Introduction to Financial Engineering and Algorithms

Lecturer: William W.Y. Hsu

2013/12/18

Monte Carlo Valuation

Introduction

- Simulation of future stock prices and using these simulated prices to compute the discounted expected payoff of an option
  - Draw random numbers from an appropriate distribution
  - Uses risk-neutral probabilities, and therefore risk-free discount rate
  - Distribution of payoffs a byproduct
  - Pricing of asset claims and assessing the risks of the asset
  - Control variate method increases conversion speed
  - Incorporate jumps by mixing Poisson and lognormal variables
  - Simulated correlated random variables can be created using Cholesky distribution

What is a “Monte Carlo” Method?

- The phrase “Monte Carlo method” is very general.
- It describes a method of investigating a problem that uses random numbers and probability statistics to estimate some non-random value.
- Monte Carlo methods appear in many math related fields: physics, engineering, economics, etc.

A General Algorithm for any Monte Carlo Method

- Draw a bunch of random numbers
- Process each of these random numbers in some way, i.e. plug them into an equation or use them in some way
- Analyze the results to find an estimation for a non random value

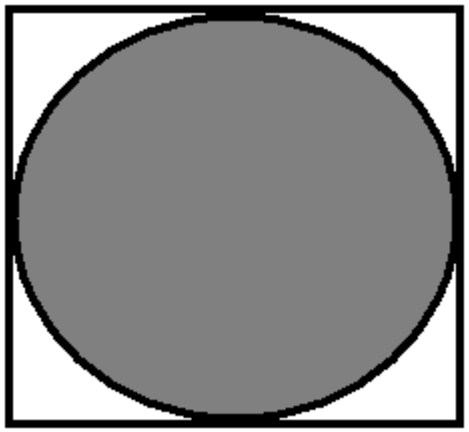
More about Monte Carlo methods

- Monte Carlo methods can help us solve problems that are too complicated to solve using equations, or problems for which no equations exist.
- They are useful for problems which have lots of uncertainty
- They can also be used as an alternate way to solve problems that have equation solutions.
- However, they have drawbacks: Monte Carlo methods are often slower and less accurate than solutions via equations.
- Basically, a Monte Carlo method is a way of solving complex problems through approximation using many random numbers. They are very versatile, but are often slower and less accurate than other available methods.

Example: Monte Carlo Approximation of π

- If one point was drawn at every possible point on this square, then:

$$\frac{\#\text{ darts hitting shaded area}}{\#\text{ darts hitting inside square}} = \frac{\pi r^2}{2(r)^2} = \frac{\pi r^2}{4r^2} = \pi/4 \qquad \pi = 4\frac{\#\text{ darts hitting shaded area}}{\#\text{ darts hitting inside square}}$$



Example: Monte Carlo Approximation of π

- Thus, a simple Monte Carlo method to approximate π would be to simply generate lots of random coordinates on this square and then do:

$$\frac{\text{number of points not in the circle}}{\text{number of points total}}$$

- This should come increasingly close to π.

.

A Monte Carlo Method for Financial Derivatives

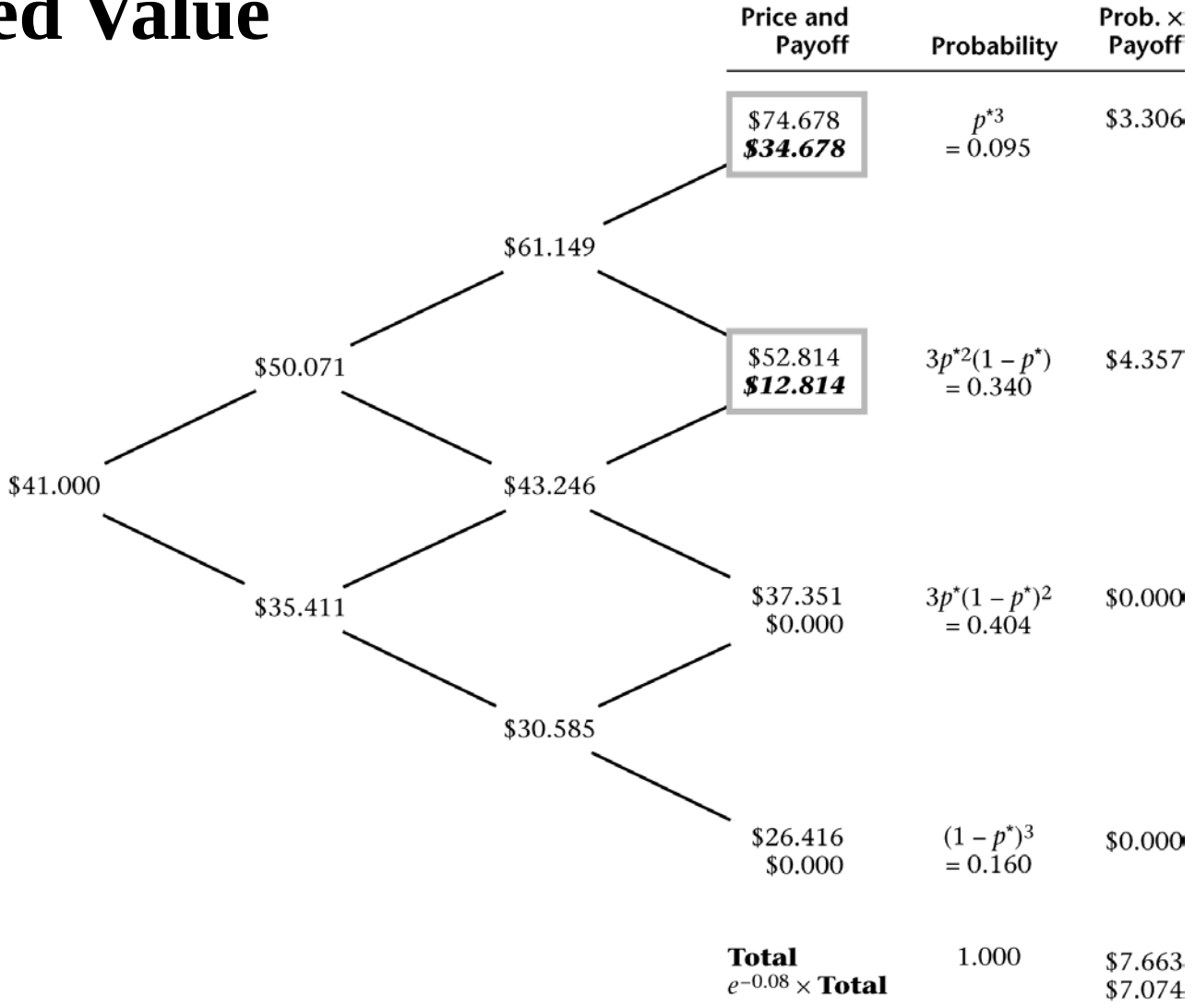
- We don’t know what a stock will do in the future over time T.
  - Black-Scholes assumes a Brownian motion to figure out what is most likely to happen to the price of a stock after time T.
- If we figure out a way to approximate a possible Brownian walk of a stock over some period of time T, we can approximate the Black-Scholes value by:
  1. Doing many sample walks of the stock
  2. Computing the value of a derivative for each sample walk
  3. Averaging the trial derivative values together to come up with an expected value for the derivative

Computing the Option Price as a Discounted Expected Value

- Assume a stock price distribution 3 months from now
- For each stock price drawn from the distribution compute the payoff of a call option (repeat many times)
- Take the expectation of the resulting option payoff distribution using the risk-neutral probability p^*
- Discount the average payoff at the risk-free rate of return
- In a binomial setting, if there are *n* binomial steps, and *i* down moves of the stock price, the European Call price is:

$$\text{European Call Price} = e^{-rT}\sum_{i=1}^{n} max[0, Su^{n-i}d^{i} - K](p^*)^{n-1}(1-p^*)^{i}\frac{n!}{(n-i)!i!}$$

Computing the Option Price as a Discounted Expected Value



Computing the Option Price as a Discounted Expected Value

TABLE 19.1

Computation of option price using expected value calculation and true probabilities. The stock price tree and parameters are the same as in Figure 11.4. The column entitled "Discount Rates along Path" reports the node-specific true annualized continuously compounded discount rates from that figure. "Discount Rate for Path" is the compound annualized discount rate for the entire path. "Prob. of Path" is the probability that the particular path will occur, computed using the true probability of an up move (52.46%). The last column is the probability times the payoff, discounted at the continuously compounded rate for the path.

Path	Discount Rates along Path			Discount Rate for Path	Prob. of Path	Payoff ($)	Discounted ($) (Prob. × Payoff)
uuu	35.7%	32.3%	26.9%	31.64%	0.1444	34.678	3.649
uud	35.7%	32.3%	26.9%	31.64%	0.1308	12.814	1.222
udu	35.7%	32.3%	49.5%	39.18%	0.1308	12.814	1.133
duu	35.7%	49.5%	49.5%	44.91%	0.1308	12.814	1.070
udd	—	—	—	—	—	0	0
dud	—	—	—	—	—	0	0
ddu	—	—	—	—	—	0	0
ddd	—	—	—	—	—	0	0
					Sum		7.074

Computing Random Numbers

- There are several ways for generating random numbers:
  - Use “RAND” function in Excel to generate random numbers between 0 and 1 from a uniform distribution U(0,1).
  - To generate random numbers (approximately) from a standard normal distribution N(0,1), sum 12 uniform (0,1) random variables and subtract 6.
  - To generate random numbers from any distribution D (for which an inverse cumulative distribution D^{-1} can be computed),
    - generate a random number x from U(0,1)
    - find z such that $D(z) = x$, i.e., $D^{-1}(x) = z$
    - repeat

Simulating Lognormal Stock Prices

- Recall that if Z ~ N(0,1), a lognormal stock price is

$$S_t = S_0 e^{(r-0.5\sigma^2)t+\sigma\sqrt{t}Z}$$

- Randomly draw a set of standard normal Z's and substitute the results into the equation above.
  - The resulting S_t's will be lognormally distributed random variables at time t.
- To simulate the path taken by S (which is useful in valuing path-dependent options) split t into n intervals of length Δt.

$$S_{\Delta t} = S_0 e^{(r-0.5\sigma^2)\Delta t+\sigma\sqrt{\Delta t}Z(1)}$$

$$S_{2\Delta t} = S_{\Delta t} e^{(r-0.5\sigma^2)\Delta t+\sigma\sqrt{\Delta t}Z(2)}$$

$$S_{3\Delta t} = S_{2\Delta t} e^{(r-0.5\sigma^2)\Delta t+\sigma\sqrt{\Delta t}Z(3)}$$

Examples of Monte Carlo Valuation

- If $V(S_t, t)$ is the option payoff at time *t*, then the time-0 Monte Carlo price $V(S_0, 0)$ is

$$V(S_0, 0) = \frac{1}{N} e^{rT} \sum_{i=1}^{n} V(S_T^i, T)$$

  where $S_T^{\,1}, \ldots, S_T^{\,n}$ are *n* randomly drawn time-*T* stock prices.
- For the case of a call option,

$$V\left(S_T^i, T\right) = \max(0, S_T^i - K)$$

- Monte Carlo is flexible but still have the deficiency of convergence speed $\frac{1}{\sqrt{N}}$.

Examples of Monte Carlo Valuation

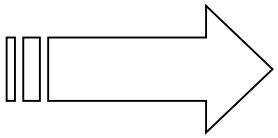
- Example: Value a 3-month European call where the S_0=\$40, K=\$40, r=8%, and σ=30%

$$S_{3m} = S_0 e^{\left(0.08 - 0.5 * 0.3^2\right) * 0.25 + \sigma\sqrt{0.25}Z}$$

- For each stock price, compute

  Option payoff = $max(0, S_{3\text{ months}} - \$40)$

- Average the resulting payoffs
- Discount the average back 3 months at the risk-free rate



\$2.804 versus \$2.78 Black-Scholes price

Examples of Monte Carlo Valuation

- Monte Carlo valuation of American options is not as easy.
- Monte Carlo valuation is inefficient:
  - 500 observations s=$0.180 6.5%
  - 2500 observations s=$0.080 2.9%
  - 21,000 observations s=$0.028 1.0%
- Monte Carlo valuation of options is especially useful when:
  - Number of random elements in the valuation problem is two great to permit direct numerical valuation.
  - Underlying variables are distributed in such a way that direct solutions are difficult.
  - The payoff depends on the path of underlying asset price.

Example

- S = 100; X = 110; r = 0.1;
- σ = 0.4; T = 6
- Exact c_t = **53.4636**
- Monte Carlo:
  - # sims: 10 — c_t = 15.4533
  - # sims: 1,000 — c_t = 54.9804
  - # sims: 1,000,000 — c_t = 53.5126
  - # sims: 100,000,000 — c_t = **53.4593**
  - # sims: 1,000,000,000 — c_t = **53.4722**

Examples of Monte Carlo Valuation

- Monte Carlo valuation of Asian options:
  - The payoff is based on the average price over the life of the option.

$$S_1 = 40e^{(0.08-0.5*0.3^2)*t/3+\sigma\sqrt{t/3}Z(1)}$$
$$S_2 = S_1 e^{(0.08-0.5*0.3^2)*t/3+\sigma\sqrt{t/3}Z(2)}$$
$$S_3 = S_2 e^{(0.08-0.5*0.3^2)*t/3+\sigma\sqrt{t/3}Z(3)}$$

  - The value of the Asian option is computed as

$$C_{asian} = e^{-rt}\mathrm{E}(max[(S_1+S_2+S_3)/3 - K, 0])$$

Optimization

- One of the main criticisms of the Monte Carlo method is how time consuming it is. This can often be fixed with clever optimization.
- There are two main ways in which a Monte Carlo program can be made more efficient.
- Time optimization – making sure the code is as efficient as possible, choosing the best procedures to use, etc
- Variance reduction – making a set of a certain number of trials more accurate.

Time Optimization

- Without any optimization, the original European Call pricing method from last time takes about 21.5 seconds to run a simulation of step size 20 and trial size 1000.
- There are a number of changes that we can make to the code itself that will make it more efficient to run.

Time Optimization

- Instead of multiplying the payoff by e^{-rt} for every trial, we can assign e^{-rt} to a variable ahead of time and use that value in the trials. (Allows us to do less * e^{-rt} calculations)
- Instead of calling another procedure to generate the next step in a random walk, we can put the code right into the main method
- these optimizations combined make the program about 5% faster, bringing the time down to ~20 seconds.

Time Optimization

- The largest benefit to this program can be achieved by changing the way we work with random numbers.
- Previously, the program would create a method to generate a random number and then generate a random number for each step.
  - If we generate the random numbers as a set outside of the loop, we can lower the time significantly.
- If we change the procedure that maple uses to generate random numbers, we can cut the run time of the procedure even more.

Chart About Time Optimization

Optimization	Run time(in seconds)
Original	22
Calculating e^{-rT} once instead of many times	21
Putting code into main method	20
changing the method used for random #'s	10
generating random #'s outside of the loop	0.7

Variance Reduction

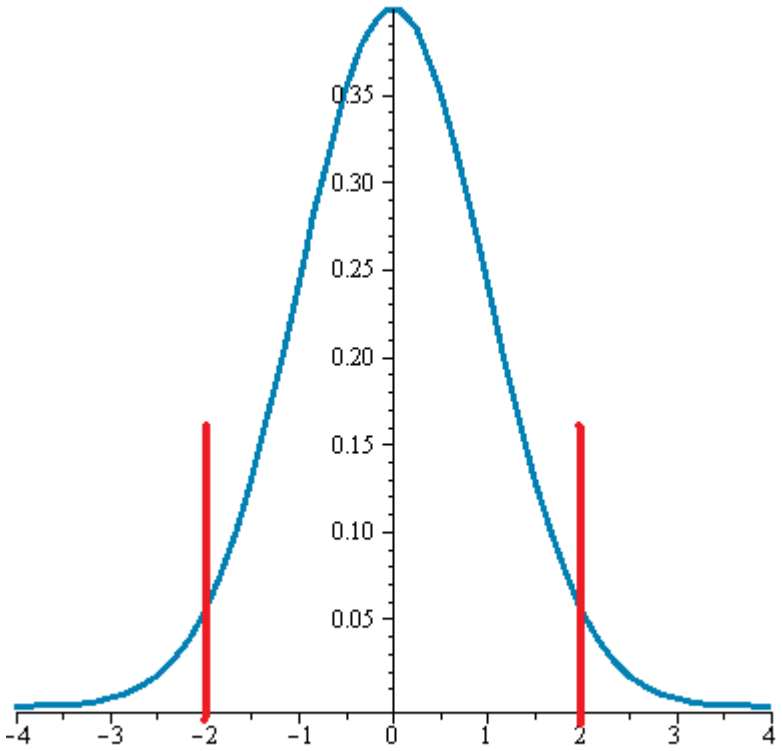
- Variance reduction is a statistical term that basically means trying to lower the standard deviation without increasing the sample size.
- Normally, in order to half the standard deviation, we must quadruple the sample size.

Variance reduction: Antithetic Variate

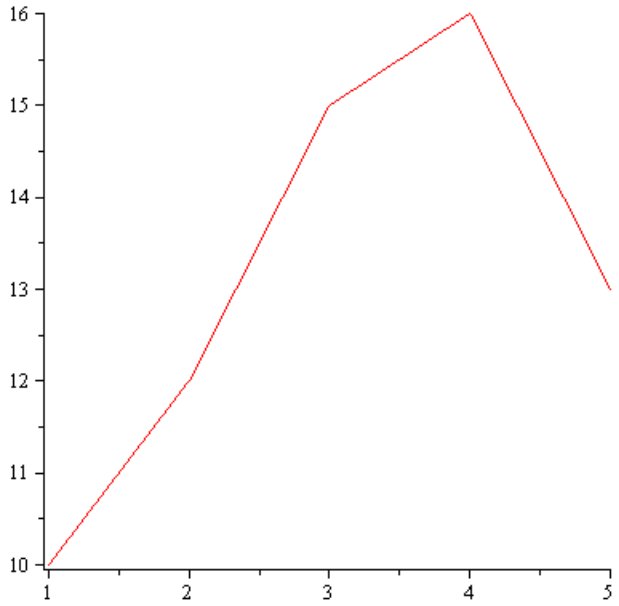
- Because the random variable used in the procedure is normally distributed with a mean of 0 and is symmetric, the probability that a random path is drawn is exactly the same as the probability that the negative of that path is drawn.

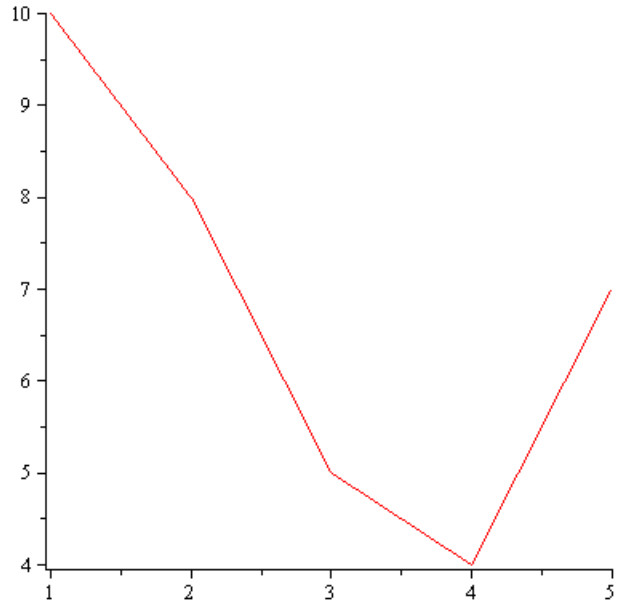
The Antithetic Variate

- In the probability distribution curve of a normal variable, the chances of drawing say, a -2 are exactly the same as drawing a 2. In this case, -2 would be the **antithetic variate** of 2 and vice versa.



The Antithetic Variate





Since the chances of drawing a random number for the random part of the brownian motion are exactly the same as drawing the negative of that number, and since the "drift" is the same for all trials, we can effectively generate two random walks from one set of random numbers.

The Antithetic Variate

- This lets us effectively double our sample size without doing much extra work.
- A 50,000 run trial using Antithetic Variates will have the accuracy of a 100,000 run trial and so on.
- This helps more for lower trial sizes.

Other Variance Reduction strategies

- Another method is the Control Variate Method.
- This method uses the value of a option that has a high correlation to the one which we are trying to predict a value for which can be priced using formulas to lower the Variance (error) of our trials.
- Let x be the simulated option value, y be the value of a highly correlated option obtained with a formula, and z be a simulated option value for y.
- Instead of just generating lots of values of x, we would generate lots of values of: x+(y-z) Essentially, this strategy evaluates (x-z)+y.
- $Var(x-z)=Var(x)+Var(z)-2*cov(x,z)$
- If $Var(z)<2*cov(x,z)$, then $Var(x-z)<Var(x)$

Randomness

Truly Random

- Follows directly from definition of random.
- Each element has equal probability of being chosen from the set.

Truly Random Examples

- Randomly emmited particles of radiation
  - Geiger Counter

- Thermal noise from a resistor
  - Intel’s Random Number Generator

The Advantages Of True Random Numbers

- No periodicities.
- Not based on an algorithm.
- No predictability of random numbers based on knowledge of preceding sequences.
- Certainty that no hidden correlations are present.

Pseudorandom

- A finite set of numbers that display qualities of random numbers.
- Tests can show that there are patterns.
- Subsequent numbers can be “guessed”.

Pseudorandom Number Generator

- The pseudo-random number generator requires a number to start with that gets plugged in to the set of equations.
- After that it uses part of the result from the last time it was used as input to the next iteration.
- This starting number is called the seed.

Generating a Gaussian: Box-Muller method

- Generate 2 uniform random numbers x_1 and x_2.
- Compute the Gaussian random number with mean 0 and variance 1 by:
  - $y_1 = \sqrt{-2 \ln x_1} \cos(2\pi x_2)$
  - $y_1 = \sqrt{-2 \ln x_1} \sin(2\pi x_2)$

Quasi-Random

- A series of numbers satisfying some mathematical random properties even though no random appearance is provided.
- Good for Monte-Carlo methods.
  - Lower discrepancies offer better convergence.

Some Tests for Randomness

- Entropy
  - Information density of the content of a sequence
    - High density usually means random
- Arithmetic Mean
- Chi-square Test
  - Provides a probability for the randomness for a sequence
- An example Pseudorandom number test
  - http://www.fourmilab.ch/random/

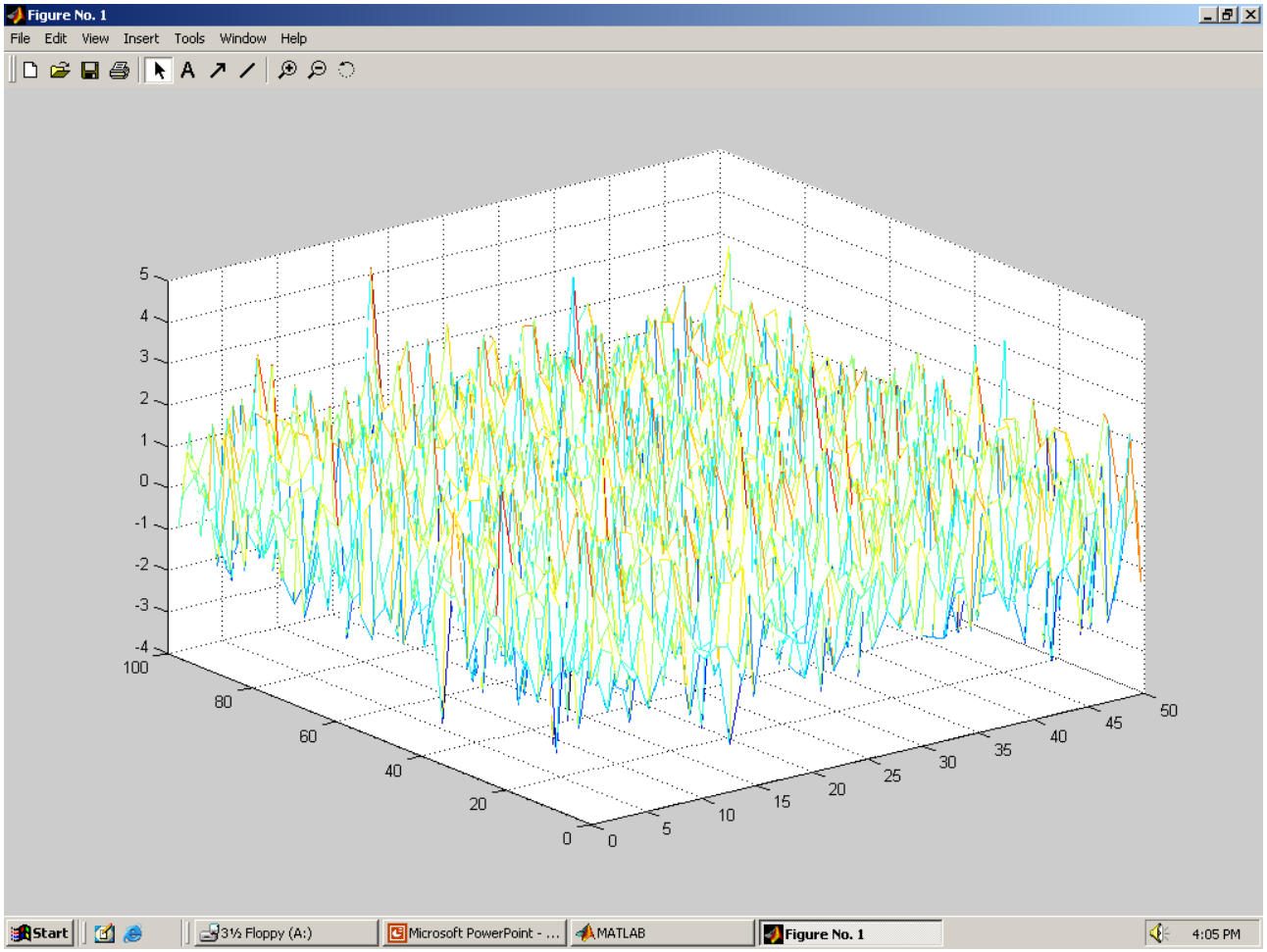
Requirements for Sequential Random Number Generators

- Uniformly distributed
- Uncorrelated
- Never repeats itself
- Satisfy any statistical test for randomness
- Reproduceable
- Portable
- Can be changed by adjusting an initial “seed” value
- Can easily be split into many independent subsequences
- Can be generated rapidly using limited computer memory

Random Number Generator in Matlab

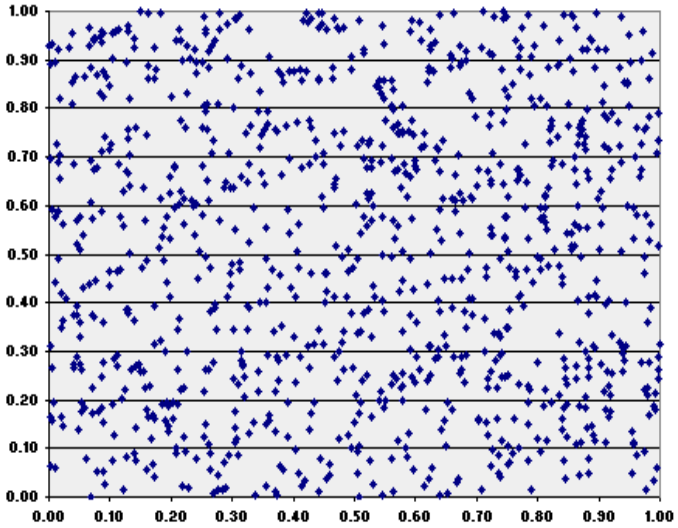
- Y = randn(m,n) or Y = randn([m n]) returns an m-by-n matrix of random entries.
- Y = randn(m,n,p,...) or Y = randn([m n p...]) generates random arrays.
- Y = randn(size(A)) returns an array of random entries that is the same size as A.
- randn, by itself, returns a scalar whose value changes each time it's referenced.

Example: x=randn(100,50)



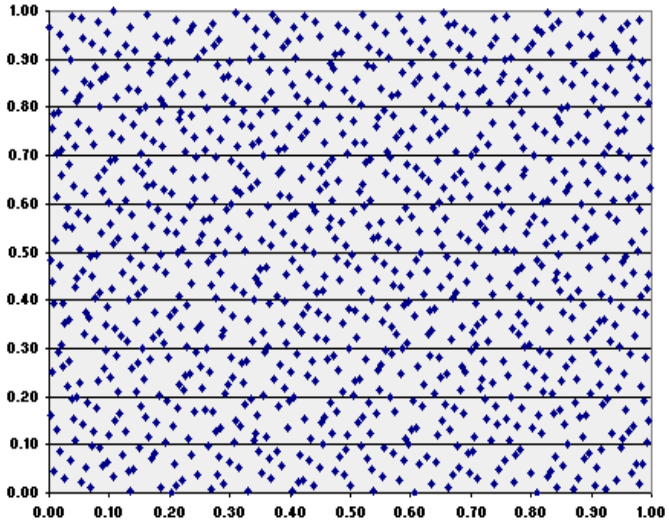
Quasi-Monte Carlo Methods

- $S_i = S_{i-1}\exp\left[\left(r - \frac{\sigma^2}{2}\right)\Delta t + \sigma\sqrt{\Delta t}Z_i\right]$
- Pseudo random numbers, i.e., rand(), provides convergence speed $\frac{1}{\sqrt{N}}$.



Quasi-Monte Carlo Methods

- Quasi-Monte Carlo methods:
  - **Halton Sequences**
  - **Faure Sequences**
  - **Sobol Sequences**
  - Provides $\frac{(\log N)^S}{N}$ convergence rate.



Discrepancy

- Discrepancy of a sequence $(X_0, X_1, X_2, \dots)$

$$D_N(X_1, X_2, \dots, X_N)$$

$$= \sup_{\substack{0 \le u_i < v_i \le 1 \\ i=1,\dots,d}} \left| \frac{\left| \{x_1, x_2, \dots, x_N\} \cap \prod_{i=1}^{d} [u_i, v_i) \right|}{1} - \prod_{i=1}^{d} (v_i - u_i) \right|$$

Low-Discrepancy Sequence

- Sequence with the property that for all N, the subsequence $x_1, x_2, \ldots, x_N$ is almost uniformly distributed and $x_1, x_2, \ldots, x_{N+1}$ is almost uniformly distributed as well.
- This is called **Quasi-random** sequence
- The "quasi" modifier is used to denote more clearly that the numbers are not random, and rather deterministic.

Halton Sequence

- $n = d_j b^j + \cdots + d_2 b^2 + d_1 b + d_0, 0 \leq d_i < b.$
- $\phi_b(n) = \frac{d_0}{b} + \frac{d_1}{b^2} + \cdots + \frac{d_j}{b^{j+1}}$

- $d_j \ldots d_2 d_1 d_0 \Rightarrow 0.d_0 d_1 \ldots d_j$

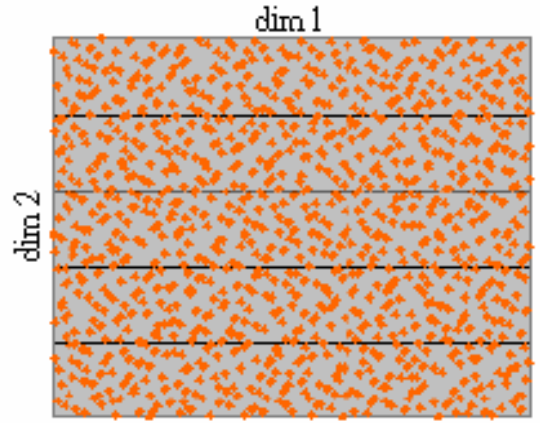
Halton Sequence

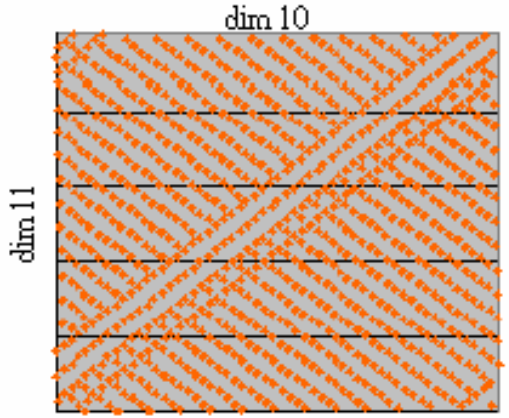
- Dim1 base=2 $\left(\frac{1}{2},\frac{1}{4},\frac{3}{4},\frac{1}{8},\frac{5}{8},\frac{3}{8},\frac{7}{8},\frac{1}{16},\cdots\right)$
- Dim2 base=3 $\left(\frac{1}{3},\frac{2}{3},\frac{1}{9},\frac{4}{9},\frac{7}{9},\frac{2}{9},\frac{5}{9},\frac{8}{9},\cdots\right)$
- Dim3 base=5 $\left(\frac{1}{5},\frac{2}{5},\frac{3}{5},\frac{4}{5},\frac{1}{25},\frac{6}{25},\frac{11}{25},\frac{16}{25},\cdots\right)$

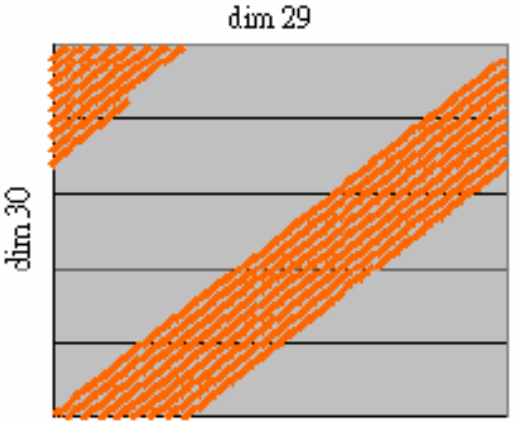
- Drawbacks:
  - Monotonically increasing
  - High correlation

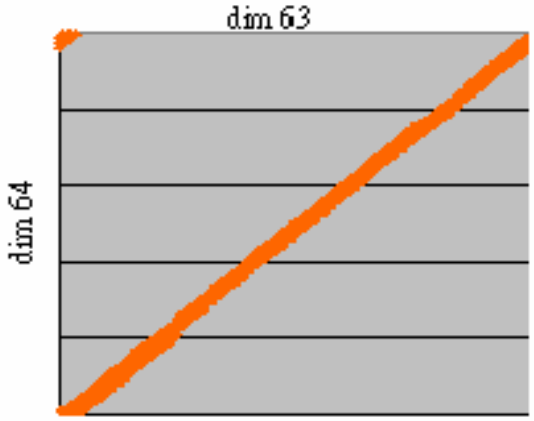
	$b=2$	$b=3$	$b=5$	$\cdots$
$n=0$	0	0	0	$\cdots$
$n=1$	0.5000	0.6667	0.6000	$\cdots$
$n=2$	0.2500	0.3333	0.2000	$\cdots$
$n=3$	0.7500	0.2222	0.8000	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

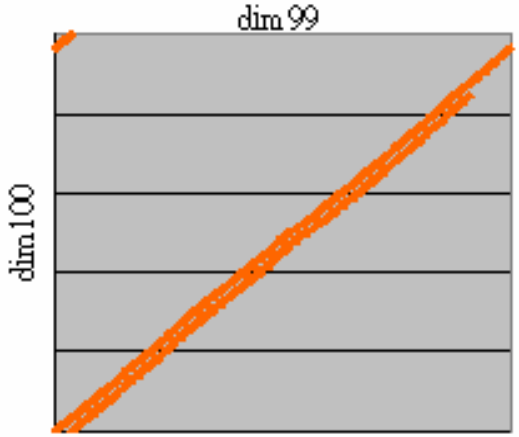
Halton Sequence - Two-Dimensional Projections











Faure Sequence

- Use the smallest prime number larger than the dimension D as base of all sequences.

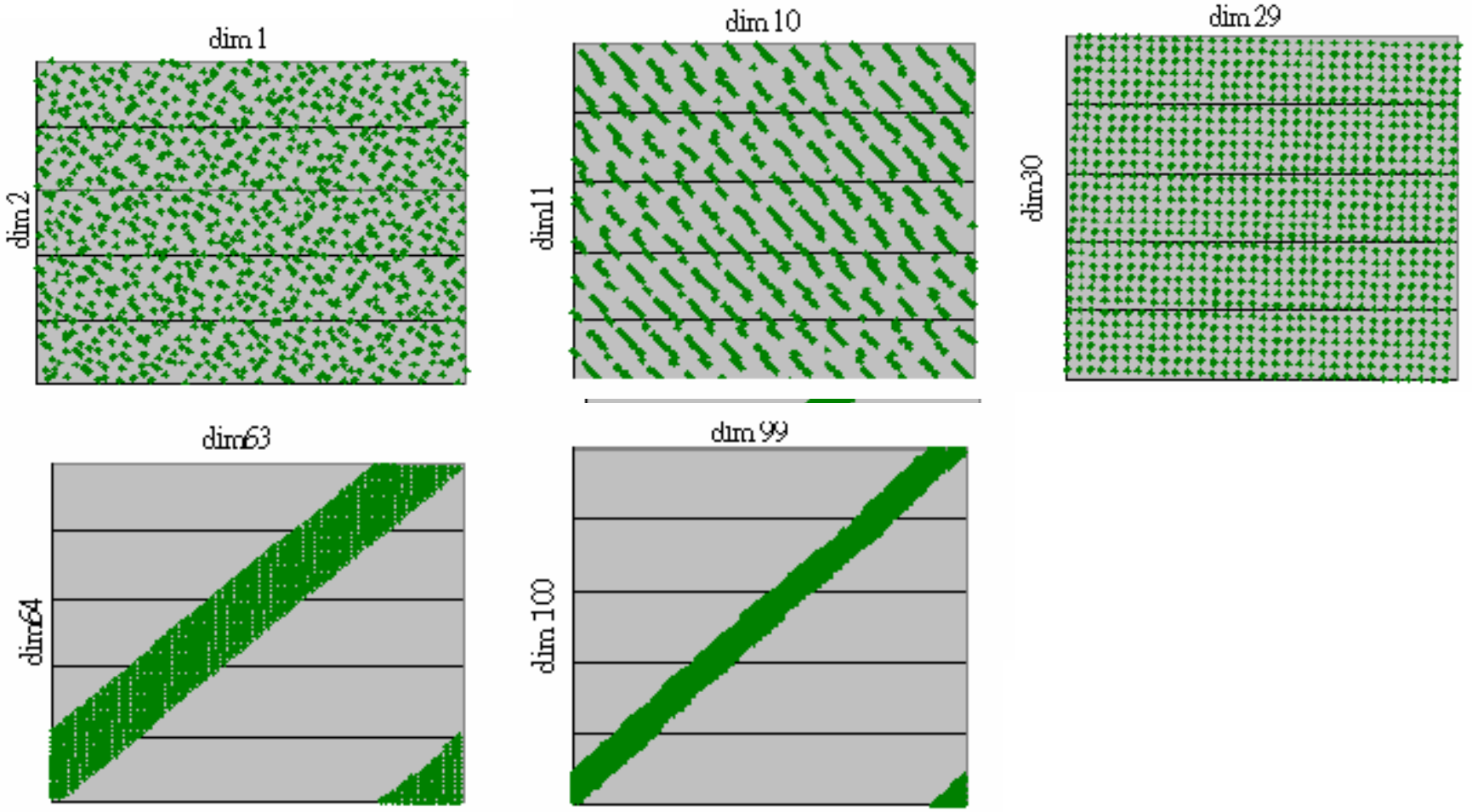
$$C_{n-1} = a_0 m^{-1} + a_1 m^{-2} + \cdots + a_r m^{-(r+1)}$$
$$C_n = b_0 m^{-1} + b_1 m^{-2} + \cdots + b_r m^{-(r+1)}$$

$$b_j = \sum_{i \geq j}^{r} \binom{i}{j} a_i \bmod m$$

Faure Sequence

- Dim1 base=3 $\left(\frac{1}{3},\frac{2}{3},\frac{1}{9},\frac{4}{9},\frac{7}{9},\frac{2}{9},\frac{5}{9},\frac{8}{9},\cdots\right)$
- Dim2 base=3 $\left(\frac{1}{3},\frac{2}{3},\frac{4}{9},\frac{7}{9},\frac{1}{9},\frac{8}{9},\frac{2}{9},\frac{5}{9},\cdots\right)$
- Dim3 base=3 $\left(\frac{1}{3},\frac{2}{3},\frac{7}{9},\frac{1}{9},\frac{4}{9},\frac{5}{9},\frac{8}{9},\frac{2}{9},\cdots\right)$
- Drawbacks (same as halton)
  - Cycle length
  - High correlation

Faure Sequence - Two-Dimensional Projections



Sobol Sequence

- To generate the *j*-th component of the points in a Sobol sequence, we need to choose a primitive polynomial of some degree s_j over the field **G**alois **f**ield(2).

$$P_j = x^{s_j} + a_{1,j}x^{s_j-1} + a_{2,j}x^{s_j-2} + \cdots + a_{s_j-1,j}x + 1,$$

  where the coefficients $a_{1,j}$, $a_{2,j}$, ..., $a_{sj-1,j}$ are either 0 or 1.
  - The error bounds for Sobol sequences indicate that we should use primitive polynomials of as low a degree as possible.
- A sequence of positive integers $\{m_{1,j}, m_{2,j}, \ldots\}$ are defined by the recurrence relation

$$m_{k,j} = 2a_{1,j}m_{k-1,j} \oplus 2^2 a_{2,j}m_{k-2,j} \oplus \cdots \oplus 2^{s_j-1}a_{s_j-1,j}m_{k-s_j+1,j} \oplus 2^{s_j}m_{k-s_j,j} \oplus m_{k-s_j,j},$$

  where $\oplus$ is the bit-by-bit exclusive-or operator.

Sobol Sequence

- The initial values $m_{1,j}$, $m_{2,j}$, ..., $m_{sj,j}$ can be chosen freely provided that each $m_{k,j}$, $1 \leq k \leq s_j$, is odd and less than 2^k.
- The so-called *direction numbers* $\{v_{1,j}, v_{2,j}, \ldots\}$ are defined by

$$v_{k,j} = \frac{m_{k,j}}{2^k}.$$

- Then $x_{i,j}$, the j-th component of the i-th point in a Sobol sequence, is given by

$$x_{i,j} = i_1 v_{1,j} \oplus i_2 v_{2,j} \oplus \cdots,$$

- where i_k is the k-th binary digit of $i = (\ldots i_3 i_2 i_1)_2$.
- Here the notation $(\cdot)_2$ denotes the binary representation of numbers.

Sobol Sequence

- Dim1 base=2 $\left(\frac{1}{2},\frac{3}{4},\frac{1}{4},\frac{3}{8},\frac{7}{8},\frac{5}{8},\frac{1}{8},\frac{3}{16},\cdots\right)$
- Dim2 base=2 $\left(\frac{1}{2},\frac{1}{4},\frac{3}{4},\frac{3}{8},\frac{7}{8},\frac{1}{8},\frac{5}{8},\frac{5}{16},\cdots\right)$
- Dim3 base=2 $\left(\frac{1}{2},\frac{3}{4},\frac{1}{4},\frac{5}{8},\frac{1}{8},\frac{3}{8},\frac{7}{8},\frac{5}{16},\cdots\right)$

Sobol Sequence - Two-Dimensional Projections