OAK RIDGE National Laboratory

Analysis of SPDEs and numerical methods for UQ

Part III: High-dimensional stochastic approximation: accuracy and efficiency

John Burkardt[†] & Clayton Webster[⋆]

Thanks to Max Gunzburger & Guannan Zhang (FSU), Fabio Nobile (MOX), Raul Tempone (KAUST)

[†]Department of Scientific Computing
Florida State University

[⋆]CEES, CASL, Computer Science and Mathematics Division
Oak Ridge National Laboratory

April 2-3, 2012

Outline

A Deterministic PDE

A common class of scientific and engineering problems can be written down as a system of time-dependent partial differential equations with boundary conditions, defined over a spatial domain $\mathcal{D}$ and time interval $[0, T]$:

$$\begin{aligned} u_t &= \mathcal{L}(u) \quad \text{for } t \in (0, T], x \in \mathcal{D} \\ \mathcal{B}(u) &= 0 \quad \text{for } t \in (0, T], x \in \partial\mathcal{D} \\ u &= u_0 \quad \text{for } t = 0, x \in \mathcal{D} \end{aligned}$$

where $\mathcal{L}$ is some spatial differential operator, $\mathcal{B}$ specifies boundary conditions, and u_0 gives initial conditions.

A mathematical solution of this problem would be a mathematical formula for the function $u(x, t)$, representing full information at absolutely every point in $\mathcal{D} \times [0, T]$.

Collocation

Computationally, we must deal with numbers, not functions.

Instead of seeking a functional solution to the PDE, we might select appropriate discretizations of the time and space derivatives and the boundary and initial conditions, and then determine a discrete set of collocation points $P = \{(x,t)_i : 1 \leq i \leq M\}$ at which to compute solutions to the problem.

The result is a set of solutions $\{u(x,t)_i : 1 \leq i \leq M\}$ to the discretized equation at the collocation points, from which we desire to make inferences about the original equation, over the entire domain.

Collocation

The solution of the problem at a collocation point can be regarded abstractly as a "function evaluation", where the procedure is much more costly than simply evaluating a simple mathematical formula.

The collocation points might be prescribed for us; otherwise, there might be reasons of simplicity or efficiency that suggest a pattern to use.

Once we have the solution data, we are familiar with techniques for constructing an interpolant over the domain or approximating the integral or other quantities of interest derived from our approximate model of the solution.

A Stochastic PDE

As a simple model of a stochastic PDE, we may assume that the random variable y represents a parameter in the problem, for which we only have a statistical representation. Mathematically, we might write our solution function as $u(x, t, y)$, with the notion that if we actually knew the specific value of y, the problem is deterministic and easily solvable.

Assuming we know the range of y, we may now form a stochastic collocation approach to the problem by choosing some discrete set of values $P = \{(x, t, y)_i : 1 \leq i \leq M\}$ as our collocation points.

Having specified a value of y at each collocation point, each of our set of problems is solvable, and we arrive at a solution set $\{u(x, t, y)_i : 1 \leq i \leq M\}$, as before.

Separating the Stochastic Component

Although the collocation approach can treat all the independent input variables equally, it is natural to think about this stochastic problem in a way that separates the deterministic and stochastic components.

In other words, we may wish to collect the discrete solution data for any given parameter value y_i, use that to construct a model of the solution function $u(x, t, y_i)$. Now, with only y as a discrete parameter, we may try to determine statistical information. For instance, if we know the probability density function for y, we are now in a position to estimate the expected value function $\bar{u}(x, t, y)$.

Collocation Based on Sampling

A sampling approach to the collocation problem might be:

```
for M steps
  Select collocation point by a sampling scheme;
  (This sets inputs and realizes random variables;)
  Solve the deterministic problem for U;
  Evaluate the quantity of interest Q(U);
end
Return the simple average of Q's encountered.
```

The sampling approach concentrates on selecting the collocation points to be well distributed as input to an arbitrary function with respect to the PDF of the stochastic variables; however, it does not investigate or model the nature of the relationship between the input and the quantity of interest.

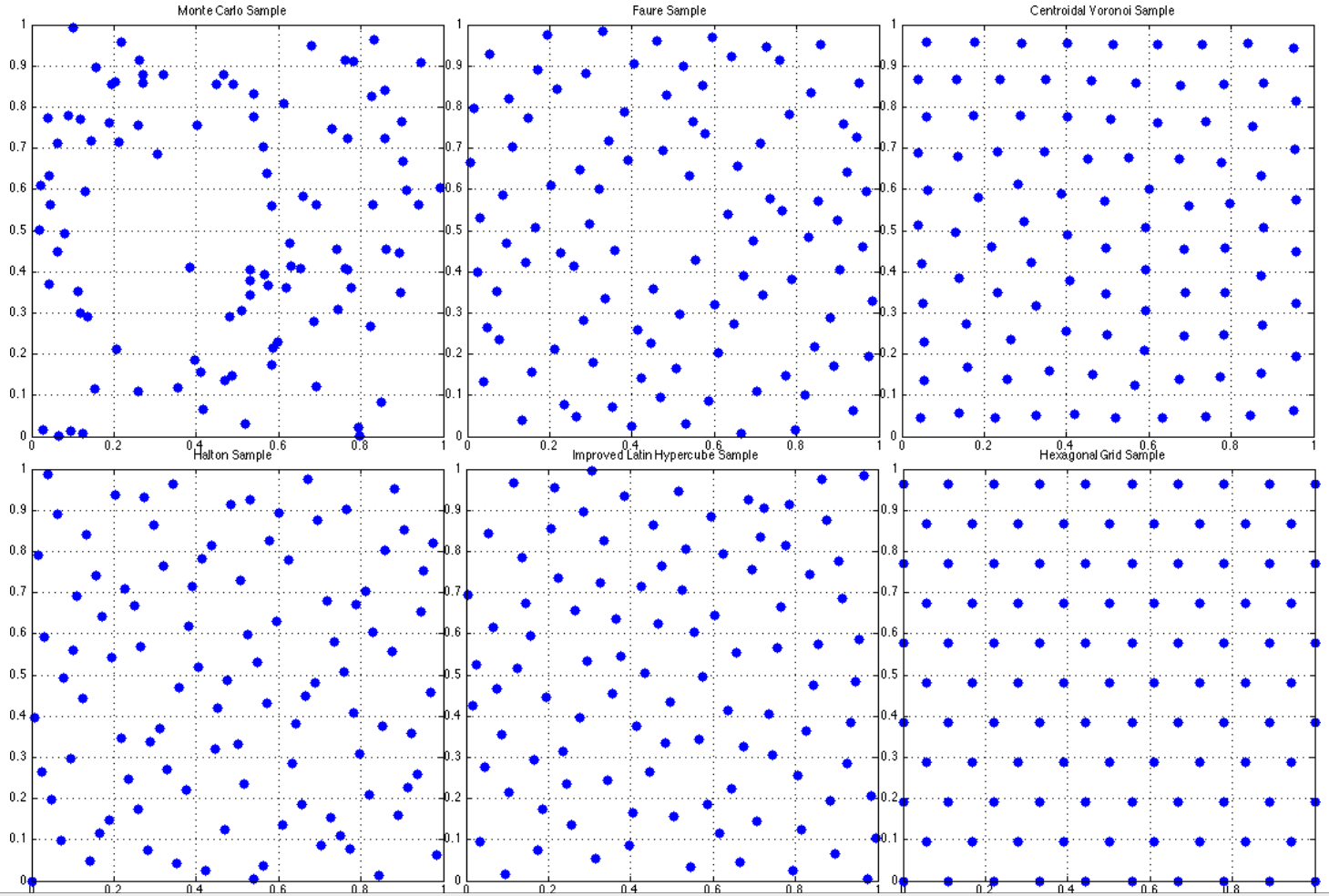
Collocation Based on Sampling

Sampling approaches include the Monte Carlo method, in which the samples are random, and quasirandom methods in which the points are chosen by a formula that attempts to force a more regular space-filling distribution of the input domain.

When **M** sample points are used to estimate the integral of a function, the Monte Carlo estimate error has a typical decay rate of $1/\sqrt{M}$. For some quasirandom methods, an asymptotic decay rate of $1/M$ is claimed.

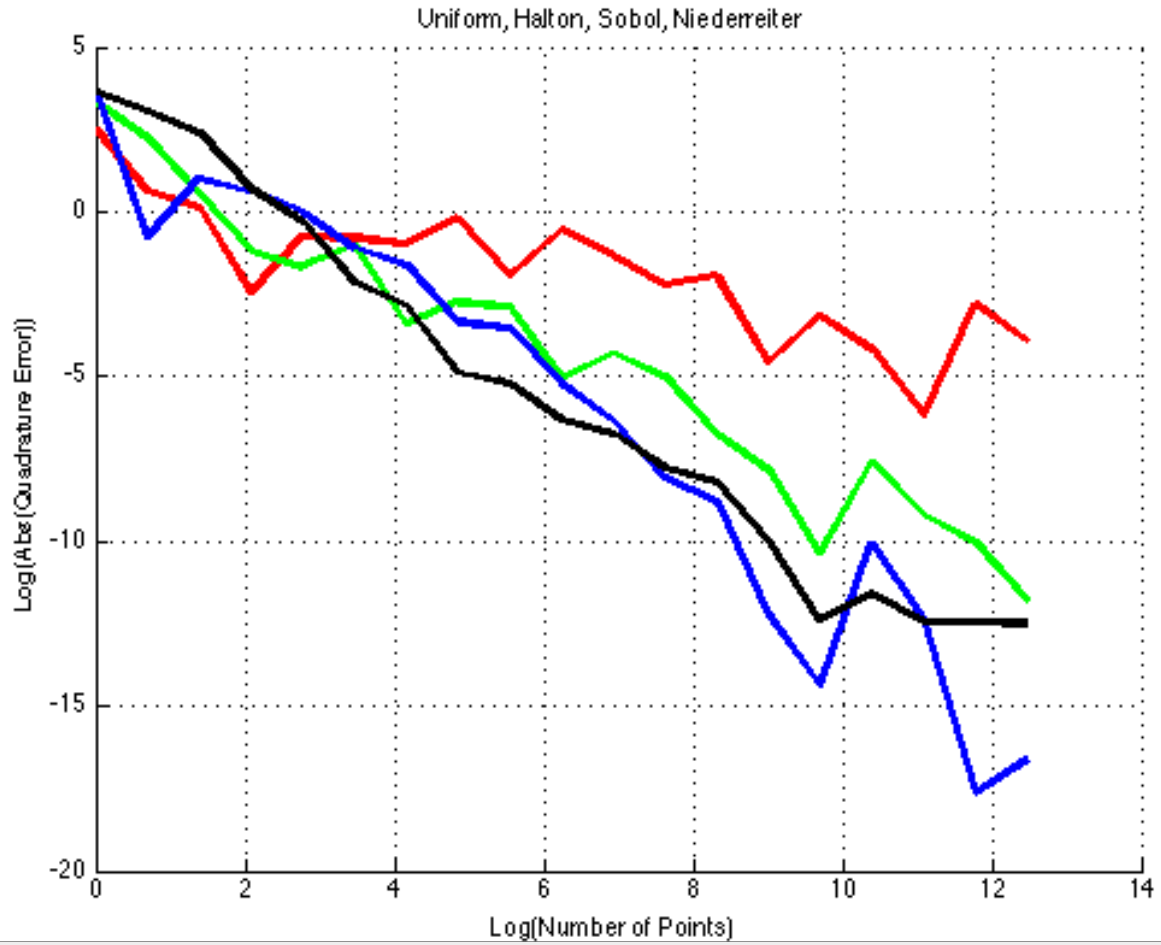
Collocation Based on Sampling

Uniform——Faure——Centroidal Voronoi
Halton——Improved Latin——Hexagonal



Quadrature Error for Sampling Methods

Quadrature error for Monte Carlo (red), Halton (green), Sobol (blue), Niederreiter (black) sampling, spatial dimension N = 3.



Quadrature Error for Sampling Methods

The previous plot was for an integrand in dimension 3. Although the data is rough, it is plausible that the error decay rate for Monte Carlo is roughly $M^{-1/2}$ and that for the quasirandom methods is about M^{-1}.

This decay rate for the Monte Carlo method remains the same no matter what the dimension. Despite the fact that a 10- or 100-dimensional problem is much more complex, the expected sampling method error decay will be the same. (It's certainly possible that the initial error will be much much larger, though.)

For the quasirandom sampling methods, the M^{-1} is really only good for low dimensions, and gradually becomes more like the Monte Carlo rate.

In any case, this insensitivity to high dimensionality is an amazing feature, especially when we see how quickly it fails for other methods!

Collocation Based on Interpolation

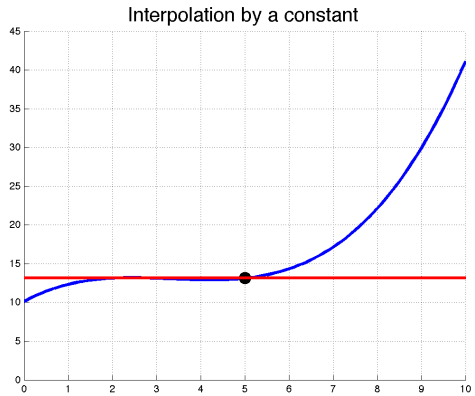
The collocation points can be used in an interpolation procedure. Such a method proposes a relationship between the input and the solution, constructing a model which agrees with the data.

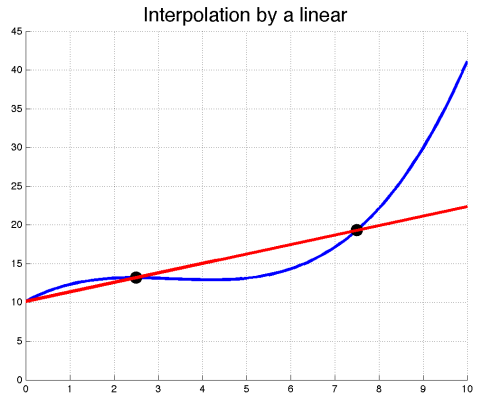
Given two sets of arguments and values for a scalar function of a scalar argument, we make a linear estimate of the function value at nearby arguments. If the true function is actually linear, the model is exact. If the true function has a bounded second derivative, the error is bounded and proportional to the square of the distance to the nearest data point.

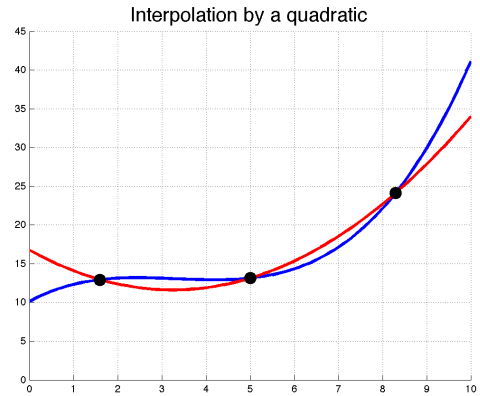
For best results, an interpolatory scheme needs to pick the collocation points in a way that produces a stable and accurate interpolant. Over [-1,+1], for instance, we prefer the zeros of the Chebyshev polynomial instead of equally spaced collocation points.

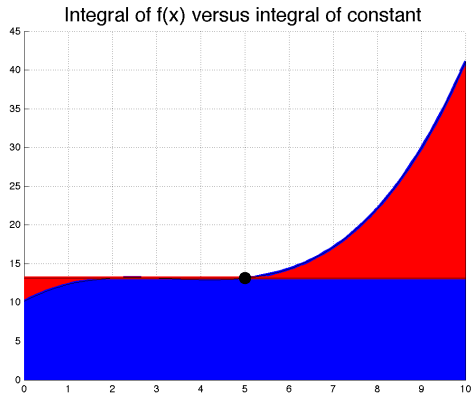
Using Interpolants To Estimate Integrals

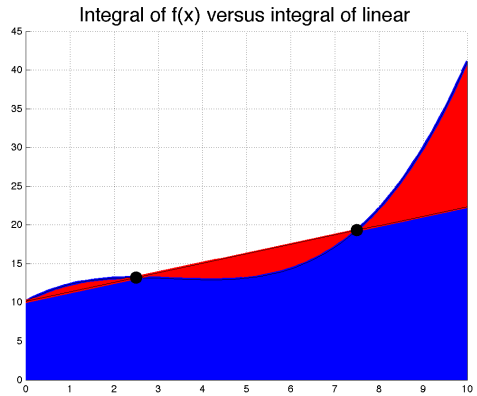
Interpolating with constant, linear, quadratic polynomials;
Estimating the integral by integrating the interpolant.

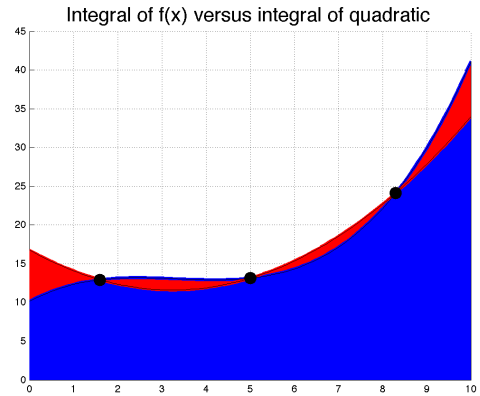












Red regions represent integration error; cancellation reduces this even more.

Sequences of Interpolatory Quadrature Rules

When we think about convergence, we must be able to invoke a sequence of approximations.

For an interpolatory quadrature rule, as we increase the number of points n, the approximation will tend to improve simply because the maximum spacing h will go down. A log-log convergence plot will tend to show a straight line decrease.

If it is also true that the interpolant function being used increases in degree, then the error will drop exponentially, that is, it should eventually decrease faster than any straight line decrease.

Higher Dimensions Using a Product Rule

A quadrature rule in dimension $N = 1$ uses M points x and weights w so that

$$I(f, [a, b]) \approx Q(f) = \sum_{i=1}^{M} w_i \, f(x_i)$$

We can approximate integrals in N dimensions by using a product rule from our 1D rule. For every Cartesian vector $(i_1, i_2, ..., i_N)$ whose components are between 1 and M, we form the point $x = (x_{i_1}, x_{i_2}, ..., x_{i_N})$ and the weight $w = w_{i_1} * w_{i_2} * ...w_{i_N}$.

If the original rule was exact for all polynomials of degree P or less, than the new rule will be exact for all polynomials of maximum degree P or less.

(This fact will turn out to hide a significant problem.)

Sequences of Interpolatory Quadrature Rules

Of course, using a high order interpolant can be dangerous, because high order interpolation can be unstable; moreover, such an approach assumes that the function we are modeling has bounded derivatives up to the order approximated by the interpolant.

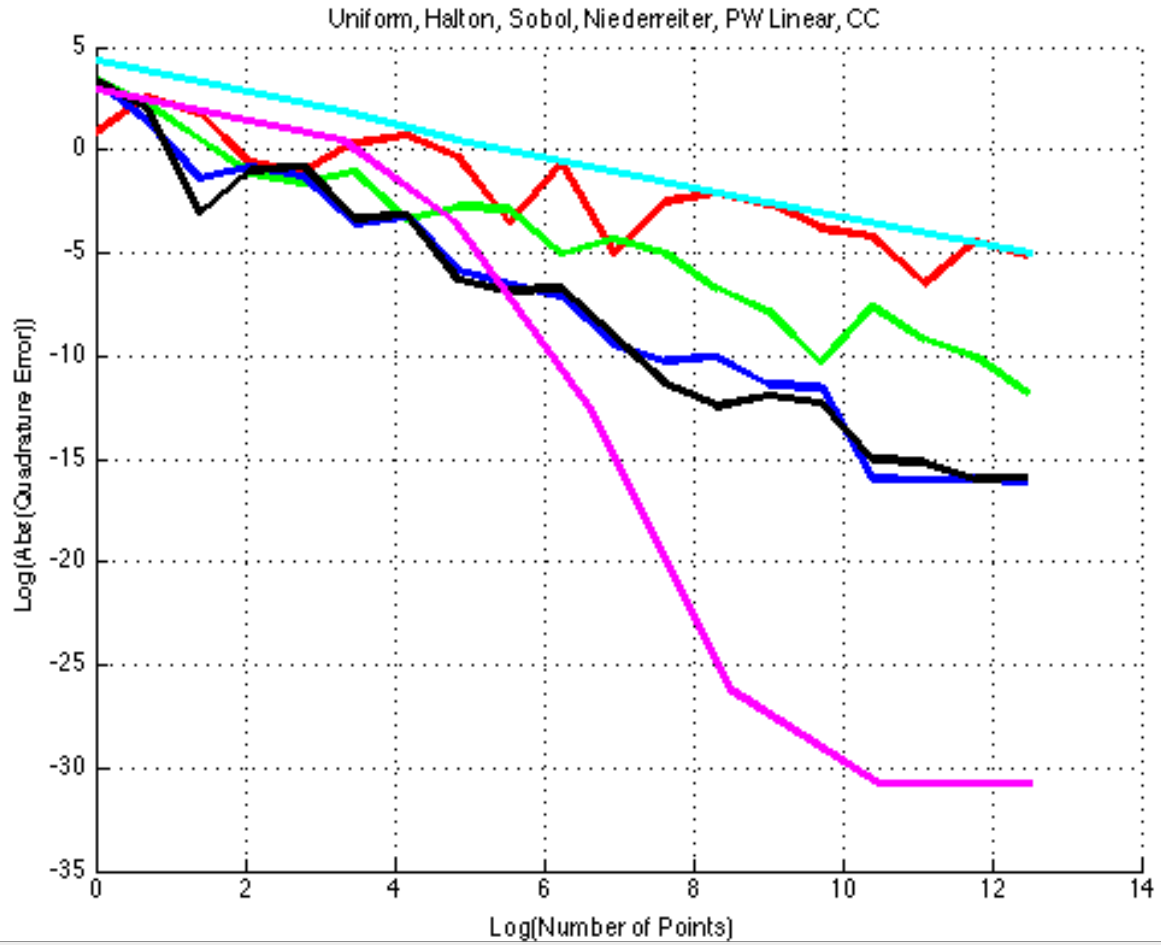
The integrand considered earlier is smooth enough for interpolation.

Let us return to that example, and apply two interpolatory product quadrature rules:

- a piecewise-linear rule (constant degree)
- an interpolant through Chebyshev points (increasing degree)

Quadrature Error for Interpolatory Methods

Monte Carlo (red), Halton (green), Sobol (blue), Niederreiter (black), PWL (cyan), CC (magenta), spatial dimension = 3.



Quadrature Error for Interpolatory Methods

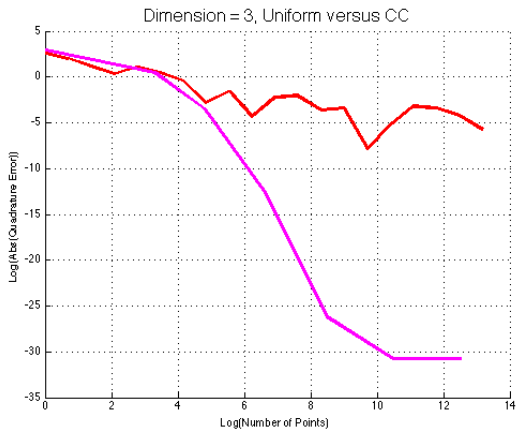
You may be surprised that the piecewise linear interpolant doesn't seem much better than a Monte Carlo interpolant. The issue here is that the convergence of sampling methods is generally independent of the spatial dimension.

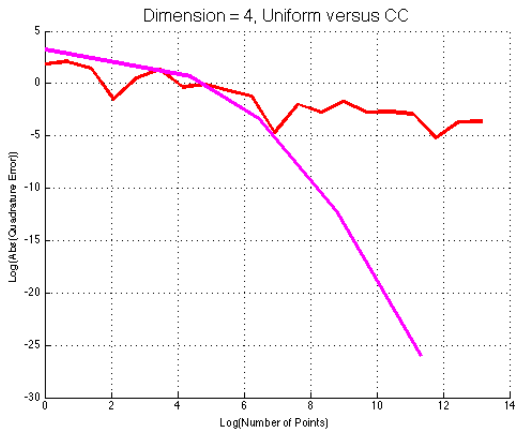
In contrast, part of the accuracy of an interpolatory method comes from reducing the size of the integration intervals...or, in higher dimensions, the integration subcells. To halve the size of a cell in N dimensions requires using about 2^N more points.

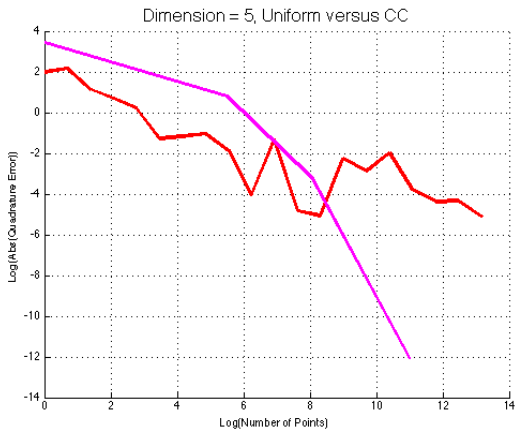
So, when we go to a higher dimension, the horizontal axis of the convergence plot will begin to lengthen quickly, which means that the dramatic drop of the Clenshaw Curtis rule happens more and more slowly. It does not require a very high dimension before we simply can't afford to run the rule out to the accuracy we desire!

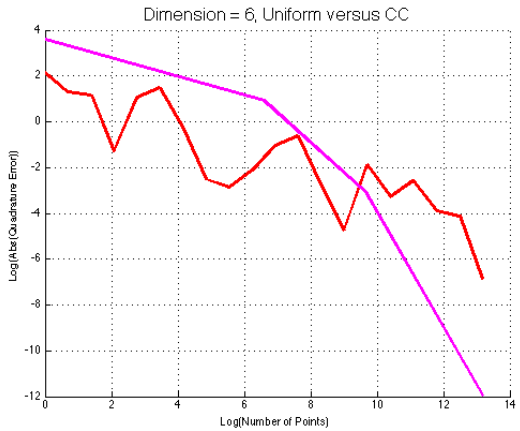
Quadrature Error for Interpolatory Methods

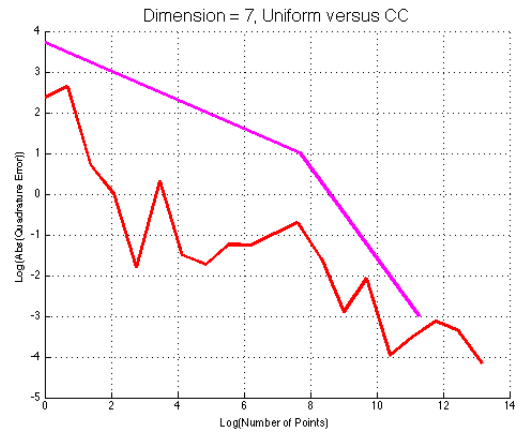
Monte Carlo (red), CC (magenta), spatial dimensions $N = 3$ to 8
Budget of no more than M=1,000,000 points!

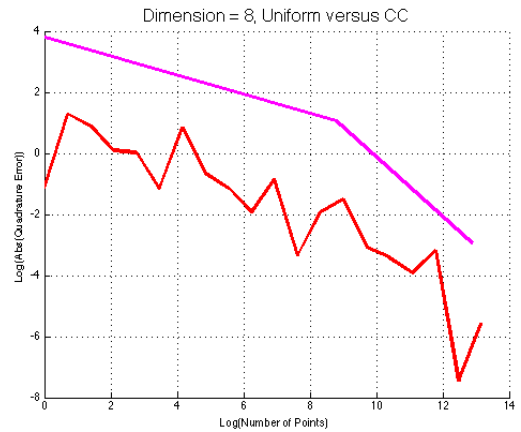












Can We Do Better?

Interpolatory rules for 1D offer a powerful means of accelerating quadrature estimates. But when we use the product rule approach to adapt them for use in higher dimensions, we make a rule that exceeds our budget before we reach the exponential convergence regime.

We seek a technique with the approximating power of interpolatory rules but with a point count that does not explode so rapidly!

Alternatives to Product Rules?

Except for the product rule, it's hard to think of any systematic way to generate quadrature rules for a multiple dimensional problem.

There are certain special families of **cubature rules** that can be developed, but these are based on special algebraic constructions, and typically are only available for a specific dimension, or for a small sequence of orders.

But let's look at the simple question of approximating an integral in two dimensions, and see if there are alternatives to the product rule that also have good convergence behavior.

Quadrature in 2D

Let Q represent a family of 1D quadrature rules, so that $Q(P)$ can integrate polynomials of degree P exactly.

Write any 2D product rule $Q(P_1, P2) = Q(P_1) \otimes Q(P_2)$

To approximate an integral in 2D, up to and including linear terms, we might use the product rule $Q(1, 1) = Q(1) \otimes Q(1)$. This product rule uses M=4 points, and precisely integrates any terms involving the monomials 1, x, y, or xy.

```
 Y2  *       *          |  y^3
  |                     |  y^2 xy^2
 Y1  *       *          |  y    xy    x^2y
  |                     |  1    x     x^2  x^3
  +--X1-----X2--->      +--------------------
  Product Rule in         Polynomial Precision
  Coordinate Space        Pascal's Triangle
```

Quadrature in 2D

Typically, to get good convergence, we want to organize our quadrature rules so that the first one gets constants, the next one adds in the linears, then quadratics, and so on. Here, our $Q(1,1)$ rule has given us a "bonus", by getting constants, linears, but also the xy monomial.

On the other hand, is it a coincidence that we can compute 4 monomials exactly, and required 4 points to do so?

The interesting question is, *can we construct a linear rule, which integrates only 1, x and y precisely, and only requires three function evaluations?*

In general, can we find a quadrature rule that marches through Pascal's triangle (or tetrahedron, or simplex) efficiently?

Quadrature in 2D

Let's focus for a moment on our simple problem of a quadrature rule that is exact for linears in 2D.

Instead of using the rule $Q(1,1) = Q(1) \otimes Q(1)$, what if we computed the following combination of three rules:

$$\mathcal{A}(\text{L=1,N=2}) = Q(1,0) + Q(0,1) - Q(0,0)$$

(The L indicates the level, and N the spatial dimension.)

Surprisingly, this quantity will give the precise integral for any polynomial in 1, x, and y...but not xy.

However, the cost of this computation would seem to have gone up, since the first two rules need 2 points each, and the last 1.

Rewrite to Form a New Quadrature Rule

Suppose the 1D rules are nested, so the second rule includes the point used in the first rule, the third rule includes the two points in the second and so on. For our example, we might have:

```
Q(0,0) = a * f(0,0)
Q(1,0) = b * f(0,0) + c * f(1,0)
Q(0,1) = d * f(0,0) + e * f(0,1)
```

where *a, b, c, d, e* are appropriate weights. If we compute the three quadrature rules as separate computations, and then combine the numeric results, the computation does indeed require 5 function evaluations. But if we take advantage of the nesting, we can get the same result by regrouping:

$$\mathcal{A}(\text{L=1,N=2}) = (b + d - a) * f(0,0) + c * f(1,0) + e * f(0,1)$$

Now we only need 3 function evaluations, and have a rule precise for the constants and linears.

Can We Go to Higher Precision, Higher Dimensions?

The linear rule can be extended to dimension $N = 3$:

$$\begin{aligned}\mathcal{A}(\text{L=1,N=3}) =& (b + d + f - a) * f(0, 0, 0) \\ &+ c * f(\alpha, 0, 0) + e * f(0, \alpha, 0) + g * f(0, 0, \alpha)\end{aligned}$$

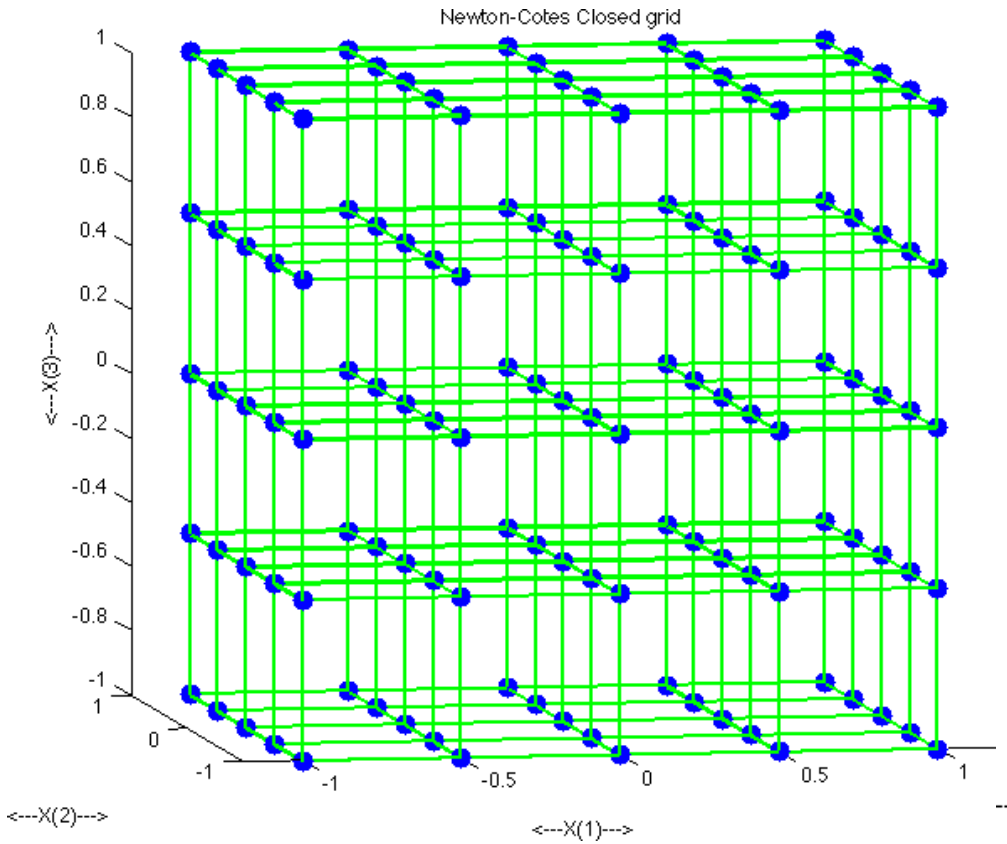
or dimension N, using $N + 1$ function values, precise for constants and linears.

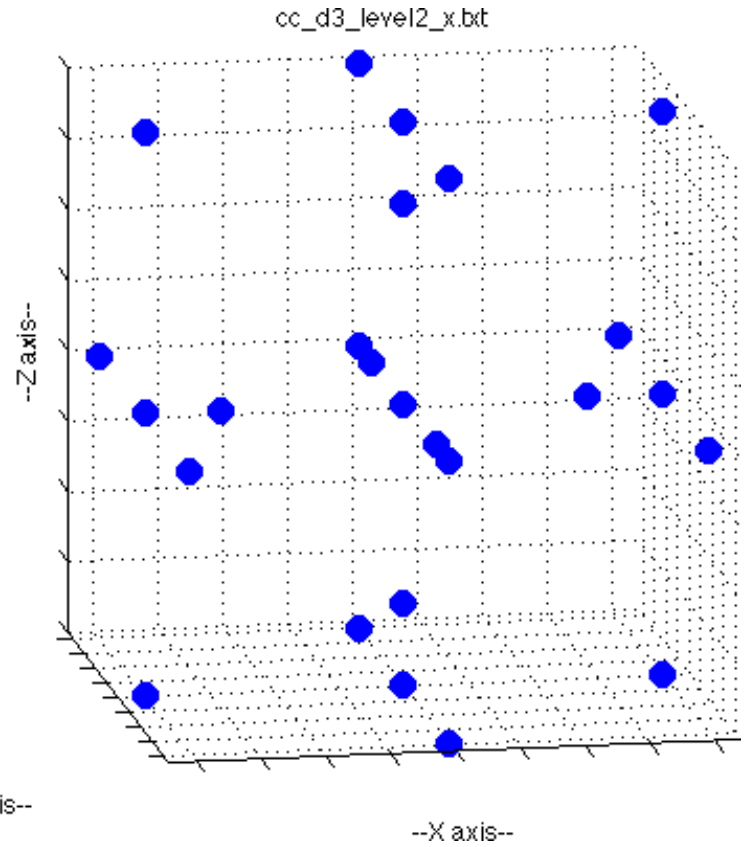
We can hope, then, to work in higher dimensions, possibly through some combination of extrapolation and nested rules.

It's not obvious how to produce higher precision extrapolation rules,; moreover, a strictly nested family of rules, increasing by 1 point at at time, is not actually a good idea even in 1D!

5x5x5 Product Rule Versus $\mathcal{A}(L{=}2,N{=}3)$

Smolyak discovered a suitable, general procedure.
Compare $M{=}125$ points in the product rule, $M{=}25$ points for Smolyak.





http://people.sc.fsu.edu/~jburkardt/m_src/tensor_grid_display/tensor_grid_display.html

The Smolyak Formulation

Smolyak determined a formula combining low-order product rules to achieve a desired precision, avoiding the excessive function evaluation of the standard product rule.

The details of precision, order, efficiency and accuracy vary depending on the underlying 1D quadrature rules.

In the most typical implementation, a nested family of 1D rules is used; this is not required, and one might prefer, for instance, a sequence of (unnested) Gauss-Hermite rules instead.

In most cases, Smolyak can match the precision of the product rule while avoiding the crushing explosion in function evaluations.

Now we'll see the formal definition of the Smolyak procedure; it makes **no assumptions** about the underlying 1D rules.

Formal Definition

We have a family of 1D quadrature rules $\mathcal{Q}^\ell$ indexed by ℓ. A typical product rule would be:

$$\mathcal{Q}^{\ell_1} \otimes \cdots \otimes \mathcal{Q}^{\ell_N}$$

We form a sparse grid $\mathcal{A}(L, N)$ for level **L** and dimension **N** by a weighted sum of N-dimensional product rules.

The vector $\vec{\ell}$ lists the levels of the component rules used, and $|\vec{\ell}| = \ell_1 + \cdots + \ell_N$ is the sum.

$$\mathcal{A}(L, N) = \sum_{L-N+1 \le |\vec{\ell}| \le L} (-1)^{L-|\vec{\ell}|} \begin{pmatrix} N-1 \\ L-|\vec{\ell}| \end{pmatrix} (\mathcal{Q}^{\ell_1} \otimes \cdots \otimes \mathcal{Q}^{\ell_N})$$

Thus, the rule $\mathcal{A}(L, N)$ is a weighted sum of N-dimensional product rules whose total level $|\vec{\ell}|$ never exceeds L.

Apply the Definition to Our Sample

To demystify the definition, let's return to our artificial example, for which the spatial dimension N was 2. Our level L will be 1, *(not because we are chasing linear functions, but because this will be the first rule built upon the basis 1 point rule.)*

We only need the first two 1D quadrature rules for [-1,+1]:

$$\mathcal{Q}^0 = \{0\}, \text{ with weight } \{2\};$$
$$\mathcal{Q}^1 = \{0, 1\} \text{ with weights } \{6/5, 4/5\}.$$

Apply the Definition to Our Sample

Our index $|\vec{\ell}|$ (the sum of the quadrature rule "exponents") will run from $L - N + 1 = 0$ to $L = 1$ and will have the form

$$\begin{aligned}\mathcal{A}(1,2) =& (-1)^1 \cdot C(1,1) \cdot \mathcal{Q}^0 \otimes \mathcal{Q}^0 \\ &+ (-1)^0 \cdot C(1,0) \cdot \mathcal{Q}^1 \otimes \mathcal{Q}^0 \\ &+ (-1)^0 \cdot C(1,0) \cdot \mathcal{Q}^0 \otimes \mathcal{Q}^1\end{aligned}$$

or

$$\mathcal{A}(1,2) = -\mathcal{Q}^0 \otimes \mathcal{Q}^0 + \mathcal{Q}^1 \otimes \mathcal{Q}^0 + \mathcal{Q}^0 \otimes \mathcal{Q}^1$$

We now have the option of:

- computing three integral estimates, combining the results,
- or collecting weights associated with repeated points and creating a unified quadrature rule.

Apply the Definition to Our Sample

The point (0,0) occurs in each rule, so we compute its weight by multiply its weight in each product rule by the Smolyak coefficient for that rule:

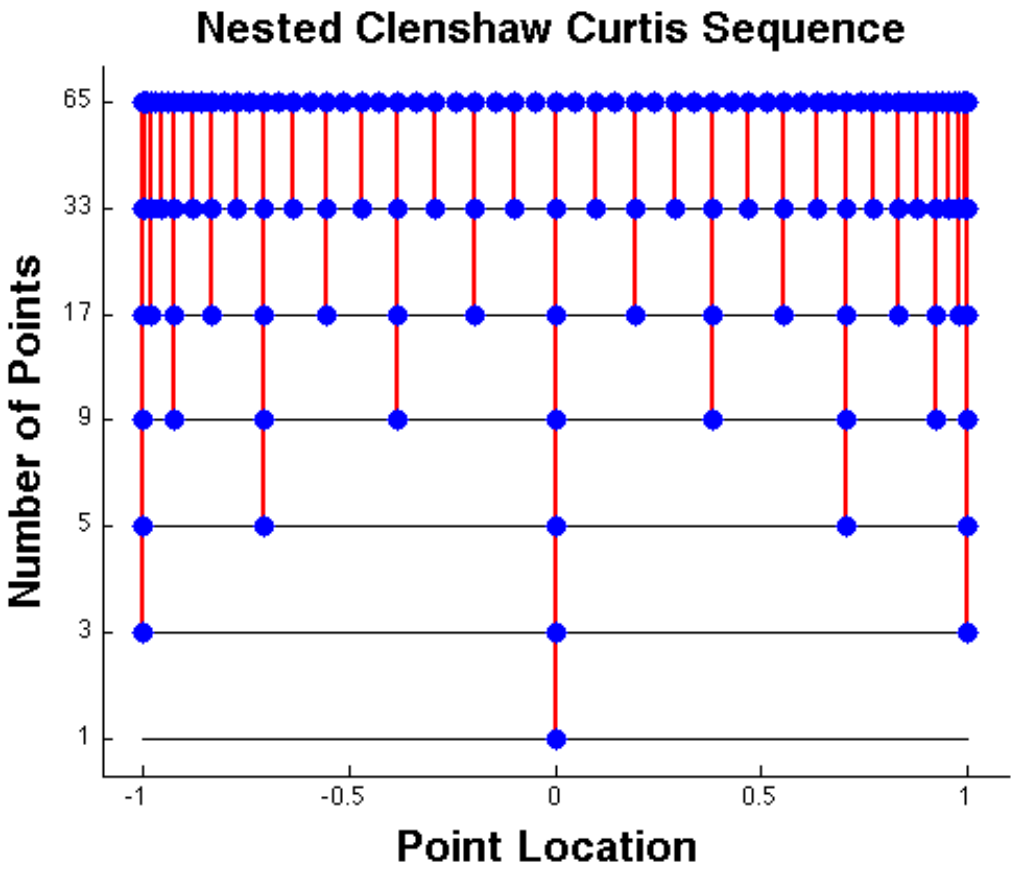
$$
\begin{aligned}
- \text{ weight for (0,0) in } \mathcal{Q}^0 \otimes \mathcal{Q}^0 &= -1 * (2/1 * 2/1) = -20/5 \\
+ \text{ weight for (0,0) in } \mathcal{Q}^1 \otimes \mathcal{Q}^0 &= +1 * (6/5 * 2/1) = +12/5 \\
+ \text{ weight for (0,0) in } \mathcal{Q}^0 \otimes \mathcal{Q}^1 &= +1 * (2/1 * 6/5) = +12/5 \\
& - - - - - - - - - - - \\
\text{weight for (0,0) in } \mathcal{A}(1,2) &= +4/5
\end{aligned}
$$

Similarly, we determine that the weights for (1,0) and (0,1) are each 8/5, so our combined rule can be written:

$$\mathcal{A}(1,2) = \frac{4}{5} f(0,0) + \frac{8}{5} f(1,0) + \frac{8}{5} f(0,1)$$

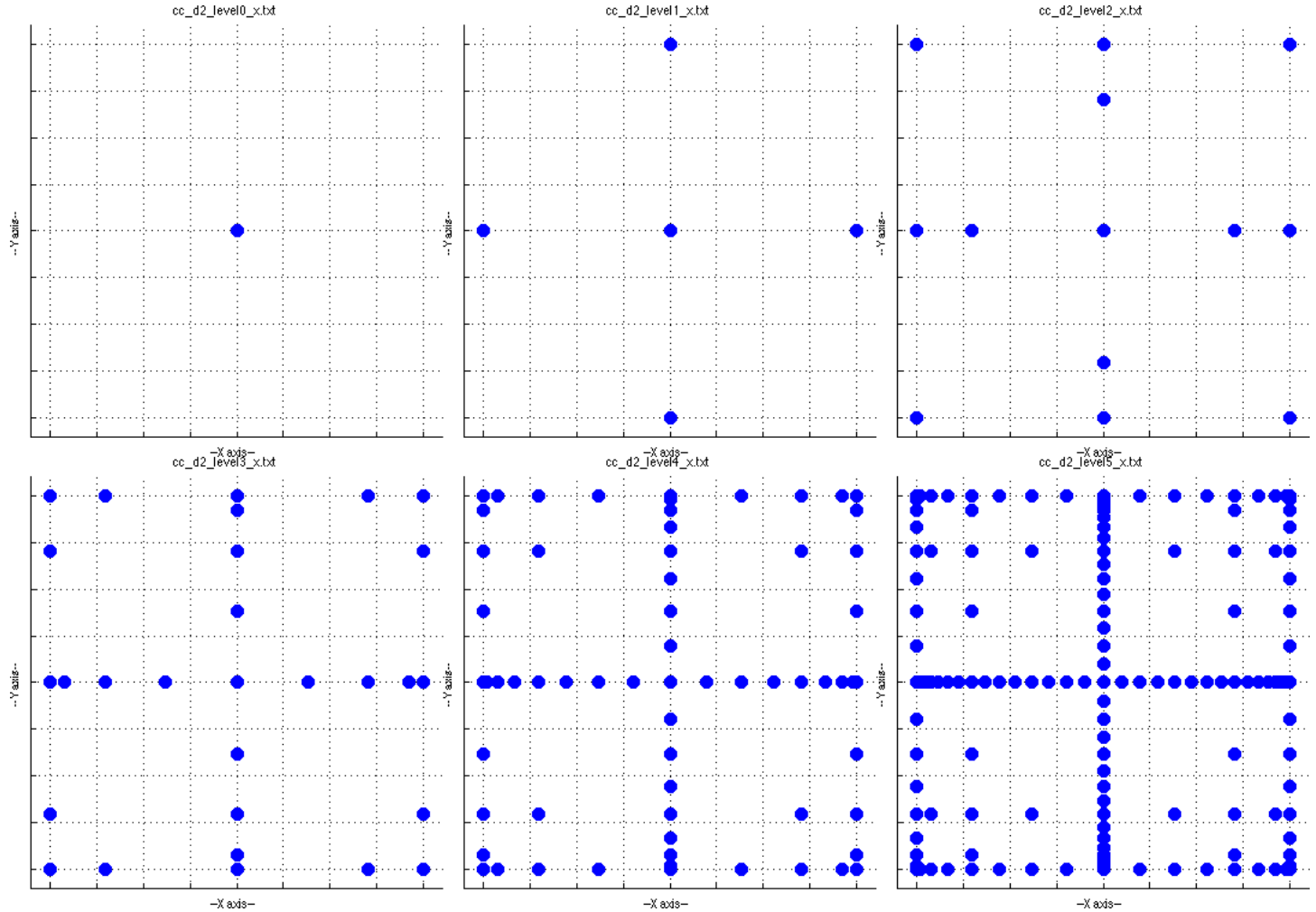
Using a Nested Family

Nestedness keeps the point count down, but if our families add one point at a time, we'll develop bad 1D rules. The Smolyak formula allows rules to grow any way we want. The popular Chebyshev points essentially double each step.



Using a Nested Family

Smolyak grids for $N = 2$ and $L = 0, 1, 2, 3, 4, 5$, using Chebyshev 1D grids:



Using a Nested Family

The 1D grid based on Chebyshev points is often called the Clenshaw Curtis (CC) grid. Although this is just one of many choices for the construction, it is worth knowing some of the properties of sparse grids based on the CC rule:

- the CC grid of level 0 uses $M = 1$ point, the origin, with weight 2^N;
- the CC grid of level 1 uses $M = 2N + 1$ points;
- the CC grid of level L contains the CC grid of level L-1;
- the CC grid of level L precisely integrates total degree 2*L+1 or less;

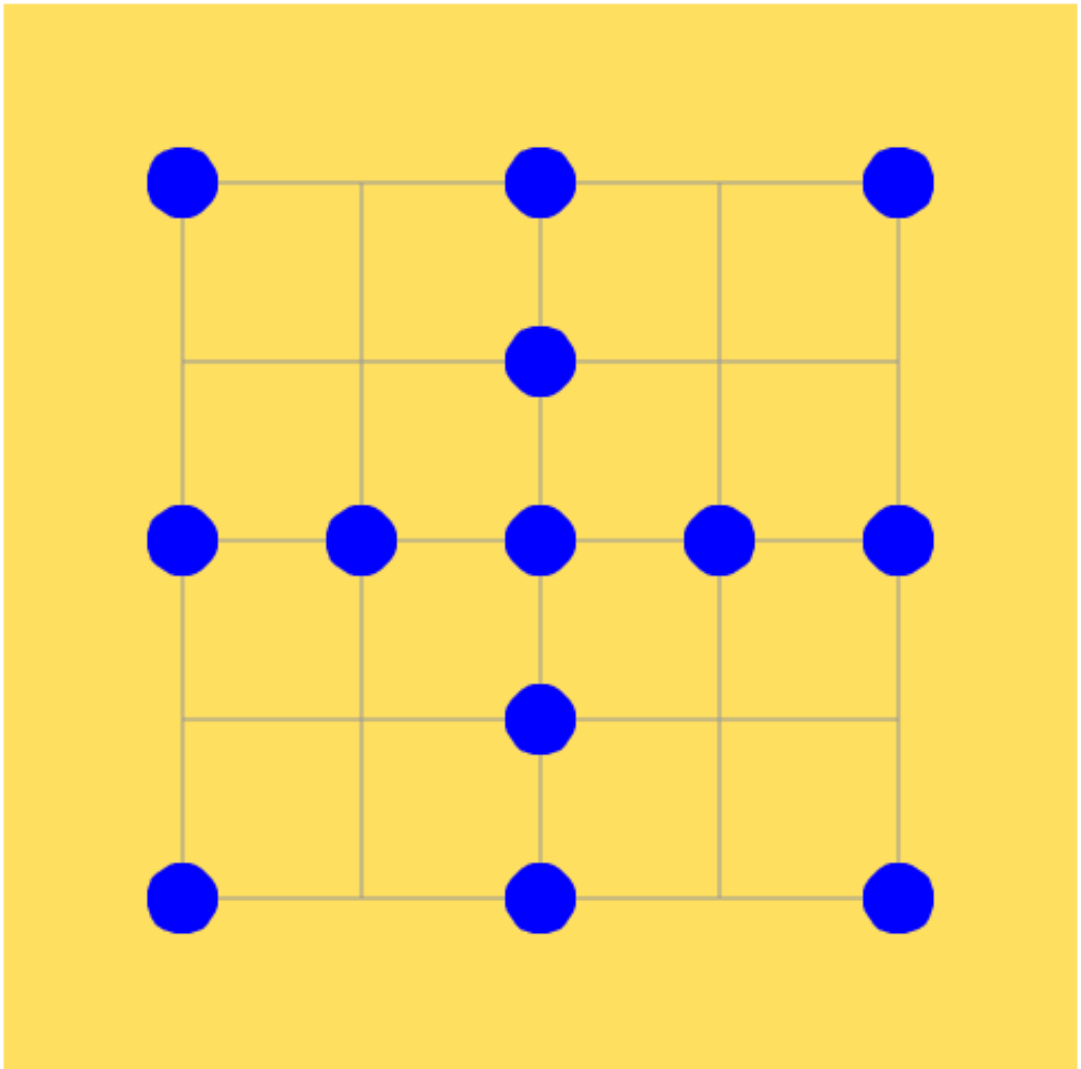
The First Three CC Rules

The CC-based Smolyak rules $\mathcal{A}(L = 0/1/2, N = 2)$ are:

$$
\begin{aligned}
\mathcal{A}(0,2) =& CC^0 \otimes CC^0 \\
\mathcal{A}(1,2) =& CC^1 \otimes CC^0 \\
&+ CC^0 \otimes CC^1 \\
&- CC^0 \otimes CC^0 \\
\mathcal{A}(2,2) =& CC^2 \otimes CC^0 \\
&+ CC^1 \otimes CC^1 \\
&+ CC^0 \otimes CC^2 \\
&- CC^1 \otimes CC^0 \\
&- CC^0 \otimes CC^1
\end{aligned}
$$

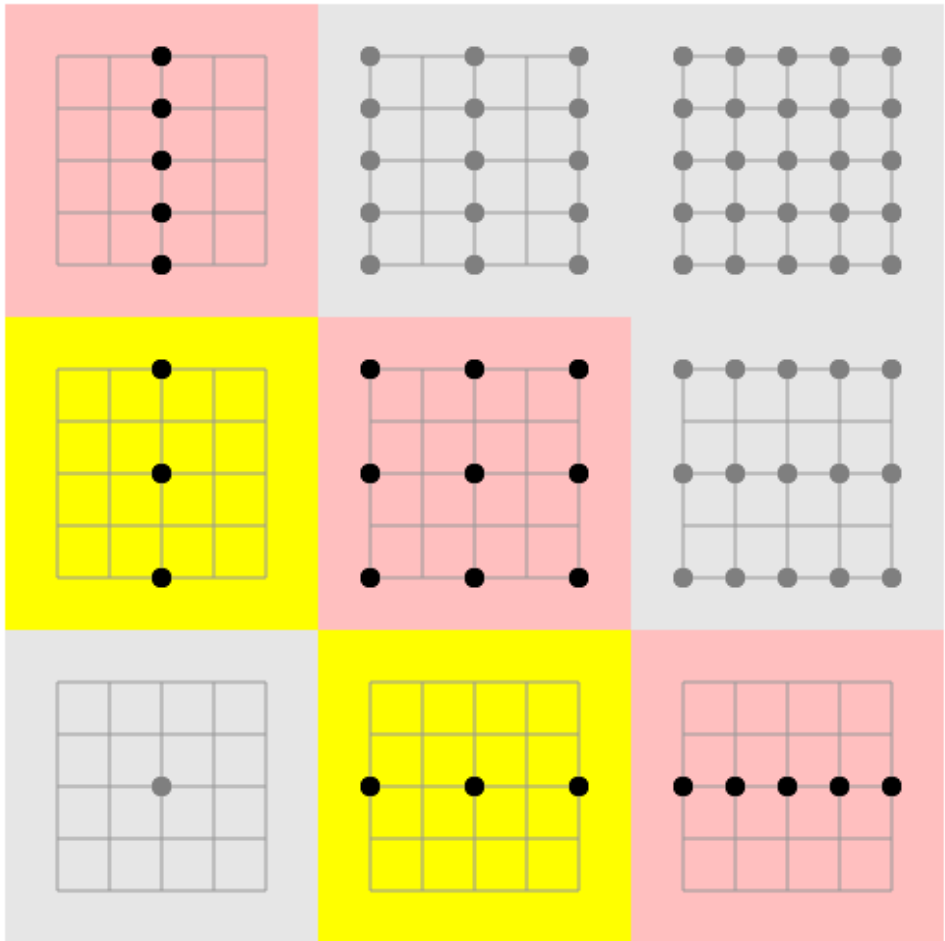
(For higher dimensions, we don't just have +1 and -1 as coefficients, but they will still be combinatorial coefficients.)

The Quadrature Points of $\mathcal{A}(2,2)$



Creating $\mathcal{A}(2,2)$

$$\mathcal{A}(2,2) = CC^2 \otimes CC^0 + CC^1 \otimes CC^1 + CC^0 \otimes CC^2 - CC^1 \otimes CC^0 - CC^0 \otimes CC^1$$



$\mathcal{A}(2,2)$ as good as $C(5) \otimes C(5)$

The picture correctly suggests that the Smolyak combination of the 5 lower order grids is (essentially) as precise as the 5x5 product grid in the upper right corner.

But the Smolyak grid $\mathcal{A}(2,2)$ uses 13 points, the product grid $C(5) \otimes C(5)$ uses 25.

The Smolyak definition chooses a collection of lower order product grids that capture the information necessary to approximate all the monomials of interest, and then combines this information correctly to produce a good integral estimate.

(Because of the nesting choice we made, the precision results are not so neat for higher levels...)

Efficiency

The space of **N**-dimensional polynomials of total degree **P** or less has dimension $\text{Pdim}(P,N) = \begin{pmatrix} P+N \\ N \end{pmatrix} \approx \frac{N^P}{P!}$.

For instance, to count the polynomials in dimension $N = 3$ with total degree $P \leq 2$, namely $1, x, y, z, x^2, xy, xz, y^2, yz, z^2$, our formula yields $\begin{pmatrix} 2+3 \\ 3 \end{pmatrix} = 10$.

The theoretical limit for large **N** shows that a Clenshaw-Curtis Smolyak rule that achieves precision **P** uses $M \approx \frac{(2N)^P}{P!}$ points; we do not see an exponent of N in the point count, which does occur for a standard product rule approach. This demonstrates that the Smolyak approach manages to avoid the exponential curse of dimensionality.

Efficiency

For practical examples, consider a CC rule of level 8, which achieves precision 17. We would hope that the number of points M would roughly coincide with the dimension $Pdim$ of the polynomial space, that is, the number of monomials of degree $P = 17$ or less that we can integrate precisely:

N	2	5	10	15	20
M	1,537	51,713	2,320,385	33,647,617	261,163,009
Pdim	342	26,334	8,436,285	565,722,720	15,905,368,710

The number of monomials outpaces the Smolyak point count!

But it's clear our precision 17 rule can't go to higher dimensions. *(This is why we need an* **anisotropic** *version of the method.)*

Relation with Stochastic Variables

In stochastic problems, the type of random variable used to model uncertainty determines the quadrature rule that should be used to handle that variable. Some common choices include:

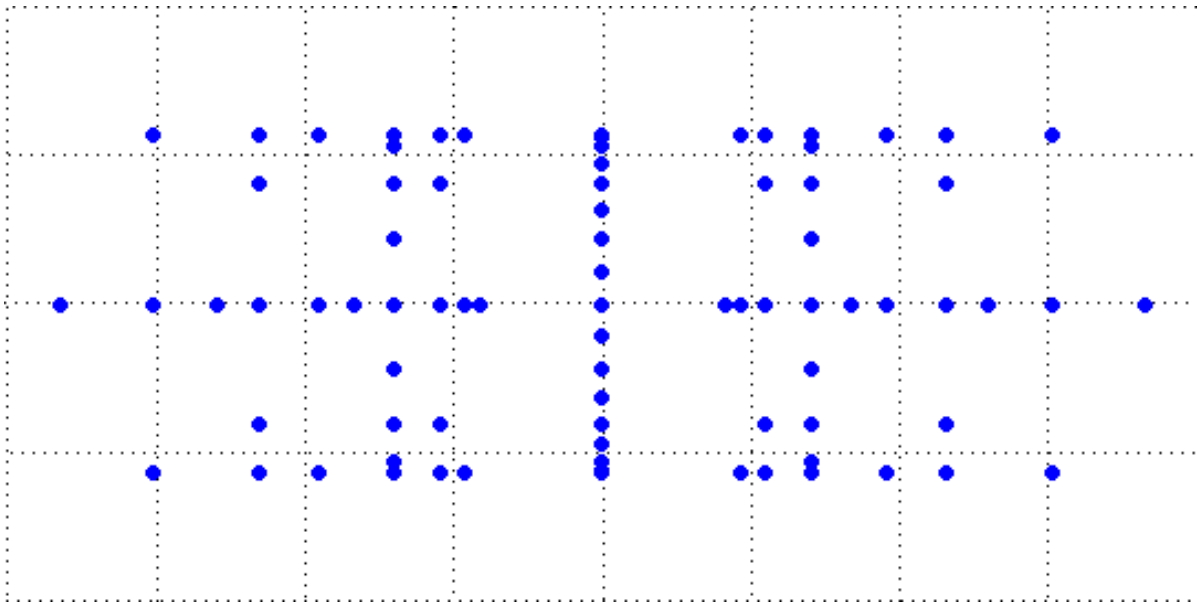
Distribution	Domain	Weight	Quadrature
Uniform	$[-1,+1]$	1	Gauss-Legendre or Clenshaw-Curtis
Gaussian	$(-\infty,+\infty)$	$e^{-(x-\alpha)^2/\beta^2}$	Gauss-Hermite
Gamma	$[0,+\infty)$	$e^{-\alpha x}$	Gauss-Laguerre
Beta	$[-1,+1]$	$(1-x)^\alpha(1+x)^\beta$	Gauss-Jacobi

The Smolyak procedure can handle all these cases, simply by using the appropriate 1D rules. If several distributions occur in the same problem, the procedure selects the appropriate rule for the corresponding dimension.

Mixed Products

The Smolyak procedure allows each dimension to be treated independently in whatever manner is appropriate for that variable.

The sparse grid technique simply combines the given rules, without requiring that they involve the same 1D domain or weight function.



Level 4 Rule, Hermite in X, Clenshaw Curtis in Y
http://people.sc.fsu.edu/∼jburkardt/m_src/sparse_grid_mixed_dataset/sparse_grid_mixed_dataset.html

Precision of a General CC Sparse Grid

Novak and Ritter show that for sparse grids based on the 1D Clenshaw Curtis rule, the precision is related to the level by:

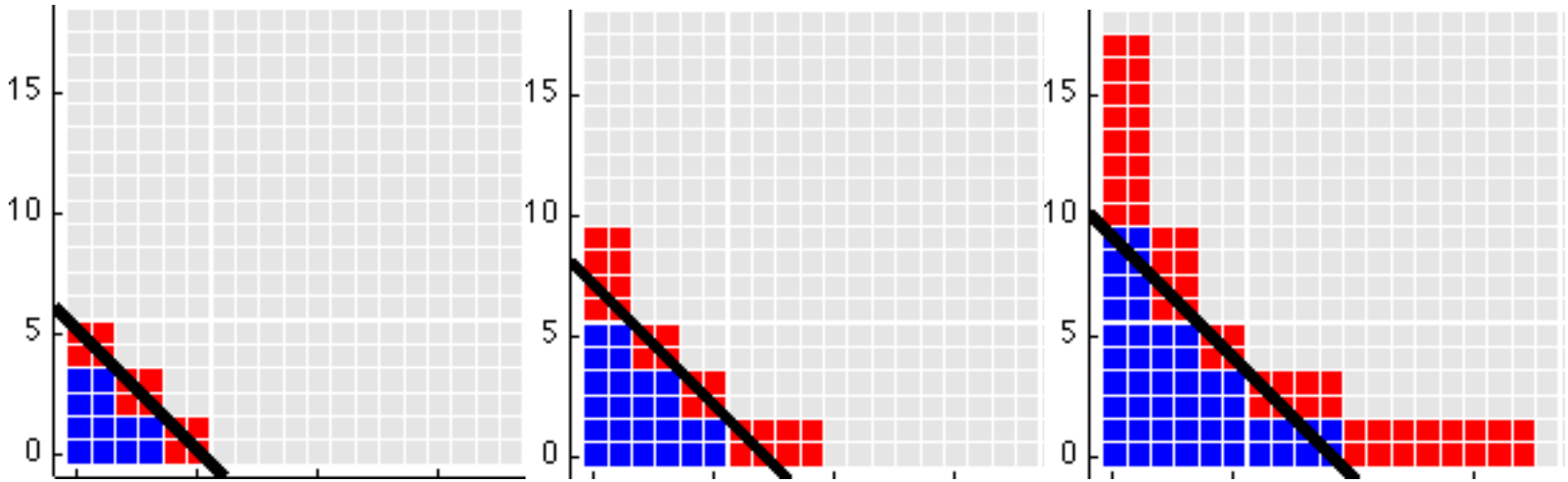
$$P = 2 * L + 1$$

The first sparse grid (L=0) picks up constants and linears, the second adds quadratics and cubics, and so on. This means that our family of sparse grids has a predictable precision behavior based on the level, regardless of dimension.

If our CC-based sparse grids were perfectly efficient, the level L rule would pick up exactly the monomials up to precision P and no more. A precision graph will show that we do better than a product rule, but still have some inefficiency.

What Monomials are Precisely Integrated?

Here are the precisions of $\mathcal{A}(2,2), \mathcal{A}(3,2), \mathcal{A}(4,2)$. The diagonal black line "fences off" the monomials of degree 2*L+1, which the sparse grid must integrate precisely. With increasing level, the rules spill over the fence, suggesting some inefficiency.



(The red squares are monomials just added on this level.)

Reducing the Nesting Overhead

Our sparse grid of level L will have the precision $P = 2 * L + 1$ as long as our family of 1D rules has **at least** that same precision sequence:

```
L                0  1  2  3  4  5  6  7   8   9   10...
P minimal:       1  3  5  7  9 11 13 15  17  19   21...
```

But our 1D rules double the number of points with each level, in order to effect nesting...which we want to control the point count.

```
P supplied:      1  3  5  9 17 33 65 129 255 513 1025...
```

We could keep nesting but reduce the cost of doubling, by reusing a 1D rule if it satisfies the precision requirement.

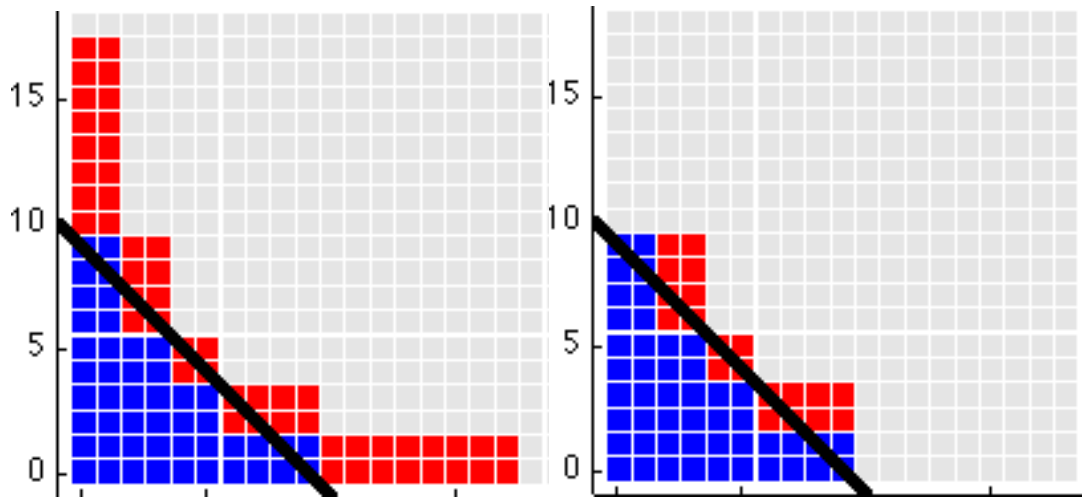
```
P suggested:     1  3  5  9  9 17 17  17  17  33   33...
```

---

http://people.sc.fsu.edu/∼jburkardt/cpp_src/sgmg/sgmg.html

Standard / Slow Growth versions of $\mathcal{A}(4, 2)$

Using the slower growth strategy preserves nesting and our overall precision requirement while cutting down on the order of the finest grid, and hence the number of function evaluations.



Both rules achieve precision $P = 2 * L + 1 = 9$;
the standard rule uses 65 points, the slow rule 49.

---

http://people.sc.fsu.edu/∼jburkardt/presentations/sgmga_ccs.pdf

Anisotropic Growth

The standard formulation of the Smolyak formula treats each dimension equally.

In the precision diagrams, this corresponds to using a diagonal line that indicates the total polynomial precision we expect on each successive rule.

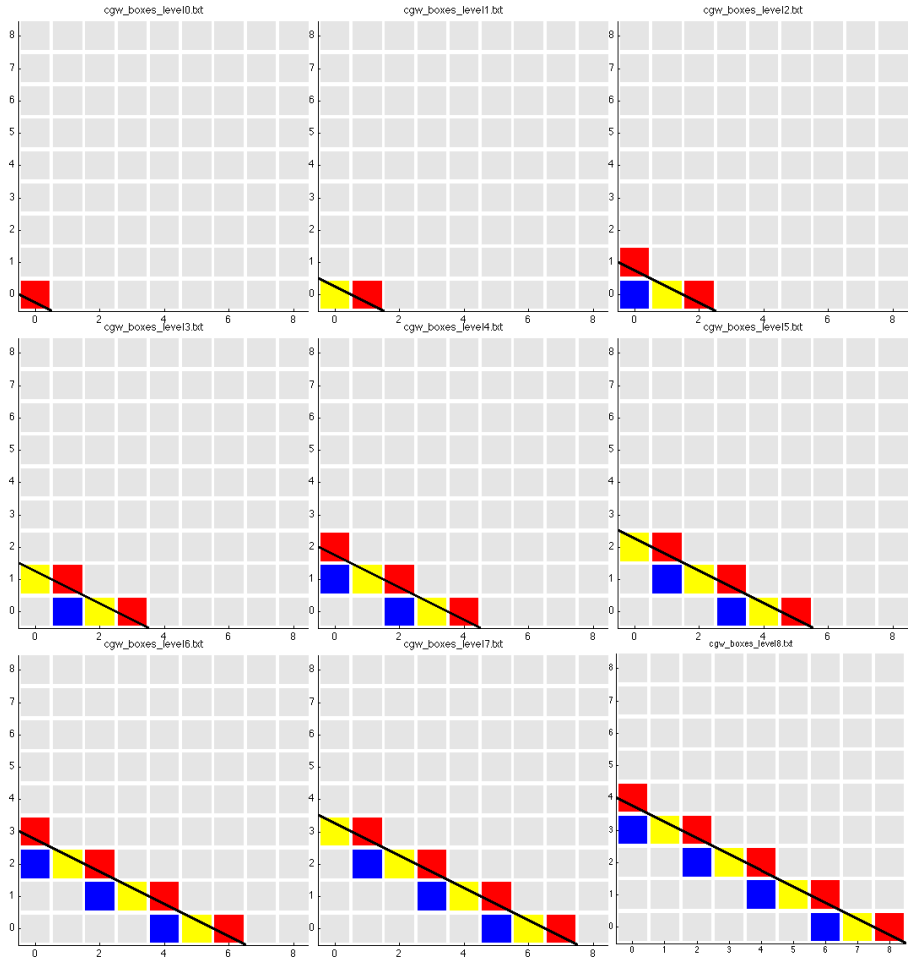
We might now in advance, or discover adaptively, that some coordinates are inactive, and that some are relatively more active than others. It is easy to modify the Smolyak formula so that, with each successive rule adds significantly more precision (more monomials) in the most active dimensions.

In this case, our precision diagram (in 2D) would involve a sequence of nondiagonal lines whose slope corresponds to the level of anisotropy.

http://people.sc.fsu.edu/∼jburkardt/cpp_src/sgmga/sgmga.html

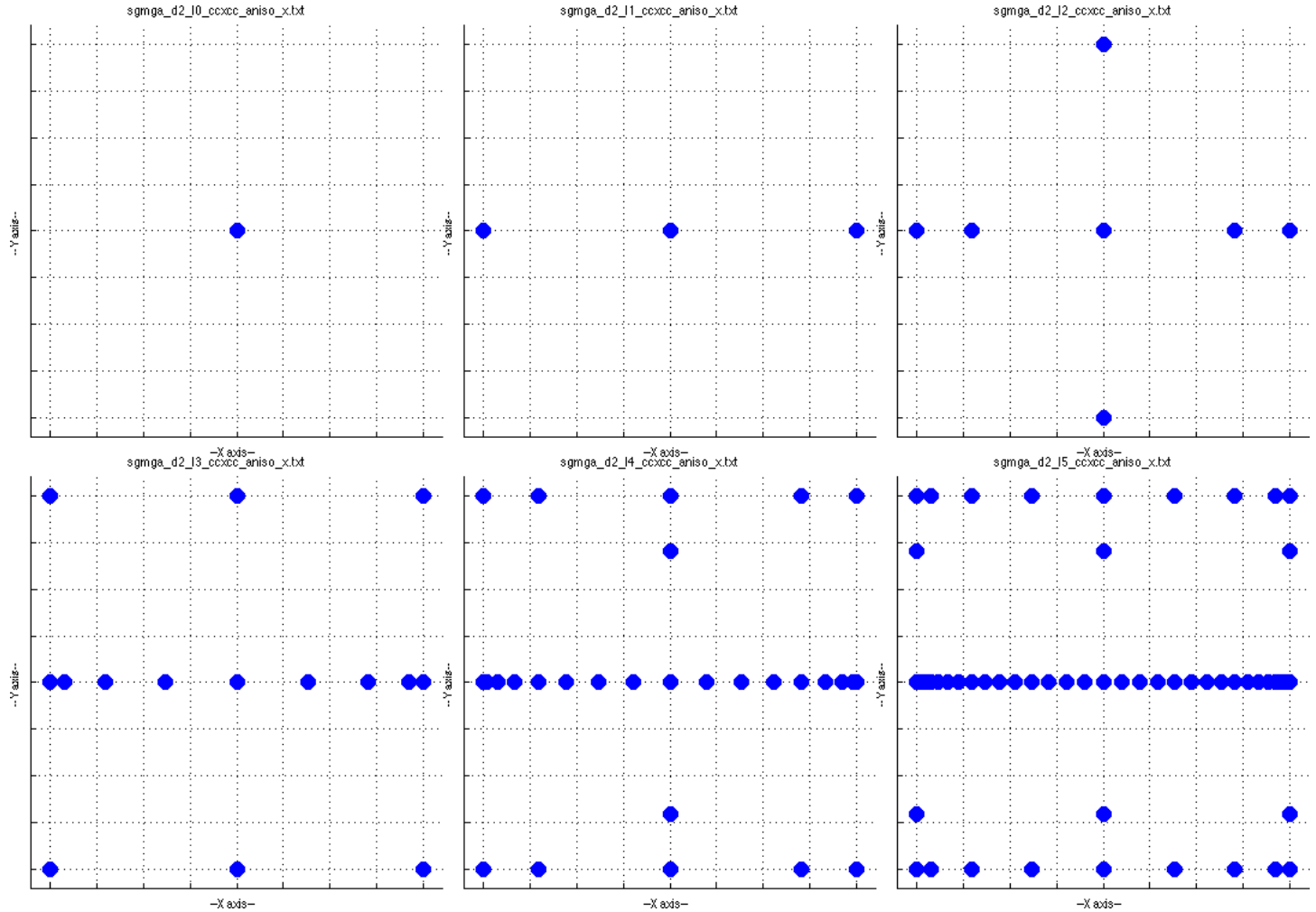
Anisotropic Growth

Here, as the level increases, we increase the precision in the x monomials twice as rapidly. (Red = +1, Yellow = 0, Blue = -1).



Anisotropic Growth

The x preference is obvious in the actual quadrature grids.



Adaptive Grid

It turns out that the Smolyak formula can also handle very general cases suitable for adaptivity.

A given 2D precision diagram is a monotonically decreasing "cityscape" of squares representing monomials we have captured.

Each "corner" of the cityscape represents a monomial we didn't capture, but which could be captured by including a single new product rule to the Smolyak formula.

Thus, an adaptive approach could start with a given precision diagram, then consider each corner and compute its potential new contribution, and take those which rise above a given tolerance in magnitude.

Naturally, the search approach itself becomes expensive as the dimension gets very high.

---

http://people.sc.fsu.edu/~jburkardt/cpp_src/sandia_sgmgg/sandia_sgmgg.html

Adaptive Grid

Our grid cells represent product rules included in the Smolyak grid. 2D coefficients are +1 (red), 0 (yellow), or -1 (blue).

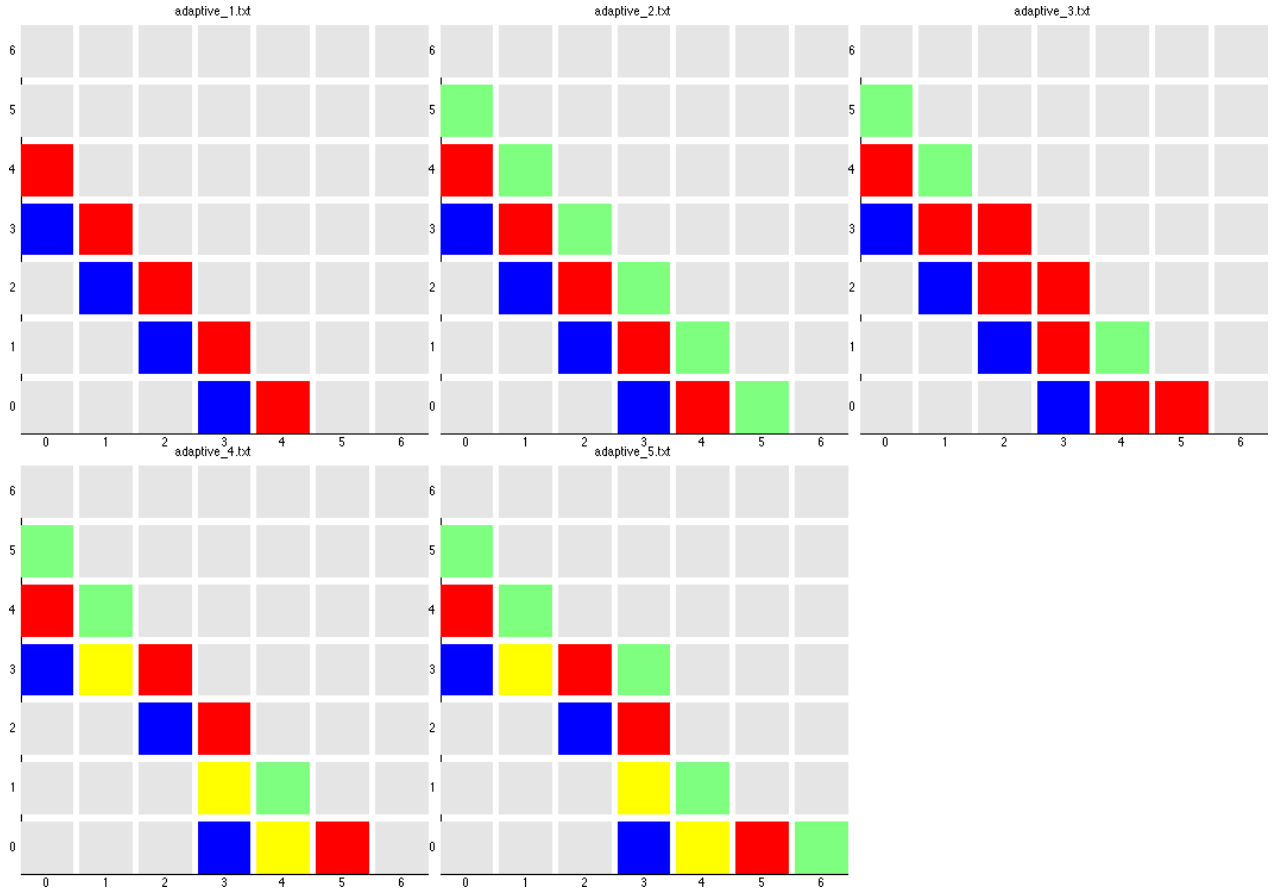
Normally, the next level adds all green cells. But any cell with immediate left and down red neighbors can be added. The adaptive algorithm selects cells to be incorporated. A natural criterion adds cells that make "significant" changes to the estimate.

After adding cells, we recompute coefficients, and discover that some blue cells drop out, some red cells become blue, some cells become yellow (in the grid, but with a zero coefficient.)

Once the grid has been recomputed, we notice that there are a few new green cells to be included in our next adaptive search.

Adaptive Grid

```
Starting grid | Green Cells Added | Some Greens go Red
--------------+-------------------+-------------------
Recompute grid| Update Green Cells|
```



Analysis of Clenshaw Curtis Sparse Grids

Refer to **Novak and Ritter*** for details about the construction of rules based on the Clenshaw Curtis / Chebyshev rule, error bounds for the approximation of integrals of smooth functions, estimates for the growth in the number of function evaluations with increasing level or dimension, and a proof that the rules are exact for all monomials up to a given degree.

The authors also indicate conditions under which other 1D quadrature rules can be used to construct Smolyak rules of comparable precision.

**High dimensional integration of smooth functions over cubes*,
Numerische Mathematik, volume 75, pages 79-97, 1996.

Function Evaluations Versus Accuracy

.

We estimate the integral of $f(x)$ over an N-dimensional hypercube $[a, b]^N$ using M points or function evaluations.

The "cost" of the estimate is M. If an estimate is not satisfactory, it's important to know how fast the error is likely to go down if we increase M and try again.

For the Monte Carlo method, the rate of error decay is known to be $O(\frac{1}{\sqrt{M}})$. The rate is independent of spatial dimension N, and essentially independent of the smoothness of $f(x)$.

Compare Monte Carlo and sparse grid values of M and accuracy.

Sparse Grids Require Smoothness!

Let $f(x)$ be the characteristic function of the unit ball dimension N=6:

M	SG Estimate	SG Error	:	MC Estimate	MC Error
1	4.000	1.167	:	0.00000	5.16771
13	64.000	58.832	:	0.00000	5.16771
85	-42.667	-47.834	:	3.01176	2.15595
389	-118.519	-123.686	:	4.77121	0.39650
1457	148.250	143.082	:	5.16771	0.01555
4865	-24.682	-29.850	:	5.41994	0.25226

Can you see why negative estimates are possible for the sparse grid, even though the integrand is never negative?

Sparse grids need smooth integrands; and because sparse grids use extrapolation, they are liable to unpleasant errors otherwise.

---

http://people.sc.fsu.edu/∼jburkardt/m_src/quadrature_test/quadrature_test.html, problem #18
http://people.sc.fsu.edu/∼jburkardt/m_src/ball_volume_monte_carlo/ball_volume_monte_carlo.html

MC Quadrature Can be Slow

Monte Carlo doesn't diverge, but look how hard we have to work to get three places of accuracy for the characteristic function of the unit ball dimension N=6.

M	MC Estimate	MC Error
1	0.00000	5.16771
32	6.00000	0.83228
1,024	4.81250	0.35521
32,768	5.39063	0.22291
1,048,576	5.18042	0.01271
33,554,432	5.16849	0.00077
∞	5.16771	0.00000

Should we want one more digit of accuracy, we can expect to need **100 times as many points** $\approx$ 3.3 billion points.

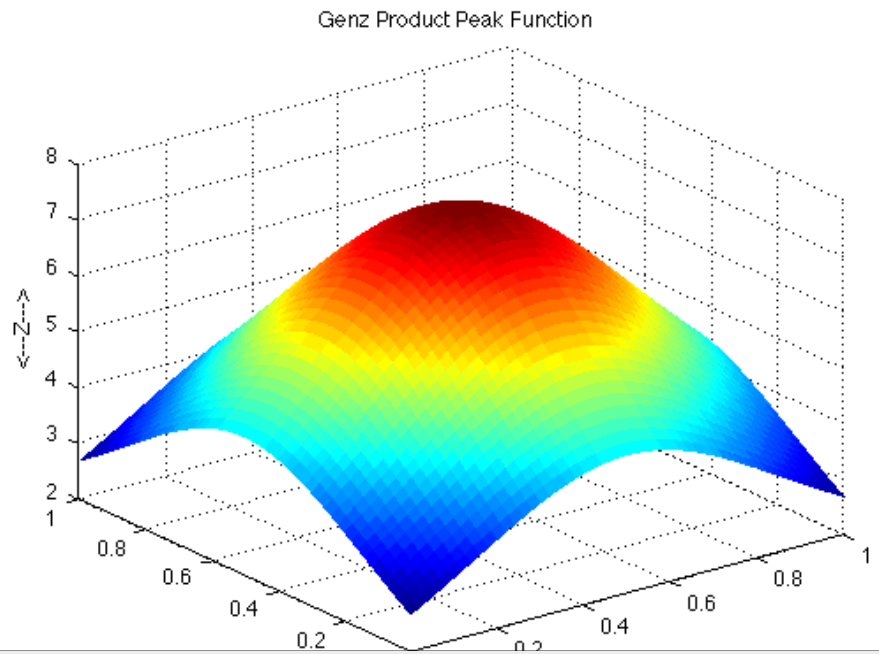
---

http://people.sc.fsu.edu/~jburkardt/m_src/ball_volume_monte_carlo/ball_volume_monte_carlo.html

Genz Product Peak Test, Dimension N=6

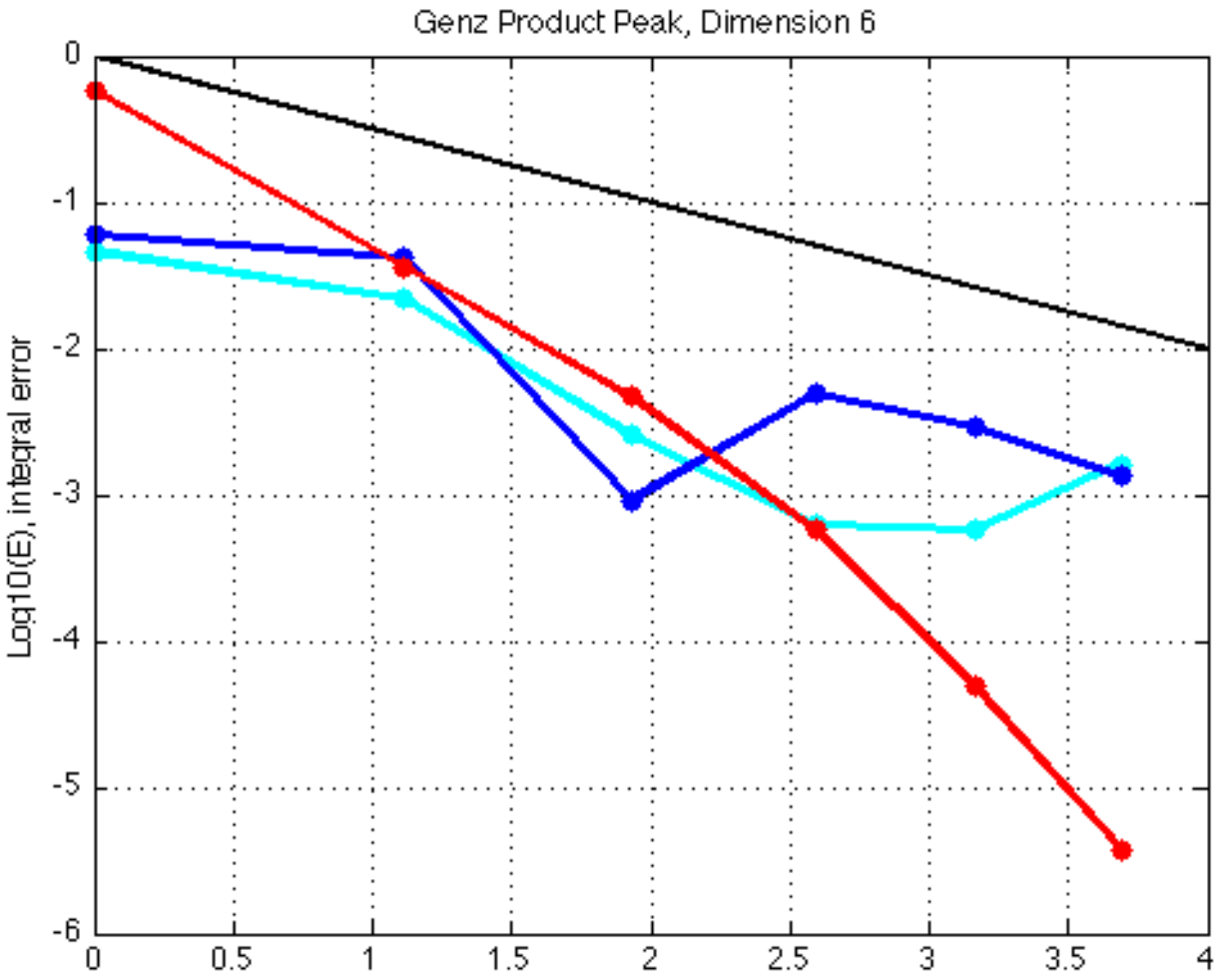
Alan Genz provided six high dimensional test integrals;
The **product peak function** is defined on the unit hypercube, with given C and Z vectors, and is smooth:

$$F(X) = \frac{1}{\prod_{i=1}^{N}(C_i^2 + (X_i - Z_i)^2)}$$



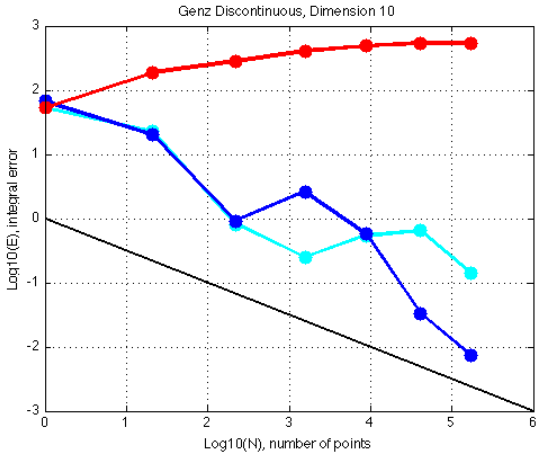
Genz Product Peak Test, Dimension N=6

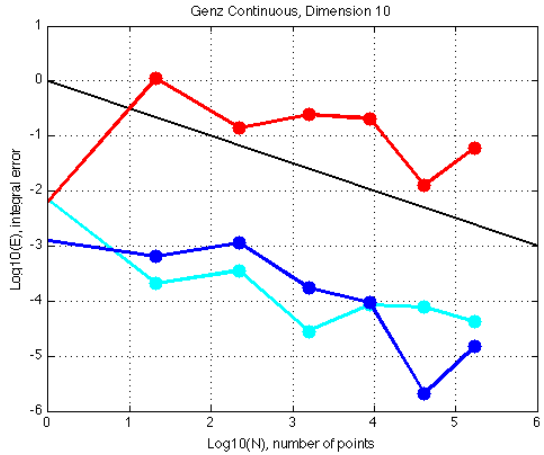
Red: Sparse grid estimate
Blue & Cyan: MC estimates
Black: Expected MC Rate of Decrease

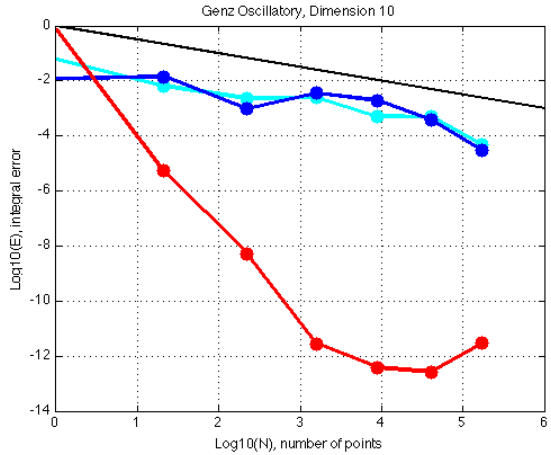


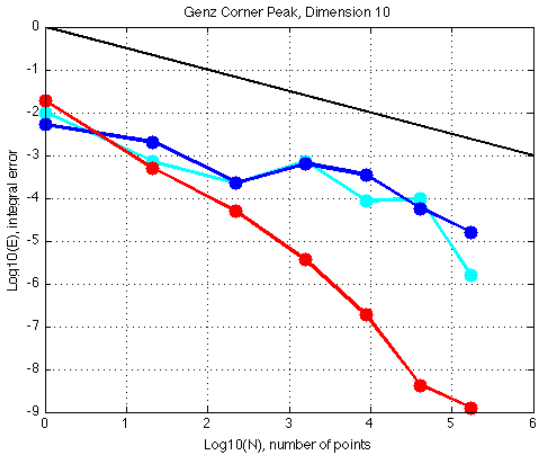
Genz Tests, dimension N=10

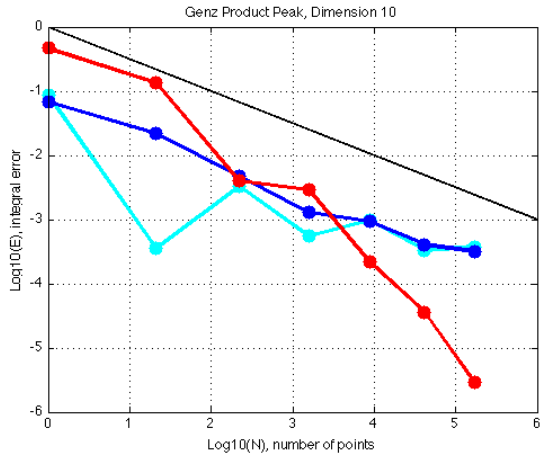
Discontinuous, Continuous, Oscillatory
Corner Peak, Product Peak, Gaussian
(sparse grid estimate in red)

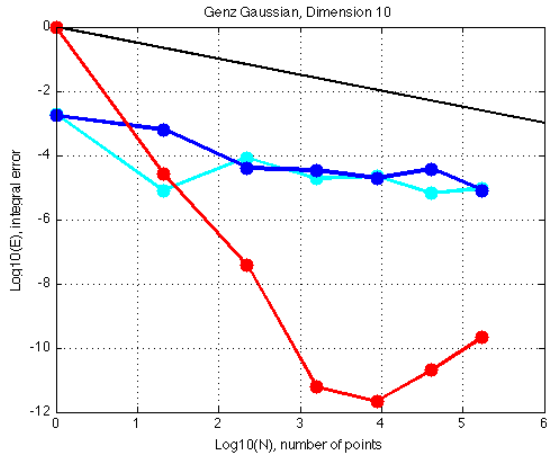












The Poisson Equation

Let's return to the stochastic Poisson equation considered earlier:

$$-\nabla \cdot (a(x, y; \omega)\, \nabla u(x, y; \omega)) = f(x, y)$$

Our integration problem seeks the expected value of $u(x, y; \omega)$, assuming we have a probabilistic model for the stochastic influence.

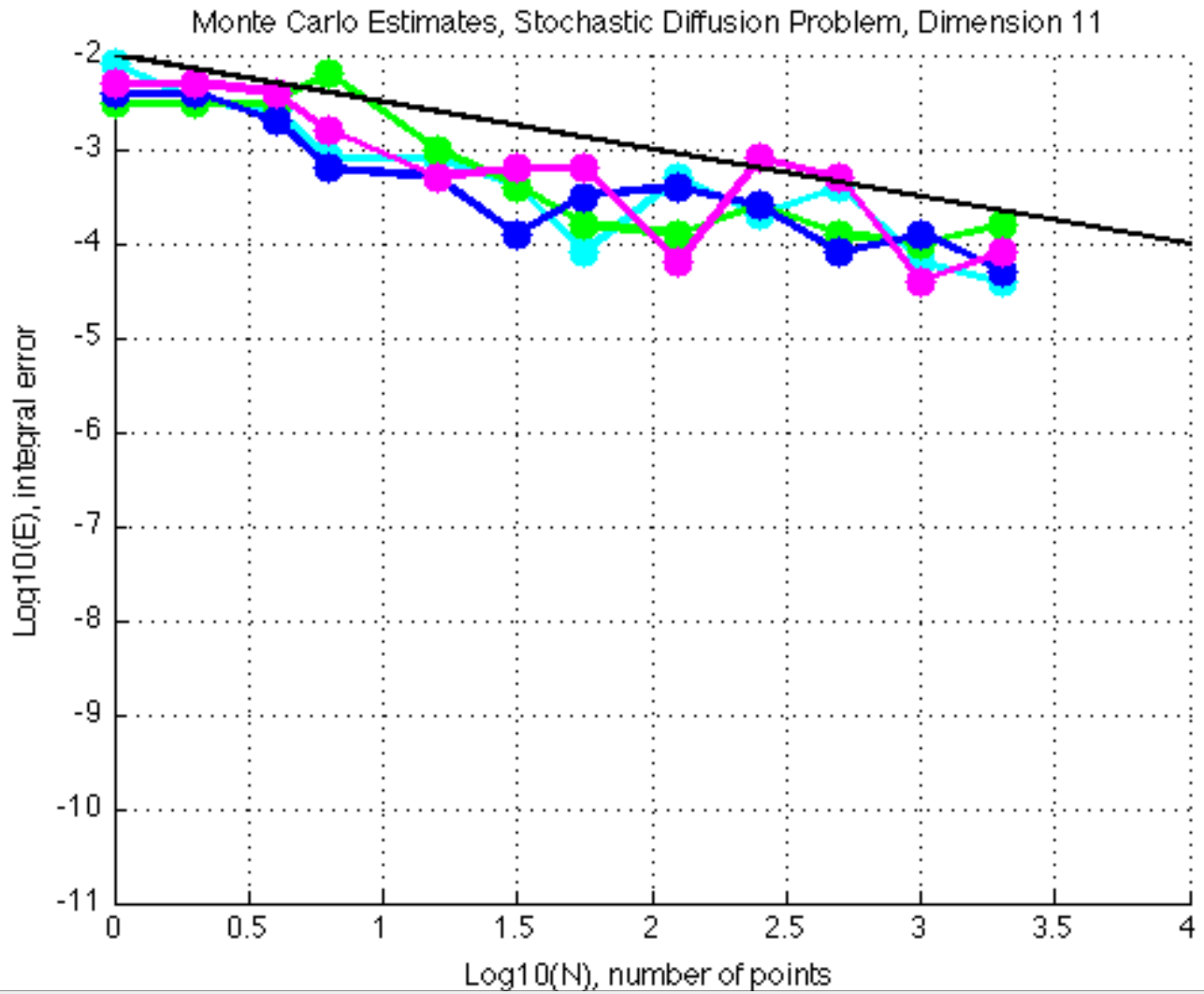
Monte Carlo: select a random set of parameters ω according to $pr(\omega)$, solve the Poisson equation for u, and average.

Sparse grid: choose a level, defining a grid of ω values, solve the Poisson equation for u, multiply by the probability, and take a weighted average.

Clayton Webster, *Sparse grid stochastic collocation techniques for the numerical solution of partial differential equations with random input data*, PhD Thesis, Florida State University, 2007.

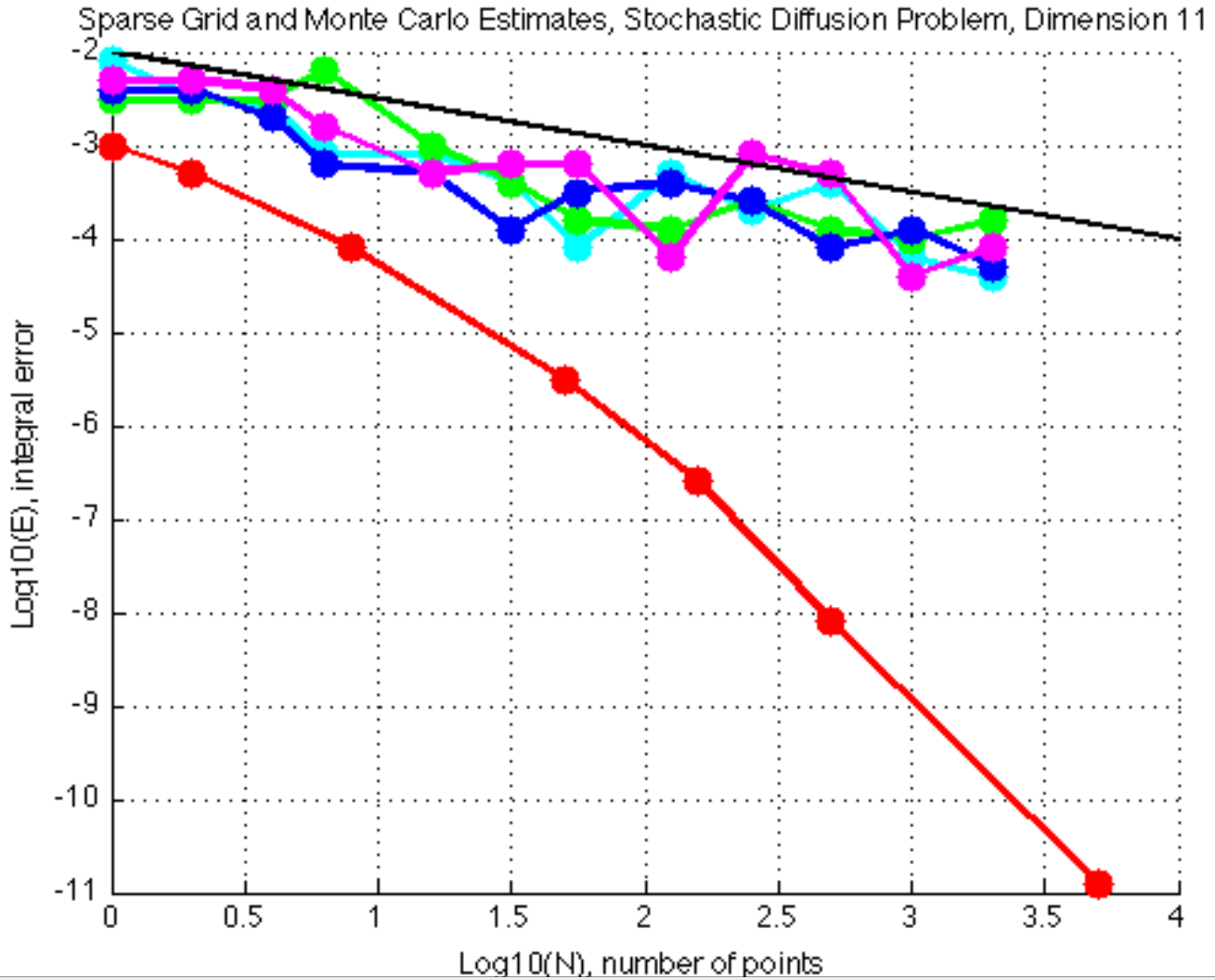
Four Monte Carlo Estimates

The black line is the Monte Carlo trend.



Sparse Grid Versus Monte Carlo

The sparse grid estimates converge rapidly.



Remarks

For the stochastic diffusion problem, $u(x, y; \omega)$ has a very smooth dependence on the perturbations in ω.

For this reason, a sparse grid can sample the solution for a small set of perturbations ω and accurately estimate the expected value.

If we had a code to solve the original Poisson equation for a given conductivity field, the sparse grid procedure simply needs to call that unmodified code with different conductivities.

This is why sparse grids are called a nonintrusive method. Other procedures for dealing with uncertain or stochastic influences may require extensive changes, new variables, and a larger coupled system to solve.

Summary

Collocation is a general framework a wide range of nonintrusive methods that can be thought of as analyzing the stochastic problem by selecting values for the free components and examining the resulting collection of solutions.

A typical quantity of interest is the expected value of the solution.

The Monte Carlo / QMC methods compute this immediately by averaging.

Interpolatory methods use a polynomial model, which, for smooth problems, finds a very accurate result quickly.

For high dimensions, a straightforward product rule approach becomes too costly. The Smolyak procedure shows how to assemble a set of low-degree product rules to achieve the precision of a product rule at much lower cost.

Depending on the problem, it is easy to modify to Smolyak procedure to handle cases where the stochastic data has various probabilistic distributions; where some components are idle and others vary in importance, or in which the behavior of the components must be discovered adaptively.

Summary

In this talk, we have seen plots that suggest that collocation methods can converge more rapidly than Monte Carlo, and more cheaply than standard product grid methods, for the appropriate class of functions.

In the next discussion, we will look more carefully at this question of **convergence analysis** and cost.

The collocation method is an example of a **nonintrusive** method; the next discussion will look at the features and costs of intrusive and nonintrusive methods.

Finally, most partial differential equations include physical parameters such as viscosity, volatility, permeability, conductivity, impedance, or initial values, which influence the observable or computable state solution. The **inverse problem** asks to what extent we can determine the value of the parameters from the observed solution data. The next discussion will consider this inverse problem in the case where the PDE includes stochastic influences as well.