An Introduction to High Dimensional Sparse Grids

John Burkardt,
Information Technology Department,
Virginia Tech.
....
Mathematics Department,
Ajou University,
Suwon, Korea,
11 May 2009
....
http://people.sc.fsu.edu/∼jburkardt/presentations/sparse_intro_2009_ajou.pdf

Let the Exploration Begin!

An Introduction to High Dimensional Sparse Grids

Introduction: Physical Laws and Regions

Computational science begins with the simulation of physical laws.

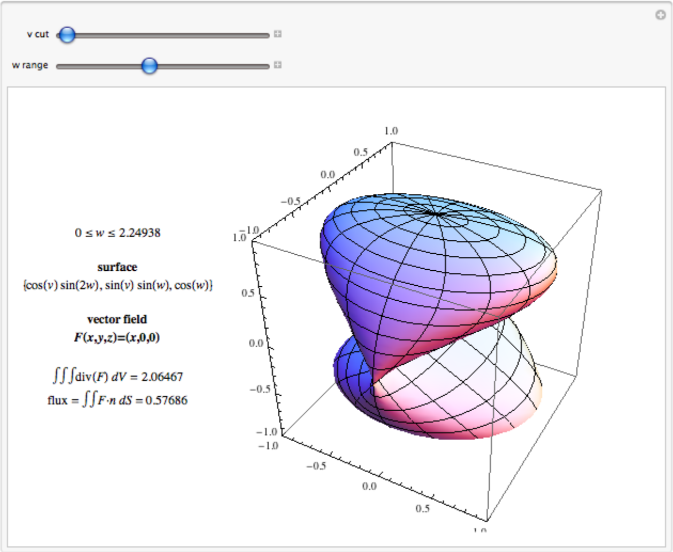
These laws often involve an integral operator applied to a function $f(x)$ over some integration region Ω.

Physical spaces are *low dimensional* (1D, 2D or 3D), so we don't worry about the effects of dimensionality.

If the space is low dimensional, we worry about:

- **shape** (polyhedral or curved boundaries)
- **embedding** (on the surface of a sphere, say).
- **nonsmoothness** of the integrand

Introduction: Physical Laws and Regions



Introduction: Nonphysical Laws and Regions

Mathematics has created nonphysical spaces where important problems can be posed.

Many problems arise from a probabilistic or stochastic setting.

A point p in a 10-dimensional space might represent the values of 10 physical parameters that affect an output quantity $g(p)$.

Perhaps the p come from a probabilistic space Ω, with a weighting function $w(\omega)$.

To compute the expected value of $\mathbf{g}$, we can write:

$$\overline{g(p)} = \frac{\int_\Omega \quad g(p(\omega)) \quad w(\omega) \quad d\omega}{\int_\Omega w(\omega)\, d\omega}$$

Introduction: Very High Dimensional Problems

Computational science explores problems in high dimensions:

- 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)

Introduction: Nonphysical Laws and Regions

Mathematically physical and nonphysical problems are the same.

But *computationally*, integration problems in high dimensional spaces often are quite different:

- **smoothness** of $f(x)$ is more likely;
- **geometry** of the integration region is simpler;
- **high dimensionality** becomes the greatest problem!

Introduction: The High Dimensional Challenge

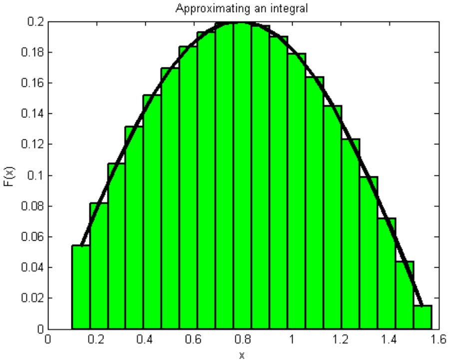
Even the simplest regions Ω and integrand functions $\mathbf{f(x)}$ may be almost impossible to integrate approximately if the dimension is too high!

We will discuss how computational approaches to multidimensional quadrature either break down, or are unable to produce accurate results, when the dimension becomes too high.

We will show that, for a particular kind of integrand and integration region, **sparse grids** can reach far into high dimensional space and extract the information we want.

We will discuss some software that is publicly available.

Sampling Quadrature: Approximating an 1D Integral



Quadrature allows us to estimate integrals.

In 1D, we can look at this picture as either **sampling** the integration region, or as **interpolating** the integration function.

Sampling Quadrature: 1D Monte Carlo

The *Monte Carlo* method (MC) views the integral as the average of sampled values.

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

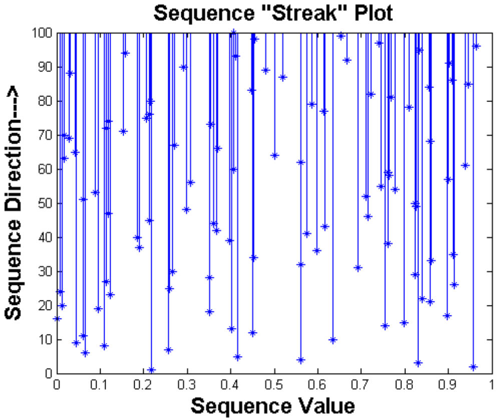
Sampling Quadrature: 1D Monte Carlo

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.

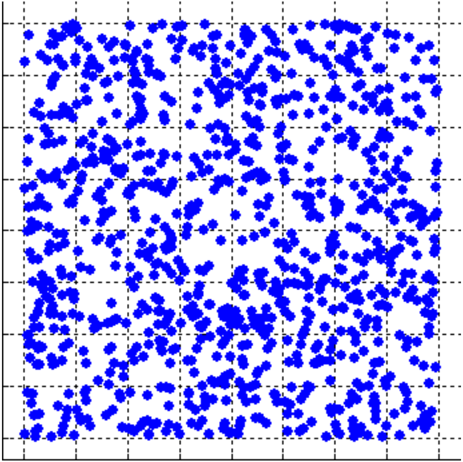
- If more accuracy needed, current values can be included;
- Accuracy hampered because of large “gaps” in sampling.
- Accuracy improvement rate is independent of spatial dimension.

Sampling Quadrature: 1D Monte Carlo



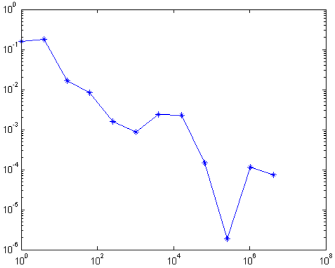
Notice the "gaps" and "clusters".

Sampling Quadrature: 2D Monte Carlo



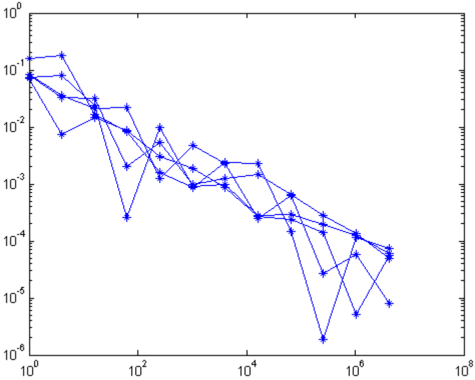
Notice the "gaps" and "clusters".

Sampling Quadrature: 6D Monte Carlo Error



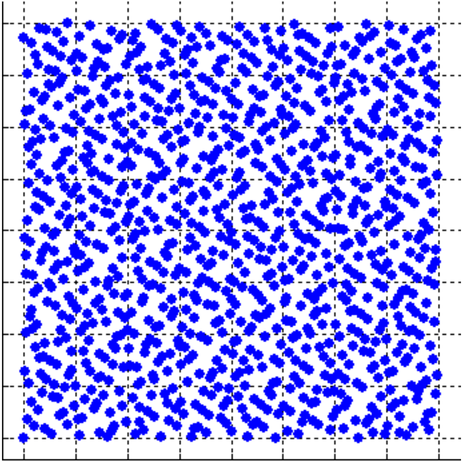
N	Estimate	Error
1	0.796541	0.160759
16	0.652621	0.016838
256	0.637351	0.001569
65536	0.635926	0.000144
4194304	0.635856	0.000074
∞	0.635782	0.0000

Sampling Quadrature: 6D Monte Carlo Error



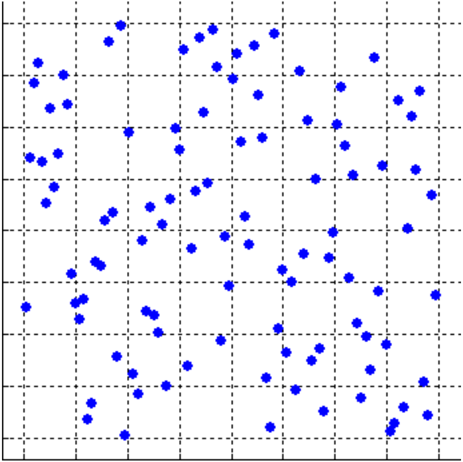
If we try five times, we get five different sets of results.
This data suggests that error decreases like $1/\sqrt{N}$.

Sampling Quadrature: 2D Quasi Monte Carlo



Quasi-Monte Carlo methods produce well spaced sampling.

Sampling Quadrature: 2D Latin Hypercube



Latin Hypercube Sampling ensures good spacing in each 1D component (but allows gaps and clusters in multidimensions.)

Sampling Quadrature: Ignore Properties of F(X)

Quadrature using sampling concentrates attention on sampling the geometry of the integration region.

The basic sampling method makes almost no assumptions about the smoothness of **f(x)**; the function values could be arbitrarily shuffled without affecting the result.

Sampling is robust - not easily affected by singularities or discontinuities.

While sampling error decreases slowly, the rate of decrease is independent of spatial dimension.

But if the function **f(x)** is smooth, a model of the function could be used to make a much more accurate integral estimate.

Interpolatory Quadrature: 1D Example

If the function $f(x)$ is "well-behaved", the sample values **F(X)** contain strong clues about $f(x)$ and its integral.

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.*

Interpolatory Quadrature: 1D Example

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

Evaluating at one point can give us the constant.

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

A second evaluation gives us the coefficient of x:

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

Evaluating at N points gives the first N coefficients.

Interpolatory Quadrature: Integrating Monomials

An approximate formula can be integrated exactly.

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

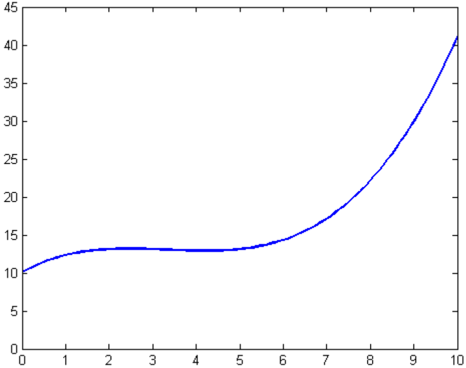
$$1, x, x^2, ..., x^{N-1},$$

and all functions made up of them.

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

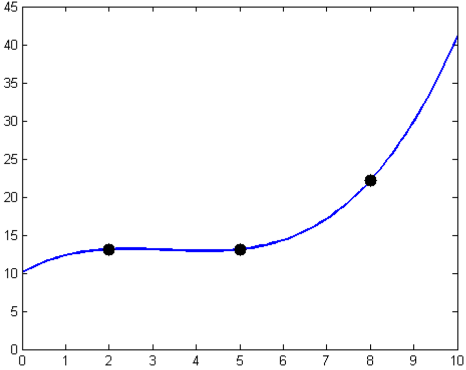
Increasing N increases the monomials we can "capture".

Interpolatory Quadrature: A Function to Integrate



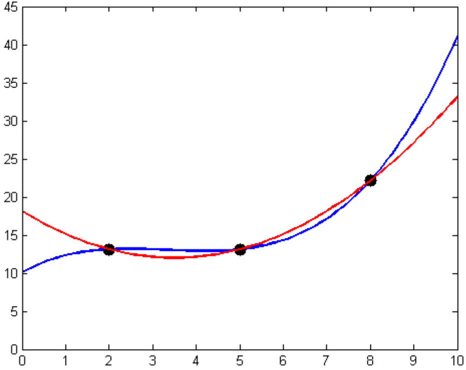
A function f(x) is given.

Interpolatory Quadrature: Selected Function Values



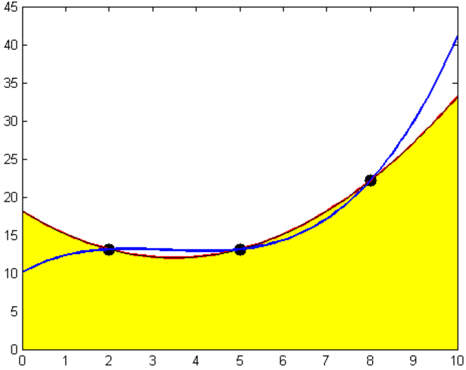
We evaluate it at N points.

Interpolatory Quadrature: The interpolant



We determine the approximating polynomial.

Interpolatory Quadrature: The integrated interpolant



We integrate the approximating polynomial exactly.

Interpolatory Quadrature: Features

- uses a regular grid of **N** points;
- interpolates up to the **P**th derivative (in 1D, **P** $=$ **N**-1);
- Evaluates each $f(x)$;
- Computes a *weighted* average of the function values.
- **The error can drop with an exponent of P+1.**

Of course, the function f(x) must be sufficiently smooth for the interpolation to be effective.

1. Introduction
2. Sampling Quadrature
3. Interpolatory Quadrature
4. **Product Rules**
5. Smolyak Quadrature
6. Numerical Software
7. File Format for Quadrature Rules
8. Conclusion

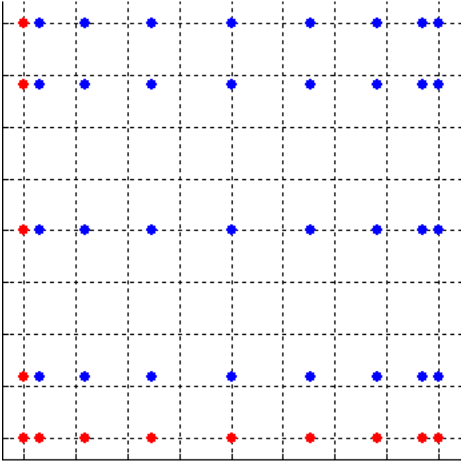
Product Rules

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

The number of points N in a product grid is the product $N = N_1 * N_2$ of the orders of the 1D rules.

The resulting rule captures monomials up to $x^{P_1} y^{P_2}$ where P_1 and P_2 are the individual precisions. So the precision is $P = \min(P_1, P_2)$.

Product Rules: a 9x5 rule



A product of 9 point and 5 point rules.

Product Rules: Point Growth with Dimension

Suppose we take products of a modest 4 point rule:

- 1D: 4 points;
- 2D: $4^2 = 16$ points;
- 3D: $4^3 = 64$ points;
- 4D: $4^4 = 256$ points;
- 5D: $4^5 = 1024$ points;
- 10D:4^{10} = a million points;
- 20D: 4^{20} = a trillion points.
- 100D: not representable on a computer!

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

Product Rules: Component Degree and Total Degree

In multi-dimensions, what is the **DEGREE** of a monomial?

If we consider the **component degree**, (the maximum of the degrees of the component variables) then monomials of component degree 4 include x^4 and x^3y^2 and even x^4y^4.

If we consider the **total degree**, we sum all the exponents. Then monomials of total degree 4 are exactly

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

The asymptotic accuracy of a quadrature rule is determined by the highest **total degree** N for which we can guarantee that all monomials will be integrated exactly.

As soon as we miss one monomial of a given total degree, our rule will have "run out of accuracy".

Product Rules: Only Complete Rows Help!

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					

Because a product rule misses x^5 and y^5, the other monomials below the line don't help asymptotically.

Product Rules

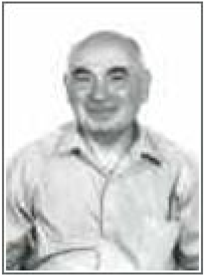
As the dimension increases, the useless monomials predominate.

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.

Each of his combined "sparse grids":

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

We have an indexed family of 1D quadrature rules $\mathcal{U}^i$.
We form dimension **d** rules, indexed by "level" **q** starting at **d**.
Here $\mathrm{i} = i_1 + \cdots + i_d$.

$$\mathcal{A}(q,d) = \sum_{q-d+1 \le |\mathrm{i}| \le q} (-1)^{q-|\mathrm{i}|} \begin{pmatrix} d-1 \\ q-|\mathrm{i}| \end{pmatrix} (\mathcal{U}^{i_1} \otimes \cdots \otimes \mathcal{U}^{i_d})$$

Thus, the rule $\mathcal{A}(q,d)$ is a weighted sum of product rules.

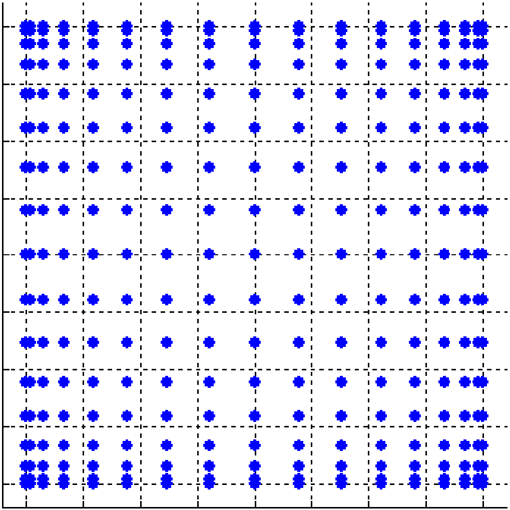
Smolyak Quadrature: Point Growth and Precision

The 10D point count is an example of how N grows.

Level L	1D count	10D count N	Precision P
0	1	1	1
1	3	21	3
2	5	221	5
3	9	1581	7
4	17	8801	9
5	33	41265	11
6	65	171425	13

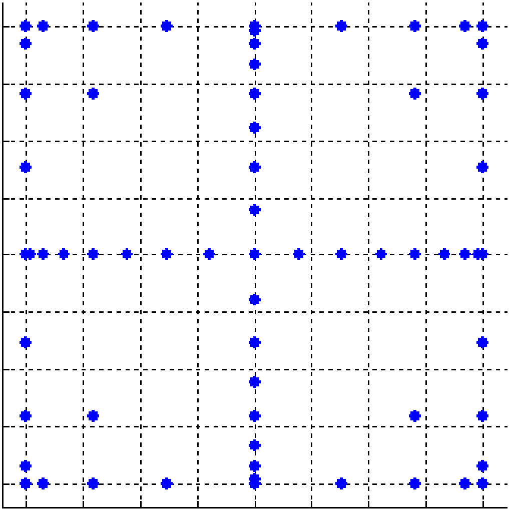
Precision P = 2 * L + 1 for any dimension above 1.

Smolyak Quadrature: 2D Order 17 Product Rule



A 17x17 product grid (289 points).

Smolyak Quadrature: 2D Level4 Smolyak Grid



A sparse grid of Level 4 (65 points).

Smolyak Quadrature

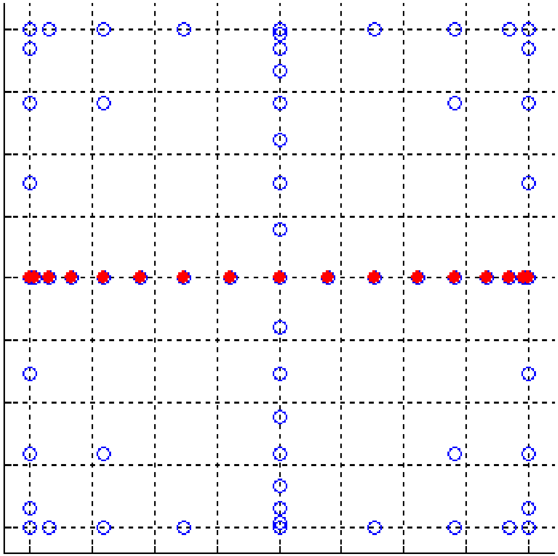
To capture only “desirable” monomials, we essentially add product grids which are sparse in one direction if dense in the other.

Because of nesting, the grids reuse many points.

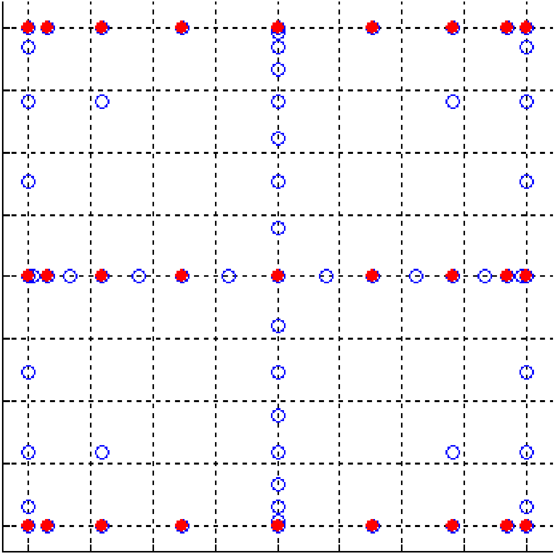
The big savings comes from entirely eliminating most of the points of the full product grid.

The improvement is greater as the dimension or level increases.

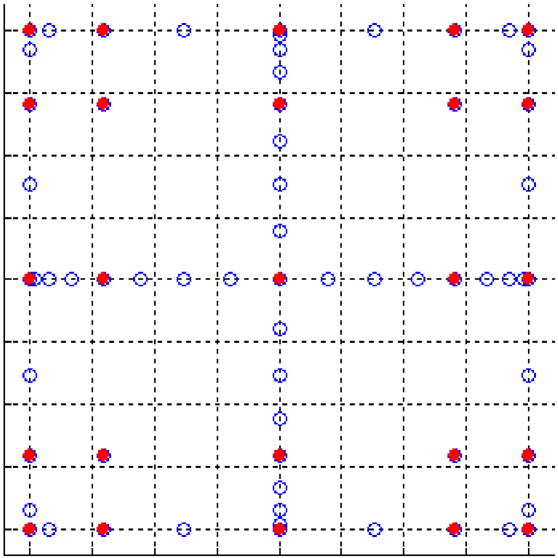
Smolyak Quadrature: 2D Level4 17x1 component



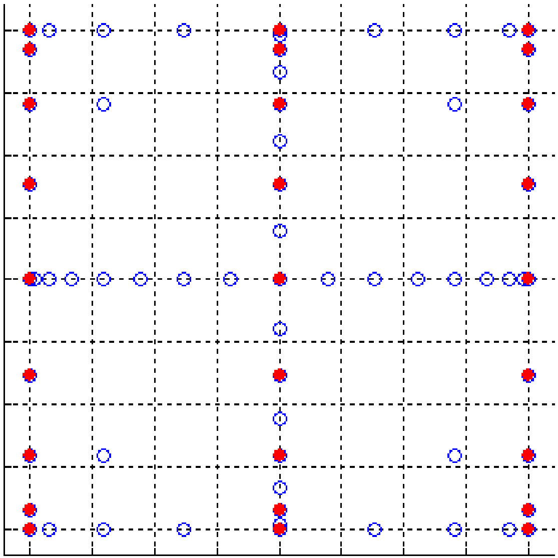
Smolyak Quadrature: 2D Level4 9x3 component



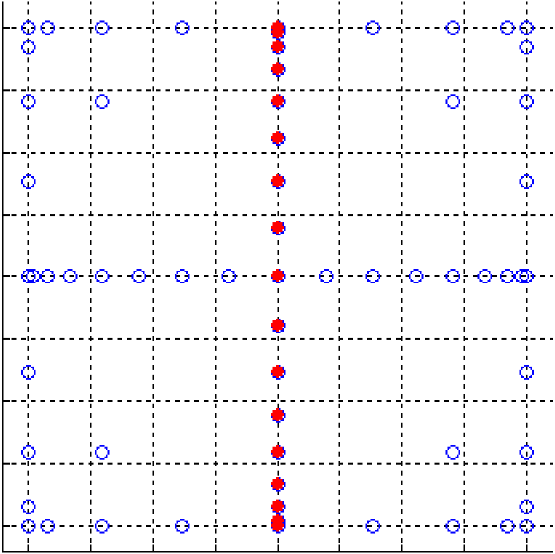
Smolyak Quadrature: 2D Level4 5x5 component



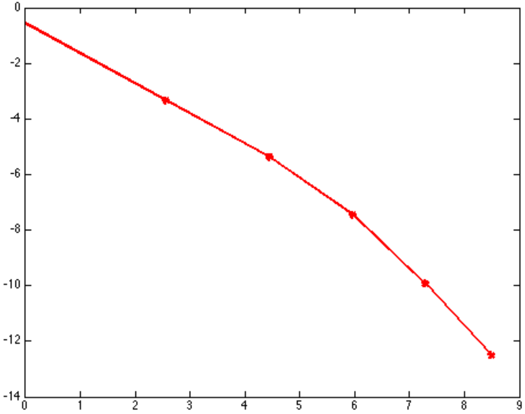
Smolyak Quadrature: 2D Level4 3x9 component



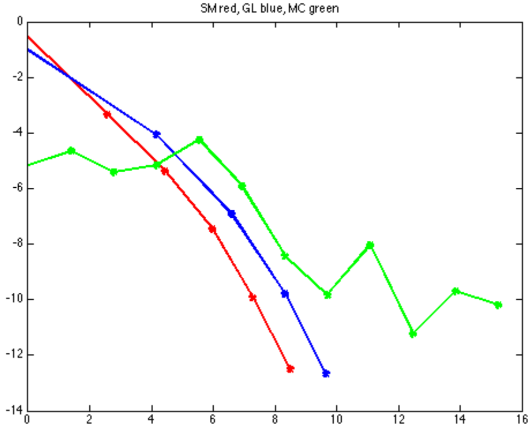
Smolyak Quadrature: 2D Level4 1x17 component



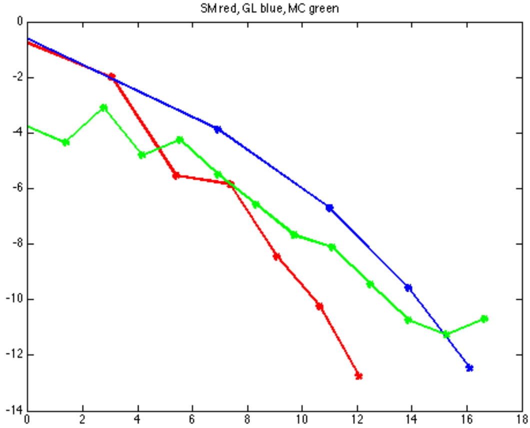
Smolyak Quadrature:6D Smolyak



Smolyak Quadrature: 6D Smolyak/GL/MC

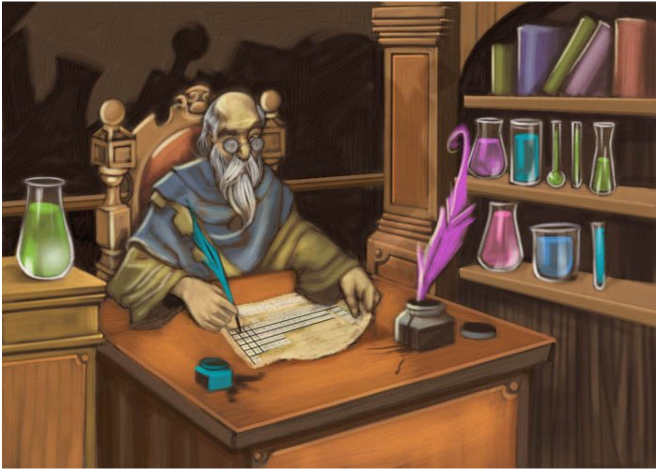


Smolyak Quadrature: 10D Smolyak/GL/MC



Numerical Software

Smolyak's definition of sparse grids is almost magical; but it can take the novice a while to master the tricks. So it's important to bottle some of that magic in accessible tools!



Numerical Software: L, N, and D

The family of sparse grid rules is indexed by **L**, the level.

L starts at 0 which corresponds to the $\mathbf{N} = 1$ point rule.

As **L** increases, the growth in the number of points **N** depends on **L**, the spatial dimension **D**, and the nesting of the underlying rule.

Numerical Software: How N Grows

For a Clenshaw Curtis rule, here is how **N** increases with **L** and **D**.

D L	1	2	3	4	5	10
0	1	1	1	1	1	1
1	3	5	7	9	11	21
2	5	13	25	41	61	221
3	9	29	69	137	241	1581
4	17	65	177	401	801	8801
5	33	145	441	1105	2433	41265
6	65	321	1073	2929	6993	171425

Numerical Software: Packages Using a Single Rule

When the same rule is used for each dimension, there are sparse grid packages available for several families:

The routines of interest to a user are:

- **sparse_grid_cc** the Clenshaw Curtis rule for [-1,+1]
- **sparse_grid_gl** the Gauss-Legendre Rule for [-1,+1]
- **sparse_grid_laguerre** the Gauss-Laguerre Rule for [0,∞)
- **sparse_grid_hermite** the Gauss-Hermite Rule for (-∞,∞)

Numerical Software: SPARSE_GRID_LAGUERRE

For example, if you are interested in sparse grid rules based on the 1D **laguerre** rule, then the package **sparse_grid_laguerre** is available, in C++, FORTRAN90 and MATLAB.

The routines of interest to a user are:

- **sparse_grid_laguerre_size** returns the number of points
- **sparse_grid_laguerre** returns the weights and abscissas
- **monomial_quadrature** tests a quadrature rule on a monomial

Numerical Software: SPARSE_GRID_LAGUERRE

To set up a MATLAB program using **sparse_grid_laguerre** to integrate function **f(x)**, you need to

- write a function that evaluates **f(x)** ;
- define the spatial dimension **D**
- choose a sparse grid level **L**
- give **D** and **L** to **sparse_grid_laguerre_size** to get **N**
- give **D**, **L** and **N** to **sparse_grid_laguerre** to get **W** and **X**
- evaluate **F** at the points **X**, and weight the sum by **W** to form the estimate

Numerical Software: Use a Sparse Laguerre Rule

```
D = 6;

 for L = 0 : 5

   N = sparse_grid_laguerre_size ( D, L );

   [ W, X ] = sparse_grid_laguerre ( D, L, N  );

   quad = W  * F(X)': %  quad = W(1:N) * F ( X(1:D,1:N) )

   fprintf ( 1, '  L = %d, N = %d, quad = %f\n',
     L, N, quad );

 end
```

Numerical Software: Choices for Rules

A sparse grid rule is the sum of product rules. In the simplest case, the product rules are products of a single 1D quadrature rule.

Common choices for the 1D quadrature rule include:

- **CC**, Clenshaw Curtis
- **F2**, Fejer Type 2 rule
- **GL**, Gauss-Legendre rule
- **GJ**, Gauss-Jacobi rule
- **LG**, Gauss-Laguerre rule
- **GLG**, Generalized Gauss-Laguerre rule
- **GH**, Gauss-Hermite rule
- **GGH**, Generalized Gauss-Hermite rule

Numerical Software: Relation with Stochastic Variables

Some of the rules are especially useful for stochastic problems.

When solving stochastic problems using polynomial chaos, the distribution of the variables is related to the kind of quadrature rule needed.

Random variable	Domain	Quadrature rule
uniform	$[-1, +1]$	Gauss-Legendre
gaussian	$(-\infty, +\infty)$	Gauss-Hermite
gamma	$[0, +\infty)$	Gauss-Laguerre
beta	$[-1, +1]$	Gauss-Jacobi

Numerical Software: A Package Using Multiple Rules

A complicated problem may required different quadrature rules in different dimensions. In that case, you may use the package **sparse_grid_mixed**.

This package allows the user to request any combination of the rules mentioned earlier (including Fejer 2, Generalized Laguerre and Hermite, Jacobi).

The interface to this software is more complicated, since the rule for each dimension must be specified, and some of those rules require extra information.

File Format

A file format for quadrature rules means that software programs can communicate;

Results can be precomputed.

Files can easily be checked, corrected, plotted, emailed.

The basic format uses 3 files:

- **R file**, 2 lines, D columns, the "corners" of the region
- **W file**, N lines, 1 column, the weight for each abscissa
- **X file**, N lines, D columns, the abscissas

File Format

The "columns" are simply numbers separated by blanks.

A single file could have been used, but it would have internal structure.

To determine D and N, a program reads the X file and counts the number of "words" on a line, and the number of lines.

No particular ordering for the abscissas is assumed, but each line of the W and X files must correspond.

I have used this format for a 3x3 Clenshaw Curtis product rule and a sparse grid rule for integration in 100D!

File Format

```
        R file
       ---------
       -1.0 -1.0
       +1.0 +1.0

W file         X file
------       -----------
0.111        -1.0  -1.0
0.444        -1.0   0.0
0.111        -1.0  +1.0
0.444         0.0  -1.0
1.777         0.0   0.0
0.444         0.0  +1.0
0.111        +1.0  -1.0
0.444        +1.0   0.0
0.111        +1.0  +1.0
```

File Format

Another advantage of exporting quadrature rules to a file is that it is possible to precompute a desired family of rules and store them.

These files can be read in by a program written in another computer language; they can be mailed to a researcher who does not want to deal with the initial rule generation step.

File Format: Precision Testing

Once we have quadrature rules stored in files, we can easily run degree of precision tests.

An executable program asks the user for the quadrature file names, and M, the maximum polynomial degree to check.

The program determines the spatial dimension D implicitly from the files, as well as N, the number of points.

It then generates every appropriate monomial, applies the quadrature rule, and reports the error.

File Format: Precision Checking

```
23 October 2008   8:04:55.816 AM

NINT_EXACTNESS
  C++ version

  Investigate the polynomial exactness of a quadrature
  rule by integrating all monomials of a given degree
  over the [0,1] hypercube.

NINT_EXACTNESS: User input:
  Quadrature rule X file = "ccgl_d2_o006_x.txt".
  Quadrature rule W file = "ccgl_d2_o006_w.txt".
  Quadrature rule R file = "ccgl_d2_o006_r.txt".
  Maximum total degree to check =        4
  Spatial dimension =        2
  Number of points  =        6
```

File Format: Precision Checking

Error	Degree	Exponents
0.0000000000000001	0	0 0
0.0000000000000002	1	1 0
0.0000000000000002	1	0 1
0.0000000000000002	2	2 0
0.0000000000000002	2	1 1
0.0000000000000002	2	0 2
0.0000000000000002	3	3 0
0.0000000000000002	3	2 1
0.0000000000000000	3	1 2
0.0000000000000001	3	0 3
0.0416666666666665	4	4 0
0.0000000000000001	4	3 1
0.0000000000000000	4	2 2
0.0000000000000001	4	1 3
0.0277777777777779	4	0 4

1. Introduction
2. Sampling Quadrature
3. Interpolatory Quadrature
4. Product Rules
5. Smolyak Quadrature
6. Numerical Software
7. File Format for Quadrature Rules
8. **Conclusion**

Conclusion: Future Work

- Precompute quadrature rules, for parallel application
- Careful use of “partial” composite rules.
- Detect **anisotropy** in the data (some dimensions more important).
- Respond to **anisotropy** (use higher degrees in some dimensions).
- Estimate quadrature error.
- Work with Laguerre, Hermite and other rules of interest in stochastic problems

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
- Abstract probability spaces may generate suitably 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**.

Conclusion: Software

On my web page, look at
http://people.sc.fsu.edu/~jburkardt/f_src/sparse_grid_cc /sparse_grid_cc.html

- **f_src** for FORTRAN90 code
- **cpp_src** for C++ code
- **m_src** for MATLAB code.

There are packages for sparse grids based on Gauss-Legendre, Laguerre, and Hermite rules, and arbitrary mixtures of rules.

The file **sandia_rules** gathers many 1D quadrature rules together, and **sparse_grid_mixed** allows the user to specify mixed rules.

The programs **nint_exactness** and **nint_exactness_mixed** test polynomial accuracy of a product rule or sparse grid rule.

Conclusion: References

Volker **Barthelmann**, Erich **Novak**, Klaus **Ritter**,
High Dimensional Polynomial Interpolation on Sparse Grids,
Advances in Computational Mathematics, 12(4) 2000, p273-288.

John **Burkardt**, Max **Gunzburger**, Clayton **Webster**,
Reduced Order Modeling of Some Nonlinear Stochastic PDE's,
IJNAM, 4(3-4) 2007, p368-391.

Thomas **Gerstner**, Michael **Griebel**,
Numerical Integration Using Sparse Grids,
Numerical Algorithms, 18(3-4), 1998, p209-232.

Sergey **Smolyak**,
Quadrature and Interpolation Formulas for Tensor Products of Certain Classes of Functions,
Doklady Akademii Nauk SSSR, 4, 1963, p240-243.