Multidimensional Adaptive Quadrature Over Simplices

Kevin Ray Pond

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY
in
Mathematics

Jeff Borggaard, Chair
John Burns
Eugene Cliff
Terry Herdman
Lizette Zietsman

6 August 2010
Blacksburg, Virginia

Key words: multidimensional, adaptive, quadrature, simplices

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

Abstract

The objective of this work is the development of novel, efficient and reliable multidimensional adaptive quadrature routines defined over simplices (MAQS). MAQS provides an approximation to the integral of a function defined over the unit hypercube and provides an error estimate that is used to drive a global subdivision strategy. The quadrature estimate is based on Lagrangian interpolation defined by using the vertices, edge nodes and interior points of a given simplex. The subdivision of a given simplex is chosen to allow for the reuse of points (thus function evaluations at those points) in successive refinements of the initial tessellation. While theory is developed for smooth functions, this algorithm is well suited for functions with discontinuities in dimensions three through six. Other advantages of this approach include straight-forward parallel implementation and application to integrals over polyhedral domains.

To Mom and Dad

Thank you for being you and encouraging me to be me.

Acknowledgments

This dissertation is a result of what I have learned while attending Virginia Polytechnic Institute and State University. However, the dissertation itself would not exist without the lifetime of love and support I have received from my family, friends and mentors over the years. Here, I would like to express my gratitude to these individuals who helped me achieve this milestone in my life.

I wish to thank my committee members. You are a dedicated team of professors who foster an environment of both professionalism and camaraderie. I consider myself lucky to have worked under you and hope to do so again in the future.

From this group of established professionals, I must especially thank my advisor, Professor Jeff Borggaard. I realized from the courses I had taken under you, that your style and composure were a few traits among many that I wanted to model my own after. I definitely enjoyed your classes and our meetings over the past three years. Your mixture of being "hands off" yet directive at the same time allowed me to refine my skills as a mathematician without wasting valuable time spinning my wheels. I still have a lot to learn but if I ever learn to be as confident, dedicated, and approachable as you I will consider that as great an achievement as this degree.

While working at ICAM I was lucky enough to befriend several individuals: Brian McBee, Carlos Rautenberg, Hans-Werner van Wyk, Vitor Nunes, Weiwei Hu, and Zhu Wang. I enjoyed your willingness to share ideas and how you gently reminded me that I was not so infrequently wrong. You are a great group of people, all driven to succeed and destined to. I want to specifically thank Dr. John Burkardt. You were a constant

source of help and more importantly friendship. I wish all of you the best in your endeavors and also look forward to working together in the future.

To all my friends from Virginia Tech, I owe so much of this achievement to you. If I would not have met you, this success of mine very well might not have happened. I must specifically acknowledge my newfound brothers and sister: Thierry Platini, Sven Dorosz and Franziska Hinkelmann. You three were fundamental in my success as a student and growth as a person. I absolutely could not have made it to this point without you. No matter where your success takes you, you will always have a place to stay here in the U.S..

Finally, I want to thank my family. All of my achievements stem from your support and love. Specifically, this work is dedicated to you Mom and Dad. You created a wonderful environment that allowed your children to pursue their own way. I can not thank you enough for being you, but I will keep trying by doing my best to be me.

Thank you all so very much,
Kevin

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The mission of this dissertation is to approximate the integration of functions $f \in \mathcal{F}$, a class of functions defined over the d-dimensional hypercube, $\Omega := [0,1]^d$, that are well approximated by piecewise polynomial interpolants. Thus, we consider approximation of the integral operator,

$$I : \mathcal{F} \to \mathbb{R}, \quad \text{where} \quad If := \int_\Omega f(\mathbf{x}) d\mathbf{x},$$

for functions in this class. By approximating f by piecewise polynomial interpolants, we can readily compute an approximation to the integral. This numerical integration can be interpreted as a linear combination of function evaluations and scalar weights, known in numerical analysis as quadrature.

One interesting set of functions in $\mathcal{F}$ are those exhibiting jump discontinuities. Many of these functions can be well-approximated when polynomial segments that represent the portion of the function near the discontinuity have small support. In many cases, the location of the discontinuity is not known *a priori*. Thus, approximation algorithms need to be developed with the ability to automatically detect these features.

As a motivating example, consider the one-dimensional duct flow problem used in the optimization study of Frank and Shubin [16]. In their study, the authors were concerned with steady, inviscid flows in ducts with varying cross-sectional areas. These

flows can be modeled using a special form of the Euler equations,

$$\mathcal{H}_x + \mathcal{G} = 0, \text{ with } x \in [0,1],$$

where

$$\mathcal{H} = \begin{pmatrix} \rho u A \\ (\rho u^2 + p)A \\ (\rho E + p)uA \end{pmatrix}, \qquad \text{and} \qquad \mathcal{G} = \begin{pmatrix} 0 \\ -pA_x \\ 0 \end{pmatrix}.$$

For this system, x is the distance along the duct, ρ is the density, u is the velocity, $E = e + u^2/2$, where e is specific internal energy, and p is the pressure. For a given set of boundary conditions and cross-sectional area, A, the solution, (ρ, u, p), exhibits a jump discontinuity [16]. The varying cross-sectional area is described by the value of the function $A(x, \zeta)$, where $x \in [0,1]$ and each component $\zeta_i \in [\zeta_i^{\min}, \zeta_i^{\max}]$. The parameter ζ describes variations in the cross-sectional area. Even though the above model is well understood, asking the natural question to find the average velocity over a range of area parameters leads to a computationally challenging problem. The resultant integrand is not known *a priori* in $\Omega = [0,1] \times [\zeta^{\min}, \zeta^{\max}]$ and hence we must solve the Euler equations for each point (x, ζ), which is costly in terms of computation time. However, the real complication comes from fact that the resultant integrand is discontinuous.

From this example we see that efficient, accurate methods for approximating multidimensional integrals are necessary. Luckily, the concept of integral approximation has been around since Archimedes attempted to compute π. Therefore, there exists a deep and broad wealth of knowledge. However, as developed as this field is, there are still limitations on the classes of problems where we can find accurate approximations. In the survey [11], the authors categorize approximation of integrals into three main subdivisions based on the dimension of the problem[1].

- Range I: integrals over dimension 3 to about 6 or 7,
- Range II: integrals over dimensions 7 or 8 to about 15,

[1]For one and two dimensions, the situation is considered "satisfactory."

- Range III: integrals over dimensions greater than 15.

In Range I, the accuracy provided by adaptive quadrature methods can overcome the influence of dimension. Assuming that the integral has no particular difficulties such as, singularities, highly oscillatory integrands or discontinuities, 5–6 digits of accuracy are possible.

Range II is considered the "borderline range." Even when using adaptive quadrature methods, the smoothness of the integrand plays a major role in the accuracy of the approximation. However, in this range, adaptive Monte Carlo and quasi-Monte Carlo methods become competitive. Either way, only 3–4 digits of accuracy can be expected.

Range III is considered to have "really high dimensionally." Even sophisticated variance reduction methods exhibit a strong dependence of complexity on dimension, causing Monte Carlo and quasi-Monte Carlo methods to be the best choice. However, one can not hope for more than two digits of accuracy using current methods.

This dissertation presents a multidimensional adaptive quadrature routine evaluated over simplices (MAQS). This approach increases the class of functions covered in Range I by improving quadrature for functions with discontinuities.

The outline of this work is as follows. In Chapter 2 we formulate the mathematical problem and introduce the main notation used throughout. We also describe quadrature methods over hypercubes and simplices, including sparse grids, interpolatory quadrature, and various sampling methods. In Chapter 3 we formulate and justify the tools used by the algorithm presented in Chapter 4. In Chapter 4 we provide a detailed description of the inner workings of our Multidimensional Adaptive Quadrature routine over Simplices (MAQS). In Chapter 5 we present results for several experiments over two different test suites of functions. Here our main goal is to validate the theory developed in Chapter 4 and demonstrate both the strengths and weaknesses of MAQS. In Chapter 6 we summarize the significance of our work and offer concluding remarks as well as immediate and future directions we intend to pursue.

Chapter 2

Quadrature Review

2.1 Introduction

Multidimensional integrals appear in several fields, for example: statistical mechanics, the evaluation of financial derivatives, the discretization of partial differential equations with random inputs, integral equations and the numerical computation of path integrals, to name a few. However, the effectiveness of state of the art methods that approximate these integrals are limited by the smoothness of the integrand and the "curse of dimensionality." In other words, problems arise when dealing with functions that have discontinuities and even for smooth functions the computing cost grows exponentially with the increasing number of dimensions in the problem. The goal of this dissertation is to present a method that effectively approximates the integral of piecewise continuous functions in two to seven dimensions (Range I in the introduction). With this goal in mind, the following sections review the conventional methods for approximating multivariate integrals. Starting with a basic one-dimensional overview and ending with the current methods for error estimation in adaptive quadrature methods, the idea is to express the cusp of the field along with its limitations.

We first discuss the sparse grid method in Section 2.2. This method takes well chosen points and weights from the tensor product of one-dimensional quadrature rules and

constructs a new quadrature rule. It is designed for integration over a d-dimensional rectangle.

Since the algorithm presented in this dissertation is based on adaptively refining a tessellation of the domain, we present existing methods for integration over a d-dimensional triangle or simplex in Section 2.3.

Section 2.4 covers other well-known methods for numerical integration of multidimensional functions, such as Monte Carlo and Quasi-Monte Carlo methods [57]. These methods come into play when the dimension of the integral gets high enough for even the craftiest adaptive algorithm to suffer from the curse of dimensionality. This section covers a brief construction of some of these alternatives and describes when they are best utilized.

After this brief survey of existing quadrature methods, we discuss the concept of adaptive quadrature. We begin with the most basic algorithm, then describe a few specific variations to give the reader a basic background for understanding the algorithm presented in this dissertation. This is followed by error estimation and convergence criteria, which are essentially the most important part of any adaptive quadrature scheme.

Each section closes with statements highlighting both the advantages of a given method and its limitations. The chapter itself closes with a comparison of the limitations of various approaches. In [17] it is shown that these algorithms suffer in practice when the integrand has discontinuities. It is precisely this area or gap that MAQS proposes to fill as seen in Chapter 5.

2.2 Sparse Grids

2.2.1 Introduction

The objective of this section is to review quadrature methods based on the sparse grid approach. We cover construction of these grids and give examples of the resultant

graphs in two-dimensions. We are specifically interested in the rule based on the Gauss-Patterson formulas which have the highest possible polynomial exactness among all nested quadrature formulas using the same number of function evaluations [23]. Since this rule reuses previous function evaluations at each step, the Gauss-Patterson formulation is a good method to compare with MAQS.

The benefit of these constructs is that for spaces where functions have bounded mixed derivatives, Smolyak's construction [72] effectively combats the curse of dimensionality. This method constructs what are called sparse grids by using multidimensional quadrature formulas based on tensor products of one-dimensional formulas. This process keeps the number of function evaluations and the numerical accuracy independent of the dimension of the problem, up to a logarithmic factor [23].

2.2.2 Applications

The Smolyak construction has been applied to numerical integration by several authors, using the midpoint rule [1], the trapezoidal rule [6], the Clenshaw-Curtis rule [59, 60] and Gauss rules [61] as the base one-dimensional integration rule. Further studies exist concerning extrapolation methods [6], discrepancy measures [15] and complexity questions [79].

Outside of numerical integration, applications of Smolyak's construction can be found for estimating eigenvalues of the Schrödinger equation in six-dimensions [26], fluid dynamics [27, 30], numerical integration in finance and insurance [34], parabolic equations for option pricing [64], the solution of elliptic and hyperbolic partial differential equations [25, 31], wavelet-based sparse grid for parabolic problems [78], integral equations [28, 63], stochastic differential equations [68], interpolation and approximation [1, 76], Fokker-Planck equations [14], global optimization [58], and data compression [22], to name a few.

2.2.3 Problem Formulation

We consider numerical approximations to integrals of functions f from a function class $\mathcal{F}$ over the d-dimensional hypercube $\Omega := [0,1]^d$. In other words, we seek approximations to

$$I : \mathcal{F} \to \mathbb{R}, \quad \text{where} \quad If := \int_\Omega f(\mathbf{x}) d\mathbf{x},$$

by a sequence of n_l^d-point quadrature formulas with level $l \in \mathbb{N}$ and $n_l^d < n_{l+1}^d$. Thus, we define the quadrature rule of level l for approximation to integrals of functions defined in dimension d by

$$If \approx Q_l^d f := \sum_{i=1}^{n_l^d} w_{li} f(\mathbf{x}_{li}),$$

using the weights w_{li} and abscissas $\mathbf{x}_{li}$. The *grid* of the resulting quadrature formula is defined by

$$\Gamma_l^d := \{\mathbf{x}_{li} : 1 \leq i \leq n_l^d\} \subset \Omega.$$

This new notation allows us to build up high-dimension integration rules using low-dimensional integration rules, specifically those rules that satisfy

$$\Gamma_l^d \subset \Gamma_{l+1}^d.$$

Quadrature formulas are said to be nested if their corresponding grids are nested. This means that if the rule is nested for the one-dimensional case, then the sparse grid based on that rule is also nested.

Define the error of a given quadrature rule by

$$E_l^d f := |If - Q_l^d f|.$$

We discuss error bounds on the above error functional after we describe the construction of the Smolyak method below.

2.2.4 Smolyak Construction

In 1963, a Russian mathematician, Smolyak [72], described a way of constructing multivariate quadrature formulas on the basis of one-dimensional quadrature formulas in a reduced tensor product setting. The analysis of these formulas assumes that functions have bounded mixed derivatives of a fixed order r. In other words,

$$\mathcal{W}_d^r := \left\{ f : \Omega \to \mathbb{R}, \quad \left\| \frac{\partial^{|\alpha|} f}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}} \right\|_\infty < \infty, \ \alpha \in \mathbb{N}^d, \ \alpha_i \leq r \right\},$$

with $|\alpha| := \sum_{i=1}^d \alpha_i$.

The Algorithm

To construct this rule, take a sequence of one-dimensional quadrature formulas for a univariate function f

$$Q_l^1 f := \sum_{i=1}^{n_l^1} w_{li} f(\mathbf{x}_{li}).$$

Now, define the difference quadrature formula by

$$\Delta_\alpha^1 f := (Q_\alpha^1 - Q_{\alpha-1}^1) f, \quad \text{with} \quad Q_0^1 f := 0.$$

These difference formulas are, in general, quadrature formulas on the union of the grids $\Gamma_k^1 \cup \Gamma_{k-1}^1$. In the nested case, this is just Γ_k^1. Smolyak's construction for d-dimensional functions f for $l \in \mathbb{N}$ and $\mathbf{k} \in \mathbb{N}^d$ is defined by

$$Q_l^d f := \sum_{|\alpha| \leq l+d-1} (\Delta_{\alpha_1}^1 \otimes \cdots \otimes \Delta_{\alpha_d}^1) f.$$

For the tensor product of d quadrature formulas $(Q_{l_1}^1 \otimes \cdots \otimes Q_{l_d}^1)$, we simply sum over all possible combinations

$$(Q_{l_1}^1 \otimes \cdots \otimes Q_{l_d}^1) f := \sum_{i_1=1}^{n_{l_1}^1} \cdots \sum_{i_d=1}^{n_{l_d}^1} w_{l_1 i_1} \cdots w_{l_d i_d} f(x_{l_1 i_1}, \cdots, x_{l_d i_d}).$$

Here is the above expression using the difference formula notation.

$$(Q^1_{l_1} \otimes \cdots \otimes Q^1_{l_d})f := \sum_{j=1}^{d} \sum_{1 \le \alpha_j \le l} (\Delta^1_{\alpha_1} \otimes \cdots \otimes \Delta^1_{\alpha_d})f.$$

Alternatively, Smolyak's formula can be written in terms of $Q^1_{\alpha_j}$ instead of the difference operator $\Delta^1_{\alpha_j}$ [12]

$$Q^d_l f = \sum_{l \le |\alpha| \le l+d-1} (-1)^{l+d-|\alpha|-1} \binom{d-1}{|\alpha|-l} (Q^1_{l_1} \otimes \cdots \otimes Q^1_{l_d})f.$$

This is known as the combination technique [29]. Dimension recursive versions of this formula exist and are often used in proofs [62, 77]

$$Q^d_l f = \sum_{\alpha=1}^{l-1} (\Delta^l_\alpha \otimes Q^{d-1}_{l-\alpha})f,$$

and

$$Q^{d+1}_{l+1} f = \sum_{|\alpha| \le l+N-1} (\Delta^1_{\alpha_1} \otimes \cdots \otimes \Delta^1_{\alpha_d} \otimes Q_{l+d-|\alpha|})f.$$

If desired, the weights can be computed independently using the formulas presented in [23, 59].

Sparse Grids

Figure 2.1 shows the Clenshaw-Curtis, Chebyshev and Gauss-Patterson sparse grids of increasing order of polynomial exactness. Figure 2.2 shows the comparison of the full tensor product with that of the Smolyak construction using the same one-dimensional quadrature rule, in this case Clenshaw-Curtis. This figure shows the dramatic reduction in points required for sparse grids.

To make the usefulness more apparent, Figure 2.3 shows the growth of points for a fixed rule with increasing dimension, comparing the full tensor product with that of the Smolyak construction using the same one-dimensional quadrature rule (Clenshaw-Curtis, nine point rule).

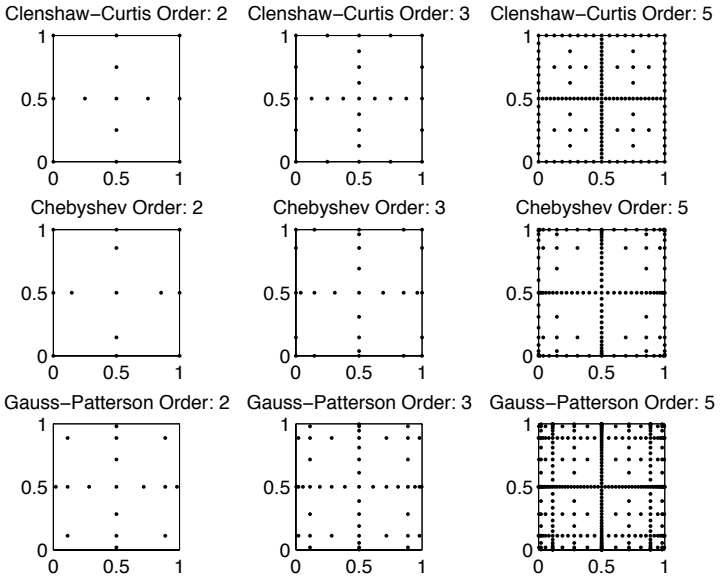


Figure 2.1: Clenshaw-Curtis, Chebyshev and Gauss-Patterson sparse grids of increasing degree

Error Bounds

The power of the sparse grid method is its ability to use a relatively small number of function evaluations while keeping the accuracy of the computed integral high. To understand the error bounds on these quadrature schemes, we must first describe the polynomial degree of exactness for a Smolyak quadrature formula. Let $\mathcal{P}_l^1$ be the space of one-dimensional polynomials of degree $\leq l$. In the multivariate case we consider the polynomial space

$$\mathcal{P}_l^d := \{\mathcal{P}_{\alpha_1}^1 \otimes \cdots \otimes \mathcal{P}_{\alpha_d}^1, |\alpha| = l + d - 1\}.$$

This is the space of polynomials spanned by multivariate monomials of order l or less. If Q_l^1 is exact for $\mathcal{P}_l^1$, then Q_l^d is exact for $\mathcal{P}_l^d$ [58], in other words,

$$E_l^d f = 0, \text{ for all } f \in \mathcal{P}_l^d.$$

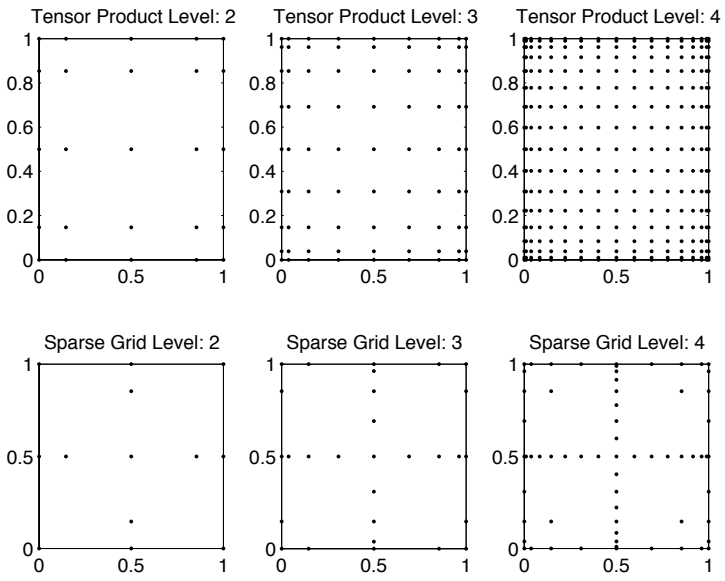


Figure 2.2: Comparison of Tensor Products Grids and Sparse Grids

In order to formulate error bounds for Smolyak's formula, we start with the error bound for the one-dimensional quadrature formulas for functions $f \in C^r(\Omega)$,

$$|E_l^1 f| = O((n_l^1)^{-r}).$$

This bound holds, for example, for all interpolatory quadrature formulas with positive weights, such as the Clenshaw-Curtis and Gauss-Patterson formulas. Taking one such quadrature formula as a one-dimensional basis, if $f \in W_d^r$ and $n_l^1 = O(2^l)$, the error of Smolyak's quadrature formula is of order

$$|E_l^d f| = O(2^{-lr} l^{(d-1)(r+1)}).$$

For the classical spaces $C^r(\Omega)$ error bounds can be derived [59], but they indicate an exponential dependence on the dimension.

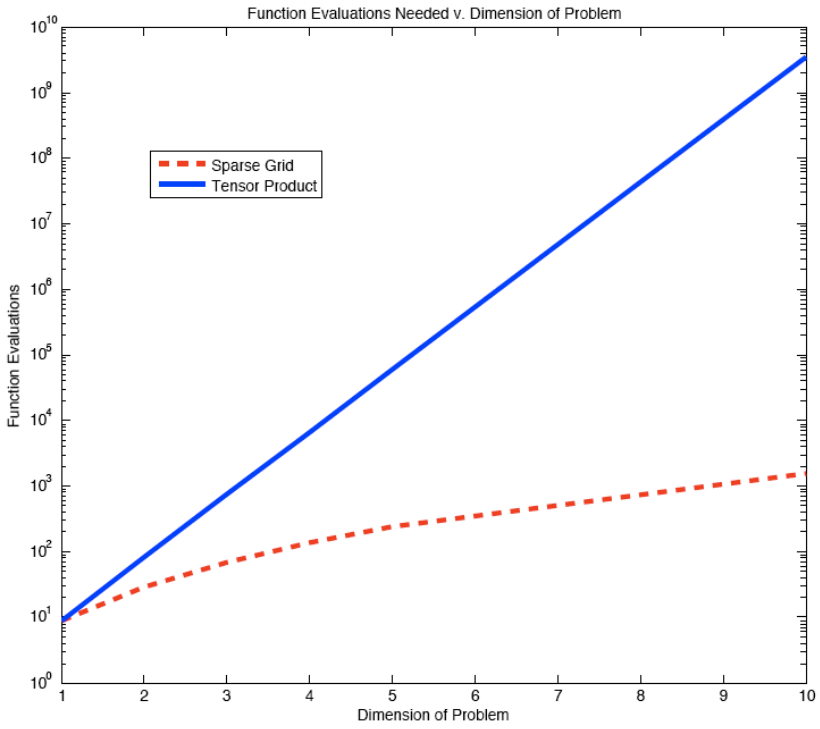


Figure 2.3: Savings in Function Evaluations Using Sparse Grids Compared to Tensor Product Grids as Problem Dimension Increases

2.2.5 Adaptivity

The idea behind adaptive quadrature methods is to dynamically decide which areas of the domain need more attention to sufficiently approximate the integral. These concepts are discussed in depth in Section 2.5 but here we emphasize that when dealing with multidimensions the complexity of the problem dramatically increases. This does not necessarily mean that the function is not smooth. In [24], it was shown that sparse grids can be refined using so-called non-isotropic (anisotropic) formulas. In other words, one can modify the original Smolyak algorithm and define a non-isotropic method of the form

$$Q_l^d f := \sum_{|\alpha| \leq l+d-1} (\Delta_{\alpha_1}^1 \otimes \cdots \otimes \Delta_{\alpha_d}^1) f,$$

with

$$|\alpha|_{\mathbf{v}} := \sum j = 1^d v_j \alpha_j, \qquad v_j > 0.$$

This corresponds to a weighting of the dimension. Thus small values for v_j can be used in smoother dimensions.

This construction is a special case of a wider class of quadrature formulas which can be generated by the following generalization of Smolyak's construction. We denote the jth unit vector as $\mathbf{e}_j$ and define index sets I_l such that for all $\alpha \in I$

$$\alpha - \mathbf{e}_j \in I_l \quad \text{for} \quad 1 \leq j \leq d, \qquad \alpha_j > 1,$$

holds. This way, a general Smolyak quadrature formula is defined by

$$Q_l^d f := \sum_{\alpha \in I_l} (\Delta_{\alpha_1}^1 \otimes \cdots \otimes \Delta_{\alpha_N}^1) f.$$

An appropriate choice of I_l can reduce the computing cost significantly for special integrands, especially in high-dimensional problems. The index set I_l can be obtained *a priori* or by a self-adaptive procedure [21, 76]. However, self-adaptive strategies for high-dimensional integrals are costly and involved.

2.2.6 Conclusions

This section covered the construction of sparse grids and briefly described how the formulas presented may be modified to account for varying degrees of smoothness in different dimensions. Error bounds were presented for a class of functions which have bounded mixed derivatives. This method, however, still suffers from the curse of dimensionality as the number of dimensions increases.

2.3 Quadrature Rules over Simplices

2.3.1 Introduction

Initially, quadrature was treated as an iterative application of one-dimensional quadrature (e.g. the tensor product method we have already seen). Much like the sparse grid method over hypercubes, other geometries have been investigated. These include triangles, circles, hexagons, octahedron, cubical and spherical shells, parabolic regions, cones, tori, pyramids and their d-dimensional extensions where applicable [74]. However, each of these different geometries demand special attention, and the cost of experimentation inhibited their theoretical development.

The theory of integration over simplices is less developed than the theory of the hypercube, however, due to the ability of simplices to approximate a wider range of domains by tessellation as well as their use in Finite Element Methods, research in integration over simplices is quite active. In this section, we will look at existing quadrature methods for cases where the domain is a simplex. The first method described is the Duffy transformation [13]. This reduces the problem to a quadrature rule over a hypercube. We then look at quadrature rules specifically designed for the simplex domain [32]. Special attention is paid to the construction of such rules and to the number of function evaluations required to obtain polynomial degree r. Next, we look at the theory of quadrature using extrapolation [51]. This method is marginally less cost-effective (i.e. uses more function evaluations), but this approach has significant advantages in terms of generality and convenience over the corresponding cubature rules when dealing with algebraic or logarithmic singularities on the edge or at a vertex of the simplex in question.

Remark: Since any nonsingular affine transformation takes one simplex into another and transforms the quadrature rule in the same manner, retaining the degree of polynomial exactness, the results are independent of the actual simplex used for the calculation.

2.3.2 Duffy Transformation

A natural approach to integration over a simplex is to transform the simplex in to an hypercube. Here is a two-dimensional example:

$$\int_0^1 \left[\int_0^x f(x,y)dy \right] dx = \int_0^1 x \int_0^1 f(x,xt)dtdx.$$

This is known as the Duffy transformation [13].

After applying this transformation, one could go on to use either a tensor product of a one-dimensional rule or a sparse grid rule. The drawback to the tensor product rule has already been presented. However, even using the sparse grid rule is inefficient in this case. For example, in the two-dimensional case one uses the same amount of points to integrate a polynomial of degree r on both a square and a triangle, when the triangle has half the area of the square. Thus, even with the ease of implementation, the Duffy transformation is not ideal as it adds substantially more function evaluations.

2.3.3 Cubature Rules Over Simplices

As with cubature (i.e., quadrature in dimensions three and higher) formulas over hypercubes the theory of numerical integration over a simplex is related to polynomial approximation. Points and weights are conventionally chosen so that the formula is exact for all polynomials up to a certain degree r.

Here we provide a formula for numerical evaluation of integrals over the d-dimensional simplex presented in [32]. This formula is used later in the Chapter 4.

The Formula for Cubic Polynomials

Let the vertices of the simplex, Δ be $\{v_0, v_1, \cdots, v_d\}$ with the centroid given by

$$c \ = \sum_{i=0}^{e} v_i/(e+1).$$

Let vol(Δ) be the hyper-volume of Δ.

Theorem 1. *An exact integration formula for the general cubic polynomial over S for $d \geq 1$ is given by*

$$\int_{\Delta} f dV = a_d \sum_{i=0}^{d} f(x_i) + b_d f(c)$$

where

$$a_e = \frac{(d+3)^2}{4(d+1)(d+2)}\text{vol}(\Delta), \qquad b_d = \frac{-(d+1)^2}{4(d+2)}\text{vol}(\Delta)$$

and

$$x_i = \frac{2}{d+3}v_i + \frac{d+1}{d+3}c, \quad \textit{for} \quad i = 0, \ldots, d.$$

Remark: The formula for the cubic polynomial may well be used as an exact integration formula for any polynomial of degree at most three. Also, with this formulation the general cubic polynomial has $(d+1)(d+2)(d+3)/6$ terms.

For a complete list of all the cubature formulas for the simplices see [9] and [74].

2.3.4 Extrapolation

The first application of extrapolation, where convergence is achieved as the number of function evaluations taken approach infinity, in one-dimensional quadrature is known as Romberg integration [65]. Extrapolation was later applied to integration over a square then applied to simplices.

In [51], a particular subdivision of a simplex S_d into m^d sub-simplices of equal volume is described. A quadrature rule is defined in terms of a set of $d!$ simplex quadrature rules, one for each distinct type of sub-simplex occurring in the subdivision. A d-dimensional analogue of the Euler-Maclaurin asymptotic expansion for the simplex valid for analytic integrand functions $\phi(x)$ and any quadrature rule set Q is derived by integrating the representation for $\phi(x)$ obtained from [50].

This has the form

$$Q^{(m)}(S_d)\phi - I(S_d)\phi \approx \sum A_q(Q;\phi)/m^q$$

and vanishes when $\phi(x)$ is a polynomial. For rule sets based on many familiar simplex rules, the odd terms vanish leaving an even expansion.

This method uses an adaptive type procedure by dividing up the domain and repeating the same quadrature rule. Thus, it is better suited to deal with irregularities in the integrand than the other rules presented in this section.

2.4 Monte Carlo and Other Sampling Methods

2.4.1 Introduction

As stated in [11], numerical quadrature rules are inefficient for multidimensional integrals of large dimension ($d \geq 15$). In this section we introduce Monte Carlo integration. We show that for Monte Carlo integration, the error scales like $O(\frac{1}{\sqrt{n}})$, where n is the number of samples, independent of the number of dimensions. However, the convergence a rate of $O(\frac{1}{\sqrt{n}})$ is extremely slow, if substantial accuracy is required. Therefore, we survey several techniques to improve the efficiency of Monte Carlo integration including variance reduction methods, Quasi-Monte Carlo, and Adaptive Monte Carlo Methods.

Further reading on Monte Carlo methods include [33, 38, 81]. If one has little or no background in probability and statistics, references [36] and [73] are places to begin. For quasi-Monte Carlo methods see [43], [57] and [71].

2.4.2 Monte Carlo

As in the previous section, we are concentrating on the integral of a function f over the unit hypercube $\Omega = [0,1]^d$. For this section, we assume that f is square-integrable. We denote a point in the unit hypercube by $\mathbf{x} = (x_1, \ldots, x_d)$ and the function evaluated at this point by $f(\mathbf{x}) = f(x_1, \ldots, x_d)$. The Monte Carlo estimate for the integral

$$If = \int_\Omega f(\mathbf{x}) d\mathbf{x},$$

is given by

$$F_n = \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}_i),$$

where n is the number of samples and $\mathbf{x}_i$ is chosen randomly in Ω. Figure 2.4 gives an example of what a Monte Carlo sampling might look like. This figure also depicts one of the drawbacks of this method, namely that while some areas of the domain are sampled heavily, others are not sampled at all (known as clustering).

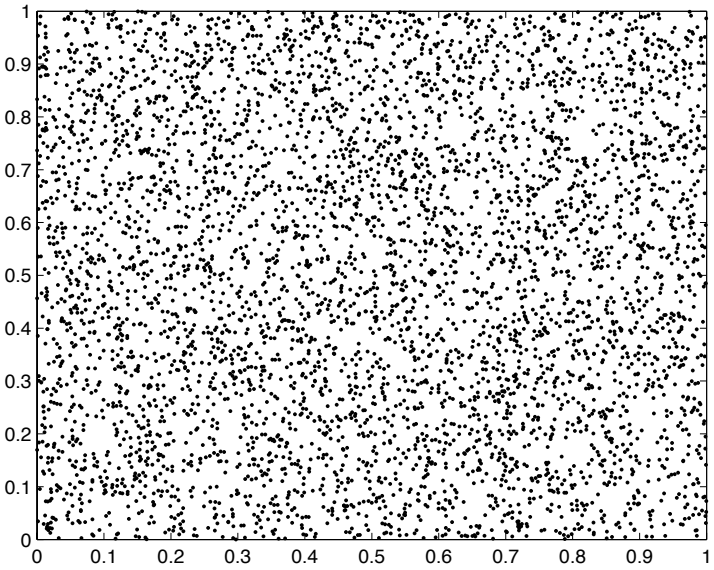


Figure 2.4: Monte Carlo Sampling

Now, from the Law of Large Numbers, we have that the Monte Carlo estimate converges to the true value of the integral as $n \to \infty$,

$$\lim_{n\to\infty} F_n = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}_i) = If.$$

In order to discuss the error estimate for finite n, we must introduce the variance $\sigma^2(f)$ of the function $f(\mathbf{x})$:

$$\sigma^2(f) = \int_\Omega (f(\mathbf{x}) - I)^2 d\mathbf{x}.$$

For F_n we have that

$$\sigma^2(F_n) = \frac{\sigma^2(f)}{n}.$$

Thus the standard deviation $\sigma(F_n) = \frac{\sigma(f)}{\sqrt{n}}$. Now, from the central limit theorem we have the following:

$$\lim_{n\to\infty} P\left(-a\frac{\sigma(f)}{\sqrt{n}} \leq \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}_i) - I \leq b\frac{\sigma(f)}{\sqrt{n}}\right) = \frac{1}{\sqrt{2\pi}}\int_{-a}^{b} \exp\left(\frac{-t^2}{2}\right) dt.$$

This tells us that our calculated value is close to the actual value with a probability based on our choice of a, b and n and also the variance of the function f that we are integrating. Many Monte Carlo variations exist to exploit this fact. In the following subsection we will look at different ways one can reduce the variance of the function being integrated.

2.4.3 Variance Reduction Methods

We have seen that the error estimate of a Monte Carlo integration scales like $O(\frac{1}{\sqrt{n}})$. The main advantage of Monte Carlo integration is that the error estimate is independent of the dimension of the integral. However, the price for that estimate is that the integral converges slowly to the true value as the sample size increases. Several techniques exist to improve the situation. Here we discuss a few of the variations, specifically stratified sampling, importance sampling, and control variates.

Stratified Sampling

This technique consists of dividing the domain of integration into sub-domains, then performing a Monte Carlo integration in each sub-domain, and summing the results.

Specifically, we split the domain Ω into j sub-domains Ω_j where $j = 1, \ldots, m$. In each region we perform a Monte Carlo integration with n_j points. For the integral I we obtain the estimate

$$Q_{MC}^{SS}(f) = \sum_{j=1}^{m} \frac{|\Omega_j|}{n_j} \sum_{i=1}^{n_j} f(\mathbf{x}_{ij}).$$

Our variance now becomes:

$$\sigma^2(F_n) = \frac{\sigma^2(f)}{n} = \sum_{j=1}^{m} \frac{|\Omega_j|}{n_j} \sigma^2(f|_{\Omega_j}),$$

where

$$\sigma^2(f|_{\Omega_j}) = \frac{1}{|\Omega_j|} \int_{\Omega_j} \left(f(\mathbf{x}) - \frac{1}{|\Omega_j|} \int_{\Omega_j} f(\mathbf{x}) d\mathbf{x} \right)^2 d\mathbf{x}.$$

With this method, the error estimate converges at a rate of $O(n^{-1/2(1+2/d)})$, if the function has a bounded first derivative. If the function is only piecewise continuous, then the convergence is $O(n^{-1/2(1+1/d)})$ [55]. This shows that stratified sampling can increases the convergence rate noticeably in low-dimensional problems. However, as the number of dimensions increase we see this method converges to the same rate as the basic Monte Carlo method.

Importance Sampling

Importance sampling is another variance reduction method where one chooses a density function p that is similar to the integrand f. Mathematically, importance sampling corresponds to a change of integration variables

$$\int f(\mathbf{x}) d\mathbf{x} = \int \frac{f(\mathbf{x})}{p(\mathbf{x})} p(\mathbf{x}) d\mathbf{x} = \int \frac{f(\mathbf{x})}{p(\mathbf{x})} dP(\mathbf{x})$$

where

$$p(\mathbf{x}) = \frac{\partial^d}{\partial x_1 \dots \partial x_d} P(\mathbf{x}).$$

If we choose $p(\mathbf{x}) = cf(\mathbf{x})$ with

$$c = \frac{1}{\int_\Omega f(\mathbf{x}) d\mathbf{x}},$$

then $p(\mathbf{x})$ normalizes to unity

$$\int p(\mathbf{x}) d\mathbf{x} = 1.$$

This gives us an estimation to the integral with zero variance, since

$$\frac{f(\mathbf{x_i})}{p(\mathbf{x_i})} = \frac{1}{c}, \quad \forall \mathbf{x_i} \in \Omega.$$

Unfortunately this method is not practical. If we had the exact value for c, we would be done! In practice one chooses $p(\mathbf{x})$ such that it approximates $|f(\mathbf{x})|$ reasonably well in shape and such that one can generate random numbers distributed according to $P(\mathbf{x})$. One disadvantage of importance sampling is that it is dangerous to choose functions $p(\mathbf{x})$, which become zero, or approach zero quickly. If p goes to zero somewhere where f is not zero, $\sigma^2(\frac{f}{p})$ may be infinite and the usual technique of estimating the variance from the sample points may not detect this fact in the regions where $p = 0$, is small.

Control Variates

The method of control variates is very similar to importance sampling. Here, we look for an integrable function p which approximates the function f to be integrated. However, this time the two functions are subtracted rather than divided. Mathematically, this technique is based on the linearity of the integral :

$$\int f(\mathbf{x})d\mathbf{x} = \int (f(\mathbf{x}) - p(\mathbf{x}))d\mathbf{x} + \int p(\mathbf{x})d\mathbf{x}$$

If the integral of p is known, the only uncertainty comes from the integral of $(f - p)$, which will have smaller variance than f, if p has been chosen carefully. The method of control variates is more stable than importance sampling, since zeros in p cannot induce singularities in $(f - p)$. Another advantage over importance sampling is that the integral of the approximating function p need not be inverted analytically.

Both importance sampling and the use of control variates are effective, but choosing the "right" method depends on the particular f and p. In general if $f - p$ is constant function, then p should be used as a control variate; if f/p is a constant, then p should be used for importance sampling [38].

2.4.4 Quasi-Monte Carlo Methods

Quasi-Monte Carlo methods are precisely the Monte Carlo methods already reviewed except they do not require that the samples be chosen randomly. The idea is to deter-

ministically distribute the samples uniformly. This introduces the idea of measuring the irregularity of the distribution of the samples. These measures are called discrepancy measures and are defined with respect to the shape of the domain [57].

The error of Quasi-Monte Carlo integration is at most $O((\log n)^{d-1}/n)$.

2.4.5 Adaptive Monte Carlo Methods

The idea of adaptive sampling (also called sequential sampling) is to take more samples where the integrand has the most variation. This is done by examining the samples that have been taken so far, and using this information to control the placement of future samples. Typically this involves computing the variance of the samples in a given domain, which is then refined by taking more samples if the variance exceeds a given threshold.

Like importance sampling, the goal of adaptive sampling is to concentrate samples where they will do the most good. However, there are two important differences. First, importance sampling attempts to place more samples in regions where the integrand is large, while adaptive sampling attempts to place more samples where the variance is large. (Of course, with adaptive sampling we are free to use other criteria as well.) A second important difference is that with adaptive sampling, the sample density is changed adaptively rather than using *a priori* information.

Another problem with adaptive sampling is that it is not very effective for high-dimensional problems. The same problems are encountered as with stratified sampling: there are too many possible dimensions to refine. For example, if we split the domain to be refined into two pieces along each axis, then we have 2^d new sub-domains to sample. If most of the sampling error is due to variation along only one or two of these axes, the refinement will be very inefficient.

Examples of adaptive sampling include the VEGAS algorithm [48] which learns about the integrand as it proceeds, and multi-channel Monte Carlo [39] which is useful when the integrand is sharply peaked in different regions of the integration region.

2.4.6 Conclusions

Monte Carlo integration offers a tool for numerical evaluation of integrals in high dimensions ($d \geq 15$). Furthermore, Monte Carlo integration works for smooth integrands as well as for integrands with discontinuities. This allows for an easy application to problems with complicated integrands. However the probabilistic error estimate scales with $O(\frac{1}{\sqrt{n}})$. To improve the situation we introduced the classical variance reducing techniques (stratified sampling, importance sampling, control variates) as well as adaptive methods (VEGAS-algorithm, multi-channel Monte Carlo). However effective these methods are, in lower dimensions they still converge to $O(\frac{1}{\sqrt{n}})$ for large values of n.

2.5 Adaptive Quadrature

Adaptive algorithms are widely used in integral approximation. These algorithms have been developed for a variety of geometric regions including hypercubes, hyperspheres and simplices. This section provides a conceptual outline of adaptive quadrature routines. We first set up the framework with key terms and motivation for adaptivity. Then, we discuss the basic algorithm. We declare what we want to give an algorithm (input) and get back (output) and ways we could measure the usefulness of a given algorithm. We also cover some specific variations. The idea here is to depict what is currently in use and motivate the strategies chosen for MAQS.

2.5.1 Setup

For a given function f, a quadrature routine estimates the value of $If = \int_\Omega f(x)dx$. Basic quadrature rules give estimates of the form $Qf = \sum_i w_i f(x_i)$ for specifically chosen weights w_i, and nodes x_i. Adaptive quadrature uses some algorithm to dynamically choose these nodes and weights during the computation to adapt to the particular properties of the integrand f. Alternatively, these algorithms (or routines) vary either

the degree of the quadrature rule being used or the domain that the quadrature rule is being applied to or both, in order to adapt to the behavior of the integrand.

Classically, the first case, where we are not changing the domain but the order of the rule used, is called *automatic quadrature*. This is analogous with the modern concept of p refinement when dealing with the Finite Element Method. The term *adaptive quadrature* specifically applies to routines where the order of the quadrature rule used is fixed and the domain is divided up based on the behavior of the integrand. In this case the modern terminology is h refinement.

Why is adaptivity needed?

Adaptive routines are a necessity when little or nothing at all is known about the function being integrated. In general, adaptive quadrature or a hybrid of adaptive and automatic quadrature (usually just referred to as *adaptive quadrature*, associated with h^p refinement) is chosen over just the automatic version. This allows for larger classes of functions to be integrated more precisely. In [52] the problems with using just an automatic scheme are addressed.

2.5.2 Basic Algorithm

Input / Output

For a general adaptive algorithm one might focus on the following input and output terms.

For input:

- The function being integrated $f : \mathbb{R}^d \longrightarrow \mathbb{R}$,
- the number of dimensions d,
- the tolerance requested E_{tol} and
- the maximum number of function evaluations allowed F_{max}.

For output:

- The integral estimate Qf and
- the error estimate $E(Qf) \geq |Qf - If|$.

Other input items might include:

- A set of points to start the algorithm off with.
- A lower limit on the number of function values to be used.
- An estimate of the noise level in function values (some routines use interval arithmetic to account for these variations [10].
- A minimum area for any subregion.
- Absolute and Relative error tolerance (some routines attempt to satisfy error bound $\leq max$(AbsTol, RelTol$|Qf - If|$). Where $|Qf - If|$ is approximated.

Other output items might include:

- The number of calls to the integrand function subroutine.
- The points used (this could allow the function to be restarted if more accuracy is requested).
- A warning that the minimum area has been reached.
- An infinite or not-a-number function value encountered.

Several other input and output options appear in the literature, including those that exercise control one might want over the adaptive quadrature routine. For a more complete list, see [52].

Basic Steps

As briefly discussed earlier, effective adaptive algorithms break up the domain in some way. This common treatment leads to a general structure that consists of the following four main steps:

1. Choose a subregion from the set of subregions.
2. Subdivide the chosen subregion.
3. Apply the integration rule to the new subregions; update the subregion set.
4. Update global integral and error estimates; check for convergence.

2.5.3 Specific Strategies

The corresponding components that distinguish adaptive algorithms are :

1. A data structure for the subregion set,
2. a subdivision strategy and
3. an integration rule and an error estimator.

Possible Data Structures

Following the recommendations made in [54] for one-dimensional globally adaptive algorithms, one should use the heap subregion data structure, in order to reduce the overheads associated with maintenance of the set. MAQS uses a heap data structure for the subregion set. However, the handling of the data structures is controlled by the programming environment used[1] and not specifically by MAQS.

[1] Matlab Version 2009b

Error Handling

Error handling comes in many forms. Most are combinations of the error estimate of a previous iteration along with the estimate at the current level. These estimates are based on either variation in the order of rule used over a given region or a combination of variates along with information gained by using null rules.

The term null rule was first used in [49]. A rule $Nf = \sum_i u_i f(x_i) = 0$ is a *null rule* if and only if it has at least one nonzero weight, and in addition $\sum_i u_i = 0$. A null rule is furthermore said to have degree r if it integrates to zero all polynomials of degree $\leq r$ and fails to do so with $f(x) = x^{r+1}$. Connections between null rules and divided differences are expressed in [3].

In [3] the error estimator was designed based on a sequence of symmetric null rules of different polynomial degrees. Later, [4] suggested both symmetric and antisymmetric null rules for a slightly modified error estimator. This modification made it possible to construct error estimators by using fewer function evaluations in the basic integration rule.

Now that the error has been established for a given iteration, the next step is to decide where to proceed to subdivide the domain. In other words, which subregion should be divided next. The two variation in the step are based on a local or a global strategy, described in the next subsection.

Subdivision Strategies

Assuming a geometry of dividing up the domain has been chosen and is consistent. In other words if you start by dividing up your domain into cubes or simplices etc. then as you subdivide you continue to have cubes and simplices etc, respectively. As the number of dimensions increases an adaptive routine must be more selective about how much and where the domain is subdivided.

For example, consider the case where you are subdividing a region using hypercubes. At each stage in the adaptive algorithm, a selected subregion must be subdivided. A

simple (natural) subdivision method is to cut at the midpoint of each edge, but this produces 2^d pieces, which is untenable for large d.

Some strategies, use a division of the largest error subregion into at most $m < 2^d$ new pieces, taking into account differences in integrand's behavior in different directions, allows the algorithm to proceed from one stage to the next in a controlled manner. Examples of this type of algorithm are in [5] and [19].

In order for the algorithm to be efficient, a method is needed for selecting good edges for subdivision, and therefore some measure of integrand irregularity is needed. A popular measure of integrand irregularity (detecting high levels of curvature) that has been successfully used with adaptive algorithms for hypercubes is the fourth order divided difference of the integrand. See [5] and [20].

Local v. Global Strategies

The two main strategies for this step are the global and the local strategy. The differences between these two alternatives are investigated here and in [54].

Global Strategy

The globally adaptive algorithm uses successive refinements or subdivisions of Ω, where each subdivision is used to provide a better approximation to If. These subdivisions are designed to dynamically concentrate the computational work in the subregions of Ω where the integrand f is most irregular, and thus to adapt to the behavior of the integrand. The general structure of the globally adaptive algorithm consists of the following five main steps:

1. Select a subregion with largest estimated error from current set of subregions.

2. Divide the selected subregion.

3. Apply a local quadrature rule to the new subregions.

4. Update the subregion set.

5. Update the global integral and error estimates, and check for convergence.

The initial subregion set for the algorithm is the original collection of simplices Ω. The required input for such an algorithm is Ω, the integrand f, a limit on the number of function evaluations allowed, and a requested error tolerance. The algorithm terminates when the estimated global error is less than E_{tol} or when further subdivision would require too many function evaluations.

Another way of interpreting this subdivision strategy is that the cells comprising the region of integration are always arranged in a "priority queue." (This is a structure in which elements are serviced according to some ordering relation. In our case H_Δ defined in Chapter 4.) In the context of this algorithm, the cell at the top of the queue is to be processed next, and this is the cell with the largest error estimate E_Δ, also in Chapter 4. The basic loop is then while the error estimates $> E_{tol}$, reduce error estimate on top cell of queue.

Algorithms of this type have been extensively used for numerical cubature over hyper-rectangular regions, [5], and also for triangles [4] and tetrahedra [2]. Because procedures for the Steps (1), (4) and (5) are the same for all regions, these have been implemented as part of the basic CUBPACK framework in [8].

Local Strategy

The classical way to estimate the local error is based on the knowledge that asymptotically, for f sufficiently smooth and domain of integration sufficiently small, we have $|I - Q| < |Nf|$. Where $|Nf|$ is a null rule of f.

Local error estimation focuses on estimating the local truncation error as reliably and economically as possible. However, rounding errors influence both the estimate of the local integral and the evaluation of all null rules.

Local v. Global

The intention of the global strategy is to select subdomains so that the local errors are roughly equal in magnitude, rather than when scaled by the size of the subdomain. Thus algorithms using global strategies should work no harder on subdomains where

the integrand is difficult to integrate than on subdomains where it is easy to do so. Hence a global criterion has the potential both for reducing the number of subdomains used and for generating a large list of data associated with the pending set.

The advantages of global strategies over local strategies are seen to be a space-time tradeoff. The reduced numbers of required function evaluations and the apparent increase in domain of applicability using the global strategy comes at the expense of additional memory requirements. However, the memory requirements are not unreasonable for many applications, and when a higher order local quadrature rule is used the additional memory requirements are very small.

Generally, a global strategy is preferred when one is concerned with the number of function evaluations used.

Specific Adaptive Strategies

Here are some examples of what differs with each strategy.

Matlab

The computer program Matlab[2] has bundled with its software a few adaptive quadrature schemes. These include:

- `dblquad` - a 2D adaptive Simpson quadrature over a rectangle, based on [17].
- `quad2d` - a 2D adaptive quadrature over a planar region [69, 70] with Gauss-Kronrod (3,7) pair with degrees of precision 5 and 11.
- `triplequad` - a 3D adaptive Labatto quadrature over a rectangular prism, based on [17].

Adaptive Routines over Simplices

In dealing with the simplex geometry specifically, there are a few routines in the literature, found in [2] and [37]. The method in [2], entitled "Algorithm 720," uses a

[2]Matlab Version 2009b

local strategy with cubature formulas of degree 8. The method in [37] uses a global strategy based on interplatory quadrature, without proof.

2.5.4 Conclusions

As we have seen, there are two basic types of adaptive quadrature, automatic and adaptive (p and h). However, the most useful version is a combination of the two and is simply termed "adaptive" (h^p). For this algorithm, there are four main steps.

1. Choose a subregion from the set of subregions.
2. Subdivide the chosen subregion.
3. Apply the integration rule to the new subregions; update the subregion set.
4. Update global integral and error estimates; check for convergence.

Variations in routines occur in dealing with three main subcategories.

1. A data structure for the subregion set,
2. A subdivision strategy and
3. An integration rule and an error estimator.

Of these variations a global strategy is the most useful for keeping function evaluations low and accuracy high, or rather making the algorithm both economical and reliable.

For the other two subcategories, this is where the main differences in the routines lie.

2.6 Conclusions

In this chapter we covered the various methods of approximating the integral of a function in d-dimensions. We looked at sparse grids and their formulation along with possible methods of adapting said formulation to suit the behavior of a given function.

We then discussed the different geometries quadrature methods are based on. Specifically, that of the simplex.

We also reviewed Monte Carlo and other sampling based methods. Which are best suited for large dimension.

Having covered these areas, we then looked at the topic of adaptive quadrature methods. This section plays a key role in the presentation of Chapter 4. For further reading on the subject of numerical quadrature, see [11] or [42].

Chapter 3

Approximation by Piecewise Polynomials

3.1 Introduction

In this chapter we present the theoretical basis for the MAQS algorithm presented in Chapter 4. The main interest being two theorems, one bounding the error of interpolation over a simplex and the other an error bound on intrplatory based quadrature over a simplex. Background is presented followed by the relevant theorems. The formulation and flow of this chapter follow that of [18, 66] and [67]. However, key sources include [45, 47, 46] and [75].

3.1.1 Notation

Using the standard multi-index notation for $\alpha \in \mathbb{N}_0^d$, we define:

$$|\alpha| := \sum_{i=1}^{d} \alpha_i, \qquad \text{and} \qquad D^\alpha = \frac{\partial^{|\alpha|}}{\partial_{x_1}^{\alpha_1} \cdots \partial_{x_d}^{\alpha_d}}$$

With these definitions, we can express any monomial in $\mathbb{R}^d$ by

$$\mathbf{x}^\alpha = \prod_{i=1}^{d} x_i^{\alpha_i}, \quad \text{where} \quad \mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d.$$

Letting $\mathcal{P}^d$ represent the space of all polynomials in d variables, and letting $\mathcal{P}_r^d$ represent the subspace of polynomials of total degree at most r, we see that given any nonnegative integers r and d, we can represent elements $p \in \mathcal{P}_r^d$ by the form

$$p(\mathbf{x}) := \sum_{0 \le |\alpha| \le r} c_\alpha \mathbf{x}^\alpha,$$

where $\{c_\alpha\} \subset \mathbb{R}$. Note that for each d there are $m_r^d = \binom{r+d}{d}$ monomials $\mathbf{x}^\alpha$ with total degree $|\alpha| \le r$. This fact along with proper location of points assures uniqueness of interpolants, which is addressed in Section 3.2.

3.1.2 Norms of Polynomials on Simplices

Given compact any domain $\Omega \subset \mathbb{R}^d$, we define the usual ∞-norm of a function by

$$\|f\|_\Omega := \text{ess sup}_{u \in \Omega} |f(u)|.$$

If f is continuous on Ω, we can replace the essential supremum by the maximum. For $1 \le q < \infty$, we define the usual q-norm by

$$\|f\|_{q,\Omega} := \left[\int_\Omega |f(u)|^q du \right]^{1/q}.$$

To measure the smoothness of f, we make use of the semi-norm

$$|f|_{m,\Omega} := \max_{|\alpha|=m} \|D^\alpha f\|_\Omega.$$

Also, for convenience, we define the diameter of Ω by

$$|\Omega| := \max_{u,v \in \Omega} |u - v|,$$

where $|\cdot|$ is the Euclidean distance in $\mathbb{R}^d$.

3.2 Interpolation by d-variate Polynomials

In its general form, interpolation is the problem of constructing a function p belonging to a finite dimensional linear space from a given set of data. Here our finite dimensional

linear space is $\mathcal{P}_r^d$. We obtain our data by sampling another function f at a specific set of points $\mathcal{X}$. Since both f and p are equal on $\mathcal{X}$, we say that p interpolates f. The one dimensional case is a classical problem and is well understood. However, interpolation by multivariate polynomials is more complicated and is an active area of research. In this section we discuss how we choose our points and from there construct our interpolant.

3.2.1 Point Selection

We say that a set of points $\mathcal{X}_m = \{\mathbf{x}_i\}_{i=1}^m$ is *poised* with respect to $\mathcal{P}_r^d$ if and only if for all $p, q \in \mathcal{P}_r^d$ satisfying $p(\mathbf{x}) = q(\mathbf{x})$ for each $\mathbf{x} \in \mathcal{X}_m$ implies $p \equiv q$.

In one dimension, a sufficient condition for $\mathcal{X}_m$ to be poised (with respect to $\mathcal{P}_{m-1}^1$) is simply that the points themselves be distinct. In multiple dimensions the points must be distinct and satisfy additional geometric conditions.

To view the geometric conditions, we can look at the formulation of an interpolant using the *Vandermonde matrix*,

$$\mathcal{V}_r^d = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1d} & x_{11}x_{12} & x_{11}x_{13} & \cdots & x_{1d}^r \\ 1 & x_{21} & x_{22} & \cdots & x_{2d} & x_{21}x_{22} & x_{21}x_{23} & \cdots & x_{2d}^r \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & x_{m2} & \cdots & x_{md} & x_{m1}x_{m2} & x_{m1}x_{m3} & \cdots & x_{md}^r \end{bmatrix}.$$

Here, we represent the j^{th} component of $\mathbf{x}_i$ by x_{ij}. Below are two examples of the above in two dimensions with a linear and a quadratic matrix

$$\mathcal{V}_1^2 = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}, \qquad \mathcal{V}_2^2 = \begin{bmatrix} 1 & x_1 & y_1 & x_1y_1 & x_1^2 & y_1^2 \\ 1 & x_2 & y_2 & x_2y_2 & x_2^2 & y_2^2 \\ 1 & x_3 & y_3 & x_3y_3 & x_3^2 & y_3^2 \\ 1 & x_4 & y_4 & x_4y_4 & x_4^2 & y_4^2 \\ 1 & x_5 & y_5 & x_5y_5 & x_5^2 & y_5^2 \\ 1 & x_6 & y_6 & x_6y_6 & x_6^2 & y_6^2 \end{bmatrix}.$$

For a unique interpolant this matrix must be invertible or equivalently, $\det \mathcal{V}_n \neq 0$. This requires $m = m_r^d = \binom{r+d}{d}$. For the linear case it is easy to see that the points can not be collinear, otherwise,

$$\mathcal{V}_1^2 = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & ax_1 + b \\ 1 & x_2 & ax_2 + b \\ 1 & x_3 & ax_3 + b \end{bmatrix},$$

resulting in a singular matrix. However, as the number of dimensions increases the restriction that two points not be a linear combination of each other is not enough to ensure that $\det \mathcal{V}_r \neq 0$.

For our work here, we follow the approach of choosing our points or nodes from a grid of the unit simplex formed by the points,

$$\mathbf{x}_\alpha = \frac{\alpha}{r}\ ,\ |\alpha| \leq r. \tag{3.1}$$

See Figures 3.1 and 3.2. These figures indicate the interpolated points for linear, quadratic and cubic interpolants as well as the relationship between the points and the associated Pascal "triangle" of monomial terms that can be interpolated exactly. The theoretical background for this type of point selection in two dimensions and other more general, two dimensional geometries is covered in [35, 44]. The idea of principal lattices, affinely equivalent to the triangular sets, was introduced in [56] and later adapted to multi-dimensions by [7]. The approach therein is based on the idea of taking the intersections of hyperplanes as interpolation nodes, so that products of affine functions can be used to find the interpolation polynomial explicitly, guaranteeing poisedness.

In other words, a set $\mathcal{X}_m$ is poised if $m = \binom{r+d}{d}$ and for each $\mathbf{x}_i \in \mathcal{X}_m$ there exist hyperplanes $H_{il}, l = 1, \ldots, m$, such that $\mathbf{x}_i$ is not on any of these hyperplanes, and all of the other points in $\mathcal{X}_m$ lie on at least one.

By choosing our points using (3.1) we satisfy the conditions in [7], assuring that our selection of points is poised. Now that we have a manageable set of points we explain how to construct the interpolating polynomial.

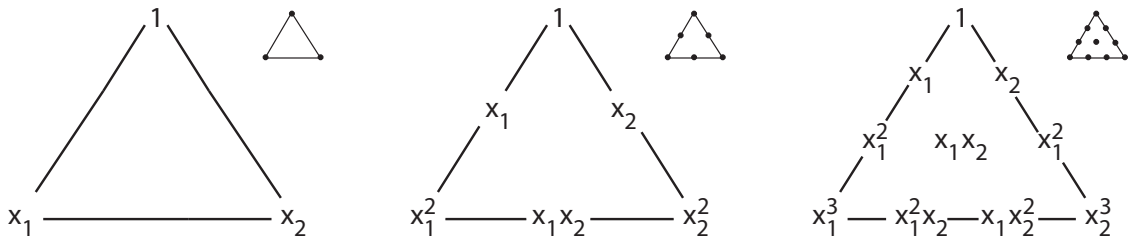


Figure 3.1: Two Dimensional Interpolation Nodes: Linear, Quadratic, and Cubic

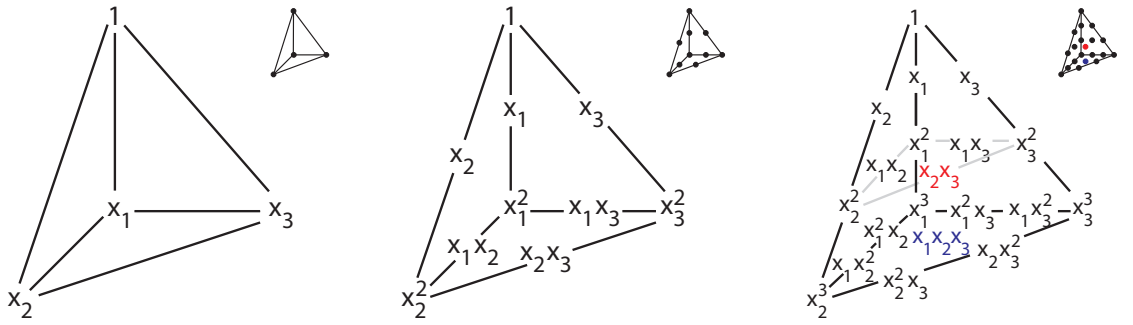


Figure 3.2: Three Dimensional Interpolation Nodes: Linear, Quadratic, and Cubic

3.3 Interpolant Construction

In this section we discuss interpolation over the d-dimensional arbitrary non-degenerative simplex Δ, defined as the convex hull of the $d+1$ points $\{\mathbf{x}_i\}_{i=1}^{d+1}$. Let

$$\mathbb{S}_h = \{\Delta : \max_{u,v\in\Delta} |u - v| \leq h\},$$

be the collection of simplices with diameter at most h. Furthermore, let

$$L_r f(\mathbf{x}) = \sum_{|\alpha|\leq r} c_\alpha \mathbf{x}^\alpha,$$

be the r^{th} degree interpolating polynomial of $f(\mathbf{x})$ over Δ. For $r = 1$ we interpolate at the vertices of Δ. For $r = 2$ at the vertices of Δ and the midpoints of each edge are used. We now define

$$C_M^r(\Delta) = \{f \in C^r(\Delta) : |f|_{\Delta,r} \leq M\}$$

and set

$$a_{r,d} = \sup\left\{ \frac{\|f - L_r f\|_\Delta}{M h^{r+1}} : \ f \in C_M^r(\Delta),\ \Delta \in \mathbb{S}_h,\ h > 0 \right\}. \tag{3.2}$$

Now, we can state the error bounds associated with linear and quadratic interpolation over an arbitrary non-degenerate simplex in d-dimensions. The proof of these results are contained in [75].

Theorem 2. *For any $d \in \mathbb{N}$ we have the equality*

$$a_{1,d} = \frac{1}{4}\frac{d}{d+1}. \tag{3.3}$$

Theorem 3. *For any $d \in \mathbb{N}$ with $d > 2$ we have the estimates*

$$\frac{\sqrt{2}}{48}\frac{d\sqrt{d}}{(d+1)\sqrt{d+1}} \leq a_{2,d} \leq \frac{5}{16}\frac{d}{d+1}. \tag{3.4}$$

Now we make a precise statement about the bound on the 1-norm of the error for the linear and quadratic interpolating polynomials.

Corollary 3.3.1. *For $f \in C_M^{r+1}(\Delta)$, $d \in \mathbb{N}$ with $d > 2$ and $|\Delta| < 1$ we have the following :*

$$\|L_1 f - f\|_{1,\Delta} \leq \mathrm{vol}(\Delta)|\Delta|^2, \tag{3.5}$$

and

$$\|L_2 f - f\|_{1,\Delta} \leq \mathrm{vol}(\Delta)|\Delta|^3. \tag{3.6}$$

Proof. For the linear Lagrange interpolation case:

$$\begin{aligned}
\|L_1 f - f\|_{1,\Delta} &= \int_\Delta |L_1 f(\mathbf{x}) - f(\mathbf{x})| d\mathbf{x} \\
&\leq \int_\Delta \|L_1 f(\cdot) - f(\cdot)\|_\Delta d\mathbf{x} \\
&\leq \mathrm{vol}(\Delta)\|L_1 f(\cdot) - f(\cdot)\|_\Delta \\
&\leq \frac{1}{4}\frac{d}{d+1}\mathrm{vol}(\Delta)|\Delta|^2 \qquad \text{by (3.3)} \\
&\leq \mathrm{vol}(\Delta)|\Delta|^2.
\end{aligned}$$

For the quadratic case:

$$\begin{aligned}
\|L_2 f - f\|_{1,\Delta} &= \int_\Delta |L_2 f(\mathbf{x}) - f(\mathbf{x})| d\mathbf{x} \\
&\leq \int_\Delta \|L_2 f(\mathbf{x}) - f(\mathbf{x})\|_\Delta d\mathbf{x} \\
&\leq \mathrm{vol}(\Delta)\|L_2 f(\mathbf{x}) - f(\mathbf{x})\|_\Delta \\
&\leq \frac{5}{16}\frac{d}{d+1}\mathrm{vol}(\Delta)|\Delta|^3 \qquad \text{by (3.4)} \\
&\leq \mathrm{vol}(\Delta)|\Delta|^3.
\end{aligned}$$

□

These two statements are crucial in the next chapter, when proving convergence of the algorithm.

Chapter 4

The Algorithm (MAQS)

4.1 Introduction

In this chapter we present the main focus of this dissertation, the Multidimensional Adaptive Quadrature routine over Simplices (MAQS). This algorithm takes a given function and approximates the integral over the unit hypercube. This is accomplished by dividing the hypercube up into simplices then obtaining an estimate of the integral, along with an estimate of the error of the approximation, for each simplex. The algorithm then creates a priority queue of the simplices, based on the size and estimated error associated with each simplex. The algorithm then proceeds to process the simplex associated with the highest priority, in order to decrease the estimated error as much as possible at each iteration. The algorithm processes each simplex by dividing it into 2^d sub-simplices and obtains integral and error estimates for each new simplex. This process is repeated until the required error tolerances or other stopping criteria (e.g. maximum allowed function evaluations) are met.

The integral estimate is based on symmetric interpolatory quadrature rules. This allows for function evaluations to be used and reused until the algorithm is terminated. Additionally, each simplex can be processed independently, which allows for the parallel processing of multiple simplices at once. Due to the use of interpolatory based

quadrature methods, restarting the calculation for greater precision requires no additional function evaluations than the desired tolerance initially required. Also, using simplices allows for the algorithm to handle arbitrary polyhedral domains that can be approximated by tessellation.

4.2 Basic Strategy

4.2.1 Formulation

We are concerned with approximating the value of the integral of functions f from a function class $\mathcal{F}$ over the d-dimensional hypercube $\Omega := [0,1]^d$. Let

$$I : \mathcal{F} \to \mathbb{R}, \quad \text{be defined as} \quad If := \int_\Omega f(\mathbf{x}) d\mathbf{x}.$$

We obtain our integral estimate Q_Δ over each simplex Δ from the interpolation defined in Section 3.3 and the quadrature over a simplex defined in Section 2.3.3. We then sum over all $\Delta \in \mathscr{T}_n$ to obtain our integral estimate for the entire domain Ω,

$$Q_\Omega = \sum_{\Delta \in \mathscr{T}_n} Q_\Delta.$$

The error estimate for our approximation on a given simplex Δ is defined by

$$E_\Delta = \|L_1 f - L_2 f\|_{1,\Delta}.$$

We define our total error estimate as the summation of all error estimates for a fixed tessellation $\mathscr{T}_n$,

$$E_{\Omega,n} = \sum_{\Delta \in \mathscr{T}_n} E_\Delta.$$

Error estimation is covered in Section 4.5.

The center piece of any adaptive quadrature routine is the strategy for refining the mesh and producing successively more accurate integral estimates. There are two main strategies for choosing how to proceed with the next iteration of the algorithm.

A local strategy, where the goal is to have the estimated error of each simplex below a given tolerance and a global strategy, where the goal is to have the sum of all the estimated errors for each element less than a given tolerance. The method presented here, MAQS, uses a global strategy. This allows the algorithm to avoid instances where the error might be high in one simplex but the simplex in question is so small that it does not significantly contribute to the integral estimate. This is done by defining a heuristic value, H_Δ, that can be used to arrange the simplices in a priority queue. For MAQS, a fixed number of simplices at the top of the queue (with the highest heuristic values) are processed next in an effort to have maximal gains in error reduction. We enforce an upper bound on the number of function evaluations used, F_{evals} to ensure termination of algorithm. Below is the statement of the MAQS algorithm.

4.2.2 The Algorithm

Given a function f, an error tolerance E_{tol}, and a maximum number of allowable function evaluations F_{max}, our algorithm proceeds as follows.

1. The domain, $\Omega = [0,1]^d$ is divided into $d!$ simplices.
2. The values $(Q_\Omega, E_\Omega, H_\Delta)$ are initialized to $(0, \infty, \infty)$.
3. The main *while* loop is initiated.
   - Here we have the following criteria
   $$\text{while } \left(E_\Omega = \sum_{\Delta \in \mathscr{T}_n} E_\Delta > \epsilon \text{ or } F_{evals} < F_{max} \right).$$
4. For the current iteration, n, the simplex (simplices) at the top of the priority queue or the simplex (simplices) with the largest H_Δ value is (are) processed.
   - Here the midpoints of each edge of the simplex are evaluated to get a quadratic approximation to f over the simplex.

- This sampling also allows for a linear approximation to f over the 2^d sub-simplices.
- From the linear approximation, a linear quadrature estimate for each new simplex is returned as the Q_Δ value.
- From the integral of the absolute value of the difference of the linear interpolation over each new simplex with that of the quadratic interpolation over the parent simplex, the error estimate E_Δ is obtained for each new simplex.
- After the Q_Δ and E_Δ values are computed we can assign each new simplex a priority value H_Δ.
- These three values are stored for each simplex.

5. After processing is complete, we calculate the new integral and error estimates and repeat the loop.

Other criteria can be added. For common options see Section 2.5.2.

4.3 Using Simplices

Quadrature in one-dimension only involves intervals. In more than one dimension there is a wide variety of possible regions, however most algorithms consider one fixed class of geometric objects. The algorithm we have implemented uses the simplex as its basic element. By this we mean that the input region must be described as a union of finitely many simplices which are disjoint except perhaps along n-faces, and that subdivision produces additional simplices. Our motivation for choosing such a geometry include:

- General regions can be approximately represented by a union of simplices than by cubes.
- Quadrature formulas of interpolatory type are very natural simplices.

The $d+1$ vertices of a simplex are natural nodes for a quadrature formula which is exact for the $d+1$ linear polynomials. These same vertices plus the $(d(d+1)/2)$ midpoints can serve as nodes for a formula that is exact for quadratics. These $(d+1)$ vertices can serve as vertices for the 2^d sub-simplices. This opportunity to reuse integrand values both for the subdivision and for different formulas allows for the possibility of a quadrature algorithm which is both variable order and variable mesh. MAQS uses only the linear and quadratic approximation on each simplex. However, natural geometric extensions are possible

4.4 Subdivision Strategy

MAQS uses the following heuristic value to prioritize each simplex Δ,

$$H_\Delta = a|\Delta| + bE_\Delta \tag{4.1}$$

where $a, b \geq 0$.

An important step in the algorithm is the subdivision of an arbitrary simplex into 2^d sub-simplices which have as vertices those of the original simplex and their midpoints. For a hypercube there is only one possible decomposition, while for 2-dimensional problems there are already four possible non-overlapping sub-triangulations. The subdivision we use is one that produces 2^d equal volume sub-simplices which are not necessarily congruent to each other in higher dimensions.

For dissection of each simplex we use a Delaunay tessellation [80]. Specifically, for a set $\mathcal{X}$ of points in the (d-dimensional) Euclidean space, a Delaunay tessellation is a tessellation $\text{tri}(\mathcal{X})$ such that no point in $\mathcal{X}$ is inside the circumscribed hypersphere of any simplex in $\text{tri}(\mathcal{X})$.

Lemma 1. *Consider the algorithm in Section 4.2.2, if (a, b) is chosen such that $H_\Delta = a|\Delta| + bE_\Delta = |\Delta|$, i.e. $(a, b) = (1, 0)$, so that iteration is dedicated to processing the*

largest simplex, then for $\epsilon > 0$ *the number of iteration* n *needed to have exactly* $2^{(m-1)d}d!$ *simplices with maximal edge length* $|\Delta| = \sqrt{2}/2^{m-1} \leq \epsilon$*, is*

$$n = \sum_{i=1}^{M} 2^{(i-2)d}d! + 1, \tag{4.2}$$

where

$$M = \left\lceil \log_2 \left(\sqrt{2}/\epsilon\right) + 1 \right\rceil.$$

Proof. By initially sampling at the corners of the hypercube of d-dimensions we can construct $d!$ simplices, with $|\Delta| = \sqrt{2}$ and $\text{vol}(\Delta) = 1/d!$, for all $\Delta \in \mathscr{T}_1$. If we then set $H_\Delta = |\Delta|$ by choosing $(a, b) = (1, 0)$, the algorithm (4.2.2) will always process the simplex with the largest edge. The number of iterations to divide up the current mesh is $d!$. At that point, there are $2^d d!$ simplices. This stems from by dividing each edge in half. Now we have $|\Delta| = \sqrt{2}/2$ and $\text{vol}(\Delta) = 1/2^d d!$, for all $\Delta \in \mathscr{T}_2$ (since the sub-simplices have equal volume).

The number of iterations to completely refine the mesh one more level is $2^d d!$, one iteration for each simplex. At that point, there $|\mathscr{T}_n| = 2^{2d}d!$ simplices with $|\Delta| = \sqrt{2}/4$ and $\text{vol}(\Delta) = \frac{1}{2^{2d}d!}$. The pattern continues, see Table (4.1).

Level	Iteration n	$\lvert\mathscr{T}_n\rvert$	Edge Length $= \lvert\Delta\rvert$	$\text{vol}(\Delta)$
1	1	$d!$	$\sqrt{2}$	$1/d!$
2	$d!$	$2^d d!$	$\sqrt{2}/2$	$1/(2^d d!)$
3	$2^d d!$	$2^{2d} d!$	$\sqrt{2}/4$	$1/(2^{2d} d!)$
4	$2^{2d} d!$	$2^{3d} d!$	$\sqrt{2}/8$	$1/(2^{3d} d!)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
M	$2^{(m-2)d} d!$	$2^{(M-1)d} d!$	$\sqrt{2}/2^{M-1}$	$1/(2^{(M-1)d} d!)$

Table 4.1: Complexity of Uniform Refinement: Iterations Needed, Edge Length and Volume per Level of Uniform Refinement

Therefore, if we fix $\epsilon > 0$, let $M = \left\lceil \log_2 \left(\sqrt{2}/\epsilon\right) + 1 \right\rceil$ and take $n = \sum_{i=1}^{M} 2^{(i-2)d}d! + 1$

steps through the algorithm we have

$$\begin{aligned} |\Delta| &= \sqrt{2}/2^{M-1} \\ &= \sqrt{2}/(2^{\lceil \log_2(\sqrt{2}/\epsilon)+1 \rceil - 1}) \\ &< \sqrt{2}/(2^{\log_2(\sqrt{2}/\epsilon)}) \\ &< \sqrt{2}/(\sqrt{2}/\epsilon) \\ &< \epsilon. \end{aligned}$$

This proves the Lemma. □

Lemma 2. *For every iteration n and for all $\Delta \in \mathscr{T}_n$, the heuristic function H_Δ defined by (4.1) has the following bounds*

$$K_1|\Delta| \leq H_\Delta \leq K_2|\Delta|, \tag{4.3}$$

where $K_1, K_2 \in \mathbb{R}^+$ and $K_1 < K_2$.

Proof. Given, $H_\Delta = a|\Delta| + bE_\Delta$ and $E_\Delta = \|L_1 f - L_2 f\|_{1,\Delta}$. The lower bound can be show by setting $K_1 = a$. Since $b, E_\Delta \geq 0$ we have,

$$a|\Delta| \leq H_\Delta. \tag{4.4}$$

For the upper bound, we use the estimates from Corollary (3.3.1) to obtain

$$\begin{aligned} E_\Delta &= \|L_1 f - L_2 f\|_{1,\Delta} \\ &\leq \|L_1 f - f\|_1 + \|f - L_2 f\|_{1,\Delta} \\ &\leq \mathrm{vol}(\Delta)|\Delta|^2 + \mathrm{vol}(\Delta)|\Delta|^3 \\ &\leq C_1 \mathrm{vol}(\Delta)|\Delta|^2. \end{aligned} \tag{4.5}$$

We have,

$$\begin{aligned} H_\Delta &= a|\Delta| + bE_\Delta \\ &\leq a|\Delta| + C_2 \mathrm{vol}(\Delta)|\Delta|^2 \\ &\leq C_3|\Delta|. \end{aligned} \tag{4.6}$$

Therefore, setting $K_2 = C_3$ above and combining (4.4) and (4.6) we have,

$$K_1|\Delta| \leq H_\Delta \leq K_2|\Delta|.$$

□

Theorem 4.

$$\lim_{n\to\infty} \sup_{\Delta\in\mathscr{T}_n} |\Delta| = 0$$

Proof. Let $\epsilon > 0$, and define $\hat{\epsilon} = \frac{K_1}{K_2}\epsilon$, where K_1 and K_2 are bounds on the heuristic function in (4.3). Now, choose $n = \sum_{i=1}^{M-1} 2^{(i-2)d}d! + 1$, where $M = \left\lceil \log_2\left(\sqrt{2}/\hat{\epsilon}\right) + 1 \right\rceil$. We have chosen our n to be the largest number of iterations such that for all simplices $\Delta \in \mathscr{T}_n$ no more than one level of refinement is required before $|\Delta| \leq \hat{\epsilon}$. At this number of iterations, there could definitely be simplices with $|\Delta| < \hat{\epsilon}$ but there can be none with a value larger than $2\hat{\epsilon}$ or no values of $|\Delta|$ more than one level behind in refinement. The idea is that there is a bounded number of iterations the algorithm can go through before each simplex has been refined, if $H_\Delta = |\Delta|$, as shown in Lemma 1

Now, proceeding one step further in the algorithm we must process the simplex with the largest $H_{\hat{\Delta}}$ per algorithm 4.2.2. From Lemma 4.3 we have

$$H_{\hat{\Delta}} \leq K_2|\hat{\Delta}|.$$

However, from our choice of n, we know $|\hat{\Delta}| \leq \hat{\epsilon}$. In other words,

$$H_{\hat{\Delta}} \leq K_2|\hat{\Delta}| \leq K_2\frac{K_1}{K_2}\epsilon \leq K_1\epsilon.$$

Now, since all $\Delta \in \mathscr{T}_n$ from our last step were ordered in the priority queue by their H_Δ values, we know the for the simplex that was just processed

$$H_{\hat{\Delta}} \geq H_\Delta, \quad \text{for all} \quad \Delta \in \mathscr{T}_n$$

Therefore, since H_Δ is bounded below by (4.4) we have

$$K_1|\Delta| \leq H_\Delta \leq H_{\hat{\Delta}} \leq K_1\epsilon.$$

Which gives us the final result,

$$|\Delta| \leq \epsilon.$$

□

Corollary 4.4.1.

$$\lim_{n\to\infty} \sup_{\Delta\in\mathscr{T}_n} \|L_i f - f\|_{1,\Delta} = 0, \quad for \;\; i = 1, 2$$

Proof. From Corollary 3.3.1 and the above theorem, we have

$$\begin{aligned} \lim_{n\to\infty} \sup_{\Delta\in\mathscr{T}_n} \|L_i f - f\|_{1,\Delta} &\leq \lim_{n\to\infty} \sup_{\Delta\in\mathscr{T}_n} \mathrm{vol}(\Delta)|\Delta|^{i+1} \\ &\leq \lim_{n\to\infty} \sup_{\Delta\in\mathscr{T}_n} |\Delta| \\ &= 0 \end{aligned}$$

for $i = 1, 2$. □

4.5 Error Estimation

For the error estimate over a given simplex Δ, we define the following:

$$E_\Delta = \|L_1 f - L_2 f\|_{1,\Delta}. \tag{4.7}$$

Taking the summation of all error estimates for a fixed tessellation we define or total error estimate to be

$$E_{\Omega,n} = \sum_{\Delta\in\mathscr{T}_n} E_\Delta. \tag{4.8}$$

A crucial step in any algorithm is the proof that in the absence of machine error, limited time, memory and maximum allowed function evaluations, the algorithm should be able to achieve any level of accuracy requested. The following theorem guarantees this for the MAQS algorithm.

Theorem 5.

$$\lim_{n\to\infty} E_{\Omega,n} = 0.$$

Proof. By 4.5, we know

$$E_\Delta \leq C_1 \text{vol}(\Delta)|\Delta|^2.$$

Thus,

$$\begin{aligned} E_{\Omega,n} &= \sum_{\Delta \in \mathscr{T}_n} E_\Delta \\ &\leq \sum_{\Delta \in \mathscr{T}_n} C_1 \text{vol}(\Delta)|\Delta|^2. \end{aligned}$$

This C_1 is a fixed constant independent of Δ and n, thus

$$E_{\Omega,n} \leq C_1 \sum_{\Delta \in \mathscr{T}_n} \text{vol}(\Delta)|\Delta|^2.$$

Now, let $\hat{\Delta}$ be the largest simplex in the mesh at iteration n. This yields,

$$\begin{aligned} E_{\Omega,n} &\leq C_1 \sum_{\Delta \in \mathscr{T}_n} \text{vol}(\Delta)|\hat{\Delta}|^2 \\ &\leq C_1|\hat{\Delta}|^2 \sum_{\Delta \in \mathscr{T}_n} \text{vol}(\Delta). \end{aligned}$$

We know at each iteration, $\mathscr{T}_n$ is a tessellation of the domain Ω. Since $\Omega = [0,1]^d$, we have $\sum_{\Delta \in \mathscr{T}_n} \text{vol}(\Delta) = 1$. Which implies,

$$E_{\Omega,n} \leq C_1|\hat{\Delta}|^2.$$

Therefore, by Theorem 4, we know that for all $\Delta \in \mathscr{T}_n$, $|\Delta| \overset{n\to\infty}{\longrightarrow} 0$ and thus $|\hat{\Delta}| \overset{n\to\infty}{\longrightarrow} 0$, completing the proof. □

From Corollary 4.4.1 and Theorem 5, we have that both the true error and the error estimate of our approximation converge to zero, which is a curtail property of an error estimator. Satisfying this property, we now want to know in the presence of machine error, time limitations and a bounded number of function evaluations, what sort of error estimate can we have for a finite number of iterations in the algorithm.

For this algorithm, we have decided to use E_Ω, based on the decision to use a global strategy, as our error estimate.

Now, for a given Δ, we examine the ratio

$$\frac{\|L_1 f - L_2 f\|_{1,\Delta}}{\|L_1 f - f\|_{1,\Delta}}.$$

We then notice the following relationship

$$\begin{aligned}
\frac{\|L_1 f - L_2 f\|_{1,\Delta}}{\|L_1 f - f\|_{1,\Delta}} &= \frac{\|L_1 f - f + f - L_2 f\|_{1,\Delta}}{\|L_1 f - f\|_{1,\Delta}} \\
&\leq \frac{\|L_1 f - f\|_{1,\Delta}}{\|L_1 f - f\|_{1,\Delta}} + \frac{\|L_2 f - f\|_{1,\Delta}}{\|L_1 f - f\|_{1,\Delta}} \\
&\leq 1 + \frac{\|L_2 f - f\|_{1,\Delta}}{\|L_1 f - f\|_{1,\Delta}}.
\end{aligned}$$

At this point we make a common assumption. That is, since we have from (3.5) and (3.6), $\|L_2 f - f\|_{1,\Delta} = O(|\Delta|^3)$ and $\|L_1 f - f\|_{1,\Delta} = O(|\Delta|^2)$ respectively, we assume $\frac{\|L_2 f - f\|_{1,\Delta}}{\|L_1 f - f\|_{1,\Delta}} = O(|\Delta|)$. Which implies

$$\lim_{|\Delta| \to 0} \frac{\|L_1 f - L_2 f\|_{1,\Delta}}{\|L_1 f - f\|_{1,\Delta}} = 1. \tag{4.9}$$

This tells us that our error estimate is a good estimate of the true error. However, for finite n we do not know what our exact error is but we now have reason to believe it is close to our estimate with this assumption. In the next chapter we will report on the above ratio 4.9 to verify that this is a fair assessment.

4.6 Additional Properties

4.6.1 Restarting

By storing the points used and the function evaluation of those points, one could go through the algorithm again with the goal of achieving a higher level a accuracy. This could happen without requiring anymore function evaluations than if the new level of accuracy had been initially requested.

4.6.2 Parallel Possibilities

In order to parallelize this algorithm all that is needed is to send any number of simplices to different machines. Simply take the top p simplices from the priority queue and process them simultaneously.

One complication in the implementation is duplication of points on adjacent interfaces. Thus for expensive function evaluations, the selection of points should be performed on one processor. Then the evaluation of the function at these points can be done in parallel. In addition, once all the function values are known, the evaluation of the integral and heuristic functions can be done in parallel (the trade-off in data exchange and processing time will depend on the number of dimension, d).

Chapter 5

Numerical Validation

5.1 Introduction

In this chapter we present the numerical results obtained from testing MAQS on two suites of functions. The first test suite is a family of common functions used to compare quadrature rules over the unit hypercube. This specific set of functions were chosen from [20]. We refer to this family as the Genz-Malik suite of test functions (see Table 5.3). The second family of functions are three examples that motivate the need for the algorithm presented in this dissertation, selected from references [16, 53] as well as the authors' own creation. Each test suite has a special attribute, a difficulty parameter, and/or random parameters.

5.2 Testing Procedures

In these experiments, the domain of integration for all test families is the unit hypercube. These experiments correspond to a maximum $F_{max} = 120,000$ integrand evaluations. For each test function with random variables, an aggregate based on 20 samples is used.

For the two dimensional tests, we compared the results of MAQS against those of

Matlab's `quad2d`. This comparison is based on the fact that `quad2d` is also a domain adaptive quadrature like MAQS. More information on Matlab's `quad2d` can be found in [69] and Section 2.5.3.

For the three- through five-dimensional cases, MAQS is compared to Monte Carlo integration (MCi). Our reasoning for choosing MCi over the sparse grid software offered by the Klimke Matlab toolbox [40], based on the algorithm in [41], is due to the fact that the software is only degree adaptive and has been shown to fail when functions have discontinuities.

5.3 Validation of Error Estimation and Convergence Theory

Table 5.3 lists the five test families considered for the first suite of experimentation. The parameters $\{\alpha_i, \beta_i\} \subset [0, 1]$ are chosen using a uniform distribution provided by the `rand` function in Matlab using the following commands:

```
samples = 20;
stream = RandStream('mrg32k3a');
rand_parameters = rand(stream, samples, number_of_parameters);
```

The inputs are: number_of_parameters – the number of parameter values per sample and `'mrg32k3a'` – one of the six multiple recursive generators offered by Matlab. This provides "random" data while allowing for repeatable experiments.

5.3.1 Uniform Refinement Experiments

The first experiment is the numerical validation of Equation (4.9). For this test we use our interplatory quadrature over uniformly refined meshes to obtain our integral and error estimates. We do this to provide evidence that for the given test family, which

$f(\mathbf{x}) = \cos(2\pi\beta + \sum_{i=1}^{d} \alpha_i x_i))$	Oscillatory
$f(\mathbf{x}) = \prod_{i=1}^{d} (\alpha_i^{-2} + (x_i - \beta_i)^2)^{-1}$	Internal Peak
$f(\mathbf{x}) = (1 + \sum_{i=1}^{d} \alpha_i x_i)^{-(d+1)}$	Corner Peak
$f(\mathbf{x}) = \exp(-\sum_{i=1}^{d} (\alpha_i^2 (x_i - \beta_i)^2))$	Gaussian
$f(\mathbf{x}) = \exp(-\sum_{i=1}^{d} \alpha_i \lvert x_i - \beta_i \rvert)$	C^0 Function

Table 5.1: Functions in the Genz-Malik Test Suite

falls under the theory provided in Chapter 3, our error estimator is a good choice. The data from these experiments are provided in Tables 5.2, 5.3, 5.4, 5.5, and 5.6. Figure 5.1 shows the points used and the associated triangular mesh at various stages in the uniform refinement study. Note the C^0 function does not satisfy the conditions that justify our error estimator. In this case, the ratio of the estimate to the true error seems to converge to 4 in Table 5.6 (instead of the ideal value of 1). However, although not optimal, this still confirms the form of our heuristic function H_Δ in (4.1), although with a different value of the constant b.

5.3.2 MAQS Experiment Demonstrating Decreasing $\sup|\Delta|$

We conducted one experiment to validate Theorem 4, showing that as MAQS proceeds through iterations, the maximum size of the largest simplex decreases. Figure 5.2 depicts the function $f(\mathbf{x}) = (x_1 + x_2)^3$ used in this study along with different snapshots in the execution of MAQS. Table 5.7 shows the size of the maximum simplex with respect to iteration. Note that Figure 5.2 indicates that a pattern forms as the mesh is refined. This can be explained by noting the smooth variation in the function contours from the lower left to the upper right where the curvature is sharpest. Since MAQS was designed to hunt out discontinuities in an arbitrary location within the domain, the initial mesh is not uniformly defined nor biased toward any direction. This causes some areas to be refined more than others when the function is smooth due to the

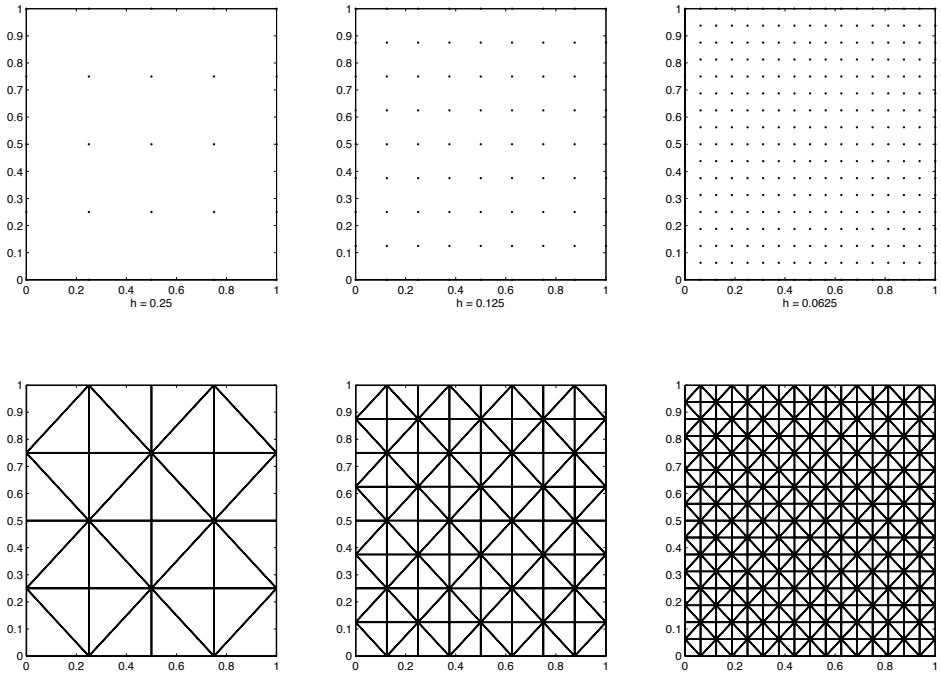


Figure 5.1: Points and Simplices Used in Uniform Refinement

$\lvert\Delta\rvert/\sqrt{2}$	Q_Ω	$\lVert L_1 f - L_2 f\rVert$	$\lVert L_1 - f\rVert$	$\frac{\lVert L_1 f - L_2 f\rVert}{\lVert L_1 - f\rVert}$
0.1000000000	0.5001324357	0.0001162541	0.0001094281	1.0623794615
0.0500000000	0.5002204117	0.0000281309	0.0000278043	1.0117481870
0.0250000000	0.5002413238	0.0000071062	0.0000064301	1.1051332142
0.0125000000	0.5002466284	0.0000017789	0.0000018474	0.9628814279

Table 5.2: Uniform Refinement: Oscillatory

$\lvert\Delta\rvert/\sqrt{2}$	Q_Ω	$\lVert L_1 f - L_2 f\rVert$	$\lVert L_1 - f\rVert$	$\frac{\lVert L_1 f - L_2 f\rVert}{\lVert L_1 - f\rVert}$
0.1000000000	0.3992788651	0.0001613944	0.0001576382	1.0238283004
0.0500000000	0.3993996335	0.0000404425	0.0000429194	0.9422880173
0.0250000000	0.3994299419	0.0000100742	0.0000090795	1.1095602350
0.0125000000	0.3994374809	0.0000025194	0.0000027258	0.9242742800

Table 5.3: Uniform Refinement: Internal Peak

$\lvert\Delta\rvert/\sqrt{2}$	Q_Ω	$\lVert L_1 f - L_2 f\rVert$	$\lVert L_1 - f\rVert$	$\frac{\lVert L_1 f - L_2 f\rVert}{\lVert L_1 - f\rVert}$
0.1000000000	0.2911109874	0.0002578628	0.0002597141	0.9928715315
0.0500000000	0.2909212881	0.0000679562	0.0000596666	1.1389322473
0.0250000000	0.2908699346	0.0000165897	0.0000172477	0.9618555390
0.0125000000	0.2908575227	0.0000041770	0.0000041565	1.0049375807

Table 5.4: Uniform Refinement: Corner Peak

$\lvert\Delta\rvert/\sqrt{2}$	Q_Ω	$\lVert L_1 f - L_2 f\rVert$	$\lVert L_1 - f\rVert$	$\frac{\lVert L_1 f - L_2 f\rVert}{\lVert L_1 - f\rVert}$
0.1000000000	0.8444379607	0.0004095016	0.0004008291	1.0216364313
0.0500000000	0.8447448094	0.0001026224	0.0000969753	1.0582319354
0.0250000000	0.8448218633	0.0000255535	0.0000233453	1.0945875936
0.0125000000	0.8448410247	0.0000063890	0.0000068409	0.9339356924

Table 5.5: Uniform Refinement: Gaussian

$\lvert\Delta\rvert/\sqrt{2}$	Q_Ω	$\lVert L_1 f - L_2 f\rVert$	$\lVert L_1 - f\rVert$	$\frac{\lVert L_1 f - L_2 f\rVert}{\lVert L_1 - f\rVert}$
0.1000000000	0.6335233771	0.0010095902	0.0005727571	1.7626847337
0.0500000000	0.6336873849	0.0001366776	0.0000479603	2.8498070894
0.0250000000	0.6337762602	0.0000514701	0.0000115348	4.4621724054
0.0125000000	0.6337821267	0.0000076275	0.0000033677	2.2648723352

Table 5.6: Uniform Refinement: C^0 Function

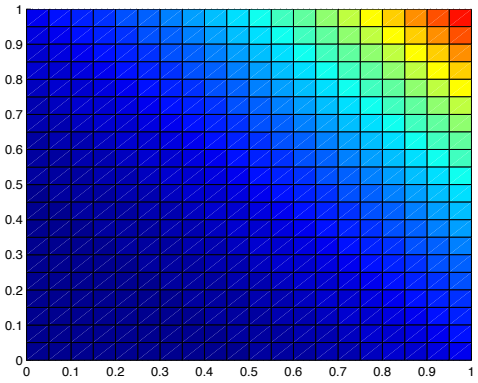
initial orientation of the simplices. For example the top right corner has simplices with the longest edge going down from left to right, while the top left corner is the exact opposite. Thus, there is a natural bias to subdivide simplices that have their longest edge parallel to the $x_1 = x_2$ line. Again, this is not optimal in this case but, we feel this heuristic placement is a sound choice when trying to assume little about the function being integrated. This behavior is also seen in Figure 5.3 for one sample of the Oscillatory family.

5.4 Test Suite I: Genz-Malik Functions

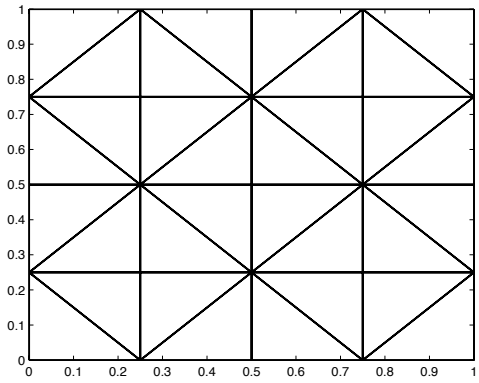
5.4.1 Genz-Malik 2-Dimensional Study

This section covers the common family of test functions used to benchmark many quadrature methods. The functions are listed in Table 5.3. Here we depict each function that is being integrated and then show the final mesh at MAQS termination, the points MAQS used, along with the points Matlab's `quad2d` function used. After this we provide a table that depicts the number of iterations n the values of the integral estimate Q_Ω, the error estimate, E_Ω, the actual error, and the ratio of the estimated error and actual error to numerically validate our choice for E_Ω.

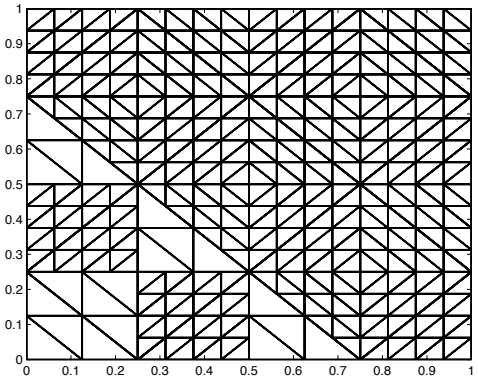
The experiments for the two dimensional study are provided in Table 5.8. Note that every case comes very close to having a ratio of the error estimate to the true error of 1.



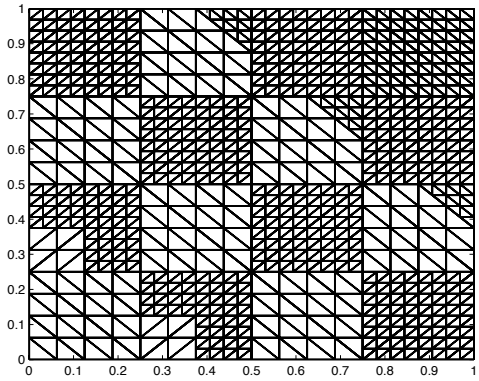
(a) $f(\mathbf{x}) = (x_1 + x_2)^3$



(b) Initial Mesh: $n = 1$



(c) Mesh at iteration: $n = 317$



(d) Mesh at iteration: $n = 641$

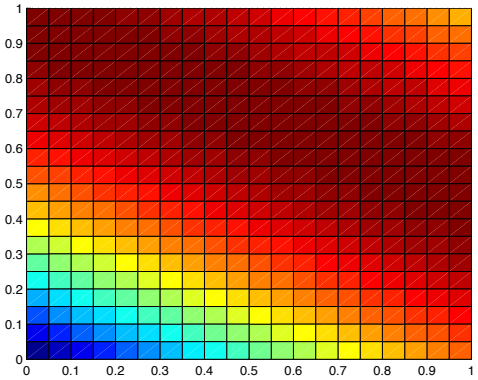
Figure 5.2: Demonstration that $\max_{\Delta \in \mathscr{T}_n} |\Delta| \overset{n \to \infty}{\longrightarrow} 0$: $f(\mathbf{x}) = (x_1 + x_2)^3$

n	$\max_{\Delta \in \mathscr{T}_n} \lvert\Delta\rvert$
1	0.0312500000
2	0.0078125000
130	0.0019531250
642	0.0004882812
2690	0.0001220703

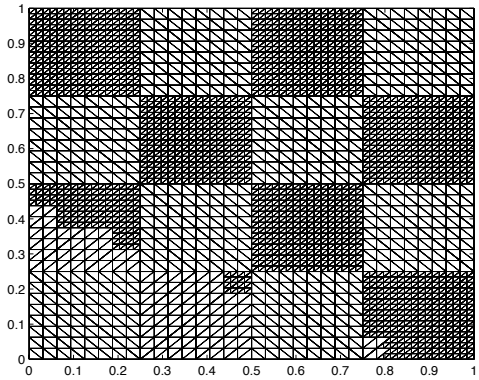
Table 5.7: $\max_{\Delta \in \mathscr{T}_n} |\Delta|$ at Iteration n

Function	Points	Convergence History	Final Error Ratio
Oscillatory	Figure 5.3	Table 5.9	0.990508
Internal Peak	Figure 5.4	Table 5.10	1.000667
Corner Peak	Figure 5.5	Table 5.11	0.999899
Gaussian	Figure 5.6	Table 5.12	1.000264
C^0 Function	Figure 5.7	Table 5.13	0.976029

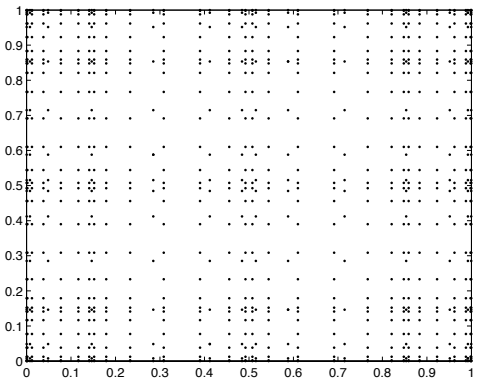
Table 5.8: Summary of 2D Experiments



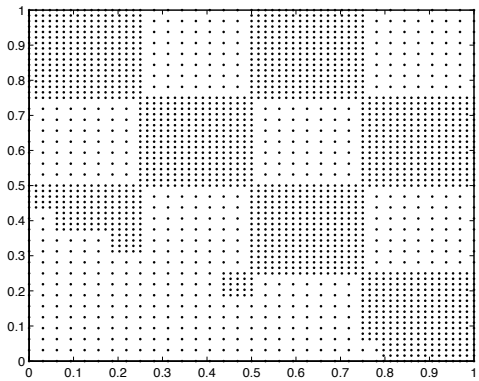
(a) $f(\mathbf{x}) = \cos(2\pi\beta + \sum_{i=1}^{d} \alpha_i x_i))$



(b) Final Mesh



(c) Matlab Points

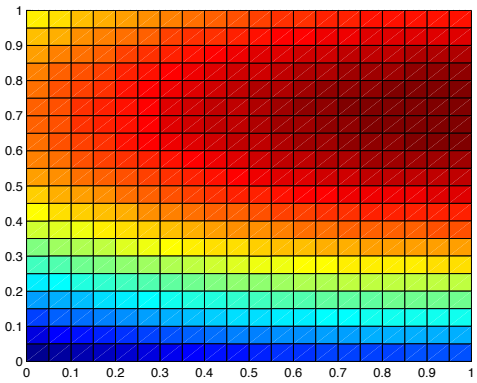


(d) MAQS Points

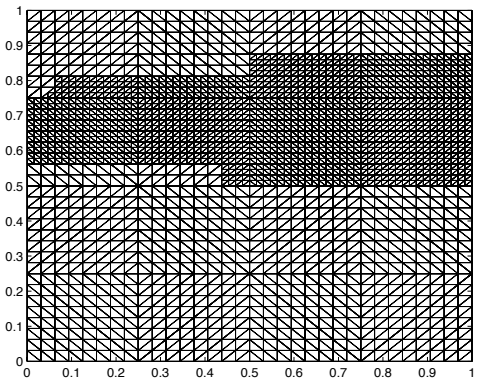
Figure 5.3: Oscillatory Family

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.2098584052	0.0000733401	0.0000737964	0.9938177157
19	0.2098779244	0.0000536157	0.0000542771	0.9878127601
37	0.2098902739	0.0000413919	0.0000419277	0.9872223933
55	0.2098996946	0.0000320686	0.0000325070	0.9865136662
73	0.2099060223	0.0000256463	0.0000261793	0.9796392129
91	0.2099097468	0.0000216963	0.0000224530	0.9662966796
109	0.2099114566	0.0000194285	0.0000200008	0.9713872537
127	0.2099129489	0.0000179804	0.0000185085	0.9714662437
145	0.2099142759	0.0000166537	0.0000171815	0.9692790167
163	0.2099156014	0.0000154578	0.0000158591	0.9746941604
181	0.2099165847	0.0000143059	0.0000147397	0.9705682182
199	0.2099176658	0.0000134054	0.0000135060	0.9925517535
217	0.2099185362	0.0000125329	0.0000126357	0.9918596334
235	0.2099193422	0.0000117336	0.0000118297	0.9918828574
253	0.2099200626	0.0000110038	0.0000111092	0.9905087283

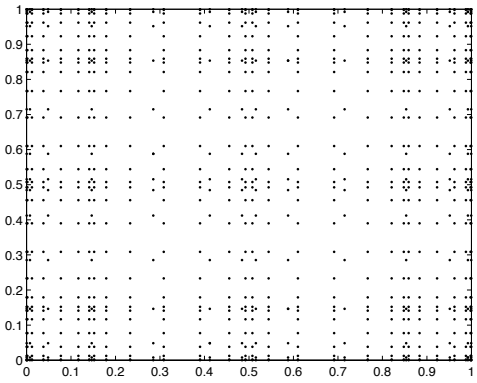
Table 5.9: Oscillatory 2D: Error Estimation



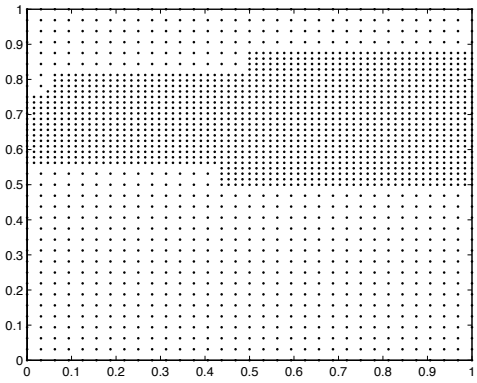
(a) $f(\mathbf{x}) = \prod_{i=1}^{d} (\alpha_i^{-2} + (x_i - \beta_i)^2)^{-1}$



(b) Final Mesh



(c) Matlab Points

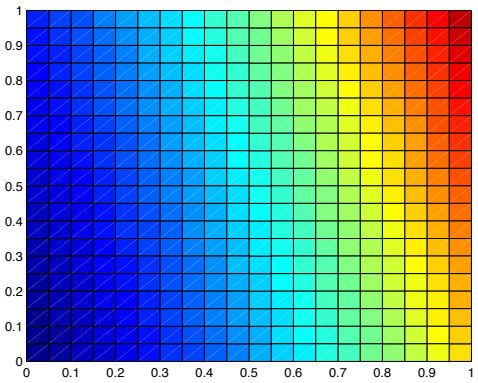


(d) MAQS Points

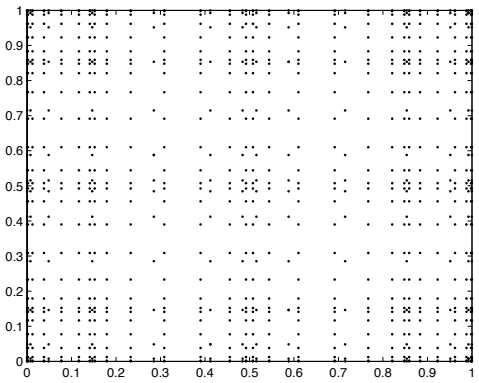
Figure 5.4: Internal Peak

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.0442807269	0.0000585699	0.0000583827	1.0032058250
21	0.0442894079	0.0000498464	0.0000497017	1.0029112643
41	0.0442973860	0.0000418095	0.0000417236	1.0020592345
61	0.0443049541	0.0000342947	0.0000341555	1.0040781715
81	0.0443118141	0.0000273187	0.0000272955	1.0008503569
101	0.0443180759	0.0000211405	0.0000210337	1.0050773806
121	0.0443227966	0.0000163229	0.0000163130	1.0006068622
141	0.0443248571	0.0000142633	0.0000142525	1.0007543628
161	0.0443254080	0.0000137118	0.0000137016	1.0007415260
181	0.0443259489	0.0000131697	0.0000131607	1.0006843034
201	0.0443264832	0.0000126383	0.0000126264	1.0009402919
221	0.0443270079	0.0000121106	0.0000121017	1.0007367475
241	0.0443275283	0.0000115883	0.0000115813	1.0006060236
261	0.0443280430	0.0000110744	0.0000110666	1.0007027353
281	0.0443285496	0.0000105671	0.0000105600	1.0006671146

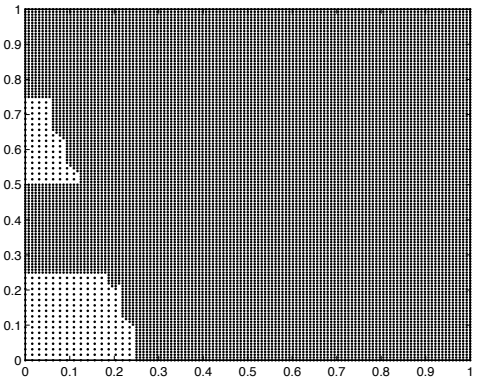
Table 5.10: Internal Peak 2D: Error Estimation



(a) $f(\mathbf{x}) = (1 + \sum_{i=1}^{d} \alpha_i x_i)^{(-d+1)}$



(b) Matlab Points

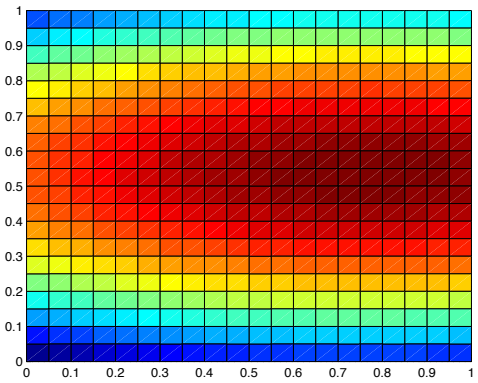


(c) MAQS Points

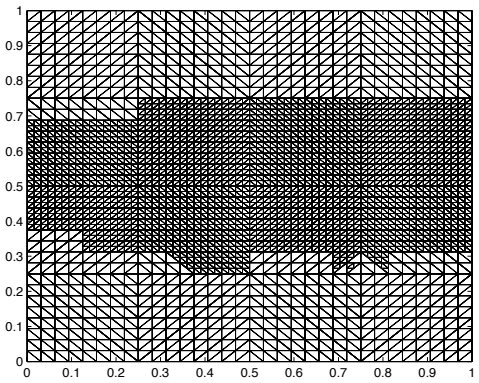
Figure 5.5: Corner Peak

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	4.1680234985	0.0086223986	0.0086223986	1.0000000000
27	4.1658905418	0.0064831499	0.0064894419	0.9990304226
53	4.1640995039	0.0046949483	0.0046984040	0.9992645024
79	4.1629026987	0.0035020105	0.0035015987	1.0001175867
105	4.1621287411	0.0027266789	0.0027276412	0.9996472220
131	4.1615455036	0.0021442183	0.0021444037	0.9999135337
157	4.1614023202	0.0020011122	0.0020012202	0.9999460382
183	4.1612654817	0.0018641963	0.0018643817	0.9999005469
209	4.1611350480	0.0017339480	0.0017339480	1.0000000000
235	4.1610110839	0.0016100726	0.0016099840	1.0000550540
261	4.1608919487	0.0014907408	0.0014908488	0.9999275651
287	4.1607773223	0.0013762998	0.0013762223	1.0000562620
313	4.1606681070	0.0012670071	0.0012670071	1.0000000000
339	4.1605657703	0.0011644850	0.0011646704	0.9998407974
365	4.1604693676	0.0010681597	0.0010682677	0.9998989115

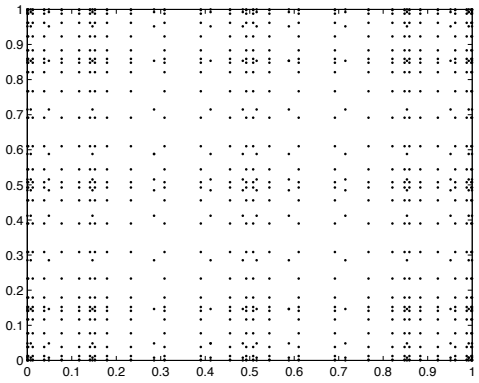
Table 5.11: Corner Peak 2D: Error Estimation



(a) $f(\mathbf{x}) = \exp(-\sum_{i=1}^{d}(\alpha_i^2(x_i - \beta_i)^2))$



(b) Final Mesh

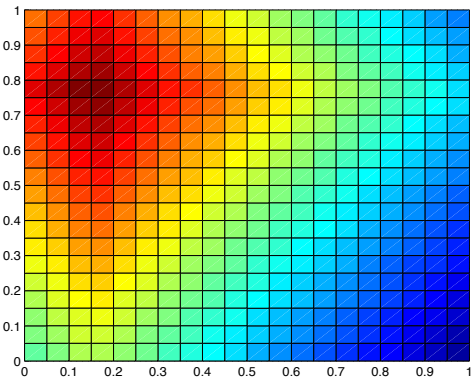


(c) Matlab Points

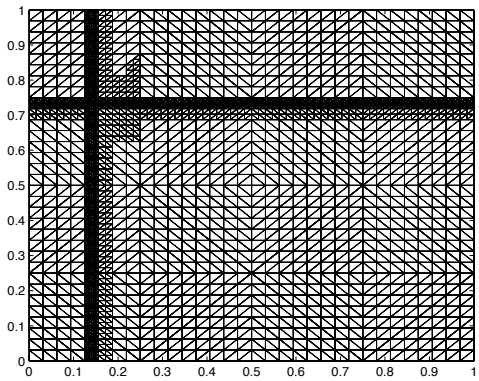
Figure 5.6: Gaussian Function

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.9284983453	0.0013762990	0.0013737154	1.0018807318
27	0.9287371837	0.0011376692	0.0011348770	1.0024603731
53	0.9289613997	0.0009117011	0.0009106610	1.0011421515
79	0.9291756379	0.0006969870	0.0006964227	1.0008102805
105	0.9293714971	0.0005009940	0.0005005636	1.0008599083
131	0.9295299285	0.0003422914	0.0003421322	1.0004653284
157	0.9295451539	0.0003270677	0.0003269068	1.0004922228
183	0.9295601520	0.0003120325	0.0003119086	1.0003971398
209	0.9295750054	0.0002971875	0.0002970552	1.0004453466
235	0.9295896919	0.0002825005	0.0002823688	1.0004663594
261	0.9296041890	0.0002679625	0.0002678717	1.0003389465
287	0.9296185652	0.0002536492	0.0002534954	1.0006068452
313	0.9296326536	0.0002395000	0.0002394070	1.0003882479
339	0.9296465836	0.0002256250	0.0002254770	1.0006560538
365	0.9296601812	0.0002119353	0.0002118794	1.0002639184

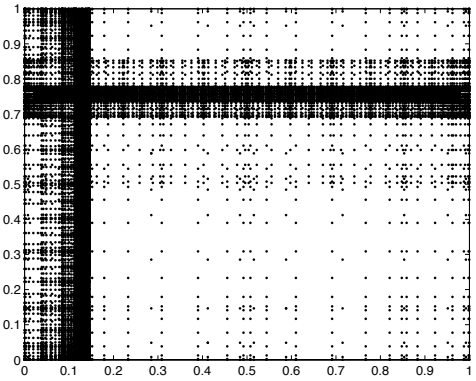
Table 5.12: Gaussian 2D: Error Estimation



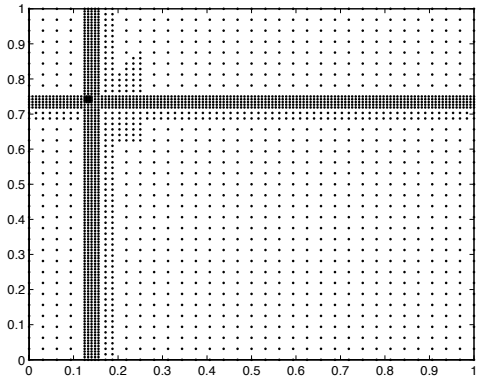
(a) $f(\mathbf{x}) = \exp(-\sum_{i=1}^{d} \alpha_i |x_i - \beta_i|)$



(b) Final Mesh



(c) Matlab Points: 20,070



(d) MAQS Points: 1,850

Figure 5.7: $C^0(\Omega)$ Function

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.7275120657	0.0026390158	0.0033537018	0.7868963677
27	0.7288787497	0.0013046245	0.0019144762	0.6814524511
53	0.7294631751	0.0009546706	0.0012935324	0.7380338044
79	0.7296313827	0.0007905514	0.0011204686	0.7055542366
105	0.7298046201	0.0006603136	0.0009164031	0.7205492991
131	0.7299750551	0.0005934926	0.0006121900	0.9694581735
157	0.7298998596	0.0004936310	0.0005053606	0.9767896670
183	0.7299284682	0.0004294684	0.0004452131	0.9646356753
209	0.7299321534	0.0003708808	0.0003930201	0.9436688479
235	0.7299333329	0.0003216948	0.0003342790	0.9623543995
261	0.7299232708	0.0002944636	0.0003009372	0.9784884540
287	0.7299148189	0.0002860141	0.0002924853	0.9778751306
313	0.7299070437	0.0002782532	0.0002847101	0.9773210273
339	0.7298995641	0.0002707396	0.0002772305	0.9765866907
365	0.7298922513	0.0002634475	0.0002699178	0.9760289011

Table 5.13: C^0 Function 2D: Error Estimation

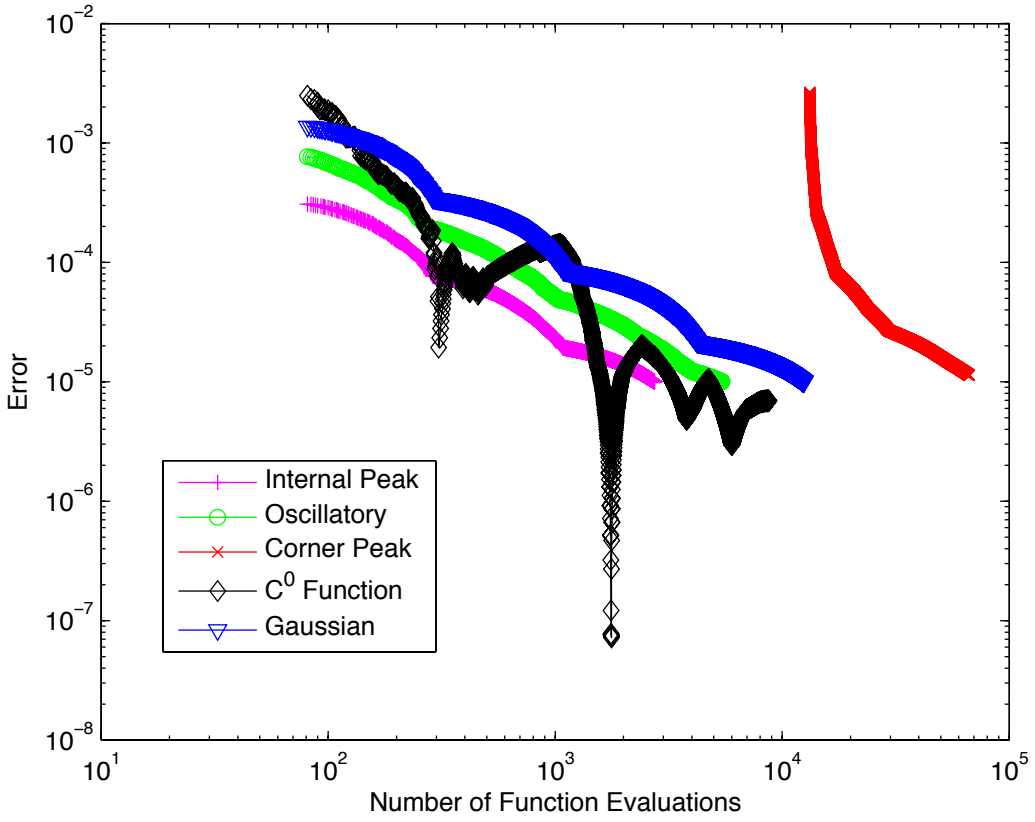


Figure 5.8: Genz-Malik Family 2D: Error v. Function Evaluations

5.4.2 Genz-Malik d-Dimensional Study

The Figures 5.9 and 5.10 covers the Genz-Malik suite in multi-dimensions. As with the two dimensional cases, MAQS does converge but is not best suited for such families of functions. One can vary the heuristic value H_Δ by decreasing the value of a when dealing with "smooth" functions such as those in this test suite. However, if the function being integrated are know to be smooth, one should choose a more appropriate algorithm.

After this we provide a table that depicts the number of iterations n the values of the integral estimate Q_Ω, the error estimate, E_Ω, the actual error, and the ratio of the estimated error and actual error to numerically validate our choice for E_Ω.

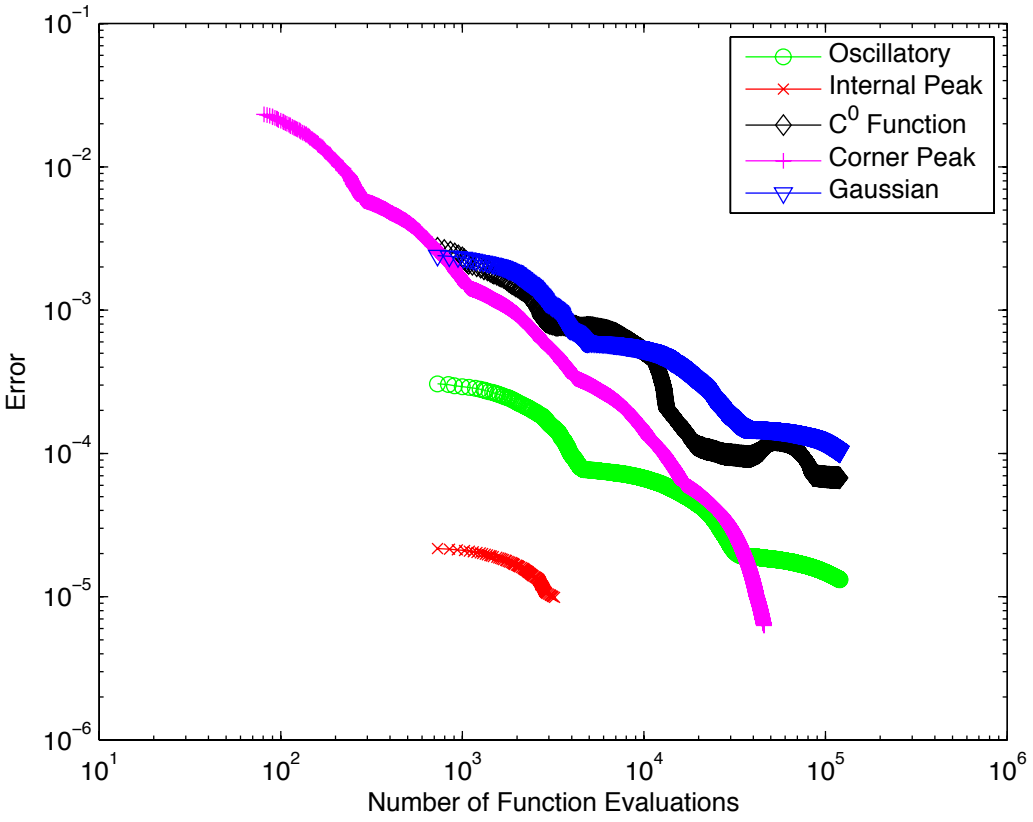


Figure 5.9: Genz-Malik Family 3D: Error v. Function Evaluations

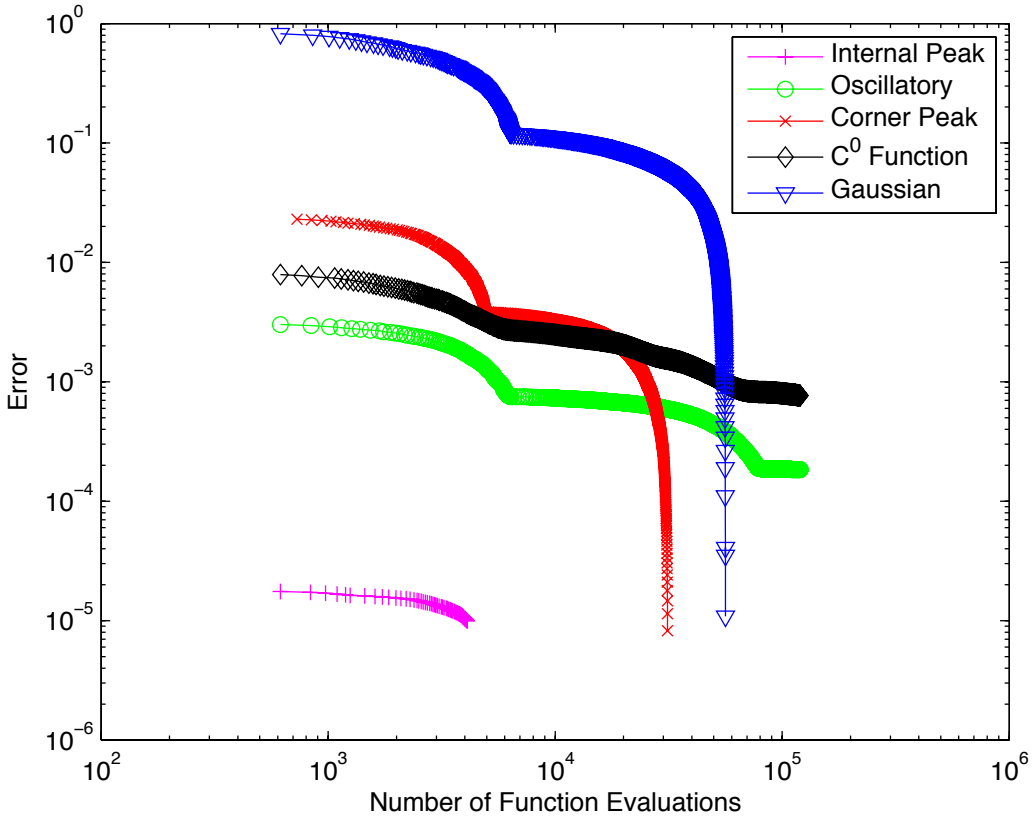


Figure 5.10: Genz-Malik Family 4D: Error v. Function Evaluations

Function	Convergence History	Final Error Ratio
Oscillatory	Table 5.18	0.9936791738
Corner Peak	Table 5.22	0.9973162180
Gaussian	Table 5.26	1.0178939714
C^0 Function	Table 5.30	0.3875167354

Table 5.14: Summary of 3D Experiments

Function	Convergence History	Final Error Ratio
Oscillatory	Table 5.19	0.9794227153
Corner Peak	Table 5.23	0.9885974140
Gaussian	Table 5.27	0.9987269904
C^0 Function	Table 5.31	0.9376326542

Table 5.15: Summary of 4D Experiments

Function	Convergence History	Final Error Ratio
Oscillatory	Table 5.20	0.9101316896
Corner Peak	Table 5.24	0.9172751691
Gaussian	Table 5.28	1.0088018636
C^0 Function	Table 5.32	0.7129455118

Table 5.16: Summary of 5D Experiments

Function	Convergence History	Final Error Ratio
Oscillatory	Table 5.21	0.8928452408
Corner Peak	Table 5.25	1.0444862590
Gaussian	Table 5.29	0.9059946320
C^0 Function	Table 5.33	0.5340226662

Table 5.17: Summary of 6D Experiments

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.3664651582	0.0052821063	0.0052737520	1.0015841116
8	0.3675647394	0.0044142497	0.0041746737	1.0573879571
15	0.3680530359	0.0035681904	0.0036903680	0.9668928431
22	0.3687315330	0.0028537546	0.0030129975	0.9471479824
29	0.3693796418	0.0022526771	0.0023652481	0.9524062522
36	0.3698717154	0.0017389451	0.0018735769	0.9281418320
43	0.3700953965	0.0013901935	0.0016499193	0.8425827425
50	0.3704051610	0.0013356617	0.0013401824	0.9966267797
57	0.3704485045	0.0012882268	0.0012968413	0.9933573464
64	0.3704973499	0.0012432505	0.0012479984	0.9961955551
71	0.3705410933	0.0012009397	0.0012042573	0.9972451697
78	0.3705837121	0.0011619410	0.0011616411	1.0002581981
85	0.3706191724	0.0011218656	0.0011261828	0.9961665090
92	0.3706557918	0.0010845483	0.0010895656	0.9953951345
99	0.3706902913	0.0010483991	0.0010550680	0.9936791738

Table 5.18: Oscillatory 3D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.5800862408	0.0120416265	0.0133521419	0.9018497960
27	0.5819301420	0.0104711122	0.0115087333	0.9098405489
53	0.5833448280	0.0093662251	0.0100943888	0.9278645139
79	0.5843891037	0.0084429504	0.0090503742	0.9328841288
105	0.5853333672	0.0075938879	0.0081063221	0.9367858691
131	0.5861530046	0.0068390239	0.0072868502	0.9385432228
157	0.5868116035	0.0061451133	0.0066283646	0.9270934410
183	0.5874836153	0.0055149176	0.0059564695	0.9258701925
209	0.5880852041	0.0049435869	0.0053549865	0.9231744961
235	0.5885619178	0.0044402562	0.0048784668	0.9101745332
261	0.5890425303	0.0040373472	0.0043979245	0.9180119473
287	0.5894637282	0.0036868027	0.0039767803	0.9270823189
313	0.5897961686	0.0034072363	0.0036443685	0.9349318782
339	0.5900734224	0.0032302835	0.0033671376	0.9593559531
365	0.5902118813	0.0031622502	0.0032286879	0.9794227153

Table 5.19: Oscillatory 4D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.6809373764	0.0152080061	0.0163460047	0.9303806270
27	0.6813596267	0.0147421550	0.0159238624	0.9257901528
53	0.6816370578	0.0144472278	0.0156464812	0.9233531523
79	0.6818767097	0.0141862553	0.0154068763	0.9207742721
105	0.6821038698	0.0139536621	0.0151797586	0.9192281948
131	0.6823229929	0.0137292149	0.0149606713	0.9176870900
157	0.6825355926	0.0135134215	0.0147481071	0.9162817568
183	0.6827453621	0.0133049812	0.0145383780	0.9151626952
209	0.6829394808	0.0131114447	0.0143442910	0.9140531732
235	0.6831328161	0.0129225472	0.0141509909	0.9131902680
261	0.6833180006	0.0127428074	0.0139658413	0.9124267649
287	0.6834948765	0.0125704172	0.0137889971	0.9116266426
313	0.6836662206	0.0124060923	0.0136176851	0.9110279907
339	0.6838318547	0.0122468879	0.0134520815	0.9104083896
365	0.6840022999	0.0120880685	0.0132816698	0.9101316896

Table 5.20: Oscillatory 5D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.8671752433	0.0369909541	0.0413720926	0.8941040154
27	0.8671885281	0.0369744369	0.0413588104	0.8939917907
53	0.8672005926	0.0369592889	0.0413467483	0.8938862278
79	0.8672121079	0.0369446615	0.0413352353	0.8937813275
105	0.8672234097	0.0369307693	0.0413239357	0.8936895444
131	0.8672348183	0.0369172555	0.0413125294	0.8936091797
157	0.8672453834	0.0369038738	0.0413019664	0.8935137237
183	0.8672558699	0.0368910193	0.0412914820	0.8934292869
209	0.8672660607	0.0368783195	0.0412812932	0.8933421562
235	0.8672760312	0.0368658653	0.0412713246	0.8932561697
261	0.8672858906	0.0368535251	0.0412614670	0.8931704990
287	0.8672958490	0.0368413036	0.0412515106	0.8930898071
313	0.8673054695	0.0368291806	0.0412418919	0.8930041497
339	0.8673148820	0.0368174478	0.0412324811	0.8929234154
365	0.8673242187	0.0368058898	0.0412231461	0.8928452408

Table 5.21: Oscillatory 6D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	6.6885389784	0.3642231097	0.3643066959	0.9997705610
27	6.4880562536	0.1634253419	0.1637891939	0.9977785345
53	6.4177939585	0.0934408598	0.0935202418	0.9991511776
79	6.4077330537	0.0834729755	0.0834590175	1.0001672436
105	6.3994483166	0.0751738303	0.0751740054	0.9999976718
131	6.3918727618	0.0678980352	0.0675982405	1.0044349488
157	6.3853706876	0.0614244799	0.0610958506	1.0053789142
183	6.3799895453	0.0558843295	0.0557144067	1.0030498895
209	6.3751175664	0.0509257656	0.0508421506	1.0016445999
235	6.3707687735	0.0464596509	0.0464931951	0.9992785146
261	6.3666014522	0.0421530069	0.0423257279	0.9959192449
287	6.3628050776	0.0384406114	0.0385292135	0.9977003917
313	6.3592236019	0.0348659483	0.0349475699	0.9976644550
339	6.3559480858	0.0315987565	0.0316719777	0.9976881370
365	6.3532624396	0.0289084855	0.0289862783	0.9973162180

Table 5.22: Corner Peak 3D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	35.1825351932	3.4306309924	3.6203712071	0.9475909502
27	34.5787224536	2.8638292747	3.0163323122	0.9494409032
53	34.1547231803	2.4919863906	2.5921526989	0.9613578674
79	33.8590718932	2.2120055872	2.2963743086	0.9632600308
105	33.6077239318	1.9783752512	2.0449456712	0.9674463625
131	33.3852995995	1.7773157329	1.8224325418	0.9752436330
157	33.1978473759	1.5924245883	1.6349124715	0.9740121359
183	33.0456764187	1.4457285224	1.4827096484	0.9750584169
209	32.9076308250	1.3107433611	1.3446209435	0.9748051058
235	32.7913512980	1.1937042552	1.2283046751	0.9718307513
261	32.6993728480	1.1075809734	1.1363032944	0.9747230153
287	32.6214563168	1.0321134613	1.0583739639	0.9751878792
313	32.5588661704	0.9740455249	0.9957691193	0.9781841051
339	32.5018784804	0.9260859936	0.9387677893	0.9864910196
365	32.4775442197	0.9040030870	0.9144299532	0.9885974140

Table 5.23: Corner Peak 4D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	176.3572379285	25.3439602759	27.8228951531	0.9109030579
27	175.6775666151	24.6719392532	27.1427091789	0.9089711381
53	175.0873603447	24.1390622098	26.5521240261	0.9091198198
79	174.5529719351	23.6478376767	26.0173763981	0.9089247630
105	174.0567488781	23.2242266691	25.5208740516	0.9100090625
131	173.6031483999	22.8100671075	25.0670028569	0.9099638771
157	173.1415262031	22.4244421000	24.6051279522	0.9113727083
183	172.7011654988	22.0537020257	24.1645413511	0.9126472423
209	172.2650302866	21.7041188478	23.7282196432	0.9146964742
235	171.8611353440	21.3673781363	23.3241360036	0.9161058799
261	171.4811249074	21.0454660038	22.9439669255	0.9172548963
287	171.1344457343	20.7321719611	22.5971119078	0.9174699867
313	170.7931851113	20.4228689495	22.2556973557	0.9176467771
339	170.4749497772	20.1257821793	21.9373349343	0.9174214753
365	170.1703575666	19.8430531091	21.6326068534	0.9172751691

Table 5.24: Corner Peak 5D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	3864.7621622647	1059.4117647932	1005.7362827024	1.0533693405
27	3864.5718833503	1058.2155191711	1005.5457427707	1.0523792943
53	3864.4125581706	1057.2494796262	1005.3862200846	1.0515854092
79	3864.2748756999	1056.3512485759	1005.2483498888	1.0508360931
105	3864.1485495890	1055.5037003743	1005.1218588643	1.0501251078
131	3864.0181725776	1054.6930924846	1004.9913242676	1.0494549227
157	3863.8773331662	1053.9160458911	1004.8503150024	1.0488288954
183	3863.7519528162	1053.1584376018	1004.7247882084	1.0482058868
209	3863.6209902055	1052.4186064802	1004.5936903855	1.0476062278
235	3863.5037824832	1051.6953188623	1004.4763441502	1.0470085483
261	3863.3924981464	1050.9893476780	1004.3649173432	1.0464218030
287	3863.2807639013	1050.3068082017	1004.2530342668	1.0458587352
313	3863.1540049292	1049.6454828161	1004.1261273251	1.0453323086
339	3862.9693181026	1049.0286372423	1003.9412922727	1.0449103402
365	3862.7980785591	1048.4238570386	1003.7698897524	1.0444862590

Table 5.25: Corner Peak 6D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.7930225220	0.0249543923	0.0251281575	0.9930848390
27	0.8053069604	0.0137731988	0.0128552404	1.0714073274
53	0.8118947831	0.0062772031	0.0062704394	1.0010786708
79	0.8124068996	0.0057723300	0.0057584478	1.0024107520
105	0.8128791225	0.0053031246	0.0052863447	1.0031741807
131	0.8133213544	0.0048548644	0.0048442203	1.0021972823
157	0.8137400861	0.0044423891	0.0044255868	1.0037966210
183	0.8141393488	0.0040485448	0.0040264134	1.0054965431
209	0.8145184219	0.0036709816	0.0036474207	1.0064596127
235	0.8148623679	0.0033170920	0.0033035439	1.0041010645
261	0.8151974202	0.0030203223	0.0029685622	1.0174360793
287	0.8154668062	0.0027539991	0.0026992184	1.0202950173
313	0.8157170750	0.0025078815	0.0024489822	1.0240505128
339	0.8159396659	0.0022746575	0.0022264158	1.0216678622
365	0.8161424076	0.0020599036	0.0020236917	1.0178939714

Table 5.26: Gaussian 3D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.8076751472	0.0260371788	0.0265903564	0.9791963075
27	0.8111732966	0.0230863719	0.0230941192	0.9996645363
53	0.8132666168	0.0211618279	0.0210019136	1.0076142713
79	0.8148568945	0.0195636453	0.0194124575	1.0077881815
105	0.8164038614	0.0180697985	0.0178662740	1.0113915482
131	0.8178920046	0.0166268982	0.0163788513	1.0151443403
157	0.8192962591	0.0152478472	0.0149752883	1.0182005777
183	0.8206068686	0.0138863871	0.0136652948	1.0161791094
209	0.8219257474	0.0126355613	0.0123470676	1.0233653668
235	0.8231287817	0.0114389181	0.0111445806	1.0264108138
261	0.8242654699	0.0102750120	0.0100083658	1.0266423321
287	0.8253737176	0.0091672332	0.0089005955	1.0299572813
313	0.8263507907	0.0081607111	0.0079239155	1.0298836567
339	0.8271717081	0.0072343385	0.0071033370	1.0184422549
365	0.8275329553	0.0067337140	0.0067422971	0.9987269904

Table 5.27: Gaussian 4D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.8355196811	0.0276659670	0.0283103099	0.9772399923
27	0.8364752037	0.0268380811	0.0273555128	0.9810849201
53	0.8370905569	0.0263150091	0.0267404733	0.9840891306
79	0.8376548510	0.0258684420	0.0261764396	0.9882337837
105	0.8382037064	0.0254223220	0.0256278951	0.9919785386
131	0.8386768950	0.0250135696	0.0251549778	0.9943785196
157	0.8390949756	0.0246352381	0.0247370725	0.9958833274
183	0.8394962327	0.0242933745	0.0243359878	0.9982489606
209	0.8398531606	0.0239673056	0.0239791919	0.9995043057
235	0.8401921962	0.0236566384	0.0236402884	1.0006916151
261	0.8405444754	0.0233612641	0.0232881372	1.0031400930
287	0.8408543890	0.0230659335	0.0229783275	1.0038125507
313	0.8411519782	0.0227827568	0.0226808376	1.0044936259
339	0.8414580029	0.0225239362	0.0223749123	1.0066603135
365	0.8417539530	0.0222733977	0.0220790608	1.0088018636

Table 5.28: Gaussian 5D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.6216083215	0.0469616602	0.0519085166	0.9047004867
27	0.6216279283	0.0469491825	0.0518889210	0.9048016730
53	0.6216460742	0.0469375215	0.0518707854	0.9048932095
79	0.6216636409	0.0469263569	0.0518532291	0.9049842756
105	0.6216810021	0.0469157334	0.0518358780	0.9050822562
131	0.6216976346	0.0469051749	0.0518192549	0.9051688415
157	0.6217148826	0.0468948963	0.0518020167	0.9052716337
183	0.6217309658	0.0468848643	0.0517859427	0.9053589040
209	0.6217471629	0.0468746888	0.0517697545	0.9054454522
235	0.6217633989	0.0468649258	0.0517535279	0.9055406986
261	0.6217792279	0.0468553677	0.0517377083	0.9056328399
287	0.6217952263	0.0468461442	0.0517217192	0.9057344762
313	0.6218109506	0.0468368425	0.0517060036	0.9058298693
339	0.6218259480	0.0468276570	0.0516910143	0.9059148416
365	0.6218408578	0.0468182809	0.0516761129	0.9059946320

Table 5.29: Gaussian 6D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.5363312133	0.0181714254	0.0336736826	0.5396328530
27	0.5565004389	0.0118086001	0.0147600107	0.8000400811
53	0.5614161740	0.0086344439	0.0082782870	1.0430230253
79	0.5618374871	0.0073301269	0.0072589945	1.0097992062
105	0.5621323422	0.0061770118	0.0067975171	0.9087158821
131	0.5623522096	0.0052166234	0.0064235560	0.8121083502
157	0.5627876721	0.0044835232	0.0059319495	0.7558262583
183	0.5630353312	0.0038220194	0.0055911529	0.6835834104
209	0.5633293252	0.0032678445	0.0052233881	0.6256177838
235	0.5635067458	0.0027884231	0.0049478191	0.5635661006
261	0.5636714628	0.0023925948	0.0046539696	0.5140976358
287	0.5636567786	0.0020845565	0.0044755726	0.4657630805
313	0.5636410532	0.0018672484	0.0042893271	0.4353243308
339	0.5635626528	0.0016919603	0.0041612303	0.4066009836
365	0.5634429153	0.0015698930	0.0040511618	0.3875167354

Table 5.30: C^0 Function 3D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.4746678425	0.0291139595	0.0370009576	0.7868434074
27	0.4795184899	0.0258328465	0.0322639451	0.8006722803
53	0.4834361095	0.0238324615	0.0285238285	0.8355281463
79	0.4867260227	0.0223357305	0.0253055911	0.8826401426
105	0.4891173012	0.0207204289	0.0230636649	0.8984014014
131	0.4914236110	0.0193390357	0.0208186826	0.9289269663
157	0.4930686559	0.0178175657	0.0192388200	0.9261257031
183	0.4946383700	0.0164208479	0.0176997005	0.9277472140
209	0.4963799936	0.0151133543	0.0160982967	0.9388169824
235	0.4980711264	0.0139798230	0.0144987053	0.9642118210
261	0.4996772558	0.0129063130	0.0129387795	0.9974907603
287	0.5009571689	0.0118677830	0.0117023491	1.0141368055
313	0.5019068138	0.0109171902	0.0107295740	1.0174858934
339	0.5021422079	0.0099591810	0.0102757969	0.9691881947
365	0.5022362335	0.0094459986	0.0100743063	0.9376326542

Table 5.31: C^0 Function 4D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.4167123440	0.0288002151	0.0415202143	0.6936432175
27	0.4176569157	0.0281760888	0.0405848458	0.6942514667
53	0.4184193047	0.0277536818	0.0398238563	0.6969109575
79	0.4191509683	0.0273922012	0.0390799723	0.7009268320
105	0.4198360230	0.0270508798	0.0383958350	0.7045264098
131	0.4204798302	0.0267223597	0.0377533098	0.7078150174
157	0.4210719338	0.0264315644	0.0371630440	0.7112324919
183	0.4215713980	0.0261200890	0.0366743889	0.7122160667
209	0.4220214641	0.0258126851	0.0362336262	0.7123958556
235	0.4224047783	0.0255158844	0.0358508263	0.7117237455
261	0.4227891096	0.0252608119	0.0354716273	0.7121413326
287	0.4231742614	0.0250304298	0.0350873660	0.7133744324
313	0.4234285706	0.0247636134	0.0348466388	0.7106456824
339	0.4237686277	0.0245515485	0.0345140087	0.7113502439
365	0.4241319440	0.0243516743	0.0341564311	0.7129455118

Table 5.32: C^0 Function 5D: Error Estimation

n	Q_Ω	$\|L_1 f - L_2 f\|$	$\|L_1 - f\|$	$\frac{\|L_1 f - L_2 f\|}{\|L_1 - f\|}$
1	0.2774238666	0.0210624216	0.0400232963	0.5262540454
27	0.2775647670	0.0210083896	0.0398815056	0.5267702236
53	0.2776895312	0.0209654898	0.0397553358	0.5273629173
79	0.2777962266	0.0209242943	0.0396489722	0.5277386311
105	0.2779063739	0.0208869523	0.0395395237	0.5282550302
131	0.2780078316	0.0208489035	0.0394382590	0.5286466498
157	0.2781098495	0.0208158357	0.0393357266	0.5291839640
183	0.2782240007	0.0207861374	0.0392203610	0.5299833276
209	0.2783278501	0.0207553502	0.0391158388	0.5306124280
235	0.2784333811	0.0207259530	0.0390093818	0.5313068824
261	0.2785370795	0.0206958039	0.0389044671	0.5319647187
287	0.2786316995	0.0206654561	0.0388095125	0.5324843002
313	0.2787227526	0.0206347084	0.0387179389	0.5329495592
339	0.2788209433	0.0206062320	0.0386197595	0.5335670721
365	0.2789081310	0.0205777415	0.0385334609	0.5340226662

Table 5.33: C^0 Function 6D: Error Estimation

5.5 Test Suite II: Non-smooth Functions

In this section we test MAQS against three different function that exhibit either some from of discontinuity or singularity like behavior. We want to emphasize that for this suite of functions, many common quadrature rules converge slowly or fail to converge completely.

5.5.1 Zero-One Circle Function

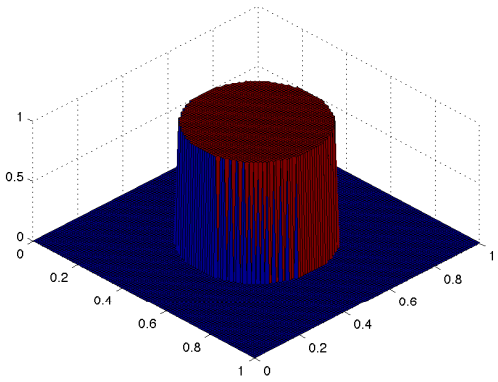
This function simply takes the value of one inside a circle and a value of zero outside. This function and its multidimensional analog is challenging for polynomials to approximate. However, since MAQS uses piecewise linear approximation, it is well suited to adapt appropriately and approximate this function. In Figure 5.11 we see how many fewer points MAQS used than `quad2d` while coming up with a better approximation to the volume.

5.5.2 Line Singularity

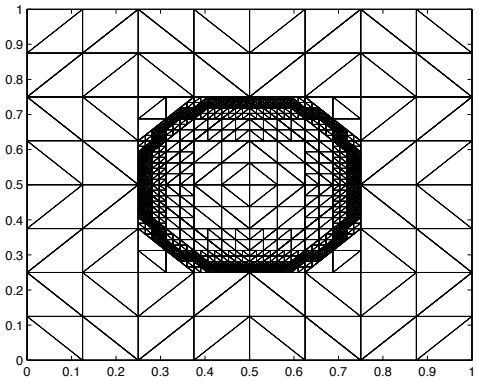
This example originated in [53]. The method used therein is a piecewise linear approximation, like MAQS, but based on sparse grid interpolation which uses hypercubes as a basic geometry. The function of interest has a line singularity that is not along the grid lines, see Figure 5.12.

5.5.3 Shock Problem

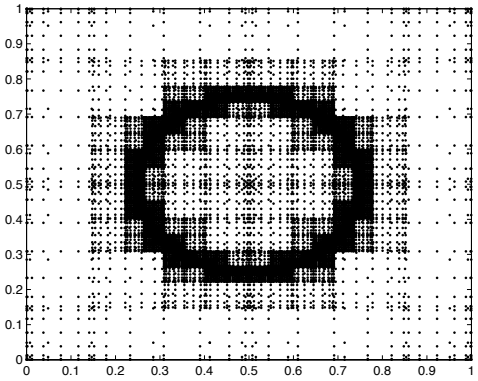
This example is based on a quasi-one dimensional duct flow problem in [16]. The premise is that we want to calculate the average velocity of air flow over a wing in a tunnel. Due to the shape of the wing, there is a shock (discontinuity) where the air speed changes from supersonic to subsonic. The exact location of this shock is not known *a priori*, without solving a nonlinear system of equations. Therefore, for



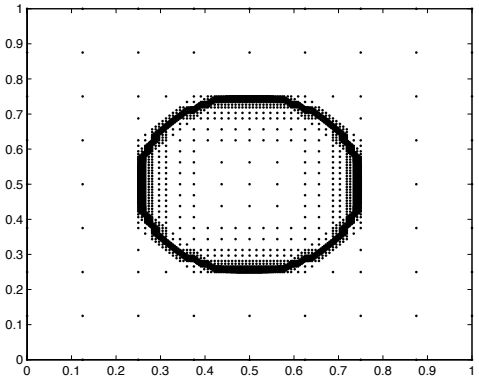
(a) Zero-One Function



(b) Final Mesh

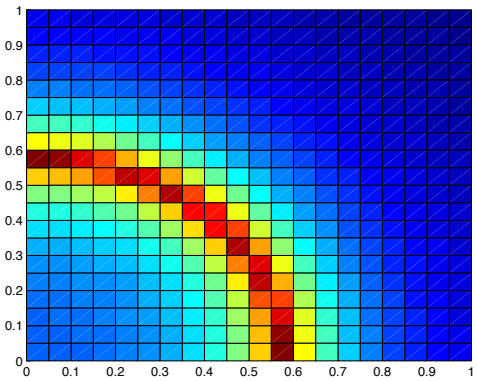


(c) Matlab Points: 391,810

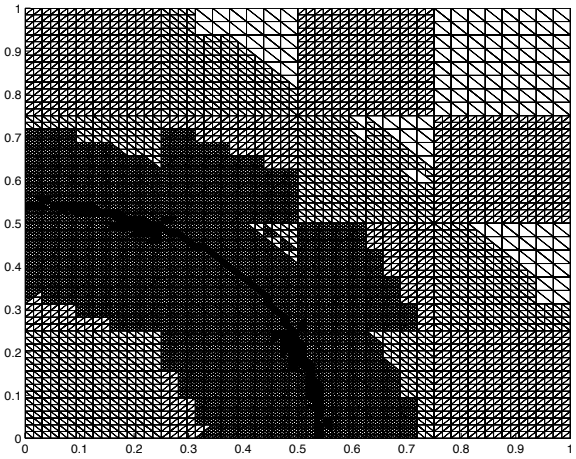


(d) MAQS Points: 19,166

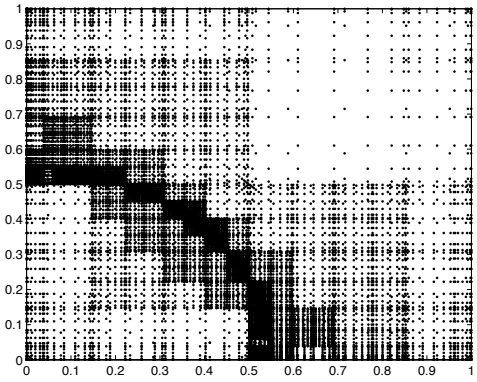
Figure 5.11: Zero-One Function



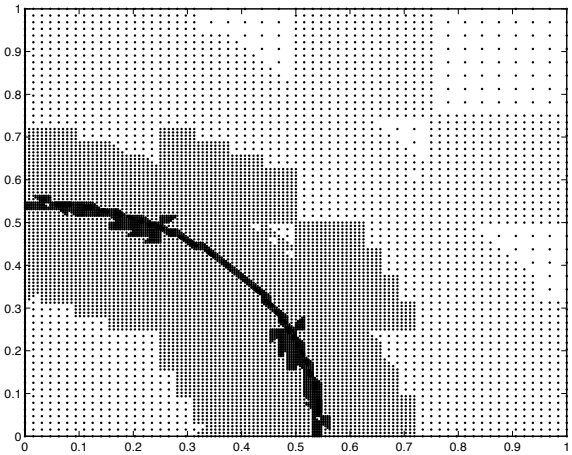
(a) $f(\mathbf{x}) = 1/(|0.3 - (\sum_{i=1}^{d} x_i^2)| + \delta))$



(b) Final Mesh



(c) Matlab Points



(d) MAQS Points

Figure 5.12: Line Singularity

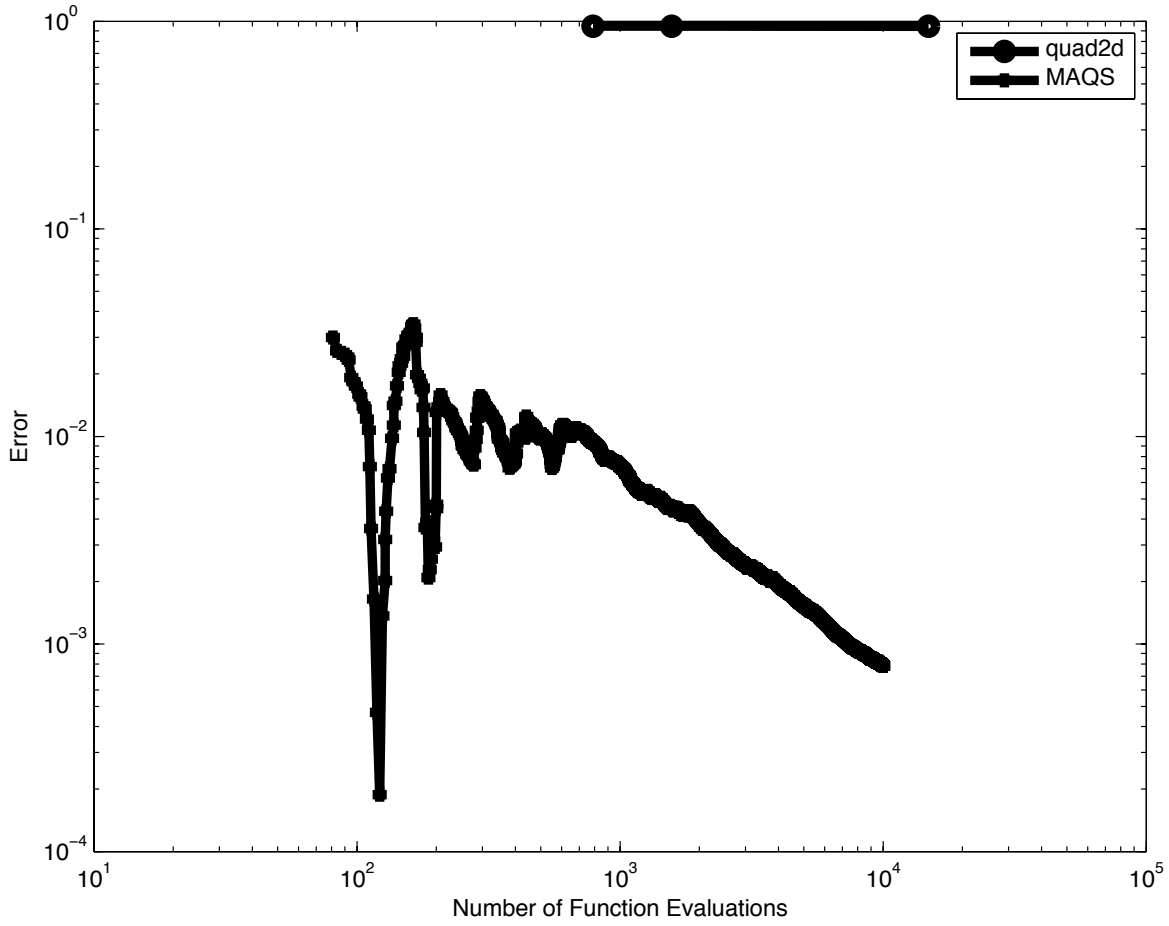
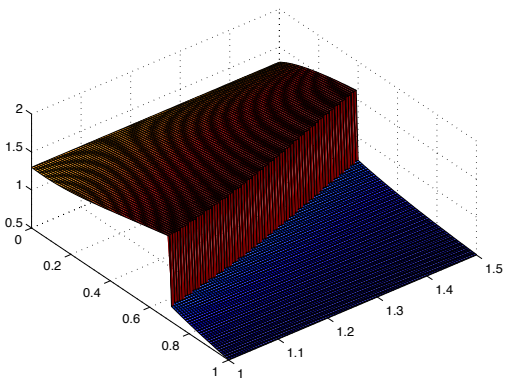


Figure 5.13: Line Singularity MAQS v. `quad2d`: Error v. Function Evaluations

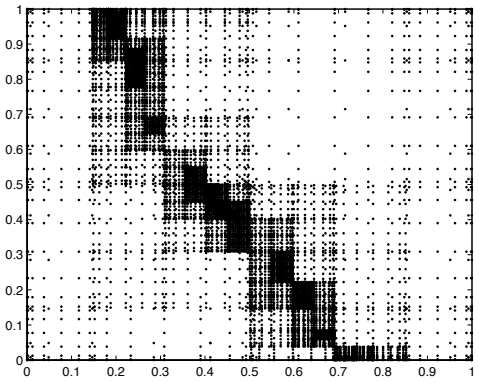
each function evaluation in our quadrature scheme, we must solve this system of equations. For this given (one dimensional) problem this is not entirely complicated or time consuming with today's computing power. However, this is indicative of classes of problems where each function evaluation takes more than a few seconds to solve and thus for cases where we don't know much about the function and we are forced to sample blindly, we would like to sample as wisely as possible. MAQS provides a method for attacking such a problem. As depicted in Figure 5.14 we see that even in two dimensions MAQS is quite adept at handling this type of problem and much more so than the industrial solution offered by Matlab. Note: For this problem we let the length of the duct be 1 and varied the parameter of the system vary from 1 to 1.5.

5.6 Nonsmooth Test Suite in d-Dimensions

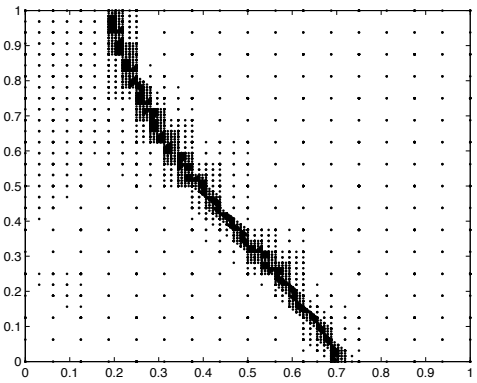
In this section we continue our investigation of the circle, line singularity and shock integrands but in higher dimensional volumes. For these cases we compare MAQS with Monte Carlo Integration. In the discontinuous cases, as with the two dimensional experiments, MAQS proves to be the better method.



(a) Function Defined by 100k nodes



(b) Matlab Points: 14,902



(c) MAQS Points: 2,102

Figure 5.14: Shock 2D

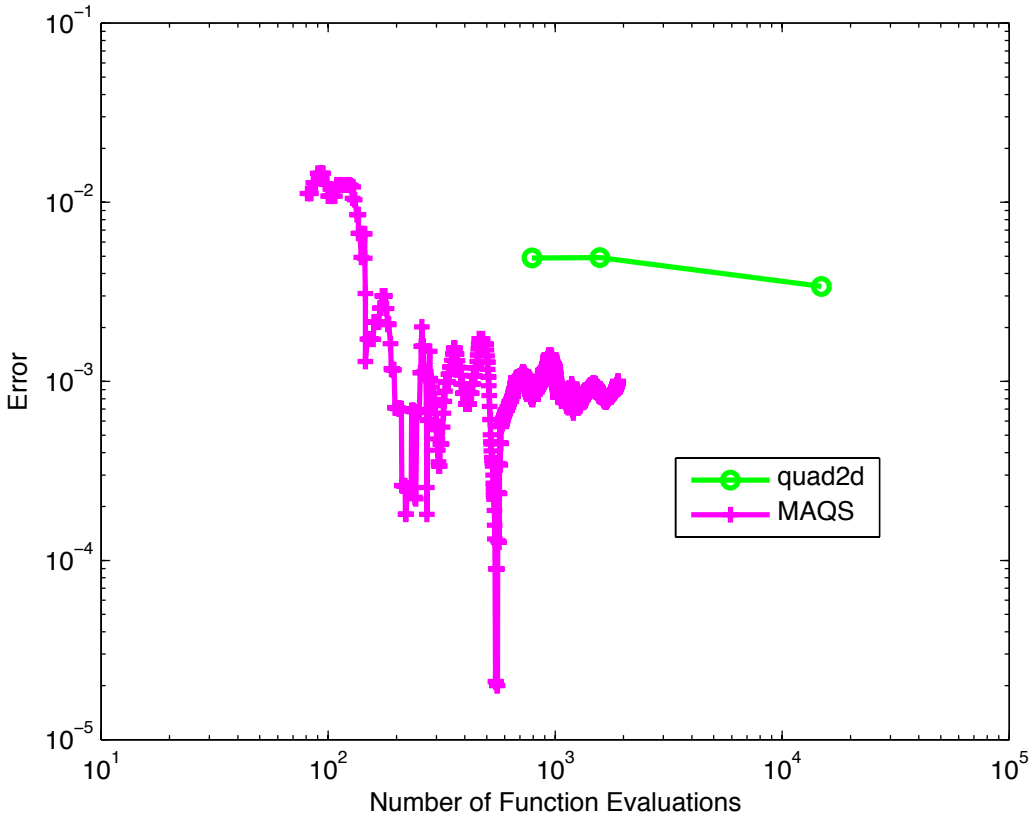
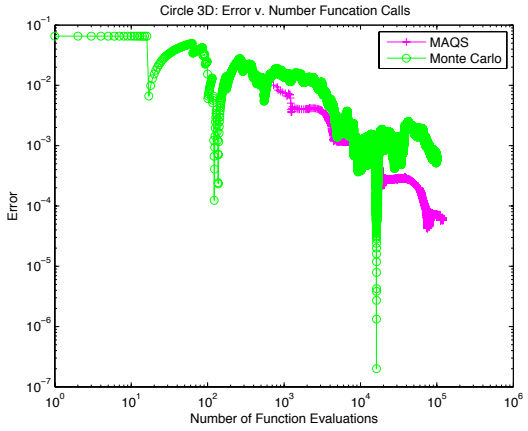
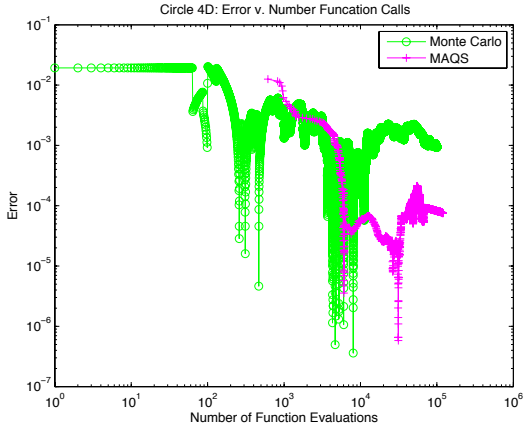


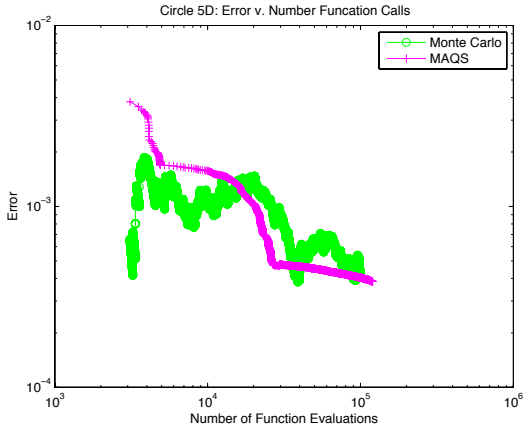
Figure 5.15: Shock MAQS v. `quad2d`: Error v. Function Evaluations



(a) Zero-One Problem in 3D

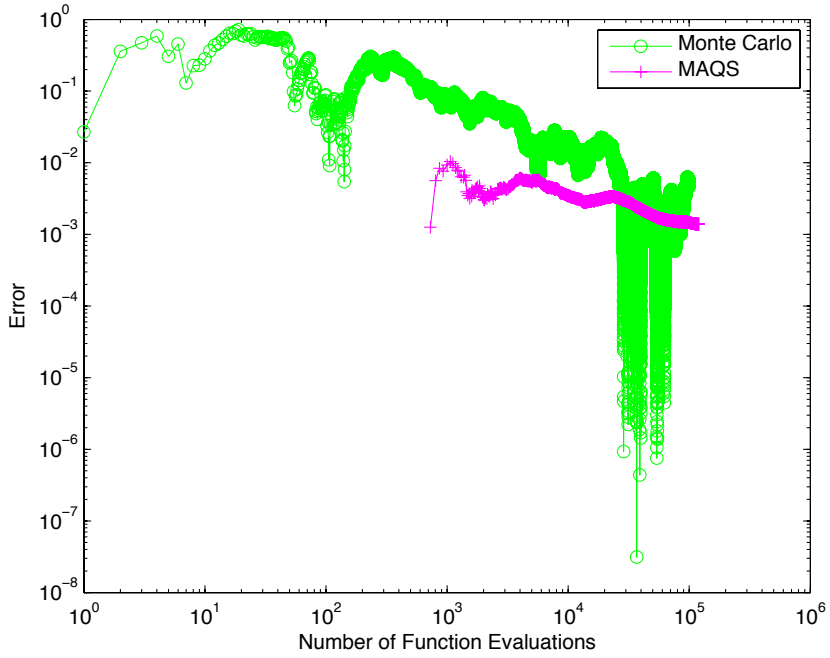


(b) Zero-One Problem in 4D

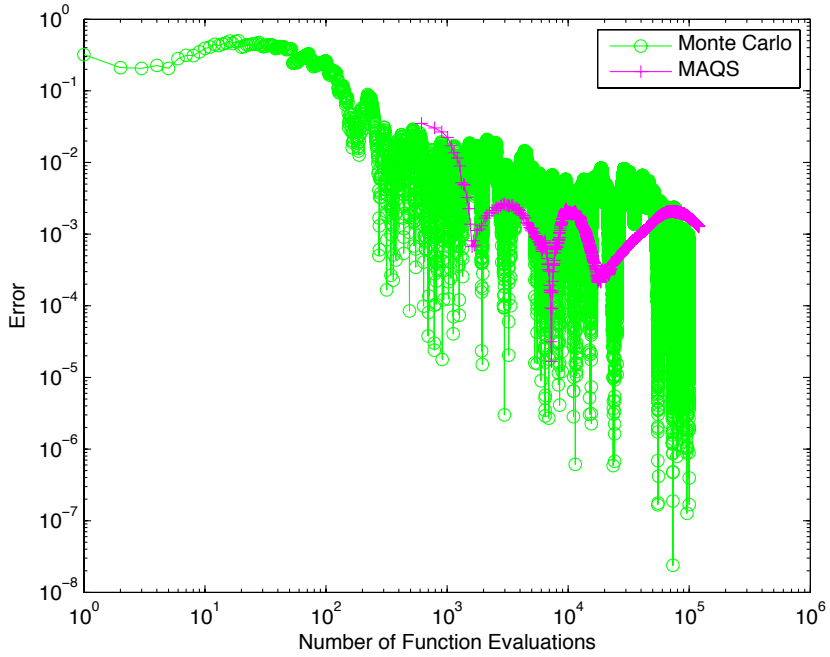


(c) Zero-One Problem in 5D

Figure 5.16: Circle: MAQS v. Monte Carlo in 3, 4 and 5 Dimensions

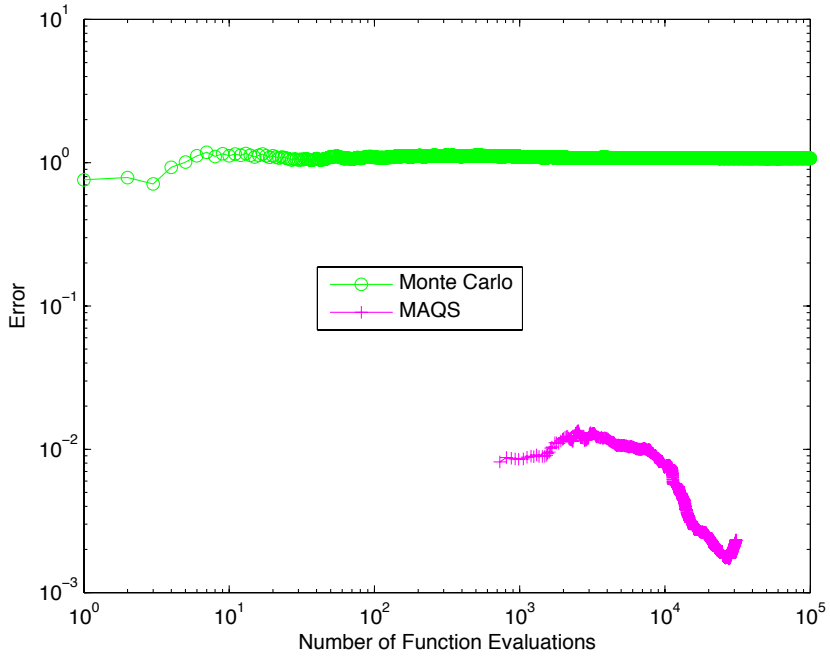


(a) $3D$

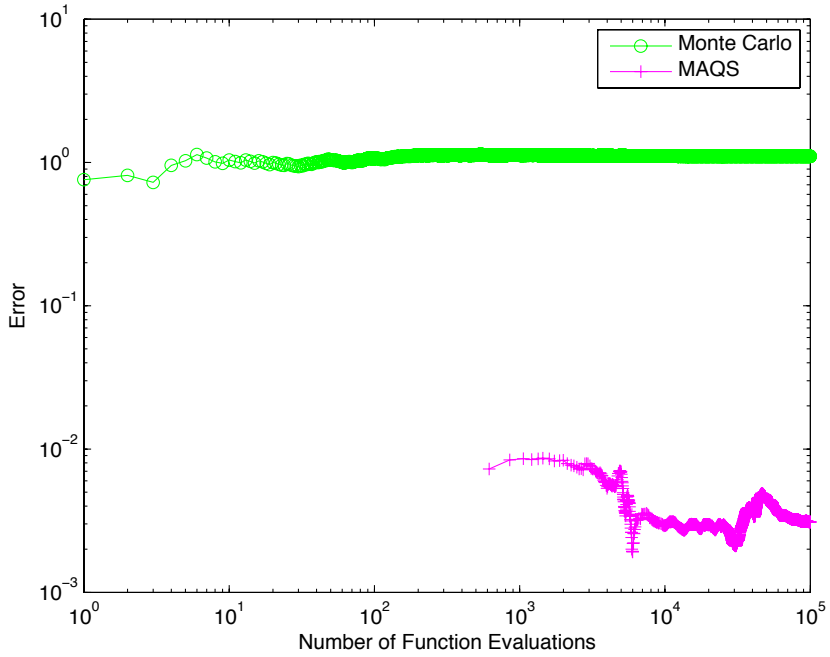


(b) $4D$

Figure 5.17: Line Singularity: MAQS v. Monte Carlo in 3 and 4 Dimensions



(a) $3D$



(b) $4D$

Figure 5.18: Shock: MAQS v. Monte Carlo in 3 and 4 Dimensions

Chapter 6

Conclusions

The major contribution of this work was the development and analysis of a novel multidimensional adaptive quadrature routine defined over simplices titled MAQS. The methodology behind MAQS is based on the error estimates and heuristic adaptive strategy derived in this work. MAQS can be viewed as a natural extension of other multivariate adaptive quadrature rules but specifically tailored to the simplex geometry. Unlike many adaptive methods, this routine is well suited to handle cases where the function being integrated has discontinuities.

The numerical experiments included in this work give computational validation to the theoretical results and demonstrate the superiority of MAQS over industrial standards and Monte Carlo integration in dimensions 2–6 when integrating piecewise continuous and discontinuous functions.

Although the error estimator derived here uses quadratic polynomials, the integral estimate is based solely on the linear quadrature estimate. In the future we would like to develop the theory to support degree adaptivity as well as mesh adaptivity. Specifically, the next step is to have the algorithm use up to quartic polynomials for both integral and error estimation.

Direct applications for MAQS include Uncertainty Quantification and Parameter Estimation. We are eager to investigate problems arising in these fields where a system

in question is known to have discontinuities in the solution, as these are systems where MAQS has clear advantages.

Bibliography

[1] G. Baszenski and F.-J. Delvos. Multivariate Boolean midpoint rules. In H. Brass and G. Hämmerlin, editors, *Numerical Integration IV*, volume 112 of *International Series of Numerical Mathematics*, pages 1–11. Birkhäuser, Basel, 1993.

[2] J. Berntsen, R. Cools, and T.O. Espelid. Algorithm 720: An algorithm for adaptive cubature over a collection of 3-dimensional simplices. *ACM Transactions on Mathematical Software (TOMS)*, 19(3):320–332, 1993.

[3] J. Berntsen and T.O. Espelid. Error estimation in automatic quadrature routines. *ACM Transactions on Mathematical Software (TOMS)*, 17(2):233–252, 1991.

[4] J. Berntsen and T.O. Espelid. Algorithm 706: DCUTRI: An algorithm for adaptive cubature over a collection of triangles. *ACM Transactions on Mathematical Software (TOMS)*, 18(3):329 – 342, September 1992.

[5] J. Berntsen, T.O. Espelid, and A. Genz. An adaptive algorithm for the approximate calculation of multiple integrals. *ACM Transactions on Mathematical Software (TOMS)*, 17(4):437–451, 1991.

[6] T. Bonk. A new algorithm for multi-dimensional adaptive numerical quadrature. In W. Hackbusch and G. Wittum, editors, *Adaptive methods–algorithms, theory and applications: Proceedings of the Ninth GAMM-Seminar, Kiel, January 22-24, 1993*, pages 54–68. Friedr Vieweg & Sohn Verlagsgesellschaft, Vieweg, Braunschweig, 1994.

[7] K. Chung and T. Yao. On lattices admitting unique Lagrange interpolations. *SIAM Journal on Numerical Analysis*, 14(4):735–743, 1977.

[8] R. Cools and A. Haegemans. Algorithm 824: CUBPACK: A package for automatic cubature; framework description. *ACM Transactions on Mathematical Software (TOMS)*, 29(3):287 – 296, 2003.

[9] R. Cools and P. Rabinowitz. Monomial cubature rules since "Stroud": A compilation. *Journal of Computational and Applied Mathematics*, 48:309–326, 1993.

[10] G.F. Corliss and L.B. Rall. Adaptive, self-validating numerical quadrature. *SIAM Journal on Scientific and Statistical Computing*, 8:831 – 847, 1987.

[11] P.J. Davis and P. Rabinowitz. *Methods of numerical integration*. Academic press, New York, 1984.

[12] F.J. Delvos. d-Variate Bolean interpolation. *Journal of Approximation Theory*, 34(2):99–114, 1982.

[13] M.G. Duffy. Quadrature over a pyramid or cube of integrands with a singularity at a vertex. *SIAM Journal on Numerical Analysis*, 19(6):1260–1262, 1982.

[14] C. Feuersanger. An efficient sparse grid method for the high dimensional Fokker Planck equation. Slides, 2009.

[15] K. Frank and S. Heinrich. Computing discrepancies of Smolyak quadrature rules. *Journal of Complexity*, 12(4):287–314, 1996.

[16] P.D. Frank and G.R. Shubin. A comparison of optimization-based approaches for a model computational aerodynamics design problem. *Journal of Computational Physics*, 98(1):74 – 89, 1992.

[17] W. Gander and W. Gautschi. Adaptive quadrature—revisited. *BIT Numerical Mathematics*, 40(1):84–101, 2000.

[18] M. Gasca and T. Sauer. Polynomial interpolation in several variables. *Advances in Computational Mathematics*, 12(4):377–410, 2000.

[19] A. Genz. Numerical computation of multivariate normal probabilities. *Journal of Computational and Graphical Statistics*, 1(2):141–149, 1992.

[20] A. Genz and R. Cools. An adaptive numerical cubature algorithm for simplices. *ACM Transactions on Mathematical Software (TOMS)*, 29(3):297–308, 2003.

[21] A.C. Genz and A.A. Malik. An adaptive algorithm for numerical integration over an N-dimensional rectangular region. *Journal of Computational and Applied Mathematics*, 6(4):295–302, 1980.

[22] T. Gerstner. Adaptive hierarchical methods for landscape representation and analysis. In S. Hergarten and H. Neugebauer, editors, *Process Modeling and Landform Evolution*, volume 78 of *Lecture Notes in Earth Sciences*, pages 75–92. Springer Berlin / Heidelberg, 1999. 10.1007/BFb0009720.

[23] T. Gerstner and M. Griebel. Numerical integration using sparse grids. *Numerical Algorithms*, 18(3):209–232, 1998.

[24] T. Gerstner and M. Griebel. Dimension–adaptive tensor–product quadrature. *Computing*, 71(1):65–87, 2003.

[25] M. Griebel. A parallelizable and vectorizable multi-level algorithm on sparse grids. *Parallel Algorithms for partial differential equations, Notes on Numerical Fluid Mechanics*, 31:94–100, 1990.

[26] M. Griebel and J. Hamaekers. Sparse grids for the Schrödinger equation. *Mathematical Modelling and Numerical Analysis*, 41(2):215–247, 2007.

[27] M. Griebel and F. Koster. Multiscale methods for the simulation of turbulent flows. *Numerical Flow Simulation III: CNRS-DFG Collaborative Research Programme, Results 2000-2002*, pages 203–215, 2002.

[28] M. Griebel, P. Oswald, and T. Schiekofer. Sparse grids for boundary integral equations. *Numerische Mathematik*, 83(2):279–312, 1999.

[29] M. Griebel, M. Schneider, and C. Zenger. A combination technique for the solution of sparse grid problems. *Iterative Methods in Linear Algebra*, pages 263–281, 1992.

[30] M. Griebel and V. Thurner. Efficient solution of fluid dynamics problems by the combination technique. *International Journal of Numerical Methods for Heat and Fluid Flow*, 5(3):251–269, 1995.

[31] M. Griebel and G. Zumbusch. Adaptive sparse grids for hyperbolic conservation laws. In *Hyperbolic Problems: Theory, Numerics, Applications. Proc. of the 7th int. conf., Zürich*, volume 1, pages 411–422, 1999.

[32] P.C. Hammer and A.H. Stroud. Numerical integration over simplexes. *Mathematical Tables and Other Aids to Computation*, 10(55):137–139, 1956.

[33] J.M. Hammersley and D.C. Handscomb. *Monte Carlo methods*. Methuen Young books, 1964.

[34] M. Holtz. *Sparse Grid Quadrature in High Dimensions with Applications in Finance and Insurance*. Doktorarbeit, Institut für Numerische Simulation, Universität Bonn, 2008.

[35] E. Isaacson and H.B. Keller. *Analysis of Numerical Methods*. Wiley, New York, 1966.

[36] F. James. Monte Carlo theory and practice. *Reports on Progress in Physics*, 43:1145–1189, 1980.

[37] D.K. Kahaner and M.B. Wells. An experimental algorithm for N-dimensional adaptive quadrature. *ACM Transactions on Mathematical Software (TOMS)*, 5(1):86–96, 1979.

[38] M.H. Kalos and P.A. Whitlock. *Monte Carlo methods*. Wiley-VCH, 2008.

[39] R. Kleiss and R. Pittau. Weight optimization in multichannel Monte Carlo. *Computer Physics Communications*, 83(2-3):141–146, 1994.

[40] A. Klimke. Sparse grid interpolation toolbox–user's guide. *IANS report*, 1, 2006.

[41] A. Klimke and B. Wohlmuth. Algorithm 847: SPINTERP: piecewise multilinear hierarchical sparse grid interpolation in MATLAB. *ACM Transactions on Mathematical Software (TOMS)*, 31(4):561 – 579, 2005.

[42] V.I. Krylov. *Approximate calculation of integrals*. Macmillan, New York, 1962.

[43] L. Kuipers and H. Niederreiter. *Uniform distribution of sequences*. Wiley New York, 1974.

[44] K.S. Kunz. *Numerical Analysis*. McGraw-Hill, US, 1957.

[45] M. Lai. The multivariate splines and their applications. Meyers: Encyclopedia of Complexity and Systems Science, 2008.

[46] M. Lai and L. Schumaker. *Spline functions on triangulations*. Cambridge University Productions, 2007.

[47] M.J. Lai and L.L. Schumaker. On the approximation power of bivariate splines. *Advances in Computational Mathematics*, 9(3):251–279, 1998.

[48] G.P. Lepage. A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics*, 27(2):192–203, 1978.

[49] J.N. Lyness. Symmetric integration rules for hypercubes I. Error coefficients. *Mathematics of Computation*, 19(90):260–276, 1965.

[50] J.N. Lyness. Quadrature over a simplex: Part 1. A representation for the integrand function. *SIAM Journal on Numerical Analysis*, 15(1):122–133, 1978.

[51] J.N. Lyness. Quadrature over a simplex: Part 2. A representation for the error functional. *SIAM Journal on Numerical Analysis*, 15(5):870–887, 1978.

[52] J.N. Lyness and J.J. Kaganove. Comments on the nature of automatic quadrature routines. *ACM Transactions on Mathematical Software (TOMS)*, 2(1):65–81, 1976.

[53] X. Ma and N. Zabaras. An adaptive hierarchical sparse grid collocation algorithm for the solution of stochastic differential equations. *Journal of Computational Physics*, 228(8):3084 – 3113, 2009.

[54] M.A. Malcolm and R.B. Simpson. Local versus global strategies for adaptive quadrature. *ACM Transactions on Mathematical Software (TOMS)*, 1(2):129–146, 1975.

[55] D.P. Mitchell. Consequences of stratified sampling in graphics. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 277 – 280. ACM, 1996.

[56] R.A. Nicolaides. On a class of finite elements generated by Lagrange interpolation. *SIAM Journal on Numerical Analysis*, 9(3):435–445, 1972.

[57] H. Niederreiter. *Random number generation and quasi-Monte Carlo methods*. Society for Industrial Mathematics, 1992.

[58] E. Novak and K. Ritter. Global optimization using hyperbolic cross points. *Nonconvex Optimization and its Applications*, 7:19–34, 1996.

[59] E. Novak and K. Ritter. High dimensional integration of smooth functions over cubes. *Numerische Mathematik*, 75(1):79–97, 1996.

[60] E. Novak and K. Ritter. The curse of dimension and a universal method for numerical integration. In *Multivariate Approximation and Splines: Conference in Mannheim, September 7-10, 1996 (International Series of Numerical Mathematics)*, pages 177–188. Birkhäuser Basel, 1997.

[61] E. Novak and K. Ritter. Simple cubature formulas with high polynomial exactness. *Constructive approximation*, 15(4):499–522, 1999.

[62] E. Novak, K. Ritter, and A. Steinbauer. A multiscale method for the evaluation of Wiener integrals. *Approximation Theory IX*, 2:251–258, 1998.

[63] S.V. Pereverzev. An estimate of the complexity of the approximate solution of Fredholm equations of the second kind with differentiable kernels. *Ukrainian Mathematical Journal*, 41(10):1225–1227, 1989.

[64] C. Reisinger and G. Wittum. On multigrid for anisotropic equations and variational inequalities "pricing multi-dimensional european and american options". *Computing and Visualization in Science*, 7(3):189–197, 2004.

[65] W. Romberg. Vereinfachte numerische integration. *Norske Vid. Selsk. Forhdl*, 28(7):30–36, 1955.

[66] T. Sauer and Y. Xu. A case study in multivariate Lagrange interpolation. *NATO ASI Series C Mathematical and Physical Sciences-Advanced Study Institute*, 454:443–452, 1995.

[67] T. Sauer and Y. Xu. On multivariate Lagrange interpolation. *Mathematics of Computation*, 64(211):1147–1170, 1995.

[68] C. Schwab and R.A. Todor. Sparse finite elements for elliptic problems with stochastic loading. *Numerische Mathematik*, 95(4):707–734, 2003.

[69] L.F. Shampine. MATLAB program for quadrature in 2D. *Applied Mathematics and Computation*, 202(1):266–274, 2008.

[70] L.F. Shampine. Vectorized adaptive quadrature in MATLAB. *Journal of Computational and Applied Mathematics*, 211(2):131–140, 2008.

[71] I.H. Sloan and H. Wozniakowski. When are quasi-Monte Carlo algorithms efficient for high dimensional integrals? *Journal of Complexity*, 14(1):1–33, 1998.

[72] S.A. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. In *Dokl. Akad. Nauk SSSR*, volume 4, pages 240–243, 1963.

[73] I.M. Sobol. *A primer for the Monte Carlo method*. CRC, US, 1994.

[74] A.H. Stroud. *Approximate calculation of multiple integrals*. Prentice-Hall, Englewood Cliffs, NJ, 1971.

[75] Y.N. Subbotin. Error of the approximation by interpolation polynomials of small degrees on n-simplices. *Mathematical Notes*, 48(4):1030–1037, 1990.

[76] V.N. Temlyakov. Approximation of periodic functions. *Computational Mathematics and Analysis Series*, 1993.

[77] P. van Dooren and L. de Ridder. An adaptive algorithm for numerical integration over an n-dimensional cube. *Journal of Computational and Applied Mathematics*, 2(3):207–217, 1976.

[78] T. von Petersdorff and C. Schwab. Numerical solution of parabolic equations in high dimensions. *Mathematical Modelling and Numerical Analysis*, 38(1):93–127, 2004.

[79] G.W. Wasilkowski and H. Wozniakowski. Explicit cost bounds of algorithms for multivariate tensor product problems. *Journal of Complexity*, 11(1):1–56, 1995.

[80] D.F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The computer journal*, 24(2):167–172, 1981.

[81] S. Weinzierl. Introduction to Monte Carlo methods. *Arxiv preprint hep-ph/0006269*, 2000.