High Dimensional Sparse Grids
(stalking the wild integral)

John Burkardt
School of Computational Science
Florida State University

.....

Science for Lunch Bunch
Pittsburgh Supercomputing Center
27 June 2007

.....

https://people.sc.fsu.edu/~jburkardt/presentations/...
sparse_2007_psc.pdf

February 10, 2024

Let the Exploration Begin!

Integration

Integration is a natural expression of physical laws.

A complicated thing is understood by **adding** tiny components.

Integration computes the area under a curve.

$$G[a,b] = \int_a^b f(x)\,dx$$

It can also be seen as an *averaging* process.

$$\overline{f(x)} = \frac{\int_a^b f(x)\,dx}{(b-a)}$$

Integration: Multiple Dimensions

$$Volume[pool] = \int_c^d \int_a^b depth(x, y)\, dx\, dy$$

Integration: Multiple Dimensions: More than 3!

Mathematicians left 3D space long ago!

- Financial mathematics: 30D or 360D
- ANOVA decompositions: 10D or 20D
- Queue simulation (expected average wait)
- Stochastic differential equations: 10D, 20D, 50D
- Particle transport (repeated emission/absorption)
- Light transport (scattering)
- Path integrals over a Wiener measure (Brownian motion)
- Quantum properties (Feynman path integral)

Integration: No Formulas for Interesting Problems!

Freshman memorize "antiderivatives" of formulas $f(x)$.

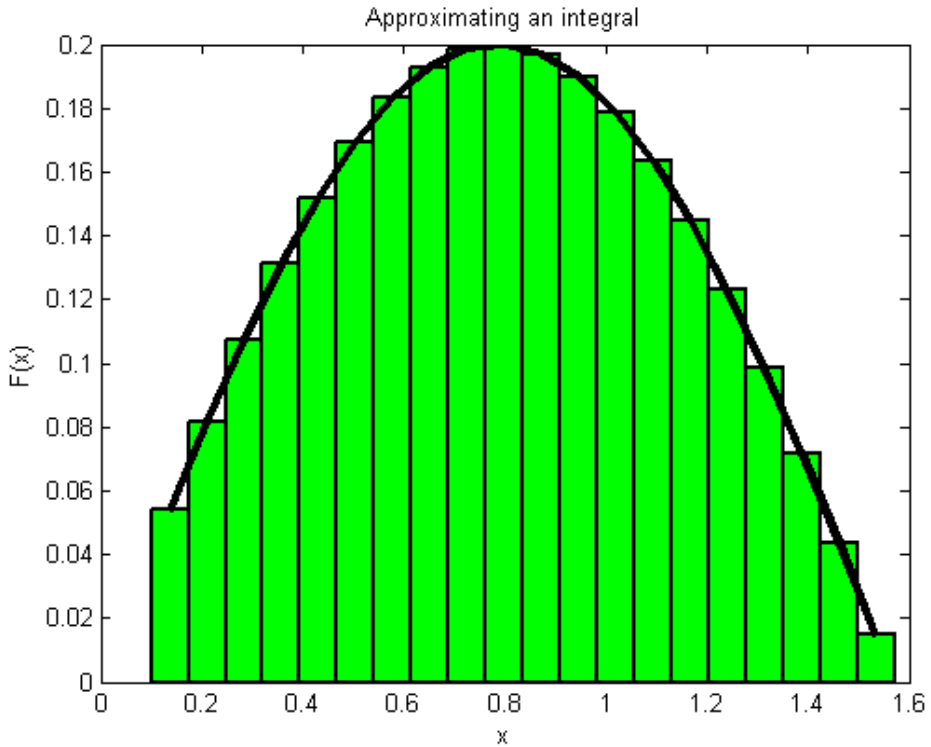
$$\int x^3 \, dx = \frac{x^4}{4} + C$$

But most formulas have no antiderivative!

And most things we want to integrate are not formulas!

$$\int_\Omega \nabla \mathsf{v}^h \cdot \nabla \psi_i + \nabla p^h \, \psi_i \, d\Omega = ?$$

1D Quadrature: Approximating an Integral



Quadrature allows us to estimate integrals.

This integration region is 1D. Similar methods apply in 2D (the swimming pool) and higher dimensions.

1D Quadrature: Monte Carlo

The *Monte Carlo* algorithm views the integral as an average.

- "Choose" N random points N points x_i;
- Evaluate each $f(x_i)$;
- Average the values.

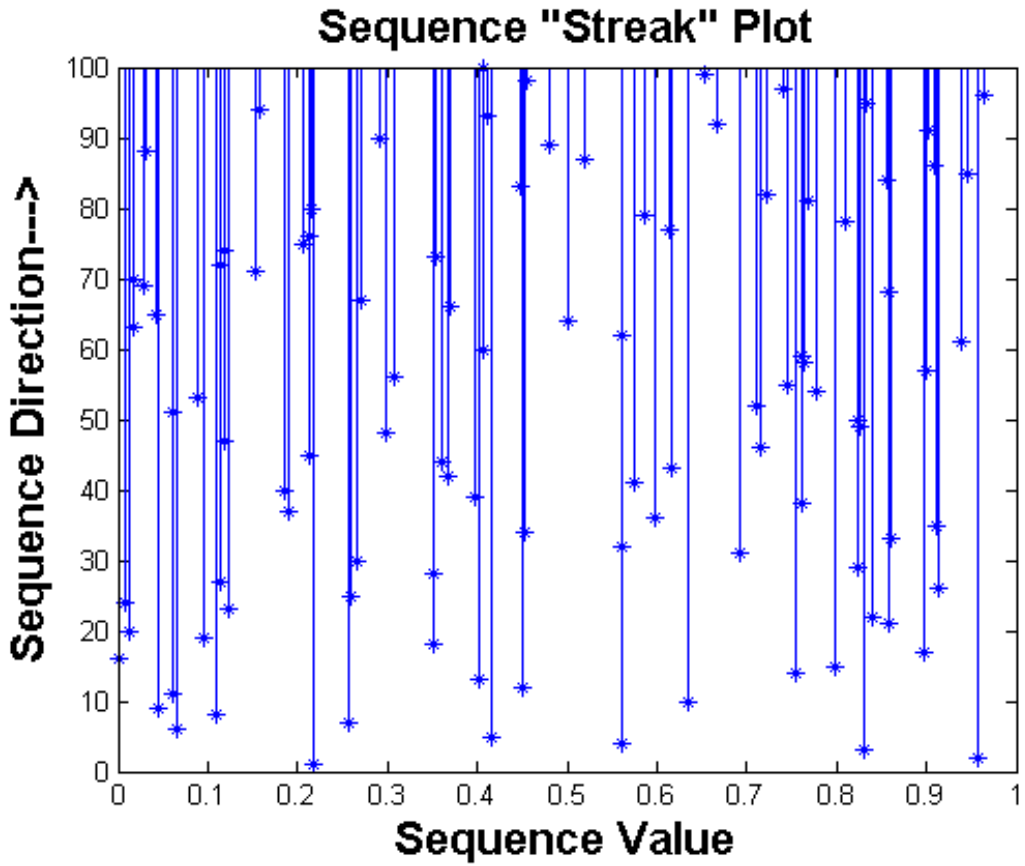
1D Quadrature: Monte Carlo

To improve an MC estimate, increase **N**, the size of your sample.

The Law of Large Numbers says that convergence will be like $\sqrt{N}$. To reduce the error by a factor of 10 (one more decimal place) requires 100 times the data.

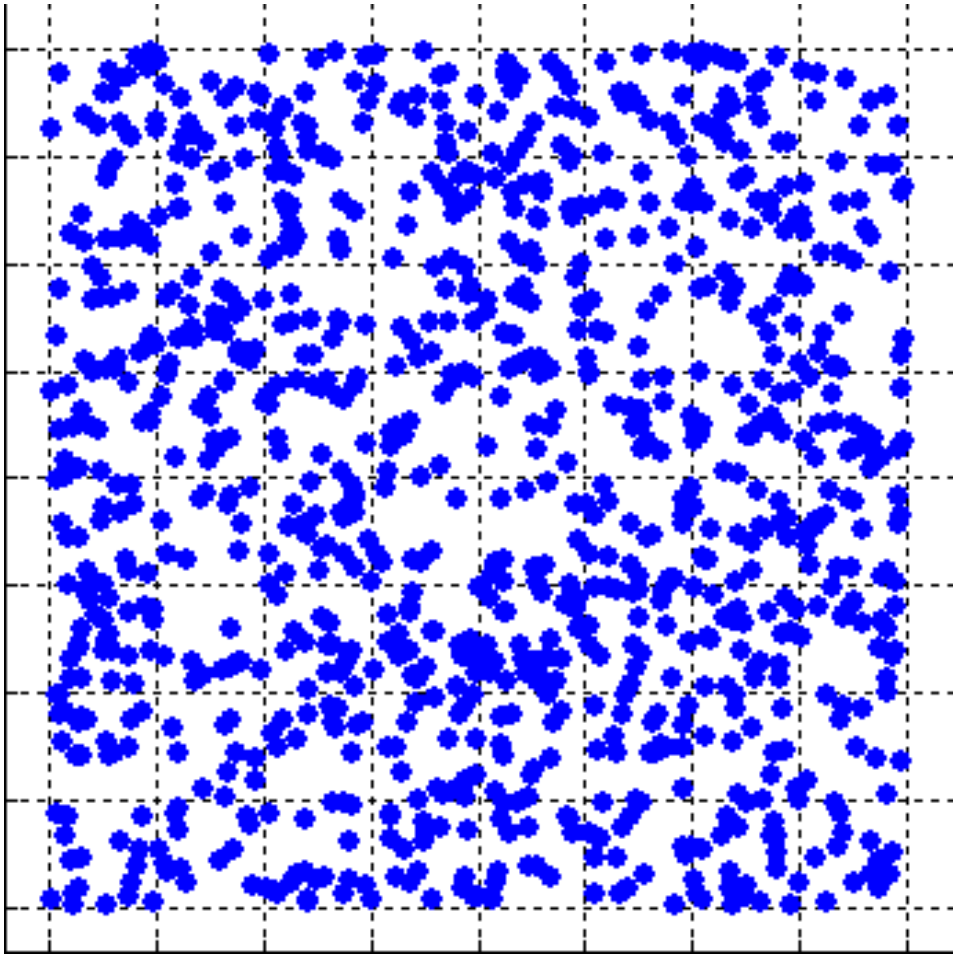
- If more accuracy needed, current values can be included;
- Accuracy hampered because of large "gaps" in sampling.
- Accuracy improvement rate is independent of spatial dimension.

1D Quadrature: Monte Carlo



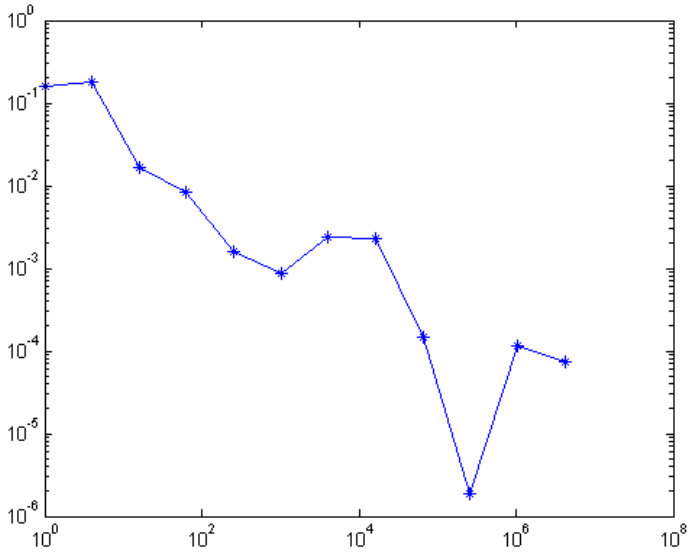
Notice the clustering and gaps.

2D Quadrature: Monte Carlo



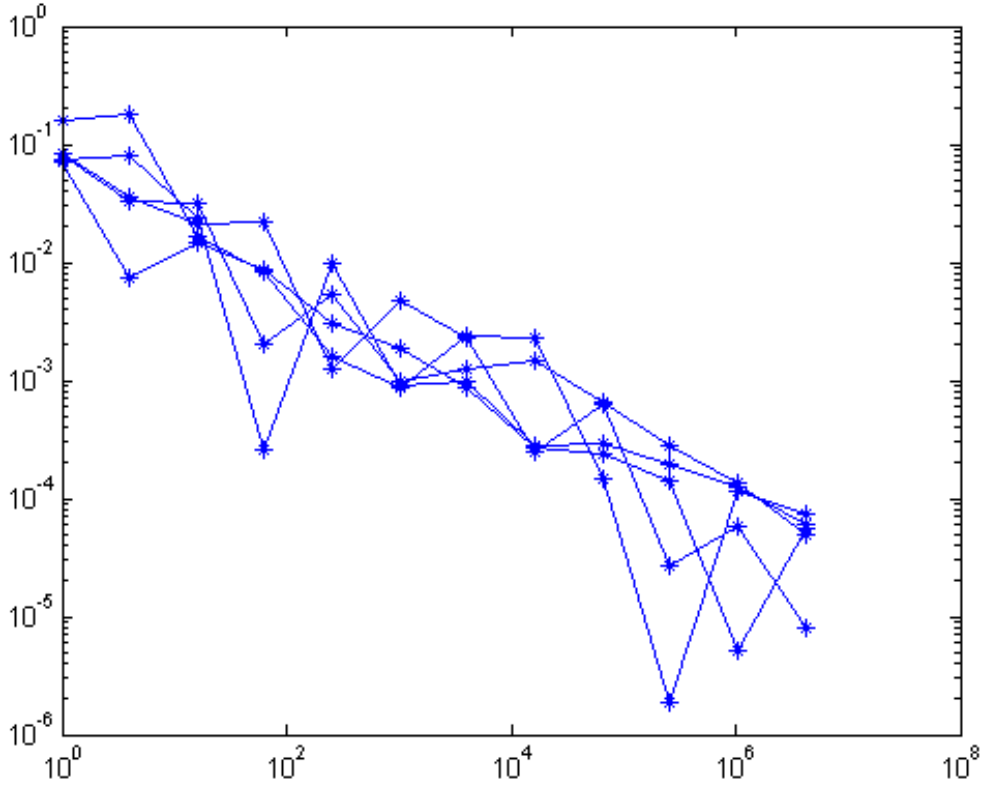
Notice the "gaps"

6D Quadrature: Monte Carlo Error



N	Estimate	Error
1	0.796541	0.160759
16	0.652621	0.016838
256	0.637351	0.001569
65536	0.635926	0.000144
4194304	0.635856	0.000074
∞	0.635782	0.0000

6D Quadrature: Monte Carlo Error



If we try five times, we get five different sets of results.

Alternatives to Sampling?

Sampling methods treat "misbehaving" functions the same as well-behaved ones.

The fact that it works "the same" for all functions can be a **strength** or a **weakness**

Sampling ignores the extra information in a well-behaved function.

If $f(x)$ is well-behaved, we can get more accuracy faster.

1D Quadrature: Interpolatory

Is $f(x)$ approximately a sum of monomials (powers of x)?

$$f(x) \approx 4.5 + 6.3x + 0.8x^2 + 2.1x^3 + 0.7x^4 + \ldots$$

If so, the beginning of the formula can be determined and integrated exactly.

This assumption is **not true** *for step functions, piecewise functions, functions with poles or singularities or great oscillation.*

1D Quadrature: Interpolatory

To find the initial part of the representation, sample the function.

Evaluating at one point can give us the constant.

$$f(x) \approx 4.5... + 6.3x + 0.8x^2 + 2.1x^3 + 0.7x^4 + ...$$

A second evaluation gives us the coefficient of x:

$$f(x) \approx 4.5 + 6.3x... + 0.8x^2 + 2.1x^3 + 0.7x^4 + ...$$

Evaluating at N points gives the first N coefficients.

1D Quadrature: Interpolatory

An approximate formula can be integrated exactly.

With N samples, we can integrate the first N monomials,

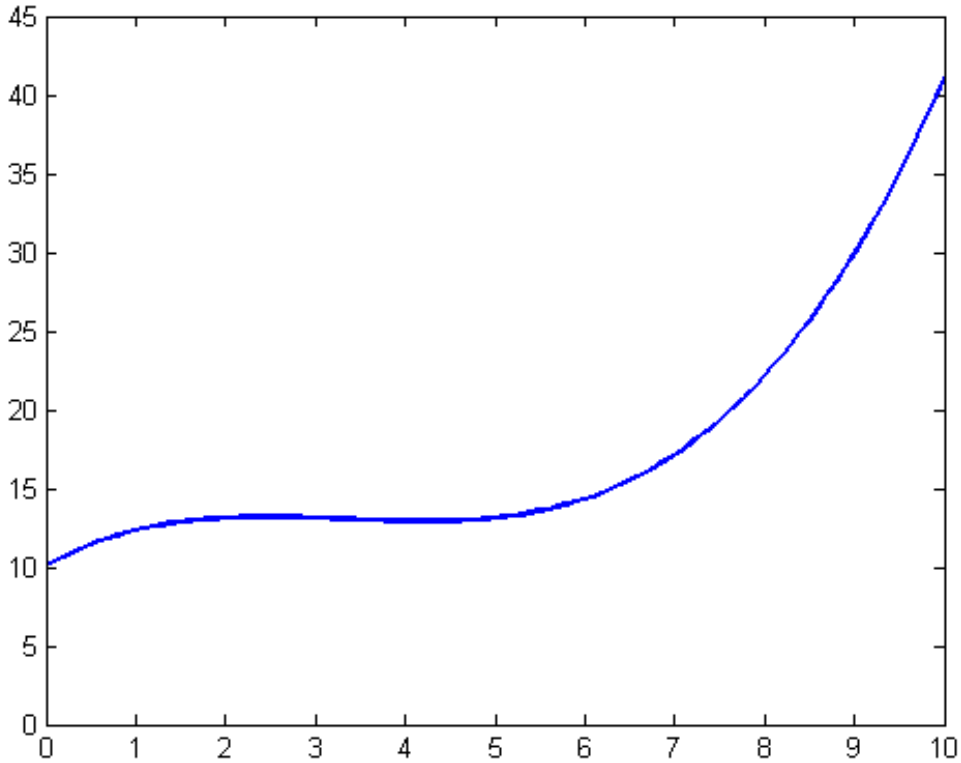
$$1, x, x^2, ..., x^{N-1},$$

and all functions made up of them.

The error behaves like h^N, where h is the spacing between sample points.

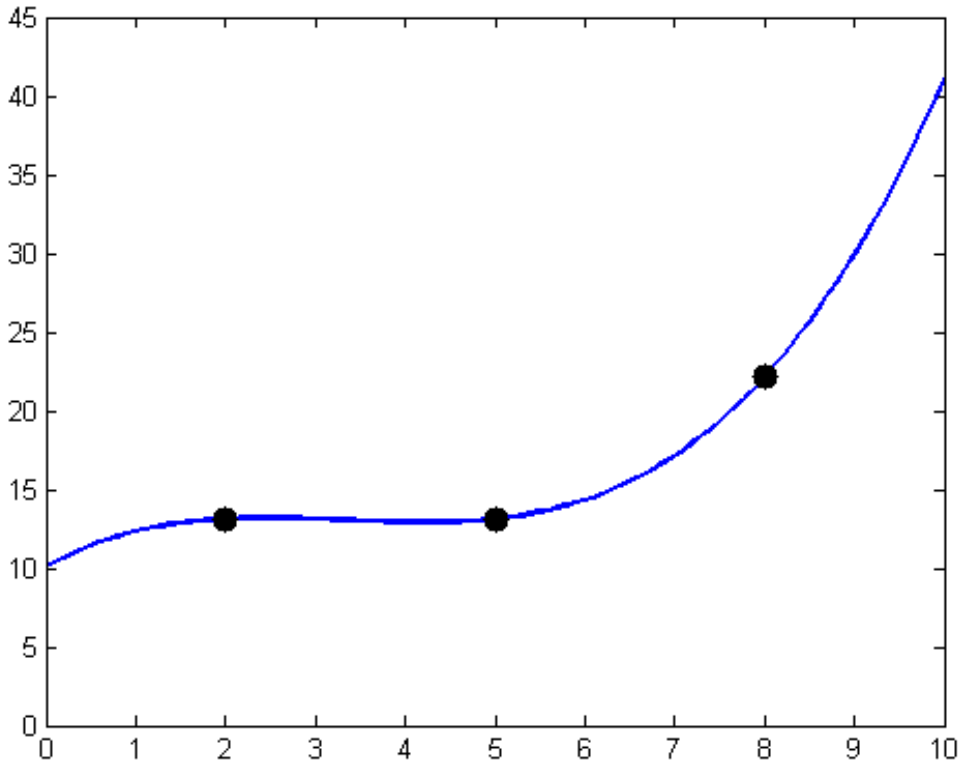
Increasing N increases the monomials we can "capture".

1D Quadrature: Interpolatory



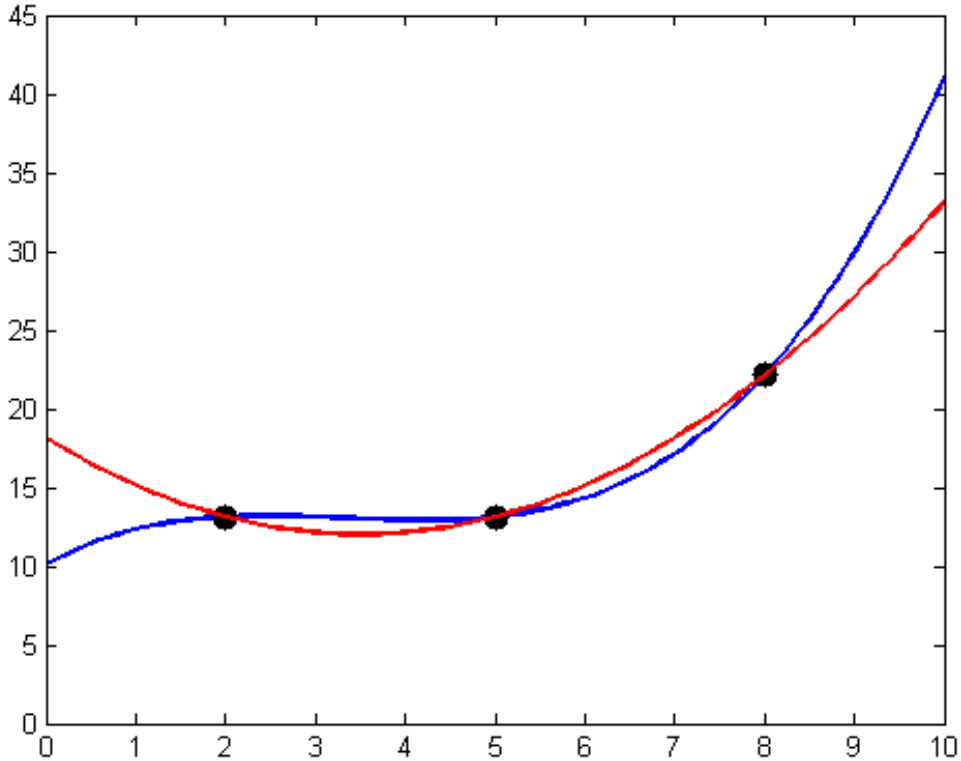
A function f(x) is given.

1D Quadrature: Interpolatory



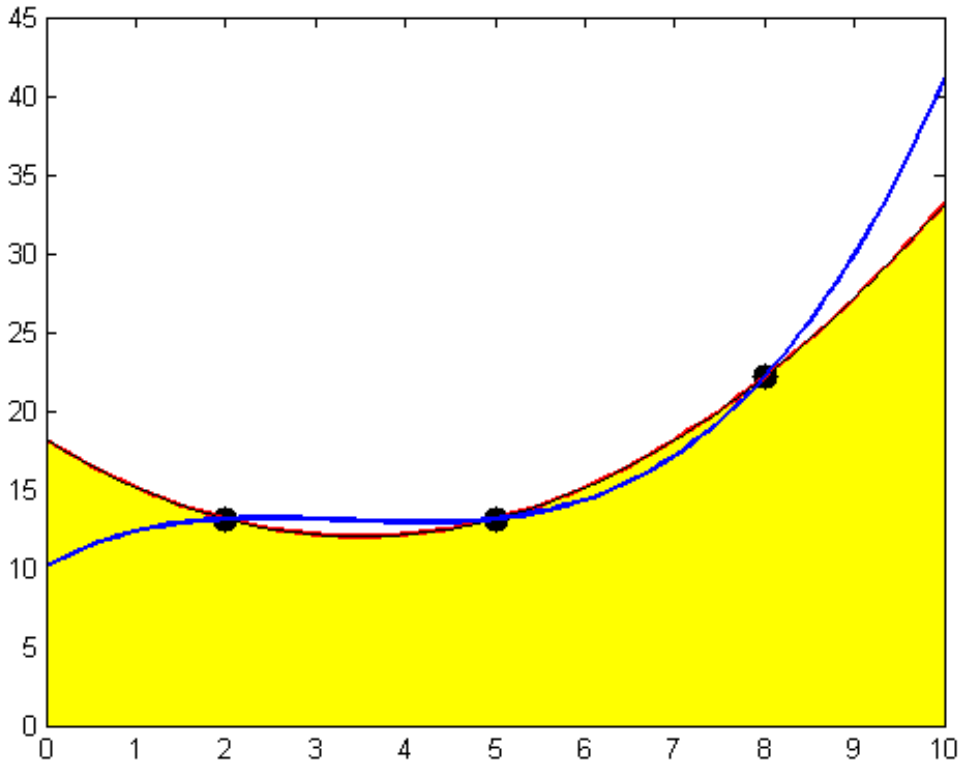
We evaluate it at N points.

1D Quadrature: Interpolatory



We determine the approximating polynomial.

1D Quadrature: Interpolatory



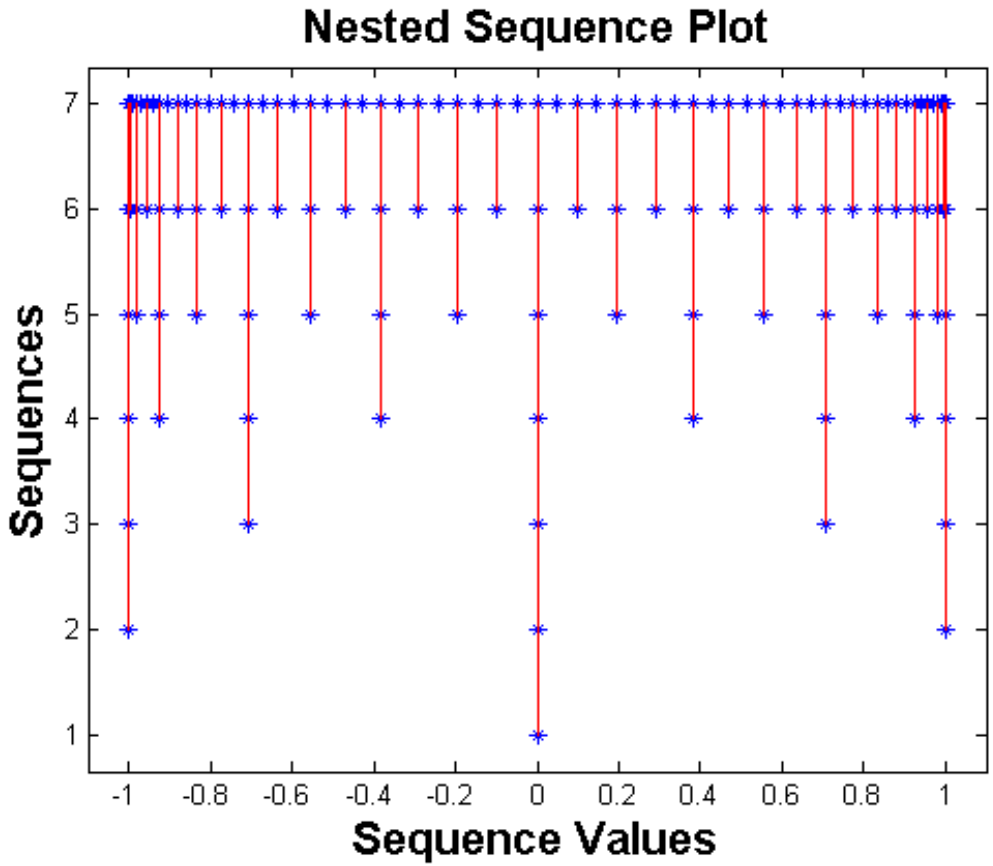
We integrate the approximating polynomial exactly.

1D Quadrature: Interpolatory

- uses a regular grid of N points;
- Evaluates each $f(x)$;
- Computes a *weighted* average of the function values.

- To reuse data, the grids must be "nested".
- **The error can drop with an exponent of N**

1D Quadrature: Interpolatory



Our nested rules roughly double in size at each step.

Monomials and Accuracy

Interpolatory quadrature works well if $f(x)$ can be well approximated by a sum of monomials.

A 1D rule has accuracy N if it "captures" all monomials from $1, x, x^2$, up to x^N.

The lowest monomial we can't capture determines the error. A rule of accuracy N can't capture x^N and so its error behaves like h^{N+1}.

These monomials are the creatures we are "stalking".

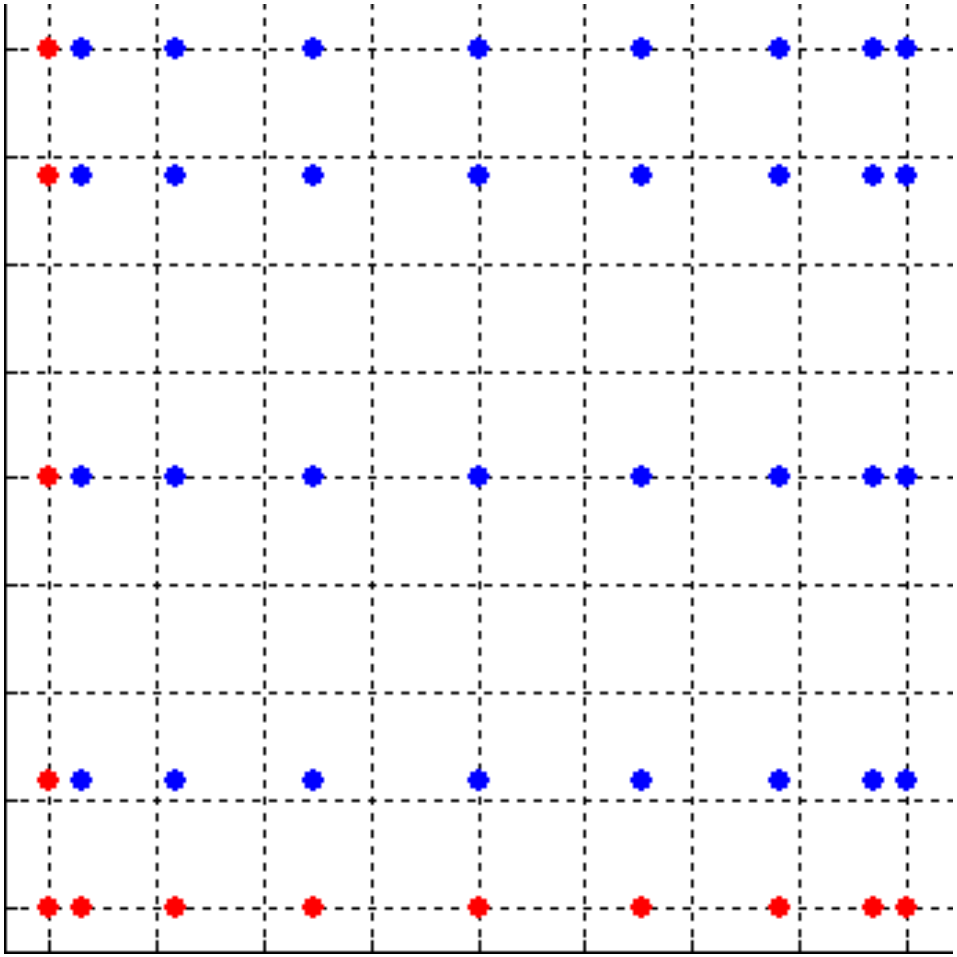
Product Rules

A 2D product rule can be made by taking two 1D rules and combining pairs of values.

The number of points in a product grid is the product of the sizes of the 1D rules.

The resulting rule captures monomials up to $x^{N1}y^{N2}$ where $N1$ and $N2$ are the individual accuracies.

Product Rules



A product of 9 point and 5 point rules.

Product Rules

Suppose we take products of a modest 4 point rule:

- 1D: 4 points;
- 2D: 16 points;
- 3D: 64 points;
- 4D: 256 points;
- 5D: 1024 points;
- 10D: a million points;
- 20D: a trillion points.
- 100D: don't ask!

Conclusion: *Product rules can't go very far!*

Product Rules

The degree of a monomial $x^{N1}y^{N2}$ is $N1 + N2$. Unlike the 1D case, in 2D there are *many* monomials of a given degree.

There are six (x, y) monomials of degree 5:

$$x^5, x^4y, x^3y^2, x^2y^3, xy^4, y^5.$$

A rule of accuracy 5 must capture every one of these monomials.

Accuracy means getting all the monomials up to a given degree.

Monomials up to 4th degree

0					1				
1				x		y			
2			x^2		xy		y^2		
3		x^3		x^2y		xy^2		y^3	
4	x^4		x^3y		x^2y^2		xy^3		y^4
5		x^4y		x^3y^2		x^2y^3		xy^4	
6			x^4y^2		x^3y^3		x^2y^4		
7				x^4y^3		x^3y^4			
8					x^4y^4				

Monomials appearing below the line **are not needed**.

Product Rules

As the dimension increases, the useless monomials predominate.

Suppose we take products of a modest rule of accuracy 10, and limit the exponent total to 10. How many "good" and "useless" monomials do we capture?

Dim	Good	Useless
1D	10	0
2D	55	45
3D	120	880
4D	210	9790
5D	252	99748

Conclusion: *A "cut down" product rule might work!*

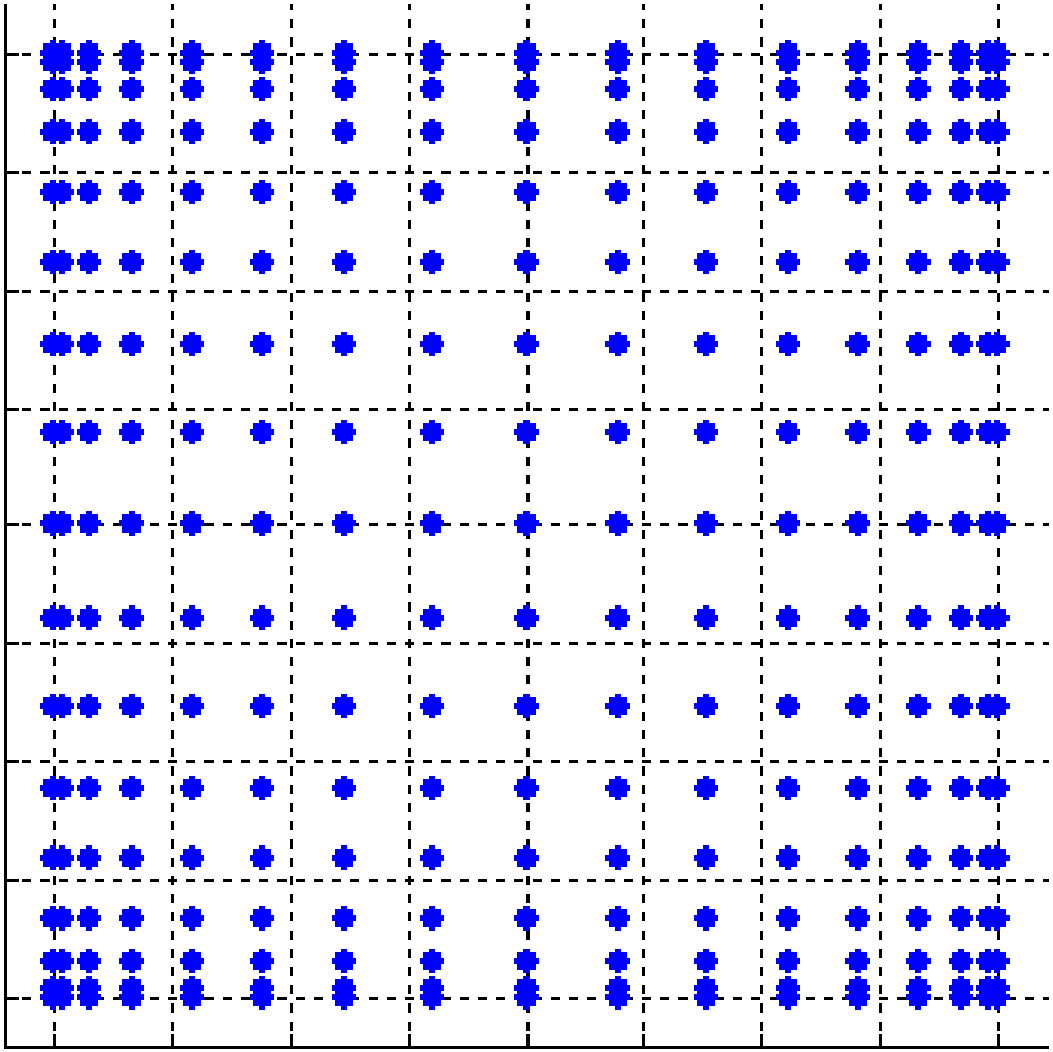
Smolyak Quadrature

Sergey Smolyak (1963) added low order grids together.

His combined "sparse grid":

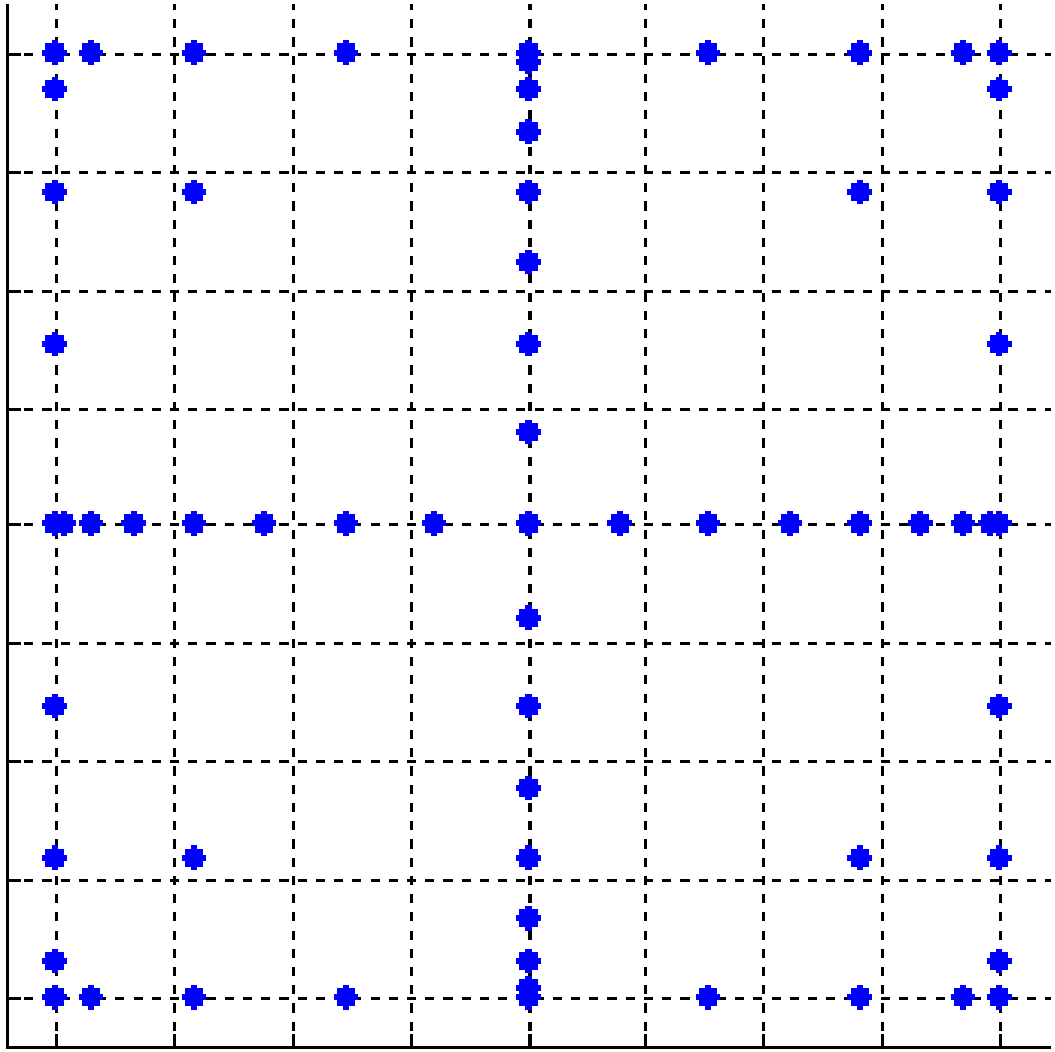
- had the same accuracy as a product grid.
- was a subset of the points of the product grid.
- used *far fewer* points.

2D Quadrature:



A 17x17 product grid (289 points).

2D Quadrature: Level4 Sparse Grid



An "equivalent" sparse grid (65 points).

2D Quadrature: Level4 Sparse Grid

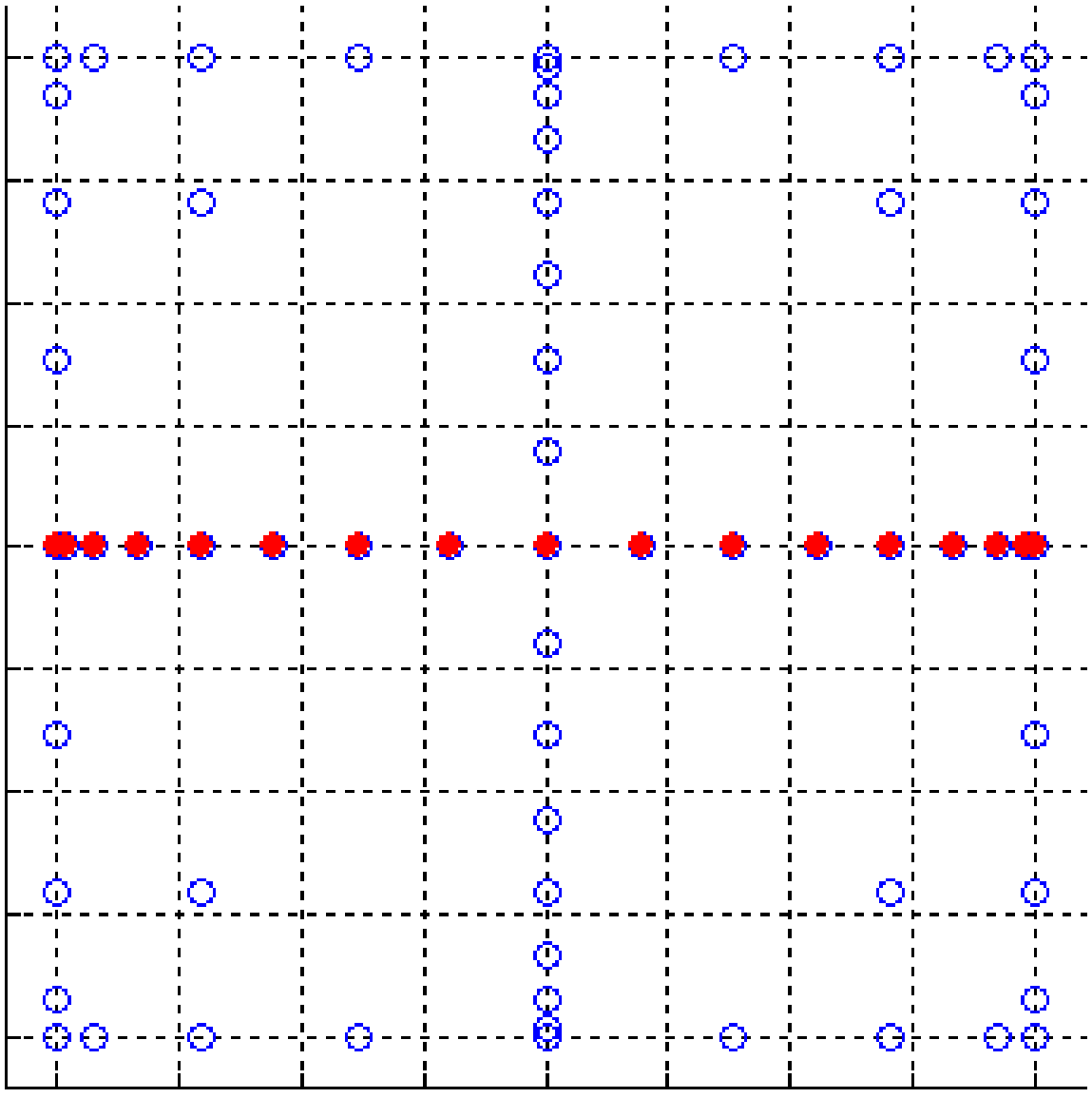
To capture only "desirable" monomials, we essentially add product grids which are sparse in one direction if dense in the other.

Because of nesting, the grids reuse many points.

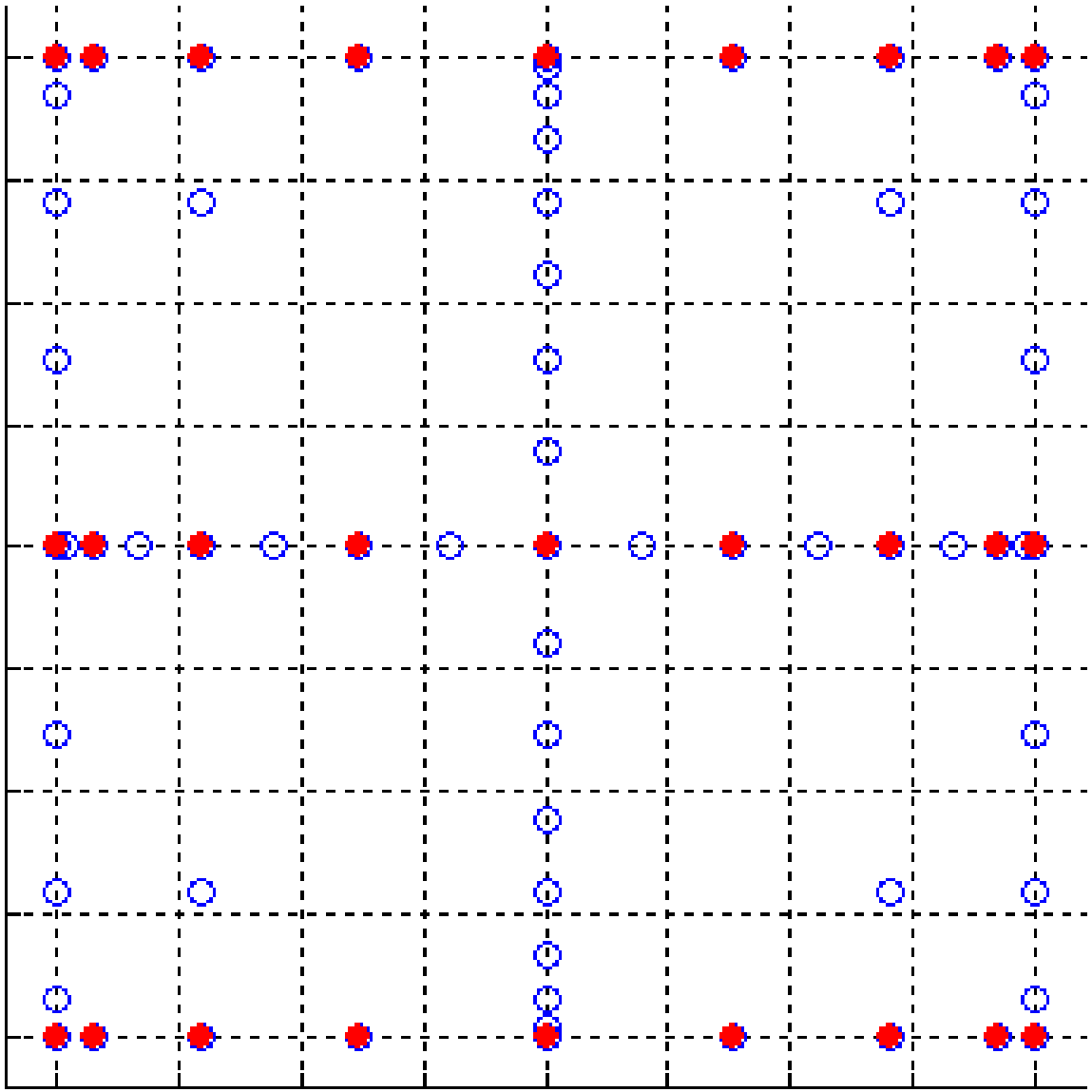
The big savings comes from entirely eliminating most of the points of the full product grid.

The improvement is greater as the dimension or level increases.

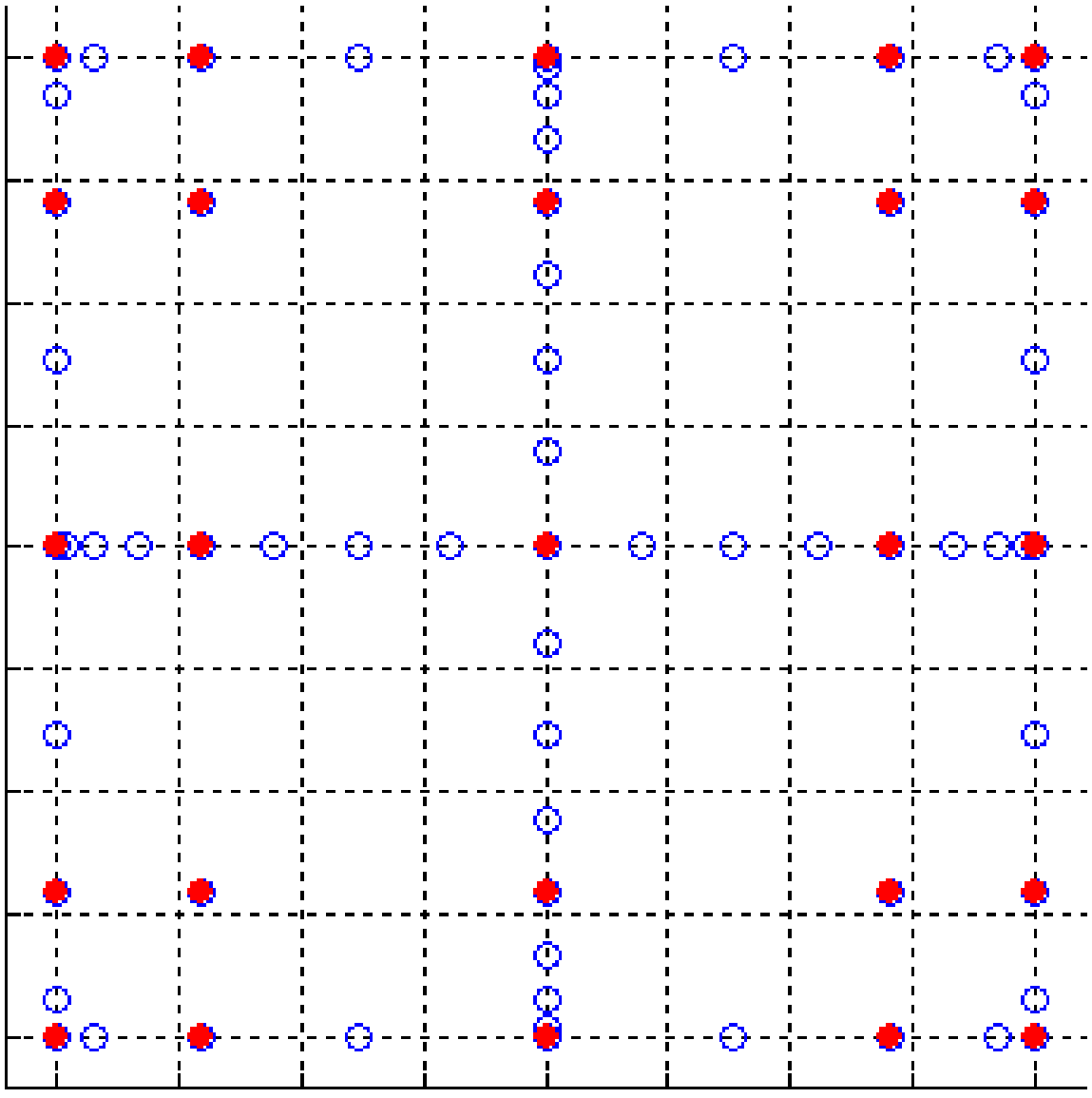
2D Quadrature: Level4 17x1 component



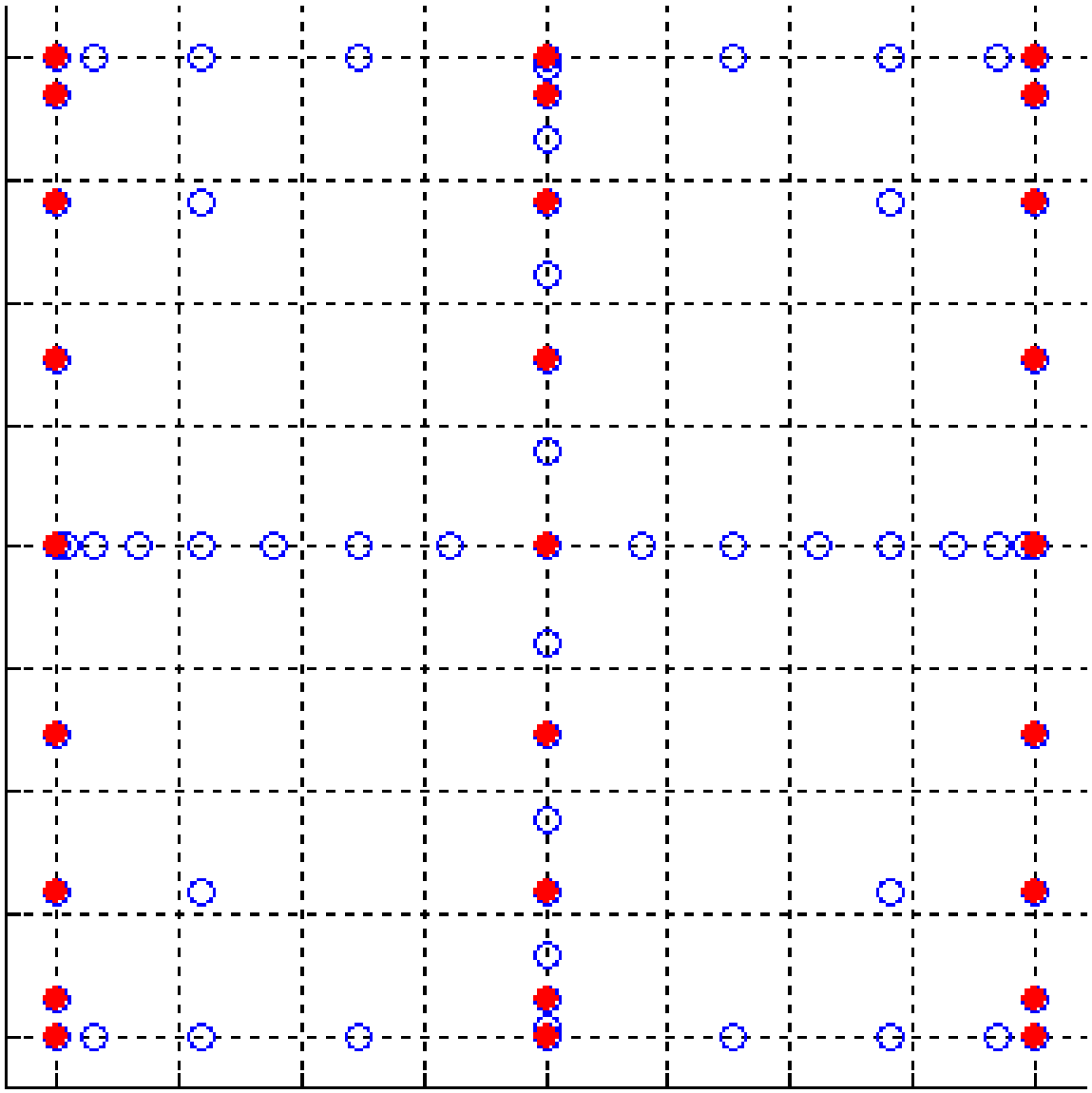
2D Quadrature: Level4 9x3 component



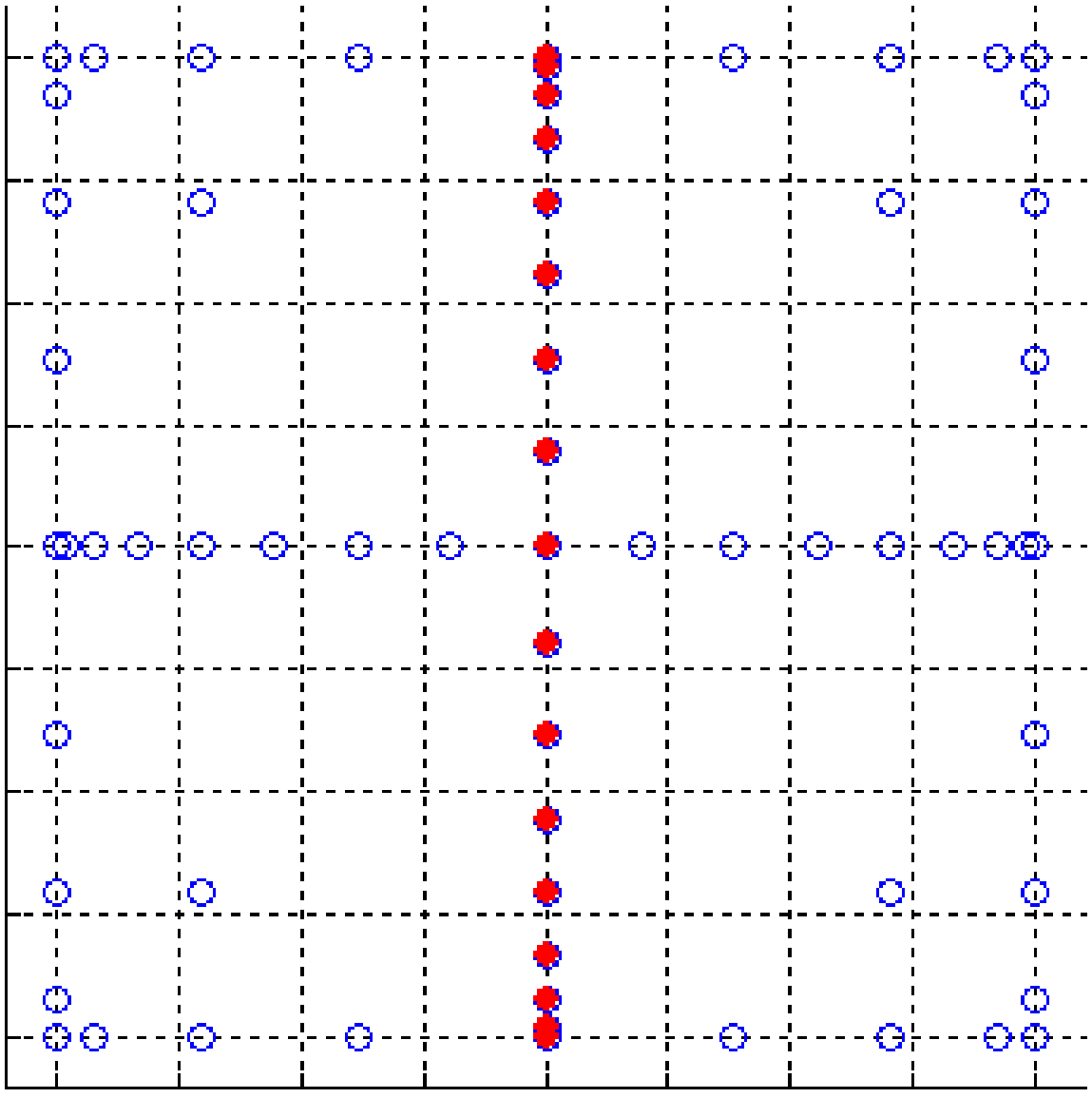
2D Quadrature: Level4 5x5 component



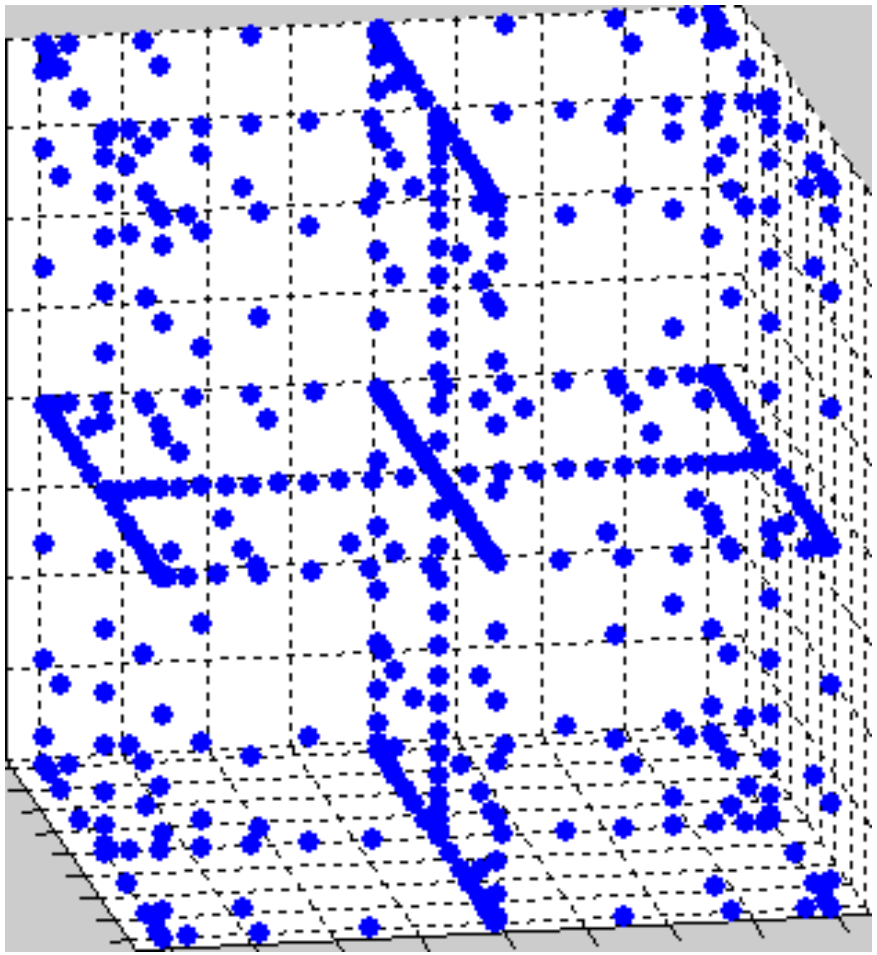
2D Quadrature: Level4 3x9 component



2D Quadrature: Level4 1x17 component

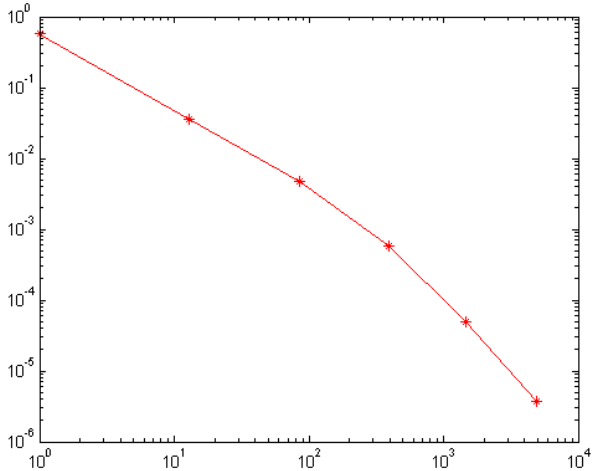


3D Quadrature: Level5 Sparse Grid



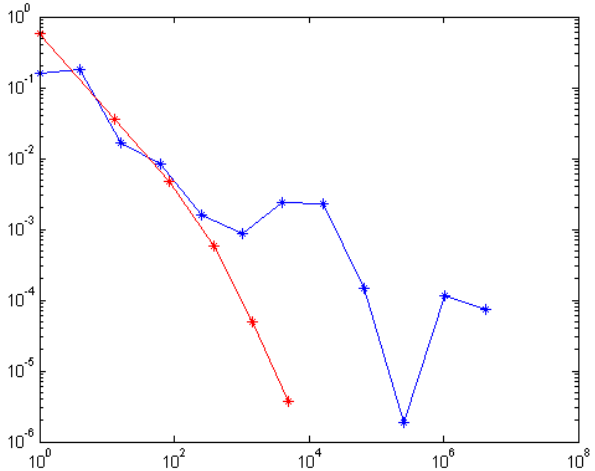
Sparse grid = 441 points; Product grid would have 35,937.

6D Quadrature: Sparse Grid Error



N	Estimate	Error
1	0.062500	0.573282
13	0.600000	0.0357818
85	0.631111	0.00467073
389	0.636364	0.000582152
1457	0.635831	0.0000492033
4865	0.635778	0.00000375410
∞	0.635782	0.0000

6D Quadrature: Monte Carlo vs Sparse Grid



SG N	SG Estimate	—	MC N	MC Estimate
1	0.062500	—	1	0.796541
13	0.600000	—	16	0.652621
85	0.631111	—	256	0.637351
389	0.636364	—	4096	0.633428
1457	0.635831	—	65536	0.635926
4865	0.635778	—	1048576	0.635666
∞	0.635782	—	∞	0.635782

High Dimensional Quadrature: A Quote

"When good results are obtained in integrating a high-dimensional function, we should conclude first of all that an especially tractable integrand was tried and not that a generally successful method has been found.

"A secondary conclusion is that we might have made a very good choice in selecting an integration method to exploit whatever features of f made it tractable."

Art Owen, Stanford University.

The Diffusion Equation

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

u is an unknown quantity, like temperature;

a is a **known** physical property, the conductivity, which controls how quickly hot or cold spots average out.

- heat conduction;
- slow subsurface flow of water;
- particle diffusion;
- Black-Scholes equation (flow of money!).

The Diffusion Equation: Uncertain Conductivity

Using a fixed value for $a(x, y)$ might be unrealistic.

Without variations in $a(x, y)$, we might never see the bumps and swirls typical of real physical problems.

We might think of $a(x, y)$ as a *random field* $a(\omega; x, y)$.

The ω represents the unknown variation from the average.

The Diffusion Equation: Uncertain Solution

If $a(\omega; x, y)$ has an "unknown" component, then so does our solution, which we write $u(\omega; x, y)$.

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

Now if we don't know what the equation is, we can't solve it!

Can we still extract information from the equation?

The Diffusion Equation: Expected Values

Each variation ω determines a solution u.

If we added up every variation, we'd get an average or expected value for the solution.

The expected value is an important first piece of information about a problem with a random component.

$$E(u(x,y)) = \int_\Omega u(\omega; x, y)\, pr(\omega)\, d\omega$$

It's like using weather records to estimate the *climate*.

The Diffusion Equation: Approximate Integral

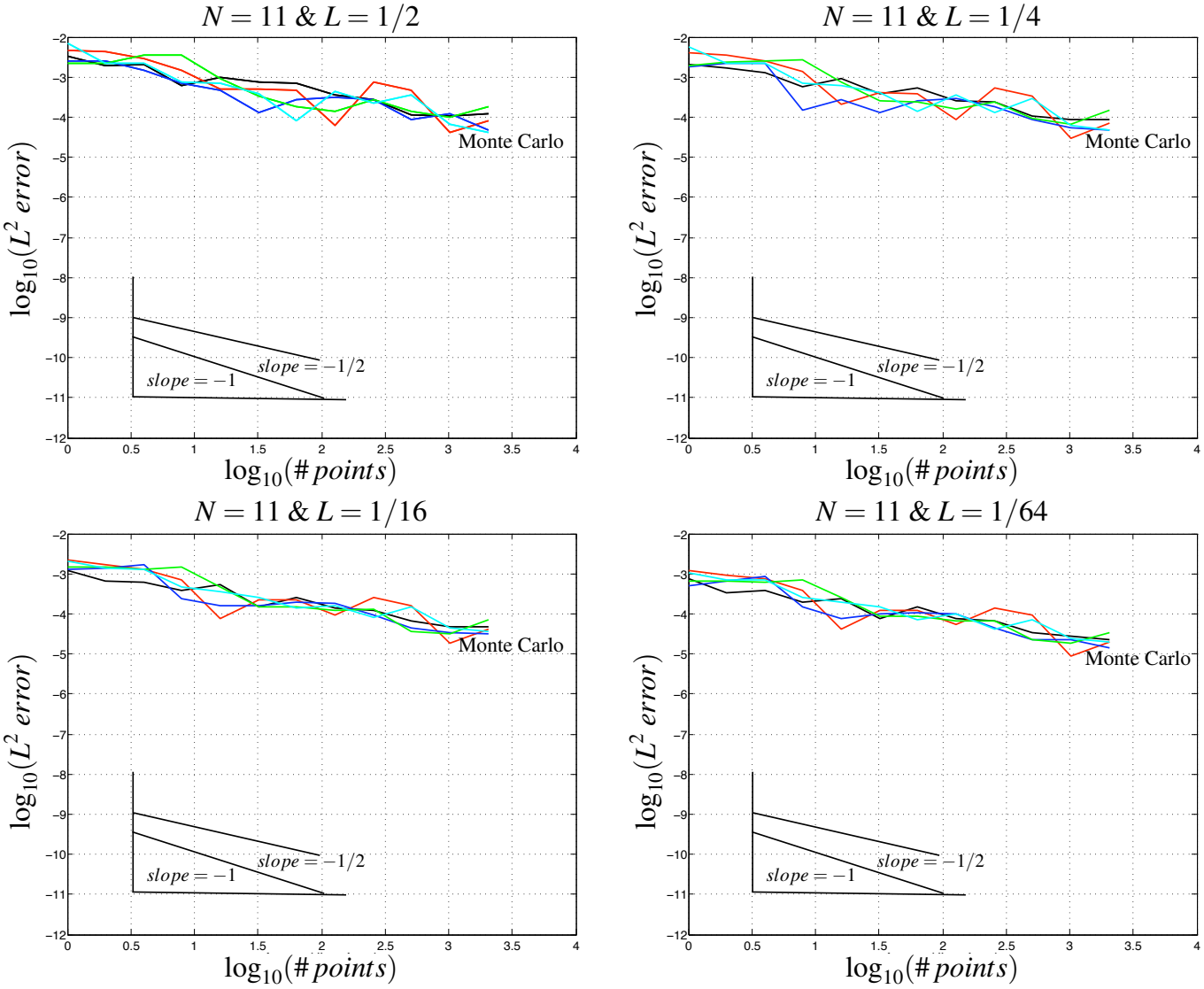
We approximate the function space Ω by an M-dimensional space Ω_M, of linear sums of perturbations ω_M.

We now estimate the integral of $u(\omega_M; x, y)$ in Ω_M.

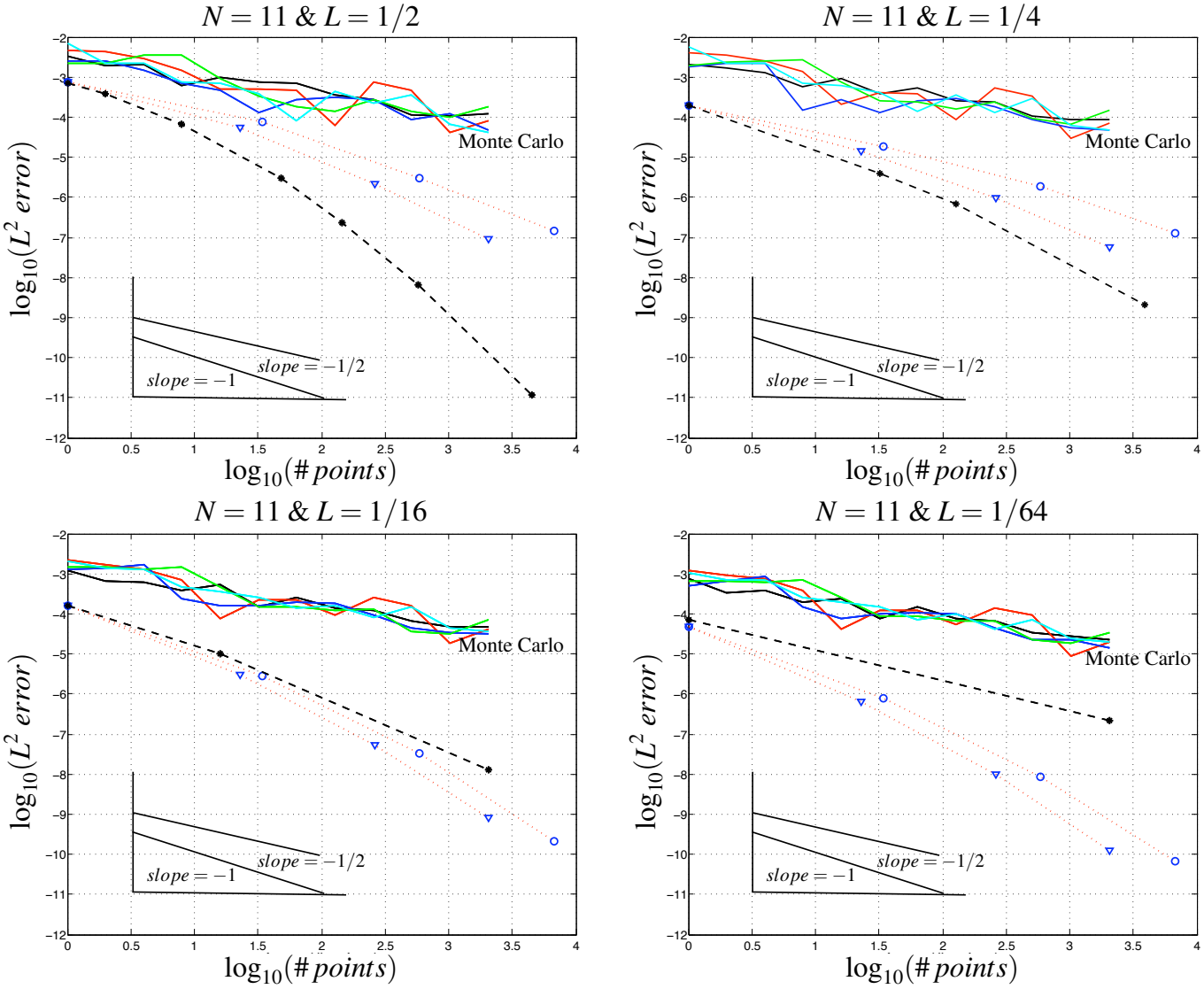
Monte Carlo: select a random set of parameters ω_M, solve for u, multiply by the probability, and average.

Sparse grid: choose a level, defining a grid of ω_M values in Ω_M, solve for each u, multiply by the probability, and take a weighted average.

The Diffusion Equation: Monte Carlo



The Diffusion Equation: Smolyak



Conclusion: Future Research

- Precompute quadrature rules, for parallel application
- Composite version for decomposition of a few dimensions.
- Modify the algorithm so that some dimensions may be approximated more carefully.
- Detect anisotropy in the data.
- Estimate the quadrature error cheaply.

Conclusion: The End

- High dimensional integration is a feature of modern algorithms
- Accurate Monte Carlo results take a long time
- Product rules quickly become useless
- "Smooth" data can be well integrated by Smolyak grids
- High dimensional probability spaces, for example, generate smooth data

Conclusion: Software

SMOLPACK, a C library by Knut Petras for sparse integration.

SPINTERP, ACM TOMS Algorithm 847, a MATLAB library by Andreas Klimke for sparse grid **interpolation**.

SPARSE_GRID_CC a directory on my website containing examples of sparse grids generated from Clenshaw Curtis rules.

Conclusion: References

Volker **Barthelmann**, Erich **Novak**, Klaus **Ritter**, *High Dimensional Polynomial Interpolation on Sparse Grids*, Advances in Computational Mathematics, Volume 12, Number 4, March 2000, pages 273-288.

Thomas **Gerstner**, Michael **Griebel**, *Numerical Integration Using Sparse Grids,*, Numerical Algorithms, Volume 18, Number 3-4, January 1998, pages 209-232.

Sergey **Smolyak**, *Quadrature and Interpolation Formulas for Tensor Products of Certain Classes of Functions*, Doklady Akademii Nauk SSSR, Volume 4, 1963, pages 240-243.