\frac{1}{2}(x_1+x_3)$	$\frac{1}{2}(x_0+x_3)$
6=110	2	(2,2)	(2,3)	(1,3)	(0,3)	x_2	$\frac{1}{2}(x_2+x_3)$	$\frac{1}{2}(x_1+x_3)$	$\frac{1}{2}(x_0+x_3)$
7=111	3	(3,3)	(2,3)	(1,3)	(0,3)	x_3	$\frac{1}{2}(x_2+x_3)$	$\frac{1}{2}(x_1+x_3)$	$\frac{1}{2}(x_0+x_3)$

TABLE IV: Symmetric subdivision of a tetrahedron ($p = 3$)

k	$b(k)$	$V^{(0)}(k_1,k_2)$	$V^{(1)}(k_1,k_2)$	$V^{(2)}(k_1,k_2)$	$V^{(3)}(k_1,k_2)$	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$
0=000	0	(0,0)	(0,1)	(0,2)	(0,3)	x_0	$\frac{1}{2}(x_0+x_1)$	$\frac{1}{2}(x_0+x_2)$	$\frac{1}{2}(x_0+x_3)$
1=001	1	(0,1)	(1,1)	(1,2)	(1,3)	$\frac{1}{2}(x_0+x_1)$	x_1	$\frac{1}{2}(x_1+x_2)$	$\frac{1}{2}(x_1+x_3)$
2=010	1	(0,1)	(0,2)	(1,2)	(1,3)	$\frac{1}{2}(x_0+x_1)$	$\frac{1}{2}(x_0+x_2)$	$\frac{1}{2}(x_1+x_2)$	$\frac{1}{2}(x_1+x_3)$
3=011	2	(0,2)	(1,2)	(2,2)	(2,3)	$\frac{1}{2}(x_0+x_2)$	$\frac{1}{2}(x_1+x_2)$	x_2	$\frac{1}{2}(x_2+x_3)$
4=100	1	(0,1)	(0,2)	(0,3)	(1,3)	$\frac{1}{2}(x_0+x_1)$	$\frac{1}{2}(x_0+x_2)$	$\frac{1}{2}(x_0+x_3)$	$\frac{1}{2}(x_1+x_3)$
5=101	2	(0,2)	(1,2)	(1,3)	(2,3)	$\frac{1}{2}(x_0+x_2)$	$\frac{1}{2}(x_1+x_2)$	$\frac{1}{2}(x_1+x_3)$	$\frac{1}{2}(x_2+x_3)$
6=110	2	(0,2)	(0,3)	(1,3)	(2,3)	$\frac{1}{2}(x_0+x_2)$	$\frac{1}{2}(x_0+x_3)$	$\frac{1}{2}(x_1+x_3)$	$\frac{1}{2}(x_2+x_3)$
7=111	3	(0,3)	(1,3)	(2,3)	(3,3)	$\frac{1}{2}(x_0+x_3)$	$\frac{1}{2}(x_1+x_3)$	$\frac{1}{2}(x_2+x_3)$	x_3

3. An important question is whether the maximum side length of the subdivided simplexes decreases at each stage of subdivision. For Kuhn simplexes, the answer is immediate, since subdivision results in Kuhn simplexes of half the dimension. Since the longest side of a unit Kuhn simplex in dimension p has length $\sqrt{p}$, after ℓ subdivisions the maximum side length

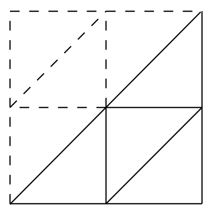

FIG. 7: Symmetric subdivision of a Kuhn simplex.

will be

$$L_{\max}^{\rm Kuhn} = \sqrt{p}/2^{\ell} \quad , \tag{43}$$

irrespective of whether recursive or symmetric subdivision is used. For standard simplexes, we can get an upper bound on the maximum side length by noting that a unit standard simplex on axes $y_1, ..., y_p$ is obtained from a unit Kuhn simplex on axes $x_1, ..., x_p$ by the linear transformation $y_p = x_p$, $y_{p-1} = x_{p-1} - x_p$, $y_{p-2} = x_{p-2} - x_{p-1}, ..., y_2 = x_2 - x_3$, $y_1 = x_1 - x_2$, since this maps the components of the Kuhn simplex vertices given in Eq. (37) to the corresponding components of the standard simplex vertices given in Eq. (30). By linearity, this relation also holds between vertices of corresponding subdivided simplexes obtained from the initial unit standard and Kuhn simplexes by applying the same midpoint subdivision method (either symmetric or recursive) successively to each. Consequently, the length $L^{\rm standard}$ of an edge with components $E^{\rm S}_{1,...,p}$ of a subdivided standard simplex can be expressed in terms of the components $E^{\rm K}_{1,...,p}$ of the corresponding edge of the related Kuhn simplex by

$$\begin{aligned} L^{\rm standard} \equiv& [\sum_{j=1}^{p} (E_j^S)^2] = [\sum_{j=1}^{p-1} (E_j^K - E_{j+1}^K)^2 + (E_p^K)^2]^{\frac{1}{2}} \\ \leq& 2[\sum_{j=1}^{p} (E_j^K)^2]^{\frac{1}{2}} = 2L^{\rm Kuhn} \quad . \end{aligned} \tag{44}$$

Thus the length $L^{\rm standard}$ is bounded from above by twice the maximum length corresponding to a subdivided Kuhn simplex, and so

$$L_{\max}^{\rm standard} \leq \sqrt{p}/2^{\ell-1} \quad . \tag{45}$$

We have verified this inequality numerically for both the recursive and symmetric subdivision algorithms. The numerical results suggest that the symmetric subdivision algorithm is in fact a factor of 2 better than the bound of Eq. (45), so that

$$L_{\max}^{\rm standard;\ symmetric} \leq \sqrt{p}/2^{\ell} \quad , \tag{46}$$

but we do not have a proof of this. We already see evidence of this difference between the symmetric and recursive algorithms in Tables III and IV. As noted above, from Table IV we saw that symmetric subdivision of a $p = 3$ standard simplex gives an edge of length $\sqrt{3}/2$, and it is easy to see that this is the longest edge. However, from Table III we see that for $k = 5$ there are vertices x_2 and $\frac{1}{2}(x_1 + x_3)$, the edge joining which, for an initial standard simplex, has length $\sqrt{6}/2$.

4. The result of Eqs. (45) and(46) suggests the stronger conjecture, that after any number ℓ of symmetric (recursive) subdivisions of a standard simplex, the resulting subsimplexes each fit within a hypercube of side $1/2^\ell$ $(1/2^{\ell-1})$. A simple argument shows this to be true for $\ell = 1$ in any dimension p. Although we do not have a proof for general ℓ, we will use this conjecture in certain of the algorithms constructed below. For Kuhn simplexes, an analogous statement with a hypercube of side $1/2^\ell$ is true for both symmetric and recursive subdivision, as noted above in the discussion preceding Eq. (43).

5. Finally, we note that although the symmetric algorithm gives the same simplex subdivision after permutation of the starting vertices in dimension $p = 2$, as can be verified from Table II, it is not permutation symmetric in dimension $p = 3$, as can be verified from Table IV. For example, interchanging the labels 0 and 1 in the $k = 2$ line of Table IV gives a set of vertices that is not in the table. This means that with symmetric (as well as recursive) subdivision in dimension $p \geq 3$, inequivalent simplex subdivisions can be generated by permuting the labels of the starting vertices. However, we have not incorporated this feature into our programs.

The properties just listed show that the symmetric and recursive subdivision algorithms are well suited to adaptive higher dimensional integration. They are easily computable in terms of the vertex coordinates for a general simplex, and give subsimplexes of equal volume, so that it is not necessary to calculate a determinant to obtain the volume. Additionally, the bound on the maximum side length decreases as a constant times $1/2^\ell$ with increasing order of subdivision ℓ, so that the application of high order integration formulas gives errors that decrease rapidly with ℓ.

VI. HYPERCUBE SUBDIVISION AND PROPERTIES

We have discussed simplexes first, because as noted in Sec. III, our direct approach to hypercube integration will be based on following as closely as possible the methods that we develop for simplex integration. In our direct hypercube programs (i.e., the ones not based on tiling a side 1 hypercube with Kuhn simplexes), we will start from a half-side 1 hypercube with base region

$$(-1, 1) \otimes (-1, 1) \otimes ... \otimes (-1, 1) \quad . \tag{47}$$

This region has inversion symmetry around the origin, and consequently the only monomials that have non-vanishing integrals over this region are ones in which *each* coordinate appears with an even exponent, considerably simplifying the calculations needed to construct higher order integration rules.

Since we restrict ourselves to axis-parallel hypercubes, only $p + 1$ real numbers are needed to uniquely specify a hypercube: the p coordinates of the centroid x_c and the half-side length S. For example, for the region of Eq. (47), the centroid is $x_c = (0, 0, ..., 0)$ and the half-side is 1. Once we have adopted this labelling, we can give a very simple subdivision algorithm for hypercubes, constructed in direct analogy with Moore's simplex subdivision algorithms.

The hypercube subdivision algorithm proceeds as follows. Start from a hypercube with centroid x_c and half-side S, with sides parallel to the p unit axis vectors

$$\begin{aligned}
\hat{u}_1 =&(1, 0, 0, ..., 0)\\
\hat{u}_2 =&(0, 1, 0, ..., 0)\\
&............\\
\hat{u}_{p-1} =&(0, 0, ..., 1, 0)\\
\hat{u}_p =&(0, 0, ..., 0, 1) \quad .
\end{aligned} \tag{48}$$

To subdivide it into 2^p subhypercubes, take the new half-side as $S/2$. To get the new centroids $x_{c;k}$, labelled by $k = 0, ..., 2^p - 1$, scan along the binary representation of k from right (the units digit) to left. Denoting the p digits in this representation by $1 \leq j \leq p$, let us label the units digit as $j = 1$, the power of 2 digit as $j = 2$, the power of 4 digit as $j = 3$, and so forth. For all $1 \leq j \leq p$, if the jth digit is 0, add $\frac{1}{2}S\hat{u}_j$ to x_c, and if the jth digit is 1, add $-\frac{1}{2}S\hat{u}_j$ to x_c. For each given k, this gives the centroid of the kth subhypercube. This algorithm is illustrated for the case of a cube ($p = 3$) in Table V.

This algorithm is simpler than the ones for subdividing simplexes, since it only needs the Fortran IBITS function, but does not require subsequent computation of the bitcount function. It evidently has properties analogous to those of the simplex subdivision algorithms: each subhypercube has the same volume, equal to the original hypercube volume divided by 2^p, and every linear dimension of each subhypercube is a factor of 2 smaller than the corresponding linear dimension of the hypercube that preceded it in the subdivision chain. This latter implies that after ℓ subdivisions, the resulting subhypercubes all have dimension reduced by a factor $1/2^{\ell}$.

TABLE V: Subdivision of a cube of half-side S and centroid x_c ($p = 3$)

k	$x_{c;k} - x_c$
0=000	$(S/2, S/2, S/2)$
1=001	$(-S/2, S/2, S/2)$
2=010	$(S/2, -S/2, S/2)$
3=011	$(-S/2, -S/2, S/2)$
4=100	$(S/2, S/2, -S/2)$
5=101	$(-S/2, S/2, -S/2)$
6=110	$(S/2, -S/2, -S/2)$
7=111	$(-S/2, -S/2, -S/2)$

For a hypercube with centroid x_c and half-side S, and for a general point x, let us define the coordinate relative to the centroid as $\tilde{x} = x - x_c$, as we did in the simplex case in Eq. (15). Consider now the set of $2p$ points $\tilde{x}_j \ , \quad j = 1, ..., 2p$ defined by

$$\begin{aligned}
\tilde{x}_1 =& (S, 0, 0, ..., 0) \\
\tilde{x}_2 =& (0, S, 0, ..., 0) \\
& .... \\
\tilde{x}_p =& (0, 0, ..., S) \\
\tilde{x}_{p+1} =& (-S, 0, 0, ..., 0) \\
\tilde{x}_{p+2} =& (0, -S, 0, ..., 0) \\
& .... \\
\tilde{x}_{2p} =& (0, 0, ..., -S) \qquad .
\end{aligned} \tag{49}$$

These points are the centroids of the maximal boundary hypercubes, and will play a role in the direct hypercube algorithm analogous to that played by the simplex vertices in the simplex adaptive algorithm. For future use, we need the following result, analogous to that of Eqs. (28) and (29) for the simplex case. Consider the sum

$$\tilde{X} = \sum_{i=1}^{N} \lambda_i \tilde{x}_i \quad , \tag{50}$$

with the coefficients λ_i obeying

$$\lambda_i > 0, \quad i = 1, ..., N$$
$$\sum_{i=1}^{N} \lambda_i < 1 \quad , \tag{51}$$

with the points $\tilde{x}_i$ any of the hypercube boundary points of Eq. (49). Some of these points may be omitted (in which case the corresponding coefficient λ_i is 0), and some used more than once, in the sum of Eq. (50). Then the point $\tilde{X}$ lies inside the hypercube. To see this, we note that the projection of $\tilde{X}$ along any axis j is of the form $\tilde{X}_j = S(\mu_+ - \mu_-)$, with $\mu_\pm$ each a sum of some subset of the coefficients λ_i, and hence $0 \leq \mu_\pm < 1$. Therefore $-S < -S\mu_- \leq \tilde{X}_j \leq S\mu_+ < S$ for each axis component $\tilde{X}_j$, and thus X lies within the hypercube. This proof, again, is simpler than the corresponding result in the simplex case.

VII. PARAMETERIZED HIGHER ORDER INTEGRATION FORMULAS FOR A GENERAL SIMPLEX

We turn next to deriving higher order integration formulas for a general simplex, which are expressed directly in terms of the set of simplex vertices, and which involve parameters that can be changed to sample the function over the simplex in different ways. Two different choices of the parameters then give two different integration rules of the same order, which can be compared to give a local error estimate for use in adaptive integration.

Since we want to derive integration rules up to ninth order in accuracy, we start from an expansion of a general function $f(\tilde{x})$ up to ninth order, with $\tilde{x}$ as before the p dimensional coordinate referred to the simplex centroid as origin. The expansion reads,

$$\begin{aligned} f(\tilde{x}) =& A + B_{i_1}\tilde{x}_{i_1} + C_{i_1 i_2}\tilde{x}_{i_1}\tilde{x}_{i_2} + D_{i_1 i_2 i_3}\tilde{x}_{i_1}\tilde{x}_{i_2}\tilde{x}_{i_3} + E_{i_1 i_2 i_3 i_4}\tilde{x}_{i_1}\tilde{x}_{i_2}\tilde{x}_{i_3}\tilde{x}_{i_4} + F_{i_1...i_5}\tilde{x}_{i_1}...\tilde{x}_{i_5} \\ &+ G_{i_1...i_6}\tilde{x}_{i_1}...\tilde{x}_{i_6} + H_{i_1...i_7}\tilde{x}_{i_1}...\tilde{x}_{i_7} + I_{i_1...i_8}\tilde{x}_{i_1}...\tilde{x}_{i_8} + J_{i_1...i_9}\tilde{x}_{i_1}...\tilde{x}_{i_9} + ... \quad . \end{aligned} \tag{52}$$

We next need expressions for the integral of the monomials appearing in the expansion of Eq. (52) over a general simplex with vertices $x_0, ..., x_p$. A general formula for these integrals has been given by Good and Gaskins (1969, 1971). They define $m(\nu)$ as the generalized moment

$$m(\nu) = \int_{\rm simplex} dx_1...dx_p \tilde{x}_1^{\nu_1}...\tilde{x}_p^{\nu_p} \quad , \tag{53}$$

and show that $m(\nu)$ is equal to the coefficient of $t_1^{\nu_1}...t_p^{\nu_p}$ in the expansion of

$$\frac{V p\,!\nu_1\,!...\nu_p\,!}{(p+\nu_1+...+\nu_p)\,!}\exp[\sum_{s=2}^{\infty}\frac{1}{s}W_s] \quad . \tag{54}$$

Here W_s is a double sum over ith components of the $p+1$ simplex vertices labelled by $a=0,...,p$, given by

$$W_s=\sum_{a=0}^{p}[\sum_{i=1}^{p}\tilde{x}_{ai}t_i]^s \quad , \tag{55}$$

and V is the simplex volume. Good and Gaskins derive this formula by first transforming the original simplex to a standard simplex, followed by lengthy algebraic manipulations to express the resulting formula symmetrically in terms of standard simplex vertices. We give in Sec. VIII below a derivation that proceeds directly, and with manifest symmetry, from the vertices of the original simplex.

To proceed to 9th order we need an expansion of the exponential in Eq. (54) through 9th order. Since each W_s is of degree s in the coordinates, the terms in this expansion are as follows:

$$\begin{aligned}
\text{second order}:&\quad \frac{W_2}{2}\\
\text{third order}:&\quad \frac{W_3}{3}\\
\text{fourth order}:&\quad \frac{W_2^2}{8}+\frac{W_4}{4}\\
\text{fifth order}:&\quad \frac{W_2W_3}{6}+\frac{W_5}{5}\\
\text{sixth order}:&\quad \frac{W_2^3}{48}+\frac{W_3^2}{18}+\frac{W_2W_4}{8}+\frac{W_6}{6}\\
\text{seventh order}:&\quad \frac{W_2^2W_3}{24}+\frac{W_3W_4}{12}+\frac{W_2W_5}{10}+\frac{W_7}{7}\\
\text{eighth order}:&\quad \frac{W_2^4}{384}+\frac{W_2W_3^2}{36}+\frac{W_2^2W_4}{32}+\frac{W_4^2}{32}+\frac{W_3W_5}{15}+\frac{W_2W_6}{12}+\frac{W_8}{8}\\
\text{ninth order}:&\quad \frac{W_2^3W_3}{144}+\frac{W_3^3}{162}+\frac{W_2W_3W_4}{24}+\frac{W_2^2W_5}{40}+\frac{W_4W_5}{20}+\frac{W_3W_6}{18}+\frac{W_2W_7}{14}+\frac{W_9}{9} \quad .
\end{aligned} \tag{56}$$

We are interested in integrals of monomials of the form $\tilde{x}_{i_1}...\tilde{x}_{i_n}$, with n ranging from 1 to 9. Good and Gaskins note that it suffices to consider the case in which all the indices $i_1,...,i_n$ are distinct (which is always possible for $p \geq n$), since the combinatoric factors are such that this gives a result that also applies to the case when some of the component indices are equal, as must necessarily be the case when $p < n$. So we can take $\nu_i = 1\,,\ i=1,...,n$, and $\sum_i \nu_i = n$, with n the order of the monomial. We now can infer from Eq. (56) the moment integrals

$$\frac{1}{V}\int_{\text{simplex}} dx_1 ... dx_p \tilde{x}_{i_1} ... \tilde{x}_{i_n} = \frac{p\,!}{(p+n)\,!}\mathcal{S}_n \quad , \tag{57}$$

with the quantities $\mathcal{S}_n$ (with tensor indices suppressed) given in terms of tensors $S_{i_1 ... i_n}$ defined by sums over the vertices,

$$S_{i_1 ... i_n} = \sum_{j=0}^{p} \tilde{x}_{ji_1} ... \tilde{x}_{ji_n} \quad , \tag{58}$$

as follows:

$$\begin{aligned}
\mathcal{S}_2 =& S_{i_1 i_2} \\
\mathcal{S}_3 =& 2S_{i_1 i_2 i_3} \\
\mathcal{S}_4 =& S_{i_1 i_2} S_{i_3 i_4} + S_{i_1 i_3} S_{i_2 i_4} + S_{i_1 i_4} S_{i_2 i_3} + 6S_{i_1 i_2 i_3 i_4} = S_{i_1 i_2} S_{i_3 i_4} + 2\,\text{terms} + 6S_{i_1 i_2 i_3 i_4} \\
\mathcal{S}_5 =& 2(S_{i_1 i_2} S_{i_3 i_4 i_5} + 9\,\text{terms}) + 24 S_{i_1 i_2 i_3 i_4 i_5} \\
\mathcal{S}_6 =& S_{i_1 i_2} S_{i_3 i_4} S_{i_5 i_6} + 14\,\text{terms} + 4(S_{i_1 i_2 i_3} S_{i_4 i_5 i_6} + 9\,\text{terms}) + 6(S_{i_1 i_2} S_{i_3 i_4 i_5 i_6} + 14\,\text{terms}) \\
&+ 120 S_{i_1 i_2 i_3 i_4 i_5 i_6} \\
\mathcal{S}_7 =& 2(S_{i_1 i_2} S_{i_3 i_4} S_{i_5 i_6 i_7} + 104\,\text{terms}) + 12(S_{i_1 i_2 i_3} S_{i_4 i_5 i_6 i_7} + 34\,\text{terms}) \\
&+ 24(S_{i_1 i_2} S_{i_3 i_4 i_5 i_6 i_7} + 20\,\text{terms}) + 720 S_{i_1 i_2 i_3 i_4 i_5 i_6 i_7} \\
\mathcal{S}_8 =& S_{i_1 i_2} S_{i_3 i_4} S_{i_5 i_6} S_{i_7 i_8} + 104\,\text{terms} + 4(S_{i_1 i_2} S_{i_3 i_4 i_5} S_{i_6 i_7 i_8} + 279\,\text{terms}) \\
&+ 6(S_{i_1 i_2} S_{i_3 i_4} S_{i_5 i_6 i_7 i_8} + 209\,\text{terms}) \\
&+ 36(S_{i_1 i_2 i_3 i_4} S_{i_5 i_6 i_7 i_8} + 34\,\text{terms}) + 48(S_{i_1 i_2 i_3} S_{i_4 i_5 i_6 i_7 i_8} + 55\,\text{terms}) \\
&+ 120(S_{i_1 i_2} S_{i_3 i_4 i_5 i_6 i_7 i_8} + 27\,\text{terms}) + 5040 S_{i_1 i_2 i_3 i_4 i_5 i_6 i_7 i_8} \\
\mathcal{S}_9 =& 2(S_{i_1 i_2} S_{i_3 i_4} S_{i_5 i_6} S_{i_7 i_8 i_9} + 1259\,\text{terms}) + 8(S_{i_1 i_2 i_3} S_{i_4 i_5 i_6} S_{i_7 i_8 i_9} + 279\,\text{terms}) \\
&+ 12(S_{i_1 i_2} S_{i_3 i_4 i_5} S_{i_6 i_7 i_8 i_9} + 1259\,\text{terms}) + 24(S_{i_1 i_2} S_{i_3 i_4} S_{i_5 i_6 i_7 i_8 i_9} + 377\,\text{terms}) \\
&+ 144(S_{i_1 i_2 i_3 i_4} S_{i_5 i_6 i_7 i_8 i_9} + 125\,\text{terms}) + 240(S_{i_1 i_2 i_3} S_{i_4 i_5 i_6 i_7 i_8 i_9} + 83\,\text{terms}) \\
&+ 720(S_{i_1 i_2} S_{i_3 i_4 i_5 i_6 i_7 i_8 i_9} + 35\,\text{terms}) + 40320 S_{i_1 i_2 i_3 i_4 i_5 i_6 i_7 i_8 i_9} \quad .
\end{aligned} \tag{59}$$

The rule for forming terms in Eq. (59) from those in Eq. (56) is this: for each W_s in Eq. (56) there is a tensor factor S with s indices, and the product of such factors appears repeated in all nontrivial index permutations, giving the "terms" not shown explicitly in Eq. (59). The numerical coefficient is constructed from the denominator appearing in Eq. (56), multiplied by a numerator consisting of a factor $s\,!$ for each W_s, and for each W_s^m an additional factor $m\,!$ (that is, for W_s^m there is altogether a factor $(s\,!)^m m\,!$). For example, a W_2^3 in Eq. (56) gives rise to a numerator factor of $(2\,!)^3 3\,! = 48$ in Eq. (59), and a $W_2W_3W_4$ in Eq. (56) gives rise to a numerator factor of $2\,!3\,!4\,! = 288$ in Eq. (59). In each case, the product of this numerator factor, times the number of terms in the symmetrized expansion, is equal to $n\,!$. For example, $48 \times 15 = 720 = 6\,!$, and $288 \times 1260 = 362880 = 9!$.

Our next step is to combine Eqs. (52), (57), and (59) to get a formula for the integral of the function f over a general simplex, expressed in terms of its expansion coefficients. Since we will always be dealing with symmetrized tensors, it is useful at this point to condense the notation, by labelling the contractions of the expansion coefficients with the tensors $\mathcal{S}$ by the partition of n which appears. Thus, we will write

$$\begin{aligned}
&C_{i_1i_2}S_{i_1i_2} = C_2 \\
&F_{i_1i_2i_3i_4i_5}(S_{i_1i_2}S_{i_3i_4i_5} + 9 \text{ terms}) = F_{3+2} \\
&H_{i_1i_2i_3i_4i_5i_6i_7}(S_{i_1i_2}S_{i_3i_4}S_{i_5i_6i_7} + 104 \text{ terms}) = H_{3+2+2} \quad ,
\end{aligned} \tag{60}$$

and so forth. Since the partitions of n that are relevant only involve $n \geq 2$, a complete list of partitions that appear through ninth order is as follows:

$$\begin{array}{ll}
C & 2 \\
D & 3 \\
E & 4,\ \ 2+2 \\
F & 5,\ \ 3+2 \\
G & 6,\ \ 4+2,\ \ 2+2+2,\ \ 3+3 \\
H & 7,\ \ 5+2,\ \ 3+2+2,\ \ 4+3 \\
I & 8,\ \ 6+2,\ \ 4+2+2,\ \ 2+2+2+2,\ \ 5+3,\ \ 3+3+2,\ \ 4+4 \\
J & 9,\ \ 7+2,\ \ 5+2+2,\ \ 3+2+2+2,\ \ 4+3+2,\ \ 6+3,\ \ 5+4,\ \ 3+3+3 \quad .
\end{array} \tag{61}$$

Employing this condensed notation, we now get the following master formula for the integral of f

over a general simplex,

$$\begin{aligned}
\frac{1}{V}\int_{\text{simplex}} dx_1...dx_p f(\tilde{x}) = {} & A + \frac{p\,!}{(p+2)\,!}C_2 + \frac{p\,!}{(p+3)\,!}2D_3 \\
& + \frac{p\,!}{(p+4)\,!}(6E_4 + E_{2+2}) + \frac{p\,!}{(p+5)\,!}(24F_5 + 2F_{3+2}) \\
& + \frac{p\,!}{(p+6)\,!}(120G_6 + 6G_{4+2} + 4G_{3+3} + G_{2+2+2}) \\
& + \frac{p\,!}{(p+7)\,!}(720H_7 + 24H_{5+2} + 12H_{4+3} + 2H_{3+2+2}) \\
& + \frac{p\,!}{(p+8)\,!}(5040I_8 + 120I_{6+2} + 48I_{5+3} + 36I_{4+4} + 6I_{4+2+2} + 4I_{3+3+2} + I_{2+2+2+2}) \\
& + \frac{p\,!}{(p+9)\,!}(40320J_9 + 720J_{7+2} + 240J_{6+3} + 144J_{5+4} + 24J_{5+2+2} + 12J_{4+3+2} \\
& + 8J_{3+3+3} + 2J_{3+2+2+2}) \\
& + ... \quad .
\end{aligned} \tag{62}$$

Our procedure is now to match this expansion to discrete sums over the function f evaluated at points on the boundary or interior of the simplex. We will construct these sums using parameter multiples of the vertices of the simplex (in which the summation limits for a, b, c, d are 0 to p for simplexes, and will be 1 to $2p$ later on when we apply these formulas to hypercubes),

$$\begin{aligned}
\Sigma_1(\lambda) = & \sum_a f(\lambda\tilde{x}_a)\ , \quad 0 \le \lambda \le 1 \\
\Sigma_2(\lambda,\sigma) = & \sum_{a,b} f(\lambda\tilde{x}_a + \sigma\tilde{x}_b)\ , \quad 0 \le \lambda,\sigma\ , \quad \lambda+\sigma \le 1 \\
\Sigma_3(\lambda,\sigma,\mu) = & \sum_{a,b,c} f(\lambda\tilde{x}_a + \sigma\tilde{x}_b + \mu\tilde{x}_c)\ , \quad 0 \le \lambda,\sigma,\mu\ , \quad \lambda+\sigma+\mu \le 1 \\
\Sigma_4(\lambda,\sigma,\mu,\kappa) = & \sum_{a,b,c,d} f(\lambda\tilde{x}_a + \sigma\tilde{x}_b + \mu\tilde{x}_c + \kappa\tilde{x}_d)\ , \quad 0 \le \lambda,\sigma,\mu,\kappa\ , \quad \lambda+\sigma+\mu+\kappa \le 1 \quad ,
\end{aligned} \tag{63}$$

where the conditions on the parameters $\lambda,\sigma,\mu,\kappa$ guarantee, by our discussion of simplex properties, that the points summed over do not lie outside the simplex. Clearly, once we have a formula for Σ_4, we can get a formula for Σ_3 by setting $\kappa = 0$ and dividing by $p+1$ (which becomes $2p$ in the hypercube case); we can then get Σ_2 by further setting $\mu = 0$ and dividing out another factor of $p+1$, and so forth. Hence we only exhibit here the expansion of Σ_4 in terms of $f(\tilde{0}) = A$ and the contractions $C_2, ..., J_{3+2+2+2}$ appearing in Eq. (62). Abbreviating $\xi = p + 1$, we have

$$\begin{aligned}
&\Sigma_4 = \xi^4 A + \xi^3(\lambda^2+\sigma^2+\mu^2+\kappa^2)C_2 + \xi^3(\lambda^3+\sigma^3+\mu^3+\kappa^3)D_3 + \xi^3(\lambda^4+\sigma^4+\mu^4+\kappa^4)E_4 \\
&+2\xi^2(\lambda^2\sigma^2+\lambda^2\mu^2+\lambda^2\kappa^2+\sigma^2\mu^2+\sigma^2\kappa^2+\mu^2\kappa^2)E_{2+2} + \xi^3(\lambda^5+\sigma^5+\mu^5+\kappa^5)F_5 \\
&+\xi^2(\lambda^2\sigma^3+\sigma^2\lambda^3+\lambda^2\mu^3+\mu^2\lambda^3+\lambda^2\kappa^3+\kappa^2\lambda^3+\sigma^2\mu^3+\mu^2\sigma^3+\sigma^2\kappa^3+\kappa^2\sigma^3+\mu^2\kappa^3+\kappa^2\mu^3)F_{3+2} \\
&+\xi^3(\lambda^6+\sigma^6+\mu^6+\kappa^6)G_6 + \xi^2(\lambda^4\sigma^2+\lambda^2\sigma^4+\lambda^4\mu^2+\lambda^2\mu^4+\lambda^4\kappa^2+\lambda^2\kappa^4+\sigma^4\mu^2+\sigma^2\mu^4 \\
&+\sigma^4\kappa^2+\sigma^2\kappa^4+\mu^4\kappa^2+\mu^2\kappa^4)G_{4+2}+6\xi(\lambda^2\sigma^2\mu^2+\lambda^2\sigma^2\kappa^2+\lambda^2\mu^2\kappa^2+\sigma^2\mu^2\kappa^2)G_{2+2+2} \\
&+2\xi^2(\lambda^3\sigma^3+\lambda^3\mu^3+\lambda^3\kappa^3+\sigma^3\mu^3+\sigma^3\kappa^3+\mu^3\kappa^3)G_{3+3}+\xi^3(\lambda^7+\sigma^7+\mu^7+\kappa^7)H_7 \\
&+\xi^2(\lambda^5\sigma^2+\lambda^2\sigma^5+\lambda^5\mu^2+\lambda^2\mu^5+\lambda^5\kappa^2+\lambda^2\kappa^5+\sigma^5\mu^2+\sigma^2\mu^5+\sigma^5\kappa^2+\sigma^2\kappa^5+\mu^5\kappa^2+\mu^2\kappa^5)H_{5+2} \\
&+2\xi(\lambda^3\sigma^2\mu^2+\lambda^3\sigma^2\kappa^2+\lambda^3\mu^2\kappa^2+\sigma^3\lambda^2\mu^2+\sigma^3\lambda^2\kappa^2+\sigma^3\mu^2\kappa^2+\mu^3\lambda^2\sigma^2+\mu^3\lambda^2\kappa^2+\mu^3\sigma^2\kappa^2 \\
&+\kappa^3\lambda^2\sigma^2+\kappa^3\lambda^2\mu^2+\kappa^3\sigma^2\mu^2)H_{3+2+2}+\xi^2(\lambda^4\sigma^3+\lambda^3\sigma^4+\lambda^4\mu^3+\lambda^3\mu^4+\lambda^4\kappa^3+\lambda^3\kappa^4+\sigma^4\mu^3+\sigma^3\mu^4 \\
&+\sigma^4\kappa^3+\sigma^3\kappa^4+\mu^4\kappa^3+\mu^3\kappa^4)H_{4+3}+\xi^3(\lambda^8+\sigma^8+\mu^8+\kappa^8)I_8+\xi^2(\lambda^6\sigma^2+\lambda^2\sigma^6+\lambda^6\mu^2+\lambda^2\mu^6 \\
&+\lambda^6\kappa^2+\lambda^2\kappa^6+\sigma^6\mu^2+\sigma^2\mu^6+\sigma^6\kappa^2+\sigma^2\kappa^6+\mu^6\kappa^2+\mu^2\kappa^6)I_{6+2}+2\xi(\lambda^4\sigma^2\mu^2+\lambda^4\sigma^2\kappa^2+\lambda^4\mu^2\kappa^2 \\
&+\sigma^4\lambda^2\mu^2+\sigma^4\lambda^2\kappa^2+\sigma^4\mu^2\kappa^2+\mu^4\lambda^2\sigma^2+\mu^4\lambda^2\kappa^2+\mu^4\sigma^2\kappa^2+\kappa^4\lambda^2\sigma^2+\kappa^4\lambda^2\mu^2+\kappa^4\sigma^2\mu^2)I_{4+2+2} \\
&+24\lambda^2\sigma^2\mu^2\kappa^2 I_{2+2+2+2}+\xi^2(\lambda^5\sigma^3+\lambda^3\sigma^5+\lambda^5\mu^3+\lambda^3\mu^5+\lambda^5\kappa^3+\lambda^3\kappa^5+\sigma^5\mu^3+\sigma^3\mu^5+\sigma^5\kappa^3 \\
&+\sigma^3\kappa^5+\mu^5\kappa^3+\mu^3\kappa^5)I_{5+3}+2\xi(\lambda^2\sigma^3\mu^3+\lambda^2\sigma^3\kappa^3+\lambda^2\mu^3\kappa^3+\sigma^2\lambda^3\mu^3+\sigma^2\lambda^3\kappa^3+\sigma^2\mu^3\kappa^3 \\
&+\mu^2\lambda^3\sigma^3+\mu^2\lambda^3\kappa^3+\mu^2\sigma^3\kappa^3+\kappa^2\lambda^3\sigma^3+\kappa^2\lambda^3\mu^3+\kappa^2\sigma^3\mu^3)I_{3+2+2}+2\xi^2(\lambda^4\sigma^4+\lambda^4\mu^4+\lambda^4\kappa^4+\sigma^4\mu^4 \\
&+\sigma^4\kappa^4+\mu^4\kappa^4)I_{4+4}+\xi^3(\lambda^9+\sigma^9+\mu^9+\kappa^9)J_9+\xi^2(\lambda^7\sigma^2+\lambda^2\sigma^7+\lambda^7\mu^2+\lambda^2\mu^7+\lambda^7\kappa^2+\lambda^2\kappa^7 \\
&+\sigma^7\mu^2+\sigma^2\mu^7+\sigma^7\kappa^2+\sigma^2\kappa^7+\mu^7\kappa^2+\mu^2\kappa^7)J_{7+2}+2\xi(\lambda^5\sigma^2\mu^2+\lambda^5\sigma^2\kappa^2+\lambda^5\mu^2\kappa^2+\sigma^5\lambda^2\mu^2 \\
&+\sigma^5\lambda^2\kappa^2+\sigma^5\mu^2\kappa^2+\mu^5\lambda^2\sigma^2+\mu^5\lambda^2\kappa^2+\mu^5\sigma^2\kappa^2+\kappa^5\lambda^2\sigma^2+\kappa^5\lambda^2\mu^2+\kappa^5\sigma^2\mu^2)J_{5+2+2} \\
&+6(\lambda^3\sigma^2\mu^2\kappa^2+\lambda^2\sigma^3\mu^2\kappa^2+\lambda^2\sigma^2\mu^3\kappa^2+\lambda^2\sigma^2\mu^2\kappa^3)J_{3+2+2+2}+\xi(\lambda^4\sigma^3\mu^2+\lambda^4\sigma^2\mu^3+\lambda^4\sigma^3\kappa^2 \\
&+\lambda^4\sigma^2\kappa^3+\lambda^4\mu^3\kappa^2+\lambda^4\kappa^2\mu^3+\sigma^4\lambda^3\mu^2+\sigma^4\lambda^2\mu^3+\sigma^4\lambda^3\kappa^2+\sigma^4\lambda^2\kappa^3+\sigma^4\mu^3\kappa^2+\sigma^4\mu^2\kappa^3 \\
&+\mu^4\lambda^3\sigma^2+\mu^4\lambda^2\sigma^3+\mu^4\lambda^3\kappa^2+\mu^4\lambda^2\kappa^3+\mu^4\sigma^3\kappa^2+\mu^4\sigma^2\kappa^3+\kappa^4\lambda^3\sigma^2+\kappa^4\lambda^2\sigma^3+\kappa^4\lambda^3\mu^2 \\
&+\kappa^4\lambda^2\mu^3+\kappa^4\sigma^3\mu^2+\kappa^4\sigma^2\mu^3)J_{4+3+2}+\xi^2(\lambda^6\sigma^3+\lambda^3\sigma^6+\lambda^6\mu^3+\lambda^3\mu^6+\lambda^6\kappa^3+\lambda^3\kappa^6+\sigma^6\mu^3+\sigma^3\mu^6 \\
&+\sigma^6\kappa^3+\sigma^3\kappa^6+\mu^6\kappa^3+\mu^3\kappa^6)J_{6+3}+\xi^2(\lambda^5\sigma^4+\lambda^4\sigma^5+\lambda^5\mu^4+\lambda^4\mu^5+\lambda^5\kappa^4+\lambda^4\kappa^5+\sigma^5\mu^4+\sigma^4\mu^5 \\
&+\sigma^5\kappa^4+\sigma^4\kappa^5+\mu^5\kappa^4+\mu^4\kappa^5)J_{5+4}+6\xi(\lambda^3\sigma^3\mu^3+\lambda^3\sigma^3\kappa^3+\lambda^3\kappa^3\mu^3+\sigma^3\mu^3\kappa^3)J_{3+3+3} \quad .
\end{aligned} \tag{64}$$

Evidently, for simplexes Eq. (63) requires $(p+1)^4$ function evaluations to compute Σ_4, $(p+1)^3$ function evaluations to compute Σ_3, etc. For hypercubes, with the simplex vertices replaced by the

$2p$ points of Eq. (49), and $\xi = 2p$, Eq. (63) requires $(2p)^4$ function evaluations to compute Σ_4, $(2p)^3$ function evaluations to compute Σ_3, etc. Since it is known that the minimal number of function evaluations for a simplex integration method of order $2t+1$ involves $p^t/t! + O(p^{t-1})$ function calls (Stroud (1971), Grundmann and Möller (1978)), and for a hypercube integration method of order $2t+1$ involves $(2p)^t/t! + O(p^{t-1})$ function calls (Lyness, 1965), we will take the leading Σs in our integration formulas to have equal arguments, e.g. $\Sigma_4(\lambda,\lambda,\lambda,\lambda)$, $\Sigma_3(\lambda,\lambda,\lambda,\lambda)$, etc. This allows the parameterized integration formulas constructed below to have an optimal leading order power dependence on the space dimension p (but reflecting the parameter freedom, the non-leading power terms will not in general be minimal). In the computer program, the following formulas are useful in evaluating the sums using a minimum number of function calls,

$$\begin{aligned}
\Sigma_4(\lambda,\lambda,\lambda,\lambda) =& 24\sum_{a<b<c<d} f(\lambda(\tilde{x}_a+\tilde{x}_b+\tilde{x}_c+\tilde{x}_d)) + 12\sum_a \sum_{b\neq a,\, c\neq a,\, b<c} f(2\lambda\tilde{x}_a + \lambda(\tilde{x}_b+\tilde{x}_c)) \\
&+6\sum_a\sum_{b<a} f(2\lambda(\tilde{x}_a+\tilde{x}_b)) + 4\sum_a\sum_{b\neq a} f(3\lambda\tilde{x}_a+\lambda\tilde{x}_b) + \sum_a f(4\lambda\tilde{x}_a) \quad , \\
\Sigma_3(\lambda,\lambda,\lambda) =& 6\sum_{a<b<c} f(\lambda(\tilde{x}_a+\tilde{x}_b+\tilde{x}_c)) + 3\sum_a\sum_{b\neq a} f(2\lambda\tilde{x}_a+\lambda\tilde{x}_b) + \sum_a f(3\lambda\tilde{x}_a) \quad , \\
\Sigma_2(\lambda,\lambda) =& 2\sum_a\sum_{b<a} f(\lambda(\tilde{x}_a+\tilde{x}_b)) + \sum_a f(2\lambda\tilde{x}_a) \quad , \\
\Sigma_3(2\lambda,\lambda,\lambda) =& 2\sum_a \sum_{b\neq a,\, c\neq a,\, b<c} f(2\lambda\tilde{x}_a+\lambda(\tilde{x}_b+\tilde{x}_c)) + 2\sum_a\sum_{b\neq a} f(3\lambda\tilde{x}_a+\lambda\tilde{x}_b) \\
&+2\sum_a\sum_{b<a} f(2\lambda(\tilde{x}_a+\tilde{x}_b)) + \sum_a f(4\lambda\tilde{x}_a) \quad , \\
\Sigma_2(3\lambda,\lambda) =& \sum_a\sum_{b\neq a} f(3\lambda\tilde{x}_a+\lambda\tilde{x}_b) + \sum_a f(4\lambda\tilde{x}_a) \quad , \\
\Sigma_2(2\lambda,\lambda) =& \sum_a\sum_{b\neq a} f(2\lambda\tilde{x}_a+\lambda\tilde{x}_b) + \sum_a f(3\lambda\tilde{x}_a) \quad .
\end{aligned} \tag{65}$$

With these preliminaries in hand, we are now ready to set up integration formulas of first through fourth, fifth, seventh, and ninth order, for integrals over general simplexes.

A. First through third order formulas

We begin here with integration formulas of first through third order, which may be more useful than high order formulas for integrating functions that are highly irregular, or as explained later on, for integrations in low dimensional spaces. Two different first order accurate estimates of the integral of Eq. (62) are clearly

$$\begin{aligned} I_a =& \Sigma_1(\lambda)/\xi = A + \text{second order} \quad , \\ I_b =& f(\tilde{0}) = A \quad , \end{aligned} \tag{66}$$

with $\tilde{x} = \tilde{0}$ the simplex centroid. Evidently I_b is the dimension p analog of the dimension one center-of-bin rule, and when $\lambda = 1$, I_a is the dimension p analog of the dimension one trapezoidal rule.

To get a second order accurate formula, we have to match the terms

$$A + \frac{p\,!}{(p+2)!} C_2 \tag{67}$$

in Eq. (62). Solving $\Sigma_1(\lambda) = \xi A + \lambda^2 C_2 + ...$ for C_2, we get

$$C_2 \simeq [\Sigma_1(\lambda) - \xi A]/\lambda^2 \quad , \tag{68}$$

which when substituted into Eq. (67) gives the second order accurate formula

$$I = \left(1 - \frac{p!}{(p+2)!}\frac{\xi}{\lambda^2}\right) f(\tilde{0}) + \frac{p\,!}{(p+2)!}\frac{1}{\lambda^2}\Sigma_1(\lambda) \quad . \tag{69}$$

Using two different parameter values $\lambda_{a,b}$ gives two different second order accurate estimates $I_{a,b}$ of the integral.

We give two different methods of getting a third order accurate formula, both of which will play a role in the methods for getting higher odd order formulas. We first note that for $\lambda = 2/(p+3)$, we have

$$\Sigma_1(\lambda) = \xi A + \lambda^2[C_2 + 2D_3/(p+3)] \quad , \tag{70}$$

and so the coefficients of C_2 and D_3 are in the same ratio as appears in Eq. (62). Hence defining an overall multiplicative factor κ_1 to make both terms match in magnitude, and adding a multiple κ_0 of A to make this term match, we get a third order accurate formula

$$\begin{aligned} I_a =& \kappa_1 \Sigma_1(\lambda) + \kappa_0 f(\tilde{0}) \quad , \\ \kappa_1 =& \frac{p\,!}{(p+2)!}\lambda^{-2} = \frac{(p+3)^2}{4(p+2)(p+1)} \quad , \\ \kappa_0 =& 1 - \xi\kappa_1 \quad . \end{aligned} \tag{71}$$

An alternative method of getting a third order accurate formula is to look for a match by writing

$$\begin{aligned} I_b =&\bar{\kappa}_0 f(\tilde{0}) + \sum_{i=1}^{2} \kappa_1^i \Sigma_1(\lambda_1^i) \\ =&\bar{\kappa}_0 A + \sum_{i=1}^{2} \kappa_1^i [\xi A + (\lambda_1^i)^2 C_2 + (\lambda_1^i)^3 D_3] \\ =&A + \frac{p\,!}{(p+2)!} C_2 + \frac{2p\,!}{(p+3)!} D_3 + \ldots \quad . \end{aligned} \tag{72}$$

Equating coefficients of A we get

$$\bar{\kappa}_0 = 1 - \xi \sum_{i=1}^{2} \kappa_1^i \quad , \tag{73}$$

while equating coefficients of C_2 and D_3, we obtain a system of two simultaneous equations for κ_1^i , $i = 1, 2$,

$$\begin{aligned} q_1 =& w_1 + w_2 \quad , \\ q_2 =& \lambda_1^1 w_1 + \lambda_1^2 w_2 \quad , \end{aligned} \tag{74}$$

where we have abbreviated

$$\begin{aligned} q_1 =& \frac{p\,!}{(p+2)!} \ , \quad q_2 = \frac{2p\,!}{(p+3)!} \quad , \\ w_i =& \kappa_1^i (\lambda_1^i)^2 \ , \quad i = 1, 2 \quad . \end{aligned} \tag{75}$$

This set of equations can be immediately solved to give

$$\begin{aligned} w_1 =& \frac{\lambda_1^2 q_1 - q_2}{\lambda_1^2 - \lambda_1^1} \quad , \\ w_2 =& \frac{\lambda_1^1 q_1 - q_2}{\lambda_1^1 - \lambda_1^2} \quad , \end{aligned} \tag{76}$$

giving a second third order accurate formula for any nondegenerate λ_1^1 and λ_1^2 lying in the interval (0,1). We will see later on, in discussing higher odd order integration formulas, that this is our first encounter with a Vandermonde system of equations.

B. Fourth order formula

Although we will subsequently focus on odd-order formulas, we next derive a fourth order formula, which follows a different pattern. Referring to Eq. (62), we see that to get a fourth order formula we have to use the sums of Eq. (63) to match the coefficients of A, C_2, D_3, E_4, and E_{2+2}. Since only the final one of these, E_{2+2}, involves two partitions of 4, we can use $\Sigma_2(\lambda,\lambda)$ to extract this, with any positive value of $\lambda \leq \frac{1}{2}$. Since the simplex subdivision algorithm uses the midpoints $\frac{1}{2}(x_a + x_b)$ as the vertices of the subdivided simplex, an efficient way to proceed in this case is to take $\lambda = \frac{1}{2}$ in $\Sigma_2(\lambda,\lambda)$, so that what is needed is the function value at the midpoints, and to compute these function values as part of the subdivision algorithm. This also yields the function values at the vertices of the subdivided simplex. We can get A from $f(\tilde{x}_c)$, and we can fit C_2, D_3, and E_4 by evaluating $\Sigma_1(\lambda)$ with three distinct values of λ. One of these values can be taken as $\lambda = 1$, corresponding to the function values at the simplex vertices. The other two are free parameters, and by making two different choices for one of these, we get two different fourth order evaluations of the integral.

We worked out the fourth order program before proceeding systematically to the odd order cases, and so used a different notation from that of Eq. (63). Let us write f_c, f_v, and f_s for the sums of function values at the centroid, the vertices, and the side midpoints,

$$\begin{aligned} f_c =& f(\tilde{x}_c) \quad , \\ f_v =& \frac{1}{p+1}\sum_a f(\tilde{x}_a) \quad , \\ f_s =& \frac{1}{(p+1)p}\sum_{a\neq b} f(\frac{1}{2}(\tilde{x}_a + \tilde{x}_b)) \quad . \end{aligned} \tag{77}$$

Let us also introduce, for $n > m$, the definition

$$p_{nm} \equiv (p+m)(p+m+1)...(p+n) \quad . \tag{78}$$

A simple calculation then shows that through fourth order terms, we have

$$\begin{aligned} &\frac{1}{V}\int_{\text{simplex}} dx_1...dx_p f(\tilde{x}) - \frac{8p}{p_{42}}(f_s + \frac{1}{p}f_v) \\ &= k_0 A + k_2 C_2 + k_3 D_3 + k_4 E_4 \quad , \end{aligned} \tag{79}$$

with coefficients given by

$$\begin{aligned} k_0 =& 1 - \frac{8(p+1)}{p_{42}} \quad , \\ k_2 =& \frac{1}{p_{21}} - \frac{4}{p_{42}} \quad , \\ k_3 =& \frac{2}{p_{31}} - \frac{2}{p_{42}} \quad , \\ k_4 =& \frac{6}{p_{41}} - \frac{1}{p_{42}} \quad . \end{aligned} \tag{80}$$

Defining now

$$f_\lambda = \frac{1}{p+1} \sum_a f(\lambda \tilde{x}_a) \quad , \tag{81}$$

so that $f_1 = f_v$, we find that through fourth order,

$$\begin{aligned} E_4 =& \frac{1}{\lambda_1 - \lambda_2}(t_1 - t_2), \\ t_j =& \frac{p+1}{1-\lambda_j}[f_v - f_c - \frac{1}{\lambda_j^2}(f_{\lambda_j} - f_c)] \ , \quad j = 1, 2 \quad , \\ D_3 =& \frac{1}{2} \sum_{j=1}^{2} [t_j - (\lambda_j + 1) E_4] \quad , \\ C_2 =& (p+1)(f_v - f_c) - D_3 - E_4 \quad . \end{aligned} \tag{82}$$

When substituted into Eq. (79), this gives a fourth order formula for the integral, with a second evaluation of the integral obtained by replacing λ_2 by a third, distinct value λ_3.

C. Fifth order formula

We turn next to deriving a fifth order formula. Referring to Eq. (62), we see that to get a fifth order formula we have to use the sums of Eq. (63) to match the coefficients of A, C_2, D_3, E_4, E_{2+2}, F_5, and F_{3+2}. Since at most two partitions appear, we can still get the leading two-partition terms from $\Sigma_2(\lambda, \lambda)$, but we must now impose a condition on λ to guarantee that E_{2+2} and F_{3+2} appear with coefficients in the correct ratio. From Eq. (62) we see that the ratio of the coefficient of F_{3+2} to that of E_{2+2} must be $2/(p+5)$, and from Eqs. (63) and (64) with $\lambda = \sigma$ and $\mu = \kappa = 0$, we see that this is obtained with

$$\lambda = \frac{2}{p+5} \quad , \tag{83}$$

which for any $p \geq 1$ obeys the condition $2\lambda < 1$. The overall coefficient of Σ_2 needed to fit E_{2+2} and F_{3+2} is easily seen to be

$$\kappa_2 = \frac{p\,!}{2\,!(p+4)\,!}\lambda^{-4} = \frac{(p+5)^4}{32p_{41}} \quad , \tag{84}$$

where we have used the abbreviated notation of Eq. (78). Thus we have, again from Eq. (64),

$$\kappa_2\Sigma_2(\lambda,\lambda) = \frac{p\,!}{(p+4)\,!}E_{2+2}+2\frac{p\,!}{(p+5)\,!}F_{3+2}+\kappa_2[\xi^2 A+\xi(2\lambda^2C_2+2\lambda^3D_3+2\lambda^4E_4+2\lambda^5F_5)] \quad , \tag{85}$$

with $\xi = p+1$. Since there are four single partition terms, we look for an integration formula of the form

$$\kappa_2\Sigma_2(\lambda,\lambda) + \sum_{i=1}^{4}\kappa_1^i\Sigma_1(\lambda_1^i) + \kappa_0 A \quad , \tag{86}$$

which is to be equated to the sum of terms through fifth order in Eq. (62).

The equation for matching the coefficient of A can immediately be solved in terms of the coefficients κ_1^i, giving

$$\kappa_0 = 1 - R_0 \ , \quad R_0 = \xi^2\kappa_2 + \xi\sum_{i=1}^{4}\kappa_1^i \quad . \tag{87}$$

The four equations for matching the coefficients of C_2, D_3, E_4, and F_5 give a $N=4$ Vandermonde system that determines the four coefficients κ_1^i. Writing an order N Vandermonde system in the standard form

$$\sum_{i=1}^{N} x_i^{k-1}w_i = q_k \ , \quad k = 1,...,N \quad , \tag{88}$$

the equations determining the κ_1^i take this form with

$$\begin{aligned} x_i =& \lambda_1^i \ , \quad w_i = \kappa_1^i(\lambda_1^i)^2 \quad , \\ q_1 =& \frac{1}{p_{21}} - 2\xi\kappa_2\lambda^2 \quad , \\ q_2 =& \frac{2}{p_{31}} - 2\xi\kappa_2\lambda^3 \quad , \\ q_3 =& \frac{6}{p_{41}} - 2\xi\kappa_2\lambda^4 \quad , \\ q_4 =& \frac{24}{p_{51}} - 2\xi\kappa_2\lambda^5 \quad . \end{aligned} \tag{89}$$

Solving this system of linear equations, for any nondegenerate values of the parameters $0 < \lambda_1^i < 1$, gives the coefficients κ_1^i and completes specification of the integration formula.

D. Vandermonde solvers

Since we will repeatedly encounter Vandermonde equations in setting up parameterized higher order integration formulas, both for simplexes and for hypercubes, we digress at this point to discuss methods of solving a Vandermonde system. The explicit inversion of the Vandermonde system is well known (see, e.g. Neagoe (1996), Heinen and Niederjohn (1997)), and takes the form

$$w_1 = \frac{q_N - S_1(x_2,...,x_N)q_{N-1} + S_2(x_2,...,x_N)q_{N-2} - ... + (-1)^{N-1}x_2....x_N q_1}{(x_1-x_2)(x_1-x_3)....(x_1-x_N)} \quad , \tag{90}$$

with $S_j(x_2,...,x_N)$ the sum of j-fold products of $x_2,...,x_N$,

$$\begin{aligned} S_1(x_2,...,x_N) =& x_2 + ... + x_N \quad , \\ S_2(x_2,...,x_N) =& x_2x_3 + ... + x_2x_N + x_3x_4 + ... + x_3x_N + ... + x_{N-1}x_N \quad , \end{aligned} \tag{91}$$

and so forth. The remaining unknowns w_2 through w_N are obtained from this formula by cyclic permutation of the indices $i = 1,...,N$ on the w_i and the x_i, with the q_k held fixed. For N not too large it is straightforward to program this solution, and we include subroutines for the $N = 2, 3, 4, 6, 8$ cases in the programs. This suffices to solve the Vandermonde equations appearing in the fifth through ninth order simplex formulas, and in the fifth through ninth order hypercube formulas derived below.

For large N, programming the explicit solution becomes inefficient and a better procedure is to use a compact algorithm for solving the Vandermonde equations for general N, based on polynomial operations, which has running time proportional to N^2. A good method of this type, that we have tested, is the algorithm vander.for given in the book *Numerical Recipes in Fortran* by Press et al. (1992). A similar algorithm for inverting the Vandermonde matrix (that we have not tested) can be found in an on-line paper of Dejnakarintra and Banjerdpongchai, searchable under the title "An Algorithm for Computing the Analytical Inverse of the Vandermonde Matrix".

E. Seventh order formula

To get a seventh order formula, we use the sums of Eq. (63) to match the coefficients appearing in Eq. (62) through the term H_{3+2+2}. Since at most three partitions appear, we can get the leading three-partition terms G_{2+2+2} and H_{3+2+2} from $\Sigma_3(\lambda,\lambda,\lambda)$ by imposing the condition

$$\lambda = \frac{2}{p+7} \quad , \tag{92}$$

which guarantees that their coefficients are in the correct ratio, and which for any $p \geq 1$ obeys the condition $3\lambda < 1$. The overall coefficient of Σ_3 needed to fit G_{2+2+2} and H_{3+2+2} is

$$\kappa_3 = \frac{p\,!}{3\,!(p+6)\,!}\lambda^{-6} = \frac{(p+7)^6}{384 p_{61}} \quad . \tag{93}$$

We now look for an integration formula of the form

$$\kappa_3 \Sigma_3(\lambda,\lambda,\lambda) + \kappa_2' \Sigma_2(2\lambda,\lambda) + \sum_{i=1}^{U} \kappa_2^i \Sigma_2(\lambda_2^i,\lambda_2^i) + \sum_{i=1}^{6} \kappa_1^i \Sigma_1(\lambda_1^i) + \kappa_0 A \quad , \tag{94}$$

with $U \leq 6$ since there are 6 two-partition terms to be matched. Equating coefficients of the two-partition terms, we find that the equations for $G_{4+2}-G_{3+3}$ and $H_{5+2}-H_{4+3}$ are both automatically satisfied by taking

$$\kappa_2' = 3\kappa_3 \quad . \tag{95}$$

This leaves only the two-partition terms E_{2+2}, F_{3+2}, G_{4+2}, and H_{5+2} to be matched, so we can take the upper limit in the Σ_2 summation as $U = 4$. The four coefficients κ_2^i are then determined by solving an $N = 4$ Vandermonde system with inhomogeneous terms $q2_i$, $\quad i = 1, ..., 4$,

$$\begin{aligned} x_i =& \lambda_2^i \ , \quad w_i = 2\kappa_2^i (\lambda_2^i)^4 \quad , \\ q2_1 =& \frac{1}{p_{41}} - (6\xi + 24)\kappa_3 \lambda^4 \quad , \\ q2_2 =& \frac{2}{p_{51}} - (6\xi + 36)\kappa_3 \lambda^5 \quad , \\ q2_3 =& \frac{6}{p_{61}} - (6\xi + 60)\kappa_3 \lambda^6 \quad , \\ q2_4 =& \frac{24}{p_{71}} - (6\xi + 108)\kappa_3 \lambda^7 \quad . \end{aligned} \tag{96}$$

We next have to match the 6 single partition terms, using Σ_1 sums. To save function calls, we take four of the parameters λ_1^i to be equal to $2\lambda_2^i$, with the other two λ_1^i taken as new, independent parameters. Equating the coefficients of the single partition terms C_2 through H_7 then gives a $N = 6$ Vandermonde system determining the coefficients κ_1^i, with inhomogeneous terms $q1_i$, $\quad i =$

$1, ..., 6,$

$$
\begin{aligned}
x_i =&\lambda_1^i \ , \quad w_i = \kappa_1^i(\lambda_1^i)^2 \quad , \\
q1_1 =&\frac{1}{p_{21}} - 2\xi \sum_{i=1}^{4} \kappa_2^i(\lambda_2^i)^2 - (3\xi^2 + 15\xi)\lambda^2\kappa_3 \quad , \\
q1_2 =&\frac{2}{p_{31}} - 2\xi \sum_{i=1}^{4} \kappa_2^i(\lambda_2^i)^3 - (3\xi^2 + 27\xi)\lambda^3\kappa_3 \quad , \\
q1_3 =&\frac{6}{p_{41}} - \xi q2_1 - (3\xi^2 + 51\xi)\lambda^4\kappa_3 \quad , \\
q1_4 =&\frac{24}{p_{51}} - \xi q2_2 - (3\xi^2 + 99\xi)\lambda^5\kappa_3 \quad , \\
q1_5 =&\frac{120}{p_{61}} - \xi q2_3 - (3\xi^2 + 195\xi)\lambda^6\kappa_3 \quad , \\
q1_6 =&\frac{720}{p_{71}} - \xi q2_4 - (3\xi^2 + 387\xi)\lambda^7\kappa_3 \quad .
\end{aligned}
\tag{97}
$$

Note that in $q1_3, ..., q1_6$, the sums $2\sum_{i=1}^{4} \kappa_2^i(\lambda_2^i)^j$, $j = 4, ..., 7$ have been eliminated in terms of $q2_1, ..., q2_4$ by using the Vandermonde system of Eq. (96). Finally, matching the coefficient of A we get, using Eq. (95)

$$\kappa_0 = 1 - R_0 \ , \quad R_0 = (\xi^3 + 3\xi^2)\kappa_3 + \xi^2 \sum_{i=1}^{4} \kappa_2^i + \xi \sum_{i=1}^{6} \kappa_1^i \quad . \tag{98}$$

F. Ninth order formula

To get a ninth order formula, we use the sums of Eq. (63) to match the coefficients appearing in Eq. (62) through the final exhibited term $J_{3+2+2+2}$. Since at most four partitions appear, we can get the leading four-partition terms $J_{3+2+2+2}$ and $I_{2+2+2+2}$ from $\Sigma_4(\lambda, \lambda, \lambda, \lambda)$ by imposing the condition

$$\lambda = \frac{2}{p + 9} \quad , \tag{99}$$

which guarantees that their coefficients are in the correct ratio, and which for any $p \geq 1$ obeys the condition $4\lambda < 1$. The overall coefficient of Σ_4 needed to fit $J_{3+2+2+2}$ and $I_{2+2+2+2}$ is

$$\kappa_4 = \frac{p\,!}{4\,!(p + 8)\,!}\lambda^{-8} = \frac{(p + 9)^8}{6144 p_{81}} \quad . \tag{100}$$

We now look (with benefit of hindsight) for an integration formula of the form

$$\kappa_4\Sigma_4(\lambda,\lambda,\lambda,\lambda)+\kappa_3'\Sigma_3(2\lambda,\lambda,\lambda)+\kappa_2''\Sigma_2(3\lambda,\lambda)+\sum_{i=1}^{4}\kappa_3^i\Sigma_3(\lambda_3^i,\lambda_3^i,\lambda_3^i)+\sum_{i=1}^{4}\kappa_2^i\Sigma_2(2\lambda_3^i,\lambda_3^i)$$
$$+\sum_{i=1}^{6}\bar\kappa_2^i\Sigma_2(\lambda_2^i,\lambda_2^i)+\sum_{i=1}^{4}\kappa_1^i\Sigma_1(3\lambda_3^i)+\sum_{i=1}^{4}\bar\kappa_1^i\Sigma_1(2\lambda_3^i)+\kappa_0A \quad , \tag{101}$$

with four of the λ_2^i taken equal to the four λ_3^i, and the other two λ_2^i additional parameters. (Again, we reuse parameters wherever similar structures are involved in Eq. (65), so as to save function calls.)

We proceed to sketch the remaining calculation, without writing down the detailed form of the resulting Vandermonde equations (which can be read off from the programs, and is fairly complicated). We begin with the three-partition terms. The equations for $J_{5+2+2}-J_{4+3+2}$, $J_{4+3+2}-J_{3+3+3}$, and $I_{4+2+2}-I_{3+3+2}$ are all automatically satisfied by taking

$$\kappa_3' = 6\kappa_4 \quad . \tag{102}$$

This leaves four independent matching conditions for G_{2+2+2}, H_{3+2+2}, I_{4+2+2}, and J_{5+2+2}, which lead to a $N=4$ Vandermonde system determining the coefficients κ_3^i. We turn next to the two-partition terms. We find that the equations for $I_{6+2}-4.5I_{5+3}+3.5I_{4+4}$ and $J_{7+2}-3.5J_{6+3}+2.5J_{5+4}$ are automatically satisfied by taking

$$\kappa_2'' = 8\kappa_4 \quad . \tag{103}$$

The four equations for $G_{4+2}-G_{3+3}$, $H_{5+2}-H_{4+3}$, $I_{6+2}-I_{4+4}$, and $J_{7+2}+J_{6+3}-2J_{5+4}$ then give a $N=4$ Vandermonde system determining the coefficients κ_2^i. The remaining independent equations matching two-partition terms, for E_{2+2}, F_{3+2}, G_{4+2}, H_{5+2}, I_{6+2}, and J_{7+2}, then give a $N=6$ Vandermonde system determining the coefficients $\bar\kappa_2^i$.

Turning to the single partition terms, the eight equations obtained by matching coefficients for C_2, D_3, E_4, F_5, G_6, H_7, I_8, and J_9 give a $N=8$ Vandermonde system determining simultaneously the four coefficients κ_1^i and the four coefficients $\bar\kappa_1^i$. Finally, equating coefficients of A gives

$$\kappa_0 = 1-R \ , \quad R=(\xi^4+6\xi^3+8\xi^2)\kappa_4+\xi^3\sum_{i=1}^{4}\kappa_3^i+\xi^2\Big(\sum_{i=1}^{4}\kappa_2^i+\sum_{i=1}^{6}\bar\kappa_2^i\Big)+\xi\sum_{i=1}^{4}(\kappa_1^i+\bar\kappa_1^i) \quad . \tag{104}$$

G. Leading term in higher order

We have not systematically pursued constructing integration formulas of orders higher than ninth, but this should be possible by the same method. One can, however, see what the pattern will be for the leading term in such formulas. An integration formula of order $2t+1$ will have a leading term $\Sigma_t(\lambda, ..., \lambda)$, with t arguments λ. The only partition t terms appearing in the continuation of Eq. (62) will be $2+2+....+2$, containing t terms 2, and $3+2+...+2$, with one 3 and $t-1$ terms 2. Requiring these to have coefficients in the correct ratio restricts λ to be

$$\lambda = \frac{2}{p+2t+1} \quad , \tag{105}$$

and the leading term in the integration formula will be $\kappa_t\Sigma_t(\lambda, ..., \lambda)$, with κ_t given by

$$\kappa_t = \frac{p\,!}{t\,!(p+2t)\,!\lambda^{2t}} \quad . \tag{106}$$

Where nonleading terms give multiple equations of the same order, corresponding to inequivalent partitions of $2t+1$, $2t$, ..., one has to include terms proportional to $\Sigma_{t-1}(2\lambda, \lambda, ..., \lambda)$, $\Sigma_{t-2}(3\lambda, \lambda, ..., \lambda)$, and other such structures with asymmetric arguments summing to $t\lambda$, for the differences of these multiple equations to have consistent solutions. Once such multiplicities have been taken care of, the remaining independent equations will form a number of sets of Vandermonde equations.

VIII. DERIVATION OF THE SIMPLEX GENERATING FUNCTION

We give here a simple proof of the simplex generating function formulas of Eqs. (54) and (55), using the standard simplex integral

$$\int_{\text{standard simplex}} dx_1...dx_p\,(1-x_1-x_2-...-x_p)^{\nu_0}x_1^{\nu_1}...x_p^{\nu_p} = \frac{\prod_{a=0}^{p}\nu_a\,!}{(p+\sum_{a=0}^{p}\nu_a)\,!} \quad , \tag{107}$$

$\big($which we obtain later on as a specialization of the multinomial beta function integral of Eq. (168)$\big)$, the simplex volume formula of Eq. (42), and the expansion formulas of Eqs. (17) through (19). We start from

$$\int_{\text{simplex}} dx_1...dx_p \sum_{\nu_1...\nu_p=0}^{\infty} \frac{\prod_{i=1}^{p}(\tilde{x}_i t_i)^{\nu_i}}{\prod_{i=1}^{p}\nu_i\,!} = \int_{\text{simplex}} dx_1...dx_p e^{\sum_{i=1}^{p}\tilde{x}_i t_i} \quad , \tag{108}$$

and substitute the expansion of Eq. (17) on the right hand side, giving

$$\int_{\text{simplex}} dx_1...dx_p e^{\sum_{a=0}^{p}\alpha_a\sum_{i=1}^{p}\tilde{x}_{ai}t_i} \quad . \tag{109}$$

We now express the integral over the general simplex in terms of an integral over its barycentric coordinates α_a. Since $\sum_{a=0}^p \alpha_a = 1$, we can rewrite Eq. (19), by subtraction of x_0 from both sides, as

$$x - x_0 = \sum_{a=0}^{p} (x_a - x_0)\alpha_a = \sum_{a=1}^{p} (x_a - x_0)\alpha_a \quad . \tag{110}$$

From this we immediately find for the Jacobian

$$\left|\det\left(\frac{\partial x_1 ... \partial x_p}{\partial \alpha_1 ... \partial \alpha_p}\right)\right| = |\det(x_a - x_0)_i| = Vp\,! \quad , \tag{111}$$

with V the volume of the simplex. Since the $\alpha_a \ ,\ \ a = 1, ..., p$ span a standard simplex, we have transformed the integral of Eq. (109) to the form

$$Vp\,! \int_{\text{standard simplex}} d\alpha_1 ... d\alpha_p e^{\sum_{a=0}^p \alpha_a \sum_{i=1}^p \tilde{x}_{ai} t_i} \quad . \tag{112}$$

Expanding the exponential on the right in a power series, we have

$$Vp\,! \int_{\text{standard simplex}} d\alpha_1 ... d\alpha_p \sum_{\nu_1 ... \nu_p = 0}^{\infty} \frac{\prod_{a=0}^p (\alpha_a)^{\nu_a} (\sum_{i=1}^p \tilde{x}_{ai} t_i)^{\nu_a}}{\prod_{a=0}^p \nu_a\,!} \quad , \tag{113}$$

and then recalling that $\alpha_0 = 1 - \sum_{a=1}^p \alpha_a$, and using Eq. (107) to evaluate the integral over the standard simplex, we get

$$Vp\,! \sum_{\nu_1 ... \nu_p = 0}^{\infty} \frac{\prod_{a=0}^p (\sum_{i=1}^p \tilde{x}_{ai} t_i)^{\nu_a}}{(p + \sum_{a=0}^p \nu_a)\,!} \quad . \tag{114}$$

Let us now define P_n as the projector on terms with a total of n powers of the parameters t_i, since this is the projector that extracts the nth order moments. Applying P_n to Eq. (114), the denominator is converted to $(p + n)\,!$, which can then be pulled outside the sum over the ν_i, permitting these sums to be evaluated as geometric series,

$$\begin{aligned} &P_n Vp\,! \sum_{\nu_1 ... \nu_p = 0}^{\infty} \frac{\prod_{a=0}^p (\sum_{i=1}^p \tilde{x}_{ai} t_i)^{\nu_a}}{(p + \sum_{a=0}^p \nu_a)\,!} \\ =&\frac{Vp\,!}{(p+n)\,!} P_n \sum_{\nu_1 ... \nu_p = 0}^{\infty} \prod_{a=0}^{p} (\sum_{i=1}^{p} \tilde{x}_{ai} t_i)^{\nu_a} \\ =&\frac{Vp\,!}{(p+n)\,!} P_n \prod_{a=0}^{p} [1 - \sum_{i=1}^{p} \tilde{x}_{ai} t_i)]^{-1} \quad . \end{aligned} \tag{115}$$

Finally, applying to each factor in the product over a the rearrangement

$$(1 - y)^{-1} = \exp[-\log(1 - y)] = \exp[\sum_{s=1}^{\infty} \frac{y^s}{s}] \quad , \tag{116}$$

we get

$$\frac{Vp\,!}{(p+n)\,!}P_n\exp[\sum_{s=2}^{\infty}\frac{\sum_{a=0}^{p}[\sum_{i=1}^{p}\tilde{x}_{ai}t_i]^s}{s}] \quad , \tag{117}$$

where we have used the fact that the $s = 1$ term in the sum vanishes because $\sum_a \tilde{x}_{ai} = 0$. Comparing Eq. (117) with the starting equation Eq. (108), we get Eqs. (54) and (55).

IX. PARAMETERIZED HIGHER ORDER INTEGRATION FORMULAS FOR AXIS-PARALLEL HYPERCUBES

We turn in this section to the problem of deriving higher order integration formulas for axis-parallel hypercubes, in analogy with our treatment of the simplex case. Our formulas can be viewed as a generalization of those obtained by Lyness (1965) and McNamee and Stenger (1967). We consider an axis-parallel hypercube of half-side S, and denote by $\tilde{x}$ coordinates referred to the centroid of the hypercube. Through ninth order, the expansion of a general function $f(\tilde{x})$ is given as before by Eq. (52). Consider now the moment integrals

$$m(\nu) = \int_{\rm hypercube} dx_1...dx_p\tilde{x}_1^{\nu_1}...\tilde{x}_p^{\nu_p} \quad . \tag{118}$$

Since the limits of integration for each axis are $-S$, S, the moment integral factorizes and can be immediately evaluated as

$$\begin{aligned} m(\nu) =& \prod_{\ell=1}^{p}\frac{S^{\nu_\ell+1}}{\nu_\ell+1}[1+(-1)^{\nu_\ell}] \\ =&0 \quad \text{any } \nu_\ell \text{ odd} \quad , \\ =&\prod_{\ell=1}^{p}\frac{2S\,S^{\nu_\ell}}{\nu_\ell+1} \quad \text{all } \nu_\ell \text{ even} \\ =&V\prod_{\ell=1}^{p}\frac{S^{\nu_\ell}}{\nu_\ell+1} \quad \text{all } \nu_\ell \text{ even} \quad , \end{aligned} \tag{119}$$

with $V = (2S)^p$ in the final line the hypercube volume.

We now reexpress this moment integral in terms of sums over the set of $2p$ hypercube points $\tilde{x}_j$ given in Eq. (49), which will play a role in hypercube integration analogous to that played by simplex vertices in our treatment of simplex integration. In analogy with Eq. (58), we define the sum

$$S_{i_1...i_n} = \sum_{j=1}^{2p}\tilde{x}_{ji_1}...\tilde{x}_{ji_n} \quad . \tag{120}$$

Since $\tilde{x}_{ji} = S\delta_{ji}$ for $1 \le j \le p$ and $\tilde{x}_{ji} = -S\delta_{ji}$ for $p+1 \le j \le 2p$, this sum vanishes unless n is even and all of the indices i_1,...,i_n are equal, in which case it is equal to $2S^n$. The tensors of Eq. (120) and their direct products form a complete basis on which we can expand moment integrals over the hypercube. We have carried out this calculation two different ways. First, by matching the non-vanishing moment integrals through eighth order, we find

$$\begin{aligned}
\frac{1}{V}\int_{\text{hypercube}} dx_1...dx_p\tilde{x}_{i_1}\tilde{x}_{i_2} =& \frac{1}{6}S_{i_1i_2} \quad ,\\
\frac{1}{V}\int_{\text{hypercube}} dx_1...dx_p\tilde{x}_{i_1}\tilde{x}_{i_2}\tilde{x}_{i_3}\tilde{x}_{i_4} =& \frac{1}{36}(S_{i_1i_2}S_{i_3i_4}+S_{i_1i_3}S_{i_2i_4}+S_{i_1i_4}S_{i_2i_3}) - \frac{1}{15}S_{i_1i_2i_3i_4}\\
=& \frac{1}{36}(S_{i_1i_2}S_{i_3i_4}+2\,\text{terms}) - \frac{1}{15}S_{i_1i_2i_3i_4} \quad ,\\
\frac{1}{V}\int_{\text{hypercube}} dx_1...dx_p\tilde{x}_{i_1}\tilde{x}_{i_2}\tilde{x}_{i_3}\tilde{x}_{i_4}\tilde{x}_{i_5}\tilde{x}_{i_6} =& \frac{1}{216}(S_{i_1i_2}S_{i_3i_4}S_{i_5i_6}+14\,\text{terms})\\
&-\frac{1}{90}(S_{i_1i_2}S_{i_1i_2i_3i_4}+14\,\text{terms}) + \frac{8}{63}S_{i_1i_2i_3i_4i_5i_6} \quad ,\\
\frac{1}{V}\int_{\text{hypercube}} dx_1...dx_p\tilde{x}_{i_1}\tilde{x}_{i_2}\tilde{x}_{i_3}\tilde{x}_{i_4}\tilde{x}_{i_5}\tilde{x}_{i_6}\tilde{x}_{i_7}\tilde{x}_{i_8} =& \frac{1}{1296}(S_{i_1i_2}S_{i_3i_4}S_{i_5i_6}S_{i_7i_8}+104\,\text{terms})\\
&+\frac{1}{225}(S_{i_1i_2i_3i_4}S_{i_5i_6i_7i_8}+34\,\text{terms})\\
&-\frac{1}{540}(S_{i_1i_2}S_{i_3i_4}S_{i_5i_6i_7i_8}+209\,\text{terms})\\
&+\frac{4}{189}(S_{i_1i_2}S_{i_3i_4i_5i_6i_7i_8}+27\,\text{terms})\\
&-\frac{8}{15}S_{i_1i_2i_3i_4i_5i_6i_7i_8} \quad .
\end{aligned} \tag{121}$$

Combining these formulas with Eq. (52), and using a similar condensed notation to that used in the simplex case (but with the contractions referring now referring to the sums over the hypercube of Eq. (120)), we have for the integral of a general function over the hypercube, through ninth order,

$$\begin{aligned}
\frac{1}{V}\int_{\text{hypercube}} dx_1...dx_p f(\tilde{x}) =& A + \frac{1}{6}C_2 + \frac{1}{36}E_{2+2} - \frac{1}{15}E_4\\
&+\frac{1}{216}G_{2+2+2} - \frac{1}{90}G_{4+2} + \frac{8}{63}G_6\\
&+\frac{1}{1296}I_{2+2+2+2} + \frac{1}{225}I_{4+4} - \frac{1}{540}I_{4+2+2} + \frac{4}{189}I_{6+2} - \frac{8}{15}I_8\\
&+... \quad .
\end{aligned} \tag{122}$$

A second, and more general way, to obtain these results is to construct a generating function, analogous to that of Good and Gaskins used in the simplex case. We start from the formula

$$V^{-1}\int_{-S}^{S}dx_1 ... \int_{-S}^{S}dx_p\, e^{\,t_1x_1+...+t_px_p} = \prod_{\ell=1}^{p}\frac{\sinh St_\ell}{St_\ell} \quad , \tag{123}$$

and recall the power series expansion for the logarithm of $\frac{\sinh x}{x}$,

$$\begin{aligned}\log\left(\frac{\sinh x}{x}\right) =& \frac{1}{6}x^2 - \frac{1}{180}x^4 + \frac{1}{2835}x^6 - \frac{1}{37800}x^8 + ... \\ =& \sum_{n=1}^{\infty}\frac{(-1)^{n+1}2^{2n-1}B_{2n-1}}{n(2n)\,!}x^{2n} \quad ,\end{aligned} \tag{124}$$

with B_{2n-1} the Bernoulli numbers $B_1 = \frac{1}{6}$, $B_3 = \frac{1}{30}$, $B_5 = \frac{1}{42}$, $B_7 = \frac{1}{30}$, $B_9 = \frac{5}{66}$, ... Defining, in analogy with the simplex case (with $\tilde{x}_{ai}$ now the components of the $2p$ vectors $\tilde{x}_a$ of Eq. (49)),

$$\begin{aligned}W_u =& \sum_{a=1}^{2p}\left(\sum_{i=1}^{p}\tilde{x}_{ai}t_i\right)^u \\ =& 0 \;, \quad \text{u odd} \quad , \\ =& 2S^u\sum_{i=1}^{p}t_i^u \;, \quad \text{u even} \quad ,\end{aligned} \tag{125}$$

we rewrite Eq. (123), using Eq. (124), as

$$V^{-1}\int_{-S}^{S}dx_1 ... \int_{-S}^{S}dx_p\, e^{\,t_1x_1+...+t_px_p} = e^{\sum_{n=1}^{\infty}K_nW_{2n}} \quad , \tag{126}$$

with

$$K_n = \frac{(-1)^{n+1}2^{2n-2}B_{2n-1}}{n(2n)\,!} \quad . \tag{127}$$

Through eighth order, the right hand side of Eq. (126) is

$$\begin{aligned}e^{W_2/12-W_4/360+W_6/5670-W_8/75600+...} =& 1 \\ &+\frac{W_2}{12} \\ &-\frac{W_4}{360}+\frac{W_2^2}{288} \\ &+\frac{W_6}{5670}-\frac{W_2W_4}{4320}+\frac{W_2^3}{10368} \\ &-\frac{W_8}{75600}+\frac{W_4^2}{259200}+\frac{W_2W_6}{68040}-\frac{W_2^2W_4}{103680}+\frac{W_2^4}{497664}+... \quad .\end{aligned} \tag{128}$$

Applying the same rule as in the simplex case, of multiplying the coefficient of each term by a combinatoric factor $(s\,!)^t t\,!$ for each factor W_s^t, we recover the numerical coefficients in the expansions of Eqs. (121) and (122). This method can be readily extended to the higher order terms of these expansions.

We now follow the procedure used before in the simplex case. We match the expansion of Eq. (122) to sums over the function f evaluated at points within the hypercube, this time constructing these sums using parameter multiples of the $2p$ points of Eq. (49), which are the centroids of the maximal boundary hypercubes. The formulas of Eqs. (63), (64), and (65) still apply, with sums that extended from 0 to p in the simplex case extending now from 1 to $2p$, and with $\xi = p + 1$ in Eq. (64) replaced by $\xi = 2p$.

We proceed to set up integration formulas of first, third, fifth, seventh, and ninth order, for integrals over an axis-parallel hypercube. Since all odd order terms in the expansion of Eq. (52) integrate to zero by symmetry, to achieve this accuracy it suffices to perform a matching of the non-vanishing terms through zeroth, second, fourth, sixth, and eighth order, respectively. We will see that as a result of the absence of odd order terms, the higher order hypercube formulas are considerably simpler than their general simplex analogs.

A. First and third order formulas

We begin our derivation of odd order hypercube integration formulas with examples of first and third order accuracy, obtained by matching the first two terms in the expansion of Eq. (122),

$$I = A + \frac{1}{6}C_2 + ... \quad . \tag{129}$$

Proceeding in direct analogy with the first order formulas of Eq. (66) in the simplex case, we get

$$\begin{aligned} I_a =& \Sigma_1(\lambda)/\xi \quad , \\ I_b =& f(\tilde{0}) \quad , \end{aligned} \tag{130}$$

with $\tilde{x} = \tilde{0}$ the centroid of the hypercube, and $\xi = 2p$. These are again analogs of the trapezoidal and center-of-bin methods for the one dimensional case.

Similarly, in analogy with the second order accurate formula of Eq. (69) for the simplex, we get the third order accurate hypercube formula

$$I = \left(1 - \frac{\xi}{6\lambda^2}\right) f(\tilde{0}) + \frac{1}{6\lambda^2}\Sigma_1(\lambda) \quad . \tag{131}$$

For any two nondegenerate values $\lambda_{a,b}$ in the interval (0,1) this gives two different third order accurate estimates $I_{a,b}$ of the hypercube integral.

B. Fifth order formula

To get a fifth order formula, we have to use the sums of Eq. (63) to match the coefficients of A, C_2, E_{2+2}, and E_4 appearing on the first line of Eq. (122). Since there is now only a single two-partition term, E_{2+2}, we can extract it from $\Sigma_2(\lambda,\lambda)$ for any λ in the interval $(0,\frac{1}{2})$. So we look for a fifth order formula of the form

$$\kappa_2\Sigma_2(\lambda,\lambda)+\sum_{i=1}^{2}\kappa_1^i\Sigma_1(\lambda_1^i)+\kappa_0 A \quad . \tag{132}$$

Matching the coefficient of E_{2+2} gives

$$\kappa_2=\frac{1}{72\lambda^4} \quad , \tag{133}$$

while matching the coefficient of A gives

$$\kappa_0=1-R_0 \ , \quad R_0=\xi^2\kappa_2+\xi\sum_{i=1}^{2}\kappa_1^i \quad . \tag{134}$$

Matching the coefficients of C_2 and E_4 gives a $N=2$ Vandermonde system $\big($c.f. Eq. (88)$\big)$ with

$$\begin{aligned} x_i =&(\lambda_1^i)^2 \ , \quad w_i=\kappa_1^i(\lambda_1^i)^2 \quad , \\ q_1 =&\frac{1}{6}-\frac{\xi}{36\lambda^2} \quad , \\ q_2 =&\frac{-1}{15}-\frac{\xi}{36} \quad . \end{aligned} \tag{135}$$

C. Seventh order formula

To get a seventh order formula, we have to match the coefficients appearing on the first two lines of Eq. (122). Since there is only one three-partition term, G_{2+2+2}, we can extract it from $\Sigma_3(\lambda,\lambda,\lambda)$ for any λ in the interval $(0,\frac{1}{3})$. We look for a seventh order formula of the form

$$\kappa_3\Sigma_3(\lambda,\lambda,\lambda)+\sum_{i=1}^{2}\kappa_2^i\Sigma_2(\lambda_2^i,\lambda_2^i)+\sum_{i=1}^{3}\kappa_1^i\Sigma_1(\lambda_1^i)+\kappa_0 A \quad , \tag{136}$$

with matching the coefficient of G_{2+2+2} requiring

$$\kappa_3=\frac{1}{1296\lambda^6} \quad . \tag{137}$$

To reduce the number of function calls, we take $\lambda_1^{i+1} = 2\lambda_2^i$, $i = 1, 2$, with only λ_1^1 an additional parameter. Matching the coefficients of the two-partition terms E_{2+2} and G_{4+2} gives a $N = 2$ Vandermonde system with

$$\begin{aligned} x_i =&(\lambda_2^i)^2 \ , \quad w_i = 2\kappa_2^i(\lambda_2^i)^4 \quad , \\ q_1 =&\frac{1}{36} - \frac{\xi}{216\lambda^2} \quad , \\ q_2 =&-\frac{1}{90} - \frac{\xi}{216} \quad , \end{aligned} \tag{138}$$

while matching the coefficient of A gives

$$\kappa_0 = 1 - R_0 \ , \quad R_0 = \xi^3\kappa_3 + \xi^2\sum_{i=1}^{2}\kappa_2^i + \xi\sum_{i=1}^{3}\kappa_1^i \quad . \tag{139}$$

Matching coefficients of the single-partition terms C_2, E_4, and G_6 gives the $N = 3$ Vandermonde system with

$$\begin{aligned} x_i =&(\lambda_1^i)^2 \ , \quad w_i = \kappa_1^i(\lambda_1^i)^2 \quad , \\ q_1 =&\frac{1}{6} - \frac{\xi^2}{432\lambda^4} - 2\xi\sum_{i=1}^{2}\kappa_2^i(\lambda_2^i)^2 \quad , \\ q_2 =&-\frac{1}{15} - \frac{\xi}{36} + \frac{\xi^2}{432\lambda^2} \quad , \\ q_3 =&\frac{8}{63} + \frac{\xi}{90} + \frac{\xi^2}{432} \quad . \end{aligned} \tag{140}$$

D. Ninth order formula

To get a ninth order formula, we have to match the coefficients of all three lines of Eq. (122). Since there is only one four-partition term $I_{2+2+2+2}$, we can extract it from $\Sigma_4(\lambda, \lambda, \lambda, \lambda)$ for any λ in the interval $(0, \frac{1}{4})$. We look for a ninth order formula of the form

$$\kappa_4\Sigma_4(\lambda,\lambda,\lambda,\lambda)+\sum_{i=1}^{2}\kappa_3^i\Sigma_3(\lambda_3^i,\lambda_3^i,\lambda_3^i)+\kappa_2'\Sigma_2(3\lambda,\lambda)+\sum_{i=1}^{3}\bar{\kappa}_2^i\Sigma_2(\lambda_2^i,\lambda_2^i)+\sum_{i=1}^{4}\kappa_1^i\Sigma_1(\lambda_1^i)+\kappa_0 A \ , \tag{141}$$

with matching the coefficient of $I_{2+2+2+2}$ requiring

$$\kappa_4 = \frac{1}{31104\lambda^8} \quad . \tag{142}$$

To reduce the number of function calls, we take $\lambda_2^i = \lambda_3^i$, $i = 1, 2$, and $\lambda_1^i = 3\lambda_2^i$, $i = 1, 2, 3$, with only λ_2^3 and λ_1^4 additional parameters. Matching the coefficients of the three-partition terms G_{2+2+2} and I_{4+2+2} gives a $N = 2$ Vandermonde system with

$$\begin{aligned} x_i =&(\lambda_3^i)^2 \ , \quad w_i = 6\kappa_3^i(\lambda_3^i)^6 \quad , \\ q3_1 =&\frac{1}{216} - \frac{\xi}{1296\lambda^2} \quad , \\ q3_2 =& -\frac{1}{540} - \frac{\xi}{1296} \quad . \end{aligned} \tag{143}$$

Taking the difference of the matching equations for I_{6+2} and I_{4+4} determines κ_2' to be

$$\kappa_2' = \frac{237}{8164800\lambda^8} = \frac{237}{262.5}\kappa_4 \quad , \tag{144}$$

while matching the coefficient of A gives

$$\kappa_0 = 1 - R_0 \ , \quad R_0 = \xi^4\kappa_4 + \xi^3\sum_{i=1}^{2}\kappa_3^i + \xi^2(\kappa_2' + \sum_{i=1}^{3}\bar{\kappa}_2^i) + \xi\sum_{i=1}^{4}\kappa_1^i \quad . \tag{145}$$

Matching the remaining independent two-partition terms E_{2+2}, G_{4+2}, and I_{4+4} gives a $N = 3$ Vandermonde system with

$$\begin{aligned} x_i =&(\lambda_2^i)^2 \ , \quad w_i = 2\bar{\kappa}_2^i(\lambda_2^i)^4 \quad , \\ q2_1 =&\frac{1}{36} - \kappa_2' 18\lambda^4 - \frac{\xi^2}{2592\lambda^4} - 6\xi\sum_{i=1}^{2}\kappa_3^i(\lambda_3^i)^4 \quad , \\ q2_2 =& -\frac{1}{90} - \kappa_2' 90\lambda^6 - \frac{\xi}{216} + \frac{\xi^2}{2592\lambda^2} \quad , \\ q2_3 =&\frac{1}{225} - \kappa_2' 162\lambda^8 + \frac{\xi}{540} + \frac{\xi^2}{2592} \quad . \end{aligned} \tag{146}$$

Finally, matching coefficients of the single-partition terms C_2, E_4, G_6, and I_8 gives a $N = 4$ Vandermonde system with

$$\begin{aligned}
x_i =&(\lambda_1^i)^2 \ , \quad w_i = \kappa_1^i(\lambda_1^i)^2 \quad , \\
q1_1 =&\frac{1}{6} - \frac{\xi^3}{7776\lambda^6} - \kappa_2' 10\xi\lambda^2 - 3\xi^2 \sum_{i=1}^{2} \kappa_3^i(\lambda_3^i)^2 - 2\xi \sum_{i=1}^{3} \bar{\kappa}_2^i(\lambda_2^i)^2 \quad , \\
q1_2 =&-\frac{1}{15} - \frac{\xi^3}{7776\lambda^4} - \kappa_2' 82\xi\lambda^4 - 3\xi^2 \sum_{i=1}^{2} \kappa_3^i(\lambda_3^i)^4 - \xi\, q2_1 \quad , \\
q1_3 =&\frac{8}{63} - \frac{\xi^3}{7776\lambda^2} - \kappa_2' 730\xi\lambda^6 - \frac{1}{2}\xi^2\, q3_1 - \xi\, q2_2 \quad , \\
q1_4 =&-\frac{8}{15} - \frac{\xi^3}{7776} - \kappa_2' 6562\xi\lambda^8 - \frac{1}{2}\xi^2\, q3_2 - \xi\, q2_3 \quad .
\end{aligned} \tag{147}$$

E. Leading term in higher order

As in the simplex analysis, in the hypercube case we have not systematically pursued constructing integration formulas of orders higher than ninth, but this should be possible by the same method. Again, one can see what the pattern will be for the leading term in such formulas. An integration formula of order $2t + 1$ will have a leading term $\Sigma_t(\lambda, ..., \lambda)$, with t arguments λ. The only t-partition term appearing in the continuation of Eq. (122) will be $2 + 2 + ... + 2$, containing t terms 2. So λ can be taken to have any value in the interval $(0, \frac{1}{t})$, and the leading term in the integration formula will be $\kappa_t\Sigma_t(\lambda, ..., \lambda)$, with κ_t given by

$$\kappa_t = \frac{1}{t\,!6^t\lambda^{2t}} \quad . \tag{148}$$

Where nonleading terms give multiple equations of the same order, corresponding to inequivalent partitions of $2t$,... one has to include terms proportional to $\Sigma_{t-2}(3\lambda, \lambda, ..., \lambda)$, and other such structures with asymmetric arguments summing to $t\lambda$, for the differences of these multiple equations to have consistent solutions. Once such multiplicities have been taken care of, the remaining independent equations will form a number of sets of Vandermonde equations.

F. One dimension revisited: comparison of Vandermonde moment fitting to standard one dimensional methods

In this subsection we address several related issues. We first set up a one-dimensional analog of the moment fitting method that we have used in general p dimensions to develop higher order integration formulas. The one dimensional moment fitting equations can be satisfied by leaving the sampling points as free parameters, giving in any order a Vandermonde equation system to determine the weights assigned to the sampling points. Alternatively, the moment fitting equations can be satisfied by restricting the sampling points, which is what is done in Gaussian quadrature, which reduces the number of function calls. We then show that the fifth (and higher) order direct hypercube integration formulas for p dimensions, when restricted to one dimension, involve a larger number of function calls than needed for the case when the sampling points are all free parameters. We interpret this as resulting from linear dependencies in low dimension among the various terms appearing in the generating function of Eq. (128). We study these linear dependencies as a function of dimension p, and suggest that the integration rule of order $2t+1$ has redundant parameters when the spatial dimension $p < t$.

Let $f(x) = f_0 + f_1 x + f_2 x^2 + f_3 x^3 + f_4 x^4 + ...$ be a function that is power series expandable on the interval (-1,1), and consider the one dimensional integral

$$\begin{aligned} I =& \frac{1}{2}\int_{-1}^{1} f(x)dx \\ =& f_0 + \frac{f_2}{3} + \frac{f_4}{5} + \frac{f_6}{7} + \frac{f_8}{9} + ... \quad . \end{aligned} \tag{149}$$

Defining $\Sigma_1(\lambda)$ by

$$\Sigma_1(\lambda) = f(\lambda) + f(-\lambda)\,, \quad 0 < \lambda \leq 1 \quad , \tag{150}$$

we look for an integration formula of the form

$$\begin{aligned} I =& \kappa_0 f(0) + \sum_{i=1}^{n} \kappa^i \Sigma_1(\lambda^i) \\ =& f_0 + \sum_{i=1}^{n} \kappa^i [f_0 + f_2(\lambda^i)^2 + f_4(\lambda^i)^4 + f_6(\lambda^i)^6 + f_8(\lambda^i)^8 + ...] \quad . \end{aligned} \tag{151}$$

Matching the coefficients of f_ℓ between Eq. (149) and Eq. (151), we get the system of equations

$$\begin{aligned} 1 =& \kappa_0 + 2\sum_{i=1}^{n} \kappa^i \quad , \\ \frac{1}{3} =& 2\sum_{i=1}^{n} \kappa^i (\lambda^i)^2 \quad , \\ \frac{1}{5} =& 2\sum_{i=1}^{n} \kappa^i (\lambda^i)^4 \quad , \\ \frac{1}{7} =& 2\sum_{i=1}^{n} \kappa^i (\lambda^i)^6 \quad , \\ \frac{1}{9} =& 2\sum_{i=1}^{n} \kappa^i (\lambda^i)^8 \quad , \end{aligned} \tag{152}$$

and similarly if one wishes to go to higher order than ninth.

There are now two ways to proceed to solve the matching equations, to give a discrete approximation to the integral to a given order of accuracy. The first, which is what we have done in getting simplex and hypercube integration formulas, is to regard all of the λ^i as adjustable parameters, and to determine the coefficients κ^i to satisfy the system of equations of Eq. (152) to the needed order. Thus, to get a first order accurate formula, we take $I \simeq f(0)$ with all the κ^i equal to zero, which is the center-of-bin rule. To get a third order accurate formula we must take κ^1 as nonzero and solve the system

$$\begin{aligned} 1 =& \kappa_0 + 2\kappa^1 \quad , \\ \frac{1}{3} =& 2\kappa^1 (\lambda^1)^2 \quad . \end{aligned} \tag{153}$$

To get a fifth order accurate formula we must take both κ^1 and κ^2 as nonzero and solve the system

$$\begin{aligned} 1 =& \kappa_0 + 2(\kappa^1 + \kappa^2) \quad , \\ \frac{1}{3} =& 2\kappa^1 (\lambda^1)^2 + 2\kappa^2 (\lambda^2)^2 \quad , \\ \frac{1}{5} =& 2\kappa^1 (\lambda^1)^4 + 2\kappa^2 (\lambda^2)^4 \quad , \end{aligned} \tag{154}$$

to get a seventh order accurate formula we must take $\kappa^{1,2,3}$ as nonzero and solve the system

$$\begin{aligned}1 =&\kappa_0 + 2(\kappa^1 + \kappa^2 + \kappa^3) \quad ,\\ \frac{1}{3} =&2\kappa^1(\lambda^1)^2 + 2\kappa^2(\lambda^2)^2 + 2\kappa^3(\lambda^3)^2 \quad ,\\ \frac{1}{5} =&2\kappa^1(\lambda^1)^4 + 2\kappa^2(\lambda^2)^4 + 2\kappa^3(\lambda^3)^4 \quad ,\\ \frac{1}{7} =&2\kappa^1(\lambda^1)^6 + 2\kappa^2(\lambda^2)^6 + 2\kappa^3(\lambda^3)^6 \quad ,\end{aligned} \tag{155}$$

and so forth. Evidently, to get an order $2t + 1$ formula, we must take $n = t$, so that there are t distinct positive sampling points $\lambda^{1,...,t}$,and to determine the coefficients $\kappa^{1,...,t}$ we must solve an order $N = t$ Vandermonde system. The resulting order $2t + 1$ integration formula uses $2t + 1$ function values.

An alternative way to proceed is to adjust the values of the sampling points so that fewer of them are needed to satisfy the matching conditions. This is what is done in the well-known Gaussian integration method, which gives a more efficient scheme, in terms of the number of function calls, starting with third order. Referring to Eq. (153), we can evidently achieve a third order match by taking

$$\begin{aligned}2\kappa_0 =&0 \ , \quad 2\kappa^1 = 1 \quad ,\\ \lambda^1 =&\frac{1}{\sqrt{3}} \quad .\end{aligned} \tag{156}$$

Similarly, referring to Eq. (154), we can evidently achieve a fifth order match by taking

$$\begin{aligned}2\kappa_0 =&\frac{8}{9} \ , \quad 2\kappa_1 = \frac{5}{9} \quad ,\\ \lambda^1 =&\frac{\sqrt{3}}{\sqrt{5}} \quad .\end{aligned} \tag{157}$$

Proceeding in this way, we can obtain the general Gaussian integration formula, which for order $2t + 1$ integration involves t points. Of course, the usual derivation of the Gaussian integration rule does not proceed this way, but instead uses an argument based on one dimensional polynomial long division to relate the special points λ^i to zeros of the Legendre polynomials. Since in higher dimensions there is no analogous polynomial division rule, there is no universal higher dimensional analog of the Gaussian integration rule, although there are a multitude of special formulas using specially chosen sampling points in higher dimensions (see, e.g., Stroud (1971)). On the other hand, as we have seen, the method of keeping all the sampling points λ^i as free parameters, and solving a set of Vandermonde equations to get the coefficients κ^i, readily extends to higher dimensions.

Let us now examine the number of function evaluations required by our general moment fitting formulas for hypercubes, when restricted to one dimension. The first order center-of-bin formula requires just the one function evaluation $f(0)$, and so is the same in all methods. The third order formula of Eq. (131) involves $f(0)$ and one $\Sigma_1(\lambda)$, and so uses 3 function values, in agreement with Eq. (153), whereas the Gaussian formula needs 2 function values. Turning to the fifth order formula of Eq. (132), calculation of $\Sigma_2(\lambda, \lambda)$ requires 3 function evaluations, calculation of each of the two $\Sigma_1(\lambda^i)$ requires 2 function evaluations, and evaluation of $f(0)$ requires one function evaluation, for a total of 8 function evaluations. This is to be compared to the one dimensional moment fitting formula of Eq. (154) which requires 5 function evaluations, and the Gaussian method, which requires 3.

The reason that the fifth order integration formula for general p, when specialized to one dimension, requires more function evaluations than the moment fitting method of Eq. (154), is that whereas in two and higher dimensions W_4 and W_2^2 are linearly independent, in one dimension they are proportional to one another by virtue of the identity $t_1^4 = (t_1^2)^2$. Hence the term $\Sigma_2(\lambda, \lambda)$ in the general integration formula is not needed to get a match, and when this is dropped one has a formula identical in form to that of Eq. (154), requiring only 3 function calls. Turning to the higher order hypercube formulas, we see that the seventh order hypercube formula of Eq. (136) has redundant parameters and function calls for dimension $p < 3$, since in 2 dimensions W_6, W_2W_4 and W_2^3 are linearly dependent by virtue of the algebraic identity

$$0 = (t_1^2 + t_2^2)^3 - 3(t_1^2 + t_2^2)(t_1^4 + t_2^4) + 2(t_1^6 + t_2^6) \quad . \tag{158}$$

Similarly, the ninth order hypercube formula of Eq. (141) has redundant parameters and function calls for dimension $p < 4$, since in 3 dimensions W_8, W_4^2, W_2W_6, $W_2^2W_4$, and W_2^4 are linearly dependent by virtue of the identity

$$\begin{aligned} 0 =&(t_1^2 + t_2^2 + t_3^2)^4 - 6(t_1^8 + t_2^8 + t_3^8) + 3(t_1^4 + t_2^4 + t_3^4)^2 \\ &+8(t_1^2 + t_2^2 + t_3^2)(t_1^6 + t_2^6 + t_3^6) - 6(t_1^2 + t_2^2 + t_3^2)^2(t_1^4 + t_2^4 + t_3^4) \quad . \end{aligned} \tag{159}$$

These results suggest the conjecture that the hypercube formula of order $2t + 1$ will involve redundant parameters and function calls for dimension $p < t$, and we expect an analogous statement to apply for the simplex formulas derived by the moment fitting method in Sec. VII. This redundancy for small p is a consequence of the fact that the integration formulas that we have derived for simplexes and hypercubes are universal, in the sense that they involve the same number of parameters irrespective of the dimension p. As p increases, the number of sampling points increases, but the number of parameters, and the size of the Vandermonde systems needed to find coefficients, remains fixed.

X. FUNCTION CALLS NEEDED FOR INTEGRATION ROUTINES OF VARIOUS ORDERS

We summarize in this section the number of function calls needed for a single call to the integration routines of various orders. These are obtained by running the programs to integrate the function fcn=1, in which case the programs exit without subdividing the base region, giving the desired function call count for two samplings of the integral at the indicated order of accuracy, as well as unity as the output integral (since the programs all compute the integral over the base region, divided by the base region volume).

In Table VI we give the function call counting for the simplex integration programs of first through fourth, fifth, seventh, and ninth order. For comparison, in Table VII we give a similar table from the paper of Genz and Cools (2003), which gives the function call counting for one evaluation of the indicated order, plus a second evaluation at a lower order used for error estimation. Unlike our method, which proceeds directly from the vertices of a general simplex, the Genz and Cools program uses integration rules for a standard p-simplex, with an affine transformation needed to treat more general simplexes. Although not directly comparable, the two tables show that the strategy we have used, of incorporating a number of free parameters into the integration which can be used to give different samplings of the integrand, does not lead to an inefficiency of more than a factor of 2 to 3 compared to the method used by Genz and Cools.

In Table VIII we give the function call counting for the direct hypercube programs of first, third, fifth, seventh, and ninth order. For comparison, in Table IX we have tabulated $t^p + (t+1)^p$, with the odd order of integration n related to t by $n = 2t + 1$; this is the number of function calls needed if one uses a p-fold direct product of Gaussian integrations of indicated order, together with a p-fold direct product of Gaussian integrations of the next higher odd order to get an error estimate. One sees from these tables that for $t = 1, 2, 3$ our parameterized method is more efficient than direct product Gaussian for dimension $p \geq 4$, and for $t = 4$ the parameterized method is more

TABLE VI: Function calls by for simplex integration of order n in dimension p by method of Sec. VII

n $p \rightarrow$	1	2	3	4	5	6	7	8	9
1	3	4	5	6	7	8	9	10	11
2	5	7	9	11	13	15	17	19	21
3	7	10	13	16	19	22	25	28	31
4	10	16	23	31	40	50	61	73	86
5	20	31	43	56	70	85	101	118	136
7	37	71	117	176	249	337	441	562	701
9	74	168	316	531	827	1219	1723	2356	3136

TABLE VII: Function calls for simplex integration of order n in dimension p from Genz and Cools (2003)

n $p \rightarrow$	2	3	4	5	6	7	8	9
3	7	9	11	13	15	17	19	21
5	16	23	31	40	50	61	73	86
7	32	49	86	126	176	237	310	396
9	65	114	201	315	470	675	940	1276

efficient for $p \geq 5$. Since the number of function calls in the parameterized method is asymptotically polynomial of order $(2p)^t/t\,!$, whereas in the direct product Gaussian method it is exponential in p, the parameterized method becomes markedly more efficient for large dimension p.

These results reinforce the indication from the previous section that, as a very rough rule of thumb, in using integration routines with $n = 2t + 1$ in dimension p, one should avoid high order routines with $t > p$. This is true both because in low dimension the higher order routines have redundant function calls, and because the extra computation involved in using a high order routine is justified only when the 2^p scaling in the number of subregions, as the program subdivides from level to level, becomes large enough. However, this is only a very general criterion, since the optimum choice or choices of integration routine order will depend on the nature of the function being integrated. Moreover, in dimension $p = 1$ the programs are so fast on current computers that use of the fifth or seventh order integration routines, while not as efficient as Gaussian integration, still gives good results.

TABLE VIII: Function calls for hypercube integration of order n in dimension p by method of Sec. VIII

n $p \rightarrow$	1	2	3	4	5	6	7	8	9
1	3	5	7	9	11	13	15	17	19
3	5	9	13	17	21	25	29	33	37
5	12	27	46	69	96	127	162	201	244
7	21	69	153	281	461	701	1009	1393	1861
9	48	192	501	1059	1966	3338	5307	8021	11644

TABLE IX: Function calls for hypercube integration of order n in dimension p by comparison of two product Gaussian rules

n $p \rightarrow$	1	2	3	4	5	6	7
3	3	5	9	17	33	65	129
5	5	13	35	97	275	793	2315
7	7	25	91	337	1267	4825	18571
9	9	41	189	881	4149	19721	94509

XI. PUTTING IT ALL TOGETHER – SKETCH OF THE ALGORITHMS

We are now ready to give a sketch of the adaptive algorithms incorporating the elements described above. The basic algorithm starts from a base region, which acts as the initial level subregion, which is either a standard simplex, a Kuhn simplex (for hypercube integration treated by tiling with Kuhn simplexes), or a half-side 1 hypercube. It then proceeds recursively through higher levels of subdivision, by evaluating the integral using an integration method of order specified by the user with two different parameter choices, giving two estimates of the integral over the subregion divided by the subregion volume, which we denote by I_a(subregion) and I_b(subregion). (Dividing out the volume is convenient because of the $1/V$ factor appearing on the left hand side of Eqs. (62) and (122).) If the level number exceeds a user-specified value *ithinlev* which determines when thinning begins, then a thinning condition is applied. When the user-specified thinning function parameter *ithinfun* is given the value 1, the thinning condition used is

$$|I_a(\text{subregion}) - I_b(\text{subregion})| < \epsilon \quad , \tag{160}$$

with ϵ an error measure specified by the user. (Further thinning options will be discussed shortly.) If this condition is met, the results are retained as contributions to the I_a and I_b estimates of the integral divided by the base region volume, and the subregion is not further subdivided. If this condition is not met, then the subregion is subdivided into 2^p subregions, and the process is repeated. The process terminates when either the thinning condition is met for all subregions, or a limit to the number of levels of subdivision set by the user is reached. In the latter case, the contributions of the remaining subregions that have not satisfied the thinning condition are added to the I_a and I_b totals, as well as to the sum of the absolute values of the local subinterval errors.

With either termination, we get the final estimates of the integral divided by the base region volume,

$$\begin{aligned} I_a &\simeq \sum_{\text{subregions}} V(\text{subregion}) I_a(\text{subregion}) \quad , \\ I_b &\simeq \sum_{\text{subregions}} V(\text{subregion}) I_b(\text{subregion}) \quad . \end{aligned} \tag{161}$$

Here V(subregion) is the subregion volume divided by the base region volume, and since the subregions are a tiling of the initial base region, we have

$$\sum_{\text{subregions}} V(\text{subregion}) = 1 \quad . \tag{162}$$

From the difference of I_a and I_b we get an estimate of the error, given by

$$|\text{outdiff}| \equiv |I_a - I_b| \quad . \tag{163}$$

We can also (as in the one dimensional illustration) compute the sum of the absolute values of the local subinterval errors,

$$\text{errsum} \equiv \sum_{\text{subregions}} V(\text{subregion}) |I_a(\text{subregion}) - I_b(\text{subregion})| \quad . \tag{164}$$

Comparing Eqs. (161), (163), and (164), we see that errsum and |outdiff| obey the inequality

$$\text{errsum} \geq |\text{outdiff}| \quad , \tag{165}$$

with equality holding if $I_a - I_b$ has the same sign in all subregions. When the condition $|I_a(\text{subregion}) - I_b(\text{subregion})| < \epsilon$ is met for all subregions, errsum reduces, using Eq. (162), to

$$\text{errsum} < \epsilon \quad . \tag{166}$$

Hence to evaluate the integral to a relative error δ, one should choose

$$\epsilon \sim \delta |I_a| \quad . \tag{167}$$

Since I_a and I_b give the integral over the base region divided by the base region volume, to get the value of the integral without normalization by the base region volume, one must multiply these outputs by the base region volume V_0. For a standard simplex, $V_0 = 1/p\,!$, for a side 1 hypercube, $V_0 = 1$, while for a half-side 1 hypercube, $V_0 = 2^p$.

Note that the thinning condition determining whether to subdivide a subregion does *not* include a factor of the subregion volume; we are testing variances of the integrand as sampled over the subregion, not variances of the net contribution to the integral. This may seem counter-intuitive, but is motivated by the formulas of Eqs. (162)–(166), by simplicity, and by the fact that it works well in practice. The problem with including a subregion volume weighting factor in the thinning condition is that at a very fine level of subdivision, there are many subregions, and so small error contributions from each can add up to a large error in the total. Since the local test does not involve comparisons of the errors from different regions, the calculation in each subregion proceeds independently from that in all the others. The local thinning condition that we use is equivalent to the "Local Subdivision Strategy" described in the monographs of Krommer and Ueberhuber (1991) and Ueberhuber (1995) using a parameter ϵ_{abs}, which plays the role of our ϵ.

Using $|I_a(\text{subregion}) - I_b(\text{subregion})|$ as the basis for a thinning decision is only one possibility of many. More generally, given $A \equiv I_a(\text{subregion})$ and $B \equiv I_b(\text{subregion})$, one can take as the thinning function any function $f(A, B)$ with the properties $f(A, B) \geq 0$ and $f(A, B) = 0$ iff $A = B$, imposing now the thinning condition $f(A, B) < \epsilon$. In the programs, we have included three options, (1) $f(A, B) = |A - B|$ as in the discussion above, (2) $f(A, B) = |A - B|/|A + B|$, and (3) $f(A, B) = (A - B)^2$. In many cases, and in particular for polynomial integrals, we found their performance (with appropriate ϵ) to be similar, but for the singular integral $\int_0^1 dx \frac{1}{\sqrt{1-x^2}}$ we found choice (3) to perform considerably better than the other two.

Three versions of the basic algorithm are presented in each of the directories of programs. In the first, the algorithm subdivides until all subregions obey the thinning condition, or until a preset limit on the level of subdivisions is reached, which is dictated by the available memory. Typically, for simple integrands and moderate dimension p, this happens rather quickly, in other words, the algorithm has saturated capabilities of the machine memory, but not of the machine speed. In a second version labelled "r", the algorithm is "recirculated" by keeping, at a level limit set by the user which is chosen to avoid exceeding machine memory capabilities, all the subintervals that do not obey the thinning condition. These are then treated one at a time by the same algorithm, up to a second level limit again set by the user. This can take hours or days for high accuracy, high p computations, with a practical limit set by the speed capabilities of the machine. Finally, a third version labelled "m" takes the "recirculating" algorithm and parallelizes it using the MPI (message passing interface) protocol, by distributing to each process of a cluster a large number of the subintervals that do not obey the thinning condition , each of which is then processed by the algorithm sequentially. This speeds up the computation by a factor of the number of processes available. All routines are coded in double precision, but since the ninth order integration formulas involve large numbers in computing coefficients, double precision computation is not enough to give double precision accuracy results, so for the fifth, seventh, and ninth order routines in both the simplex and direct hypercube cases, we also give a quadruple precision $\big(\text{real}(16)\big)$ version of the programs.

The programs present the user with various options. By an appropriate choice of *ithinlev*, thinning can be delayed, or even suppressed entirely so that all subdivisions take place to the specified subdivision limits. This can give a check that subregions with large contributions, but accidentally small error estimates, have not been harvested prematurely, and when the programs are modified, gives a useful check that the tiling condition of Eq. (162) is obeyed. By a choice of *ithinfun*, the user can choose which of three preset thinning functions to use, or by modifying the subroutine containing these functions, the user can make another choice of thinning function. For simplex integration, the user can choose whether to use the recursive or the symmetric subdivision algorithm. The user can choose the accuracy of the integration method used: first through fourth, fifth, seventh, or ninth for simplex based routines, and first, third, fifth, seventh, and ninth for the direct hypercube routines. Finally, the user can modify the free parameters in the integration routines, so as to get different samplings of the integrand, which can give a useful assessment of whether the error estimates from the initially used sampling are realistic.

XII. TEST INTEGRALS; FALSE POSITIVES AND THEIR AVOIDANCE

For verifying the higher order integration programs, and for checking the operation of the adaptive programs, it is essential to have test integrals with known answers. For the standard simplex (c.f. Eqs. (30) and (31)), a useful formula is the multinomial beta function integral,

$$\int_{\text{standard simplex}} dx_1...dx_p\,(1-x_1-x_2-...-x_p)^{\alpha_0-1}x_1^{\alpha_1-1}...x_p^{\alpha_p-1} = \frac{\prod_{a=0}^{p}\Gamma(\alpha_a)}{\Gamma(\sum_{a=0}^{p}\alpha_a)} \quad , \tag{168}$$

with Γ the usual gamma function (see the Wikipedia article on Dirichlet distributions). When $\alpha_a - 1 = \nu_a \;, \quad a = 0,...,p$ with ν_a an integer, this can be rewritten as

$$\int_{\text{standard simplex}} dx_1...dx_p\,(1-x_1-x_2-...-x_p)^{\nu_0}x_1^{\nu_1}...x_p^{\nu_p} = \frac{\prod_{a=0}^{p}\nu_a\,!}{(p+\sum_{a=0}^{p}\nu_a)\,!} \quad . \tag{169}$$

The $\nu_0 = 0$ case of this formula is the formula given by Stroud (1971) $\big($see also Grundmann and Möller (1978)$\big)$ for the integral of a general monomial over the standard simplex.

For a unit hypercube, the corresponding formula is

$$\int_0^1 dx_1 ... \int_0^1 dx_p x_1^{\nu_1} ... x_p^{\nu_p} = \prod_{\ell=1}^{p} \frac{1}{\nu_\ell + 1} \quad , \tag{170}$$

while for a half-side 1 hypercube the corresponding monomial integrals are (c.f. Eqs. (118) and (119))

$$\int_{-1}^1 dx_1 ... \int_{-1}^1 dx_p \, x_1^{\nu_1} ... x_p^{\nu_p} = 2^p \prod_{\ell=1}^{p} \frac{1}{\nu_\ell + 1} \quad \text{for all } \nu_\ell \text{ even, and zero otherwise} \quad . \tag{171}$$

Testing the simplex programs with the integral of Eq. (169), and starting thinning at level 1, shows that when the order of the monomial is less than or equal to the order of the integration formula used, the iteration terminates at the initial level, and the difference between I_a and I_b is of order the computer truncation error. When a monomial is integrated that is of higher order than the integration formula used, with a small enough error measure ϵ, the adaptive program starts to subdivide the base region.

However, a more complicated pattern is seen for the hypercube integrals when evaluated by the direct hypercube algorithms, and this brings us to the issue of false positives. As in the simplex case, when thinning is started at level 1 and the order of the test monomial is less than or equal to the order of the integration formula used, the iteration terminates again at the initial level, and $I_a - I_b$ is of order the truncation error. However, when a monomial is integrated that is of higher order than the integration formula used, the adaptive program does not always start to iterate. For example, using the fifth order hypercube formula in dimension $p = 4$, the program iterates for the integrand x_1^6, but not for the integrand $x_1^2 x_2^2 x_3^2$. The reason is that the latter function, although of higher order than that of the integration formula, vanishes on the hyperplanes spanning the axes where the fifth order integration formula samples the integrand, and so the I_a and I_b evaluations give the same answer (zero), and the thinning condition is obeyed for arbitrarily small ϵ. This is an example of a *false positive*, in which the thinning condition is obeyed even though the actual error is large. Any sampling program for evaluating integrals is subject to false positives for functions that take special values (in our case zero, or a constant) on the sampling points. Since the sampling points in the simplex integration formulas are on oblique, rather than axis-parallel, lines or planes, this problem is not so readily seen with the multinomial test functions of Eq. (169), but we have nonetheless found examples of false positives. For example, using fifth order integration and symmetric subdivision, the $p = 5$ monomial $x(1)x(2)x(3)x(4)^2x(5)$, when computed with thinning starting at any level below 3, develops a false positive at level 2 and gives an answer that is wrong in the fourth decimal place, even though the output error measures suggest much higher accuracy.

There are several general ways to guard against false positives. The simplest is to use the freedom of choosing the parameter *ithinlev* to delay thinning until several subdivisions have taken place. False positives are most dangerous if they occur in the initial few levels, since these have the largest subregions, and if a subregion is prematurely harvested, there is a possibility of significant error. On the other hand, thinning becomes most important after several subdivisions have taken place, when the number of subregions is large. So there can be a useful tradeoff between starting thinning early and starting it late. If computer time permits, one can always do an *a posteriori* check by choosing *ithinlev* greater than the limit on the number of levels, which suppresses thinning altogether, and gives the approximate Riemann sum corresponding to the level of subdivision attained.

A second general way to guard against false positives is to compute the integral using alternative options, for example, using integration programs of several different orders, or where allowed as an option for simplex integrals, to use recursive instead of symmetric subdivision. In the fifth order $p = 5$ example noted above, changing to seventh order integration, or changing from symmetric to recursive subdivision while maintaining fifth order integration, both eliminate the false positive at level two.

A third way is to add a function with known integral to the integrand, which has significantly different local behavior, and to subtract its known integral from the total at the end. For example, in the hypercube case, consider the integral

$$0 = \int_{-1}^{1} \phi_q(x) \ , \quad \phi_q(x) = \frac{1}{(q+x)^2} - \frac{1}{q^2-1} \quad , \tag{172}$$

which exists for any $q > 1$. Adding a multiple of

$$\prod_{\ell=1}^{p} \phi_q(x_\ell) \tag{173}$$

to the test monomial integrands does not change the expected answer, but forces the adaptive program to start to subdivide at level 1 (for small enough ϵ) in all monomial cases. It is of course not necessary for the added function to have an integral that can be evaluated in closed form. In the $p = 5$ simplex case discussed above, we eliminated the false positive at level 2 by numerically integrating the function $(1+x(1))^{-1}$, and then adding a multiple of this function to the integrand and subtracting its integral from the answer. When adding such an auxiliary function, it is probably a good idea to rescale it so that its order of magnitude is similar to that of the integral being evaluated. Clearly there is an infinite variety of such auxiliary functions that can be added to the integrand, each of which shifts the false positive problem to a different part of integrand function space. Even when one is dealing with generic integrands, in which the program starts to subdivide as expected, adding such functions will alter the pattern of subdivision, and can be used (in addition to changing the integration formula parameters) to give further estimates of the errors in the output values $I_{a,b}$ provided by the integration algorithm.

We do not recommend just changing the integration formula parameters as a way of eliminating false positives. The reason is that the samplings in both the simplex and hypercube cases take place on hyperplanes that are determined by the general structure of the integration formulas, but do not vary as the parameters in the integration formulas are changed. So if a false positive is associated with a zero or constant integrand value on one of these hyperplanes, it will not be eliminated by changing the parameter values. Similar remarks apply to changing the thinning function as a way of eliminating false positives.

For related reasons we have not written into the programs another way of creating a criterion for thinning, the comparison of results from integration programs of different orders (say, of fifth and seventh order). In the simplex example discussed above, doing this would eliminate the false positive, since the seventh order routine uses sampling points that avoid the problematic hyperplanes sampled by the fifth order routine. However, in this case one may as well do two seventh order samplings to set up the thinning condition , and thus benefit from the higher accuracy accruing from use of the seventh order routine for smooth integrands.

XIII. DESCRIPTION OF PROGRAMS IN THE SEVEN DIRECTORIES

A. General description

The Fortran programs are grouped into 7 directories, named simplex123, simplex4, simplex579, simplex579_16, cube13, cube579, and cube579_16. All programs are valid for arbitrary dimension $p \geq 1$

The simplex programs all perform adaptive integration over a standard simplex or a Kuhn simplex with one vertex at the origin, using real(8) precision $\big($except for simplex579_16, which uses real(16)$\big)$. The programs in simplex123 perform first through third order integration, the programs in simplex4 perform fourth order integration, and the programs in simplex 579 perform fifth, seventh, or ninth order integration.

The same adaptive program treats both the standard and Kuhn simplex cases, with a subroutine argument "i_init" determining which initialization is used. Included in all the simplex packages are programs for integration over a side 1 hypercube with one vertex at the origin, by tiling with Kuhn simplexes followed by adaptive simplex integration.

The programs in simplex4 perform fourth order adaptive integration using a different subdivision strategy from that used in all the other cases. In simplex4 the simplex vertices are used as sampling points, with the side midpoints giving the vertices at the next level of subdivision. In all the other programs, only interior points of the simplex are used for sampling. Hence, the simplex4 programs cannot be used to integrate functions which have integrable singularities at the base simplex boundary, whereas the other programs can be used in this case.

The programs in cube13 perform first or third order adaptive integration, and those in cube579 perform fifth, seventh, or ninth order adaptive integration, over half-side 1 hypercubes centered on the origin, with real(8) precision. These programs use less memory (by roughly a factor $1/p$) than the hypercube tiling programs. The cube programs are valid for arbitrary dimension $p \geq 1$.

The programs in simplex579_16 are real(16) re-writings of those in simplex579, and the programs in cube579_16 are real(16) re-writings of those in cube579. The real(16) versions are obtained from the corresponding real(8) programs by making the following global substitutions: (1) Replace "d0" by "q0", (2) replace "implicit real(8)" by "implicit real(16)", (3) replace "dabs" by "qabs", (4) replace "d20.13" by "d32.36". These changes can be made using a "replace all" utility, since the strings that have to be modified do not occur anywhere else in the programs. Note that explicit data type declarations that override the implicit ones are *not* changed.

Each directory contains a package of subprograms, labeled respectively simplexsubs123.for, simplexsubs4.for, simplexsubs579.for, simplexsubs579_16.for, cubesubs123.for, cubesubs579.for, and cubesubs579_16.for. The subroutines in these packages do not have to be accessed by the user in normal operation of the adaptive programs. If they are accessed to alter the programs, we strongly recommend doing several test integrals before and after the changes, to make sure they still operate correctly. Each directory also contains a series of main program files, and each main program file contains the main program proper, as well as a subroutine setting up the function to be integrated, subroutines setting up the free parameters used in the parameterized integrations, a subroutine setting up three options for the thinning function, and in the case of the Kuhn tiling treatment of hypercubes, a subroutine symmetrizing the function to be integrated over all its variables. Each program that requires user setting of input parameters contains comment statements giving instructions. To run the programs, the user must compile and link the subroutine package in a directory with the appropriate main program file in the same directory.

As noted in the section on Vandermonde solvers, all programs are self-contained, since their subroutine packages include Vandermonde solvers that compute the explicit solution of the Vandermonde system for the relevant values of N. Because the ninth order simplex integration routines and associated Vandermonde equations involve large numbers in computing coefficients, use of real(16) is recommended if one wants to get answers with real(8) accuracy. Solving the Vandermonde equations to get the coefficient parameters for the integrations need be done only once before adaptive integration begins; this is done in the subroutines with names beginning with "ext", the output of which is then fed to the integration programs that are used repeatedly in the adaptive integration process.

There are three generic types of main programs in each directory. Those with names not ending in "r" or "m" execute adaptive integration to a subdivision level set by the user (and limited by machine memory). Those with names ending in "r" execute the "recirculating" routines, in which after the first stage of subdivision, the remaining subregions are subdivided sequentially in a second stage to a second level of subdivision set by the user. Those with names ending in "m" execute an MPI parallel version of the "recirculating" routines, in which after the first stage of subdivision, the remaining subregions are farmed out to the available processes for a second stage of subdivision to the second level of subdivision set by the user.

In order to conserve memory, the labeling of simplex and cube points and the simplex subdivision routines use a lattice built on integer(2) arithmetic. This allows 14 levels of subdivision in the initial stage, since 2^{14}=16384, which is half the maximum integer representable in integer(2). In order to go beyond 14 levels of subdivision in one stage, say to 30 levels of subdivision, one would have to replace 16384 in the subroutines by $2^{30} = 1,073,741,824$, which is half the maximum integer representable in integer(4), replace all integer(2) data type declarations by integer(4), and enlarge the level number limits in the programs. The explicit limits in the programs on the number of levels correspond to the requirement that the minimum integer(2) lattice spacing must not be smaller than 1, since in integer arithmetic 1/2 is replaced by 0. Program stages that pass on subdivided regions have a limit of 14 levels, while output stages that do not pass on subdivided regions have a limit of 15 levels. An exception to this rule is in the simplex4 programs, where there is an explicit division by 2 in the programs, and so the corresponding limits are 13 and 14. Note that in integer(2) arithmetic, $16384/2 + 16382/2 = 16384 \neq (16384 + 16384)/2 = (-32768)/2 = -16384$, which is why in the simplex4 integration program we have not regrouped added terms into parentheses.

The recirculating and MPI programs make use of the observation that symmetric (or recursive) subdivision of standard simplexes, symmetric and recursive subdivision of Kuhn simplexes, and hypercube subdivision, all give after ℓ subdivisions a subregion that fits within a hypercube of side $1/2^\ell$ (or $1/2^{\ell-1}$). This observation, which is an unproved conjecture supported by our numerical results in the case of standard simplexes, permits a doubling of the number of levels attainable within integer(2) arithmetic in the "r" and "m" programs, as follows. At the start of the second stage of subdivision, each subregion is translated by a shift vector and is rescaled by a factor which expands it to just fit within the initial lattice containing base region. This permits another 15 (or for recursive subdivision, 14) levels in the second stage (with corresponding limits in the simplex4 programs of 14 (or 13)), and so the "r" and "m" programs can subdivide to subregions that have a dimension $2^{-28} = 3.725 \times 10^{-9}$ of the base region dimension. Whether this can be attained in practice for a given dimension p of course depends on available machine memory. Subdivision limits appropriate to the various cases have been incorporated into the main programs.

Because simplex points are represented in integer(2) arithmetic, in order to apply the simplex subroutines to a starting simplex that does not have only 0s or 1s in the vertex coordinates (for example, an equilateral triangle), one would have to change the integer(2) data type declarations to real(4) for the programs to work correctly. This change increases the memory requirements, and should not be made unless needed. We note also that with the aim of conserving memory, we have used allocatable memory to store subregion information, allocating memory where needed at each level of subdivision, and deallocating memory when no longer used.

Finally, we note that the MPI programs are written using only simple MPI_Send and MPI_Recv commands. All processes simultaneously carry out the first stage of subdivision, and then each process of rank greater than 0 takes its share of the remaining subregions after the first stage and processes them further. This wastes some processor time, but avoids large data transfers. Only at the end, when all processes of rank greater than 0 have finished, is their output combined in process 0. Because MPI can only pass real(8) numbers as messages, the real(16) MPI programs give only real(8) output. (This is one of the reasons why the explicit real(8) declarations are not modified in the conversion substitutions leading to real(16) programs.) Nevertheless, the MPI programs compute the sensitive parts of the high order integrations in real(16), converting to real(8) only at the end when process outputs are combined.

To enhance readability of the programs, we have used indents to show the different levels of "if" chains, except in one place in the MPI programs, where we have given the "if", "else if", and "end if" lines statement numbers 97,98,99. We have not indented the contents of "do" loops, since these always begin and end with a statement number, and never with an unnumbered "enddo". (The one exception to this is in the subroutine BestLex used for the symmetrization step in the

Kuhn tiling programs for integration over hypercubes, which has been taken verbatim from H. D. Knoble's (1995) website.) This is of course a matter of taste; our feeling was that indenting both the "if" chains and "do" loops would result in so many levels of indents that readability of the programs would be decreased. We also remind the reader that the direct hypercube programs were written by minimal modification of the simplex programs, changing array arguments where needed (e.g., "$ip+1$" for simplexes becomes "$2*ip$" for hypercubes), but not changing array names. So the array names in the direct hypercube subroutines are not the ones that would naturally be chosen if these programs were written without reference to the simplex case.

B. Inputs

The main programs require the following inputs to be set by the user:

1. *ithinlev* tells the program when to begin thinning subregions, by harvesting those that obey the thinning condition of Eq. (160). Thinning begins when the total level number exceeds *ithin*. Thus, with $ithin = 0$, thinning begins at level 1, while if *ithin* is greater than or equal to the maximum total level number, there is no thinning.

2. *ithinfun* tells the program which thinning function option to use. As explained in Sec. X, $ithinfun = 1$ corresponds to a thinning function $f(A,B) = |A - B|$, $ithinfun = 2$ to $f(A,B) = |A - B|/|A + B|$, and $ithinfun = 3$ to $f(A,B) = (A - B)^2$.

3. *isubdivision* tells the simplex programs whether to use symmetric subdivision ($isubdivision = 1$) or recursive subdivision ($isubdivision = 2$). This parameter does not appear in the main programs in cube13, cube579, and cube579_16, where there is no choice of subdivision methods.

4. *iaccuracy* tells the programs to use the integration program of order *iaccuracy*. For example, in the simplex123 programs, to select third order accuracy one sets $iaccuracy = 3$, and in the simplex 579 programs, to select seventh order integration one sets $iaccuracy = 7$. This parameter does not appear in the main programs in simplex4, which uses only fourth order integration.

5. *ip* gives the spatial dimension p of the simplex or hypercube being integrated over, and can take any integer value ≥ 1. Thus, to integrate over a three dimensional cube one would set $ip = 3$.

6. *eps* sets the parameter ϵ appearing in the thinning condition of Eq. (160). For *ithinfun*=1, this gives an absolute error criterion; to achieve a given level of relative error, one needs a rough estimate of the value of the integral as given by *outa* or *outb*, which can be used to readjust *eps* by use of Eq. (167). For nonsingular integrands, the *eps* value when using *ithinfun*=3 should, as a first guess, be taken as the square of the *eps* value that one used for *ithinfun*=1. Note that if *ithinlev* is greater than the total level number, so that thinning is suppressed, the results are independent of the value given to *eps*.

7. In all programs other than the hypercube tiling program, the external function is supplied by the user in the subroutine fcn.for. In the tiling programs, fcn.for is instead the symmetrization program for the external function supplied by the user in the subroutine fcn1.for.

8. *llim* sets the limit to the number of subdivisions in the programs with names not ending in "r" or "m". It can be any integer between 1 and 15, except in the simplex4 programs, where the range is 1 to 14. In practice, the effective upper limit is set by machine memory. Start with a low value of $llim$, and then to improve the accuracy, increase it until you get a diagnostic saying memory has been exceeded; the value of $llim$ one less than this is the maximum value $llim = LMAX$ that does not exceed memory. Since the final level $l = llim$ does not further subdivide, this limit is associated with the number of subregions carried forward from level $l-1$ to the final level. As $llim$ is increased the execution time will increase, and this will also impose an effective upper limit.

9. *llim1* and *llim2* in the programs with names ending in "r" and "m" set the limit to the number of subdivisions in the first and second stages of subdivision, respectively. The maximum value of *llim1* is 14 and of *llim2* is 15, except in the simplex4 programs, where the respective limits are 13 and 14, and also except for recursive subdivision, where the maximum value of *llim2* is one less than the corresponding value for symmetric subdivision. In all cases, the built-in subdivision limits prevent the program from dividing 1 by 2, giving an integer arithmetic answer of 0. As before, the effective upper limit will be set by machine memory and machine execution speed. In using the "r" and "m" programs, and setting $llim2 = 1$, the maximum value of $llim1$ that will not exceed memory is $llim1 = LMAX-1$, with $LMAX$ the corresponding maximum determined as above for the single stage program. Once $LMAX$ is determined, one can take any value $1 \leq llim2 \leq LMAX$ without exceeding memory. Because of the staging, the numerical output depends only on the sum $llim1 + llim2$, that

is, one is free to redistribute the computational effort between the first and second stages.

10. The parameters for the higher order integration routines are contained in the main program files in subprograms with names beginning with "setparam". They are given in array constructors, and have been preset to values indicated. The program variable names have been chosen to roughly correspond to the symbol names in the formulas of Secs. VII and VIII. For example, for the direct cube routines, where λ is a free parameter, it is called *aalamb*; in the order 7 routine for simplex integration, λ_1^i and λ_2^i are the respective elements of the array constructors *alamb*1 (*blamb*1) and *alamb*2 (*blamb*2) corresponding to the first (second) choice of parameter values. (In the fifth order cube and simplex routines, where only one pair of array constructors is needed, they are called *alamb* (*blamb*), even though the corresponding quantity is labeled λ_1^i in Secs. VIIC and IXB.) These presets can be changed by the user to give a different sampling of the integrand in the integration subregions, subject to the following rules: (i) The inequalities in the comment statements must be obeyed, to keep the sampling points inside the subregion, as discussed in Secs. V and VI. (ii) The parameters in each array constructor must have non-degenerate values, so that the corresponding Vandermonde equations will be solvable. If two parameters in an array constructor are very close, solution of the Vandermonde system will have large truncation errors, so care should be taken to keep the parameters in each array constructor reasonably well spaced. (iii) The "a" and "b" array constructors should have different parameter values, since these are used to give the two different integrand evaluations used in the error estimate.

C. Outputs (and their use in making memory and running time estimates)

Program outputs (except in the MPI case) are written to a file "outdat.txt" and also appear on the screen. In the MPI case, outputs are written to the output file specified by the system for a "print" statement. A brief description of output labeling follows:

1. All programs write out the user-set values of *ip*, *llim* (or *llim*1 and *llim*2), *eps*, *ithinlev*, *ithinfun*, *isubdivision*, and *iaccuracy*. They do not print out the values of the parameters in the array constructors in the subprograms setparam.

2. In all programs, *outa* and *outb* give two evaluations of the integral divided by the base region volume, corresponding respectively to the two different samplings of the integrand set by the "a" and "b" parameters in the array constructors, and $|outdiff|$ gives the difference

$|outa - outb|$. The size of $|outdiff|$ gives an estimate of the likely error in the answer; this estimate can be improved by evaluating the integral with a number of different choices of the array constructor parameters, and also by comparing the evaluations obtained using different program options as set by the user-set inputs. As noted in Sec. XI, to get the value of the integral without normalization by the base region volume, one must multiply *outa* and *outb* by the base region volume V_0. For a standard simplex, $V_0 = 1/p!$, for a side 1 hypercube, $V_0 = 1$, while for a half-side 1 hypercube, $V_0 = 2^p$.

3. In all programs, $errsum$ gives the sum of the absolute values of the local subinterval thinning tests,

$$\text{errsum} \equiv \sum_{\text{subregions}} V(\text{subregion}) |f\big(I_a(\text{subregion}, I_b(\text{subregion})\big)| \quad , \tag{174}$$

with $f(A,B)$ the thinning function. As explained above, for the choice $ithinfun = 1$ this gives an upper bound for $|outdiff|$, and when $I_a(\text{subregion} - I_b(\text{subregion})$ has uniform sign over all subregions, $errsum = |outdiff|$. However, when signs are not uniform over subregions, $errsum$ for *ithinfun*=1 can be much larger than the actual error, as in the two Gaussian example discussed below.

4. In all programs, l gives the level number, *ind* gives the number of subregions carried forward to the next level, *indmax* gives the maximum value of *ind* encountered over the course of the various levels that have been executed, $fcncalls$ gives the number of function calls, $t\_current$ gives the current elapsed time in seconds at the various levels of the first stage, and t_final gives the total elapsed execution time in seconds. In the approximation in which the geometric series summing the number of function calls over the various levels is approximated by its largest term, corresponding to the highest level attained, and when there is no thinning, $fcncalls \simeq T2^{p(llim-1)}$ for the single stage program, and $fcncalls \simeq T2^{p(llim1+llim2-1)}$ for the "r" and "m" programs, with T the appropriate function call value from Table VI or VIII. $\big($In the absence of thinning, the exact formula summing the geometric series is $fcncalls = T[2^{p(K+1)} - 1]/[2^p - 1]$, with $K = llim - 1$ for the single stage program and $K = llim1 + llim2 - 1$ for the "r" and "m" programs.$\big)$ When there is thinning, this gives an upper bound on the number of function calls.

5. In the "recirculating" programs with main program name ending in "r", $t\_restart$ gives the time at which the second stage is initiated, in which the subregions carried forward from the

first stage are subdivided sequentially. During the second stage, the program will indicate approximately when it is 10, 20, ..., 90, 100 percent finished in sequentially processing the subregions carried forward from the first stage, by printing this information to the screen (but not by writing it to file). In interactive mode, this permits one to gauge how long the calculation will take to finish; if it looks like the calculation will take longer than one wishes to wait, one can stop execution and restart with different, more tractable, parameter values. Since these numbers are computed by integer division, the actual numbers may be 9,19,... or other similar strings, depending on the residue modulo ten of the number of regions carried forward. One can also estimate the total running time by multiplying $t\_restart$ by the number of subregions ind carried forward to the second stage from the final level of the first stage, further multiplied by $2^{p(llim2-llim1)}$ to correct for a difference in the first and second stage level numbers. (Similarly, for the single stage programs, from $t\_current$ and ind at the output of any level l, the maximum running time, in the absence of thinning, to reach the level limit $llim1$ is the product $t\_current$ times ind, further multiplied by $e^{p(llim1-l)}$.)We generally found that timing values, on a laptop, varied by one or two tenths of a second between identical runs, so estimates of total running time become reliable only when one has proceeded to the point where several seconds have elapsed. The final statistics include $indcount$, which gives a sum of the ind values at each level of the second stage, and which indicates the ind value that would be needed if the second stage subdivisions were carried out in the first stage by using a larger $llim1$ value. Because of the staging strategy, the maximum ind value that is required is the much smaller number $indmax$.

6. In the MPI programs with main program name ending in "m", $t\_restart$ is the time at the end of the first stage when the subregions carried forward, numbering $indstart$ in total, are distributed to multiple processes, and $fcncalls$ gives the number of function calls up to this point. If $t\_restart$ does not appear in the output, the program has completed execution before entering the second stage. Since MPI programs are typically run in batch mode, no intermediate statistics are output during the second stage, but one can make a rough estimate of total second stage running time by multiplying $t\_restart$ by the number of subregions $indstart$ carried forward, further multiplied by $2^{p(llim2-llim1)}$ to correct for a difference in the first and second stage level numbers, and dividing by $N_{\text{process}} - 1$ (process 0 serves only as an accumulation register for the output of the remaining $N_{\text{process}} - 1$ processes). If this estimate is too large, one can stop execution and restart with less ambitious parameters.

The final statistics include *indmaxprocess*, which is the maximum of the final *indmax* over all of the processes.

XIV. SOME SAMPLE RESULTS, AND OPEN QUESTIONS

A. Sample results

We turn now to some sample results which illustrate the capabilities of our numerical integration programs. Our first example is one given in the paper on VEGAS of Lepage (1978), consisting of the sum of two spherically symmetric Gaussians equally spaced along the diagonal of a cubical integration volume,

$$I_p = \frac{1}{2}\left(\frac{1}{a\pi^{1/2}}\right)^p \int_0^1 d^px[e^{-\sum_{i=1}^p (x(i)-1/3)^2/a^2} + e^{-\sum_{i=1}^p (x(i)-2/3)^2/a^2}] \quad , \tag{175}$$

with $a = 0.1$. In this form I_p can be evaluated by the "cubetile" programs which tile a unit hypercube with Kuhn simplexes. In order to apply the direct hypercube "cube" programs, we make the change of variable $x = (1+y)/2$ to rewrite I_p as an integral over a half-side 1 hypercube,

$$I_p = \frac{1}{2}\frac{1}{2^p}\left(\frac{1}{a\pi^{1/2}}\right)^p \int_{-1}^1 d^py[e^{-\sum_{i=1}^p (y(i)+1/3)^2/(4a^2)} + e^{-\sum_{i=1}^p (y(i)-1/3)^2/(4a^2)}] \quad . \tag{176}$$

In his paper Lepage compares numerical evaluations of I_p for various p with a target value of unity, which is accurate enough for his purposes. However, the programs given here are capable of much higher accuracy with current computers, so we will need a high accuracy evaluation of I_p for comparison purposes. We can get this by noting that the two Gaussians contribute equally to I_p $\big($to see this, set $x \to -x$ in Eq. (176)$\big)$, and each individual Gaussian is the pth power of a one dimensional integral J, giving

$$\begin{aligned} I_p =& J^p \quad , \\ J =& \frac{1}{2a\pi^{1/2}} \int_{-1}^1 dy e^{-(y+1/3)^2/(4a^2)} \quad . \end{aligned} \tag{177}$$

The one-dimensional integral J can be evaluated in terms of error functions or complementary error functions,

$$\begin{aligned} J =& \frac{1}{2}[\mathrm{erf}\big(1/(3a)\big) + \mathrm{erf}\big(2/(3a)\big)] \quad , \\ =& 1 - \frac{1}{2}[\mathrm{erfc}\big(1/(3a)\big) + \mathrm{erfc}\big(2/(3a)\big)] \quad , \end{aligned} \tag{178}$$

TABLE X: Evaluation of J to 13 place accuracy using 3rd, 5th, 7th order cube routines

$iaccuracy$	$\lvert outdiff \rvert$	$fcncalls$	t_final
3	10^{-16}	0.2×10^5	$< .02$s
5	10^{-15}	0.5×10^5	$< .02$s
7	10^{-13}	0.9×10^5	$< .02$s

TABLE XI: Evaluation of powers of J to give expected values of I_p to 13 place accuracy

p	$I_p = J^p$
1	0.9999987857663
2	0.9999975715341
3	0.9999963573033
4	0.9999951430740
5	0.9999939288462
6	0.9999927146199
7	0.9999915003951
8	0.9999902861717
9	0.9999890719498

but it can also be evaluated numerically to 13 digit accuracy by running the "cube" program to a depth of twelve total levels. Running the "r" version of the programs with parameter values $ip = 1$, $llim1 = 5$, $llim2 = 7$, $ithinlev = 12$ (no thinning, which makes the results independent of eps and $ithinfun$), and $iaccuracy = 3, 5, 7$, we get from all three runs the result

$$J = 0.9999987857663 \quad . \tag{179}$$

The statistics for running time (on a MacBook Pro) and the number of function calls for these runs are given in Table X. Running with $iaccuracy = 1$ gave only 10 place accuracy with 12 levels, but gave 13 place accuracy when 20 levels were used (which took about a second, rather than hundredths of a second). Thus, for this calculation $iaccuracy = 3$ is the most cost-effective program.

Running a program to raise J to powers then gives the expected results for I_p given in Table XI, with an uncertainty of 1 in the final decimal place.

We give in Table XII results for dimensions $p =$ 2, 3, 4, and 5 as obtained from the "r" version of the programs on a laptop, and in Table XIII for $p = 7, 9$ as obtained by running the "m" version on a 64 process cluster. (Laptop runs were done on a MacBook Pro and an older Dell Inspiron, and for the latter, the timings were rescaled by a factor 0.49 to give timings for a MacBook Pro. We made cluster runs with 128 or 64 processes, and for the former, we rescaled the running time to that for 64 processes. We invite the reader to compare the running times and accuracies summarized in Tables XII and XIII with those that can be obtained from other integration programs.) For all of these runs, where thinning was used, we took $ithinfun = 1$. "Place accuracy" indicates the decimal place where differences first appear from the 13 place result in Table XI. To within an order of magnitude, this agrees with the difference |outdiff| between the two evaluations of the integral given by the program. The values of errsum for these integrals (not shown) were typically one to three orders of magnitude larger than both |outdiff| and the actual error, indicating that the local subinterval errors do not all have the same sign. Since |outdiff| can be smaller than the actual error, for an unknown integral it cannot be taken as giving the error; in this case the best way to estimate the error is to run the program with different choices of program options and to use the spread of results to estimate the error.

From Tables XII and XIII, we see that a minimum of 5 levels is needed to get good accuracy for the double Gaussian example. With 5 levels, the smallest hypercube side is $1/32 = 0.03125$, small enough to resolve the double Gaussian characteristic scale of 0.1 in good detail. On the other hand, with only 4 levels, the minimum side is $1/16 = 0.0625$, making it harder to resolve a scale of 0.1 and limiting the accuracy to 4 significant figures. Most of the cluster runs were done without thinning, and thus should characterize the accuracy attainable for any function on a unit hypercube with a characteristic scale length of 0.1. For serial runs done with thinning (Table XII), the reduction in running time was proportional to the reduction in number of function calls, and ranged from a saving in the range 30% to a factor of 3.5, for values of ϵ which yield the same or one place less accuracy as when there is no thinning. For parallel cluster runs done with thinning (Table XIII), the reduction in running time is considerably less than the reduction in number of function calls. This arises from the fact that even though the program initially distributes subregions to processes using a shuffling routine that assigns adjacent subregions in the stack to different processes, some processes get subregions (like ones near the Gaussian peak) that are "hard" and so take longer to finish, as compared with processes that get "easy", readily thinned regions near the Gaussian tails. Consequently, since the final time t_final records the time when all processes have finished, it is not reduced by thinning in proportion to the number of function calls.

From Tables XII and XIII, we see that thinning with a value of ϵ equal to the error level in the runs with no thinning leads, in the double Gaussian examples, to a reduction in accuracy. This reflects the fact that in the double Gaussian case, errsum is typically 2 to 3 orders of magnitude larger than |outdiff|, indicating that the local errors are not of constant sign, and also errsum is 2 to 3 orders of magnitude larger than ϵ, indicating that the local thinning condition is not satisfied in all subregions. When local subregion errors alternate in sign and the thinning condition is not uniformly obeyed, there can be cancelations of errors in the output integrals, leading to improved accuracy and a smaller |outdiff|, but thinning can then reduce the degree of cancelation and reduce the accuracy. Finally, we note that the 6 level run with *iaccuracy* = 9 did not give as many significant figures as the corresponding run with *iaccuracy* = 7; we believe this is due to the increased truncation errors associated with running the ninth order routine. The cluster which we used was more than an order of magnitude slower in running quadruple precision (real(16)) as opposed to double precision (real(8)) code, so it was not feasible for us to investigate this further by repeating the ninth order 6 level run in quadruple precision. (We did, however, test in quadruple precision that the ninth order routines integrate polynomials of ninth degree or lower to within expected truncation errors.)

We also studied the double Gaussian example using the "cubetile" programs. These integrate over a unit hypercube by integrating, over a single Kuhn simplex, the symmetrized function that sums over corresponding points of a Kuhn tiling of the hypercube. Results for this study are given in Table XIV. Because of the $p\,!$ symmetrization factor in the number of function calls, the "cubetile" programs take longer to run than the "cube" programs for a corresponding number of subdivision levels. Because tiling of a hypercube with Kuhn simplexes does not reduce the subregion side length, the $p\,!$ increase in number of subregions does not compensate for insufficient resolution when the attainable level number is not large enough. This can be seen from the results in Table XIV. For $p = 5$, where 5 levels can be run on the cluster in reasonable time, significantly better results are obtained from a 5 level "cubetile" run than are obtained from a 5 level "cube" run, at the price of a factor of 120 more function calls. However, for $p = 7$ it was not possible to do a 5 level calculation in reasonable cluster running time, so we had to settle for 4 levels, which as we saw above has insufficient resolution to give very high accuracy for the double Gaussian test problem. The "cubetile" results in this case are better than the 4 level "cube" results, reflecting the factor of 5040 more function calls, but the 6 place accuracy achieved is not as good as what can be achieved, in less running time, by using the "cube" program with 5 levels. We conclude that the "cubetile" programs become significantly less useful as the dimension p increases, because of the $p\,!$ symmetrization factor necessitated by Kuhn tiling.

Our next two examples illustrate results obtained from the "simplex" programs to evaluate integrals over a standard simplex. For our first example, we consider an integral based on the

TABLE XII: Double Gaussian results using the "cube" program for $ip \equiv p$ =2 ,3 ,4 ,5 ,7 (timings for MacBook Pro); $levels \equiv llim1 + llim2$ and "place accuracy" compares to Table XI

ip	$iaccuracy$	$levels$	$ithinlev$	eps	t_final	$fcncalls$	$(outa + outb)/2$	$\|outdiff\|$	place accuracy
2	3	10	no thinning	–	1.2s	0.31×10^7	0.9999975715340	0.4×10^{-12}	13
2	5	10	no thinning	–	3.4s	0.94×10^7	0.9999975715339	0.3×10^{-14}	13
2	5	10	2	10^{-13}	2.1s	0.59×10^7	0.9999975715340	10^{-14}	13
2	7	10	no thinning	–	8.5s	0.24×10^8	0.9999975715342	0.5×10^{-12}	13
2	7	10	2	10^{-13}	2.4s	0.68×10^7	0.9999975715342	0.4×10^{-12}	13
3	3	7	no thinning	–	1.7s	0.39×10^7	0.999996358	0.2×10^{-8}	9
3	3	9	2	10^{-13}	72s	0.18×10^9	0.999996357305	0.9×10^{-11}	12
3	5	7	2	10^{-9}	2.8s	0.71×10^7	0.999996356	0.3×10^{-10}	9
3	5	7	2	10^{-13}	4.1s	0.11×10^8	0.99999635730	0.2×10^{-10}	11
3	5	9	2	10^{-13}	190s	0.48×10^9	0.9999963573032	0.3×10^{-13}	13
3	7	7	2	10^{-12}	13s	0.32×10^8	0.9999963573033	0.5×10^{-12}	13
4	3	7	2	10^{-9}	55s	0.12×10^9	0.999995143	0.3×10^{-8}	9
4	5	7	2	10^{-13}	310s	0.71×10^9	0.99999514305	0.4×10^{-10}	11
4	7	5	2	10^{-7}	3.7s	0.83×10^7	0.99999510	0.3×10^{-8}	8
4	7	6	2	10^{-13}	90s	0.20×10^9	0.99999514308	0.2×10^{-10}	11
5	3	5	2	10^{-13}	6.4s	0.14×10^8	0.9999940	0.7×10^{-6}	7
5	3	6	2	10^{-13}	170s	0.37×10^9	0.99999394	0.5×10^{-7}	8
5	5	5	no thinning	–	49s	0.10×10^9	0.99999386	0.2×10^{-6}	7
5	5	5	2	10^{-13}	29s	0.63×10^8	0.99999386	0.2×10^{-6}	7
5	7	5	no thinning	–	240s	0.50×10^9	0.99999393	0.1×10^{-7}	8
5	7	6	2	10^{-9}	1700s	0.34×10^{10}	0.999993926	0.3×10^{-10}	9
7	3	5	no thinning	–	6100s	0.78×10^{10}	0.999992	0.9×10^{-6}	6
7	3	5	2	10^{-9}	840s	0.11×10^{10}	0.999991	0.9×10^{-6}	6
7	5	5	2	10^{-9}	3500s	0.62×10^{10}	0.999991	0.2×10^{-6}	6
7	7	4	no thinning	–	1200s	0.21×10^{10}	0.9997	0.4×10^{-2}	3
7	7	4	2	10^{-9}	340s	0.60×10^9	0.9995	0.4×10^{-2}	3

Feynman-Schwinger formula of Eq. (32), with $D_0 = 1$ and $D_1 = ... = D_p = a$,

$$\frac{1}{a^p} = p!\int_{\text{standard simplex}} \frac{1}{[1 + (a-1)\big(x(1) + ... + x(p)\big)]^{p+1}} \quad . \tag{180}$$

Taking $a = 0.1$ (and for $p = 5$, also $a = 0.01$) gives an integral that is sharply peaked on the diagonal hyperplane $1 = x(1) + ... + x(p)$ bounding the simplex. Results for this integral obtained from the "m" version of the program, with symmetric subdivision and no thinning on a 64 process

TABLE XIII: Double Gaussian results using the "cube" program for $ip \equiv p = 7, 9$ from a 64 process cluster; $levels \equiv llim1 + llim2$ and "place accuracy" compares to Table XI

ip	$iaccuracy$	$levels$	$ithinlev$	eps	t_final	$fcncalls$	$(outa+outb)/2$	$\|outdiff\|$	place accuracy
7	3	5	no thinning	–	11s	0.78×10^{10}	0.999992	0.9×10^{-6}	6
7	5	5	no thinning	–	120s	0.44×10^{11}	0.9999913	0.5×10^{-6}	7
7	7	5	no thinning	–	760s	0.27×10^{12}	0.99999150	0.3×10^{-7}	8
7	1	6	no thinning	–	2900s	0.52×10^{12}	0.999993	0.4×10^{-6}	6
7	3	6	no thinning	–	4300s	0.10×10^{13}	0.9999915	0.8×10^{-7}	7
7	5	6	no thinning	–	17000s	0.56×10^{13}	0.99999150	0.9×10^{-8}	8
7	7	6	no thinning	–	98000s	0.35×10^{14}	0.9999915003	0.1×10^{-9}	10
7	7	6	2	10^{-10}	37000s	0.37×10^{13}	0.99999148	0.1×10^{-9}	8
7	9	5	no thinning	–	4000s	0.14×10^{13}	0.99999144	0.6×10^{-7}	8
7	9	6	no thinning	–	510000s	0.18×10^{15}	0.99999148	0.1×10^{-7}	8
9	3	5	no thinning	–	7500s	0.25×10^{13}	0.999989	0.1×10^{-5}	6
9	5	5	no thinning	–	49000s	0.17×10^{14}	0.9999888	0.8×10^{-6}	7
9	7	5	no thinning	–	380000s	0.13×10^{15}	0.99998907	0.7×10^{-7}	8
9	7	5	2	10^{-8}	100000 s	0.52×10^{13}	0.9999885	0.7×10^{-7}	7

TABLE XIV: Double Gaussian results using the "cubetile" program with symmetric subdivision, for $ip \equiv p = 5, 7$ from a 64 process cluster; $levels \equiv llim1 + llim2$ and "place accuracy" compares to Table XI

ip	$iaccuracy$	$levels$	$ithinlev$	eps	t_final	$fcncalls$	$(outa+outb)/2$	$\|outdiff\|$	place accuracy
5	7	5	no thinning	–	73s	$120 \times 0.27 \times 10^{9}$	0.999993928884	0.7×10^{-11}	12
7	7	4	no thinning	–	12000s	$5040 \times 0.93 \times 10^{9}$	0.999988	0.1×10^{-5}	6

cluster, are given in Table XV. "Place accuracy" indicates the decimal place where differences first appear from the exact answer a^{-p}, and this correlates well with the difference |outdiff| between the two evaluations of the integral given by the program. The values of errsum for these integrals (not shown) were nearly identical to |outdiff|.

As our second simplex example, we consider the polynomial integral (c.f. Eq. (169))

$$\frac{2^{p+1}}{(3p+2)(3p+1)...(p+2)(p+1)} = p\,! \int_{\text{standard simplex}} [1 - x(1) - ... - x(p)]^2 \prod_{i=1}^{p} x(i)^2 \quad , \tag{181}$$

which is strongly suppressed at all the vertices of the simplex. Running a program to evaluate the exact answer for this integral on the left hand side of Eq. (181), and then evaluating the integral on the right on a laptop using the "simplex" programs, gives the results in Table XVI.

TABLE XV: Feynman-Schwinger integral for $ip \equiv p = 5$, 7, 9 from a 64 process cluster, with no thinning; $levels \equiv llim1 + llim2$ and "place accuracy" compares to the exact answer a^{-p}

ip	a	$iaccuracy$	$levels$	t_final	$fcncalls$	$(outa + outb)/2$	$\|outdiff\|$	place accuracy
5	0.1	7	5	0.39s	0.27×10^9	0.99998×10^5	0.3	5
5	0.01	7	9	52000s	0.22×10^{14}	0.999998×10^{10}	0.3×10^4	6
7	0.1	7	5	160s	0.12×10^{12}	0.99995×10^7	0.2×10^3	5
7	0.1	7	6	20000s	0.15×10^{14}	0.9999995×10^7	2.	7
9	0.1	7	5	56000s	0.48×10^{14}	0.9999×10^9	0.3×10^5	4

TABLE XVI: Polynomial integral for $ip \equiv p = 4$, 5 (timings for a MacBook Pro); $levels \equiv llim1 + llim2$ and "place accuracy" compares to the exact answer on the left hand side of Eq. (181)

ip	$iaccuracy$	$levels$	t_final	$fcncalls$	$(outa + outb)/2$	exact answer	$\|outdiff\|$	place accuracy
4	5	6	8.7s	0.63×10^8	$0.88095326 \times 10^{-8}$	$0.8809532619056 \times 10^{-8}$	0.1×10^{-17}	8
4	5	8	2200s	0.16×10^{11}	$0.880953261905 \times 10^{-8}$	$0.8809532619056 \times 10^{-8}$	0.3×10^{-21}	12
5	7	5	39s	0.27×10^9	$0.21591990 \times 10^{-10}$	$0.2159199171337 \times 10^{-10}$	0.2×10^{-18}	8
5	7	6	1300s	0.86×10^{10}	$0.2159199171 \times 10^{-10}$	$0.2159199171337 \times 10^{-10}$	0.7×10^{-21}	10

As our final example, we consider the 1 dimensional singular integral

$$\int_0^1 dx \frac{1}{\sqrt{1-x^2}} = \pi/2 \simeq 1.57079633 \quad . \tag{182}$$

In Table XVII we give results for this integral using the "r" version of the "cube" program, with $p = ip = 1$, $llim1 = 14$ and $llim2 = 15$ (that is, using the maximum allowed number of levels), $iaccuracy = 5$, $ithinlev = 0$ (that is, thinning starts at the outset), and $eps = 10^{-10}$, as a function of the choice of thinning function $ithinfun$. We see that in this case, $ithinfun = 3$ gives the fastest evaluation, with $ithinfun = 2$ next fastest and $ithinfun = 1$ the slowest. This differs from the double Gaussian examples, where $ithinfun = 1$ gives better results than either $ithinfun = 2$ or $ithinfun = 3$.

B. Programming extensions and open questions

There are a number of possible extensions of the programs that could be pursued in the future. (1) The MPI version of the programs could be rewritten to include redistribution of the process workload after each level ℓ of the second stage. This would make the reduction in running time when using thinning track more closely with the reduction in the number of function calls. (2) The multistage strategy could be extended to a third (or more) stages, by not harvesting the subregions

TABLE XVII: Evaluation of a one dimensional singular integral with different thinning functions. Running times were all less than $0.1s$; "place accuracy" compares to the exact answer

$ithinfun$	$fcncalls$	$(outa + outb)/2$	$\|outdiff\|$	place accuracy
1	0.24×10^6	1.570778	0.35×10^{-5}	5
2	0.13×10^5	1.570778	0.35×10^{-5}	5
3	0.45×10^4	1.570777	0.43×10^{-5}	5

that fail to obey the thinning condition at the end of the second stage, but instead writing them to a memory device which is then read sequentially by a third stage, etc. (3) One could build in an option of permuting the simplex vertices at the start of the simplex programs, which gives a different subdivision, and therefore a different evaluation of the integral for use in estimating errors. (4) Finally, we remark that the same subdivision, thinning, and staging strategies that we have used will apply with any integration formulas that give two different estimates of the answer from each subregion, not just the parameterized moment fitting formulas that we developed in Secs. VII and IX.

There are also a number of mathematical questions that we have left open. (1) We found numerical evidence that symmetric subdivision of a standard simplex obeys the bound of Eq. (46) for reduction of side length, and that after ℓ symmetric (recursive) subdivisions, the resulting subsimplexes each fit within a hypercube of side $1/2^\ell$ $(1/2^{\ell-1})$. We do not have a proof of these conjectures, but have assumed them true in constructing the programs. (2) Given the regularities in the construction of parameterized fifth, seventh, and ninth order integration formulas for the simplex and hypercube cases, it would be of interest to try to find a general all-orders rule for these. (3) We have not addressed the question of analytic error estimates for the parameterized integration formulas. (4) We have not addressed in any systematic way the question of deciding which thinning function is optimal for a given choice of integrand. (5) It would be of interest to study the systematics, in the moment fitting method, of the tradeoff between the number of parameters that are fixed by appropriate conditions, and the number of function calls.

XV. ACKNOWLEDGEMENTS

I wish to thank the School of Natural Sciences computing staff, Prentice Bisbal, Kathleen Cooper, Christopher McCafferty, and James Stephens, for their helpful support and advice throughout this project. I also want to acknowledge several helpful conversations with Prentice Bisbal about

Fortran language features, and to thank Susan Higgins for drawing the figures. I am grateful to Herman D. Knoble for permission to use in the hypercube tiling routines his program BestLex (which is in the public domain and is not covered by the copyright for this book). This work was partially supported by the Department of Energy under grant DE-FG02-90ER40542. I also wish to acknowledge the hospitality of the Aspen Center for Physics during the summers of 2009 and 2010.

XVI. REFERENCES

Note: These references make no pretense to completeness; they list what I have found useful in constructing the algorithms discussed in this book. For research groups with a continuing program, I have listed only recent publications that contain earlier references.

Cools, R. and Haegemans, A. (2003) Algorithm 824: CUBPACK: A Package for Automatic Cubature; Framework Description. *ACM Trans. Math. Software* **29**, 287-296.

Dejnakarintra, M. and Banjerdpongchai (undated), D. An Algorithm for Computing the Analytical Inverse of the Vandermonde Matrix. Searchable on-line.

Edelsbrunner, H. and Grayson, D. R. (2000). Edgewise Subdivision of a Simplex. *Discrete Comput. Geom.* **24**, 707-719.

Genz, A. and Cools, R. (2003) An Adaptive Numerical Cubature Algorithm for Simplices *ACM Trans. Math. Software* **29**, 297-308.

Good, I. J. and Gaskins, R. A. (1969) Centroid Method of Integration. *Nature* **222**, 697-698.

Good, I. J. and Gaskins, R. A. (1971) The Centroid Method of Numerical Integration. *Numer. Math.* **16**, 343-359.

Grundmann, A. and Möller, N. M. (1978) Invariant Integration Formulas for the n-Simplex by Combinatorial Methods. *Siam J. Numer. Anal.* **15**, 282-290.

Hahn, T. (2005) CUBA- a library for multidimensional numerical integration. arXiv:hep-ph/0404043.

Heinen, J. A. and Niederjohn, R. J. (1997) Comments on "Inversion of the VanderMonde Matrix". *IEEE Signal Process. Lett.* **4**, 115.

Kahaner, D. K. and Wells (1979), M. B. An Experimental Algorithm for N-Dimensional Adaptive Quadrature. *ACM Trans. Math. Software* **5**, 86-96.

Knoble, H. D. (1995), website download of BestLex subroutine.

Krommer, A. R. and Ueberhuber, C. W. (1994). *Numerical Integration on Advanced Computer Systems*, Lecture Notes in Computer Science 848. (Berlin: Springer-Verlag), p183.

Kuhn, H. W. (1960). Some combinatorial Lemmas in Topology. *IBM J. Res. and Dev.* **4**, 518-524.

Lepage, G. P. (1978). A New Algorithm for Adaptive Multidimensional Integration. *J. Comp. Phys.* **27**, 192-203.

Lyness, J. N. (1965) Symmetric Integration Rules for Hypercubes II. Rule Projection and Rule Extension. *Math. Comp.* **19**, 394-407.

McKeeman, W. M. (1962). Algorithm 145: Adaptive numerical integration by Simpson's rule. *Commun. ACM* **5**, 604.

McNamee, J. and Stenger, F. (1967). Construction of Fully Symmetric Numerical Integration Formulas. *Num. Math.* **10**, 327-344.

Moore, D. (1992) Subdividing Simplices, in *Graphics Gems III*, D. Kirk ed. (Boston: Academic Press), pp. 244-249 and pp. 534-535. See also Moore, D. (1992), Simplicial Mesh Generation with Applications, Cornell University dissertation.

Neagoe, V.-E. Inversion of the Van der Monde Matrix. *IEEE Signal Process. Lett.* **3**, 119-120.

Osborne, M. R. (2001) *Simplicial Algorithms for Minimizing Polyhedral Functions* (Cambridge: Cambridge U. Press), p. 5.

Plaza, A. (2007). The eight-tetrahedra longest-edge partition and Kuhn triangulations. *Comp. & Math. with Applications* **54**, 426-433, Fig. 1.

Pontryagin, L. S. (1952) *Foundations of Combinatorial Topology*, English translation. (Rochester: Graylock Press), pp. 10-12.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992) *Numerical Recipes in Fortran*, pp. 82-85.

Schürer, R. (2008) HIntLib Manual, available on-line at: mint.sbg.ac.at/HIntLib/manual.pdf .

Stroud, A. H. (1971) *Approximate Calculation of Multiple Integrals* (Englewood Cliffs: Prentice-Hall).

Ueberhuber, C. W. (1995) *Numerical Computation 2: Methods, Software, Analysis* (Berlin: Springer-Verlag), pp. 161-162.

Wikipedia articles on: *Adaptive Simpson's method*, *Barycentric coordinates*, *Dirichlet distribution*, *Gaussian quadrature*, *Hypercube*, *Polynomial long division*, *Simplex.*

XVII. CONTENTS OF PROGRAMS IN DIRECTORIES

Each file listed in this summary contains multiple programs, each of which begins with comment lines describing its function.

- The simplex programs take as base region the standard simplex of Eq. (30). (When used as part of the cubetile programs, the base region is the Kuhn simplex of Eq. (37).)
- The cubetile programs take as base region the side 1 hypercube of Eq. (39).
- The cube programs take as base region the half-side 1 (i.e., side 2) hypercube of Eq. (47).
- The numbers after simplex or cube indicate the integration orders that are included.

A. Directory simplex123

This directory contains:

- The subprogram file simplexsubs123.for.
- Main program files simplexmain123.for, cubetilemain123.for.
- Recirculating main program files simplexmain123r.for, cubetilemain123r.for.
- MPI parallel main program files simplexmain123m.for, cubetilemain123m.for.

B. Directory simplex4

This directory contains:

- The subprogram file simplexsubs4.for.
- Main program files simplexmain4.for, cubetilemain4.for.
- Recirculating main program files simplexmain4r.for, cubetilemain4r.for.
- MPI parallel main program files simplexmain4m.for, cubetilemain4m.for.

C. Directory simplex579

This directory contains:

- The subprogram file simplexsubs579.for.
- Main program files simplexmain579.for, cubetilemain579.for.
- Recirculating main program files simplexmain579r.for, cubetilemain579r.for.
- MPI parallel main program files simplexmain579m.for, cubetilemain579m.for.

D. Directory simplex579_16

This directory contains:

- The subprogram file simplexsubs579_16.for.
- Main program files simplexmain579_16.for, cubetilemain579_16.for.
- Recirculating main program files simplexmain579_16r.for, cubetilemain579_16r.for.
- MPI parallel main program files simplexmain579_16m.for, cubetilemain579_16m.for.

E. Directory cube13

This directory contains:

- The subprogram file cubesubs13.for.
- Main program file cubemain13.for.
- Recirculating main program file cubemain13r.for.
- MPI parallel main program file cubemain13m.for.

F. Directory cube579

This directory contains:

- The subprogram file cubesubs579.for.

- Main program file cubemain579.for.
- Recirculating main program file cubemain579r.for.
- MPI parallel main program file cubemain579m.for.

G. Directory cube579_16

This directory contains:

- The subprogram file cubesubs579_16.for.
- Main program file cubemain579_16.for.
- Recirculating main program file cubemain579_16r.for.
- MPI parallel main program file cubemain579_16m.for.