# Approximating Integrals for Stochastic Problems

John Burkardt
Department of Scientific Computing
Florida State University

.....

ISC 5936-01:
Numerical Methods for Stochastic Differential Equations
http://people.sc.fsu.edu/~jburkardt/presentations/...
stochastic_integrals.pdf

.....

Revised: 20 March 2013

26/28 February, 5/7/19/21 March 2013

# Where Did the PDE Go?

The problems we want to work on involve partial differential equations.

But, as Acting Professor Hans-Werner van Wyk suggested in his lecture, we can think of the PDE's as being a "black box" that operates in some unknown way on our input $\vec{x}$ to produce our output $\vec{u}(\vec{x})$, based on which we can compute a quantity of interest $QoI = q(\vec{u}(\vec{x}))$.

We want to think about adding uncertainty to the input $\vec{x}$. Here, by "input", we might mean the initial conditions, boundary conditions, right hand side, coefficients, or even the geometry of the problem. But we can still wrap up all these quantities into the mysterious input $\vec{x}$.

# Integration Estimates Uncertainty

If our input is uncertain, so is our output. Instead of solving one problem, we need to imagine solving every problem, and then saying something about the average or expected result.

This will be done using integration, and we can talk about integration as an abstract process that looks something like

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

The uncertainty is hidden in $\rho(x)$, the PDE is hidden inside of $f(x)$.

My lectures will concentrate on the questions of setting up an integral involving an uncertain quantity, and estimating that integral, with a final discussion of what happens if the integration space is high dimensional.

# One Page Summary

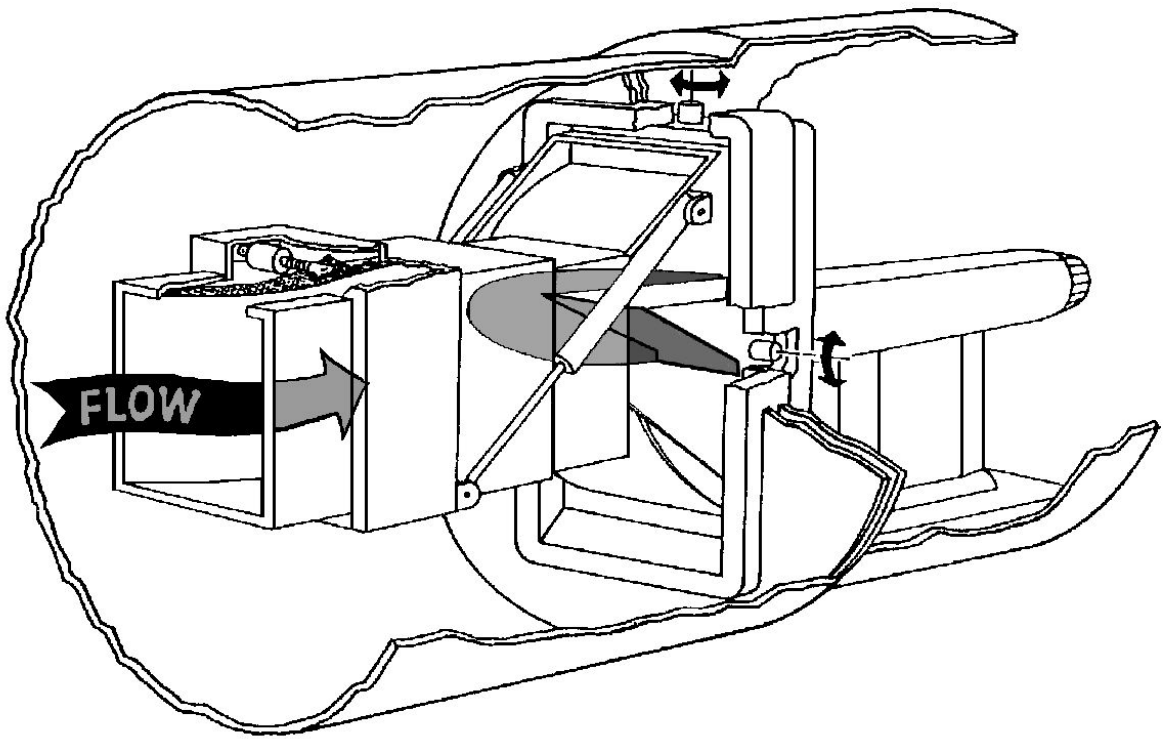
We need **integrals** to formulate the stochastic partial differential equations in a way that allows us to determine the solution functions, or (more typically) the statistical quantities that characterize them.

We need **high dimensional integrals** because the stochastic influence on the PDE typically involves a potentially unlimited number of factors or coefficients.

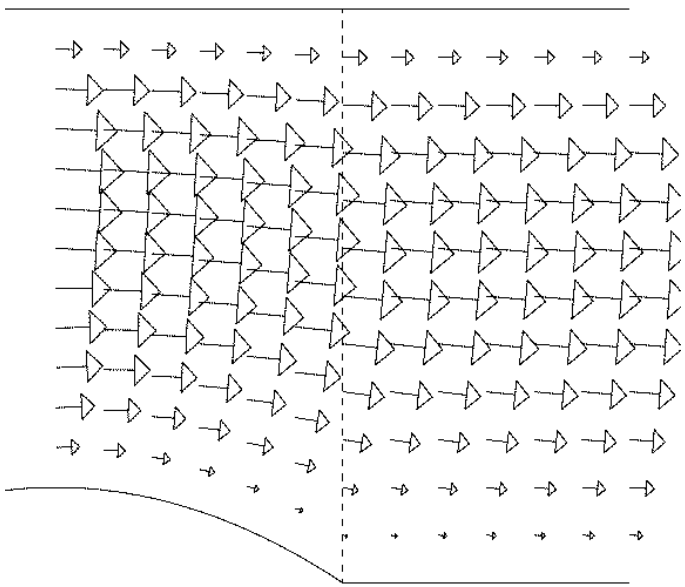
We need to **approximate high dimensional integrals** because closed-form solutions are unavailable.

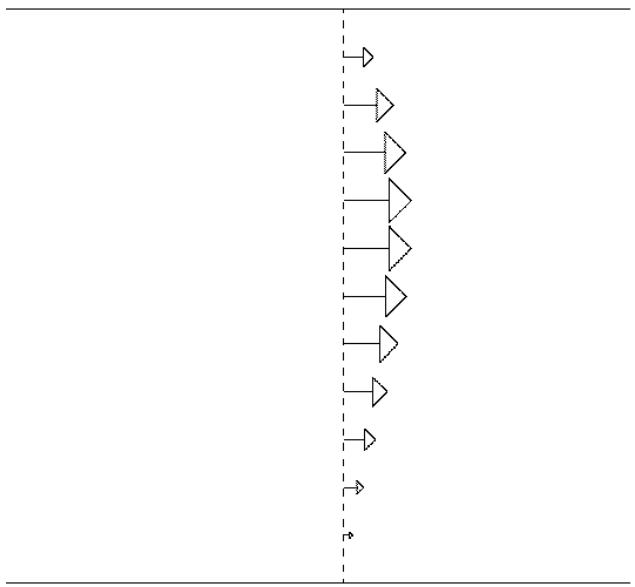
We may need **sparse grids to approximate high dimensional integrals** because the standard approximation methods for high dimensional integrals quickly become overloaded with excessive (and unnecessary!) work.

# "Reality" (Actually, an Engineering Model)

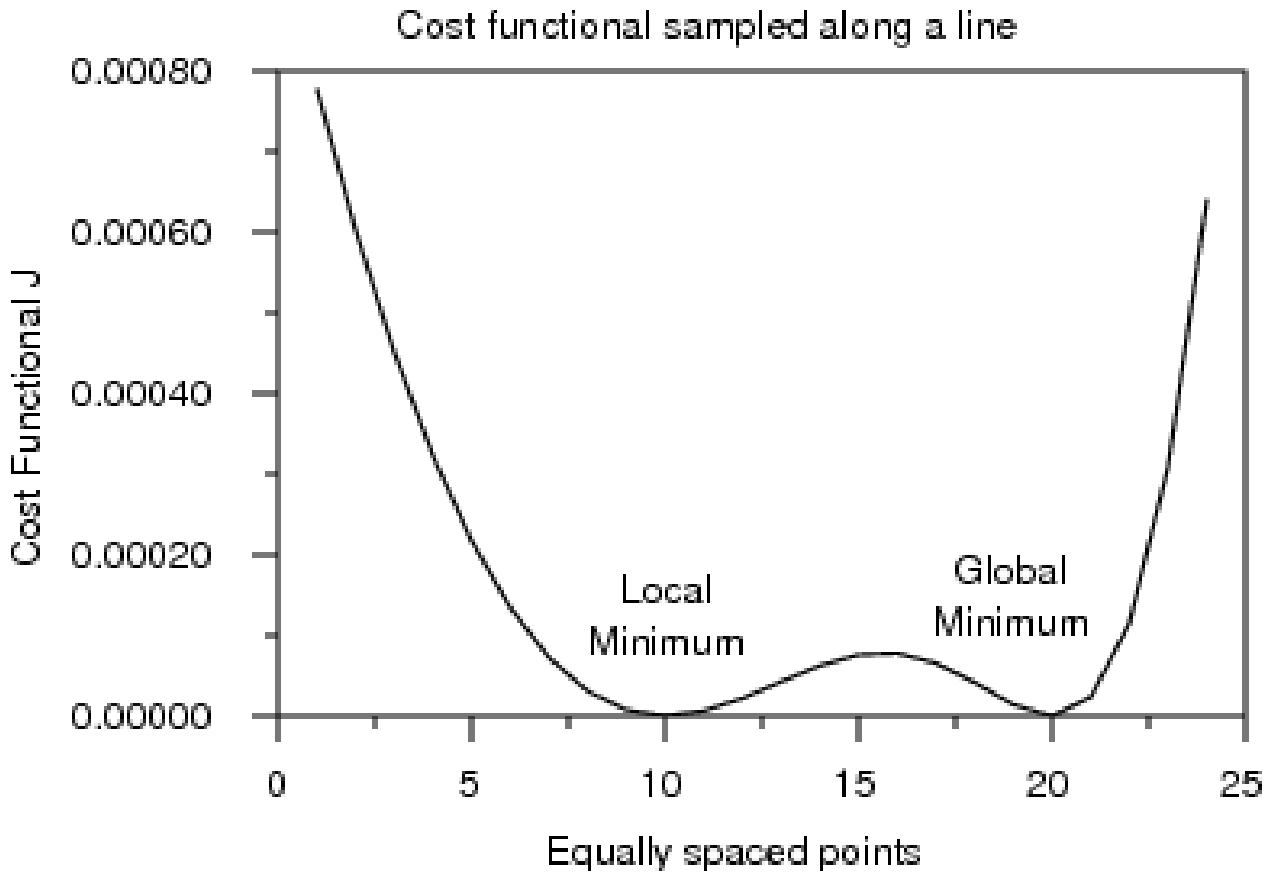


# Computational Model





# A "Simple" Function F(X)



# Focus on Effect of Uncertainty

The forebody simulator problem is an example of a deterministic problem in which input (inflow profile and bump shape) influences the solution of a PDE (Navier Stokes flow equations), resulting in a state variable (velocity field and pressure) from which we can extract a quantity of interest (integral of difference between computed and ideal flow along a profile line).

It can be convenient to "forget" that a PDE was involved, and to simplify the process to

$$x \rightarrow u(x) \rightarrow Q(u(x))$$

When we add uncertainty, our problem becomes something like

$$(x, \xi) \rightarrow u(x, \xi) \rightarrow Q(u(x, \xi)) \rightarrow E(Q(u(x, \xi)))$$

I will ignore the PDE, which explains how $(x, \xi)$ becomes $u(x, \xi)$, and concentrate on the uncertainty questions.

# Stochastic Integrals

- **What Does an Integral Tell Us?**
- The Probability Density Function
- Sampling from a Probability Density Function
- Approximating an Integral
- A Stochastic Fireball
- The Multidimensional Problem
- Approximating Multidimensional Integrals
- Sparse Grids
- Clenshaw Curtis Sparse Grids
- A Stochastic Tidal Wave

# INT: The Sum as an Average

Why are integrals so important in stochastic problems - or in mathematics, in general?

The integral is an averaging process. To see this, let's begin with the related idea of summation. If $f()$ is some quantity we have measured and indexed $n$ times, then we know the average value is:

$$\overline{f} = \frac{\sum_{i=1}^{n} f_i}{n}$$

The averaging process has produced a single value $\overline{f}$ that represents all the values we collected.

# INT: The Integral as an Average

In the integral case, we usually think that the integral of $f(x)$ from $a$ to $b$ represents the area under the curve.

Since the the region described has a base at the $x$ axis, a height of $f(x)$, and a horizontal extent of $(b-a)$, we could say that the integral is computing the area by getting an averaged height and multiplying it by the width.

In other words:

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

Thus, when we are dealing with a function over a finite range, the integral can be used to produce an average value of the quantity, and, if $f$ is continuous, we can also see that:

$$\int_a^b f(x)dx = \overline{f} \cdot (b-a) = f(\xi) \cdot (b-a) \text{ for some } \xi \in [a,b]$$

# INT: The Weighted Sum

If I roll two dice $n = 1000$ times, I can write down 1000 results $f_i$, and compute the average.

$$\overline{f} = \frac{\sum_{i=1}^{1000} f_i}{1000}$$

Or, I might realize that there are 11 possible results, and simply keep track of how many times each result occurs. Then, I have 11 kinds of results $f_j = j + 1$, $j = 1$ to 11, each with an associated value $v_j$ that indicates how many times it occurred, and I can average by:

$$\overline{f} = \frac{\sum_{j=1}^{11} v_j f_j}{1000} = \frac{\sum_{j=1}^{11} v_j f_j}{\sum_{j=1}^{11} v_j}$$

# INT: The Weighted Sum

Another way of looking at the first averaging process, (in which each outcome was equally likely) is to let $w_i = \frac{1}{1000}$ so that we can write:

$$\overline{f} = \sum_{i=1}^{1000} w_i f_i$$

Similarly, in the case where the outcomes had different likelihoods, I can normalize my weights by writing $w_j = \frac{v_j}{\sum_{j=2}^{12} v_j}$ so that we again have:

$$\overline{f} = \sum_{i=1}^{11} w_i f_i$$

# INT: Discrete Probability as a Weight

I rolled 2 dice 1,000 times, and counted how many times I got a 2, a 3, and so on up to a 12. Let's change notation a little bit, so that our index is $i$, it runs from 2 to 12, and the quantity $f_i$ indicates the frequency with which we got the value $i$. Thus, the value $f_5 = 51$ means we rolled a 5 51 times.

We assume that our results sample the behavior of the dice, and that, if we roll the dice again, the results will follow the same pattern. The easiest way to express this requires us to divide the observed frequency $f_i$ by the number of trials $n$ to get an estimate of the (discrete) probability of the event $i$:

$$p_i = \frac{f_i}{n}$$

When probabilities come from observations, many unlikely events won't be observed; our model assigns them probability 0, but hurricanes and earthquakes are...very, very "interesting".

# INT: Discrete Probability as a Weight

An estimated probablity is better than a frequency, because we can carry out $n = 100$ trials and $n = 1000$ trials and easily compare the two probability estimates. If we have observed "enough" trials, we can expect the probability estimates to be very similar.

More importantly, probabilities are weights, and are normalized to sum up to 1.

So we can use probabilities to get average outcomes.

If our probability is an estimate from observations, then the average means the average over the observed data.

But if our probability comes from a mathematical theory, then this idea allows us to **predict the average of results that we have not yet observed.**

$$\overline{f_i} = \sum_{i=1}^{n} p_i f_i$$

# INT: Estimated Probablity of Observed Data

I have rolled two dice 1,000 times. There are 36 possible outcomes, whose observed frequencies are:

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 37 | 18 | 16 | 24 | 24 | 38 |
| 2 | 19 | 29 | 23 | 28 | 38 | 28 |
| 3 | 21 | 30 | 28 | 32 | 22 | 30 |
| 4 | 22 | 29 | 33 | 31 | 34 | 31 |
| 5 | 29 | 19 | 41 | 29 | 27 | 22 |
| 6 | 30 | 27 | 37 | 22 | 23 | 29 |

Obviously, these numbers sum to 1000. To estimate probabilities, I divide frequencies by 1000. I estimate the probability of rolling (1,2) to be 0.018, and the probability of (2,1) is estimated to be 0.019.

# INT: Average of Quantity of Interest of Observed Data

A quantity of interest can be thought of as any function of our fundamental variables. In our work, these fundamental variables have some random variation.

If my fundamental variables are the values showing on my two dice, a simple quantity of interest $q$ is the sum. To compute $\overline{q}$, the average value of $q$, from my observations, I compute the sum of the following 36 terms:

$$\overline{q} = \sum_{i=1}^{6} \sum_{j=1}^{6} p(i,j) \cdot q(i,j)$$

where $p(i,j)$ is my estimated probabilities:

```
q_bar =
  0.037 * 2 + 0.018 * 3 + 0.016 * 4 + 0.024 *  5 + 0.024 *  6 + 0.038 *  7
+ 0.019 * 3 + 0.029 * 4 + 0.023 * 5 + 0.028 *  6 + 0.038 *  7 + 0.028 *  8
+ 0.021 * 4 + 0.030 * 5 + 0.028 * 6 + 0.032 *  7 + 0.022 *  8 + 0.030 *  9
+ 0.022 * 5 + 0.029 * 6 + 0.033 * 7 + 0.031 *  8 + 0.034 *  9 + 0.031 * 10
+ 0.029 * 6 + 0.019 * 7 + 0.041 * 8 + 0.029 *  9 + 0.027 * 10 + 0.022 * 11
+ 0.030 * 7 + 0.027 * 8 + 0.037 * 9 + 0.022 * 10 + 0.023 * 11 + 0.029 * 12
```

and the result of this is 7.071.

# INT: Average Quantity of Interest of Observed Data

```
Number of trials
  n = 1000;
Compute N random pairs of integers between 1 and 6
  d = randi ( [ 1, 6 ), n, 2 );
Count how many times each pair occurs.
  for i = 1 : 6
    for j = 1 : 6
      f(i,j) = length ( find ( d(:,1) == i & d(:,2) == j ) );
    end
  end
Convert frequency to estimated probablity.
  p(1:6,1:6) = f(1:6,1:6) / n;
Our quantity of interest is the sum.
  for i = 1 : 6
    for j = 1 : 6
      q(i,j) = i + j;
    end
  end
Determine the exact average quantity of interest.
  q_bar = 0.0;
  for i = 1 : 6
    for j = 1 : 6
      q_bar = q_bar + p(i,j) * q(i,j)
    end
  end

  7.071
```

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/sum_estimate.m

# INT: Average Quantity of Interest of Theoretical Data

I assume a pair of dice are constructed to be **uniform** or **fair** (1/6 chance of each result) and **independent** (so 1/6 * 1/6 chance of each pair of results.) Without rolling the dice, I can compute a table of probabilities. Every entry is $1/36 = 0.0278$.

Since we haven't actually rolled the dice, we aren't really computing the average value, but the expected value, which is a sort of average over probability space. The formula is very familiar:

$$E(q) = \sum_{i=1}^{6}\sum_{j=1}^{6} p(i,j)\cdot q(i,j) = \sum_{i=1}^{6}\sum_{j=1}^{6} \frac{1}{36}(i+j)$$

and the result is exactly 7.

Now I haven't actually rolled any dice. I am averaging events that have not occurred. I am making a prediction, or a probabilistic judgment, based on my model of the behavior of the dice.

# INT: The Expected Value

When we were talking about observed data, it was clear that we could compute $\overline{q}$, the average of the quantity of interest, by computing its value for each observation, and averaging.

Instead of estimating probabilities from observed frequencies, we can simply produce a set of probabilities in advance, as a **model** or **theory** of a process. Now, for the basic variables, or any quantity of interest, we compute an average weighted by the probabilities, and call the result $E(q)$, the expected value.

If our model is correct, actually rolling the dice many times produces a sequence of values of $q$ whose average tends to $E(q)$.

We can compute it because probabilities express our model, and integration produces the resulting average or representative value.

# INT: The Expected Value

**Exercise 1:** Suppose we have two fair dice. What is the expected value of the quantity of interest $Q$ which is the **product** of the values of the two dice?

**Exercise 2:** When we draw cards from a standard deck of 52, the second event is not independent of the first. But suppose the deck is very large, so that the probabilities stay the same after each draw. The game of Blackjack starts with the player receiving two cards from the deck. The suit of the card doesn't matter, so there are really only 13 distinct cards. Cards 2 through 10 have their face value, J, Q and K count as 10, and for simplicity we will assume the Ace counts as 11. What is the expected value of the sum of your two cards?

# Stochastic Integrals

- What Does an Integral Tell Us?
- **The Probability Density Function**
- Sampling from a Probability Density Function
- Approximating an Integral
- A Stochastic Fireball
- The Multidimensional Problem
- Approximating Multidimensional Integrals
- Sparse Grids
- Clenshaw Curtis Sparse Grids
- A Stochastic Tidal Wave

# PDF: Continuous Probability as a Weight

Now suppose that our probabilistic system does not select one of a set of discrete values. Instead, our outcome is a real number in some range $\Omega$, which is typically an interval, finite, semi-infinite, or infinite.

For the discrete case, we interpreted a probability as an expected relative frequency. If the ace of spades had a $1/52$ probability of being selected from a deck, that is, $p(A\spadesuit) = \frac{1}{52}$, then we expected to pick it about 10 times in 500 draws.

But, except for special cases, when we have a continuously varying set of outcomes, *every outcome $x$ must have zero probability,* that is, $p(x) = 0$!

We will explain this in a moment, but realize one thing: for continuous variables, an event with zero probability is not the same thing as an impossible event.

# PDF: Continuous Probability as a Weight

For continuous variables, instead of thinking of probability, we think of **probability density**. If our outcome variable is $x$, then this density is often written as $\rho(x)$, that is, "rho of $x$".

The probability density is a nonnegative weight with unit integral:

$$\int_{\Omega} \rho(x)dx = 1$$

Instead of asking about a single outcome $x$, we will want the probability of events such as $a < x$, or $x < b$, or $a < x < b$:

$$p(a < x < b) = \int_a^b \rho(x)dx$$

If the desired outcome is not an interval, we can write $D$ as the set of desired outcomes, and the probability of an outcome $e$ that belongs to the set $D$ is

$$p(e) = \int_a^b \chi_D \, \rho(x)dx = \int_D \rho(x)dx$$

# PDF: Density for a Finite Interval

Over the interval [0,1], the **uniform density** is the function $\rho(x) = 1$. The likelihood that the outcome will be in any subinterval of [0,1] is simply the length of that subinterval.

For a general finite interval $[a, b]$, the uniform density is

$$\rho(x) = \frac{1}{b-a}$$

Suppose we want a linear density for the interval $[0, 1]$? We might try the function $\rho(x) = x$, which is nonnegative (good), but whose integral is $\frac{1}{2}$. If we normalize the function by dividing by its integral, we get a linear density function for $[0, 1]$:

$$\rho(x) = 2 \cdot x$$

In general, we can propose any nonnegative function $f(x)$ as a density function, as long as we normalize it by its integral. This assumes, of course, that $f(x)$ is integrable, and has a finite integral!

# PDF: Density for a Double Infinite Interval

It's impossible to have a uniform density for an infinite interval. Any density function must go to zero, and fast, as $|x| \to \infty$.

If we want to guarantee that every polynomial function, multiplied by the weight, goes to zero, some kind of negative exponential weight is perfect.

For the domain $(-\infty, +\infty)$, the **Gaussian**, or standard **normal density** function, with mean $\mu = 0$ and variance $\sigma^2 = 1$ is:

$$\rho(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

A more general version of this density function allows you to specify any mean $\mu$ and variance $\sigma^2$:

$$\rho(\mu, \sigma; x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

# PDF: Density for a Semi-Infinite Interval

The exponential PDF applies to the semi-infinite interval $[0, +\infty)$, with PDF

$$\rho(x) = \lambda e^{-\lambda x}$$

where $\lambda$ is a parameter that must be greater than 0. In the standard case, we take $\lambda = 1$.

**Exercise 3:** What are the values of the mean $\mu$ and variance $\sigma^2$ for the exponential PDF?

**Exercise 4:** With $\lambda = 1$ in the exponential PDF, what is the expected value of $x^2$?

# PDF: The Expected Value for Continuous Variables

The density function $\rho(x)$ holds all probabilistic information about a random variable.

The zeroth moment is the integral of $\rho(x)$, and is always 1. This is the probability that something (anything) will happen.

To determine the average or expected value or first moment of the random variable, we use $\rho(x)$ to compute:

$$\mu = \overline{x} = E[x] = \int_{\Omega} x\,\rho(x)\,dx$$

In particular, for the uniform density on [0,1], we have $\mu = \frac{1}{2}$, and for the normal density on $(-\infty, +\infty)$, we have $\mu = 0$.

The variance or $\sigma^2$ or second centered moment is the expected value of the squared difference of $x$ and its expected value $\mu$:

$$\sigma^2 = E[(x-\mu)^2] = \int_{\Omega} (x-\mu)^2 \rho(x) dx$$

# PDF: Mean and Variance for Uniform Density

For a variable described by the uniform density over $[0, 1]$, we know $\mu = \frac{1}{2}$. For the variance, we compute:

$$\begin{aligned}
\sigma^2 &\equiv \int_\Omega (x-\mu)^2 \rho(x)\, dx \\
&= \int_0^1 (x - \frac{1}{2})^2 \cdot 1\, dx \\
&= \int_0^1 x^2 - x + \frac{1}{4}\, dx \\
&= (\frac{x^3}{3} - \frac{x^2}{2} + \frac{x}{4})\Big|_0^1 \\
&= \frac{1}{12} \approx 0.0833
\end{aligned}$$

**Exercise 5:** Show that, for any PDF, $\sigma^2 = E(x^2) - (E(x))^2$.

# PDF: Variance of Samples from Uniform Distribution

If we select $n$ sample variables from some random process, we can compute the sample mean and sample variance. If $n$ is large, we'd expect these to be close to the mean and variance of the PDF.

There are some minor variations in how the variance $\sigma^2$ might be calculated.

```
x = rand ( n, 1 );
mu = sum ( x ) / n;
mu2 = sum ( x.^2 ) / n;
v1 = sum ( ( x - mu  ).^2 ) / n;
v2 = mu2 - mu^2;                          <-- Actually the same as V1
v3 = sum ( ( x - mu  ).^2 ) / ( n - 1 );  <-- Statisticians prefer this.
v4 = sum ( ( x - 0.5 ).^2 ) / n;          <-- Variance of sample from PDF mean
```

| N | Mu | MU2 | V1 | V2 | V3 | V4 |
|---|---|---|---|---|---|---|
| 1 | 0.7409 | 0.5489 | 0.0000 | NaN | 0.0580 | 0.0000 |
| 10 | 0.6087 | 0.4491 | 0.0785 | 0.0872 | 0.0903 | 0.0785 |
| 100 | 0.5181 | 0.3494 | 0.0809 | 0.0817 | 0.0812 | 0.0809 |
| 1000 | 0.5055 | 0.3453 | 0.0897 | 0.0898 | 0.0897 | 0.0897 |
| 10000 | 0.4956 | 0.3293 | 0.0837 | 0.0837 | 0.0837 | 0.0837 |
| 100000 | 0.5004 | 0.3337 | 0.0833 | 0.0833 | 0.0833 | 0.0833 |
| 1000000 | 0.5004 | 0.3336 | 0.0832 | 0.0832 | 0.0832 | 0.0832 |

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/uniform_variance.m

# PDF: Mean and Variance for Normal Density

For a variable described by the normal density over $(-\infty, +\infty)$, it's easy to see that $\mu = 0$. The variance is:

$$\begin{aligned}
\sigma^2 &= \int_{\Omega} (x-\mu)^2 \rho(x) dx \\
&= \int_{-\infty}^{+\infty} x^2 \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx \\
&= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} x^2 e^{-\frac{x^2}{2}} dx \\
&= 1
\end{aligned}$$

**Exercise 6:** Show in two different ways how to determine the value of $E(x^2)$ for the normal PDF.

# PDF: Variance of Samples from Normal Distribution

```
x = randn ( n, 1 );  <-- How to get normal random values in MATLAB;

mu = sum ( x ) / n;
mu2 = sum ( x.^2 ) / n;
v1 = sum ( ( x - mu  ).^2 ) / n;
v2 = mu2 - mu^2;
v3 = sum ( ( x - mu  ).^2 ) / ( n - 1 );
v4 = sum ( ( x - 0.0 ).^2 ) / n;          <-- Same as MU2, because mean is 0.
```

| N | Mu | MU2 | V1 | V2 | V3 | V4 |
|---|---|---|---|---|---|---|
| 1 | 0.8632 | 0.7451 | 0.0000 | 0.0000 | NaN | 0.7451 |
| 10 | 0.1811 | 0.7267 | 0.6939 | 0.6939 | 0.7710 | 0.7267 |
| 100 | -0.0167 | 1.1131 | 1.1128 | 1.1128 | 1.1241 | 1.1131 |
| 1000 | -0.0041 | 1.0113 | 1.0112 | 1.0112 | 1.0123 | 1.0113 |
| 10000 | -0.0111 | 1.0113 | 1.0112 | 1.0112 | 1.0113 | 1.0113 |
| 100000 | -0.0038 | 0.9999 | 0.9998 | 0.9998 | 0.9999 | 0.9999 |
| 1000000 | -0.0004 | 1.0006 | 1.0006 | 1.0006 | 1.0006 | 1.0006 |

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/normal_variance.m

# PDF: The Expected Value of a Quantity of Interest

Suppose that $x$ is a random variable from the uniform density for $[0, 1]$. We know that the expected value of $x$ is $\frac{1}{2}$. What is the expected value of $x^2$? A natural guess might be $\frac{1}{4}$, the square of the expected value, but this is wrong.

Unless the function $f(x)$ is linear, it is not generally true that $E(f(x)) = f(E(x))$!

$$E(x^2) = \int_\Omega x^2 \rho(x) dx = \int_0^1 x^2 \cdot 1 dx$$
$$= (\frac{x^3}{3})\Big|_0^1 = \frac{1}{3} \approx 0.333... \neq 0.25 = (E(x))^2$$

**Exercise 7**: Explain, demonstrate, or prove why it is true that for a random variable $x$, $E(a \cdot x + b) = a \cdot E(x) + b$.

# Ahem!

# Approximating Integrals for Stochastic Problems

John Burkardt
Department of Scientific Computing
Florida State University

.....

ISC 5936-01:
Numerical Methods for Stochastic Differential Equations
http://people.sc.fsu.edu/~jburkardt/presentations/...
stochastic_integrals.pdf

.....

Revised: 20 March 2013

26/28 February, 5/7/19/21 March 2013

# Stochastic Integrals

- What Does an Integral Tell Us?
- The Probability Density Function
- **Sampling from a Probability Distribution Function**
- Approximating an Integral
- A Stochastic Fireball
- The Multidimensional Problem
- Approximating Multidimensional Integrals
- Sparse Grids
- Clenshaw Curtis Sparse Grids
- A Stochastic Tidal Wave

# SAM: Sampling Uniformly from [0,1]

To **sample** from a probability distribution function is a process which selects possible outcomes according to their probability.

For the uniform distribution on [0,1], we can loosely think that every value has an equal probablity of being chosen; however, single values really only have a probability density, not a probability. It's more correct to say that intervals of the same length have an equal probability that a value will be chosen from them.

To simulate a random process, we may want to compute enormous sequences of random values. On a computer, most computer languages include a procedure to sample uniformly from $[0, 1]$:

- **x = ( double ) rand() / ( double ) RAND_MAX;** in C and C++;
- **call random_number ( harvest = x )** in Fortran90;
- **x = nextDouble();** in Java;
- **x = Random[]** in Mathematica;
- **x = rand ( 1, 1 );** in Matlab;
- **x = random ( )** in Python;

These values are not random in the usual sense; perfectionists call them *pseudorandom*.

# SAM: The State of a Random Number Generator

Most random number generators can be thought of as simply a long list of numbers that have been shuffled for you, as though it were a deck of cards. When you request a random number, the generator returns the current value, and "turns over" the next card in the deck, so to speak. This has the small advantage that you're not going to get the same random value twice in a row; in fact, this guarantees that you will see every random number in the deck before you get to the end and the sequence has to start over.

The current position of the random number generator (how the deck has been shuffled, which card is on top, and so on) is called its state.

# SAM: The State of a Random Number Generator

Often, the numbers are shuffled in the same way, and the state is completely described by a number which indicates the current card, an integer, called the random seed.

The purpose of the random seed is to allow the random number generator to be controlled by you. If you ran a program yesterday and got a weird error, you want to run the very same program today, so you want to use the same sequence of random numbers, so you want to be sure the seed is the same.

If you are running a parallel program, then you want each process to use a different random number seed, so that each process generates a distinct sequence of random values.

Unfortunately, every computer language differs in how the random number state is defined, what its default setting is, and how you can access and modify it.

# SAM: The State of a Random Number Generator

```
seed = 123456789;
rng ( seed );
x = rand ( 5, 1 )

x = 0.532833
    0.534137
    0.509553
    0.713564
    0.256999

seed = 123456789;
rng ( seed );
x = rand ( 5, 1 )

x = 0.0729883
    0.216037
    0.464753
    0.62259
    0.618388

seed = 123456789;
rng ( seed );
x = rand ( 5, 1 )

x = 0.532833
    0.534137
    0.509553
    0.713564
    0.256999
```

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/seeds.m

# SAM: Uniform Density for Other Intervals

Suppose that we wish to sample a random variable $\xi$ that is uniformly distributed over the interval $[a, b]$, but our random number generator returns values $x$ in [0,1] (which is the usual situation on the computer).

We have to scale the variables to the width $(b - a)$, and shift them to start at $a$, using the formula

$$\xi = a + (b - a) \cdot x$$

To check this, just ask what happens when $x$ is 0, and when it is 1.

**Exercise 8:** If we shift the uniform density from [0,1] to [-1,+1], what is the new value of $\sigma^2$? What if we shift to $[a, b]$?

# SAM: Sampling the Normal Density

Samples from the normal distribution can take on any real value, although in practice the results will be fairly close to the mean value (how close depends on the variance.)

1): A rough approximation of normally distributed values can be found by summing 12 uniform variables and subtracting 6.0.

**Exercise 9:** Explain in 10 words or less why this method cannot be exactly equivalent to a true normal distribution!

2): A second way, which is mathematically correct, computes two normal variables $\eta_1$ and $\eta_2$ by first computing two uniform variables $x_1$ and $x_2$ and then performing the following transformation:

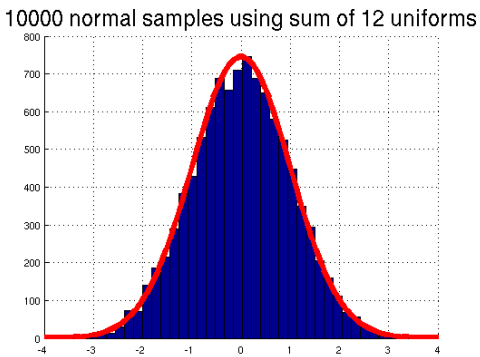
$$\eta_1 = \sqrt{-2\log(x_1)} \cdot \cos(2\pi x_2);$$
$$\eta_2 = \sqrt{-2\log(x_1)} \cdot \sin(2\pi x_2);$$

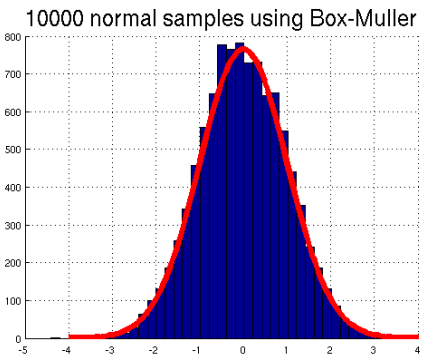
3): MATLAB supplies normal samples with the function **randn()**. Most other languages don't seem to care!

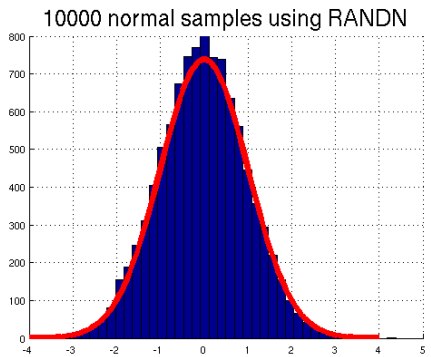
# SAM: Displaying Sampling Results

If you compute 1,000 samples from the normal density, can you get a feeling for whether you are doing the right thing? How do you display 1,000 samples? The easiest way is to compute a histogram, that is, a bar graph that displays the number of sample values that fall into each interval.

With proper scaling, you can compare your histogram to the PDF curve. (Divide your histogram data by frequency.)




http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/histogram.m

# SAM: The Cumulative Density Function

The PDF completely defines the behavior of a random variable.

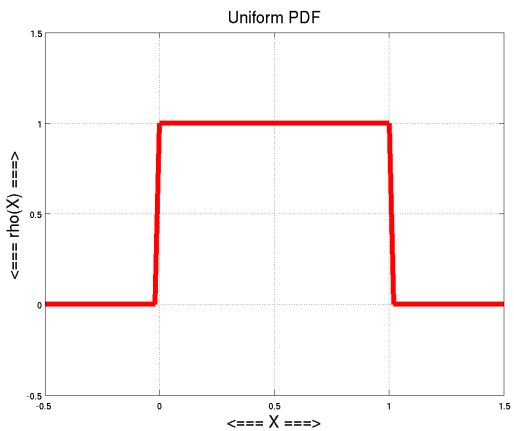
For any probabilistic situation, we may need to sample, that is, to generate hundreds of values that behave according to that PDF. If the PDF is not a "textbook" example, what do we do?

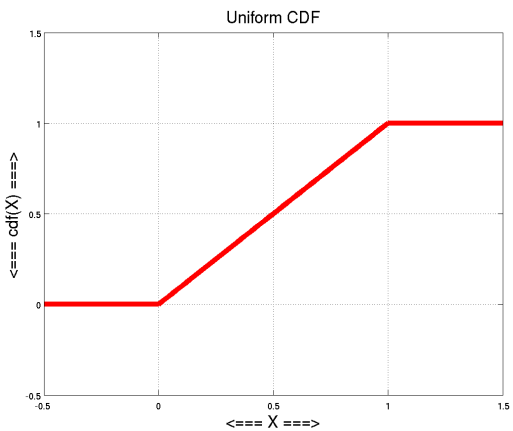
Once again, the idea of the integral comes to our rescue. We begin by defining the cumulative density function, or CDF, which is simply the integral of our PDF. For a variable defined on the interval $[a, b]$, we would write:

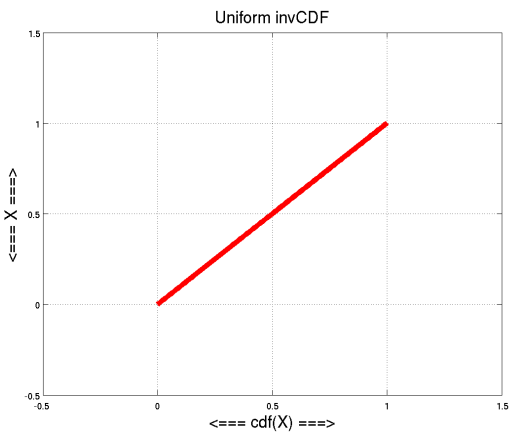
$$cdf(x) = \int_a^x \rho(s)\, ds$$

We have immediately that $cdf(a) = 0$, $cdf(b) = 1$, $\frac{d\, cdf(x)}{dx} = \rho(x)$, for $a \leq x \leq b$, $0 \leq cdf(x) \leq 1$ and $cdf(x)$ must be monotone increasing.

# SAM: Uniform PDF, CDF, invCDF

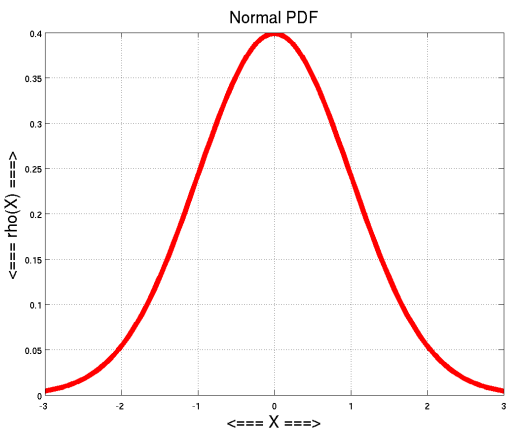


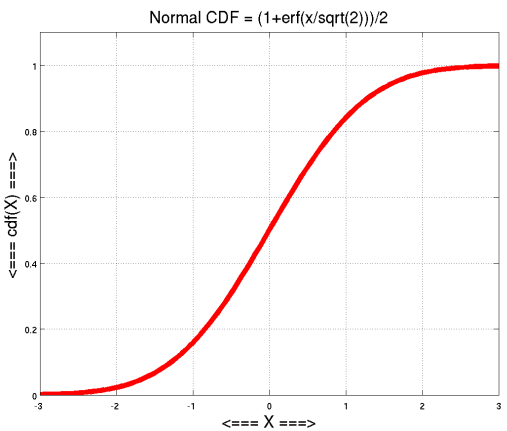



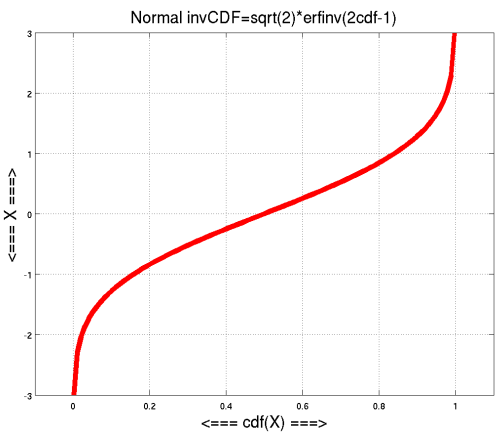




http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/uniform_plots.m

# SAM: Normal PDF, CDF, invCDF



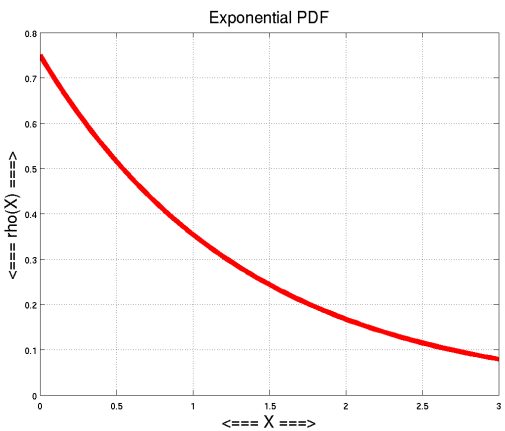



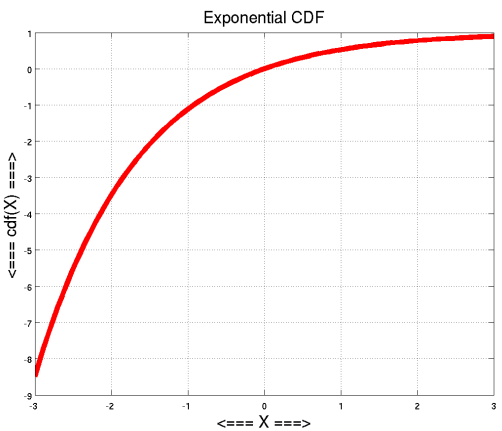



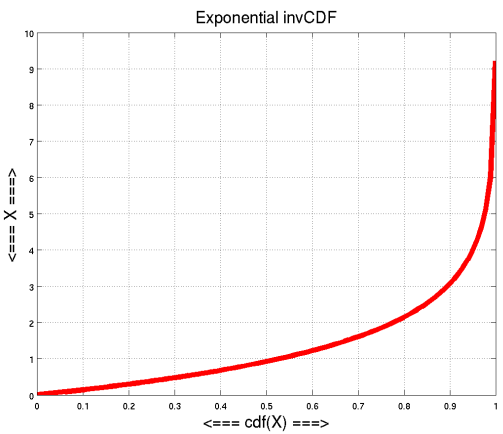

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/normal_plots.m

# SAM: Exponential PDF, CDF, invCDF

For this exponential PDF, we will assume $\lambda = 0.75$.




http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/exponential_plots.m

# SAM: Linear PDF, CDF, invCDF

**Exercise 10:** Create plots for the PDF, CDF, and inverse CDF of the linear PDF:

$$\rho(x) = 2 \cdot x$$

which is originally defined for $x \in [0, 1]$, but which you have shifted to the interval $x \in [5, 10]$.

# SAM: Kinky PDF, CDF, invCDF

**Exercise 11:** Create plots for the PDF, CDF, and inverse CDF of the "kinky" PDF:

$$\rho(x) = \begin{cases} 2/30 \text{ if } 0 \leq x \leq 3 \\ 12/30 \text{ if } 3 < x \leq 5. \end{cases}$$

which is defined for $x \in [0, 5]$.

# SAM: Sampling From an Arbitrary PDF

Now $cdf(x)$ is the probability that a random variable sampled from the given PDF will be less than or equal to $x$.

So to sample variables from an arbitrary PDF, we choose a uniform random number $u$ between 0 and 1, and seek a value $x$ for which $cdf(x) = u$. Our selected value is then $x$.

Two issues can arise here. If $\rho(x)$ is allowed to be zero within the interval, we could have that $cdf(x)$ is monotone, but not strictly monotone. Then $cdf(x)$ can have some flat spots, where more than one value $x$ could be chosen for a given $u$.

Second, we can't draw pictures of $cdf(x)$ to solve our problem. When we start with $u$ and seek $x$ such that $cdf(x) = u$, we are really trying to determine an inverse function $x = cdf^{-1}(u)$. In some cases, such an inverse can be computed as a formula. Otherwise, it must be approximated or computed.

# SAM: Sampling From the Linear PDF

Let's try an example of random sampling that we can't do directly with **rand()** or **randn()**. In fact, let's take the "linear pdf", shifted to $5 \leq x \leq 10$, mentioned in a previous exercise.

How do we figure out the formula for the PDF? It is a linear function over $[5, 10]$ which is 0 at $x = 5$. The function $\rho(x) = x - 5$ has the right value at $x = 5$, but equals 5 at $x = 10$. Without thinking, we assume we should divide by 5, to get:

$$\rho(x) = \frac{x-5}{5} \qquad \text{wrong!}$$

Wait a minute, that's not what we need. We don't need a value of 1 at $x = 10$, we need that the density integrates to 1 over the interval! We need an extra factor of $2/5$ for our PDF:

$$\rho(x) = \frac{2(x-5)}{25} \qquad \text{right!}$$

# SAM: Sampling From the Linear PDF

Our CDF is simply the indefinite integral of the PDF, so that's not bad:

$$cdf(x) = \int_5^x \frac{2(s-5)}{25}\, ds = \frac{(x-5)^2}{25}$$

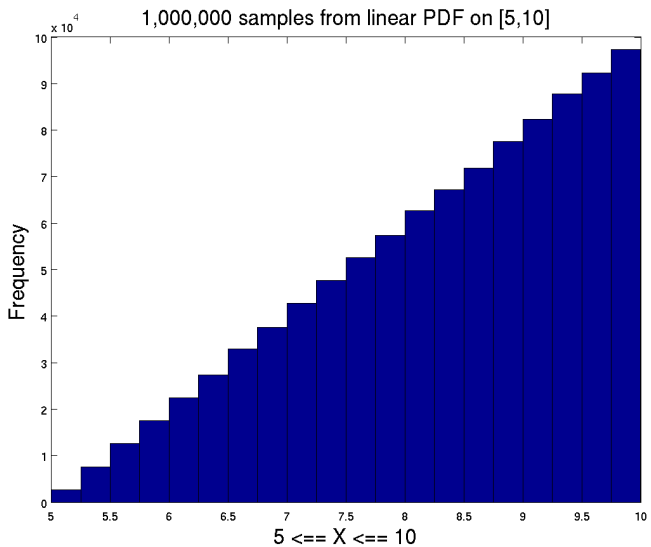
and, if we want to sample, we need to compute the inverse cdf:

$$\begin{aligned} cdf(x) &= u \\ x &= icdf(u) \\ cdf(x) = u &= \frac{(x-5)^2}{25} \\ 25u &= (x-5)^2 \\ icdf(u) = x &= 5 + \sqrt{25u} \end{aligned}$$

so, to sample from the linear PDF on $[5, 10]$, compute a uniform random $u$, and use the $icdf(u)$ formula to get $x$.

# SAM: Sampling From the Linear PDF

```
n = 1000000;
u = rand ( n, 1 );
x = 5 + sqrt ( 25 * u );
hist ( x, 20 );
```




http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/linear_sample.m

# SAM: Sampling From an Arbitrary PDF

**Exercise 12:** For the linear PDF on $[5, 10]$, show that $\mu = \frac{25}{3}$ and $\sigma^2 = \frac{25}{18}$.

**Exercise 13:** Consider the exponential PDF with $\lambda = 0.5$. Compute $n$ samples from this PDF. Compute the mean and variance of your sample data, and compare it to the mean and variance of the PDF.

# Stochastic Integrals

- What Does an Integral Tell Us?
- The Probability Density Function
- Sampling from a Probability Density Function
- **Approximating an Integral**
- A Stochastic Fireball
- The Multidimensional Problem
- Approximating Multidimensional Integrals
- Sparse Grids
- Clenshaw Curtis Sparse Grids
- A Stochastic Tidal Wave

# QUAD: The Monte Carlo Method

I said at the beginning that the integral can be regarded as an averaging process. We can turn this idea around, and assume that if we can approximate an average, we can approximate an integral.

The integral of $f(x)$ over $[a, b]$ is the average value of $f$ times the width of the interval, so the Monte Carlo approximation to the integral averages the value of $f()$ at $n$ points $x_i$ randomly chosen within $[a, b]$:

$$I(f, [a, b]) \approx MC(f, [a, b], n) = \frac{(b - a)}{n} \cdot \sum_{i=1}^{n} f(x_i)$$

The error in a Monte Carlo estimate tends to decreases like $\frac{1}{\sqrt{n}}$. This means that, to double your expected accuracy, you might need 4 times as many points. To get another decimal place of accuracy, you might need about 100 times as many points.

# QUAD: Battle of the Quadrature rules

Approximate $\int_0^\pi cos(4sin(x))dx$. We don't see the kind of rapid convergence we are used to!

| N | MC | Rui watches the error |
|---|---|---|
| 1 | 0.598 | |
| 2 | 1.892 | |
| 3 | 1.208 | |
| 4 | 1.312 | |
| 5 | 1.316 | Rui says, Look, the error actually went up a little! |
| 6 | 0.618 | |
| 7 | 0.195 | |
| 8 | 0.552 | Rui says, look, the error went up a lot! |
| 9 | 0.157 | |
| 10 | 1.061 | Rui says, look, the error went up even more! |
| 11 | 0.117 | |
| 12 | 0.517 | Rui says, oh my! |
| 13 | 0.591 | Bad! |
| 14 | 0.222 | |
| 15 | 0.034 | |
| 16 | 0.584 | Again! |
| 17 | 0.065 | |
| 18 | 0.526 | Rui says...I have nothing more to say! |
| 19 | 0.466 | |
| 20 | 0.532 | |
| 21 | 0.330 | |

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/mc_test.m

# QUAD: Battle of the Quadrature rules

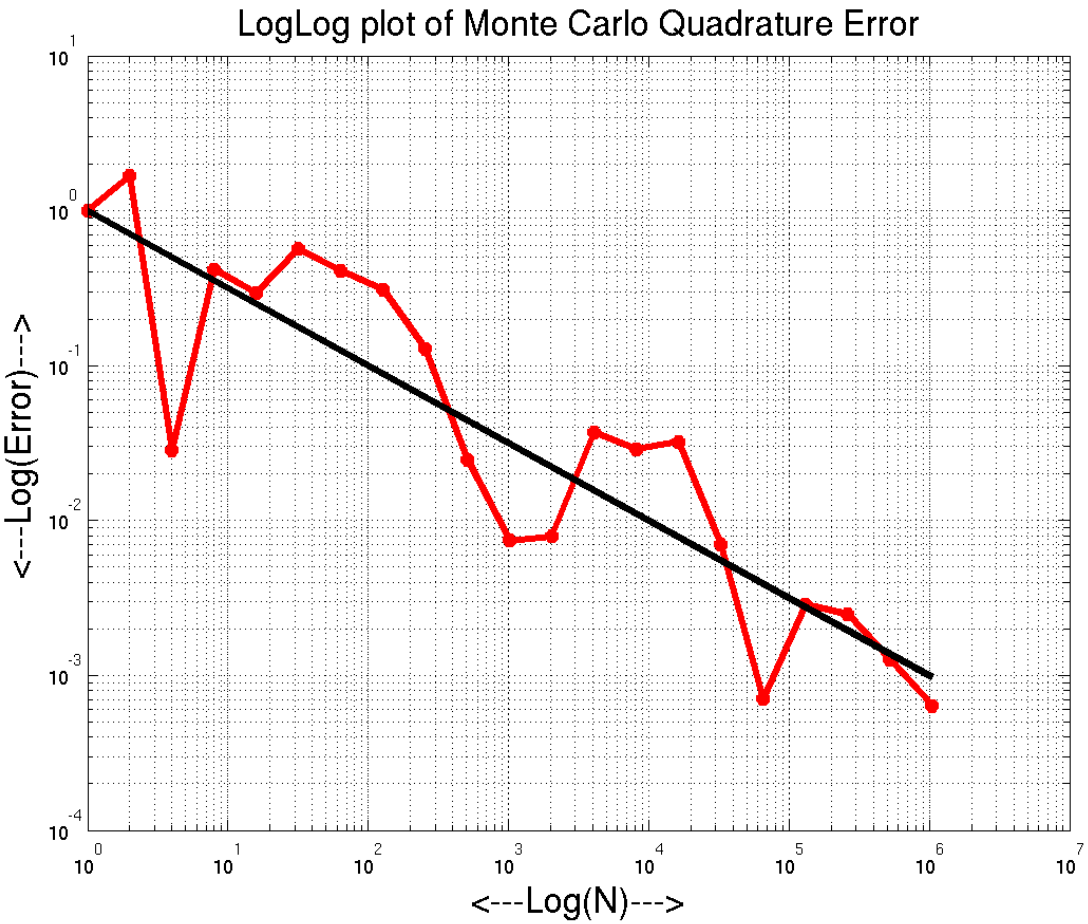
Approximate $\int_0^\pi cos(4sin(x))dx$, but this time, use a **lot** of steps!

| N | MC | Error halving? |
|---|---|---|
| 1 | 1.827 | Error about 2.0? |
| 2 | 1.57215 | |
| 4 | 0.859403 | Error about 1.0? |
| 8 | 0.461721 | |
| 16 | 0.0144315 | 0.5? |
| 32 | 0.0893733 | |
| 64 | 0.138425 | 0.25 |
| 128 | 0.299616 | |
| 256 | 0.0598655 | 0.125 |
| 512 | 0.0715631 | |
| 1024 | 0.0239286 | 0.0625 |
| 2048 | 0.0335445 | |
| 4096 | 0.0278803 | 0.03125 |
| 8192 | 0.0213013 | |
| 16384 | 0.00175709 | 0.015625 |
| 32768 | 0.00986718 | |
| 65536 | 0.00598231 | 0.0078125 |
| 131072 | 0.00148788 | |
| 262144 | 0.00625448 | 0.00390625 |
| 524288 | 0.00258764 | |
| 1048576 | 0.00159123 | 0.001853125 |

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/mc_test.m

# QUAD: LogLog Plot of N Versus MC Error

On a LogLog plot, we can see that the MC error (red line) does tend to decrease like the inverse square root of $n$ (the black line):




http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/mc_test.png

# QUAD: Alternatives to Monte Carlo?

There are actually many good things to say about the Monte Carlo method, and later on we will have to come back to it. But for now, it certainly seems to be an expensive and slow method.

One problem with it is that it doesn't notice an important feature about continuous functions, namely, if the value of $f$ is known at $x$, then nearby it probably has a similar value. And if we know its value at two points, a linear model is probably a good guess.

In cases where nearby locations have close values, it pays to include that in the model.

Let's see if we can think of another approach to integral approximation that will get us good accuracy with just 10 sample values, rather than 1,000,000!

# QUAD: The Midpoint Rule

The integral of $f(x)$ over $[a, b]$ can be defined as the limit of Riemann sums whose maximum subinterval size goes to zero.

That suggests that we can always try to approximate any integral by dividing the interval (or domain) into small pieces, and summing the product of the size of each piece times the function evaluated at a point in that piece.

Our first quadrature rule for $[a, b]$ is therefore, simply

$$\int_a^b f(x)dx \approx Q(f, [a, b]) = \sum_{i=1}^{n} f(x_i) \cdot (b_i - a_i)$$

where we have decomposed $[a, b]$ into $n$ subintervals $[a_i, b_i]$ and somehow chosen $x_i \in [a_i, b_i]$. Choosing $x_i = \frac{a_i + b_i}{2}$ gives us the composite midpoint rule, for instance.

# QUAD: The Composite Midpoint Rule

The basic midpoint rule is:

$$\int_a^b f(x)dx \approx Q(f,[a,b]) = f(\frac{a+b}{2}) \cdot (b-a)$$

and our (uniform spacing) composite midpoint rule is:

$$\int_a^b f(x)dx \approx Q(f,[a,b]) = \sum_{i=1}^{n} f(\frac{a_i+b_i}{2}) \cdot (b_i-a_i)$$

with $\Delta x = h = \frac{b-a}{n}$, and $a_i = a + (i-1)\Delta x$ and $b_i = a + i\Delta x$.

The basic midpoint rule only gives us one chance to approximate the integral; but the composite rule allows us to increase $n$, shrinking $h$, and presumably producing a sequence of better and better estimates.

# QUAD: Quadrature Rules

Notice that our uniform spacing composite midpoint rule

$$\int_a^b f(x)dx \approx Q(f,[a,b]) = \sum_{i=1}^{n} f(\frac{a_i + b_i}{2}) \cdot \Delta x$$

can be thought of as

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

where $w_i$ are the **weights** and $x_i$ are the **abscissas** or **nodes** or just plain old **points**. To estimate the integral, we only need to know how to evaluate $f()$ at a few points. A rule for approximating integrals this way is called a quadrature rule.

# QUAD: Midpoint Rule Approximation Error

Assume the function $f(x)$ is at least twice continuously differentiable and consider any interval $[x_1, x_2]$ with midpoint $x_m$. By Taylor's theorem, $f()$ at any point $x$ can be written as:

$$f(x) = f(x_m) + f'(x_m) \cdot (x - x_m) + \frac{1}{2} f''(x_m) \cdot (x - x_m)^2 + O(x - x_m)^3$$

Therefore,

$$\int_{x_1}^{x_2} f(x) dx$$

$$= \int_{x_1}^{x_2} f(x_m) + f'(x_m) \cdot (x - x_m) + \frac{1}{2} f''(x_m) \cdot (x - x_m)^2 + O(x - x_m)^3 dx$$

$$= f(x_m) \cdot (x_2 - x_1) + 0 + \frac{1}{24} f''(x_m) \cdot (x_2 - x_1)^3 + O(x_2 - x_1)^4$$

$$= f(x_m) \cdot (x_2 - x_1) + \frac{1}{24} f''(x_m) \cdot h^3 + O(h^4)$$

# QUAD: Midpoint Rule Approximation Error

So, for one subinterval, we know:

$$I(f,[a_i,b_i]) - Q(f,[a_i,b_i]) = \int_{a_i}^{b_i} f(x)dx - f(\frac{a_i+b_i}{2}) \cdot (b_i - a_i)$$
$$= \frac{1}{24} f''(x_m) \cdot h^3 + O(h^4)$$

and now we estimate

$$|I(f,[a,b]) - Q(f,[a,b])| \leq \sum_{i=1}^{n} |I(f,[a_i,b_i]) - Q(f,[a_i,b_i])|$$
$$= \sum_{i=1}^{n} |\frac{1}{24} f''(\frac{a_i+b_i}{2}) \cdot h^3 + O(h^4)|$$
$$\leq n \cdot (\frac{1}{24} ||f''||_\infty \cdot h^3 + O(h^4))$$
$$= (b-a) \cdot (\frac{1}{24} ||f''||_\infty \cdot h^2 + O(h^3))$$
$$= O(h^2).$$

# QUAD: Midpoint Rule Approximation Error

Notice that I've picked a problem for which the exact answer is about 22,000! This will make the error table seem a bit outrageous at first.

```
n = 1;
a = 0.0;
b = 10.0;
exact = exp ( b ) - exp ( a );

for nlog = 0 : 10
  q = 0.0;
  dx = ( b - a ) / n;
  for i = 1 : n
    a_i = a + ( i - 1 ) * dx;
    b_i = a +   i       * dx;
    x_i = ( a_i + b_i ) / 2.0;
    q = q + exp ( x_i) * dx;
  end
  e = abs ( exact - q );
  n = 2 * n;
end
```

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/midpoint_error.m

# QUAD: Midpoint Rule Approximation Error

Quadratic convergence: $h \longrightarrow h/2$ reduces error by 4.

| N | Q | E | Eold/E |
|---|---|---|---|
| 1 | 1484.13 | 20541.3 | |
| 2 | 9101.12 | 12924.3 | 1.58 |
| 4 | 17186.8 | 4838.68 | 2.67 |
| 8 | 20654.3 | 1371.19 | 3.52 |
| 16 | 21671.0 | 354.445 | 3.86 |
| 32 | 21936.1 | 89.3672 | 3.96 |
| 64 | 22003.1 | 22.3895 | 3.99 |
| 128 | 22019.9 | 5.60037 | 4.00 |
| 256 | 22024.1 | 1.40028 | 4.00 |
| 512 | 22025.1 | 0.350081 | 4.00 |
| 1024 | 22025.4 | 0.0875211 | 4.00 |

# QUAD: Interpolatory Quadrature

Over the interval $[a, b]$, the following constant function $f_0(x)$ interpolates $f(x)$ at the midpoint:

$$f_0(x) = f(\frac{a+b}{2})$$

The integral of $f_0(x)$ is:

$$\begin{aligned}\int_a^b f_0(x)dx &= \int_a^b f(\frac{a+b}{2})dx \\ &= f(\frac{a+b}{2}) \cdot (b-a)\end{aligned}$$

So the midpoint method can be explained as the result of finding and integrating a (constant) polynomial interpolant to $f(x)$.

This suggests that we might be able to use higher degree interpolants to get new rules.

# QUAD: Newton Cotes

One set of interpolatory rules is known as the Newton Cotes closed **NCC** rules. The midpoint rule is the first entry. The $n$-th entry uses $n$ equally spaced points in $[a, b]$, including the endpoints, constructs the interpolant to $f(x)$, and integrates it.

To compute the weights of the $n$ point rule, we simply work out the formula for the $i$-th Lagrange interpolation basis polynomial (1 at node $i$ and 0 elsewhere) and integrate it. By linearity, we have figured out how to integrate the interpolating polynomial for any given function.

Assuming we work in the standard interval $[-1, +1]$, then for any rule size $n$, we can work out the weights in advance.

Therefore, a user can simply request the points and weights for the standard interval, adjust them to any interval $[a, b]$, and compute the desired integral estimate.

# QUAD: Battle of the Quadrature rules

Approximate $\int_0^\pi cos(4sin(x))dx$:

| N | MC | NCC |
|---|---|---|
| 1 | 0.598 | 0.805 |
| 2 | 1.892 | 4.389 |
| 3 | 1.208 | 0.925 |
| 4 | 1.312 | 0.201 |
| 5 | 1.316 | 0.662 |
| 6 | 0.618 | 0.348 |
| 7 | 0.195 | 0.025 |
| 8 | 0.552 | 0.372 |
| 9 | 0.157 | 0.065 |
| 10 | 1.061 | 0.036 |
| 11 | 0.117 | 0.014 |
| 12 | 0.517 | 0.010 |
| 13 | 0.591 | 0.003 |
| 14 | 0.222 | 0.001 |
| 15 | 0.034 | 0.002 |
| 16 | 0.584 | 0.001 |
| 17 | 0.065 | 2e-4 |
| 18 | 0.526 | 1e-4 |
| 19 | 0.466 | 9e-5 |
| 20 | 0.532 | 5e-5 |
| 21 | 0.330 | 4e-5 |

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/nc_test.m

# QUAD: Adjusting a Quadrature Rule

So many quadrature rules are tabulated in a standard way that it is important to know how to adjust them to your purpose.

Suppose a quadrature rule is defined for the interval $[a, b]$, but you wish to use it over the interval $[c, d]$.

You must shift and scale the abscissas:

$$x_i \longrightarrow c + \frac{d-c}{b-a} \cdot (x_i - a)$$

and scale the weights:

$$w_i \longrightarrow w_i \cdot \frac{d-c}{b-a}$$

# QUAD: Adjusting a Quadrature Rule

```
function [ x, w ] = rule_adjust ( a, b, c, d, n, x, w )

%% RULE_ADJUST maps a quadrature rule from [A,B] to [C,D].
%
%    Input, real A, B, the endpoints of the old integration interval.
%
%    Input, real C, D, the endpoints of the new integration interval.
%
%    Input, integer N, the number of abscissas and weights.
%
%    Input, real X(N), the abscissas.
%
%    Input, real W(N), the weights.
%
%    Output, real X(N), the adjusted abscissas.
%
%    Output, real W(N), the adjusted weights.
%
  x(1:n) = ( ( b - x(1:n)     ) * c   ...
           + (     x(1:n) - a ) * d ) ...
           / ( b          - a );

  w(1:n) = ( ( d - c ) / ( b - a ) ) * w(1:n);

  return
end
```

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/rule_adjust.m

# QUAD: Newton Cotes

The 1 point **NCC** rule can integrate constants and linears precisely. The $n$ point **NCC** rule can precisely integrate polynomials of degree $n$ or less.

For example, the third entry in the **NCC** family is a 3 point rule, whose error is $O((b-a)^5)$ or $O(h^4)$ as a composite rule. It is tabulated for the interval [-1,+1] as follows:

| i | $x_i$ | $w_i$ |
|---|---|---|
| 1 | -1.0 | $1/3$ |
| 2 | 0.0 | $4/3$ |
| 3 | 1.0 | $1/3$ |

This rule is precise for constants, linears, quadratics and cubics.

**Exercise 14:** Using the shift and scale formulas above, what are the points and weights for the 3 point **NCC** rule, applied to the interval $[-\pi/2, +\pi/2]$?

# QUAD: Composite Newton Cotes

Suppose we wish to approximate an integral over $[0, 10]$ using the 3 point NCC rule. We must stretch our rule from $[-1, +1]$ to $[0, 10]$, and using only 3 points won't be very accurate.

We can instead make a composite rule, breaking up $[0, 10]$ into subintervals. If we repeatedly halve the interval size, here is the pattern.

```
    0                   5                  10
   +-----------------------------------------
1  | O                   *                  O
2  | O         *         @         *        O
4  | O    *    @    *    @    *    @    *   O
8  | O * @ * @ * @ * @ * @ * @ * @ * @ * @ * O
```

The *'s are midpoints of intervals, the O's and @'s are endpoints. The @'s are common to two subintervals. We may waste a little work evaluating the function twice there.

# QUAD: Simpson's Rule Approximation Error

```
n = 1;  a = 0.0;  b = 10.0;
exact = exp ( b ) - exp ( a );

for nlog = 0 : 10
  q = 0.0;
  dx = ( b - a ) / n;
  for i = 1 : n

    w = ( 1.0 / 6.0 ) * ( b - a ) / n;
    x = a + ( i - 1 ) * dx;
    q = q + w * exp ( x );

    w = ( 4.0 / 6.0 ) * ( b - a ) / n;
    x = x + 0.5 * dx;
    q = q + w * exp ( x );

    w = ( 1.0 / 6.0 ) * ( b - a ) / n;
    x = x + 0.5 * dx;
    q = q + w * exp ( x );

  end
  e = abs ( exact - q );
  n = 2 * n;
end
```

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/simpson_error.m

# QUAD: Simpson's Rule Approximation Error

Quartic convergence: $h \longrightarrow h/2$ reduces error by 16.

| N | Q | E | Eold/E |
|---|---|---|---|
| 1 | 37701.9 | 15676.4 | |
| 2 | 24671.0 | 2645.53 | 5.92 |
| 4 | 22276.5 | 251.031 | 10.53 |
| 8 | 22043.3 | 17.8372 | 14.07 |
| 16 | 22026.6 | 1.15352 | 15.46 |
| 32 | 22025.5 | 0.072723 | 15.86 |
| 64 | 22025.5 | 0.00455509 | 15.97 |
| 128 | 22025.5 | 0.000284848 | 15.99 |
| 256 | 22025.5 | 1.78055e-05 | 16.00 |
| 512 | 22025.5 | 1.11288e-06 | 16.00 |
| 1024 | 22025.5 | 6.95582e-08 | 16.00 |

# QUAD: Clenshaw Curtis Rules

Clenshaw Curtis **CC** rules are an interpolatory family, typically defined on the interval $[-1, +1]$, that do not use equally spaced points. These rules tend to place more points near the endpoints.

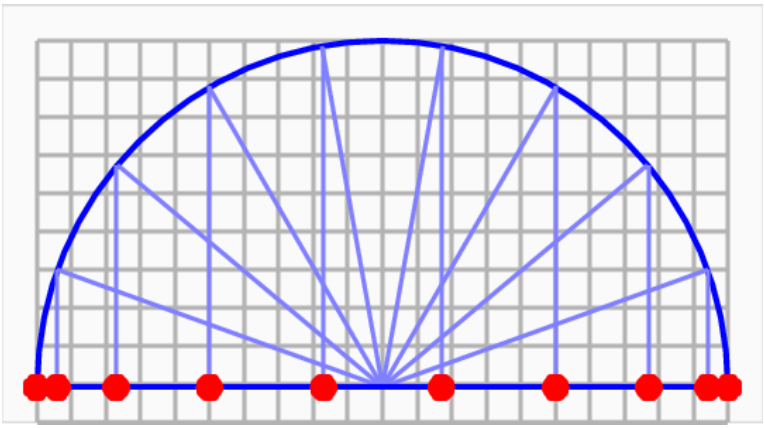
The **CC** rules of order 10, 20 or 30 can be safely used, but **NCC** rules beyond order 10 become very unreliable because of numerical instability.

For any order $n$, there is a simple formula for computing the weights and abscissas.

The 1 point **CC** rule can integrate constants and linears precisely. The $n$ point **CC** rule can precisely integrate polynomials of degree $n$ or less.

# QUAD: Clenshaw Curtis Rules

**Formula for $n$ Clenshaw Curtis points:** divide the semicircle by $n$ equally spaced rays; the cosines of their angles are the points.



The **CC** rule is just as (mathematically) precise as **NCC** rules of the same order, but the rule is (computationally) stable, while the **NCC** rules are not.

# QUAD: Clenshaw Curtis Rules

```
function [ x, w ] = clenshaw_curtis_compute ( n )

  if ( n == 1 )
    x(1) = 0.0;
    w(1) = 2.0;
    return
  end

  theta(1:n) = ( n-1 : -1 : 0 ) * pi / ( n - 1 );
  x(1:n) = cos ( theta(1:n) );

  for i = 1 : n
    w(i) = 1;
    for j = 1 : floor ( ( n - 1 ) / 2 )
      if ( 2 * j == ( n - 1 ) )
        b = 1;
      else
        b = 2;
      end
      w(i) = w(i) - b * cos ( 2 * j * theta(i) ) / ( 4 * j * j - 1 );
    end
  end

  w(1)     =     w(1)     / ( n - 1 );
  w(2:n-1) = 2 * w(2:n-1) / ( n - 1 );
  w(n)     =     w(n)     / ( n - 1 );
```

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/clenshaw_curtis_compute.m

# QUAD: Battle of the Quadrature rules

Approximate $\int_0^\pi cos(4sin(x))dx$:

| N | MC | NCC | CC |
|---|---|---|---|
| 1 | 0.598 | 0.805 | 0.805 |
| 2 | 1.892 | 4.389 | 4.389 |
| 3 | 1.208 | 0.925 | 0.925 |
| 4 | 1.312 | 0.201 | 1.059 |
| 5 | 1.316 | 0.662 | 0.294 |
| 6 | 0.618 | 0.348 | 0.066 |
| 7 | 0.195 | 0.025 | 0.028 |
| 8 | 0.552 | 0.372 | 5e-4 |
| 9 | 0.157 | 0.065 | 1e-4 |
| 10 | 1.061 | 0.036 | 2e-4 |
| 11 | 0.117 | 0.014 | 8e-5 |
| 12 | 0.517 | 0.010 | 3e-5 |
| 13 | 0.591 | 0.003 | 2e-6 |
| 14 | 0.222 | 0.001 | 1e-6 |
| 15 | 0.034 | 0.002 | 7e-7 |
| 16 | 0.584 | 0.001 | 2e-7 |
| 17 | 0.065 | 2e-4 | 9e-8 |
| 18 | 0.526 | 1e-4 | 3e-8 |
| 19 | 0.466 | 9e-5 | 6e-10 |
| 20 | 0.532 | 5e-5 | 7e-10 |
| 21 | 0.330 | 4e-5 | 8e-10 |

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/cc_test.m

# Ahem!

# Approximating Integrals for Stochastic Problems

John Burkardt
Department of Scientific Computing
Florida State University

.....

ISC 5936-01:
Numerical Methods for Stochastic Differential Equations
http://people.sc.fsu.edu/~jburkardt/presentations/...
stochastic_integrals.pdf

.....

Revised: 20 March 2013

26/28 February, 5/7/19/21 March 2013

# QUAD: Gauss Rules

For the **NCC** and **CC** rules, we picked the nodes in advance, and the interpolation requirement determined the weight values. Gauss rules make both abscissas and weights unknown, which allows them to be about twice as precise.

Gauss rules can be defined for any region and suitable pdf or weight, but the classic example uses $[-1, +1]$ with a uniform weight. In that case, the rules are sometimes called Gauss-Legendre or **GL** rules.

The one point **GL** rule is the midpoint rule, and it is precise for constants and linears. The two point **GL** rule is precise for constants, linears, quadratics and cubics. The $n$ point **GL** rule can integrate exactly any polynomial of degree $2n - 1$ or less.

# QUAD: Gauss Rules

When a function is well approximated by its Taylor series, we can see that, by increasing the order of the **GL** rule, we are capturing the initial part of the Taylor series for the function twice as fast as in the **NCC** and **CC** cases.

The abscissas and weights of a Gauss rule must either be tabulated, or computed. They are typically defined on $[-1, +1]$, so they generally must be modified to handle the user's interval $[a, b]$.

A typical usage might be:

```
[ x, w ] = legendre_ek_compute ( n );
x = pi * ( x + 1.0 ) / 2.0;          <-- Shift x from [-1,+1] to [0,pi];
w = pi * ( w / 2.0 );                <-- Shift w from [-1,+1] to [0,pi];
fx = cos ( 4.0 * sin ( x ) );        <-- Evaluate F(X)
q = w' * fx;                         <-- Q is the vector dot product of W and FX.
```

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/legendre_ek_compute.m

# QUAD: Battle of the Quadrature rules

Approximate $\int_0^\pi cos(4sin(x))dx$:

| N | MC | NCC | CC | GL |
|---|---|---|---|---|
| 1 | 0.598 | 0.805 | 0.805 | 0.805 |
| 2 | 1.892 | 4.389 | 4.389 | 1.201 |
| 3 | 1.208 | 0.925 | 0.925 | 0.654 |
| 4 | 1.312 | 0.201 | 1.059 | 3e-4 |
| 5 | 1.316 | 0.662 | 0.294 | 0.044 |
| 6 | 0.618 | 0.348 | 0.066 | 0.007 |
| 7 | 0.195 | 0.025 | 0.028 | 5e-4 |
| 8 | 0.552 | 0.372 | 5e-4 | 3e-4 |
| 9 | 0.157 | 0.065 | 1e-4 | 3e-5 |
| 10 | 1.061 | 0.036 | 2e-4 | 2e-6 |
| 11 | 0.117 | 0.014 | 8e-5 | 9e-7 |
| 12 | 0.517 | 0.010 | 3e-5 | 1e-7 |
| 13 | 0.591 | 0.003 | 2e-6 | 2e-9 |
| 14 | 0.222 | 0.001 | 1e-6 | 1e-9 |
| 15 | 0.034 | 0.002 | 7e-7 | 2e-10 |
| 16 | 0.584 | 0.001 | 2e-7 | 4e-12 |
| 17 | 0.065 | 2e-4 | 9e-8 | 1e-12 |
| 18 | 0.526 | 1e-4 | 3e-8 | 2e-13 |
| 19 | 0.466 | 9e-5 | 6e-10 | 1e-14 |
| 20 | 0.532 | 5e-5 | 7e-10 | 1e-15 |
| 21 | 0.330 | 4e-5 | 8e-10 | 6e-16 |

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/gl_test.m

# QUAD: Nesting

A nested family of quadrature rules is a sequence of rules of increasing order, with the property that each rule includes all the points of the previous one.

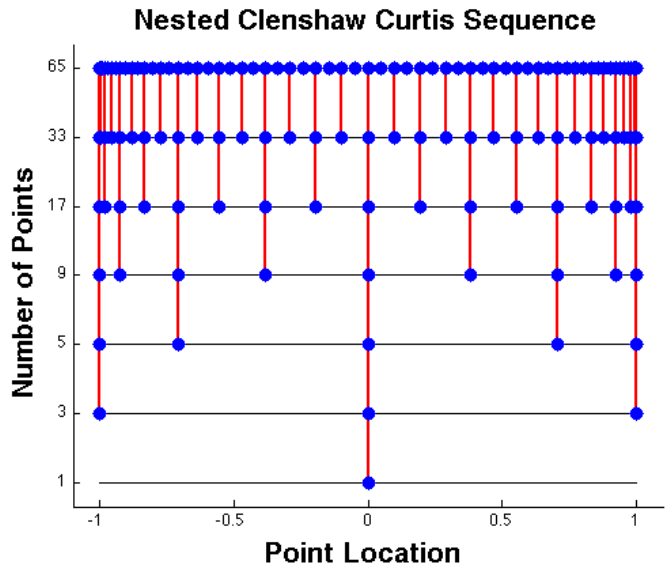
For the NCC family, the points are equally spaced. That means that we can easily create nested families. A simple example is the rules of order $1, 3, 5, 9, 17, ..., 2^n + 1$.

We like nested rules if evaluating our function is expensive, and we want an error estimate for our approximations, and if the error estimate isn't small enough, we plan to try harder.

In such a case, we might compute Q(1) and Q(3), the integral estimates using 1 and 3 points, with error estimate E(1) = Q(3) - Q(1). If E(1) is not small, we compute Q(5), with error estimate E(2) = Q(5) - Q(3), and so on. Because the rules are nested, we greatly reduce the number of function values needed.

# QUAD: Nesting

We can also produce a nested family using the CC rules. Again, the orders will be $1, 3, 5, 9, 17, ..., 2^n + 1$. Since the CC rules are stable, but the NCC rules are not, this is actually a much better nested family to use.




It is not possible to create a nested sequence of Gauss-Legendre rules, because the points that occur in a given rule will never show up in a later rule, with the single exception of the point $x = 0.0$.

# QUAD: Precision

It may be surprising that an integral can be approximated by a quadrature rule. It is even more surprising if, sometimes, the approximation turns out to be exact. For interpolatory rules, however, this should not be so surprising, since they are designed to be exact for low degree polynomials.

A quadrature rule **Q(f,[a,b])** is said to have precision $p$ or to be precise through degree $p$ if it is the case that

$$Q(f,[a,b]) = I(f,[a,b]) \equiv \int_a^b f(x)\,dx$$

whenever $f(x)$ is a polynomial of degree $p$ or less.

# QUAD: Precision Results

We have already claimed the following precision results:

| Rule | p | Range |
|---|---|---|
| midpoint rule | 1 | $(x^0, x^1)$ |
| odd n-point NCC rule | n | $(x^0 : x^n)$ |
| even n-point NCC rule | n-1 | $(x^0 : x^{n-1})$ |
| odd n-point CC rule | n | $(x^0 : x^n)$ |
| even n-point CC rule | n-1 | $(x^0 : x^{n-1})$ |
| n-point GL rule | 2*n-1 | $(x^0 : x^{2n-1})$ |

Now, what is the precision of the MC rule? No matter how many points $n$ we use, the MC rule is precise for ... a constant function, but nothing else.

Since we know that an MC approximation is eventually good enough, this reminds us that we don't need precision to get a good result. So why is it useful?

# QUAD: Precision Controls Asymptotic Accuracy

To see why precision is useful, let's suppose we have a sequence of quadrature rules $Q_i$ with the property that $Q_p$ has precision $p$.

Suppose, further, we are approximating the integral of a function $f(x)$ whose derivatives are bounded by some fixed constant $C$, or, more precisely, that $\frac{1}{k!}\frac{d^k f}{dx^k}$ is bounded by $C$.

Then the error between $Q_p(f, [a, b])$ and $I(f, [a, b])$ is bounded by $C \cdot (b - a)^p$, meaning that asymptotically, the error decreases with the precision first linearly, then quadratically, then cubically, and so on. Such behavior is sometimes called subexponential because it's almost an exponential decay.

A family of rules of increasing precision is a great tool. In 1D, the NCC, CC and GL rules give us such families. In higher dimensions, we will also be able to create precise rules, but we will see that the cost (number of points or function evaluations) required to each a given precision becomes an issue.

# QUAD: Precision Controls Asymptotic Accuracy

Let us approximate three integrands, with the idea of precision in mind, using the CC rule. Can you explain what we see?

| N | 1+2x+3x^2+4x^3 | exp(x) | x^7 |
|---|---|---|---|
| 1 | 0.75 | 0.0695606 | 0.9375 |
| 2 | 1.5 | 0.140859 | 3 |
| 3 | 0 | 0.000579323 | 0.375 |
| 4 | 8.88178e-16 | 0.000143757 | 0.0807292 |
| 5 | 0 | 3.42536e-07 | 0.00416667 |
| 6 | 0 | 4.33745e-08 | 0.000520833 |
| 7 | 0 | 6.41081e-11 | 4.44089e-16 |
| 8 | 0 | 1.60063e-11 | 1.11022e-16 |
| 9 | 0 | 1.64313e-14 | 2.22045e-16 |
| 10 | 0 | 5.32907e-15 | 0 |

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/cc_precision.m,

# QUAD: Integrals over $[0, +\infty)$

We'd like to be able to have similar tools for problems involving a semi-infinite interval, and the negative exponential weight:

$$Q(f, [0, +\infty)) \approx I(f, [0, +\infty)) = \int_0^\infty f(x) e^{-x} \, dx$$

The **Gauss-Laguerre** quadrature rules handle this problem.

Rule $Q_n()$ consists of $n$ points and weights, and the integral is approximated by a sum, in the usual way:

$$Q_n(f, [0, +\infty)) = \sum_{i=1}^{n} w_i \, f(x_i) \approx \int_0^\infty f(x) e^{-x} \, dx$$

There is no need to evaluate the negative exponential weight. Its effect is included in $w_i$. Like the Gauss-Legendre rules, the Gauss-Laguerre rule of order $n$ is precise for any function $f(x)$ which is a polynomial of degree $2n - 1$ or less.

# QUAD: Integrals over $(-\infty, +\infty)$

For the normal or Gaussian PDF, our quadrature problem is:

$$Q(f, (-\infty, +\infty)) \approx I(f, (-\infty, +\infty)) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-\frac{x^2}{2}} \, dx$$

The **Gauss-Hermite** quadrature rules handle this problem. The same remarks apply as for the Gauss-Laguerre rules.

Note that the points and weights for Gauss rules are not easy to compute. To use them, you either get them from a table, or find a program that can compute them for you. Typically, a matrix eigenvalue problem has to be solved.

# QUAD: Summary

You can estimate the integral of almost any function using an **MC** approach. It doesn't matter whether the function is oscillatory, smooth, or discontinuous. But convergence will generally be very slow. Roughly speaking, each decimal place of accuracy requires 100 times as much work as the last one.

If your function is very smooth over the whole interval (no jumps, no kinks, no singularities, all derivatives exist) then a (global) interpolatory rule like **CC** or **GL** will do very well.

If your function may have jumps or kinks, you may need to use a composite rule, (midpoint, Simpson, ...) which is essentially a local interpolatory rule. In some cases, you can estimate the location of a singularity or kink, use a very small interval just near that problem area, and get accurate and efficient results.

# QUAD: Exercises

**Exercise 15:** Let $f(x) = 7 * x^6$, so that the integral of $f()$ over $[0, 1]$ is exactly 1. How many Monte Carlo points must you use in order to estimate this integral to an error of less than 0.001?

**Exercise 16:** What is the smallest number of points you would need in order to get the previous integral *exactly* for the Clenshaw-Curtis, Newton-Cotes, or Gauss-Legendre rule?

**Exercise 17:** Consider the function $f(x) = \frac{1}{1+x^2}$. The integral of this function over $[-5, +5]$ is $2 \arctan 5$. Estimate this integral using a Gauss-Legendre rule. Can you be confident in your answer?

**Exercise 18:** Repeat the previous exercise, but use a Newton-Cotes rule. What is your confidence in your answer?

# Stochastic Integrals

- What Does an Integral Tell Us?
- The Probability Density Function
- Sampling from a Probability Density Function
- Approximating an Integral
- **A Stochastic Fireball**
- The Multidimensional Problem
- Approximating Multidimensional Integrals
- Sparse Grids
- Clenshaw Curtis Sparse Grids
- A Stochastic Tidal Wave

# BANG: A Model of Flame Propagation

When a match ignites, a ball of flame appears, and grows rapidly until reaching a critical size. At this size, there is a balance between the amount of oxygen available at the surface of the ball, and the amount of combustible material within the ball.

If $r$ denotes the radius of the sphere, then the surface area is like $r^2$, the volume like $r^3$, and if we ignore all the scaling units, then we may assume that the fireball reaches its equilibrium size when $r^2$ equals $r^3$.

This can be expressed as the ordinary differential equation (ODE):

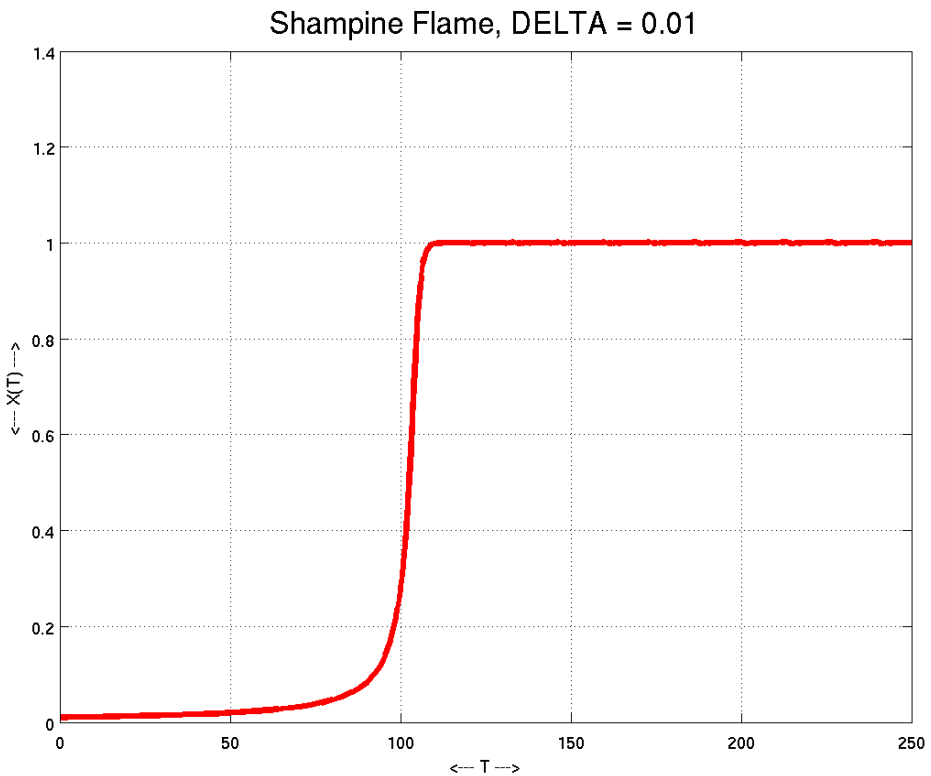
$$\frac{dr}{dt} = r^2 - r^3$$
$$r(0) = \delta$$
$$0 \leq t \leq \frac{2}{\delta}$$

# BANG: Our Model is an Idealization

A real match flame isn't spherical, we're ignoring many chemical effects, but the sudden transition to full flame is what we're really interested in.

# BANG: Solution for $\delta = 0.01$

Letting $\delta = 0.01$, we get a solution that looks like this, with the flame undergoing a very sudden jump to the radius $r = 1$. For future reference, we note that the flame reaches $r = 0.99$ at time $t = 108.176$ seconds.




http://people.sc.fsu.edu/~jburkardt/m_src/flame_ode/base_run.m

# BANG: Explosion Varies with $\delta$?

It's reasonable to assume that $\delta$, the initial size of the flame, has a significant influence on when the explosive growth of the fireball occurs. Since this growth occurs over a finite stretch of time, we can simplify our record keeping by simply noting when the flame has reached a radius of $r = 0.99$.

If we use MATLAB's solver **ode45**, it can monitor the solution and report back to us exactly when this moment occurs.

Suppose we are in the business of making matches, and these vary in size in $[\frac{\overline{\delta}}{2}, 2\overline{\delta}]$, where $\overline{\delta} = 0.01$.

Our customers ask:

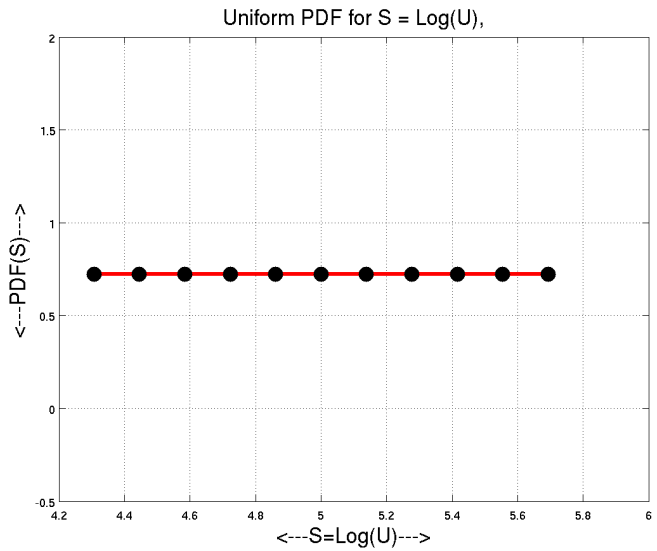
What is the typical time until the match has 99% ignited?

# BANG: PDF for $\ln(\delta)$

We need to describe the variation in the match flame size $\delta$. Suppose we discover that, relative to the standard size $\overline{\delta} = 0.1$, the actual initial flame size is always between half as big and twice as big, with a smooth variation.

This suggests that the logarithm of $\delta$ is uniformly distributed, in which case, the formula for the density is

$$\rho(\ln(\delta)) = \frac{1}{\ln(4)}, \quad \overline{\delta}/2 \leq \delta \leq 2\overline{\delta}$$

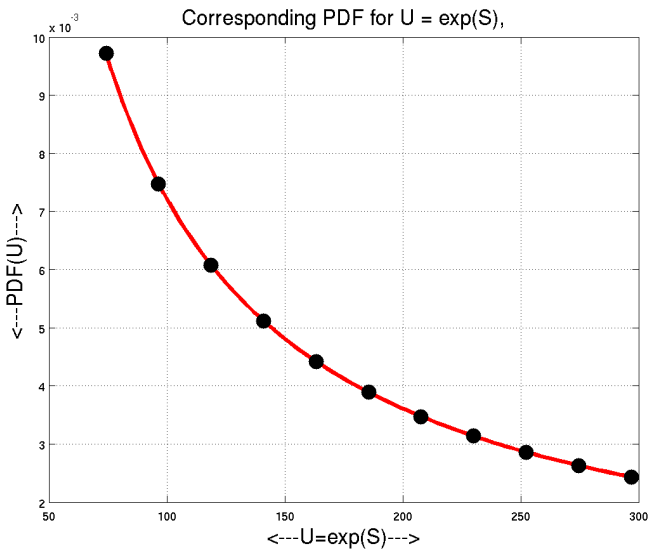


# BANG: PDF for $\delta$

If we can get a PDF for $\ln(\delta)$, then let $s = \ln(\delta)$, and figure:

$$\int_{s1}^{s2} \rho(s)ds = \int_{s1}^{s2} \frac{1}{\ln(4)} ds = \int_{s1}^{s2} \frac{1}{\ln(4)} \frac{ds}{d\delta} d\delta = \int_{\delta 1}^{\delta 2} \frac{1}{\ln(4)\delta} d\delta$$

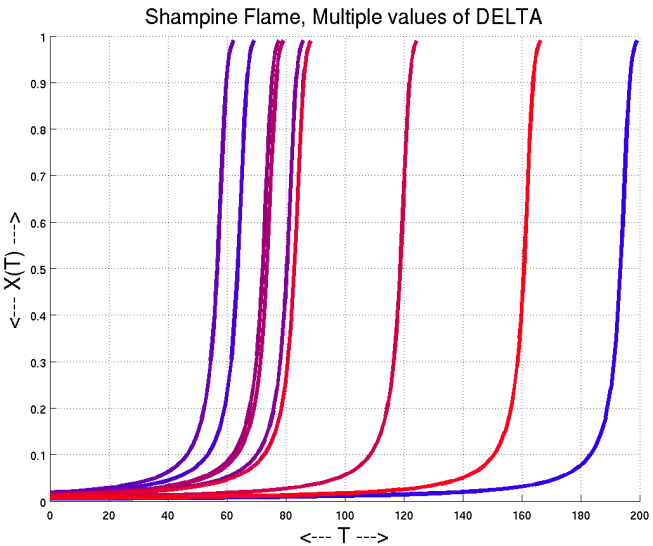
so $\rho(\delta) = \frac{1}{\ln(4)\delta}$:

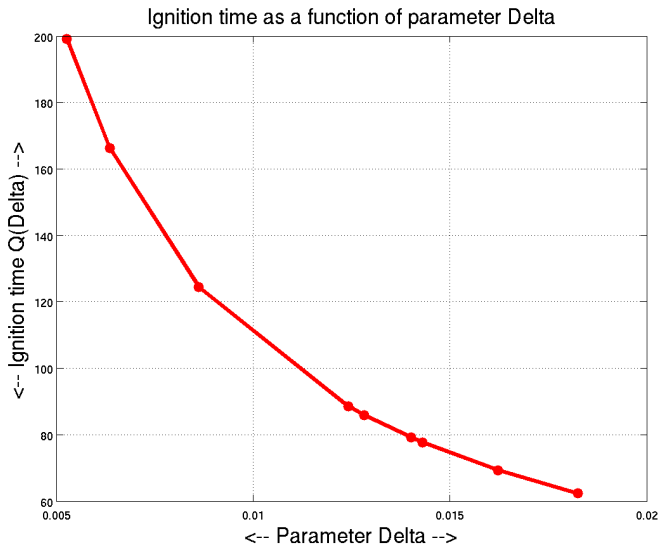



Once we have chosen a model for the PDF of $\delta$, we can calculate quantities of interest.

# BANG: Solutions for $\delta$ near 0.1

Here we have 10 runs using logarithmically uniform samples for $\delta$, as well as a plot estimating the relationship of $\delta$ to our quantity of interest, $q$=time til solution $= 0.99$.




The average time looks to be something like 100 seconds.

http://people.sc.fsu.edu/~jburkardt/m_src/flame_ode/uniform_run.m

# BANG: Expected value of explosion time

To get the expected value for $Q$, we have to be clear about our model for the uncertainty or variation in $\delta$. We can think of this as follows:

- select $u$ uniformly from $[-1, +1]$;
- set $\delta = 2^u \cdot \overline{\delta}$
- solve the ODE for $r(t)$ until time $t*$, when $r(t*) = 0.99$;
- define $q(u) = t*$;

The integral that defines $\overline{q}$ can be written

$$\overline{q} = \int_{-1}^{+1} q(u)\, \rho(u)\, du$$

and since we believe that $u$ is uniformly likely over the interval $[-1, +1]$, we know that $\rho(u) = \frac{1}{2}$.

# BANG: Computed results

Using a Clenshaw Curtis rule, which is conveniently set up exactly for the interval $[-1, +1]$, we get the following estimates:

| N | Estimated $\overline{q}$ |
|---|---|
| 1 | 108.176 |
| 2 | 133.169 |
| 3 | 116.507 |
| 4 | 116.326 |
| 5 | 116.375 |
| 6 | 116.367 |
| 7 | 116.363 |
| 8 | 116.367 |
| 9 | 116.372 |
| 10 | 116.360 |
| 11 | 116.368 |
| 12 | 116.370 |

http://people.sc.fsu.edu/~jburkardt/m_src/flame_ode/qoi_quad_output.txt

# BANG: Alien Blob Exercise

**Exercise 19:** A meteor strikes the earth at time $t_0 = 0$; the meteor includes a blob of alien life, whose volume at time $t_0$ is approximately $v_0 = 50cc$. The blob begins to grow at an exponential rate, governed by

$$\frac{dv}{dt} = k$$

where, based on previous alien infestations, the hourly growth factor $k$ is estimated to be $k = 0.35h^{-1}$. If the estimated values of $v$ and $k$ are taken to be exact, then after 10 hours, the blob will weigh about $1,655cc$. Instead, suppose $k$ is chosen uniformly at random from the interval $[0.30, 0.40]$.

What is the expected value of $v(10)$?

# BANG: Projectile Exercise

**Exercise 20:** A projectile is fired from $(0,0)$, at angle $\alpha$, with initial speed $s$. Let $q$ be the horizontal distance traveled. Then:

$$\frac{dx}{dt} = s\cos(\alpha)$$
$$\frac{dy}{dt} = s\sin(\alpha) - gt$$

with $(x(0), y(0)) = (0,0)$, and $g = 9.8m/s^2$.

We can integrate the two equations to get exact formulas for $x(t)$ and $y(t)$, with $s$ and $\alpha$ still unknown.

If $\alpha$ is 30 degrees, and $s$ is 100, what is $q$? (The $y$ formula tells when the cannonball lands, the $x$ formula how far it went.)

Suppose $s$ is 100, and $\alpha$ is uniformly in $[25, 35]$ (degrees!).

What is the expected value of $q$?

# Stochastic Integrals

- What Does an Integral Tell Us?
- The Probability Density Function
- Sampling from a Probability Density Function
- Approximating an Integral
- A Stochastic Fireball
- **The Multidimensional Problem**
- Approximating Multidimensional Integrals
- Sparse Grids
- Clenshaw Curtis Sparse Grids
- A Stochastic Tidal Wave

# MULT: A Discrete Example

A natural way in which a problem becomes multidimensional occurs if we have more than one outcome, and we cannot describe all the outcomes as the result of a single random process.

A common example involves rolling two dice, (which we can distinguish because they are different colors). Assuming fair dice, then for either die, the 6 outcomes each have a $\frac{1}{6}$ probability.

If we roll the dice together, our outcome space $\Omega$ is the **product** $\Omega = \Omega_1 \times \Omega_2$, of the outcome spaces for each separate die.

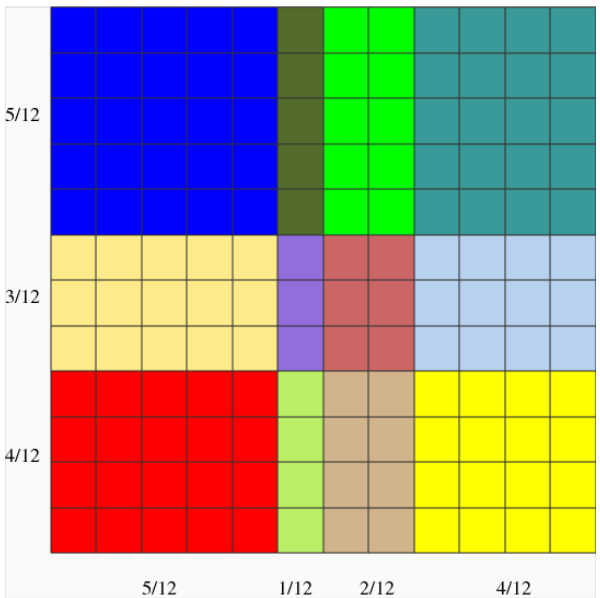
In the product space $\Omega = \Omega_1 \times \Omega_2$:

- a typical outcome is $(e_1, e_2)$.
- the number of possible outcomes is $n_1 \times n_2 = 36$
- the probability of outcome $(e_1, e_2)$ is $p = p_1 \times p_2 = \frac{1}{36}$;
- we notice that the probabilities of the 36 outcomes sum to 1.

# MULT: 2D Discrete Probabilities Sum to 1

If the 1D probabilities sum to 1, so do the 2D probabilities. We can think of this as computing the area of a square 1 inch on a side. One set of probabilities determine horizontal lines, the other verticals. The total area (probability) is 1.

The area of each box (event) is the product of the length and width of its sides (component events).




This idea extends to 3D and beyond.

# MULT: A Discrete Example

We are assuming that the separate dice outcomes are independent. Under that natural assumption, the probability of each pair of outcomes $(e_1, e_2)$ in $\Omega$ is the product of the probabilities of the outcomes in $\Omega_1$ and $\Omega_2$,

$$p_{1,2}(e_1, e_2) = p_1(e_1) \cdot p_2(e_2)$$

This is still true if the dice are not fair, that is, if some faces of each die are more likely to turn up than others. Given the individual probabilities for our unfair dice, we can work out the probabilities for rolling them together.

**Exercise 21:** Make a probability table for the case where the dice are unfair. The red die has probabilities $0.1, 0.1, 0.3, 0.2, 0.0, 0.3$ and the blue die has probabilities $0.4, 0.1, 0.0, 0.3, 0.1, 0.1$.

**Exercise 22:** Using the dice from the previous exercise, what is the expected value of the quantity of interest $e_1 + e_2$, the sum of the dice?

# MULT: PDF's For Continuous Variables

Now suppose the variables $x$ and $y$ vary continuously, and each variable represents a separate independent probabilistic process.

Let $\rho_1(x)$ and $\rho_2(y)$ be the separate PDF's for our two variables. Because of independence, the PDF for the event $(x, y)$ is:

$$\rho(x, y) = \rho_1(x) \cdot \rho_2(y)$$

The 1D PDF's each integrate to 1. Because we are assuming independence, we have the same for the 2D product PDF:

$$\begin{aligned}
\int_{\Omega} \rho(x, y)\, dx\, dy &= \int_{\Omega_2} \int_{\Omega_1} \rho_1(x) \cdot \rho_2(y)\, dx\, dy \\
&= \int_{\Omega_2} \int_{\Omega_1} \rho_1(x) dx \cdot \rho_2(y)\, dy \\
&= \int_{\Omega_2} 1 \cdot \rho_2(y)\, dy \\
&= 1
\end{aligned}$$

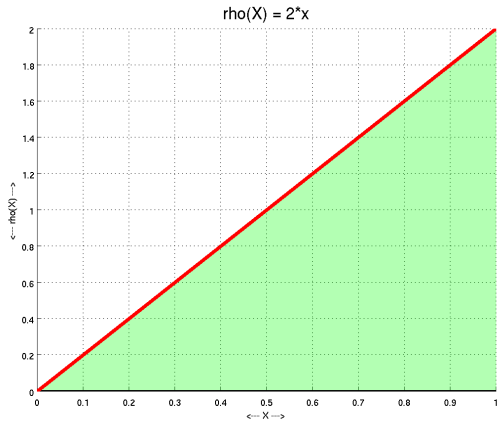
# MULT: Continuous Variables, Finite Domain

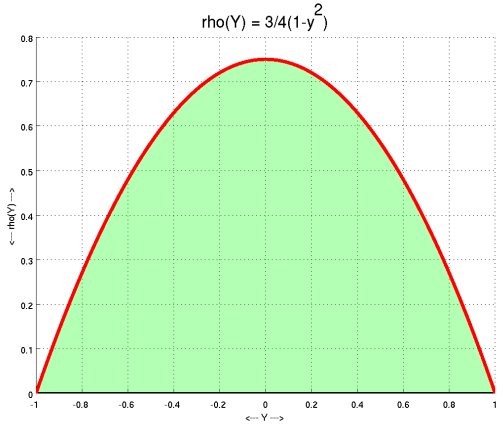
If we have

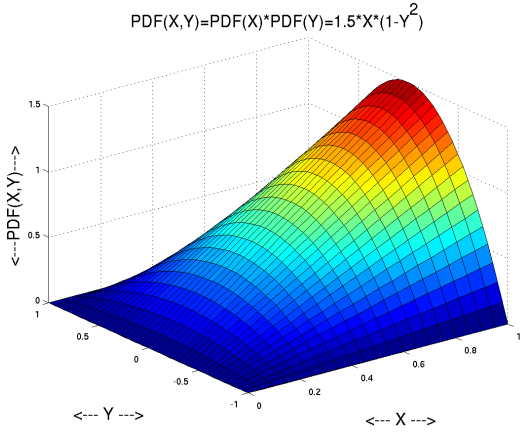
$$\rho_1(x) = 2x \text{ for } x \in [0,1]$$
$$\rho_2(y) = \frac{3}{4}(1-y^2) \text{ for } y \in [-1,+1].$$

then, assuming independence, the variable $(x,y)$ has the pdf

$$\rho(x,y) = \frac{3}{2}x(1-y^2) \text{ for } (x,y) \in [0,1] \times [-1,+1].$$



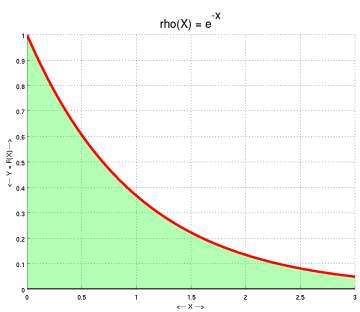
# MULT: Continuous Variables, Semi-Infinite Domain

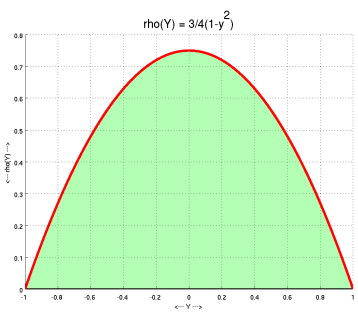
If we have

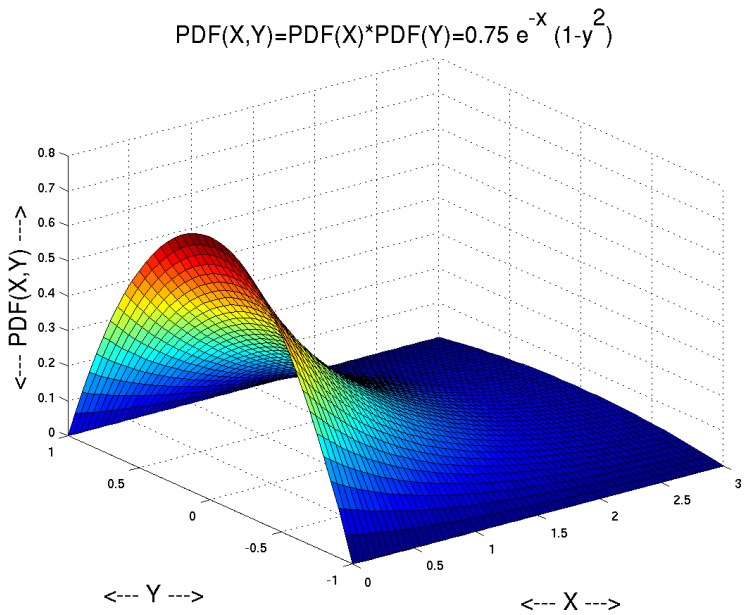
$$\rho_1(x) = e^{-x} \text{ for } x \in [0, +\infty)$$
$$\rho_2(y) = \frac{3}{4}(1 - y^2) \text{ for } y \in [-1, +1].$$

then, assuming independence, the variable $(x, y)$ has the pdf

$$\rho(x, y) = \frac{3}{4} e^{-x}(1 - y^2) \text{ for } (x, y) \in [0, +\infty) \times [-1, +1].$$



# MULT: Continuous Variables, Infinite Domain

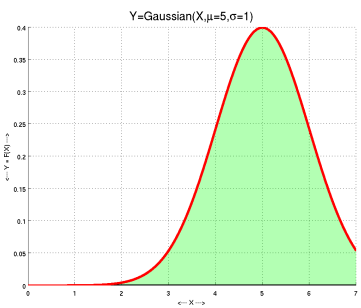
Using the normal or Gaussian PDF, we might have:

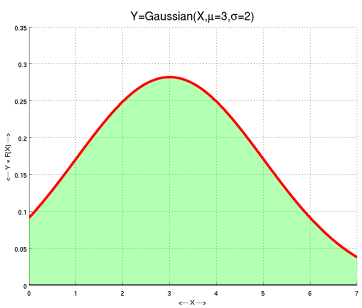
$$\rho_1(x) = e^{-\frac{(x-5)^2}{2}}/\sqrt{2\pi} \text{ for } x \in (-\infty, +\infty)$$

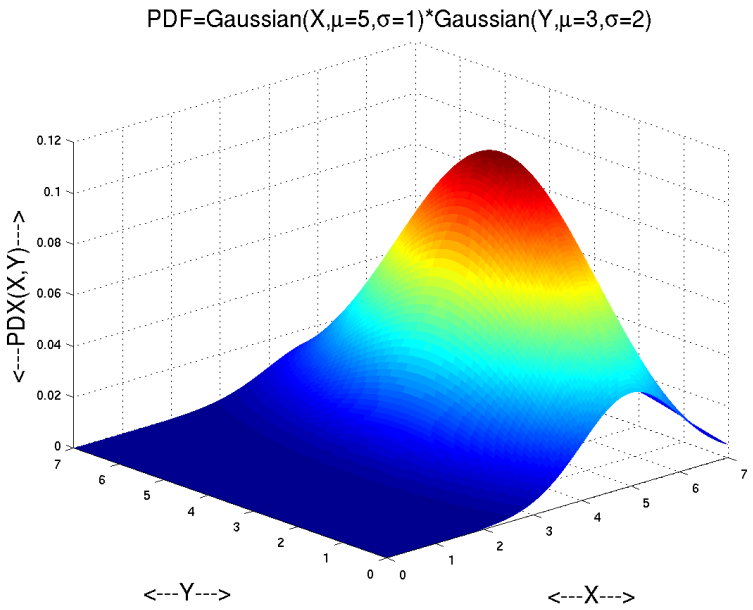
$$\rho_2(y) = e^{-\frac{(y-3)^2}{8}}/\sqrt{4\pi} \text{ for } y \in (-\infty, +\infty).$$

then, assuming independence, the variable $(x, y)$ has the pdf

$$\rho(x, y) = e^{-\frac{4(x-5)^2+(y-3)^2}{8}}/\pi/\sqrt{8} \text{ for } (x, y) \in (-\infty, +\infty)\times(-\infty, +\infty).$$



# MULT: Expected Value for Continuous Variables

In order to compute the expected value of the event $(x, y)$, we need to compute a double integral:

$$\begin{aligned}\overline{(x,y)} &= \int_{a2}^{b2}\int_{a1}^{b1}(x,y)\,\rho(x,y)dx\,dy \\ &=\left(\int_{a2}^{b2}\int_{a1}^{b1} x\,\rho_1(x)\rho_2(y)dx\,dy, \int_{a2}^{b2}\int_{a1}^{b1} y\,\rho_1(x)\rho_2(y)dx\,dy\right) \\ &=\left(\int_{a1}^{b1} x\,\rho_1(x)dx, \int_{a2}^{b2} y\,\rho_2(y)dy\right) \\ &=(\overline{x},\overline{y})\end{aligned}$$

Again, we must assume independence of $x$ and $y$ to make this statement.

**Exercise 23:** Variable $x$ is selected from the interval $[0, 1]$, with a PDF of $\rho_1(x) = 2 \cdot x$. Variable $y$ is selected from the interval $[-1, +1]$ with a PDF of $\rho_2(y) = c \cdot e^y$. Determine the value of $c$ so that $\rho_2(y)$ is a legitimate PDF. Then, assuming that $x$ and $y$

# MULT: Expected Value of QoI for Continuous Variables

In order to compute the expected value of a quantity of interest, $f(x, y)$, we need to compute a double integral in which each value of $f()$ is weighted by the probability that that particular pair of input arguments $(x, y)$ would occur. (Again, we are assuming independence.)

$$\overline{f(x,y)} = \int_{a2}^{b2} \int_{a1}^{b1} f(x,y)\rho_1(x)\rho_2(y)dx\,dy$$

**Exercise 24:** For the variables in the previous exercise, what is the expected value of the quantity of interest $f(x, y) = x \cdot y$?

# MULT: Continuous Variables

A problem can become multidimensional when we consider multiple uncertainties in the input data.

The projectile problem, as originally posed, essentially asked the question, what is the expected value of $x$, the horizontal distance that the projectile has traveled when it lands, assuming that the aiming angle $\alpha$ has a small uncertain variation.

This required us to think of the distance $x$ as a function of $\alpha$, and to compute:

$$\overline{x} = \int_{25}^{35} x(\alpha) \cdot \rho(\alpha)\, d\alpha$$

If we further assume that the initial speed $s$ is uncertain, we instead must compute

$$\overline{x} = \int_{s1}^{s2} \int_{25}^{35} x(\alpha, s) \cdot \rho_1(\alpha)\, d\alpha \cdot \rho_2(s)\, ds$$

# MULT: A Few Dimensions Becomes Many Dimensions

Obviously, we could have many uncertain input parameters, each of which could be assigned a PDF, so that we would add another layer of integration to average over that uncertainty. So you can easily imagine integrals over probability product regions of dimension 5, or maybe 10.

However, we may want to use many more dimensions than that.

Financial calculations involving a 30 year adjustable-rate mortgage must consider 360 monthly payments. At the end of each month, the mortgage rate may have changed. If we regard these rates as independent, then we need to include 360 uncertain parameters, each with a PDF, requiring 360 nested integral signs!

How can we approximate 360 iterated integrals?

# MULT: A Few Dimensions Becomes Infinite Dimensions

Theoretically, our task can be even worse. Suppose we are considering the steady heat equation over a rectangle, with an uncertain boundary condition along one side. This boundary condition might be written as $f(x)$, and it represents a random field, that is, the value of this function at every point $x$ is unknown.

Many problems are like this, involving uncertain functions rather than uncertain parameters (which are just a few numbers).

Computationally, we might handle this by discretizing $f(x)$ as a piecewise linear function. However, the more accuracy we need, the more breakpoints we must use, and the higher the dimension of our probability space.

Again, we need to approximate many iterated integrals.

# Stochastic Integrals

- What Does an Integral Tell Us?
- The Probability Density Function
- Sampling from a Probability Density Function
- Approximating an Integral
- A Stochastic Fireball
- The Multidimensional Problem
- **Approximating Multidimensional Integrals**
- Sparse Grids
- Clenshaw Curtis Sparse Grids
- A Stochastic Tidal Wave

# PROD: Approximating a Multidimensional Integral is Hard

We consider the problem of approximating a multidimensional integral that includes a weight or PDF of the vector argument $\vec{x}$:

$$I(f, \Omega) = \int_{\Omega} f(\vec{x})\rho(\vec{x})\, d\vec{x}$$

Our probability integrals will actually be much simpler than this general form. But before we point that out, let's take a moment to give some more respect to the Monte Carlo method.

One reason that the Monte Carlo method is so powerful is that we can easily see how to apply it in many situations where other methods can't be applied, or can't produce a useful result in a reasonable time.

# PROD: Monte Carlo Can Handle Hard Problems

So suppose that our domain $\Omega$ is not a rectangle, and our density $\rho()$ does not factor into $\rho_1(x) \cdots \rho_2(x_2) \cdots \rho_n(x_n)$. The Monte Carlo method doesn't care about that. All we need to be able to do is provide the area or volume of the domain, $V(\Omega)$, and be able to produce sample points within the region.

If we can sample points $u_i$ uniformly in the region, we write

$$I(f, \Omega) \approx \frac{V(\Omega)}{n} \cdot \sum_{i=1}^{n} f(\vec{u_i}) \rho(\vec{u_i})$$

If we can sample points $x_i$ in a way that matches the density $\rho(\vec{x})$, we write

$$I(f, \Omega) \approx \frac{V(\Omega)}{n} \cdot \sum_{i=1}^{n} f(\vec{x_i})$$

# PROD: Monte Carlo Can Handle Hard Problems

To estimate the integral of $x^2$ over the surface of the sphere of radius $r = 2$, and center $c = (1, 1, 1)$, with surface area $4\pi r^2$, the tricky part is getting $n$ uniform sample points on the surface.

To do this, choose $n$ triples of normal random numbers, normalize each triple to have unit norm, multiply by $r$ and add $c$.

Our Monte Carlo results are then:

| N | Estimate | Error |
|---|---|---|
| 1 | 9.57065 | 107.715 |
| 10 | 56.6802 | 60.6059 |
| 100 | 101.817 | 15.4691 |
| 1000 | 116.302 | 0.984061 |
| 10000 | 116.01 | 1.2766 |
| 100000 | 116.961 | 0.324764 |
| 1000000 | 117.337 | 0.0507664 |

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/sphere_surface.m

# Halfway there!

# Approximating Integrals for Stochastic Problems

John Burkardt
Department of Scientific Computing
Florida State University

.....

ISC 5936-01:
Numerical Methods for Stochastic Differential Equations
http://people.sc.fsu.edu/~jburkardt/presentations/...
stochastic_integrals.pdf

.....

Revised: 20 March 2013

26/28 February, 5/7/19/21 March 2013

# PROD: Probabilistic Integrals are Simpler

Our probabilistic integrals are typically simpler:

- the region $\Omega$ is actually a product of 1D regions;
- the 1D regions are typically intervals;
- the weight or PDF is actually a product of 1D PDF's.

so we can write instead:

$$I(f,\Omega) = \int_{\Omega} f(\vec{x})\rho(\vec{x})\, d\vec{x}$$
$$= \int_{a_n}^{b_n} \cdots \int_{a_1}^{b_1} f(\vec{x})\rho_1(x_1)dx_1 \cdots \rho_n(x_n)dx_n$$

# PROD: Approximate Multiple Integrals with Monte Carlo

Let's start by approximating a 2D integral of the form:

$$I(f, [a,b] \times [c,d]) = \int_c^d \int_a^b f(x,y)\, dx\, dy$$

This integral is defined over a product region $[a,b] \times [c,d]$.

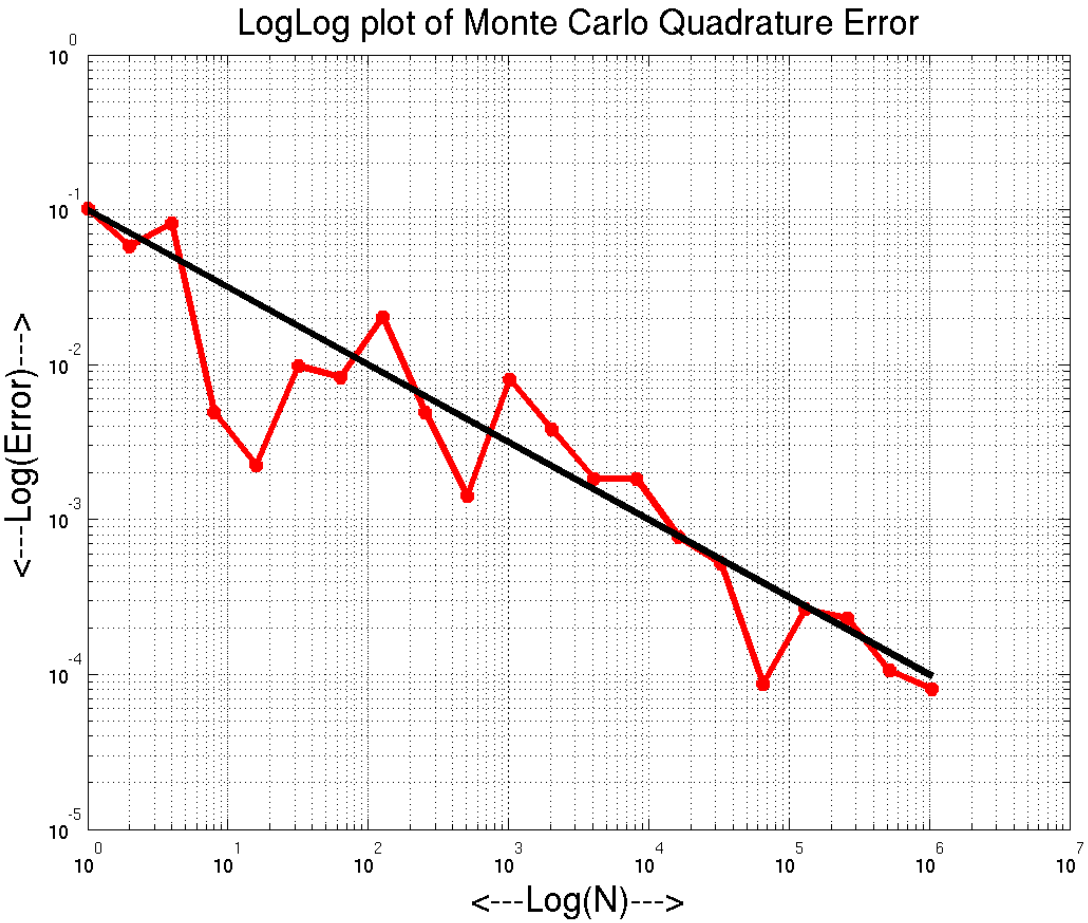
So all we have to do is select $n$ points $(x,y)$ uniformly, with $x \in [a,b]$ and $y \in [c,d]$, sum the function values and multiply by $\frac{(b-a)(d-c)}{n}$.

For instance, suppose we wish to approximate the integral of $f(x,y) = e^{-(x-0.3)^2-(y-0.4)^2}$ over $(x,y) \in [0,1] \times [0.1]$. The exact integral is

$$0.25\pi(\text{erf}(0.4)+\text{erf}(0.6))\cdot(\text{erf}(0.3)+\text{erf}(0.7)) \approx 0.8159373426556018352 6$$

# PROD: Approximate Multiple Integrals with Monte Carlo

Notice, the behavior of the Monte Carlo error is exactly the same for this 2D problem, that is, the rate of error decay seems to have the same slope as the 1D problem.




http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/mc_2d_test.png

# PROD: Approximate Multiple Integrals with Product Rules

Let's try a different approach to approximating the multidimensional integral, by considering it to be a sequence of nested 1D integrals.

For each value of $y$, we could approximate the inner integral over $x$ by a quadrature rule, leaving us with an integral over $y$.

$$\int_c^d \int_a^b f(x,y)\,dx\,dy \approx \int_c^d \sum_{i=1}^{m} u_i f(x_i,y)\,dy$$

Now we just have a function of $y$ to integrate:

$$\begin{aligned}\int_c^d \int_a^b f(x,y)\,dx\,dy &\approx \sum_{j=1}^{n} v_j \sum_{i=1}^{m} u_i f(x_i,y_j) \\ &= \sum_{j=1}^{n}\sum_{i=1}^{m} u_i\, v_j\, f(x_i,y_j)\end{aligned}$$

# PROD: Forming a Product Rule

We have a rule of order $n1$ for $x$, with points $x_i$ and weights $u_i$, and another rule of order $n2$ for $y$, with points $y_j$ and weights $v_j$.

Another way of looking at what we did on the previous slide assumes that we created a 2D quadrature rule, of order $n1 \cdot n2$, for $(x, y)$, with points $(x_i, y_j)$ and weights $u_i \cdot v_j$, as follows:

```
             u1    *f(x1)        u2    *f(x2)        ... un1    *f(x
           +-------------------------------------------------------
v1*f(y1)   | u1*v1*f(x1,y1)    u2*v1*f(x2,y1)    ... un1*v1*f(x
v2*f(y2)   | u1*v2*f(x1,y2)    u2*v2*f(x2,y2)    ... un1*v2*f(x
...        | ...               ...               ... ...
vn*f(yn2)| u1*vn2*f(x1,yn2) u2*vn2*f(x2,yn2) ... un1*vn2*f(
```

# PROD: Approximate Multiple Integrals with Product Rules

Let us apply a product rule to our sample integral $f(x,y) = e^{-(x-0.3)^2-(y-0.4)^2}$ over $(x,y) \in [0,1] \times [0.1]$.

Let's use the Gauss-Legendre rule for both directions, but the $n1$-point rule in $x$ and the $n2$ point rule in $y$.

We can compute the 1D rules, use a **for** loop to carry out the product, and we never actually have to form the 2D rule:

```
q = 0.0;
for j = 1 : n2
  for i = 1 : n1
    fxy = exp ( - ( x(i) - 0.3 )^2 -( y(j) - 0.4 )^2 );
    q = q + u(i) * v(j) * fxy;
  end
end
```

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/quad_2d.m

# PROD: Approximate Multiple Integrals with Product Rules

On the other hand, we may really want to set up the 2D rule:

```
k = 0;
for j = 1 : n2
  for i = 1 : n1
    k = k + 1;
    z(1:2,k) = [ x(i); y(j) ];
    w(k) = u(i) * v(j);
  end
end
```

In that case, we can later use the 2D rule directly:

```
q = 0.0;
for k = 1 : n1*n2
  q = q + w(k) ...
    * exp ( - ( z(1,k) - 0.3 )^2 -( z(2,k) - 0.4 )^2 );
end
```

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/quad_2d_compute.m

# PROD: A 2D Quadrature Rule

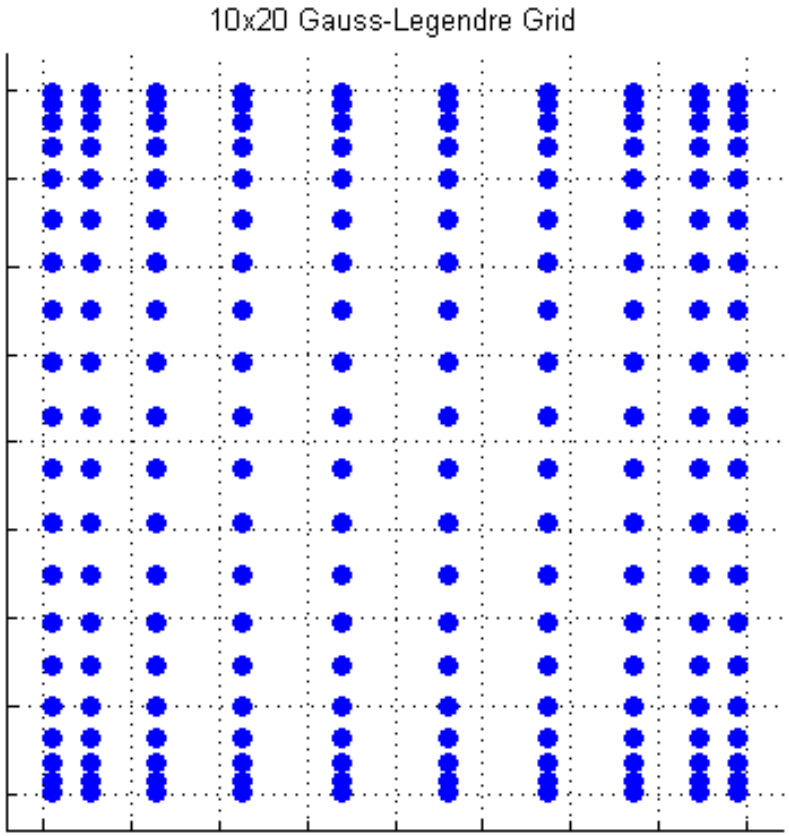
If we use the Gauss-Legendre rule again, of orders $n1 = 2$ and $n2 = 3$, adjusted to the interval $[0, 1]$ we get the following 6 point rule for 2D:

```
   W          X          Y
------     ------     ------
0.1389     0.2113     0.1127
0.1389     0.7887     0.1127
0.2222     0.2113     0.5000
0.2222     0.7887     0.5000
0.1389     0.2113     0.8873
0.1389     0.7887     0.8873
```

Notice, in particular, that the sum of the elements of W is 1!

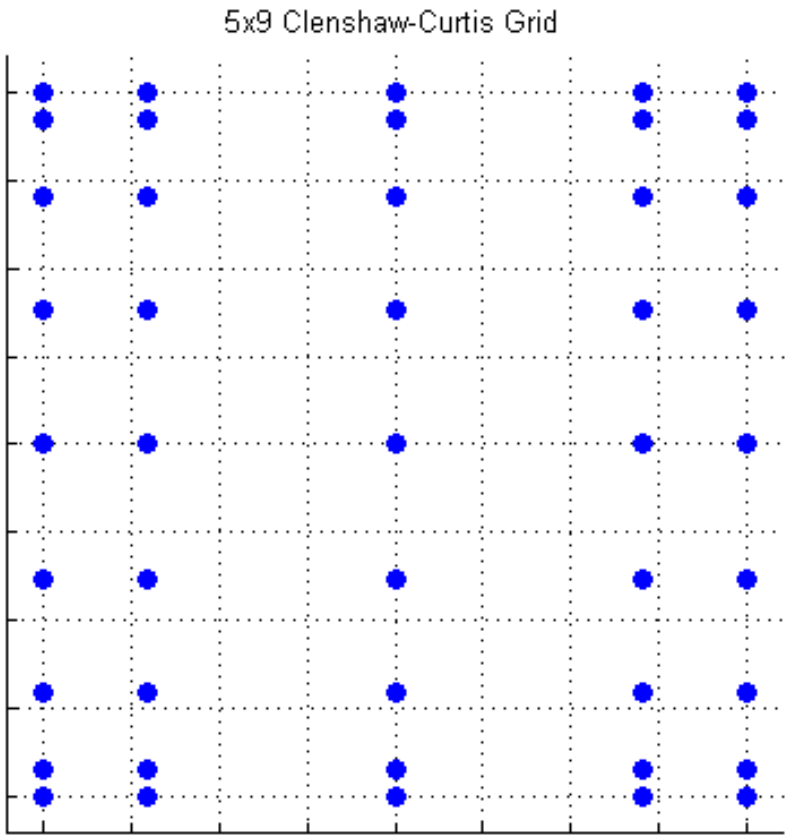
# PROD: Picture of 2D Gauss Product Rule

Here is a picture of the arrangement of points for a product of two Gauss-Legendre rules of orders 10 and 20:



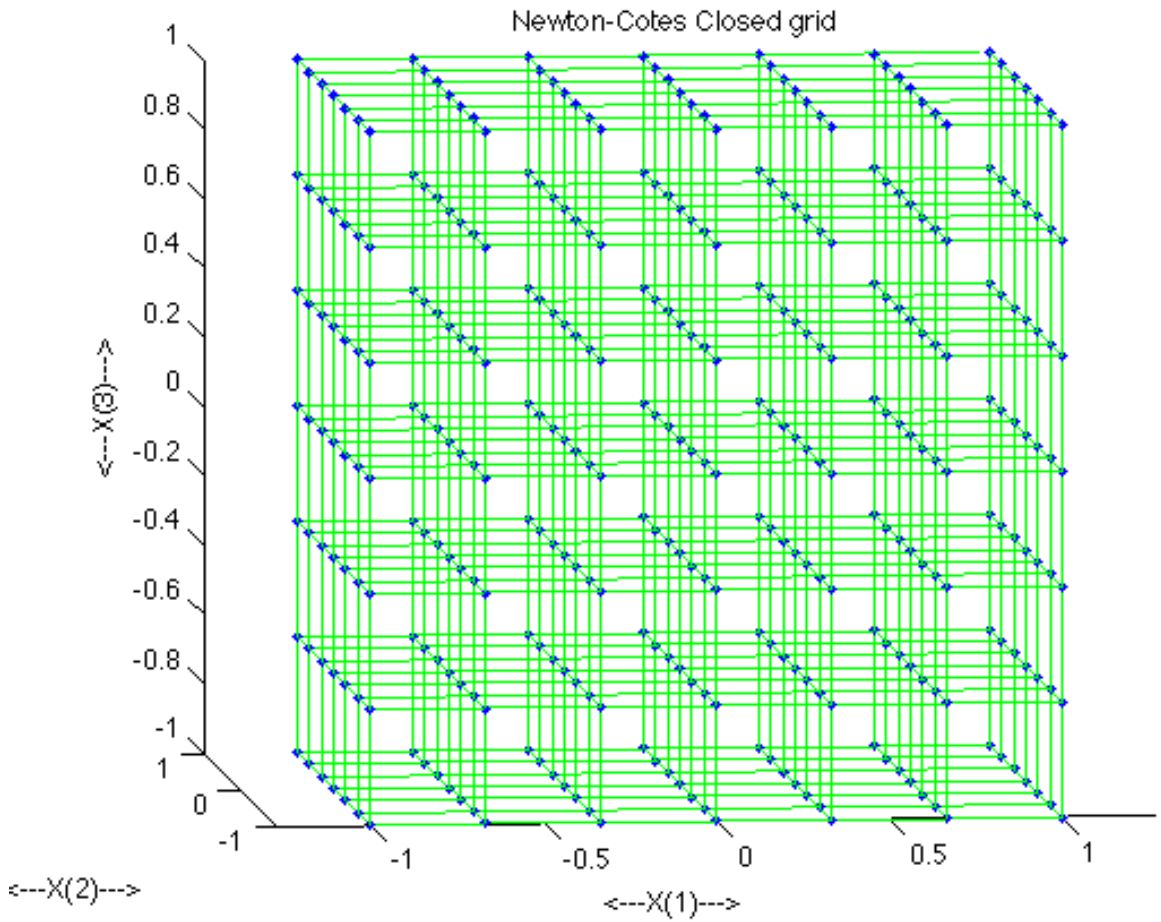
# PROD: Picture of 2D Clenshaw-Curtis Product Rule

Here is a picture of the arrangement of points for a product of two CC rules of orders 5 and 9:




# PROD: Picture of 3D Newton-Cotes Rule

Here is a picture of the arrangement of points for a product of three Newton-Cotes rules of orders 7, 7 and 7:



# PROD: A 2D Quadrature Rule

Now, whether we use the 1D rules in a pair of loops, or the 2D rule, we can apply quadrature to our integral:

| | 1 | 2 | N1 = 3 | 4 |
|---|---|---|---|---|
| 1 | 0.135292 | 0.060695 | 0.0630469 | 0.0629982 |
| 2 | 0.065077 | 0.00401373 | 0.0018354 | 0.00188054 |
| N2= 3 | 0.0671508 | 0.00210248 | 0.0000809 | 0.00003573 |
| 4 | 0.0671109 | 0.00213933 | 0.0000440 | 0.00000121 |
| 5 | 0.0671114 | 0.0021388 | 0.0000445 | 0.00000068 |

Notice that moving along the diagonal seems the best bet!

http://people.sc.fsu.edu/~jburkardt/latex/stochastic_integrals/quad_2d.m

# PROD: Product Rules

What this means is that if we have 1D quadrature rules appropriate for integrals over the $x$ and $y$ intervals, then we now have a formula for producing a quadrature rule for integrals over the 2D product region.

If the 1D rules use $n1$ and $n2$ points respectively, then the 2D rule uses $n1 \times n2$ points, whose points $(x_i, y_j)$ are all possible pairs of the 1D points, and whose weights are the corresponding products $v_i w_j$ of the 1D weights.

Moreover, the same technique extends to 3D regions and beyond. The only requirement is that the integration region must be expressed as a product region, that is, a sort of rectangle, or hyper-rectangle, which includes cases where some dimensions are semi-infinite or infinite.

# PROD: Create a Clenshaw-Curtis Product Rule

**Exercise 25:**: Approximate the integral of $f(x,y) = e^{-(x-0.3)^2-(y-0.4)^2}$ over $(x,y) \in [0,1] \times [0,1]$, by creating $3 \times 3$, $4 \times 4$ and $5 \times 5$ product rules from 1D Clenshaw-Curtis rules. How does the error in the CC product rules compare to GL rules of the same order?

**Exercise 26:**: Approximate the integral of $f(x,y,z) = x^2 y^3 z$ over $(x,y,z) \in [0,1]^3$ by creating and using a $4 \times 4 \times 4$ product rule from 1D Clenshaw-Curtis rules. Your integral estimate should be exact. What happens if you try a $3 \times 4 \times 4$ rule? How far can you reduce each 1D order and still get an exact answer? Can you guess the lowest order CC product rule you could use to integrate $x^a y^b z^c$ exactly?

# PROD: Product Rules

Wonderfully enough, if the $x$ rule is exact for polynomials up to degree $p$ and the $y$ rule is exact for polynomials up to degree $q$, then the product rule will be exact for polynomials whose $x$ degree does not exceed $p$ and whose $y$ degree does not exceed $q$.

In particular, if both rules are exact for linears, then the product rule integrates 1, $x$, $y$, and $xy$ exactly.

In general, if the quadrature rule $Q()$ is the product of rules $Q_1()$ and $Q_2()$, with precisions $p_1$ and $p_2$ respectively, then $Q()$ has total precision $p = \min(p_1, p_2)$, and component precision $(p_1, p_2)$.

# PROD: Component Precision

If a 2D quadrature rule has component precision $(3, 5)$, it can integrate exactly any polynomial with the property that, in every term, $x$ never has an exponent greater than 3, and $y$ never has an exponent greater than 5.

A 2D quadrature rule with (3,5) component precision exactly integrates:

| | | | | |
|---|---|---|---|---|
| $y^5$ | $y^5$ | $xy^5$ | $x^2y^5$ | $x^3y^5$ |
| $y^4$ | $y^4$ | $xy^4$ | $x^2y^4$ | $x^3y^4$ |
| $y^3$ | $y^3$ | $xy^3$ | $x^2y^3$ | $x^3y^3$ |
| $y^2$ | $y^2$ | $xy^2$ | $x^2y^2$ | $x^3y^2$ |
| $y$ | $y$ | $xy$ | $x^2y$ | $x^3y$ |
| 1 | 1 | $x$ | $x^2$ | $x^3$ |
| $x^i y^j$ | 1 | $x$ | $x^2$ | $x^3$ |

# PROD: Total Precision

If a 2D quadrature rule has total precision 4, it can integrate exactly any polynomial with the property that, in every term, the sum of the $x$ and $y$ exponents is always 4 or less.

A 2D quadrature rule with total precision 4 exactly integrates:

| | | | | | |
|---|---|---|---|---|---|
| $y^4$ | $y^4$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $y^3$ | $y^3$ | $xy^3$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $y^2$ | $y^2$ | $xy^2$ | $x^2y^2$ | $\cdots$ | $\cdots$ |
| $y$ | $y$ | $xy$ | $x^2y$ | $x^3y$ | $\cdots$ |
| $1$ | $1$ | $x$ | $x^2$ | $x^3$ | $x^4$ |
| $x^iy^j$ | $1$ | $x$ | $x^2$ | $x^3$ | $x^4$ |

# PROD: Total/Component Precision

**Exercise 27:** If a quadrature rule has total precision $p$, does this imply it has component precision $(p, p, \cdots, p)$? If a quadrature rule has component precision $(p, p, \cdots, p)$, does this imply it has total precision $p$?

**Exercise 28:** A quadrature rule for 3D data has component precision $(4, 2, 3)$. What is the total precision of this rule?

**Exercise 29:** A quadrature rule for 2D data has total precision 6. Specify a component precision $(p_1, p_2)$ that this rule is guaranteed to have. There are actually many possible answers.

# PROD: Pascal Triangle for 2D Monomials

Here is a sort of Pascal triangle for all monomials in $x$ and $y$.

$$
\begin{array}{ccccccccccccccccc}
 & & & & & & & & 1 & & & & & & & & \\
 & & & & & & & x & & y & & & & & & & \\
 & & & & & & x^2 & & xy & & y^2 & & & & & & \\
 & & & & & x^3 & & x^2y & & xy^2 & & y^3 & & & & & \\
 & & & & x^4 & & x^3y & & x^2y^2 & & xy^3 & & y^4 & & & & \\
 & & & x^5 & & x^4y & & x^3y^2 & & x^2y^3 & & xy^4 & & y^5 & & & \\
 & & x^6 & & x^5y & & x^4y^2 & & x^3y^3 & & x^2y^4 & & xy^5 & & & & \\
 & x^7 & & x^6y & & x^5y^2 & & x^4y^3 & & x^3y^4 & & x^2y^5 & & xy^6 & & & \\
x^8 & & x^7y & & x^6y^2 & & x^5y^3 & & x^4y^4 & & x^3y^5 & & x^2y^6 & & & &
\end{array}
$$

# PROD: Monomials up to Component Degree (4,4)

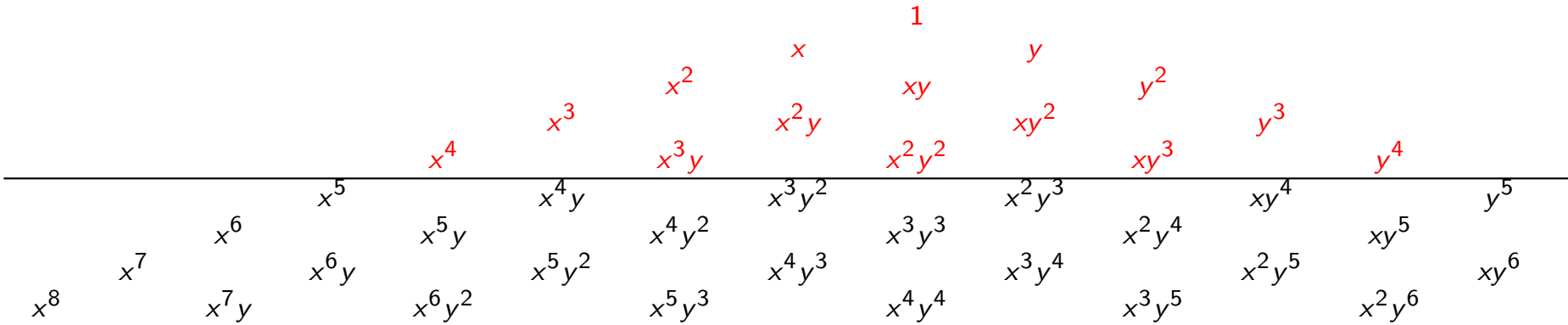
If we use a 2D quadrature rule with (4,4) component precision, we integrate exactly the 25 monomials in red.

$$
\begin{array}{ccccccccccc}
 & & & & & & 1 & & & & \\
 & & & & & x & & y & & & \\
 & & & & x^2 & & xy & & y^2 & & \\
 & & & x^3 & & x^2y & & xy^2 & & y^3 & \\
 & & x^4 & & x^3y & & x^2y^2 & & xy^3 & & y^4 \\
 & x^5 & & x^4y & & x^3y^2 & & x^2y^3 & & xy^4 & & y^5 \\
x^6 & & x^5y & & x^4y^2 & & x^3y^3 & & x^2y^4 & & xy^5 & \\
 & x^6y & & x^5y^2 & & x^4y^3 & & x^3y^4 & & x^2y^5 & & xy^6 \\
x^7 & & x^6y^2 & & x^5y^3 & & x^4y^4 & & x^3y^5 & & x^2y^6 & \\
x^8 & & x^7y & & & & & & & & &
\end{array}
$$

Roughly speaking, we can expect our error to be $O(h^5)$, because there are monomials of that degree that we have missed.

# PROD: Monomials up to Total Degree 4

If we use a 2D quadrature rule with total precision 4, we integrate exactly the 15 monomials in red.




Roughly speaking, we can expect our error to be $O(h^5)$.

# PROD: A 2D Quadrature Rule

Recall our experience integrating a function with product rules of different component precisions. It really did not pay to increase the precision in $x$ but not $y$. For a general integrand, the error in an interpolatory quadrature rule will tend to be dominated by the monomial of lowest degree that you can't integrate exactly.

| | | N = 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| | 1 | 0.135292 | 0.060695 | 0.0630469 | 0.0629982 |
| | 2 | 0.065077 | 0.00401373 | 0.0018354 | 0.00188054 |
| M = | 3 | 0.0671508 | 0.00210248 | 0.0000809 | 0.00003573 |
| | 4 | 0.0671109 | 0.00213933 | 0.0000440 | 0.00000121 |
| | 5 | 0.0671114 | 0.0021388 | 0.0000445 | 0.00000068 |

When we stray off the diagonal, we are focussing on one variable, but ignoring the other.

# PROD: Control Total Degree

To approximate the integral with an error estimate, we need a sequence of rules of increasing precision.

If the $x$ and $y$ interval sizes are roughly equal ($h_x, h_y \approx h$), and our 2D quadrature rule precisely integrates all monomials of total degree $p$, the error will behave like $h^{p+1}$.

If our 2D quadrature rule has component precision $(p, p)$, then we approximate more monomials, but our error still behaves like $h^{p+1}$.

We prefer a sequence of rules that marches down Pascal's triangle a line at a time. The extra monomials below the line don't help our asymptotic error, and they presumably cost us in function evaluations.

# PROD: Not That Many Monomials!

We already noted that, in $M$ dimensions, a product rule of 1D quadrature of order $K$ uses $N = K^M$ points. But how many M-dimensional monomials of total degree $K$ are there?

The space of $M$-dimensional polynomials of degree $K$ or less has dimension $\left( \begin{array}{c} K+M \\ M \end{array} \right) \approx \frac{M^K}{K!}$.

In particular, letting $M = 100$ and $K = 1$, we see that in 100-dimensional space, there are 101 monomials of degree 1 or less, namely, 1, $x_1$, $x_2$, ..., $x_{100}$, and only 5,151 quadratics or less, and "only" 176,851 cubics or less. A good quadrature rule should be able to capture these integrals without needing to evaluate the function as trillions of points!

# PROD: The Smolyak Approach

If we need to estimate the integral of a smooth function in a high dimensional space, then we may be able to achieve highly accurate results at low cost, using an approach proposed by Smolyak.

The idea is to construct a family of quadrature rules of increasing total precision, that is, they try to fill in Pascal's triangle (or tetrahedron or multidimensional simplex) one more line at a time.

If this is done in the right way, we can reach a desired level of accuracy, even in a fairly high dimension, at low cost in function evaluations.

The surprising thing is that these rules are constructed out of (low order versions of) the very product rules that they are going to replace!

# Stochastic Integrals

- What Does an Integral Tell Us?
- The Probability Density Function
- Sampling from a Probability Density Function
- Approximating an Integral
- A Stochastic Fireball
- The Multidimensional Problem
- Approximating Multidimensional Integrals
- **Sparse Grids**
- Clenshaw Curtis Sparse Grids
- A Stochastic Tidal Wave

# SPARSE: The Challenge of High Dimensions

Let's ignore the complications of geometry, weight functions, independence and so on, and focus on the issues that might arise if we try to approximate an integral of the form

$$\int_{[0,1]^m} f(x)\, dx$$

where the spatial dimension $m$ is going to be "large".

The natural approach to this problem would be to try a product rule. Since we have no information about the function in advance, we would use the same rule, of the same order, for each spatial dimension. If our product rule is based on a 1D $k$ point rule, then we might expect our polynomial precision to be $k$.

Our strategy, then, might be to use a sequence of product rules derived from the 1D rules of order 1, 2, 3, ... up to, say 10, which seems enough accuracy, at least for smooth integrands.

# SPARSE: Product Rules Blow Up!

Here is a partial table of the number of points $N$ for a product rule in $M$ dimensions, based on a 1D rule of order $K$:

| | K=1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| M=1 | 1 | 2 | 3 | 4 | 5 |
| 2 | 1 | 4 | 9 | 16 | 25 |
| 3 | 1 | 8 | 27 | 64 | 125 |
| 4 | 1 | 16 | 81 | 256 | 625 |
| 5 | 1 | 32 | 243 | 1,024 | 3,125 |
| 10 | 1 | 1,024 | 59,049 | 1,048,576 | 9,765,625 |
| 20 | 1 | million | | | |
| 30 | 1 | billion | | | |
| 40 | 1 | trillion | | | |
| 100 | 1 | forget it! | | | $N = K^M$ |

Even at dimension $M = 10$, we can see that a product rule based on a 1D rule of 5 points is very expensive. Except for very limited cases, we can't go far in $M$ or $K$!

# SPARSE: Seeking Alternative to Product Rules

We can't even use a product rule based on a 2 point 1D rule beyond dimension $M = 20$, when really, in that case we are essentially looking for linear behavior in the integrand.

This is especially odd because the 1D rule is essentially estimating the linear behavior of the integrand. Even in 100 dimensions, we could estimate this for each dimension by evaluating the function once at a central point, and comparing its value in each of the 100 directions, for a total of 101 points, not $2^{100}$ points.

This is a hint that there may be a solution, despite the fact that product rules are not the solution!

What we are hoping for is some family of quadrature rules that will give us a choice of a family of rules of known precision, with a few choices of increasing precision (so we can estimate error) that works in dimensions of 20 or 30 or 50, perhaps, while using less than 1,000,000 points per rule!

# SPARSE: Beat the Product Rule!

An alternative to the product rules is known as the Smolyak procedure.

We still assume that our integration region $\Omega$ is a product space of dimension $M$. To keep things simple, we'll assume for now that $\Omega = [0,1]^M$.

Let's also assume that we have a family of quadrature rules for the 1D problem, and that these are indexed by order (number of points) which we'll also take to be equivalent to precision (degree of highest polynomial the rule can integrate exactly.) We can call these 1D rules Q(0), Q(1), Q(2), ..., Q(N).

If we form a 2D product rule, we would prefer to write something like $Q(2,5)$, meaning $Q(2,5) = Q(2) \times Q(5)$, but for brevity we may occasionally just write $(2,5)$. We'll let the numbers indicate the precision.

# SPARSE: Hard to Explain

Smolyak's sparse grid procedure may seem hard to understand. But be fair, it is also hard to explain! There are several approaches to explanation:

- *"Null Physics is derived from the concept that our entire universe is the internal structure of nothingness. In other words, physical reality is an intricate, four-dimensional geometric equation that adds to zero because it exists within zero."*, Terence Witt;
- *"Explain it to me like I'm a Golden Retriever."*, a Wall Street executive rejects his advisor's report;
- *"'Shut up!', he explained."*, Ring Lardner;
- *"Sir, I have given you an argument. I am not also obliged to provide you with an understanding."*, Samuel Johnson;
- *"A little inaccuracy saves tons of explanation."*, Saki.

We will go with the last approach!

# I left you hanging last time!

# Approximating Integrals for Stochastic Problems

John Burkardt
Department of Scientific Computing
Florida State University

.....

ISC 5936-01:
Numerical Methods for Stochastic Differential Equations
http://people.sc.fsu.edu/~jburkardt/presentations/...
stochastic_integrals.pdf

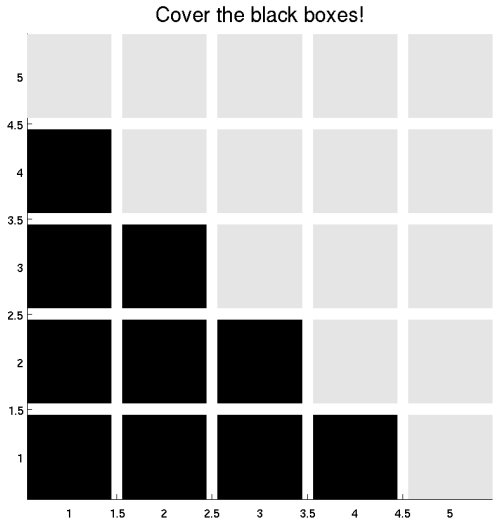
.....

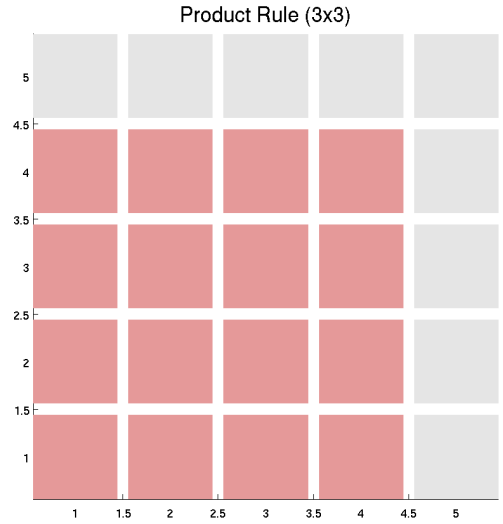
Revised: 20 March 2013

26/28 February, 5/7/19/21 March 2013

# SPARSE: Cover the Squares

Let's look at the task of devising a 2D quadrature rule that has total precision 3. That means it has to cover all the black squares on the following diagram, where you must imagine that the $x$ axis represents $1, x, x^2, x^3, x^4$, the $y$ axis is similar, and the box three positions left and two up represents the term $x^2y$.

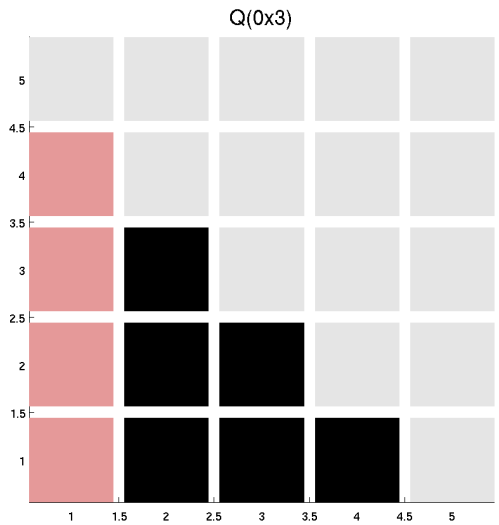


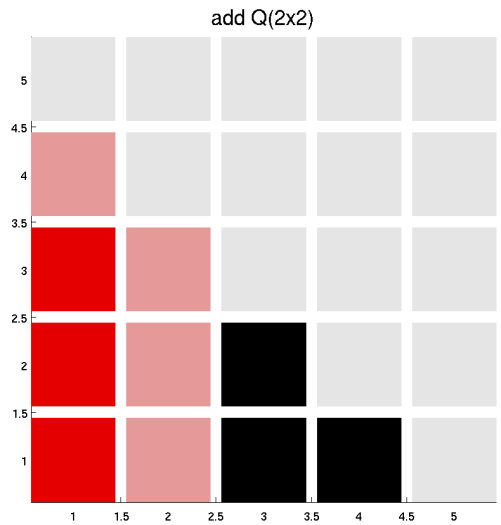




Using a product rule $Q(3,3) = Q(3) \times Q(3)$, with component precisions $(3,3)$, we cover all the black boxes. It's hard to imagine a better solution.

# SPARSE: Cover the Squares

But suppose we wanted to try to cover **only** the black boxes? A $(0, 3)$ rule would get the first column. We could "add" a $(1, 2)$ rule to get the next column, but there's some overlap (dark red).

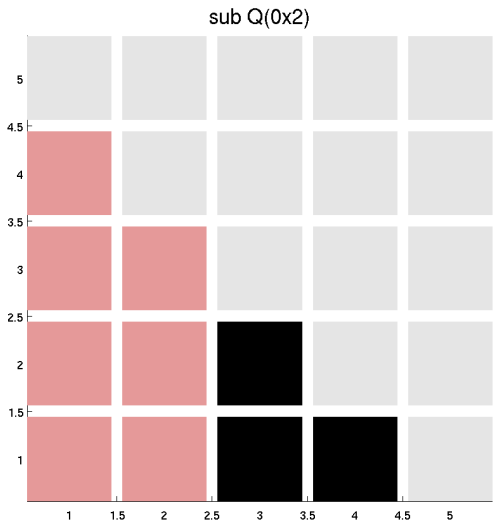


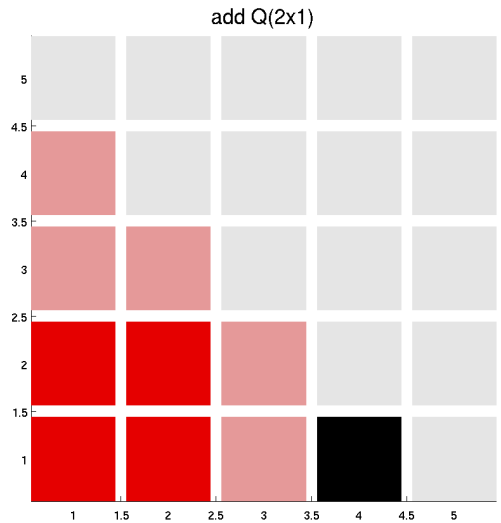




By "adding" the two quadrature rules, $Q = (0, 3) + (1, 2)$, it looks like we pick up 1, $y$ and $y^2$ twice.

# SPARSE: Cover the Squares

We can fix our overlap problem by subtracting the rule $(0, 2)$. Then we can move to the next column by adding $(2, 1)$:

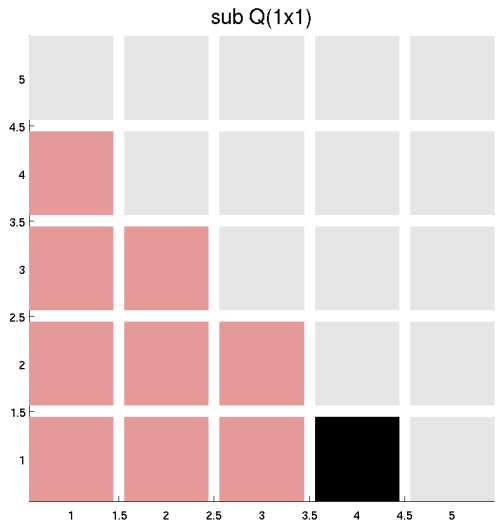


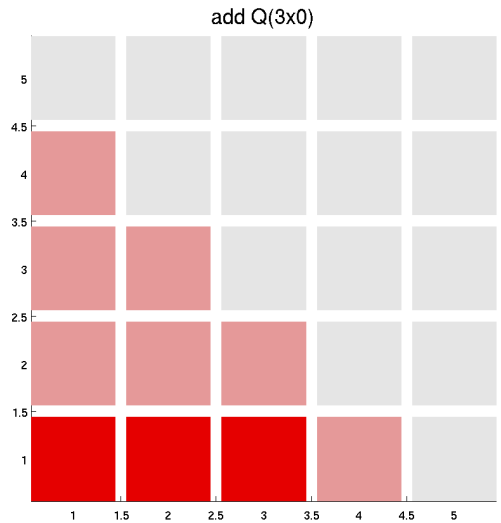




And again we have some overlap, or double counting (dark red), for this rule $Q = (0, 3) + (1, 2) - (0, 2) + (2, 1)$.

# SPARSE: Cover the Squares

Subtracting the rule $(1, 1)$ and move to the next column by adding $(3, 0)$:

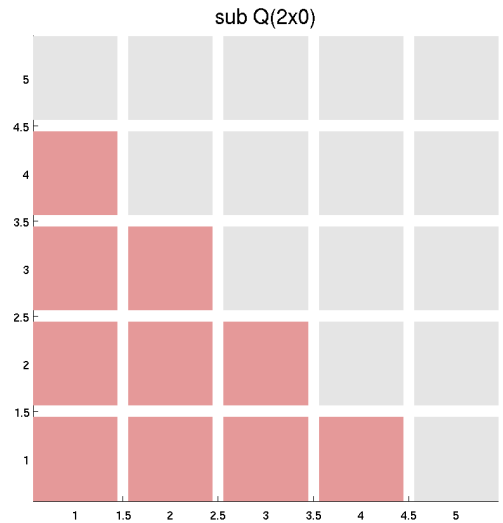


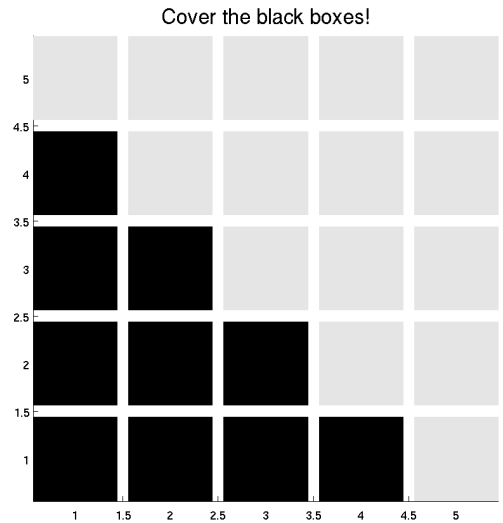




And again we have some overlap, or double counting (dark red), for this rule $Q = (0, 3) + (1, 2) - (0, 2) + (2, 1) - (1, 1) + (3, 0)$.

# SPARSE: Cover the Squares

Subtracting the rule $(2, 0)$, we cover the black squares.




There is no overlap for this rule

$Q = (0, 3) + (1, 2) - (0, 2) + (2, 1) - (1, 1) + (3, 0) - (2, 0).$

# SPARSE: Look at the Pattern

Rewrite this rule as:

```
+(0,3)
-(0,2) +(1,2)
       -(1,1) +(2,1)
              -(2,0) + (3,0)
```

and you see a geometric pattern, and the fact that, to get a rule of total precision 3, we added all rules of component precision 3, and subtracted all rules of component precision 2. In 2D, the usual Smolyak rules are all made this way:

```
0: +(0,0)
1: +(1,0) + (0,1)
   -(0,0)
2: +(2,0) + (1,1) + (0,2)
   -(1,0) - (0,1)
3: +(3,0) + (2,1) + (1,2) + (0,3)
   -(2,0) - (1,1) - (0,2)
```

# SPARSE: What is the Cost?

So that's great. We've shown that, (if you believe me) you can combine low order 2D product rules in a way that gets exactly the monomials you want in order to achieve a given total precision.

So it's **possible** to do this. But it seems like this must be far more costly than simply computing a single $(3,3)$ product rule, which gets the job done in one blow.

After all, the "cost" of the $(3,3)$ rule might be 16 points. The cost of the 3rd Smolyak rule looks like (4+6+6+4)+(3+4+3)=30 points, that is, I have to evaluate the function at every point specified by every rule that is part of the Smolyak rule.

*Here, I'm assuming that a* $(2,1)$ *rule, for instance, is the product of a 3 point rule and a 2 point rule, with a total of 6 points.*

# SPARSE: Nesting Reduces the Cost

But now, suppose we arranged our 1D rules so that they are perfectly nested. (We'll come back and modify this claim, but for now, let's believe "a little bit of inaccuracy"!)

In that case, there would be only 10 unique points used in the 7 rules that we combined to build our Smolyak rule. In the very unlikely event that our 1D rules used consecutive, evenly spaced points, then here is the "stencil" that we would have:

```
3 | (0,3) -----  -----  -----
2 | (0,2) (1,2)  -----  -----
1 | (0,1) (1,1)  (2,1)  -----
0 | (0,0) (1,0)  (2,0)  (3,0)
-----------------------------
  |  0     1      2      3
```

so our rule, involving the sum of 7 low order product rules, might use fewer points than the single (3,3) product rule, if we have nesting.

# SPARSE: Number of Nodes = Number of Monomials

Moreover, you can see that if we were able to use this perfectly nested sequence of rules, then the number of nodes (places where we evaluate the integrand) will be exactly equal to the number of monomials we want to integrate exactly...at least in this 2D example.

For instance, there are 10 monomials of total degree 3 or less, and our corresponding point pattern uses 10 abscissas or nodes or evaluation points.

This would stay true for higher degree, and for higher dimension.

*The plan we actually use is inspired by this idea, but modifies it for practical reasons!*

# SPARSE: What happens in 3D?

In 3D, you have to play "cover the black cubes". The pattern is similar, but now will generally involve combining three groups of product rules:

```
0: +(0,0,0)
1: +(1,0,0) + (0,1,0) + (0,0,1)
   -2*(0,0,0)
2: +(2,0,0) + (1,1,0)+(1,0,1)+(0,2,0)+(0,1,1)+(0,0,2)
   -2*(1,0,0) - 2*(0,1,0)-2(0,0,1)
   +(0,0,0)
3: +1 * rules that sum to 3
   -2 * rules that sum to 2
   +1 * rules that sum to 1
```

You may assume that the coefficients (1,-2,1) come from the third row of Pascal's triangle (with sign).

# SPARSE: Formal Definition

Believe it or not, the rule for creating the $L$-th Smolyak rule in dimension $M$ is as simple as that – but also as unfriendly-looking as the following formula, where $\vec{\ell}$ is the vector describing a product rule.

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

The product rules that sum to $L$ get multiplied by 1. The product rules that sum to $L-1$ get multiplied by $-(M-1)$, and so on.

# SPARSE: Computational Comments

Implementing Smolyak's formula given $L$ and $M$ requires:

- a loop on $|\vec{\ell}|$ from $L - M + 1$ to $L$;
- computing the value $(-1)^{L-|\vec{\ell}|}$;
- the combinatorial coefficient $\left( \begin{array}{c} M-1 \\ L-|\vec{\ell}| \end{array} \right)$
- the generation of every nonnegative integer $M$ vector $\vec{\ell}$ whose entries sum to $|\vec{\ell}|$;

Each of these steps raises a computational issue; let's take a few moments to be sure that everyone here could compute the correct list of product rules, with their signs and weights, for a given $\mathcal{A}(L, M)$.

# SPARSE: The loop index

The name of the loop index $|\vec{\ell}|$ is actually a symbol indicating that it is meant to control the sum of the elements of vectors we'll call $\vec{\ell}$.

Computationally, we need to give it a name, so let's call it lsum.

The formula, as quoted, allows **lsum** to run from $L - M + 1$ to $L$. However, we know that **lsum** is the sum of the entries in $\vec{\ell}$, which are nonnegative integers. So it will help to write, instead, that **lsum** runs from $\max(0, L - M + 1)$ to $L$.

This is why, in 3D, we don't begin combining three groups of rules until we get to $L = 2$. In general $M$-dimensional space, the 0'th rule is still just one group $(0, 0, ..., 0)$, the first rule involves two groups, and so on, until we reach $L = M - 1$.

# SPARSE: The power of minus 1

We rewrite the factor $(-1)^{L-|\vec{\ell}|}$ as $(-1)^{L-lsum}$.

Typically, $L$ is going to be a number less than 10, so we could be careless and compute the power of $-1$ by repeated multiplication:

$$(-1)^5 = (-1) \cdot (-1) \cdot (-1) \cdot (-1) \cdot (-1) = -1$$

But we could also note that the result is $+1$ or -1 depending on whether the exponent is even or odd, so that we have

$$(-1)^5 = (-1)^{\text{ mod (5,2)}}$$

While it won't save us much time in this case, it's good to realize that correct mathematics is not always efficient computation.

We could think of this as a function **value = mop ( n );**

# SPARSE: The Combinatorial Coefficient

The formula for $C(n,k)$, the combinatorial coefficient is:

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

For Smolyak's formula, it's possible we might have $M = 10$, 20, 30 or even 100. When $n$ is large, there are many bad ways to compute $C(n,k)$. If $k$ is a small number, we can rewrite $C(n,k)$ as:

$$C(n,k) = \frac{n \cdot (n-1) \ldots \cdot (n-k+1)}{k!}$$

We can even guarantee an integer result:

$$C(n,k) = n/1 \cdot (n-1)/2 \cdot (n-2)/3 \ldots \cdot (n-k+1)/k$$

where we evaluate from left to right, because the product of two consecutive integers is divisible by 2!, the product of three consecutive integers is divisible by 3!, and so on...

We could think of this as a function **value = choose ( n, k );**

# SPARSE: All possible vectors $\vec{\ell}$

To generate all vectors $\vec{\ell}$ whose entries sum up to **lsum**, for $M = 3$ and **lsum=4**, we have:

```
0,0,4  (start with everything on the right right)
0,1,3  rightmost nonzero increments left, rest go right.
0,2,2  rightmost nonzero increments left, rest go right.
0,3,1  rightmost nonzero increments left, rest go right.
0,4,0  rightmost nonzero increments left, rest go right.
1,0,3  rightmost nonzero increments left, rest go right.
1,1,2  rightmost nonzero increments left, rest go right.
1,2,1  rightmost nonzero increments left, rest go right.
1,3,0  rightmost nonzero increments left, rest go right.
2,0,2  rightmost nonzero increments left, rest go right.
2,1,1  rightmost nonzero increments left, rest go right.
2,2,0  rightmost nonzero increments left, rest go right.
3,0,1  rightmost nonzero increments left, rest go right.
3,1,0  rightmost nonzero increments left, rest go right.
4,0,0  ( stop when you can't move. )
```

# SPARSE: Exercises

**Exercise 30:** Write three procedures to compute the combinatorial function we called **choose(n,k)**:

- **choose1()** evaluates $n!$, $k!$, $n-k!$, then combines them;
- **choose2()** evaluates $n \cdot (n-1) \ldots \cdot (n-k+1)$, divides by $k!$;
- **choose3()** computes $n/1$, then multiplies by $(n-1)$ and then divides by 2, and so on, as described earlier;

Use these functions to compute $C(8,3)$, $C(10,2)$, $C(20,4)$, and $C(30,6)$. Comment on your results.

**Exercise 31:** Write a procedure which accepts as input an $M$-vector of nonnegative integers $\vec{\ell}$ whose entries sum up to **lsum**, and which returns the "next" vector. Test your function on the example on the previous slide, by calling your function with input vector $(0,0,4)$. Use the output from that call to call again and again until you receive as output the final vector $(4,0,0)$.

# SPARSE: Program Outline #1

Assume that $q(i,j,k)$ indicates the result of carrying out a 3D product rule constructed by $q(i,j,k) = q(i) \times q(j) \times q(k)$. Then a "simple" program to evaluate $\mathcal{A}(L,M)$, the Smolyak integral estimate in dimension $M = 3$ of level $L$, might look like:

```
s = 0.0;
for lsum = max ( 0, L - M + 1 ) to L
  c = mop ( L - lsum ) * choose ( M - 1, L - lsum );
  set (i,j,k)=(0,0,lsum);   first composition of lsum.
  begin loop
    s = s + c * q(i,j,k);   estimate integral with product rule.
    if (i,j,k)=(lsum,0,0), break from loop;
    generate next (i,j,k);
  end
end
```

# SPARSE: Program Outline #2

The previous program computes an integral, once, and does it somewhat inefficiently, since the product rules defined may include many common points.

We can try to be more efficient by working out all the points to be used, and gathering their weights to make a single quadrature rule in advance.

Here is how a user might go invoke the corresponding functions:

```
m = 3;
l = 4;
n = spgrid_size ( m, l );
x = spgrid_points ( m, l, n );
w = spgrid_weights ( m, l, n, x );

q = w' * f ( x );  <-- Weighted sum of integrand value
```

# SPARSE: Spinterp

**SPINTERP**, a MATLAB program, by Andreas Klimke, is a great way to explore the power of sparse grids.

```
x = spgrid ( l, m )
```

returns the points of a sparse grid of level **L** in dimension **M**.

To estimate an integral:

```
z = spvals ( @fun, m )  <-- fun.m is an M-file
q = spquad ( z )
```

**SPINTERP** can also interpolate and optimize using sparse grids.

---

http://www.ians.uni-stuttgart.de/spinterp/

# SPARSE: Spinterp

For example, to have **SPINTERP** estimate the integral of $f(x,y) = \sin(x) + \cos(y)$ over $[0,\pi]^2$, we could write:

```
f = @(x,y) sin(x)+ cos(y);  <-- a one-line function
m = 2;
r = [ 0.0, pi; 0.0, pi ];
z = spvals ( f, m, r );     <-- sets up the sparse grid;
q = spquad ( z );           <-- estimates the integral.
```

The **spvals()** function accepts an additional **options** argument that allows you to change from the default grid, the grid level, error tolerances, and so on.

# SPARSE: Spinterp Exercises

The following exercises introduce you to **SPINTERP**. For low dimensions, other software will probably work better - but this same program works in the same way for higher dimensions as well.

**Exercise 32:** Let $f(x, y) = \cos(\pi * x * \sin(\pi * y))$. Use **SPINTERP** to determine an interpolant to $f$ over the unit square $[0, 1]^2$. Choose 100 random points in the unit square, and report the average absolute difference between $f$ and its interpolant.

**Exercise 33:** Let $g(x, y) = \frac{1}{1-xy}$. Use **SPINTERP** to estimate the integral of $g$ over $[0, 1]^2$ and compare your result to the exact value $2\pi \ln(2)$.

**Exercise 34:** Let $h(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$. Use **SPINTERP** to estimate the location in $[-4, +4]^2$ of a point $(x, y)$ which minimizes $h(x, y)$.

# SPARSE: Other Software

**SPARSE_GRID_HW**, in C/C++/F90/MATLAB, by Heiss and Winschel, computes efficient quadrature rules for both the unit interval (Clenshaw-Curtis, Gauss-Legendre, Gauss-Patterson) and $(-\infty, +\infty)$ (Gauss-Hermite).

**SPARSE_GRID_CC**, in C/C++/F77/F90/MATLAB, includes functions to compute Clenshaw-Curtis sparse grids.

**SMOLPACK**, in C, by Knut Petras, estimates the integral of a function over a hypercube using sparse grids based on Clenshaw-Curtis or "delayed" Clenshaw-Curtis rules.

**SPARSE_GRID**, in Python, by Jochen Garcke, a basic library for Clenshaw-Curtis sparse grid calculations.

http://people.sc.fsu.edu/ jburkardt/m_src/sparse_grid_hw/sparse_grid_hw.html
http://people.sc.fsu.edu/ jburkardt/m_src/sparse_grid_cc/sparse_grid_cc.html
http://people.sc.fsu.edu/ jburkardt/c_src/smolpack/smolpack.html
http://people.sc.fsu.edu/ jburkardt/py_src/sparse_grid/sparse_grid.html

# Stochastic Integrals

- What Does an Integral Tell Us?
- The Probability Density Function
- Sampling from a Probability Density Function
- Approximating an Integral
- A Stochastic Fireball
- The Multidimensional Problem
- Approximating Multidimensional Integrals
- Sparse Grids
- **Clenshaw Curtis Sparse Grids**
- A Stochastic Tidal Wave

# CC: Implementation

Smolyak's procedure is actually very flexible.

When I described it to you, I suggested that we took a family of 1D rules $Q(0)$ using 1 point, $Q(1)$ using 2 points, and so on, and created product rules such that $Q(i,j,k) = Q(i) \times Q(j) \times Q(k)$, and combined these rules to make a Smolyak rule.

However, an important class of Smolyak rules is created using a 1D family that is defined somewhat differently. We require that $Q(0)$ has (at least) precision 1, $Q(1)$ has (at least) precision 3, $Q(2)$ precision 5, and in general, $Q(i)$ has precision at least $2i+1$.

And as our main example of such behavior, we are going to build a 1D family of Clenshaw-Curtis rules that are nested.

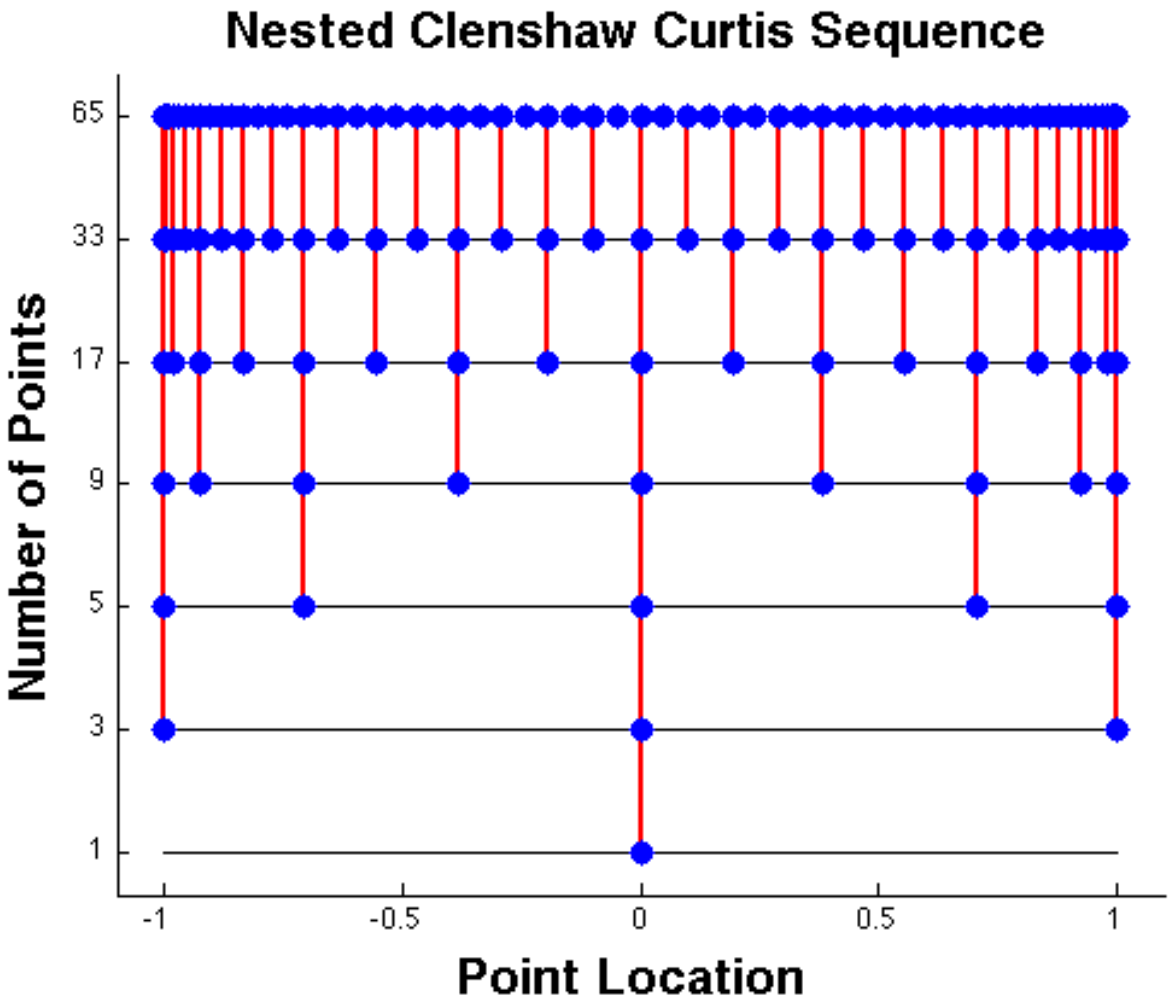
# CC: Implementation

Assume we have a sequence of 1D CC rules, of order **o**, index **i** and precision **p**, and we must select a subsequence indexed by **I**, that satisfy the precision requirement $P \geq 2 * I + 1$ **and** are nested:

| o | i | p | I | P | |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | |
| 2 | 1 | 1 | | | |
| 3 | 2 | 3 | 1 | 3 | |
| 4 | 3 | 3 | | | |
| 5 | 4 | 5 | 2 | 5 | |
| 6 | 5 | 5 | | | |
| 7 | 6 | 7 | | | (P ok, but not nested!) |
| 8 | 7 | 7 | | | |
| 9 | 8 | 9 | 3 | 7 | (P ok, and nested) |
| 10 | 9 | 9 | | | |
| ... | ... | ... | ... | ... | |
| 17 | 16 | 17 | 4 | 9 | |

# CC: Nested CC Subsequence

Here is how the nested Clenshaw-Curtis sequence is arranged:



# CC: Implementation

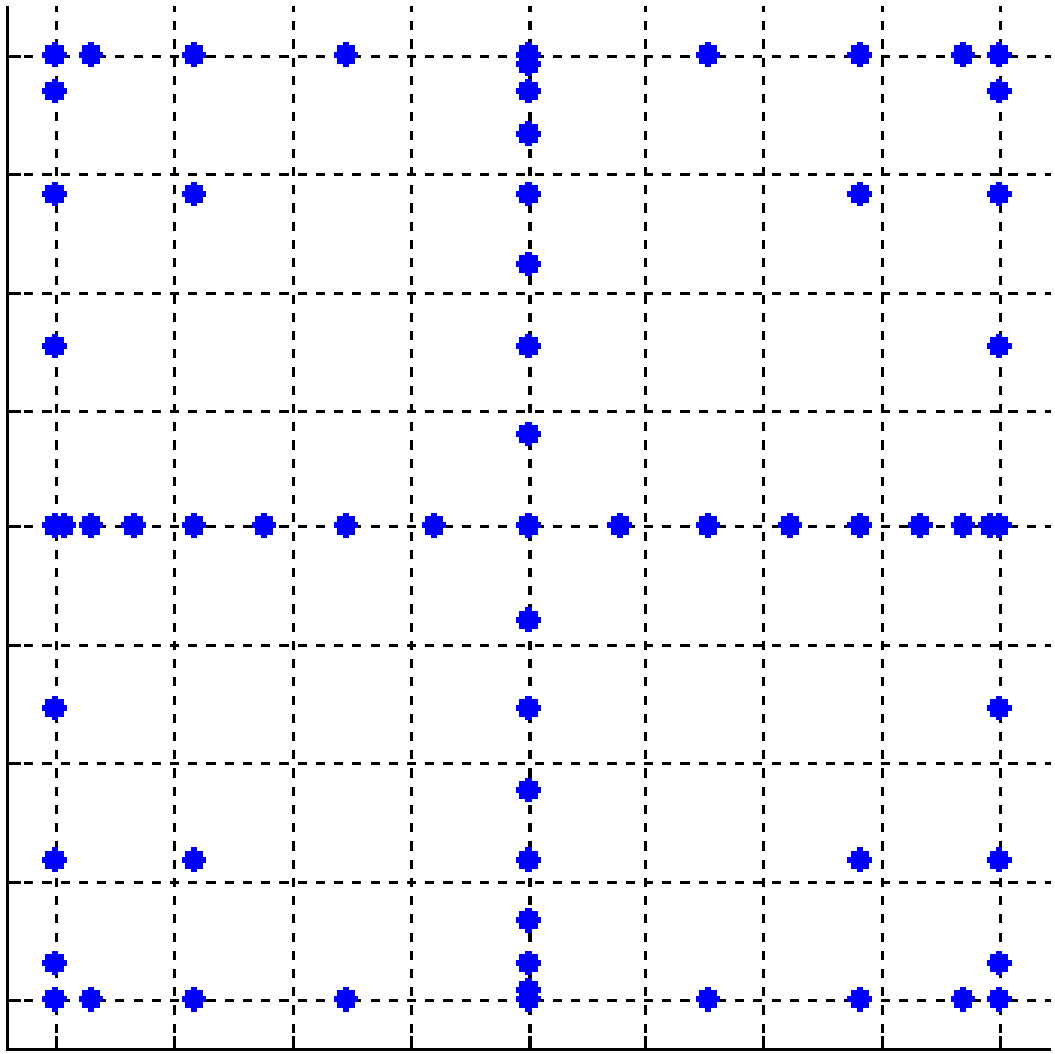
We have thus selected a subset of the Clenshaw Curtis rules that are nested and whose precision increases at least fast enough.

| 1D index $I$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| 1D order $O(I)$ | 1 | 3 | 5 | 9 | 17 | 33 | 65 | 129 | 255 |
| 1D precision $P(I)$ | 1 | 3 | 5 | 9 | 17 | 33 | 65 | 129 | 255 |
| needed precision | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 |

Now suppose we create a Smolyak rule using this CC sequence. In particular, what does $\mathcal{A}(L = 4, M = 2)$ look like?

# CC: $\mathcal{A}(L = 4, M = 2)$ Smolyak Grid

The 4th CC sparse grid in 2D uses 65 points.



# CC: $\mathcal{A}(L = 4, M = 2)$ Smolyak Grid

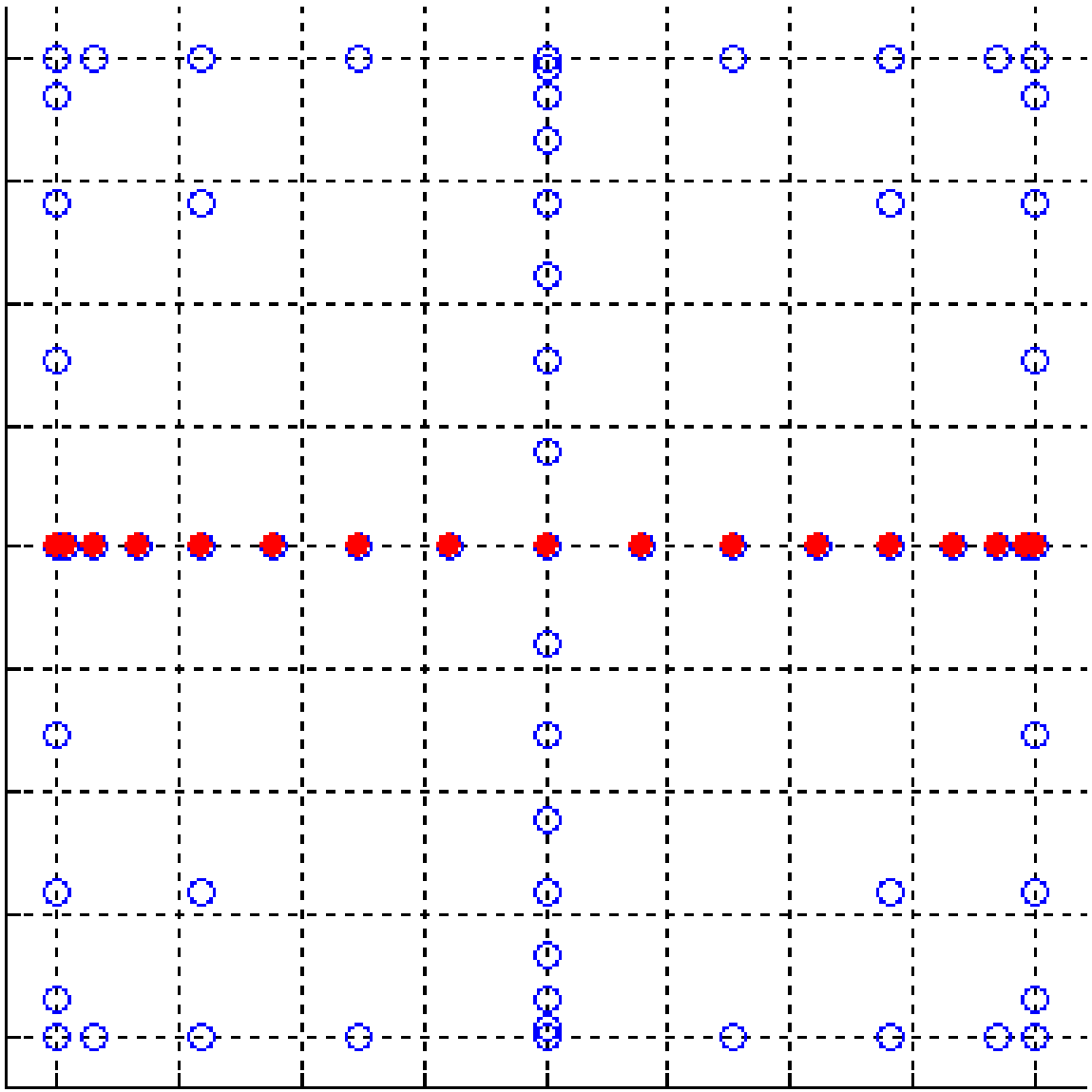
We can think of this Smolyak grid as the sum of level 4 grids minus level 3 grids where here I indicate the **index** of the component rules

```
  CC(0,4)
- CC(0,3) + CC(1,3)
          - CC(1,2) + CC(2,2)
                    - CC(2,1) + CC(3,1)
                              - CC(3,0) + CC(4,0)
```

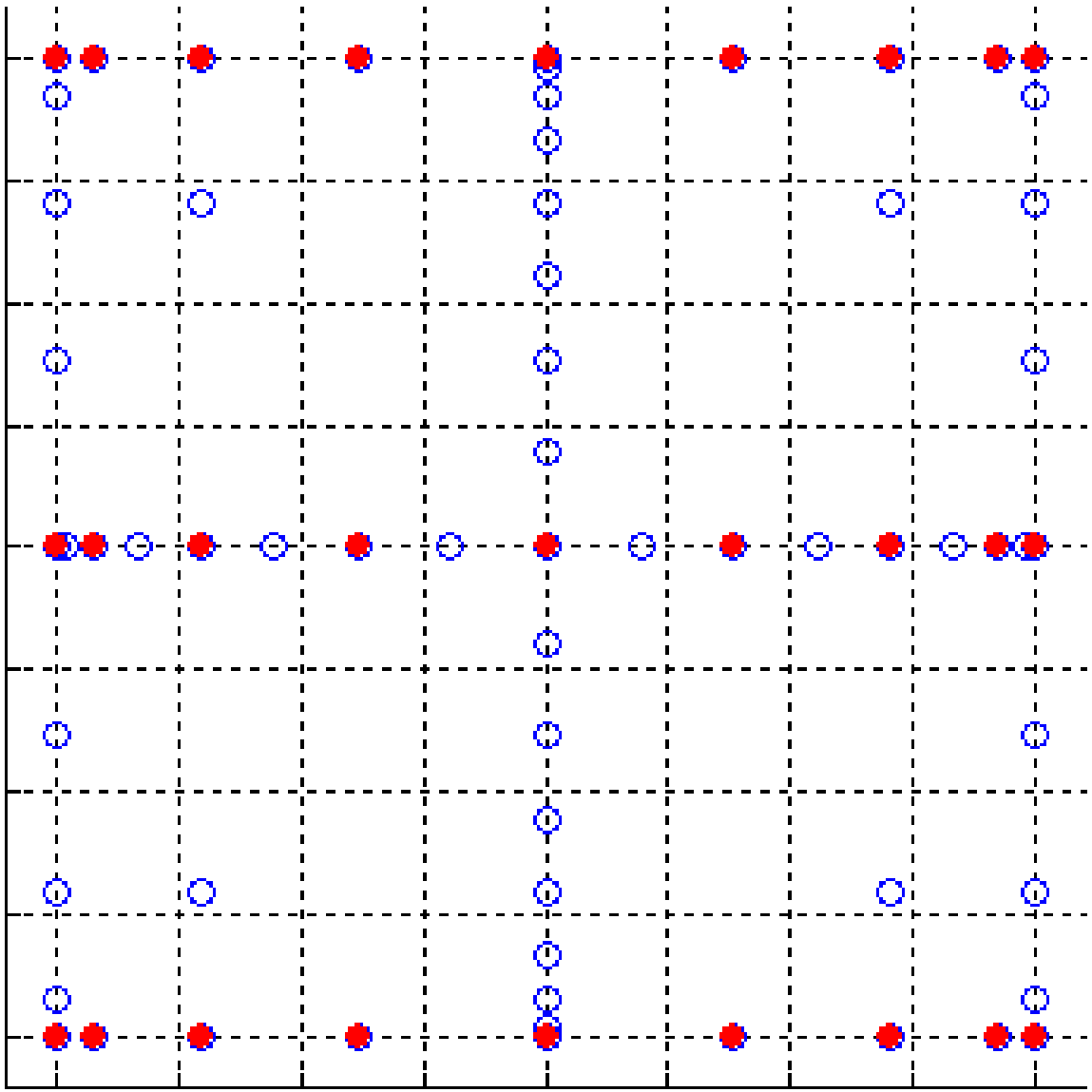
and here the **order**:

```
  CC(1,17)
- CC(1, 9) + CC(3,9)
           - CC(3,5) + CC(5,5)
                     - CC(5,3) + CC(9,3)
                               - CC(9,1) + CC(17,1)
```

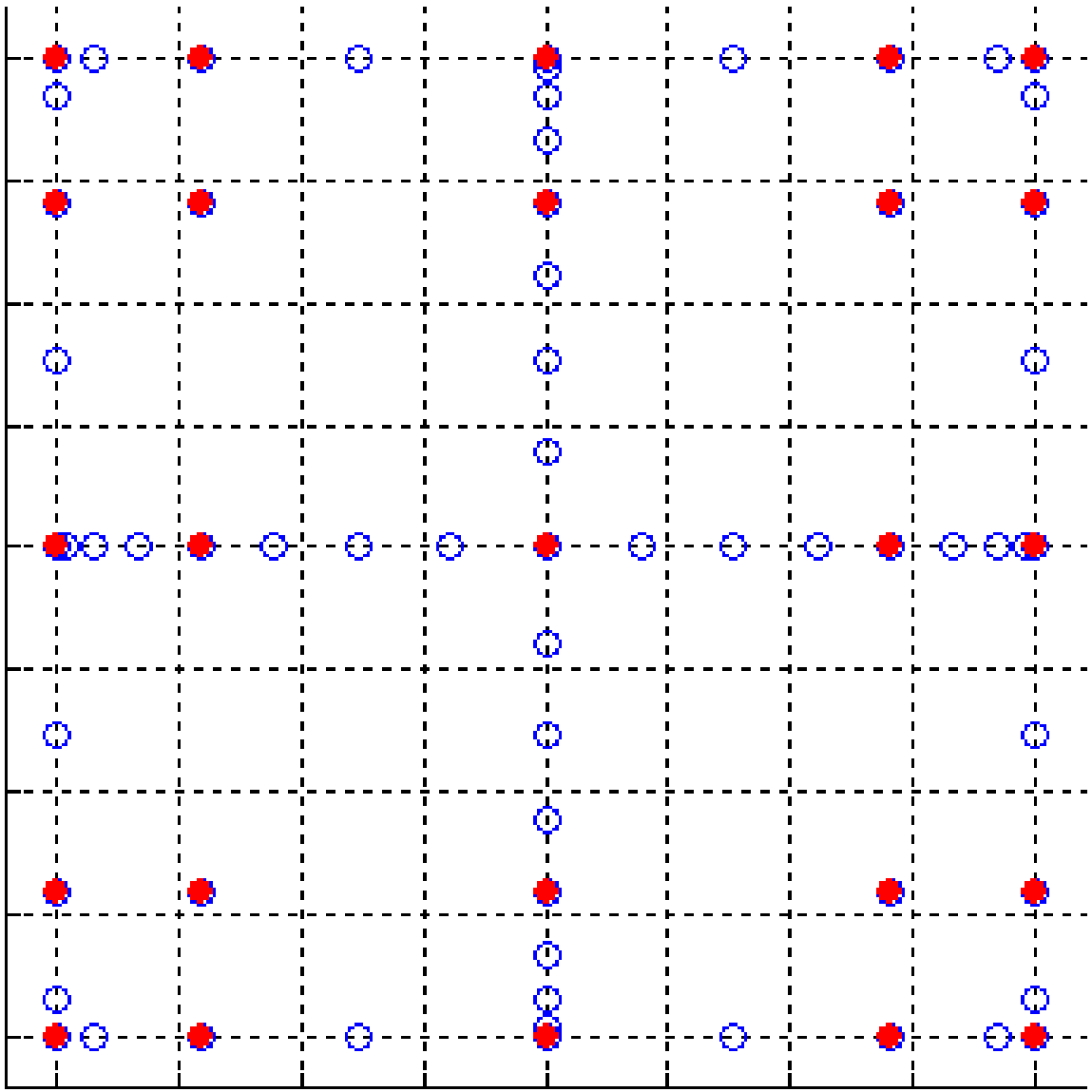
# CC: 2D Level4 17x1 component



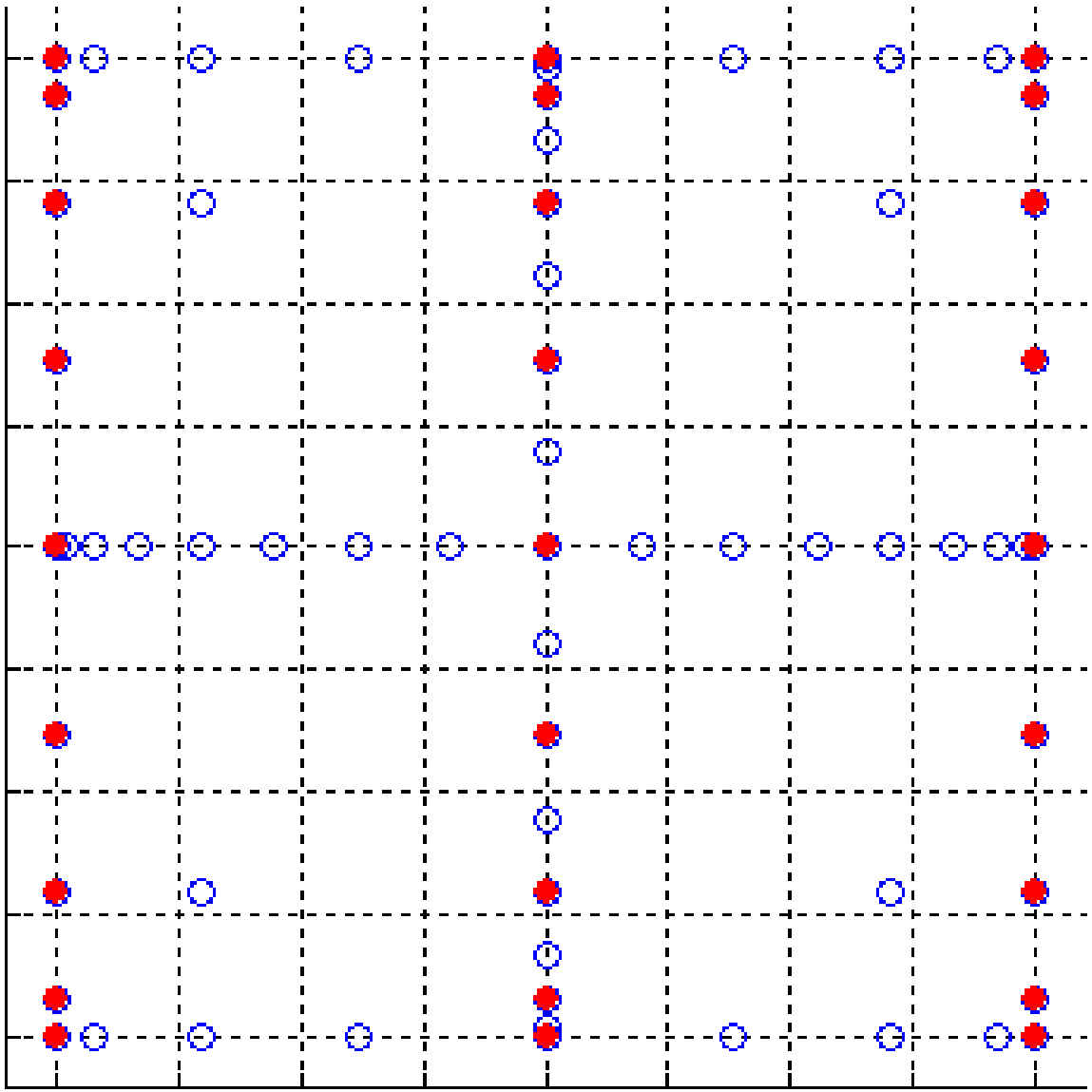
# CC: 2D Level4 9x3 component



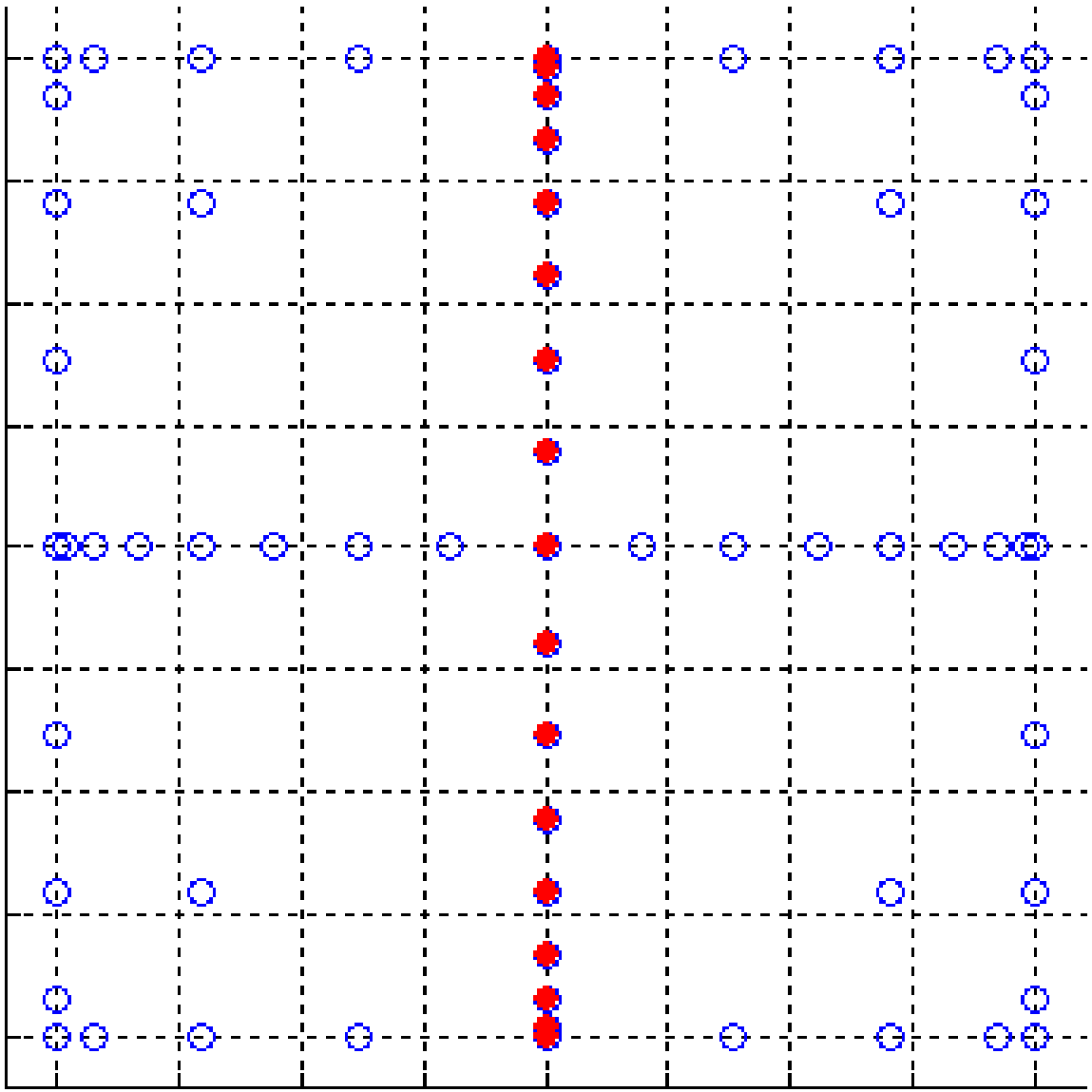
# CC: 2D Level4 5x5 component



# CC: 2D Level4 3x9 component



# CC: 2D Level4 1x17 component



# CC: Remarks

This level 4 quadrature rule in 2D also subtracts values at 4 product rules of slightly lower precision, but because of nesting, those points are already included in our plot.

Because we are using Clenshaw Curtis rules, and taking advantage of nesting, the order of the 1D rules begins to grow rapidly. This is not necessary, can be avoided, and ends up not hurting us much.

As long as the precision of the 1D rules satisfies $p(\ell) \geq 2\ell + 1$, the precision of the Smolyak rule of level $L$ will satisfy $p(L) = 2L + 1$. In other words, the 2D level 4 Smolyak rule based on the CC family we chose will have precision 9.

# CC: Combining the Rules

To understand how Smolyak rules are actually used, let's start with the simplest Smolyak rule that is actually still interesting, namely the 2D rule of level 1:

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

and let's imagine applying it to the problem of estimating the integral of $f(x,y)=e^{-(x-0.3)^2-(y-0.4)^2}$ over $(x,y)\in[0,1]\times[0.1]$.

The 1D CC quadrature rules we need, shifted to $[0,1]$ are simply the midpoint rule, and the 3 point NCC rule.

# CC: Combining the Rules

| | | | |
|---|---|---|---|
| y = 1.0 | | 0.67032 | |
| y = 0.5 | 0.90483 | 0.95122 | 0.60653 |
| y = 0.0 | | 0.81873 | |
| | x = 0.0 | x = 0.5 | x = 1.0 |

Writing $Z$, $M$ and $O$ for 0, 1/2 and 1, respectively, compute:

$$\begin{aligned} Q(1,0) &= 1/6F(ZM) + 2/3F(MM) + 1/6F(OM) = & 0.88604... \\ Q(0,1) &= 1/6F(MZ) + 2/3F(MM) + 1/6F(MO) = & 0.88232... \\ -Q(0,0) &= -1F(MM) = & -0.95122... \\ A(1,0) &= Q(1,0) + Q(0,1) - Q(0,0) = & 0.81714... \end{aligned}$$

The exact answer is 0.815937...

# CC: Combining the Rules

If you look at our calculation, we used F(MM) three times. In higher dimensions or higher levels, this happens even more often, because of nesting. So we can make our rule much more efficient by noticing all the rules in which a given point occurs, and combining the coefficients:

Writing $Z$, $M$ and $O$ for 0, 1/2 and 1, respectively, we had:

$$\begin{aligned} Q(1,0) &= 1/6F(ZM) + 2/3F(MM) + 1/6F(OM) \\ Q(0,1) &= 1/6F(MZ) + 2/3F(MM) + 1/6F(MO) \\ -Q(0,0) &= -1F(MM) \end{aligned}$$

and now we write

$$A(1,2) = 1/3F(MM) + 1/6F(ZM) + 1/6F(OM) + 1/6F(MZ) + 1/6F(MO)$$

but now this is just a 2D quadrature rule, with 5 points and their weights!

# CC: An Efficient Quadrature Rule

So, as long as we understand that a Smolyak sparse grid is built out of low level product rules, trying to match a single high level product rule, and we rearrange the pieces of the rules, the result is just a quadrature rule that tells us where to evaluate a function, and how to weight the values to estimate the integral.

Thus, to use a sparse grid, we need to compute the points $x$ and weights $w$, shift and scale them, if necessary, from their definition interval to the region $\Omega$ (which should be a product of intervals), and then just compute:

$$I(f,\Omega) \approx Q(f,\Omega) = \sum_{i=1}^{n} w_i \, f(x_i)$$

# CC: Point Growth Table

Here is the size, in function evaluations, for sparse grids of dimension $M$, level $L$, and precision $P$:

| M: | 5 | 10 | 15 | 20 | 25 |
|---|---|---|---|---|---|
| L/P | | | | | |
| 0/1 | 1 | 1 | 1 | 1 | 1 |
| 1/3 | 11 | 21 | 31 | 41 | 51 |
| 2/5 | 61 | 221 | 481 | 841 | 1,301 |
| 3/7 | 241 | 1,581 | 5,021 | 11,561 | 22,201 |
| 4/9 | 801 | 8,801 | 40,001 | 120,401 | 286,001 |
| 5/11 | 2,433 | 41,265 | 261,497 | 1,018,129 | 2,976,161 |
| 6/13 | 6,993 | 171,425 | 1,471,297 | 7,314,609 | 26,139,361 |
| 7/15 | 19,313 | 652,065 | 7,367,867 | 46,106,289 | 199,876,961 |
| 8/17 | 51,713 | 2,320,385 | 33,647,617 | 261,163,009 | 1,361,884,161 |

# CC: Point Growth Table

Compare dimension $M = 5$ sparse grids and product rules:

| M=5: | Sparse | Product |
|---|---|---|
| P | | |
| 1 | 1 | 1 |
| 3 | 11 | 243 |
| 5 | 61 | 3,125 |
| 7 | 241 | 16,807 |
| 9 | 801 | 59,049 |
| 11 | 2,433 | 161,051 |
| 13 | 6,993 | 371,293 |
| 15 | 19,313 | 759,375 |
| 17 | 51,713 | 1,419,857 |

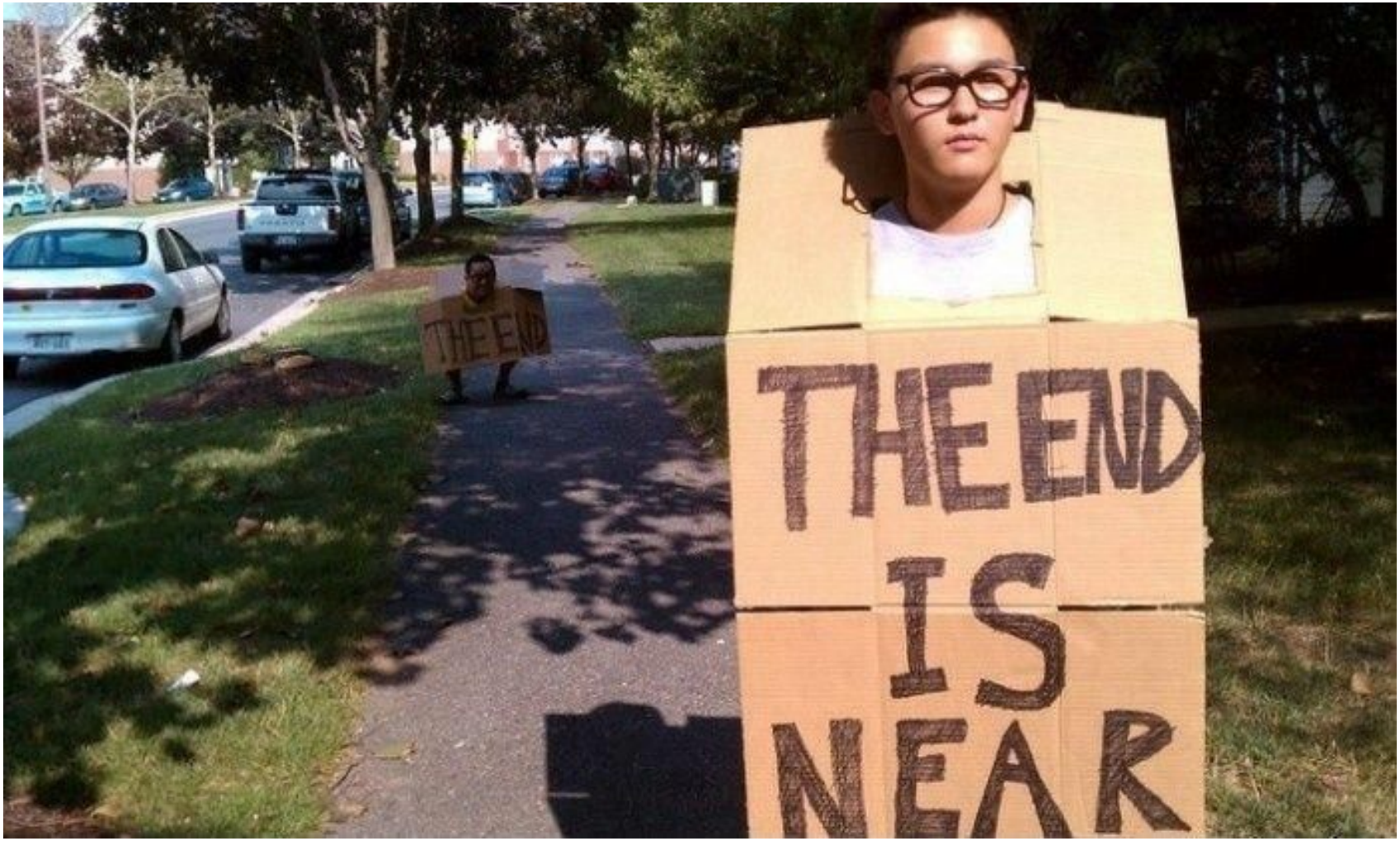
# CC: Point Growth Table

In dimension $M = 10$, the difference starts to become enormous:

| M=10: | Sparse | Product |
|---|---|---|
| P | | |
| 1 | 1 | 1 |
| 3 | 21 | 59,049 |
| 5 | 221 | 9,765,625 |
| 7 | 1,581 | 252,475,249 |
| 9 | 8,801 | — |
| 11 | 41,265 | — |
| 13 | 171,425 | — |
| 15 | 652,065 | — |
| 17 | 2,320,385 | — |

You can see that, while product rules quickly become useless, the number of function evaluations needed for a sparse grid rule are still reasonable for a precision of 17!

# End Part 5, Begin Part 6



# Approximating Integrals for Stochastic Problems

John Burkardt
Department of Scientific Computing
Florida State University

.....

ISC 5936-01:
Numerical Methods for Stochastic Differential Equations
http://people.sc.fsu.edu/~jburkardt/presentations/...
stochastic_integrals.pdf

.....

Revised: 20 March 2013

26/28 February, 5/7/19/21 March 2013

# CC: The Need for Smoothness

The simple product rule procedure that we have presented here assumes that we get more accuracy as we increase the polynomial precision. In turn, this assumes that the integrand function is "sufficiently smooth", or that is has bounded derivatives for as high as we need to apply the Taylor series error estimate.

It's easy to create integrands for which this is not true (the absolute value function, any discontinuous function, a piecewise function.) In such a case, we could still create a sparse grid, but we would need to use 1D rules that can handle the rough spots - for instance, a piecewise linear integration rule might work.

But if our integrand is not smooth, and we apply a simply sparse grid approach using polynomial interpolation, the results will generally go bad quickly!

# CC: The Need for Smoothness!

Let $f(x)$ be the characteristic function of the unit ball in 6D:

| N | 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

# CC: 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 in 6D.

| N | 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 |
| $\infty$ | 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.

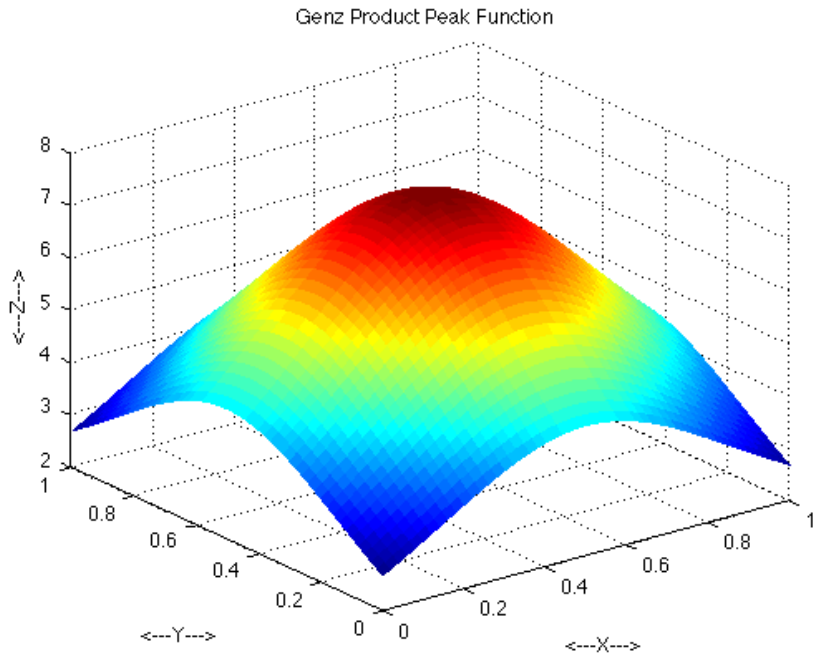
---

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

# CC: Genz Product Peak Test in 6D

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:

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

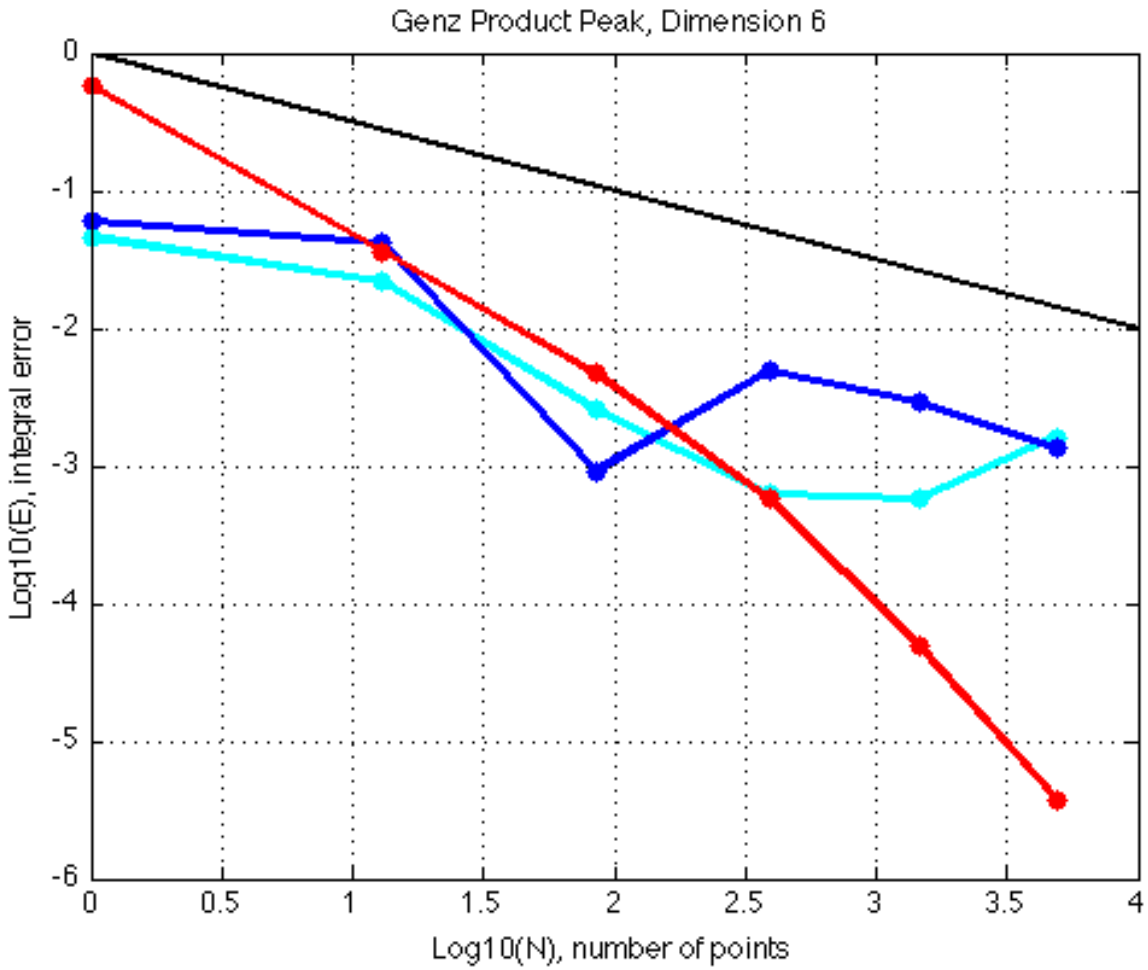


# CC: Genz Product Peak Test in 6D

Red: Sparse grid estimate

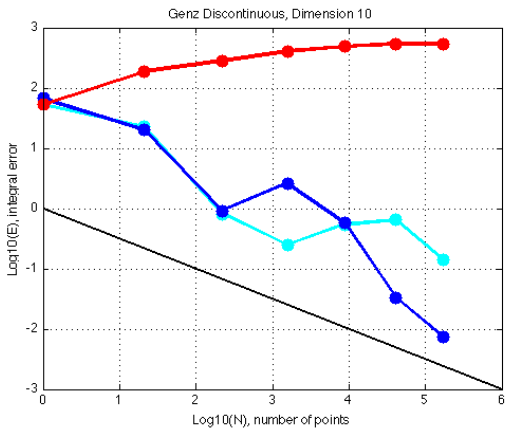
Blue & Cyan: MC estimates

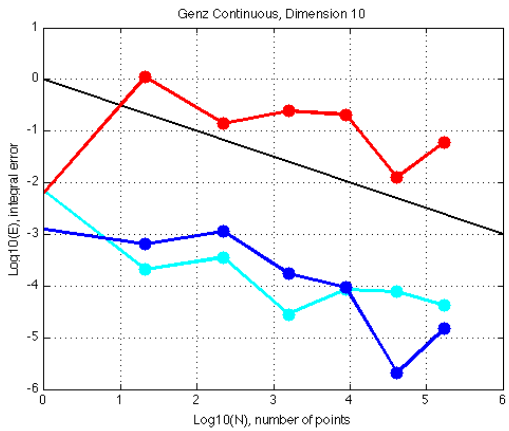
Black: Expected MC Rate of Decrease

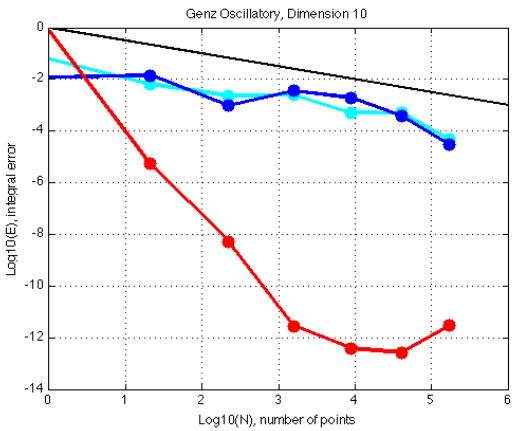


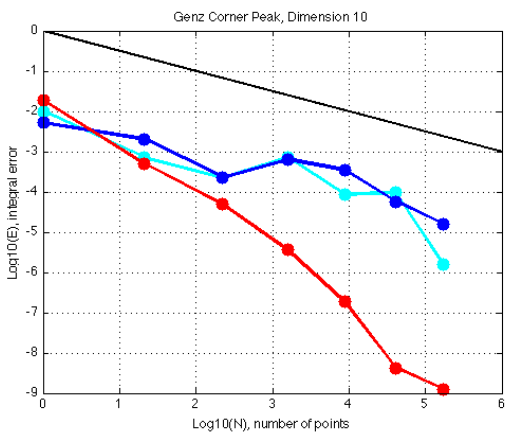
# CC: Genz Tests in 10D

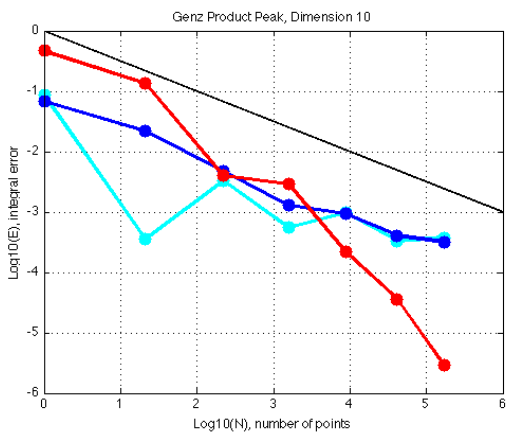
Discontinuous, Continuous, Oscillatory
Corner Peak, Product Peak, Gaussian
(sparse grid estimate in red)

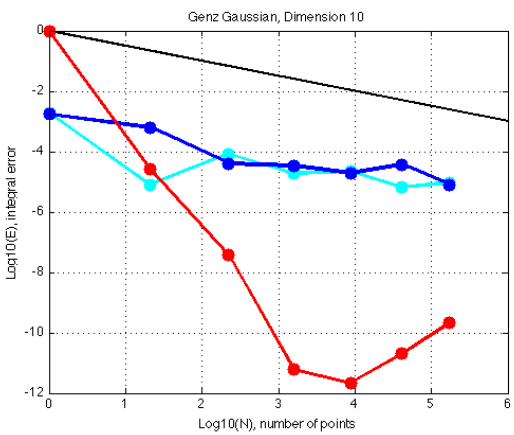


# CC: The Poisson Equation

Let's consider a Poisson equation with a stochastic diffusion coefficient:

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

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 $\omega$ according to $pr(\omega)$, solve the Poisson equation for $u$, and average.

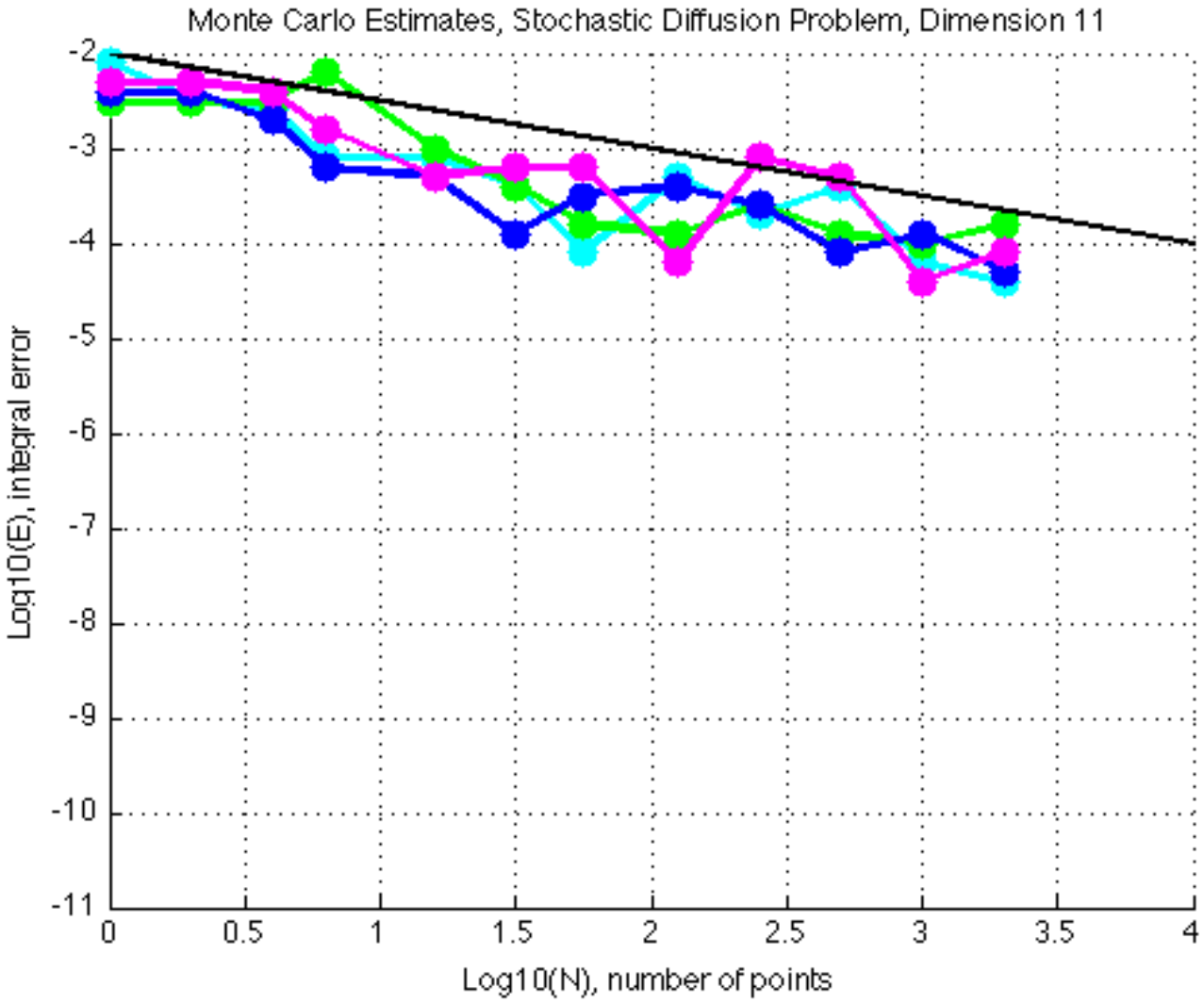
*Sparse grid*: choose a level, defining a grid of $\omega$ values, solve the Poisson equation for $u$, multiply by the probability, and take a weighted average.

---

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.

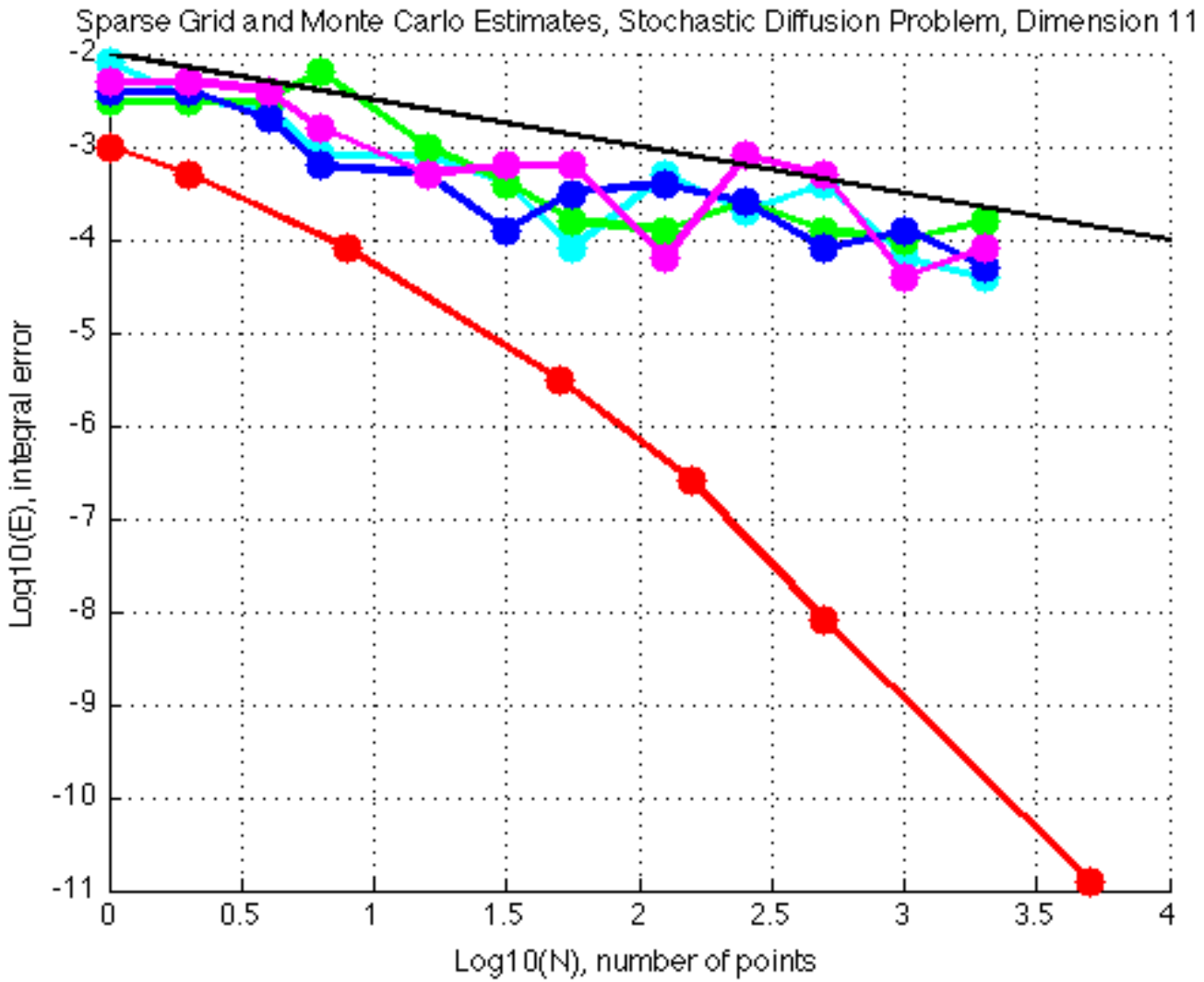
# CC: Four Monte Carlo Estimates

The black line is the Monte Carlo trend.



# CC: Sparse Grid Versus Monte Carlo

The sparse grid estimates converge rapidly.



# CC: Remarks

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

For this reason, a sparse grid can sample the solution for a small set of perturbations $\omega$ 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.

# Stochastic Integrals

- What Does an Integral Tell Us?
- The Probability Density Function
- Sampling from a Probability Density Function
- Approximating an Integral
- A Stochastic Fireball
- The Multidimensional Problem
- Approximating Multidimensional Integrals
- Sparse Grids
- Clenshaw Curtis Sparse Grids
- **A Stochastic Tidal Wave**

# TIDE: Burger's Equation

The Burgers equation, occasionally called *the poor man's Navier Stokes equation* has some interesting features:

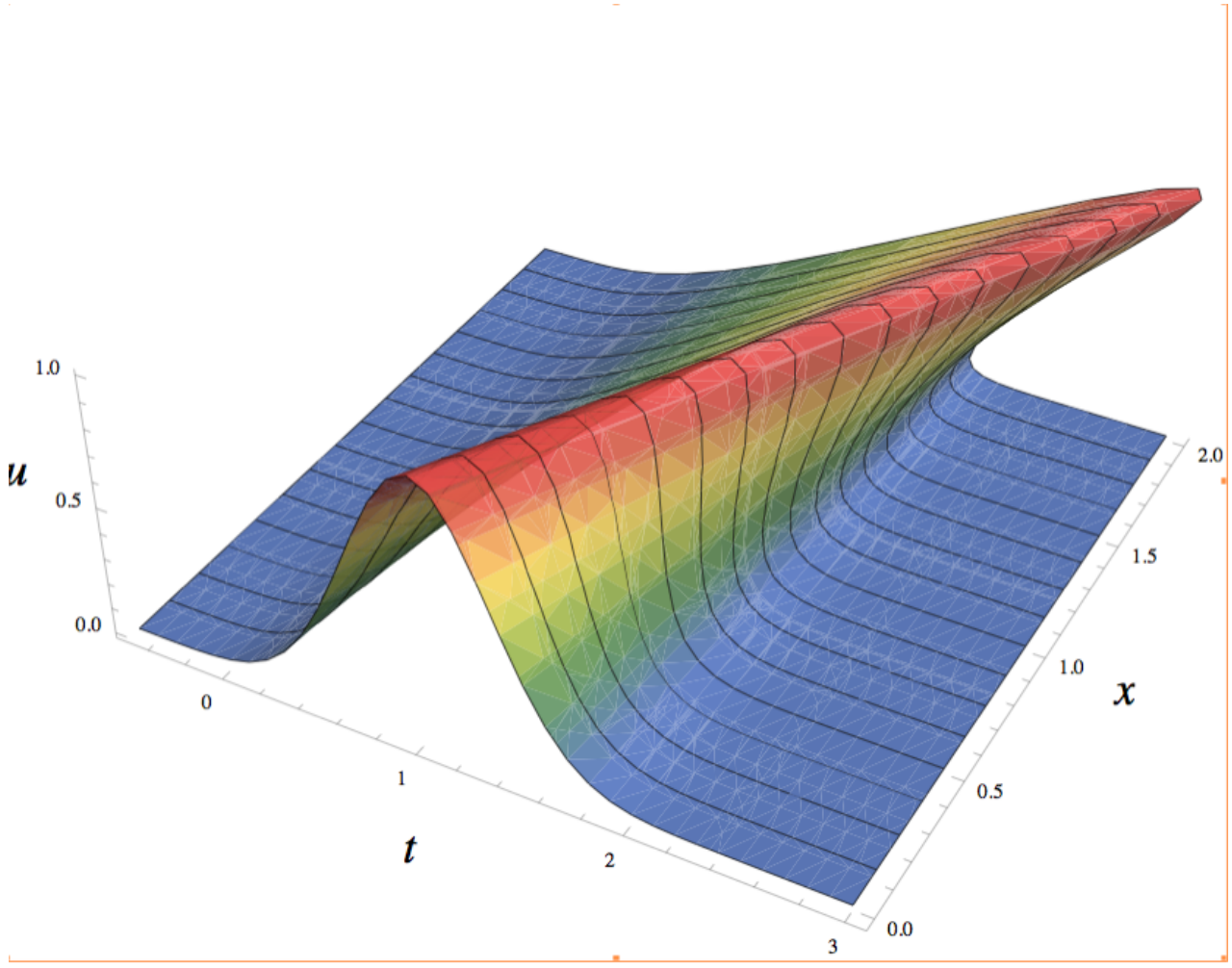
- includes a nonlinear term that can generate shocks and discontinuities;
- includes a smoothing term multiplied by a viscosity;
- definable as steady or time-dependent, viscid or inviscid.

The equation has the advantages that:

- it can be formulated in one spatial dimension $x$;
- it has only a single state variable $u(x)$ or $u(x,t)$.
- its simplicity makes it easy to define, solve, analyze, and plot.

# TIDE: Shock Waves for Inviscid Case

Here is a computational (and nonphysical!) solution of an inviscid Burgers problem, in which the peak velocity overtakes the rest of the wave.



# TIDE: Steady Viscous Burgers Equation

The steady viscous Burgers equation seeks a function $u(x)$ defined over an interval $[a, b]$, satisfying

$$u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

for which we might specify the Dirichlet boundary conditions

$$u(a) = \alpha; \quad u(b) = \beta.$$

# TIDE: Conservation Form

For technical reasons, it is preferable to rewrite the equation from its advection form to the conservation form:

$$\frac{1}{2}\frac{\partial u^2}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$

While this doesn't change the mathematics at all, it does suggest a different discretization scheme.

Here, $0 < \nu$ is the viscosity. A high viscosity corresponds to a sticky fluid, suppressing shocks and discontinuities. As $\nu$ decreases, the fluid can support steep gradients. A discretized solution technique will need greater resolution to capture the behavior.

# TIDE: Problem Parameter Values

Typical problem data might be:

$$a = -1,\ \alpha = +1,\ b = +1,\ \beta = -1,\ \nu = 0.1$$

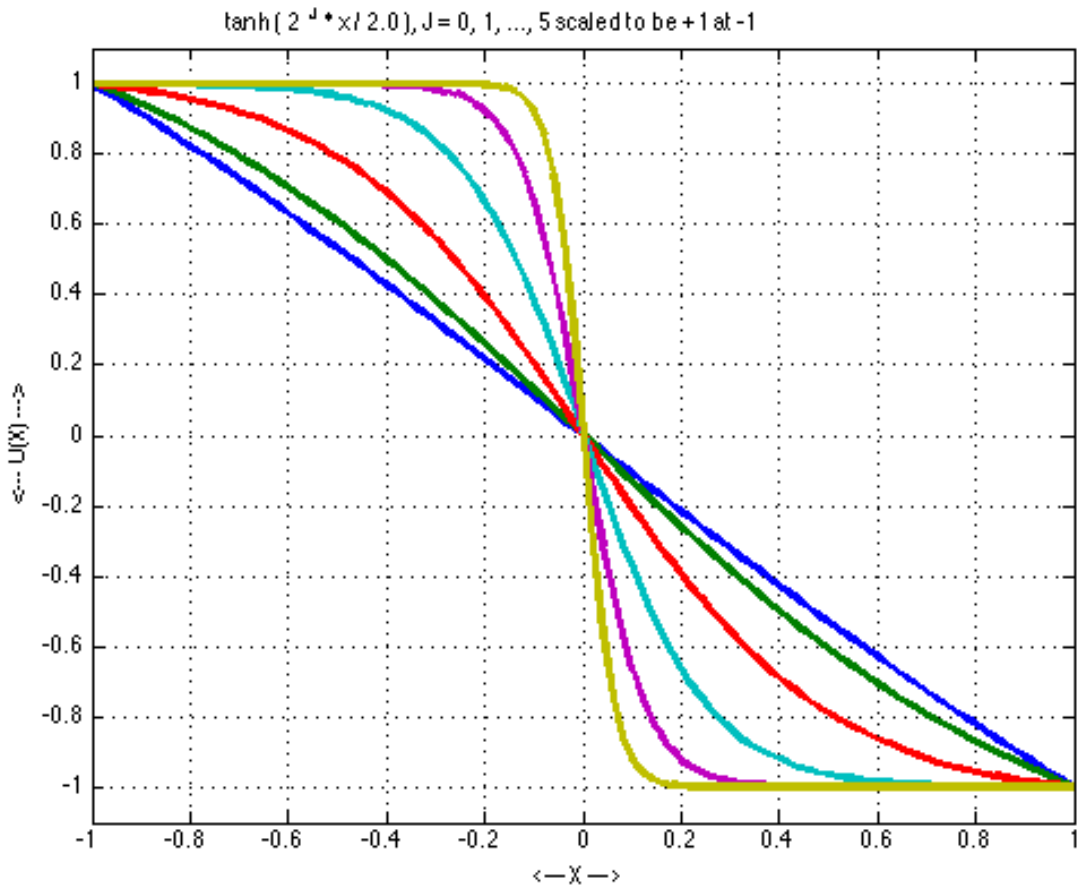
The specification of the values of these input parameters completes the definition of the analytic problem, and allows us to regard the solution $u(x)$ as a function of the parameters.

We will be interested in the relative importance of the influence of each parameter on the solution, and the effect of uncertainty in a parameter on the solution or on derived quantities of interest.

We'll use the values specified above as our "base data", and concentrate on solutions associated with relatively small perturbations of this data.

# TIDE: Solution Family For Varying Viscosity

For the given symmetric boundary conditions, the analytic solution depends on the viscosity, and has the shape of scaled **tanh(x)** function. Here is the kind of behavior we can expect from solutions to the exact equation for a variety of viscosity values.



# TIDE: Discretized Problem

A simple discretized version of the Burgers equation might use $m+1$ equally spaced nodes with spacing $dx = \frac{b-a}{m}$, with typical coordinate $x_i$, and discretized solution value $u_i$.

Since this is a nonlinear problem, we can construct system of equations $\vec{f}(\vec{u}) = 0$ that must be satisfied by the discrete solution; we can apply Newton's method to seek a solution.

At the first and last nodes, we impose the boundary conditions. At the interior nodes, we require the discrete solution to satisfy a discretized version of the Burgers equation.

# TIDE: Discretized Burgers Equation

Using $m+1$ evenly spaced points $x_i$, our discretized system is:

$$\begin{aligned} f_1 &= u_1 - \alpha \\ f_i &= \frac{1}{2}\frac{u_{i+1}^2 - u_{i-1}^2}{2dx} - \nu\frac{u_{i+1} - 2u_i + u_{i-1}}{dx^2}, \quad i = 2, \ldots, m \\ f_{m+1} &= u_{m+1} - \beta \end{aligned}$$

It is easy to write down the associated Jacobian matrix, and if we use as a starting point the linear interpolant to the two known boundary values, we can carry out the Newton procedure to obtain a solution.

# TIDE: Quantity of Interest

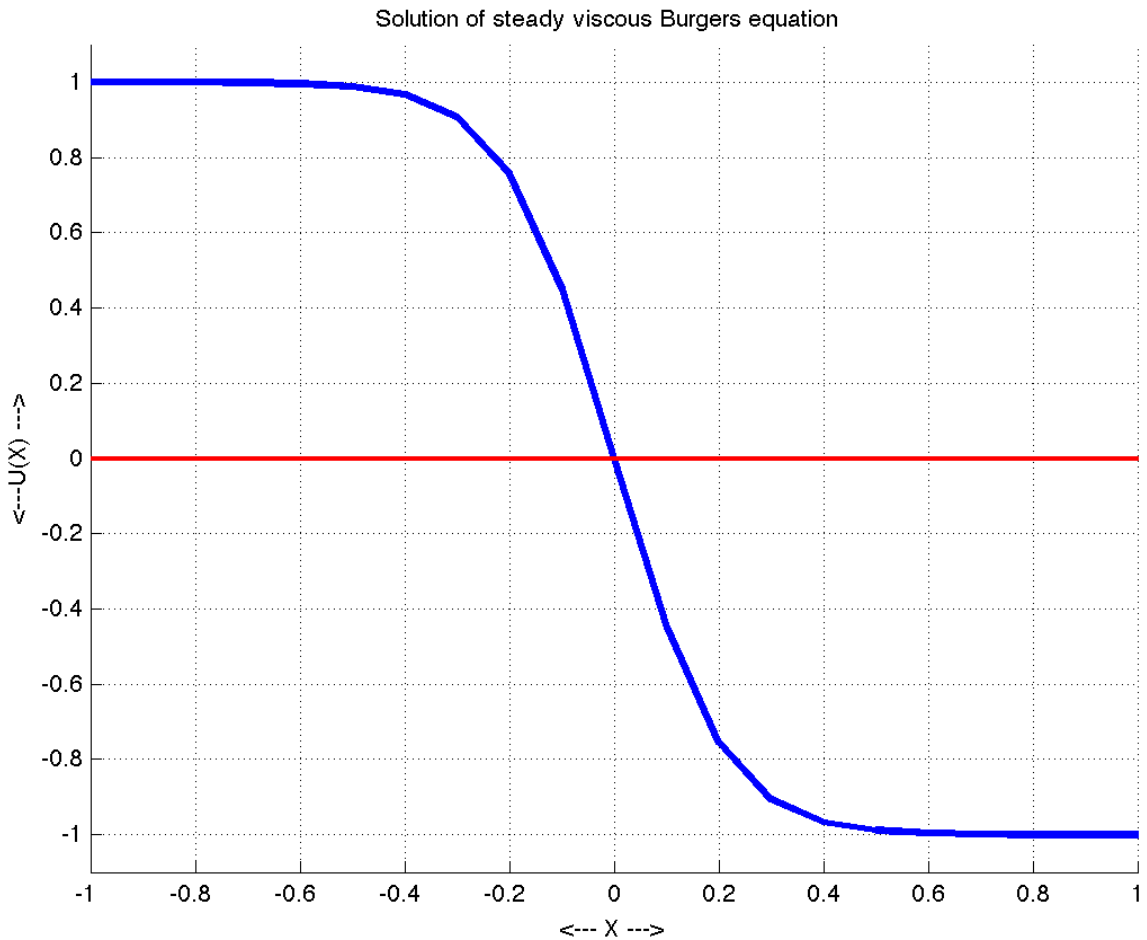
It's natural to focus on the solution function $u(x)$ as the most important object in the computation, but for many computations, one or more quantities of interest, derived from $u(x)$, might be the actual goal of the computation.

Such quantities can include the integral of the solution, the maximum deviation from some prescribed value, the lift or drag of an airfoil, the breaking point of a beam, or the total expenditure of fuel.

For this study, suppose the quantity of interest is the point $x_0$ where the solution changes sign. Since our solution is discretized, we'll use its linear interpolant to define $x_0$.

# TIDE: Base solution

Here is our computed "base" solution for the parameter values $a = -1, \alpha = +1, b = +1, \beta = -1, \nu = 0.1$.
The value of the quantity of interest is $x_0 = 0$.



# TIDE: Initial Sensitivity Analysis

We have our solution, but it depends on the parameter values we chose. If we imagine there are errors or uncertainties in this data, our computed solution will differ from the actual one.

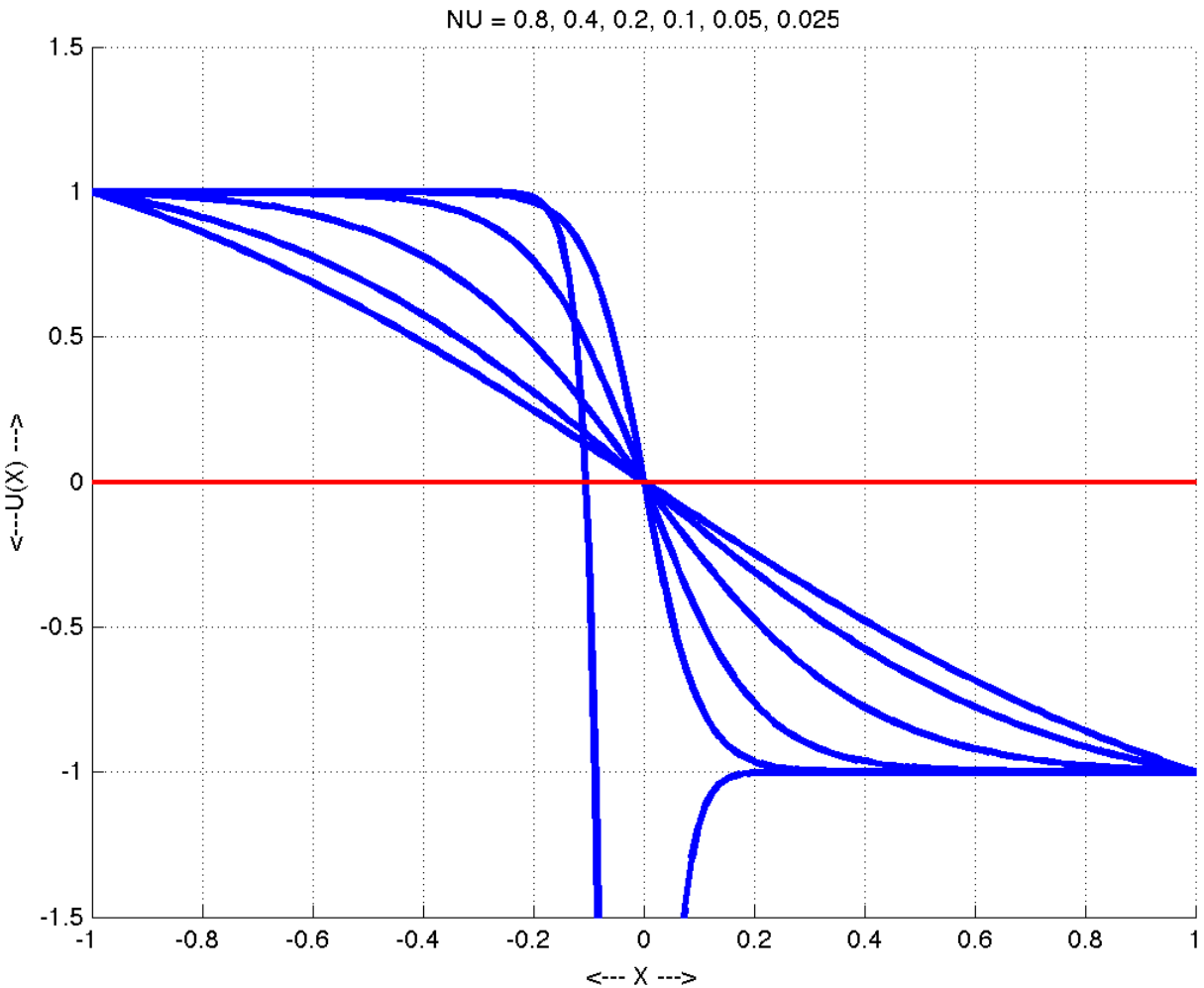
Can this effect be large for the types of errors we expect? Can we describe the types of errors we expect? Are some parameter errors more serious than others?

We can start with a crude sensitivity analysis, slightly modifying one base parameter at a time, and recomputing the solution. This suggests the strength of the dependence of $u$ on each parameter, and thus the relative importance of each parameter.

This will suggest where our uncertainty investigation should focus.

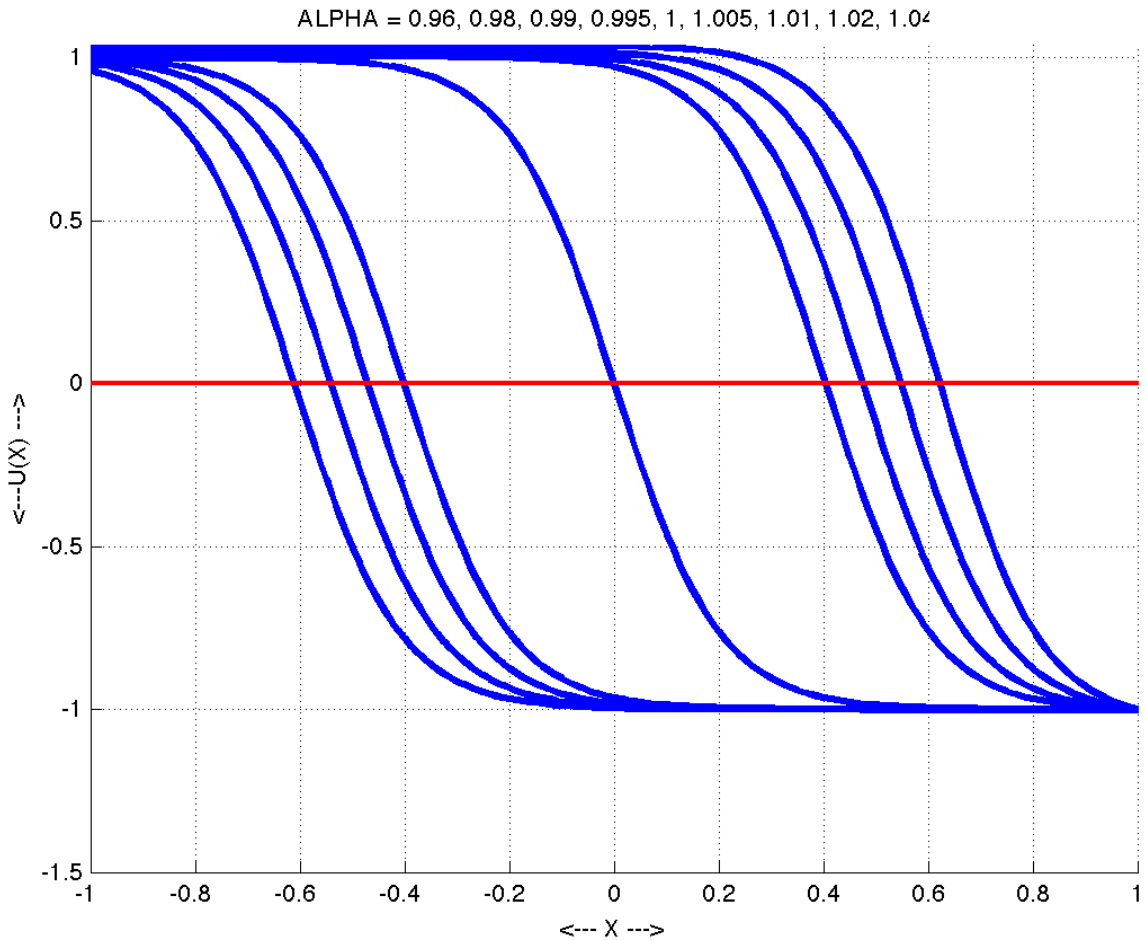
# TIDE: Sensitivity to $\nu$

Varying the viscosity $\nu$ affects the flow . . . but not $x_0$!
The peculiar result for $\nu = 0.025$ is because of nonconvergence.



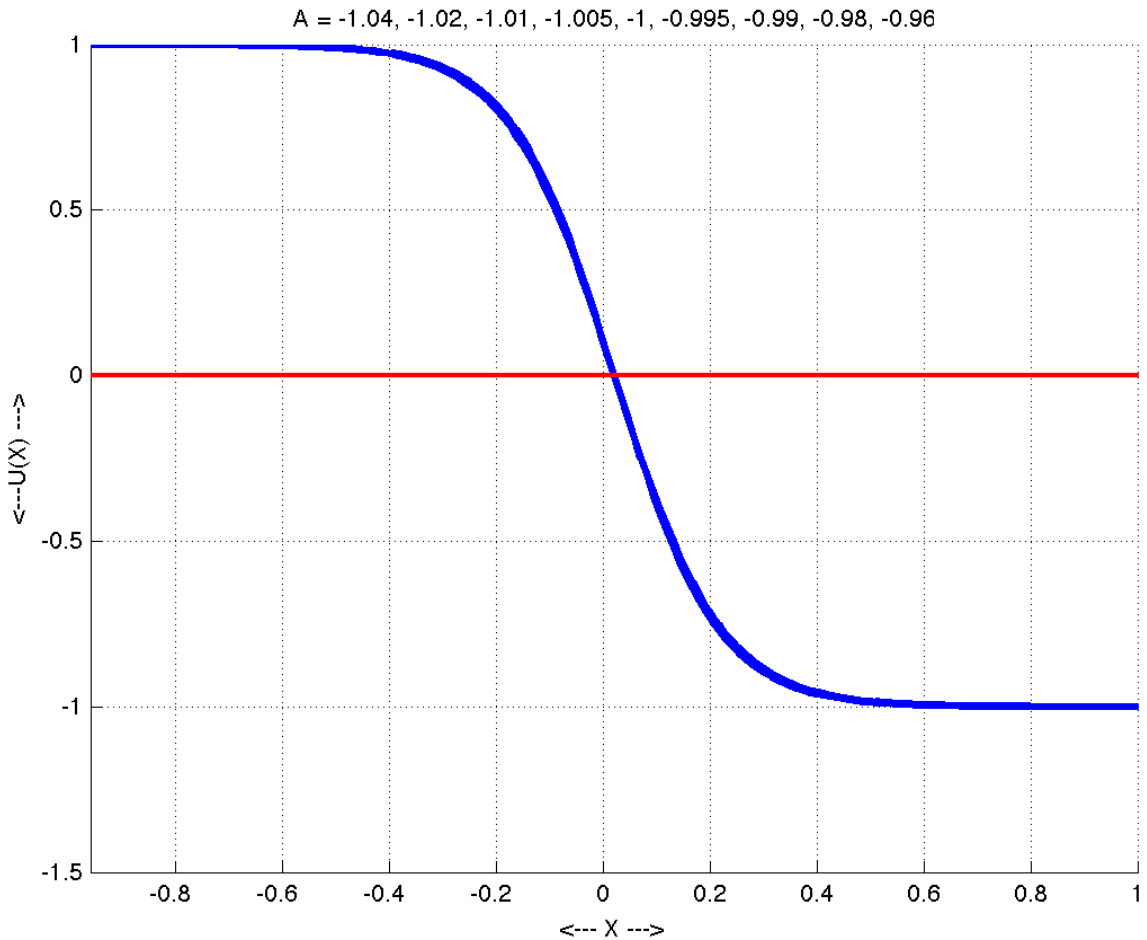
# TIDE: Sensitivity to $\alpha$

$\alpha$ determines the solution at the left endpoint $a$. The computed solution $u$ is surprisingly sensitive to this quantity. Changing $\alpha$ from 1 to 1.005 is enough to make a startling jump in $x_0$.



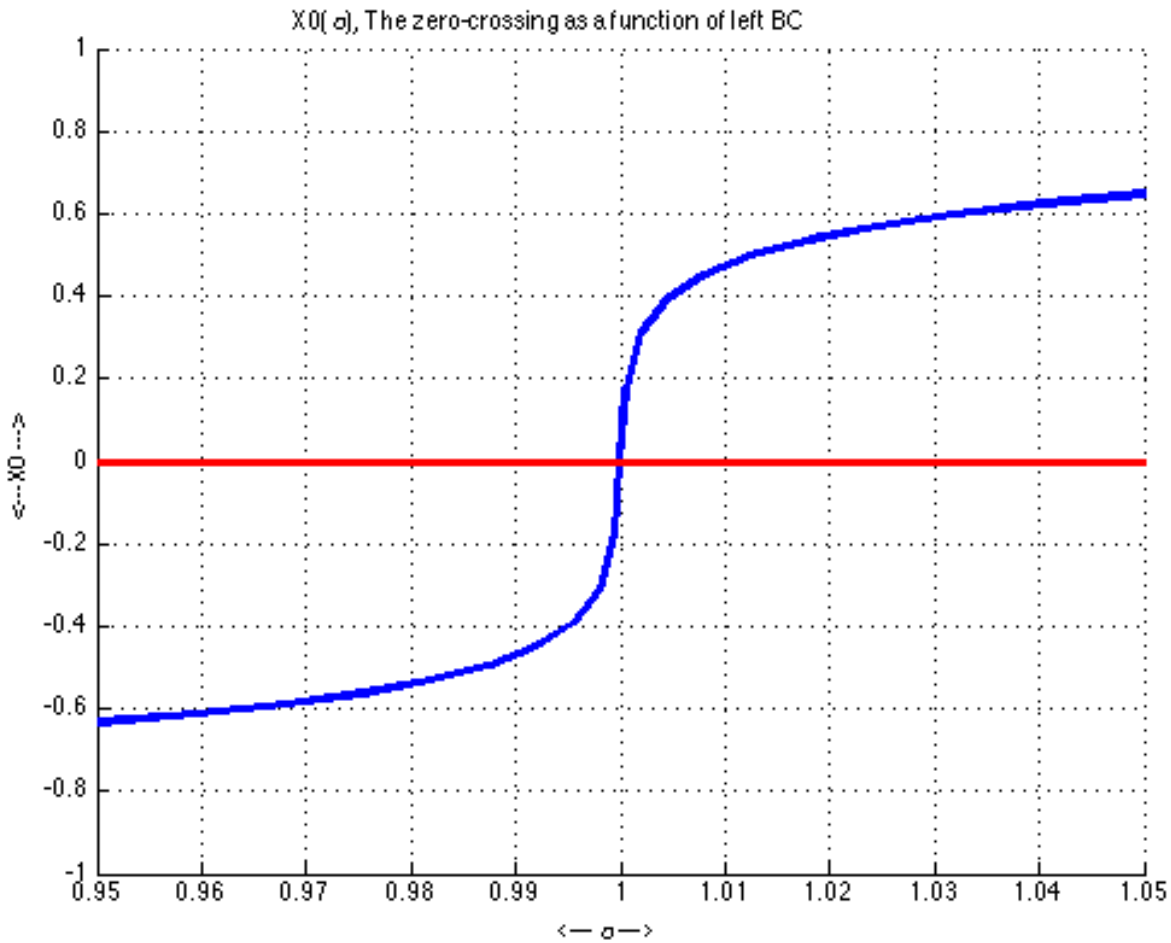
# TIDE: Sensitivity to $a$

Our solution does not seem very sensitive to the value of $a$, the location of the left endpoint. For "reasonable" perturbations, $u(x)$ seems to change not at all!



# TIDE: Computing $X_0(\alpha)$

Here is a plot suggesting the behavior of $x_0$ as a function of $\alpha$, verifying the extreme sensitivity near the base value.



# TIDE: Expected Value Estimate

If we suppose only $\alpha$ is uncertain, then the symmetry of our results suggests that $\mathbb{E}(x_0(\alpha))$ will be roughly 0, an uninteresting result. If we generate Gaussian deviates of $\alpha$ with mean 1 and standard deviation of 0.05, our estimates

| M | E(X0(ALPHA)) estimate |
|---|---|
| 16 | -0.0262784 |
| 32 | 0.00325575 |
| 64 | -0.0115093 |
| 128 | 0.00144016 |
| 256 | 0.0255031 |
| 512 | 0.0222146 |
| 1024 | -0.0142951 |
| 2048 | 0.00458531 |
| 4096 | -0.000209035 |

# TIDE: Variance Estimate

The variance estimate shows that uncertainty in $\alpha$ means we can expect crossing perturbation magnitudes of about 0.5, that is, halfway to the boundary.

| M | Var(X0(ALPHA)) estimate |
|---|---|
| 16 | 0.351927 |
| 32 | 0.343577 |
| 64 | 0.348422 |
| 128 | 0.335153 |
| 256 | 0.341199 |
| 512 | 0.346304 |
| 1024 | 0.341493 |
| 2048 | 0.348165 |
| 4096 | 0.346826 |

For this problem, the strong variance in $x_0$ persists even if we reduce the variance of $\alpha$, or if we model $\alpha$ by a uniform deviate.

# TIDE: Discussion

The plot of $x_0(\alpha)$, and the computations of $\mathbb{E}(x_0(\alpha))$ and $\sigma^2(x_0(\alpha))$ were done by evaluating the full state solution at hundreds or thousands of values of $\alpha$.

The resulting information is useful, but we really only investigated uncertainty with respect to a single parameter, on a simple 1D problem.

In practical problems, we expect that each state solution will be quite expensive. This alone might suggest using some kind of **interpolation scheme** to build a polynomial model of $x_0(\alpha)$ based on a much reduced number of sample evaluations.

However, practical problems are also likely to have tens (or even hundreds) of uncertainty parameters to investigate simultaneously, meaning our workload has the potential to explode.

# TIDE: Exercises

A MATLAB program for solving the time-independent viscous Burger's equation is available at http://people.sc.fsu.edu/~jburkardt/m_src/burgers_steady_viscous/... burgers_steady_viscous.html

**Exercise 35:** Figure out how the program accepts boundary condition data from the user. Model the left hand boundary value $\alpha$ by a Gaussian quantity with mean $+1$ and standard deviation 0.05. Carry out the Monte Carlo analysis that estimates $\mathbb{E}(x_0(\alpha))$ and the variance of $x_0(\alpha)$

**Exercise 36:** Repeat the calculation, but now use a Gauss-Hermite quadrature rule of 9 points to estimate the expected value and the variance.

# TIDE: The Time-Dependent Burgers Equation

The time dependent viscous Burgers equation is:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

Now we'll take periodic boundary conditions:

$$u(a,t) = u(b,t) \text{ for } t > 0;$$

and we specify an initial condition for $t = 0$:

$$u(x,0) = u_0(x).$$

and suppose that we will determine the solution up to time $T$.

# TIDE: Discretized Version

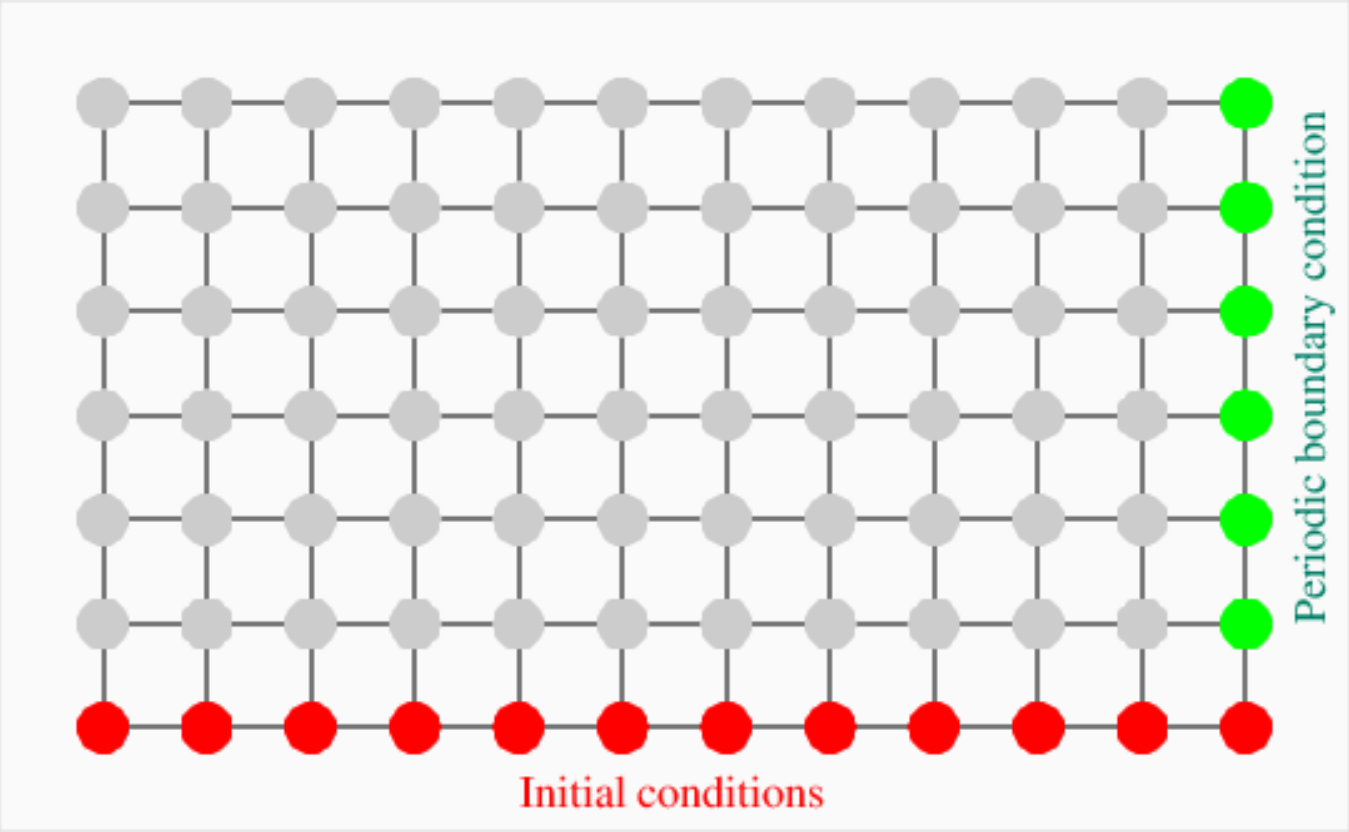
Our discretized geometry involves an $m+1$ by $n+1$ grid spaced equally in $x$ and in $t$, with the solution stored as an array.

Column 0 holds our initial condition. Column $j+1$ is computed from column $j$; entry $(m, j+1)$ is set by periodicity.

```
|  u(0, 0)      ?       ?    ...    ?         ?
S  u(1, 0)      ?       ?    ...    ?         ?
P  u(2, 0)      ?       ?    ...    ?         ?
A  u(3, 0)      ?       ?    ...    ?         ?
C    ...      ...     ...    ...   ...       ...
E  u(m-1,0)     ?       ?    ...    ?         ?
|  u(m, 0)      ?       ?    ...    ?         ?
|
              -------TIME----->
```

# TIDAL: Filling in an array

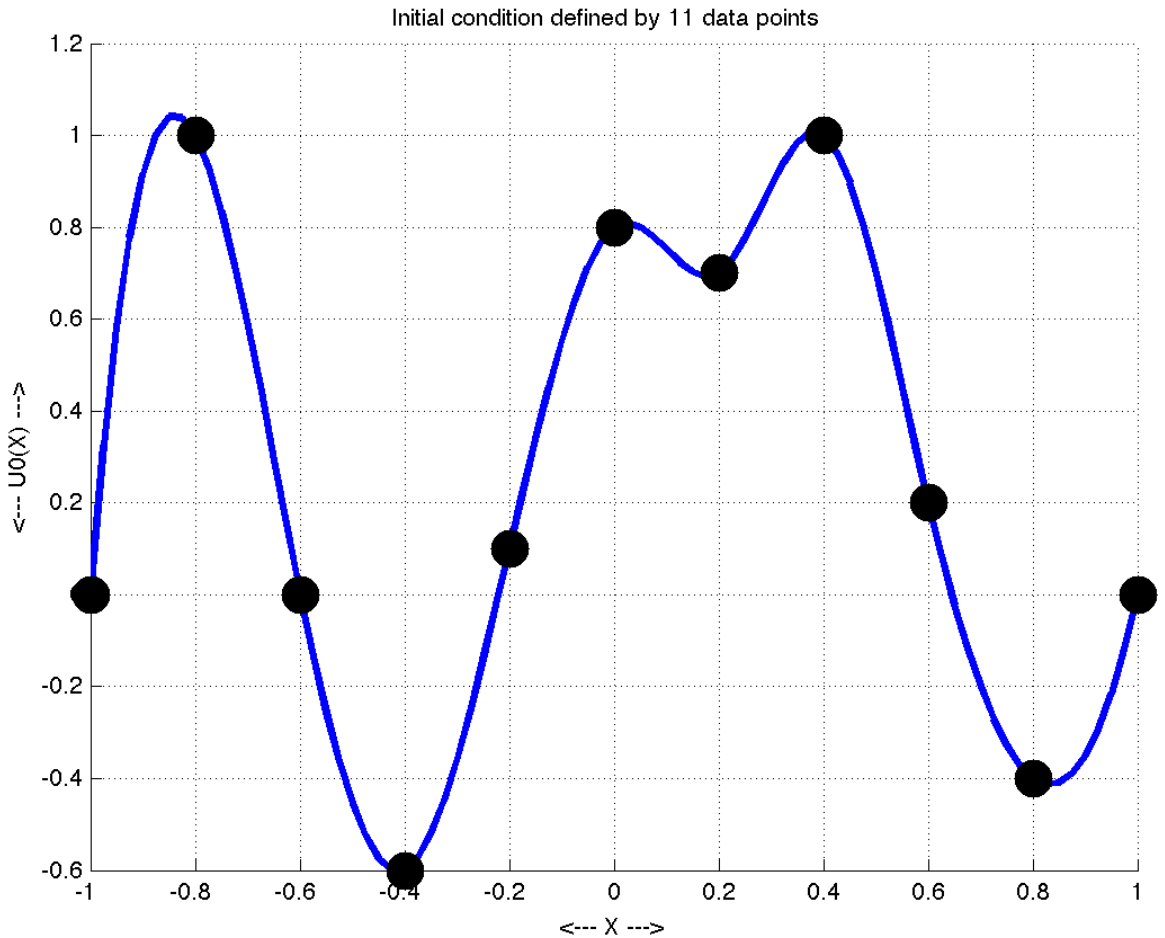
Geometrically, we imagine our problem with time as the $y$ axis, so we are given the solution at the "bottom".




Each solution is more expensive than for our steady problem. If the initial data is uncertain, we also have more parameters to consider.

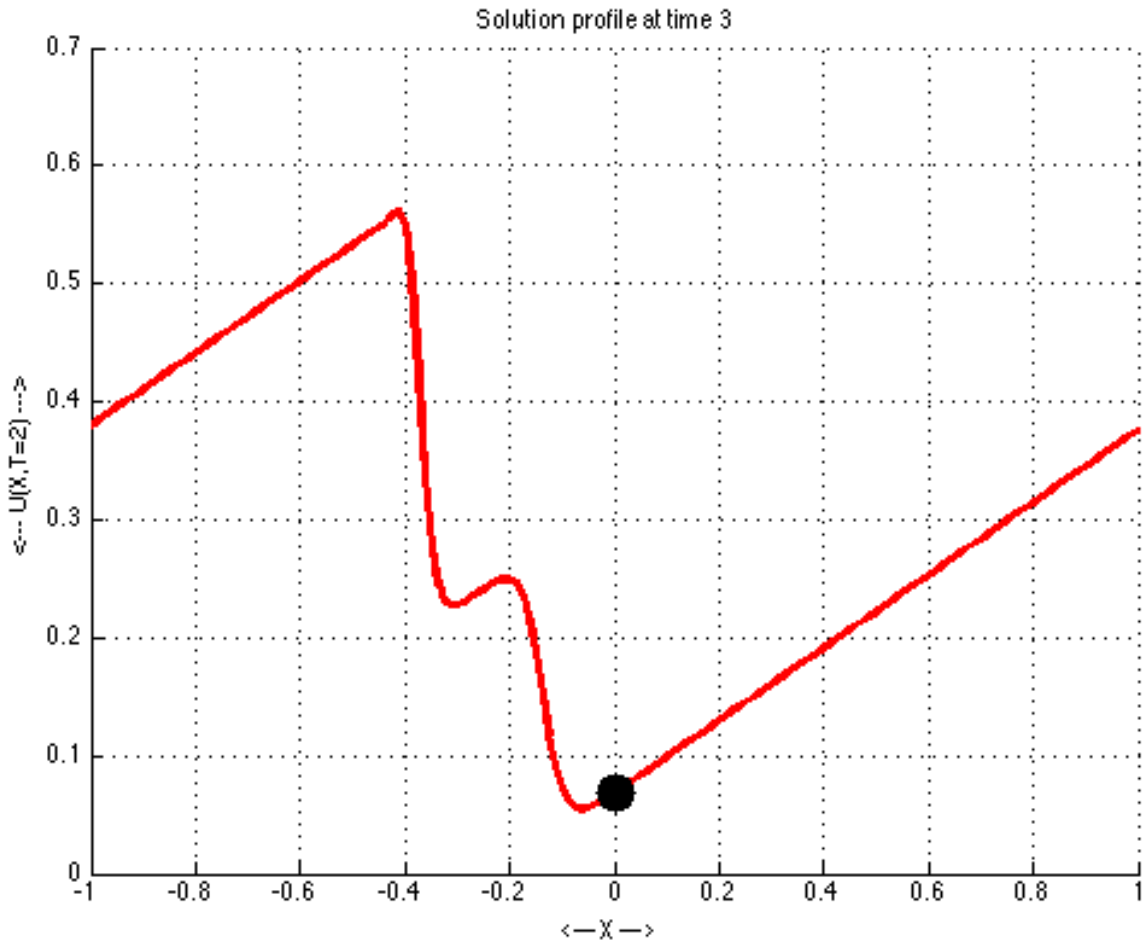
# TIDE: Data Defines the Initial Condition

We concentrate on the dependence of the solution on the initial condition, which we imagine is specified by some discrete set of data $\gamma$ through which we pass a spline:



# TIDE: U(X=0,T=3) is our Quantity of Interest

The solution profile at $T = 3$ looks like this. Suppose our quantity of interest $q(\gamma)$ is simply the profile value at $x = 0$.



# TIDE: Quantifying the Uncertainty

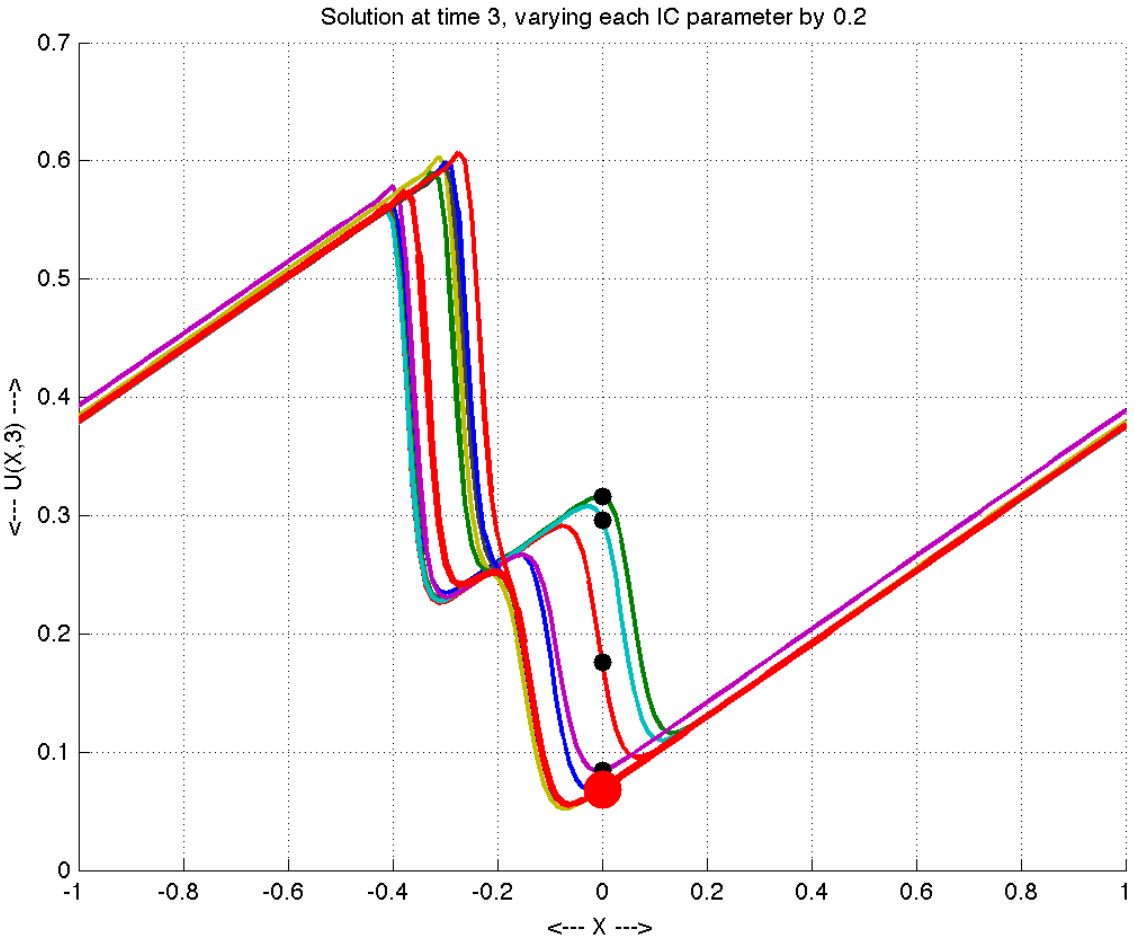
We want to estimate the uncertainty that our input data $\gamma$ induces in our quantity of interest $q(\gamma)$.

To do this requires:

- asserting a model for the input data uncertainty;
- reducing the size of the input data set, if possible;
- sampling the space of input data intelligently;
- solving the state system for each input data set;
- combining the results to estimate $\mathbb{E}(q(\gamma))$ and $\sigma^2(q(\gamma))$.

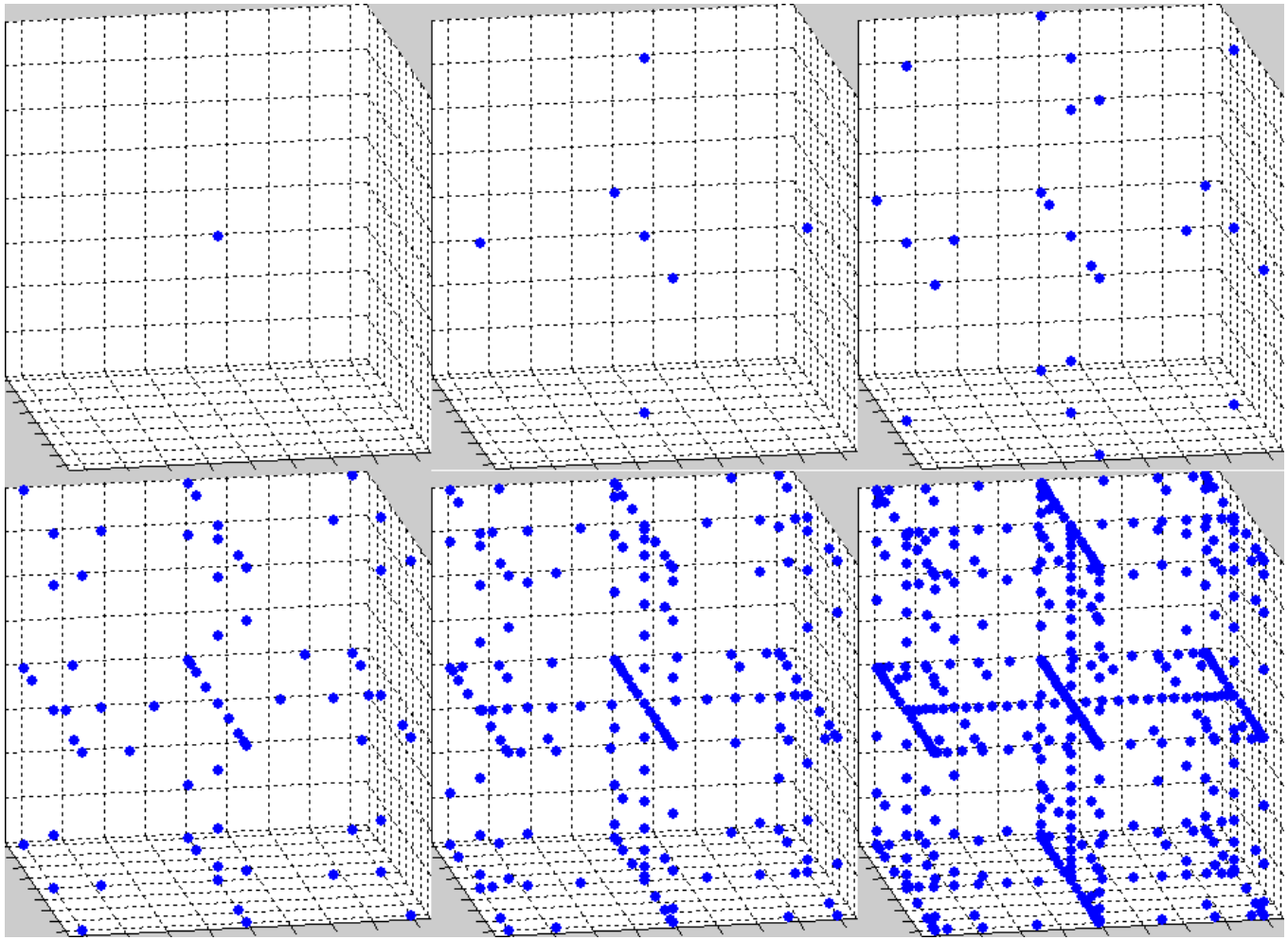
# TIDE: Modeling the Uncertainty

Assume uniform uncertainty in our initial condition parameters. Perturb each value by 0.2. Parameters 2 and 4 are strong, 3 moderate, and the rest have little influence.



# TIDE: Sampling Input Space with 3D Sparse Grid

Freeze all parameters but 2, 3 and 4. Let them vary uniformly $\pm 0.1$ from their base values. This is the space we shall sample.



# TIDE: Sparse Grids

A sparse grid, like Monte Carlo or Quasi Monte Carlo, chooses many sets of data for input to the state system solver. It does not need to alter the internal features of the solver in any way.

Sparse grids differ from other sampling schemes because the sampling pattern produces a highly accurate polynomial model of the uncertainty influence if the state function depends smoothly on the input.

The sparse grid information can be used to estimate an integral, or to produce an interpolant function (that is, a "surrogate function") to the input/output data.

# TIDE: Attempt to Quantify Uncertainty

We have already computed the quantity of interest $Q$, the solution value U at $x = 0$ and time $t = 3$, for a "base" set of parameters.

We'll assume that parameters 2, 3, and 4 vary uniformly about their base values by $\pm 0.2$, and we ask, assuming this uncertainty, what is the expected value of $Q$, written $\mathbb{E}(Q)$?

For this case, we're essentially asking for the average value of $Q$ over the given range of possible parameter values.

We estimate $\mathbb{E}(Q)$ using Monte Carlo and Sparse Grid methods.

# TIDE: Monte Carlo Program Outline

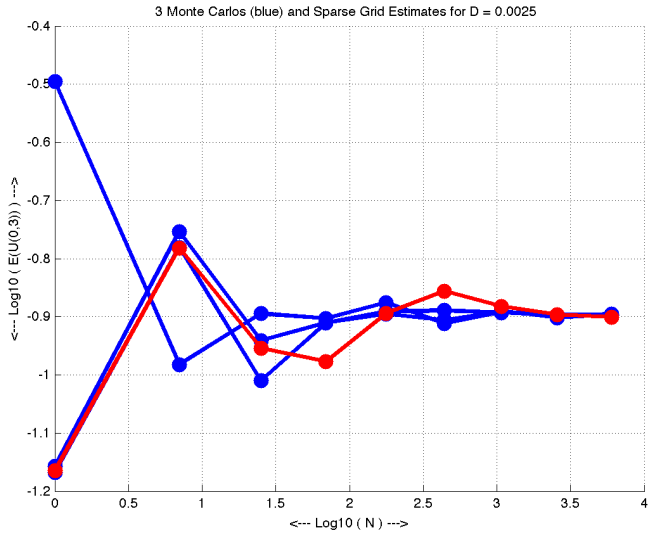
```
d = 3;                          uncertainty dimension
uk(1:9) = uk_base(1:9)          initial condition parameters
nu = ?                          viscosity

mcn = ?                         Free to choose any size
mcx = 2 * rand ( mcn, d ) - 1.0;   Sample [-1,+1]^3 uniformly
mcw = 1.0 / mcn;                The "weight" for MC

q = 0.0
for i = 1 : mcn
  uk(2:4) = uk_base(2:4) + sigma * mcx(i,1:3);
  U = burgers_solver ( uk, nu )
  q = q + mcw * U(nt,(nx+1)/2);
end
```

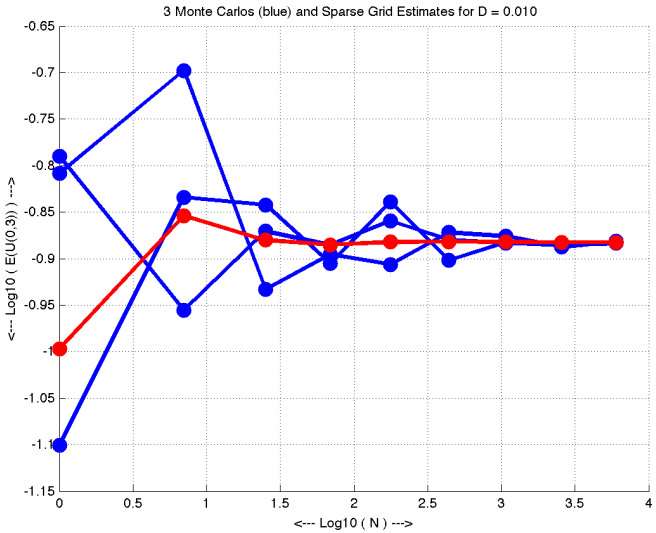
# TIDE: Sparse Grid Program Outline

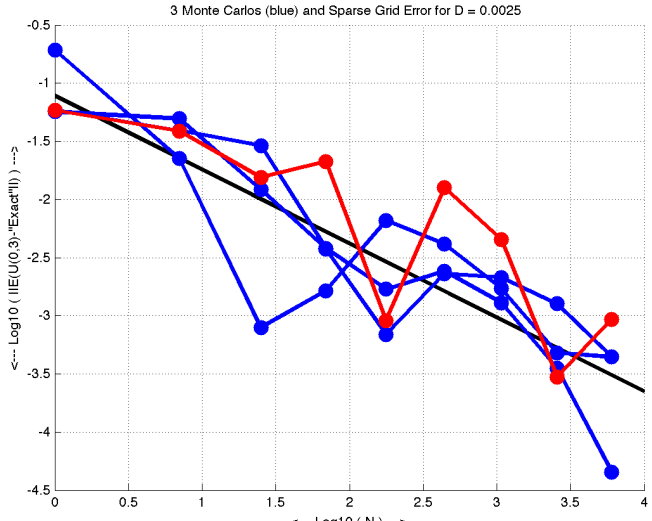
```
d = 3;                          uncertainty dimension
uk(1:9) = uk_base(1:9)          initial condition parameters
nu = ?                          viscosity

level = ?                       sparse grid level 0, 1, 2, ...
[ sgx, sgw ] = nwspgr ( 'ccu', d, level );
sgn = length ( sgw );

q = 0.0
for i = 1 : sgn
  uk(2:4) = uk_base(2:4) + sigma * sgx(i,1:3);
  U = burgers_solver ( uk, nu )
  q = q + sgw(i) * U(nt,(nx+1)/2);
end
```

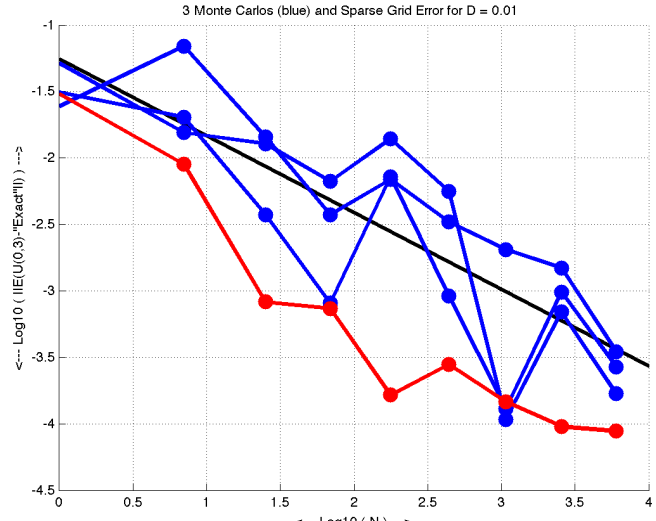
# TIDE: Convergence for Viscosity 0.0025 and 0.1

Q estimates in row 1, Q errors in row 2, SG (red), MC (blue). Sparse grids perform better when viscosity smooths the data.



# TIDE: Exercises

A MATLAB program for solving the time-dependent viscous Burger's equation is available at http://people.sc.fsu.edu/~jburkardt/m_src/burgers_time_viscous/... burgers_time_viscous.html

**Exercise 37:** Learn how the program works, and how the initial condition can be specified by a spline curve that depends on parameters. Then carry out the Monte Carlo analysis that estimates the expected value of the Burgers solution at the midpoint, at the final time, assuming that the three specified input quantities can vary uniformly by 0.2 from their specified values.

**Exercise 38:** Try the same calculation, using a Gauss-Legendre product rule to estimate the probability integral.

**Exercise 39:** Try the same calculation, using a Clenshaw-Curtis sparse grid rule to estimate the probability integral.

# TIDE: Uncertainty Quantification

We have looked at two simple computational problems, and considered the influence of uncertainty in the input data upon an output quantity of interest.

Since the typical input data set can be large, it is important to try to identify those parameters that most strongly influence the output. A rough guess can be done by perturbations of the input; a more sophisticated approach computes a reduced order model.

A probability density function must be assigned to the input parameter space, reflecting our model of the uncertainty.

The sampling approach solves the system many times, and computes a quantity of interest $Q$ by direct averaging (Monte Carlo) or evaluating an input/output model (sparse grid).

# TIDE: A Little More About Sparse Grids

A sparse grid approach may be more efficient that Monte Carlo sampling if the dependence of $Q$ is smooth.

In this example, we found that some input parameters had very little influence. In our uncertainty model, we simply kept them fixed. But sparse grids can be designed that give most, but not all, attention to the dominant variables, while paying some attention to those with known weaker influence.

We used a simple uniform probability density, and in fact, the same one, for the three input parameters. Sparse grids can model normal variation, exponential variation, and other forms, and different densities can be used for different inputs.

# TIDE: High Performance Computing

In our simple case study, parallelism is available twice:

1) The sampling procedure provides, in advance, the input data sets at which the state system must be evaluated. Each of these evaluations can be carried out independently, and has only to return the final state solution, or some associated output data.

2) Because sampling procedures are non-intrusive, the state system evaluation does not need to be rewritten or adjusted in any way. Presumably, this pre-existing "deterministic" code has already been highly optimized and parallelized.

A single state solver might run efficiently on 200 nodes. An MPI approach could request 10,000 nodes, allowing us to compute 50 states simultaneously. If we have 5,000 sample inputs to process, each state solver runs through 100 sets, and the results will be collected and analyzed on a master process.

# TIDE: High Performance Computing

Another approach to parallelism divides the problem into independent tasks, to be executed in any order, and at any time. A master task divides up the problem, submits the tasks to a queue, collects results as they (unpredictably) are completed, and reports the final result.

Such a system is available even in MATLAB, as "task computing".

Such an approach takes advantage of a heavily scheduled computing cluster; instead of waiting for enough processors to load the entire set of tasks, tasks opportunistically seize processors as they become available.

Sampling approaches such as Monte Carlo and Sparse Grid can readily be implemented with such an approach.

# TIDE: Analyzing a Stochastic Problem

Recipe for a UQ collocation on HPC:

- lay out the mathematical model;
- specify input parameters;
- for given inputs, develop solver (already available?);
- compute the basic solution $u$;
- identify influential parameters;
- reduce model (simply cut some parameters?);
- model input uncertainty;
- choose sampling approach (MC? Sparse grid?);
- implement on HPC;
- extract quantity of interest $q(u)$.

# RECAP: One Page Summary

We need **integrals** to formulate the stochastic partial differential equations in a way that allows us to determine the solution functions, or (more typically) the statistical quantities that characterize them.

We need **high dimensional integrals** because the stochastic influence on the PDE typically involves a potentially unlimited number of factors or coefficients.

We need to **approximate high dimensional integrals** because closed-form solutions are unavailable.

We may need **sparse grids to approximate high dimensional integrals** because the standard approximation methods for high dimensional integrals quickly become overloaded with excessive (and unnecessary!) work.

# END!


THANK YOU FOR YOUR ATTENTION,
NOW ASK QUESTIONS!
memegenerator.net