High Dimensional Sparse Grids
(stalking the wild integral)

http://people.sc.fsu.edu/∼jburkardt/presentations/fsu_2007.pdf

.....

John Burkardt[1]

[1]School of Computational Science
Florida State University

School of Computational Science
19 June 2007

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 $f(x)$.

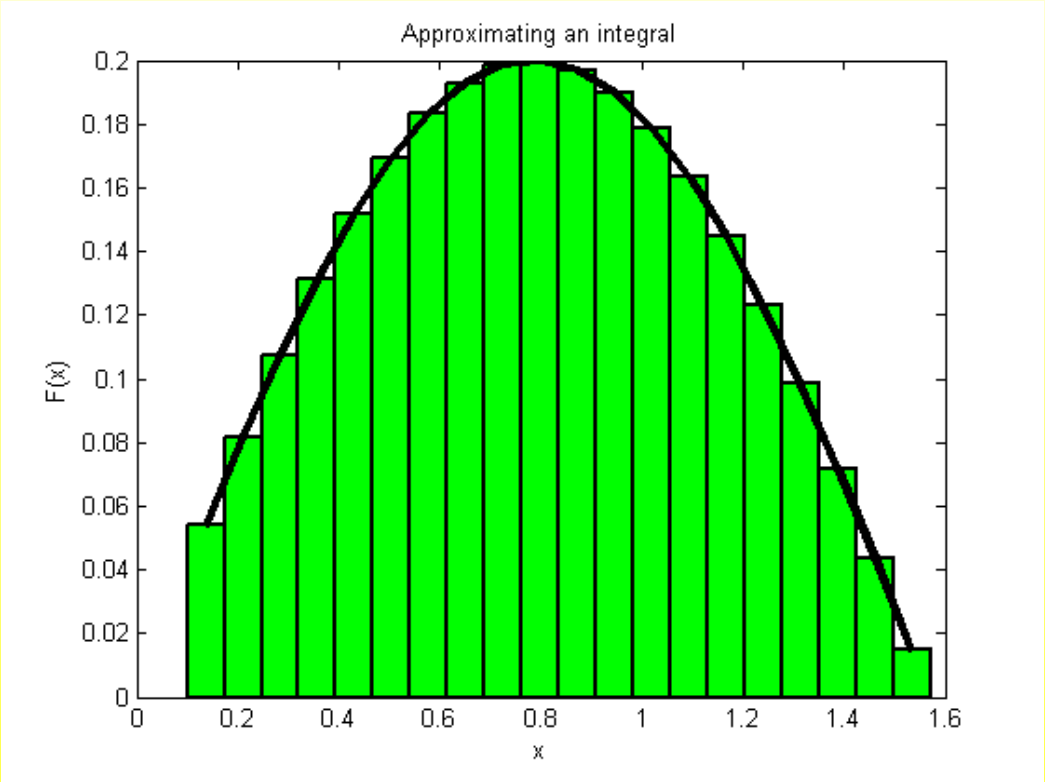
$$\int x^3 \, dx = \frac{x^4}{4} + C$$

But most formulas have no antiderivative!

And most things we want to integrate are not formulas!

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

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 recalls the definition of an integral as an averaging process.

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

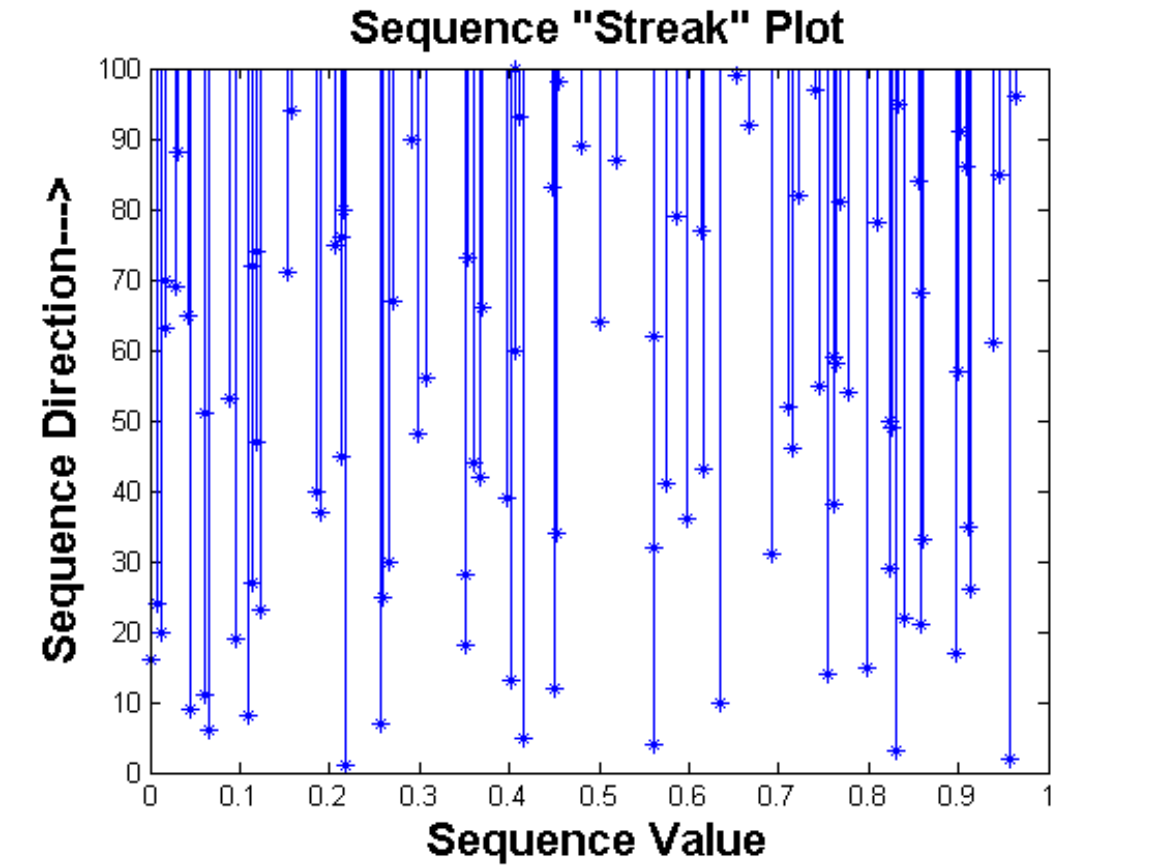
1D Quadrature: Monte Carlo

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.

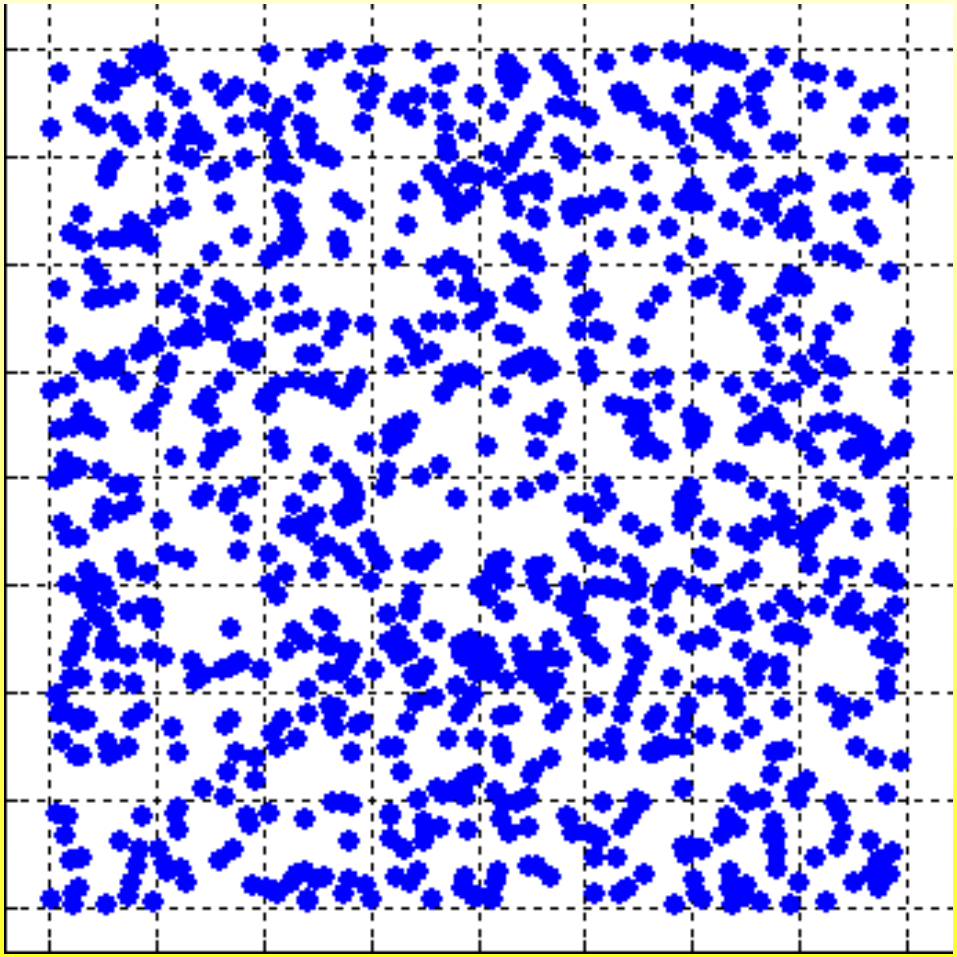
- If more accuracy needed, current values can be included;
- Accuracy improves at the rate $\sqrt{N}$.
- Accuracy hampered because of large "gaps" in sampling.
- Accuracy is independent of spatial dimension.

1D Quadrature: Monte Carlo



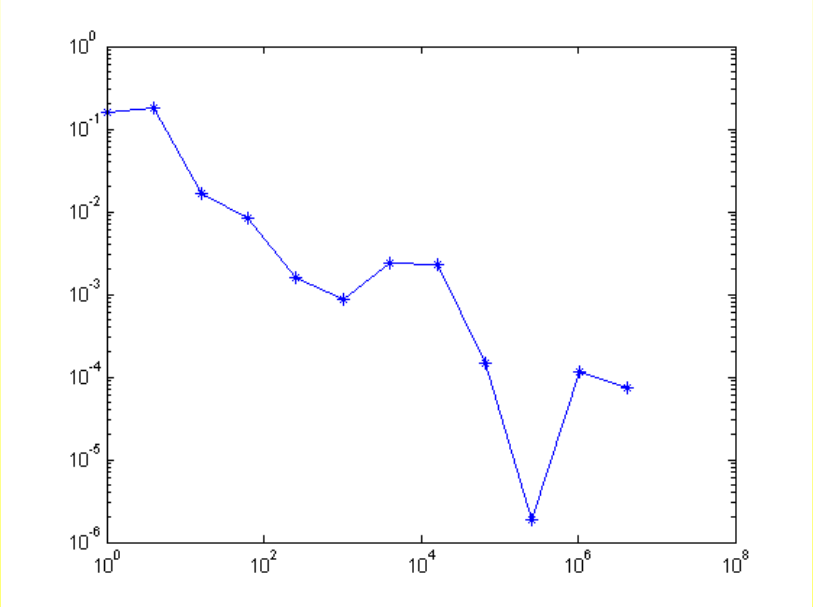
Notice the clustering and gaps.

2D Quadrature: Monte Carlo



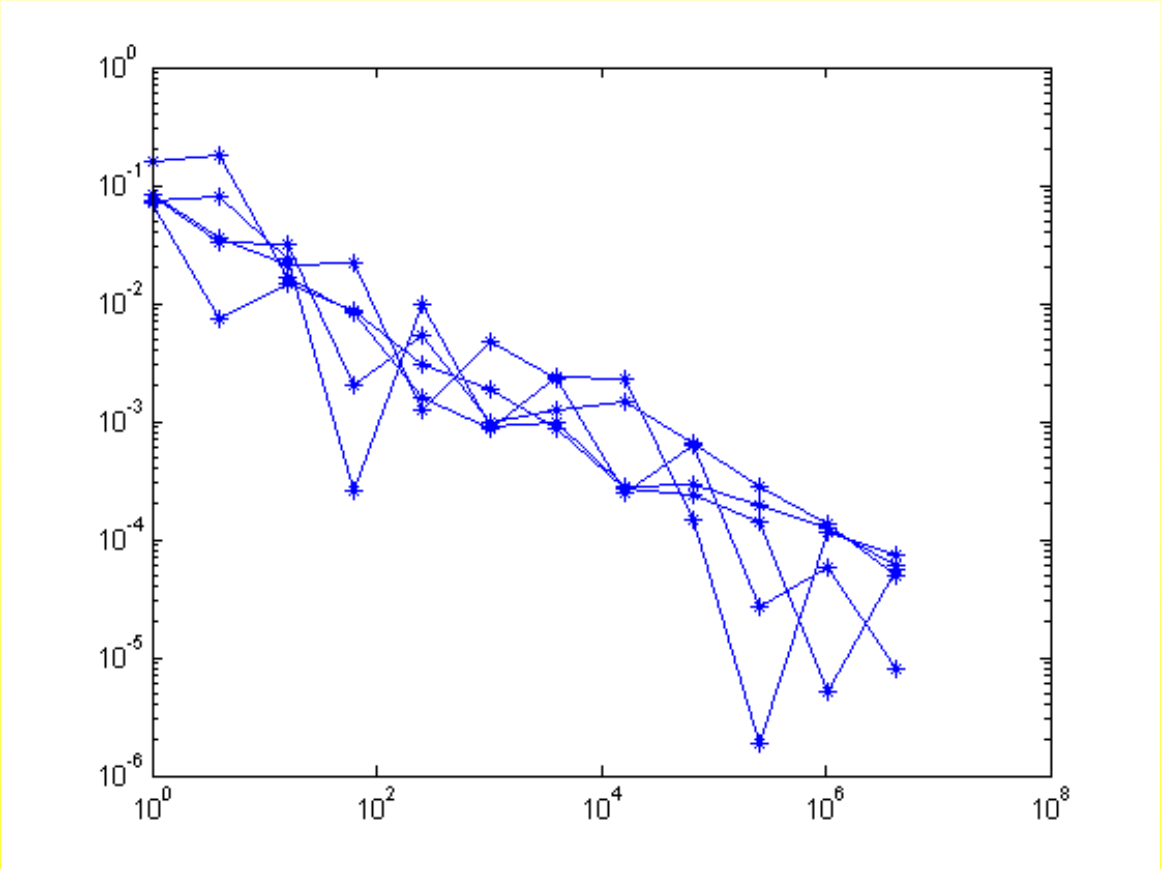
Notice the "gaps"

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



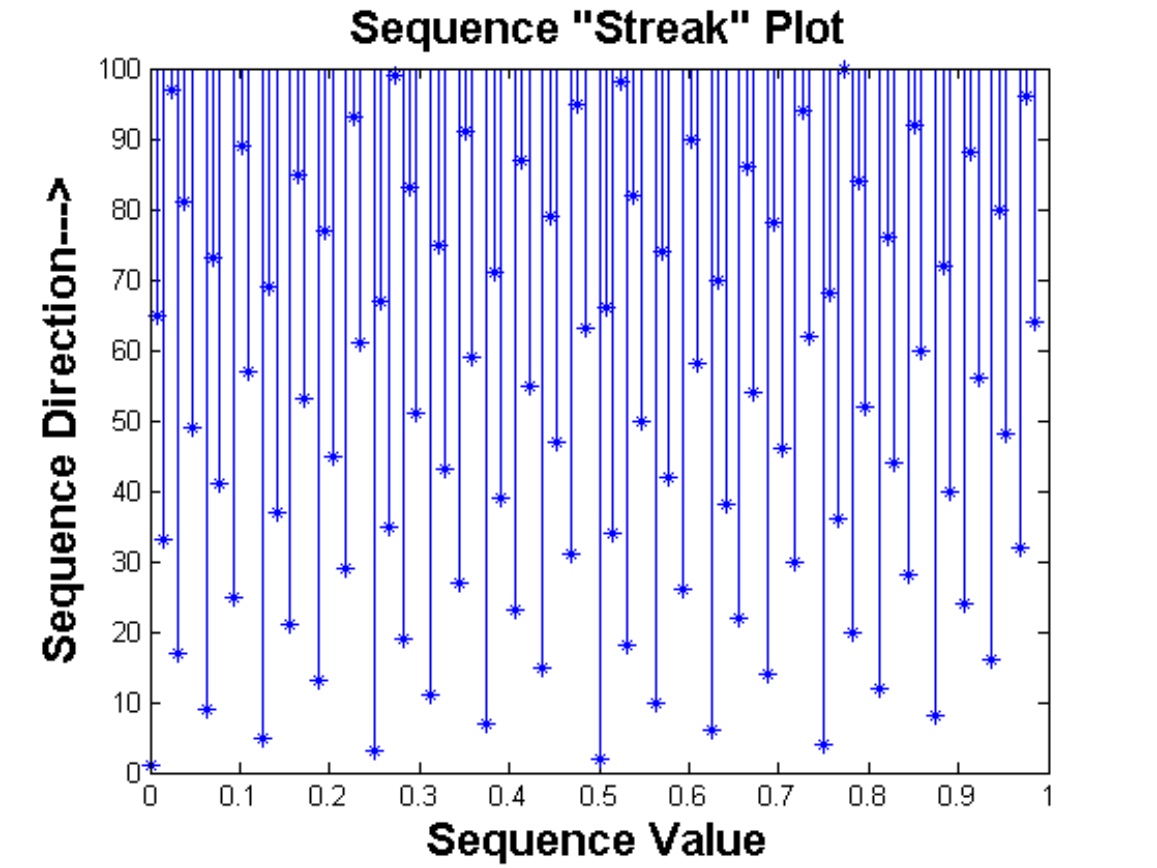
If we try five times, we get five different sets of results.

1D Quadrature: QuasiMonte Carlo

The Quasi-Monte Carlo rules try to improve the accuracy of the Monte Carlo method by choosing points a little less randomly.

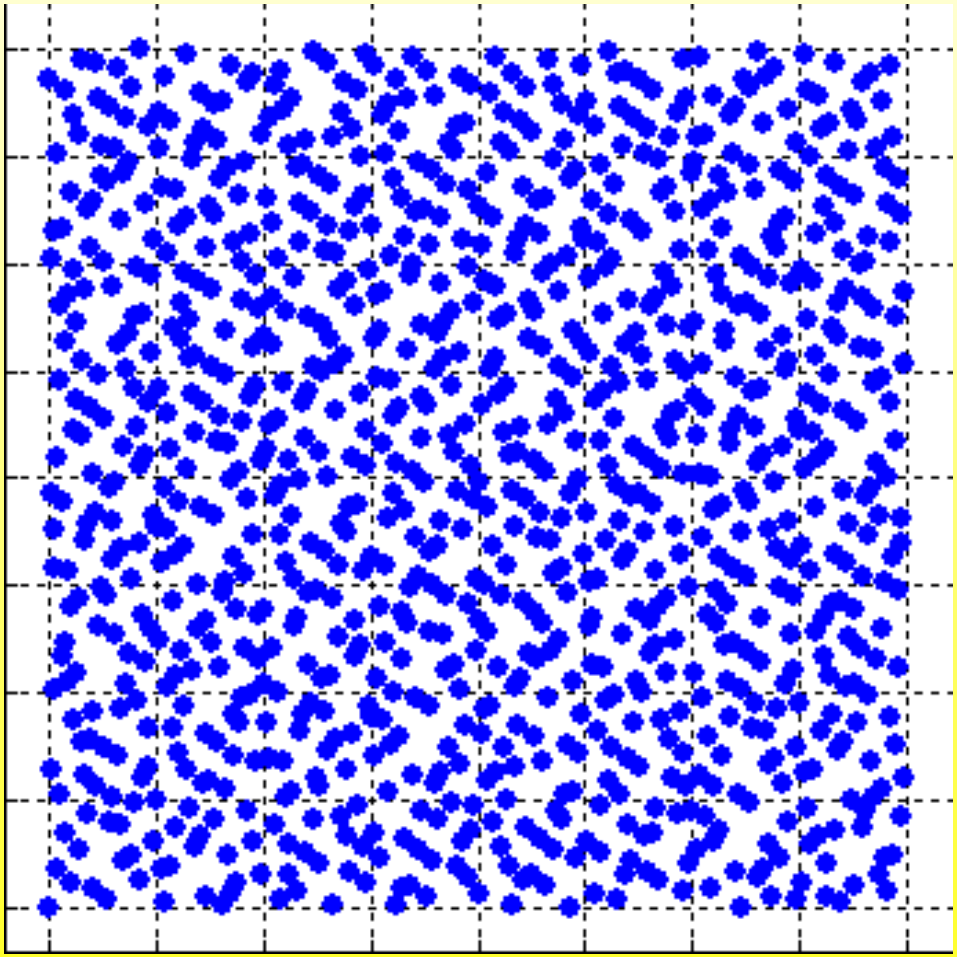
- Average function values at N "quasi-random" points x;
- If more accuracy needed, current values can be included;
- **Accuracy improved because of regularity of sampling** .
- Rate of improvement with N only slightly better than MC.

1D Quadrature: QuasiMonte Carlo



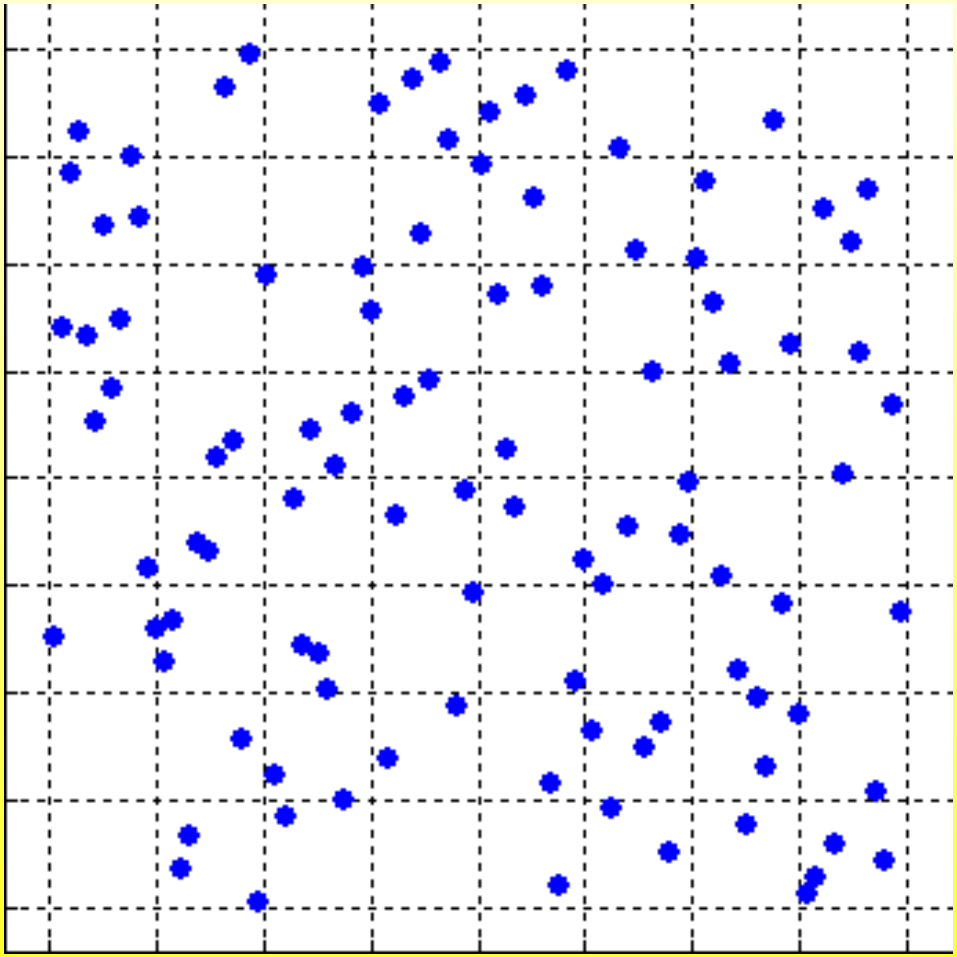
This QMC rule leaves no gaps.

2D Quadrature: QuasiMonte Carlo



This QMC rule leaves no gaps.

2D Quadrature: Latin Square



Rules based on Latin Squares sample each 1D range evenly.

Alternatives to Sampling?

A strength of sampling methods: they make **no assumptions** about $f(x)$, so they are just as good on "misbehaving" functions as on well-behaved ones.

But this is also a weakness. Sampling ignores the extra information a well-behaved function supplies.

If our function $f(x)$ is well-behaved, we can come up with better approaches.

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 determine the initial part of the function's representation, we sample the function.

Evaluating at one point can give us the constant.

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

Evaluating at a second point gives us the coefficient of x as well:

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

Evaluating at N points allows us to determine the first N coefficients.

1D Quadrature: Interpolatory

From well spaced function values, we get an approximate formula which we can integrate exactly.

Such a method will integrate *perfectly* the first N monomials,

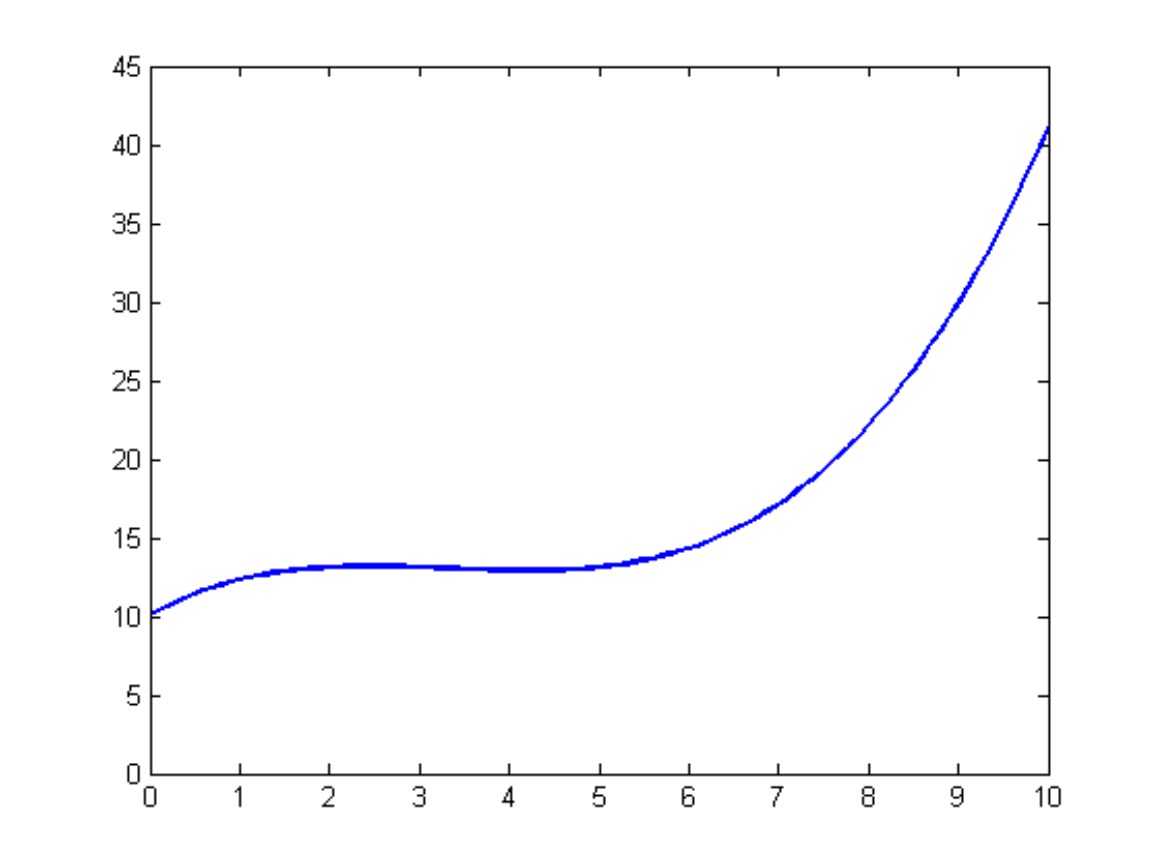
$$1, x, x^2, ..., x^{N-1},$$

as well as any linear combination.

The error is typically of the form h^N, where h is the spacing between sample points.

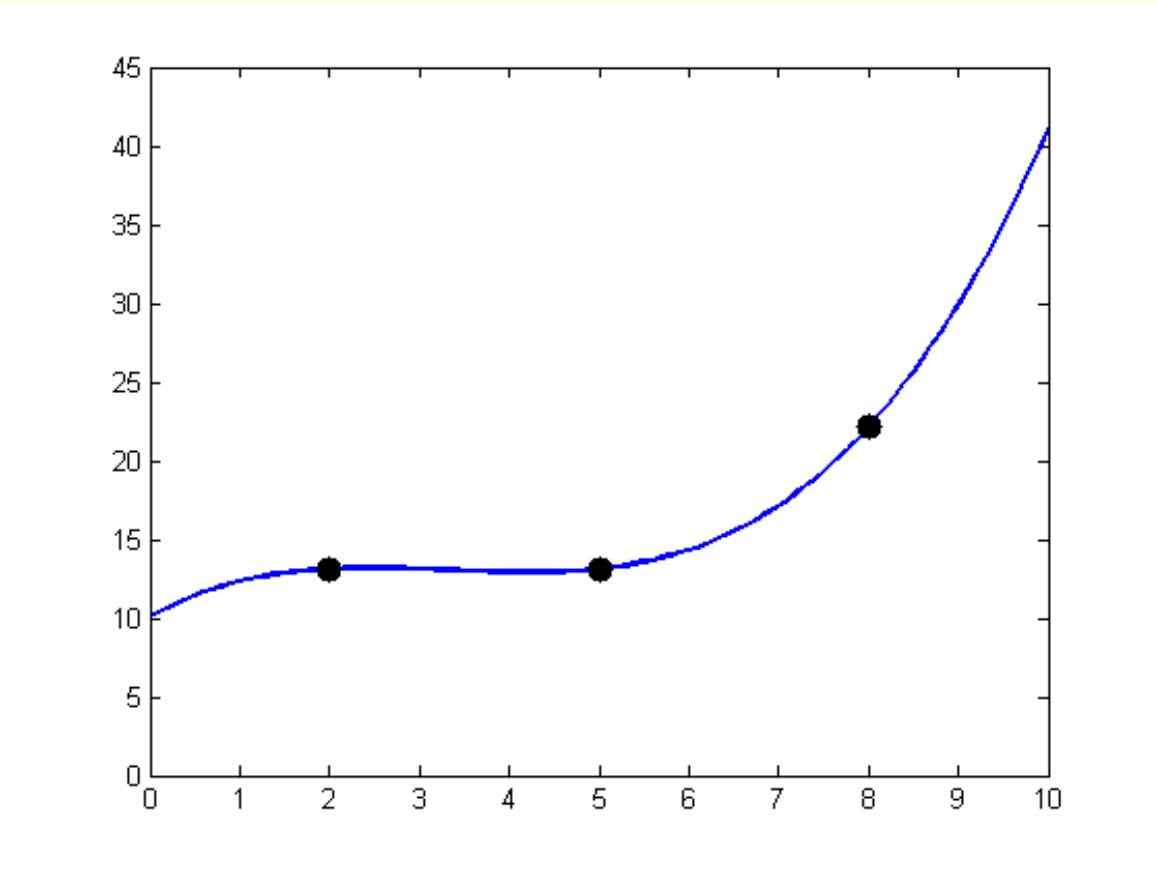
Increasing N increases the monomials we can "capture".

1D Quadrature: Interpolatory



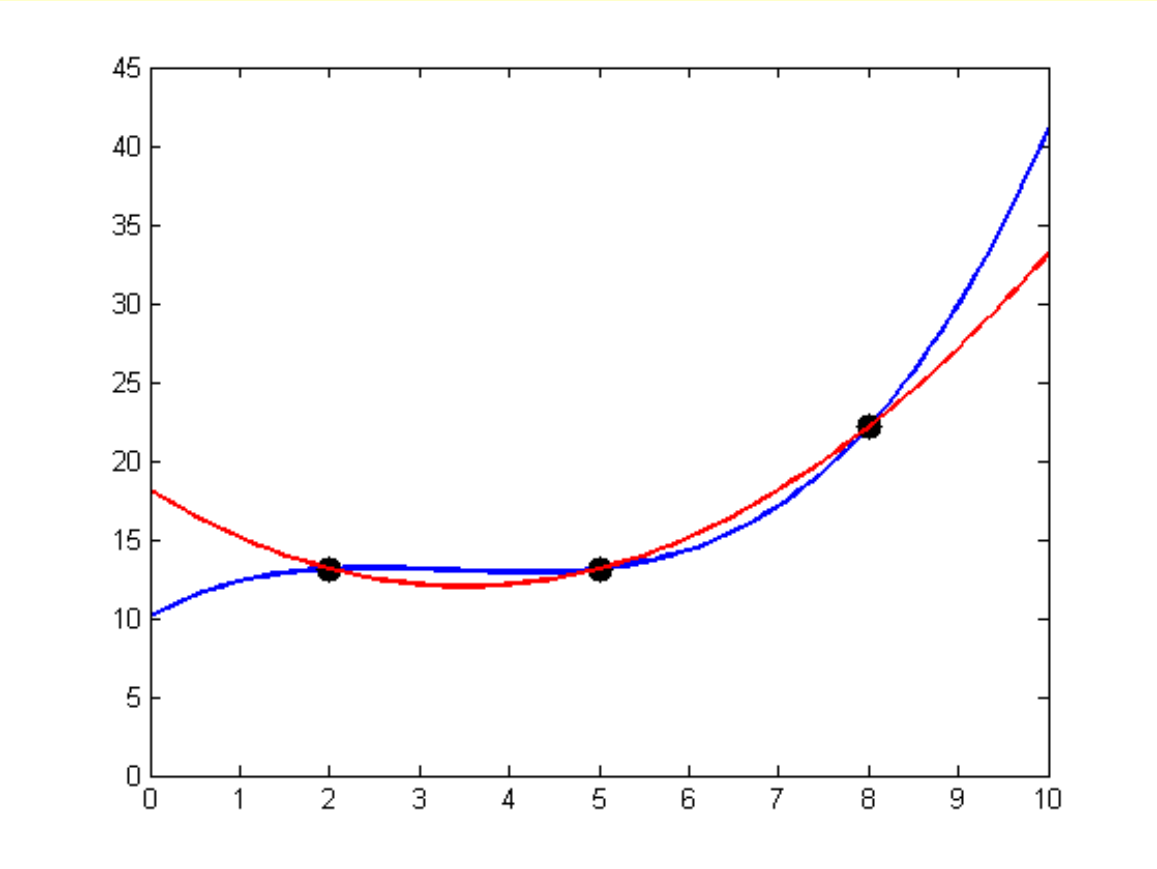
A function f(x) is given.

1D Quadrature: Interpolatory



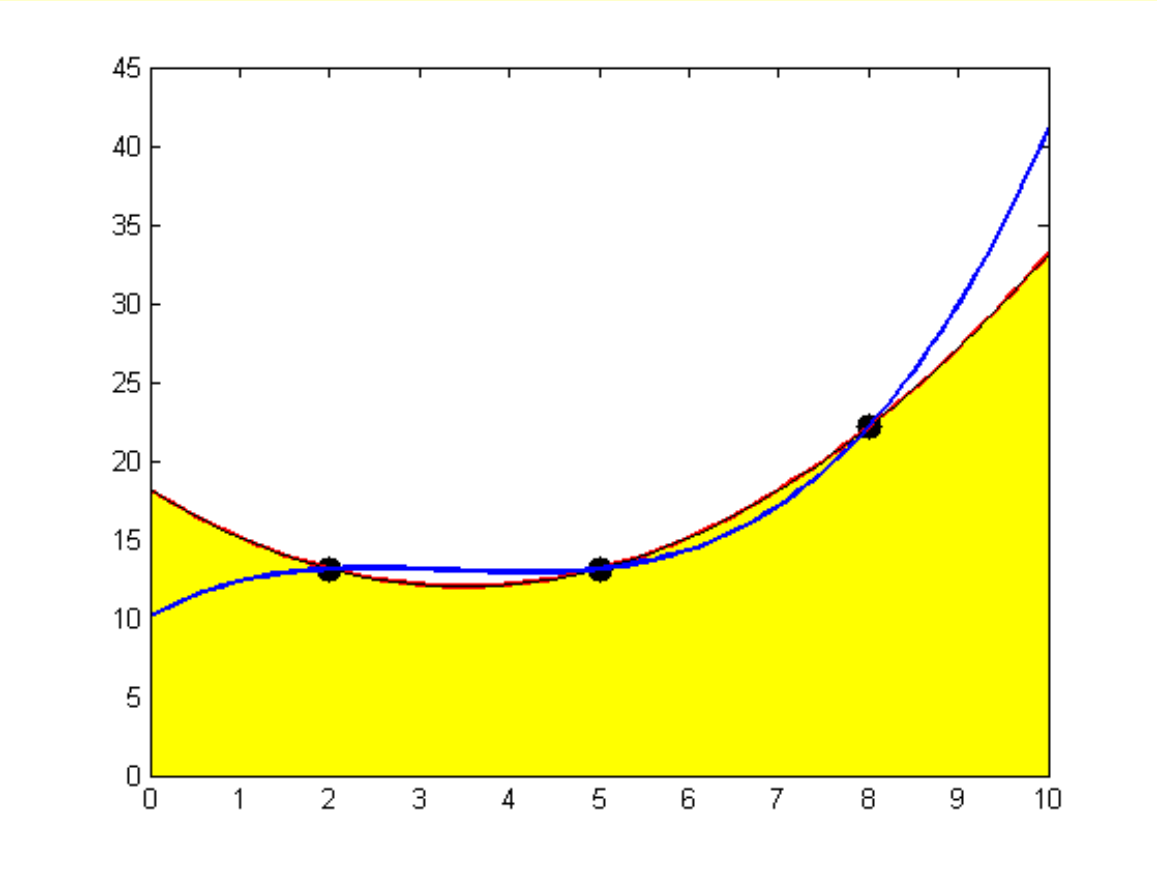
We evaluate it at N points.

1D Quadrature: Interpolatory



We determine the approximating polynomial.

1D Quadrature: Interpolatory



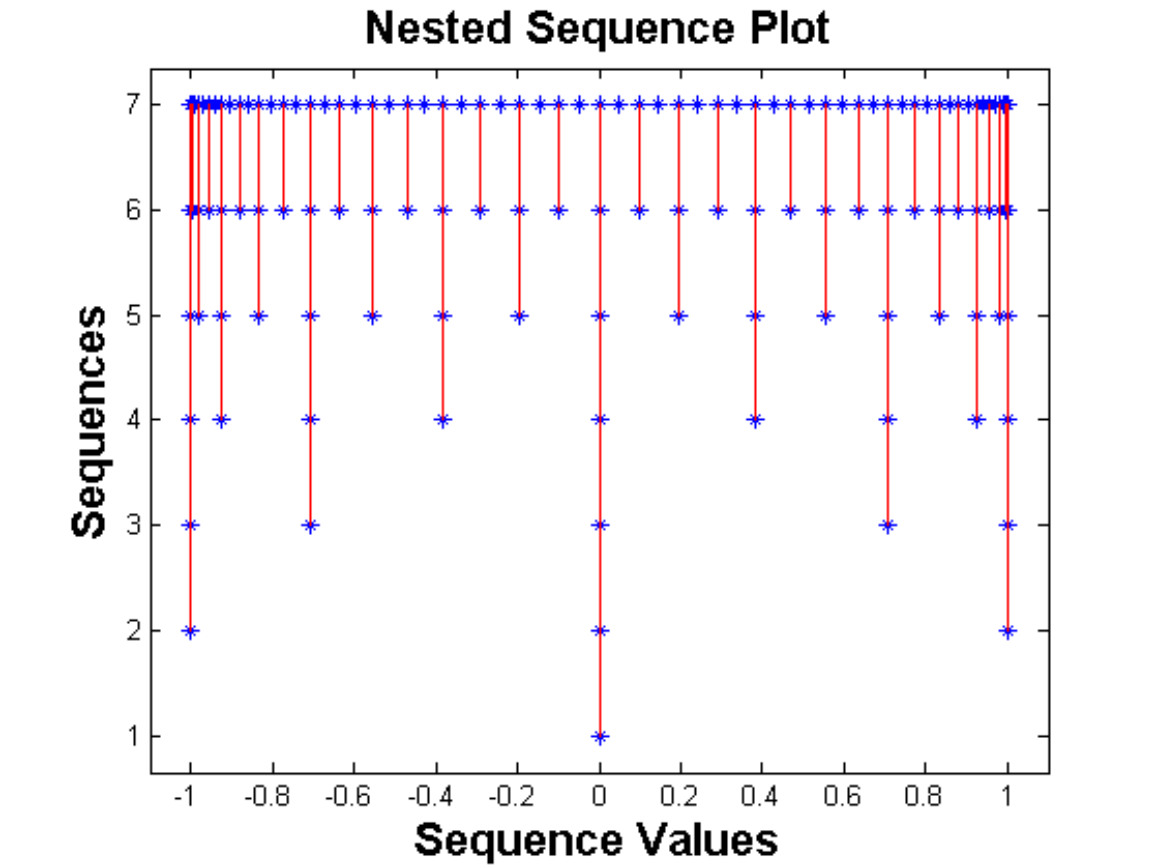
We integrate the approximating polynomial exactly.

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

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".

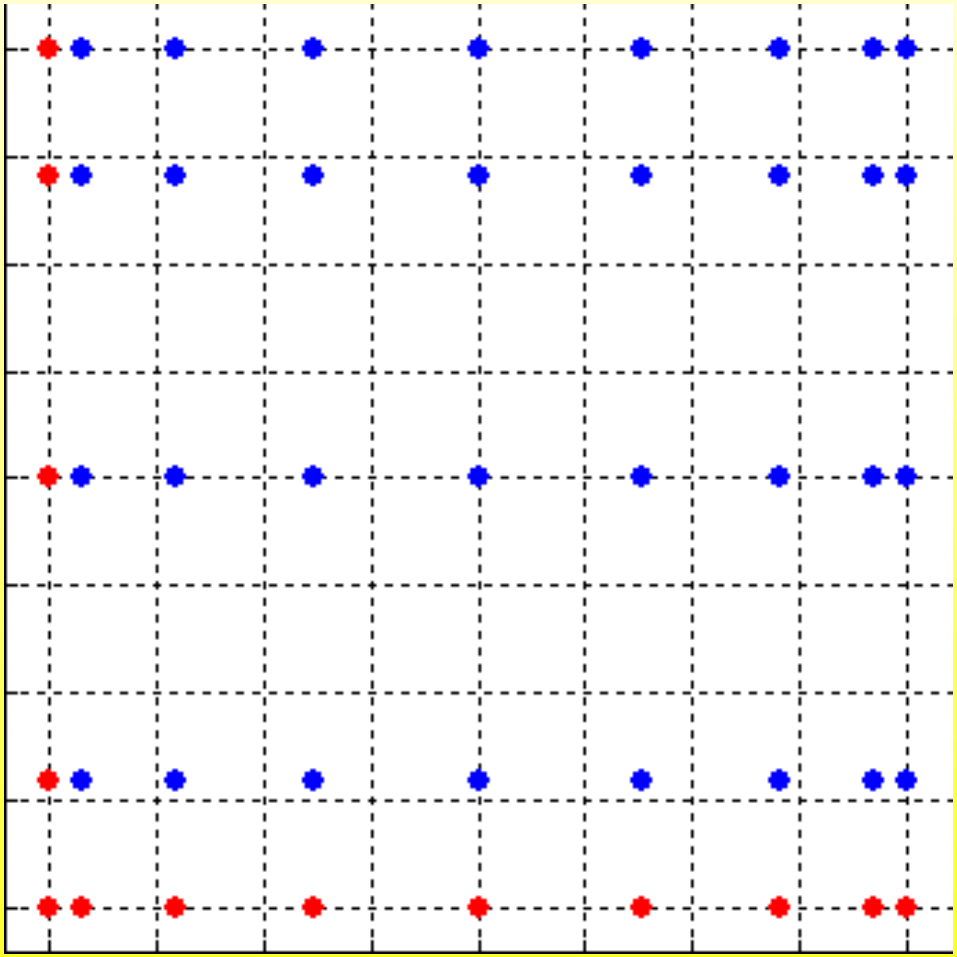
Product Rules

A product rule constructs a grid from 1D grids:
In 2D, choose rules for the X and Y directions;
the grid is all combinations of the x's and y's.

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;
- 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:

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

For a rule to have accuracy 5, it must capture all these monomials, as well as all the lower order ones.

So there's a lot of work to do. But in fact, we're actually doing too much work.

Product Rules

A rule only need to "capture" monomials up to a degree N.

In 2D, we need monomials $x^{N1}y^{N2}$ for which $N1 + N2 \leq N$.

But a product rule captures all monomials for which either $N1 \leq N$ or $N2 \leq N$.

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!*

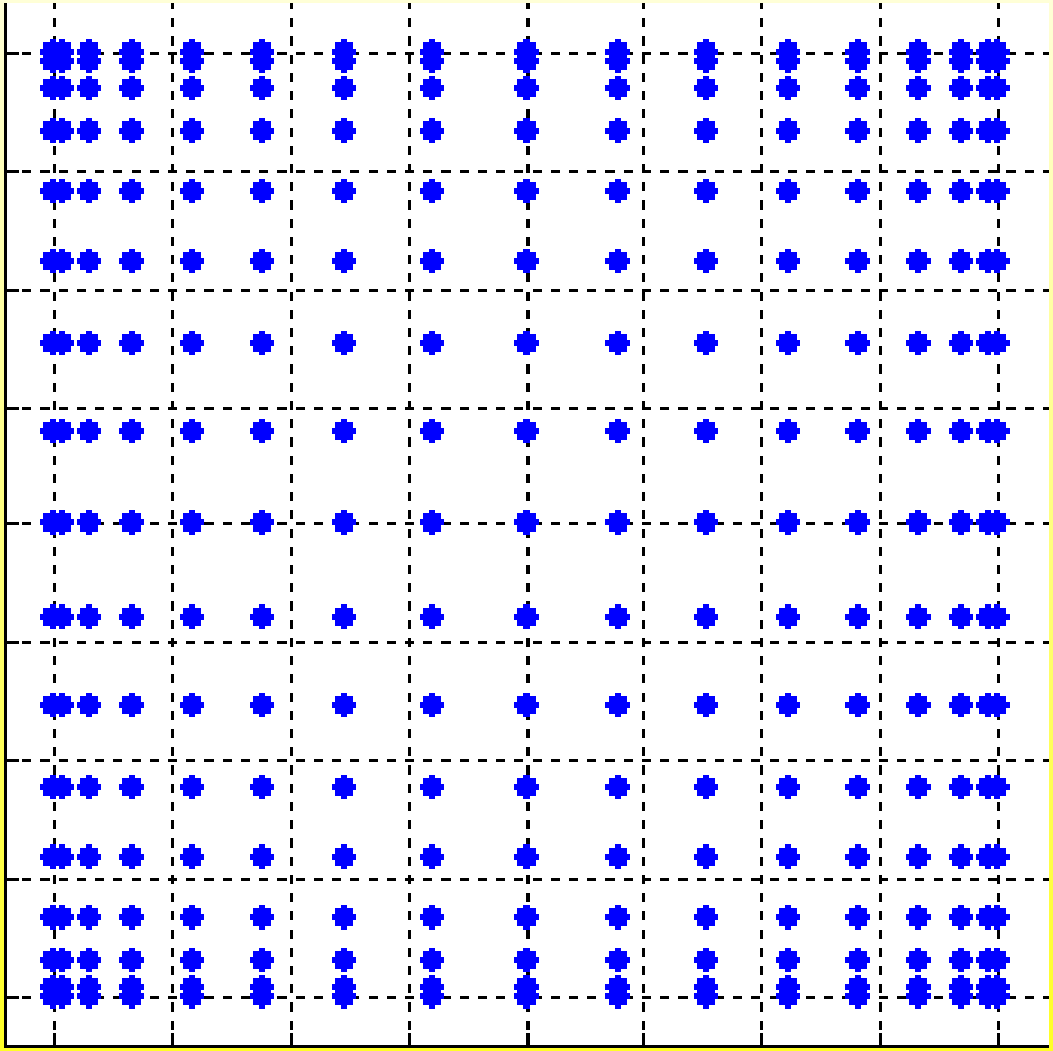
Smolyak Quadrature

Sergey Smolyak added low order grids together.

His combined "sparse grid":

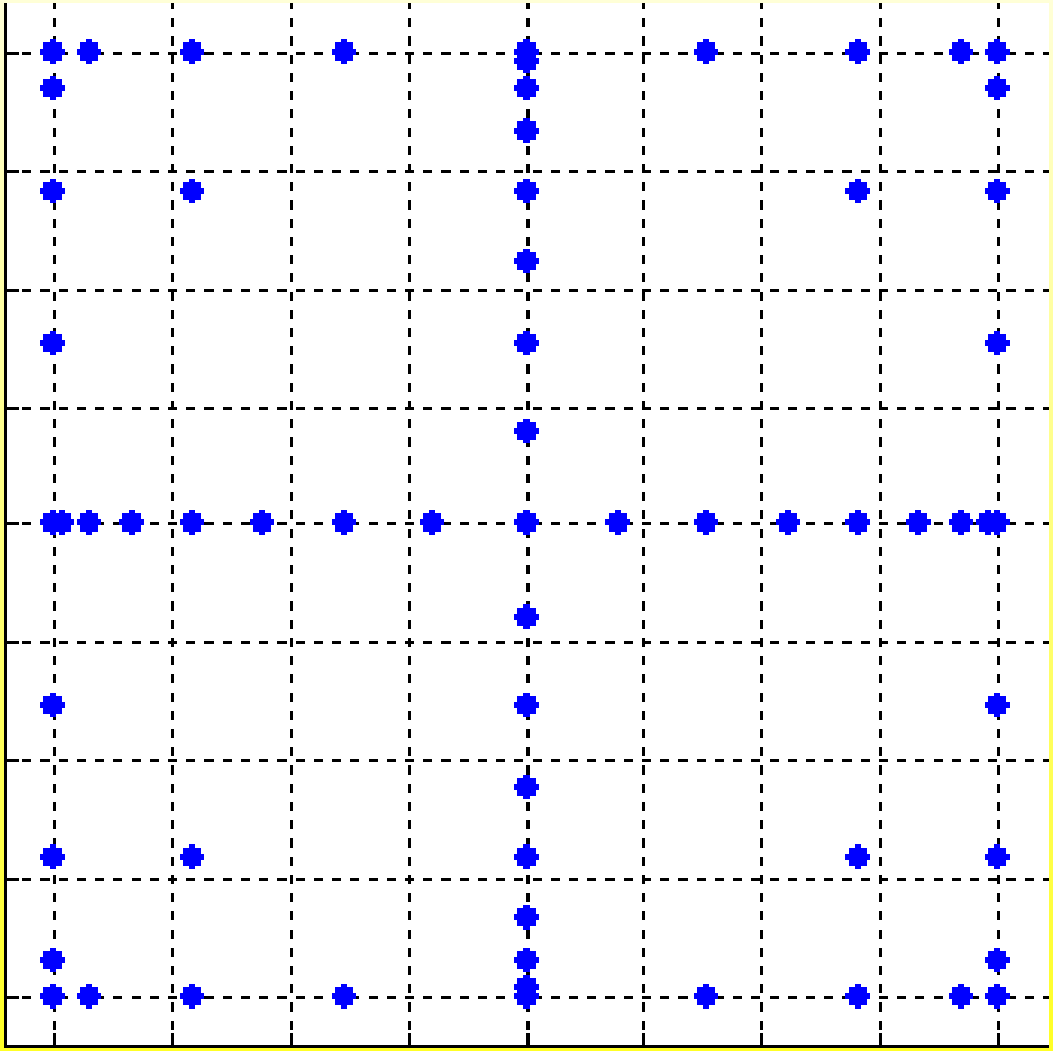
- had the same accuracy as a product grid.
- was a subset of the points of the product grid.
- used *far fewer* points.

2D Quadrature:



A 17x17 product grid (289 points).

2D Quadrature: Level4 Sparse Grid



An "equivalent" sparse grid (65 points).

2D Quadrature: Level4 Sparse Grid

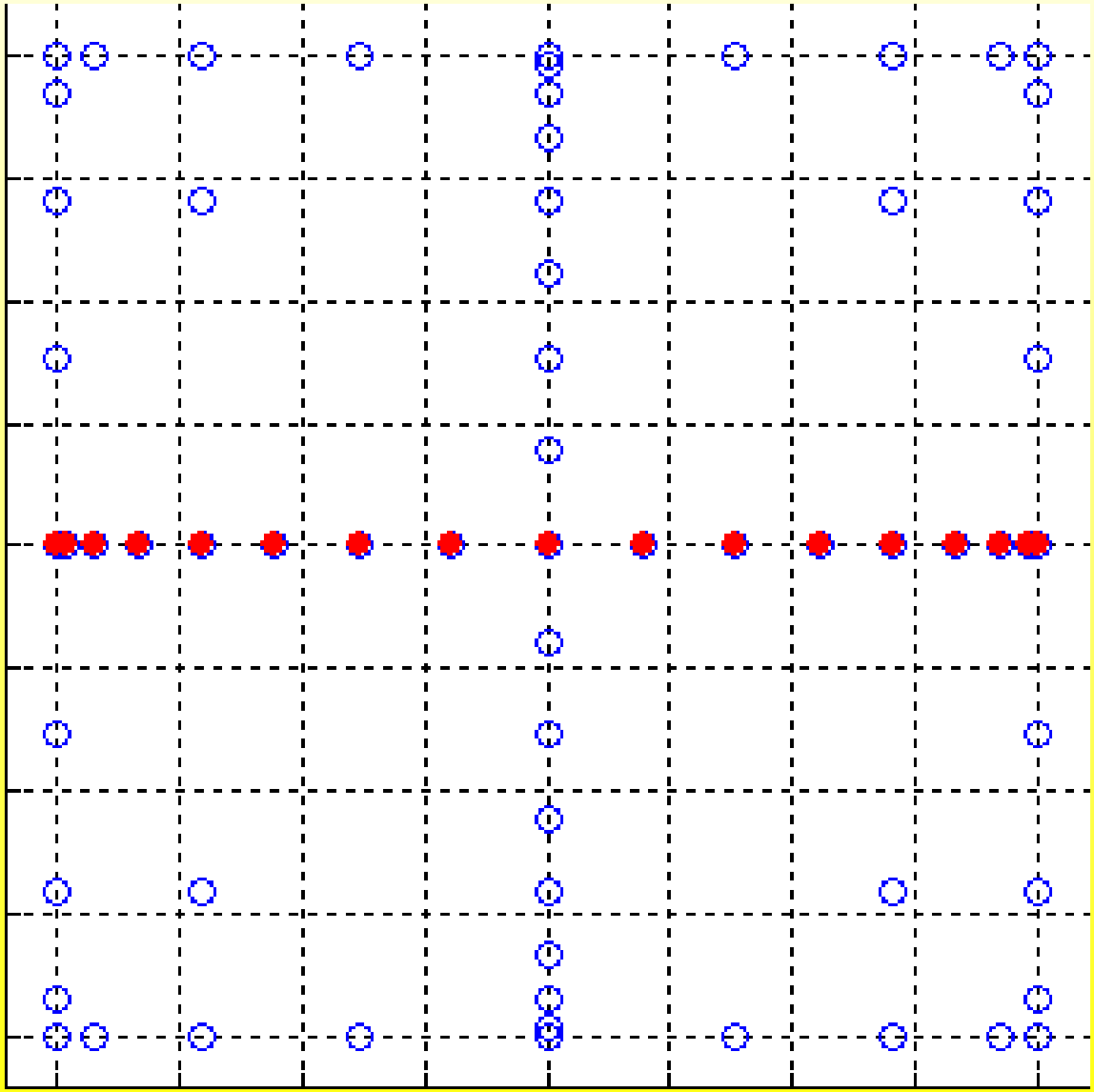
To capture only "desirable" monomials, we essentially add product grids which are sparse in one direction if dense in the other.

Because of nesting, the grids reuse many points.

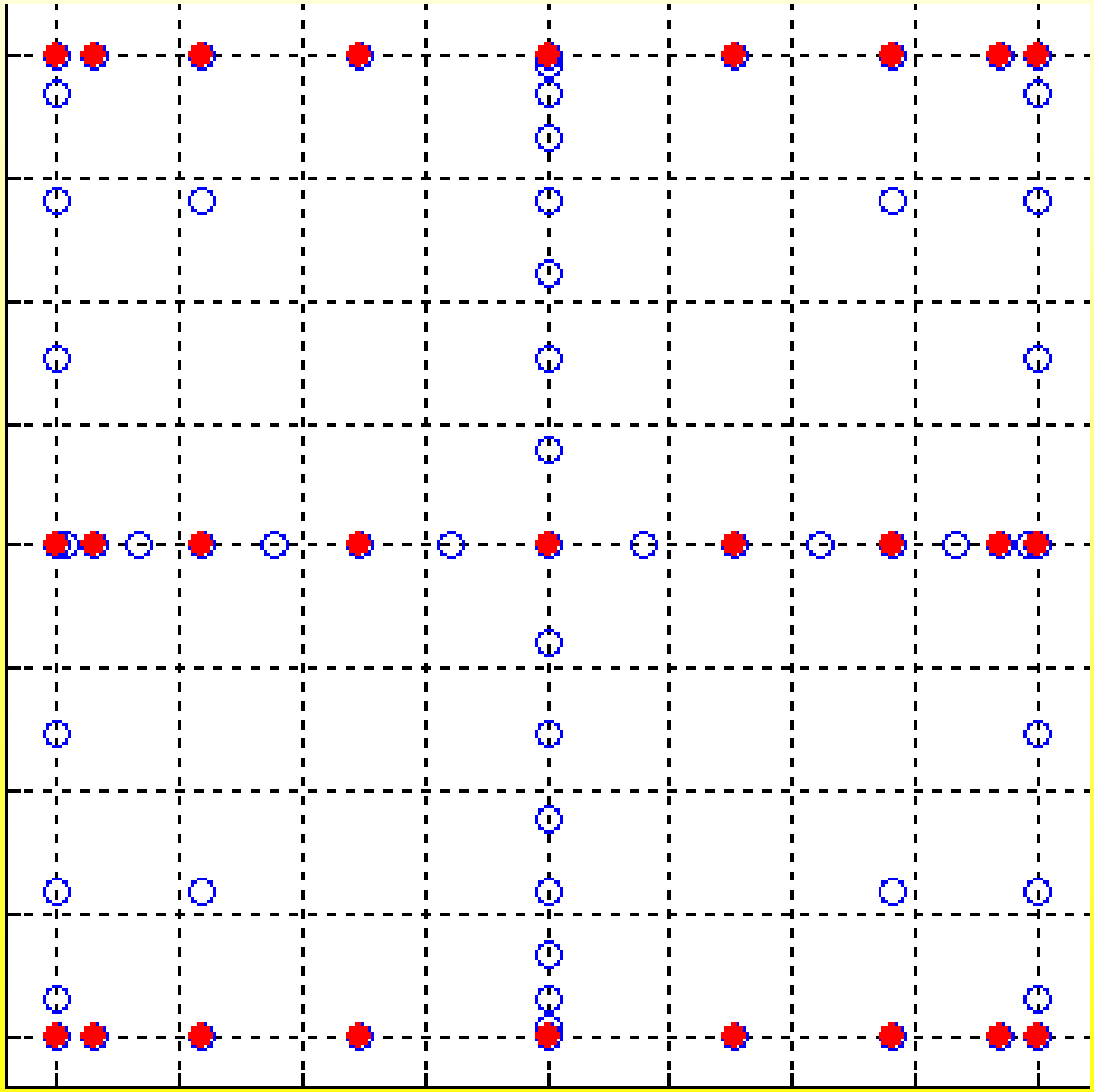
The big savings comes from entirely eliminating most of the points of the full product grid.

The improvement is greater as the dimension or level increases.

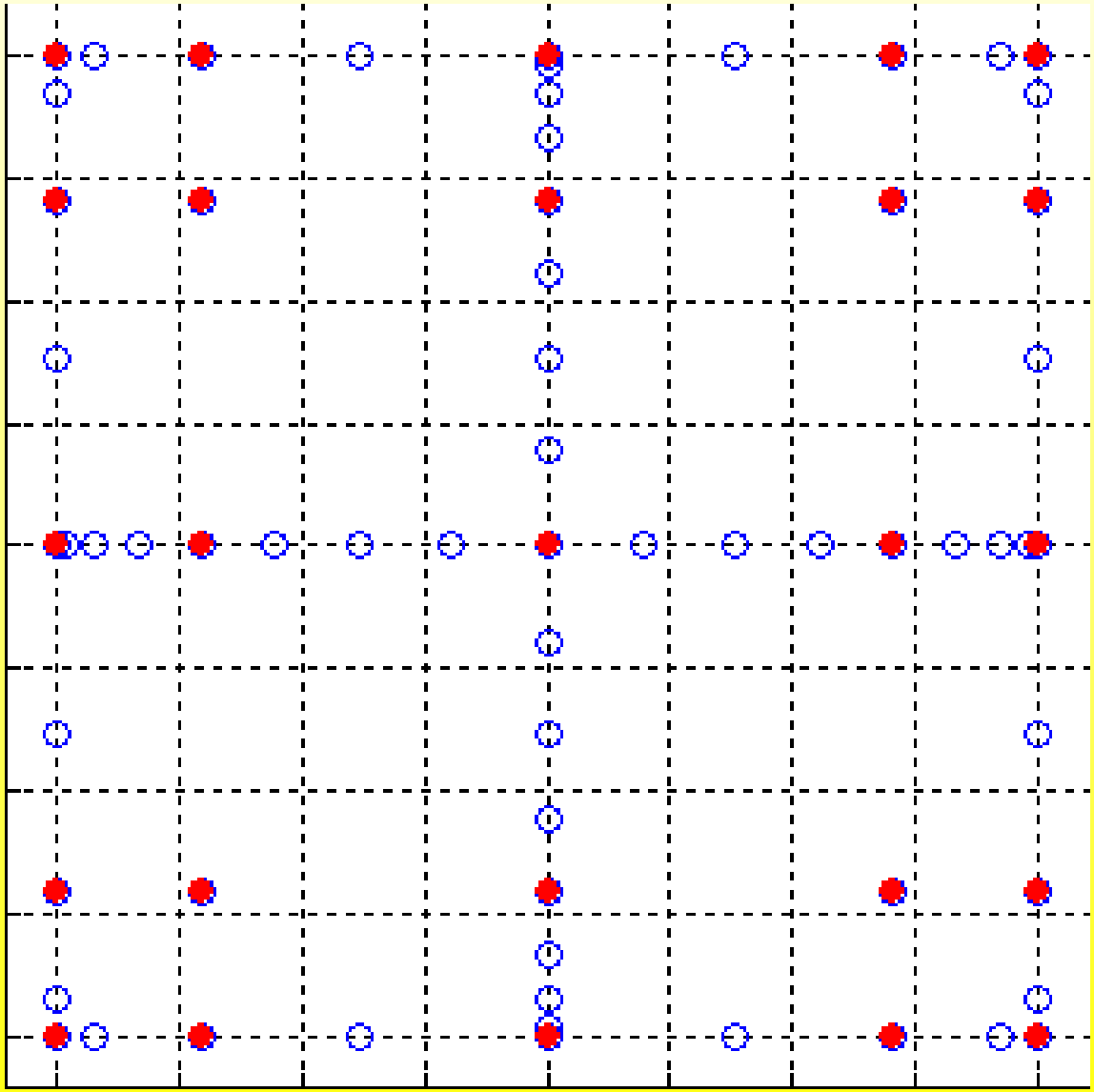
2D Quadrature: Level4 17x1 component



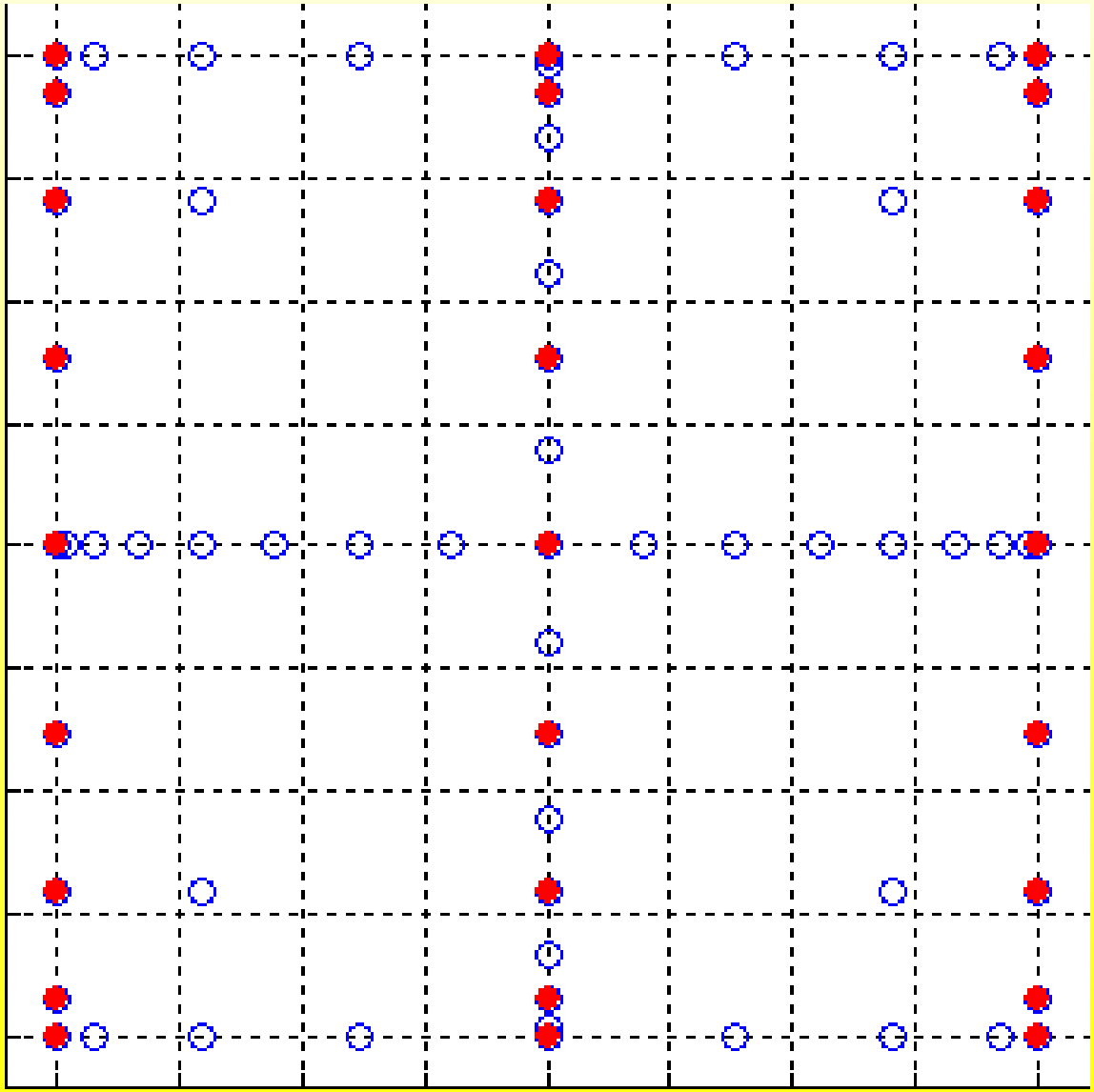
2D Quadrature: Level4 9x3 component



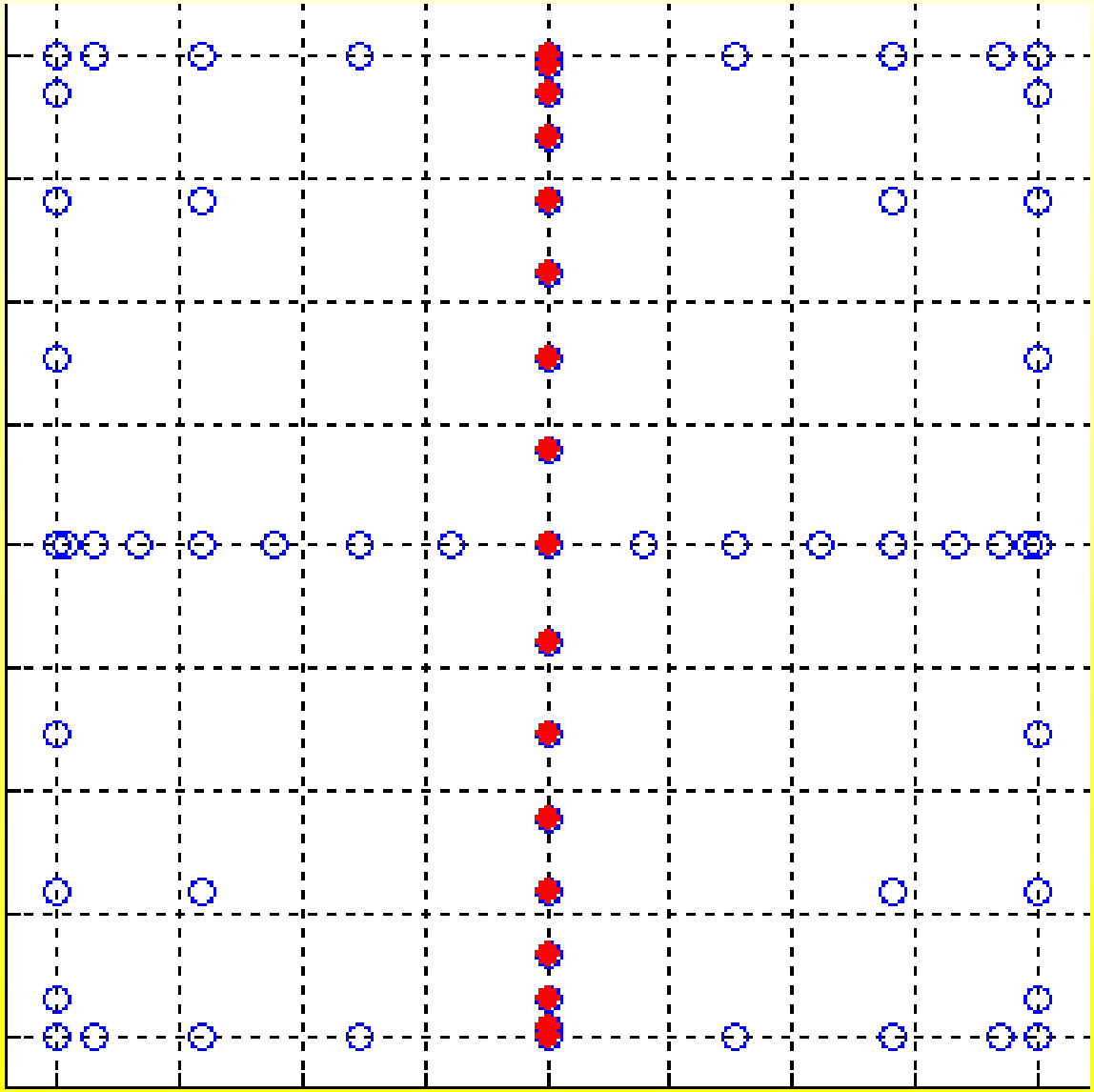
2D Quadrature: Level4 5x5 component



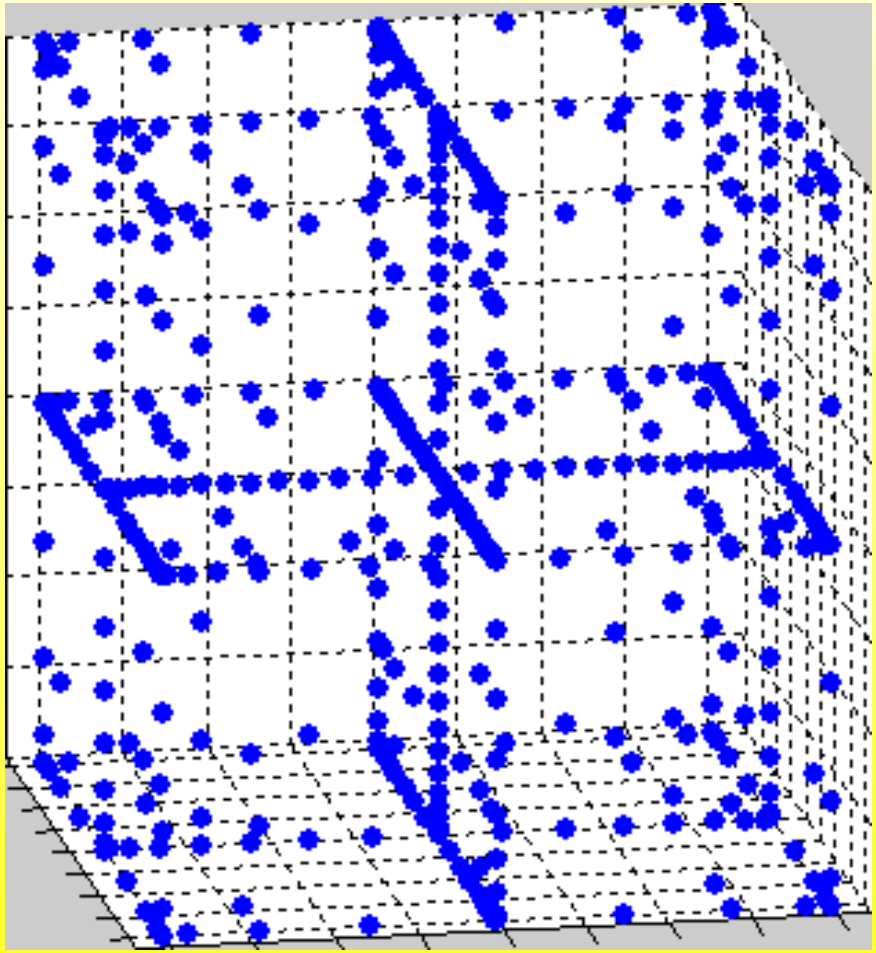
2D Quadrature: Level4 3x9 component



2D Quadrature: Level4 1x17 component

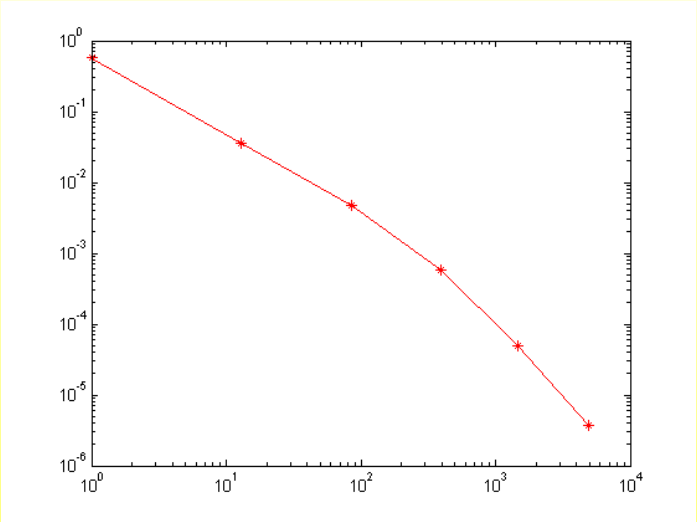


3D Quadrature: Level5 Sparse Grid



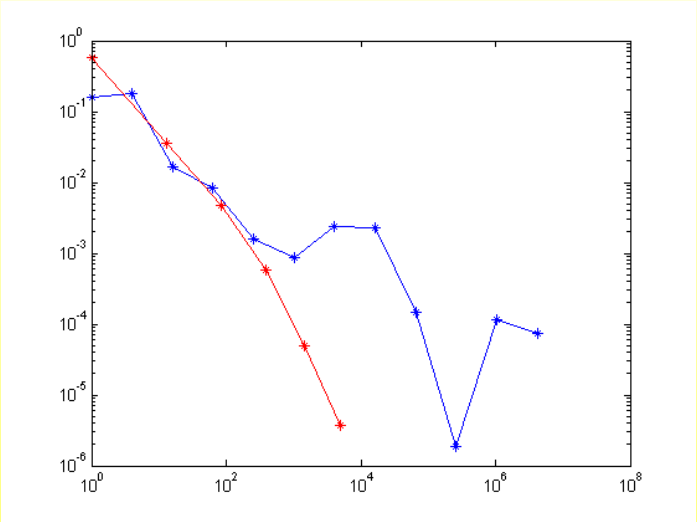
Sparse grid = 441 points; Product grid would have 35,937.

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 some quantity, like temperature;

a is the conductivity, which says how easily a hot place warms up nearby cold places.

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

The Diffusion Equation: Uncertain Conductivity

Suppose we only know a mean value of conductivity, and that there are significant, unknown, deviations from this mean.
Then a has a *stochastic* component ω, which our solution u will inherit.

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

If we pick ω, the equation becomes deterministic again.

But if we don't know ω, our solution is also unknowable!

Can we still extract information from the equation?

The Diffusion Equation: Expected Values

Even though our solution is "random", we can reasonably ask for the average or expected value; (and variances, and higher moments).

Once again, the *integral* is the perfect tool to carry out this operation. Instead of integrating over a range of numbers, we need to integrate over a range of stochastic perturbations ω from some function space Ω.

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

This averaging process is a little like modeling the *climate* instead of the *weather*.

The Diffusion Equation: Approximate Integral

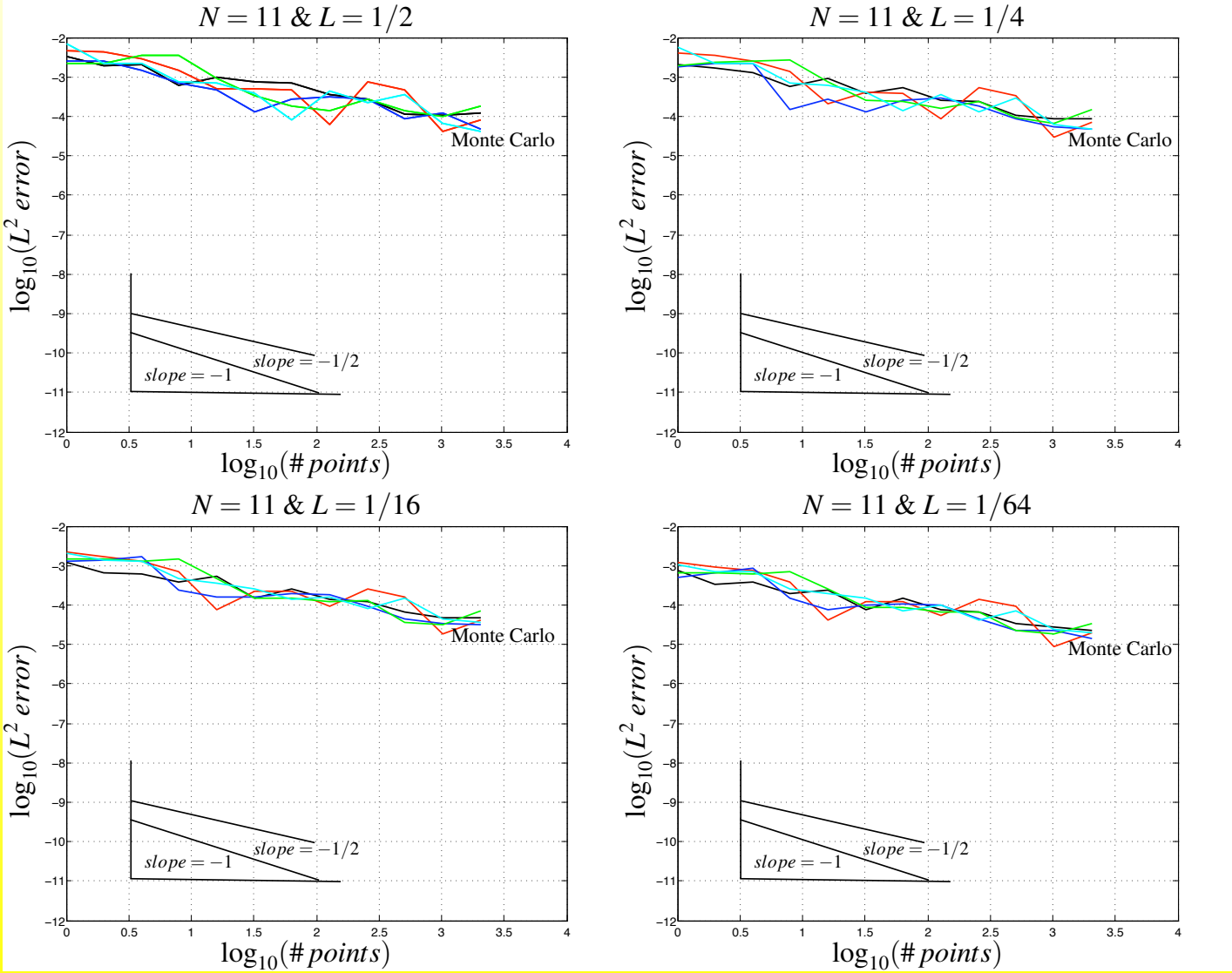
We approximate the function space Ω by an M-dimensional space Ω_M, of linear sums of perturbations ω_M.

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

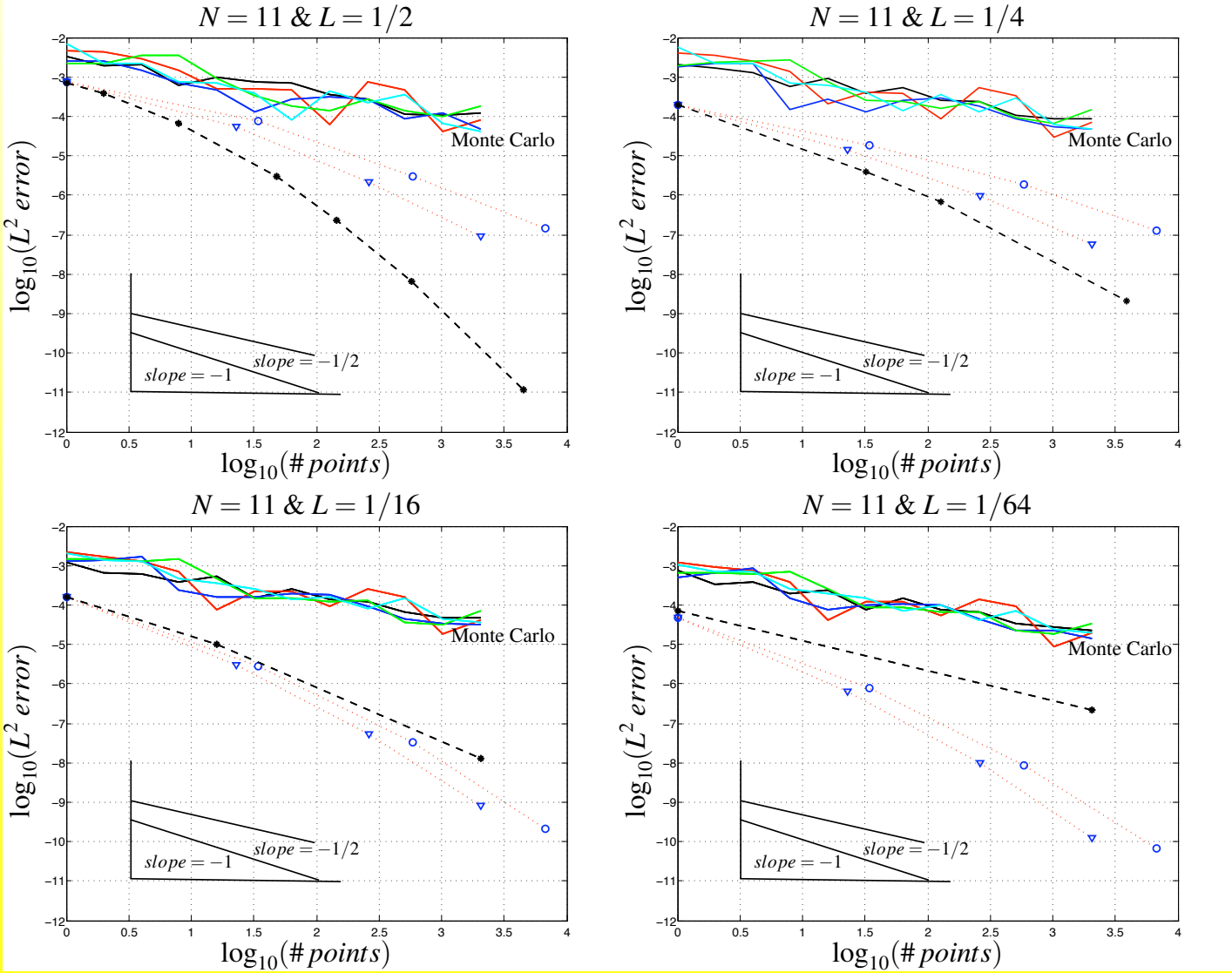
Monte Carlo: select a random set of parameters ω_M, solve for u, multiply by the probability, and average.

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

- Store quadrature rules explicitly, as tables.
- Make a "slightly" composite rule.
- Detect anisotropy in the data.
- Modify the algorithm so that some dimensions may be approximated more carefully.
- 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.