Simulation: Random Number Generation

Goals:

- Review the process for generating a simulation-based approximation
- Discuss how to generate random samples from a uniform distribution
- Discuss how to transform numbers from a uniform distribution to the distribution of our choice

Relevant literature:

- Hirsa, Chapter 6; Glasserman, Chapter 2

Review: What's in a simulation approximation?

- In the last lecture, we introduced the concept of simulation and discussed the building blocks needed to use simulation.

  In particular, we need to:

  - Generate random numbers from a desired distribution. This task can be split into generating random numbers from a uniform distribution and transforming them to the desired distribution.
  - Generate a path for the asset given a sequence of random numbers. This path may include a single timestep (as in the case of exact simulation) or a grid of timesteps. It may also include many assets, as is common in modeling futures curves, or a single asset.

→ forwards step, not for Amer-option

Review: Why is simulation important?

- We also discussed that simulation was an important technique for modeling path dependent options and that the uses for simulation extend far beyond options pricing.
- Simulation is our tool of choice when many underlying assets are involved. This is because its rate of convergence is independent of the number of dimensions.
- For problems with a single underlying asset it is a judgment call whether you prefer to use a simulation or PDE approach. In practice I often use simulation even though it is less efficient.

Review: Formulation of Simulation Algorithm

- Recall the following risk-neutral valuation formula for a European call:

$$\begin{aligned} c_0 &= \tilde{\mathbb{E}}\left[e^{-rT}\left(S_T - K\right)^+\right] \\ &= e^{-\int_0^T r_u\, du} \int_{-\infty}^{+\infty} (S_T - K)^+ \, \phi(S_T)\, dS_T \end{aligned}$$

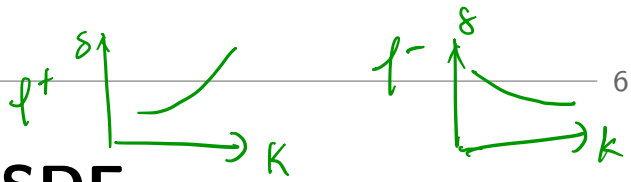
- We can estimate this problem via simulation using the following approximation:

$$c_0 \approx \frac{1}{N} \sum_{i=1}^{N} e^{-rT} (Z_i - K)^+$$

where Z_i is the value of the asset at expiry on the i^{th} simulation path and N is the total number of simulation paths.

Review: Formulation of Simulation Algorithm

- In the last class, we saw that if we know the density, $\phi(S_T)$, then we can use exact simulation.
- What if the density if unknown but the underlying SDE is known?
- As an example, let's consider the Heston model.



Simulation of Discretized SDE

- Recall that the Heston model is defined by the following system of SDE's:

$$\begin{aligned} dS_t &= rS_t\,dt + \sigma_t S_t\,dW_t^1 \\ d\sigma_t^2 &= \kappa(\theta - \sigma_t^2)\,dt + \xi\sigma_t\,dW_t^2 \\ Cov(dW_t^1, dW_t^2) &= \rho\,dt \end{aligned} \tag{1}$$

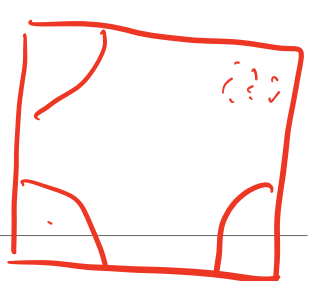
- Although the PDF for the Heston model is unknown, and therefore we cannot simulate it, if we know how to generate random normal samples, then we can generate samples of dW_t^1 and dW_t^2.
- Note that in order to do this we need to be able to generate random numbers that are **jointly normally distributed** with correlation ρ.

Simulation of Discretized SDE: Conclusions

- Clearly, simulating random numbers from a uni or multivariate normal distribution will be a key underlying tenet in our simulation methodology as we will need to do so in every process that includes a Brownian Motion term.
- Simulating the SDE will be less efficient than exact simulation, but will allow us to solve a broader set of problems and have access to the asset's entire path rather than terminal value.

How do we generate random numbers?

- In practice, it is not possible for a computer to generate numbers that are truly random.
- Given that, we want to generate numbers that are as close to random as possible. We refer to these numbers as **pseudorandom**.
- Generating **pseudorandom** uniform numbers is the core building block of almost all Monte Carlo simulations.
  - Clearly, if we weren't able to transform these numbers into more interesting distributions then the technique would not be useful.
- An alternate approach is to use **quasi-random low-discrepancy sequences**. We will come back to these later.
  - An important example of this is **Sobol nodes**



Random Number Generators

- A **random number generator** (RNG) is a method for generating pseudorandom numbers.
- Many stable algorithms exist that are random number generators for a uniform distribution and there is no need to reinvent them.
- These produce a series of numbers $u^1, \ldots, u^K \in [0, 1]$
- We would like our RNG to have the following properties:
  - Each u^k is uniformly distributed on $[0, 1]$
  - The u^k's are independent of each other
- It is relatively easy to generate random numbers that are uniformly distributed, but it is harder to generate random numbers that are truly independent.

Desireable Properties of RNG's

- **Efficiency:** Many applications require generating a large number of random uniform numbers, therefore, we want our RNG to be as fast as possible.
- **Reproducibility:** Many times, we need to regenerate a sequence of random numbers, e.g., when debugging code and checking for errors and being able to work with the same underlying random numbers will be vital.
- **Unpredictability:** Because we want the RNG to be as close to random as possible, we do not want the numbers that it generates to be predictable.
- **Robustness:** The output of an RNG should be robust to choice of operating system, programming language, etc. Otherwise, we will have difficulty creating robust simulations.

sample sequence of random numbers

Random Number Generators: Conclusions

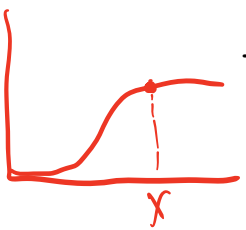
- In practice the work on RNG's has already been done and we can take the algorithms as given.
- As quants we should certainly never feel the need to write our own RNG for uniform random variables.
- Instead we should focus on picking an existing algorithm with desireable properties and transforming our uniform random numbers to our desired distribution.
  - The set of desireable properties on the previous slide can help you choose between existing RNG's.

Random Number Generation: Uniform Distribution

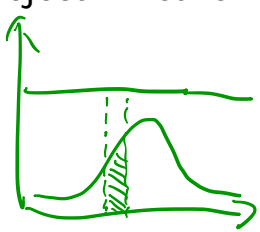
- In this course we take generating random numbers from a uniform distribution as given.
  - The book *Numerical Recipes in C* describes algorithms for implementing uniform(0,1) RNG's for those interested.
- In R, the **runif** function can be used to generate a set of independent uniform random numbers.
- Most programming languages also have functions to generate random variables from other common distribution. As an example, almost all programming languages have a function for generating jointly normal random numbers.
  - In R, this function is called **mvrnorm**.

Techniques for Transforming Uniform Random Numbers

- There are two common methods for transforming uniform random numbers into random numbers from a desired distribution:



  - **Inverse Transform Method**: Enables us to transform our random numbers by inverting the Cumulative Density Function of our desired distribution.
  - **Acceptance-Rejection Method**: Enables us to transform our random numbers by first generating samples from a known distribution and then generating an additional sample that enables us to accept or reject whether the sample is in our desired distribution.



Inverse Transform Technique

- Recall that we are starting with a uniformally distributed number, U, between 0 and 1.
- Clearly, the CDF for any distribution is also defined only on the interval between 0 and 1.
- The key insight behind the inverse transform technique is to interpret U as the point at which the cumulative probaility of our desired distribution matches the random number.
- Put another way, we can think of U as a *random percentile* and notice that this percentile is uniformly distributed.

Inverse Transform Technique

- To be more precise, let X be a random sample from the distribution of our choice. The inverse transform method chooses X such that:

$$X = F^{-1}(U) \tag{2}$$

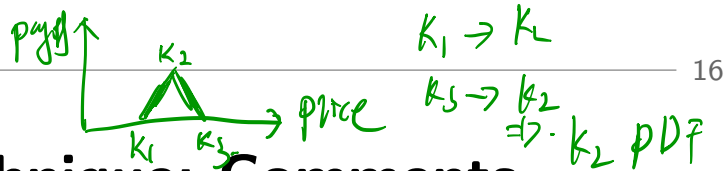
  where $U \sim \mathcal{U}(0,1)$ and F is the CDF of our desired distribution.

- To check this, let's verify that our random sample in fact comes from the desired distribution:

$$\begin{aligned} P(X < x) &= P(F^{-1}(U) < x) && (3)\\ &= P(U < F(x)) && (4)\\ &= F(x) && (5) \end{aligned}$$

CDF ⇒ PDF

- So the inverse transform technique does indeed create samples from our desired distribution.



Inverse Transform Technique: Comments

- We can apply the inverse transform technique to any distribution with a known CDF.
- In many cases, if the PDF is known, we can get the CDF by simply integrating the PDF either analytically or via quadrature methods.
- In other cases, when we know the **characterstic function** but not the PDF, then we can recover the CDF using FFT techniques.
  - For an example of how to do this, see the Digital option pricing example in our FFT lecture, and recall the natural connection between a Digital option and the CDF of its distribution.
- In some cases inverting the CDF might be challenging; in those cases we might prefer other methods.

Inverse Transform Technique Example: Uniform to Uniform

- The simplest possible example of the inverse transform technique is from a uniform(0,1) to a uniform(l, u).
- The pdf of a uniform distribution is:

$$f(x) = \frac{1}{u - l}$$

  if $l \leq x < u$ and 0 otherwise.

- This pdf has a known anti-derivative. Therefore, the cdf is:

$$F(x) = \frac{x - l}{u - l}$$

  if $l \leq x < u$, 0 if $x < l$ and 1 otherwise.

Inverse Transform Technique Example: Uniform to Uniform

- We can use the following equation to invert the CDF:

$$\begin{aligned} U &= F(X) \\ &= \frac{X - l}{u - l} \\ X &= (u - l)U + l \end{aligned}$$

$\frac{x-l}{u-l}$

Inverse Transform Technique Example: Uniform to Exponential Distribution

- The pdf of the exponential distribution is:

$$f(x;\lambda) = \lambda \exp^{-\lambda x}$$

  if $x \geq 0$ and 0 otherwise.

- This pdf has a known anti-derivative. Therefore, the cdf is:

$$F(x;\lambda) = 1 - e^{-\lambda x}$$

- The exponential distribution is the distribution that corresponds to times between jumps in a **poisson process** and has a single model parameter, λ.
- λ controls the frequency of the jumps.

Inverse Transform Technique Example: Uniform to Exponential Distribution

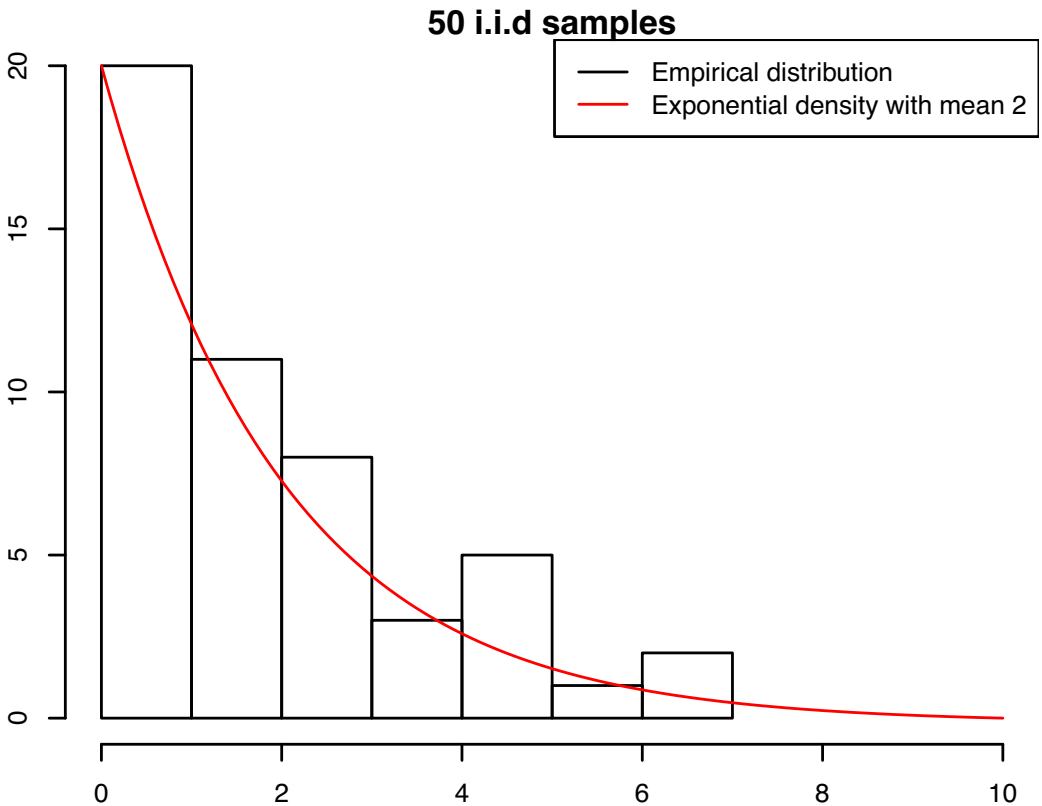
- We can use the following equation to invert the CDF:

$$\begin{aligned} U &= F(X) \\ &= 1 - e^{-\lambda X} \\ 1 - U &= e^{-\lambda X} \\ X &= -\frac{\log(1-U)}{\lambda} \end{aligned}$$

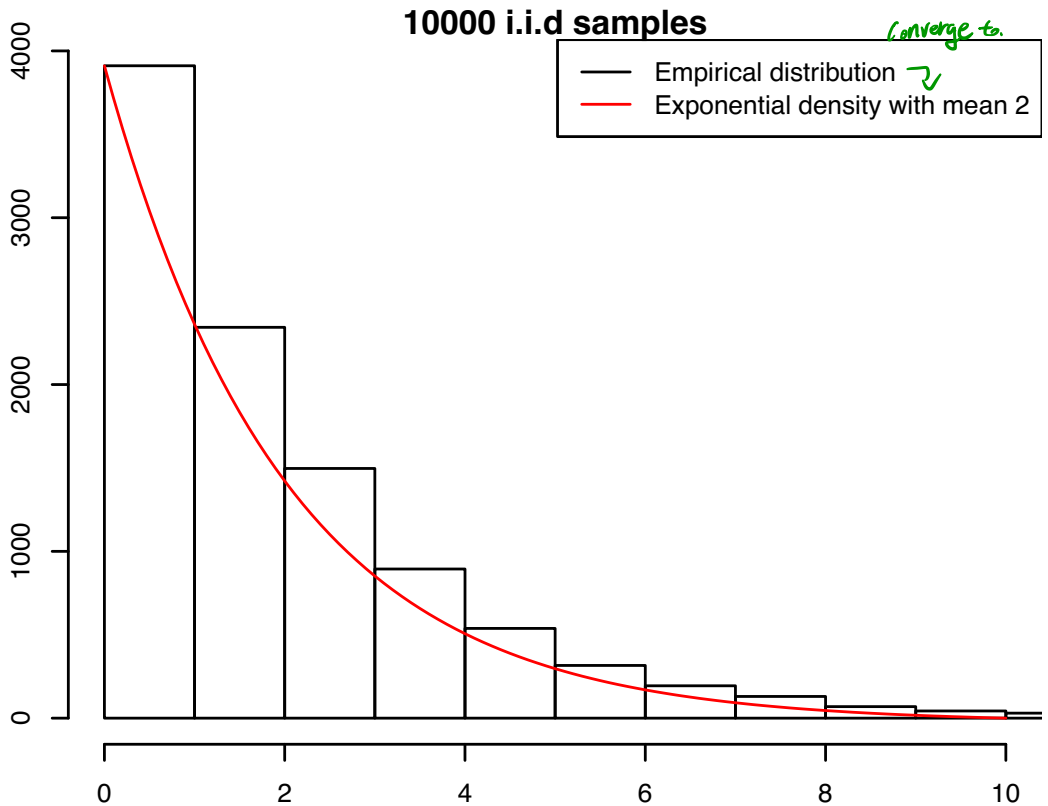
simulate time to default.

exponentials.

Generating exponential random variables: 50 draws



Generating exponential random variables: 10000 draws

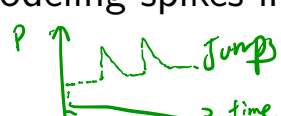


Exponential Distribution & Poisson Processes

- Exponential distributions and Poisson processes have interesting applications in finance, so its worth spending a bit of time talking about them.
- A Poisson Process is a pure jump process that models the number of discrete jumps over a given period.
  - Where might these jumps arise?
    - SDE's with Jump Processes (e.g. the Variance Gamma model)
    - Credit Risk: Modeling defaults of corporations, homeowners, etc.
    - Volatility: Modeling spikes in VIX, or other regime measures.

① Simulate from exponentials
② FFT to capture.

SPY↓



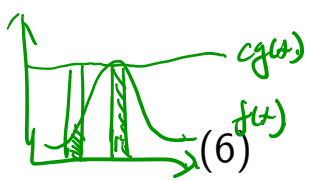
Exponential Distribution & Poisson Processes

- The related exponential distribution measures the time between arrivals in a Poisson process.
- In other words, we can use it to estimate the time-to-default of a company.
  - This is a very common application of simulation in credit modeling.
- We can also use it to model the time between catastrophic financial events.

Acceptance-Rejection Method

- Another common method for generating random samples is the acceptance-rejection method which is ideal when we are trying to sample from some distribution $f(x)$ for which it is difficult to generate random samples directly.
- In this method we begin by specifying some **envelope function**, $g(x)$, that is tractable and easy to generate samples from.
- We choose $g(x)$ such the following property holds:

$$f(x) < cg(x) \quad \forall x \tag{6}$$



  for some constant c.
- To apply this method, we generate a sample from $g(x)$ and accept this sample with probability $\frac{f(x)}{cg(x)}$.
- To accept or reject, we use a $\mathcal{U}(0,1)$ and accept if $u \leq \frac{f(x)}{cg(x)}$

Acceptance-Rejection Method: Pseudocode

```
1   N = 10000
2   samples = rep(0, N)
3
4   j = 1
5   for i = 1,...,N
6       X = GenerateSampleFromGx()
7       U = GenerateRandomUniform()
8
9       f = EvaluateFx(X)
10      g = EvaluateGx(X)
11      p = (f/cg)
12
13      if (U < p)
14          samples(j) = X
15          j = j + 1
16      end if
17  end for loop
```

Acceptance-Rejection Method

- So to generate a random sample using this method we need to generate two random numbers: one from our envelope function and another that is $\mathcal{U}(0,1)$, which is used for the accept-reject decision.
- Note that each draw we reject is completely discarded and not included in our simulation. Therefore, when using this method we should choose our envelope function such that as many samples as possible as accepted..
  - If a large portion of the samples are rejected, then this method will be inefficient.
- This means choosing c in (6) such that we maximize the portion of samples that are accepted.

Acceptance-Rejection Method Example: Standard Normal Distribution

- The PDF for the standard normal distribution is:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{7}$$

- By symmetry, if Z is a standard normal random variable we can then choose to work with $|Z|$, in which case $f(x)$ becomes:

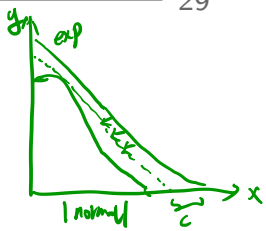
$$f(x) = \frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{8}$$

- We can then define our envelope function, $g(x)$, as having an exponential density with unit mean:

$$g(x) = e^{-x} \tag{9}$$

- Recall that we can easily sample from the exponential distribution using the inverse transform technique, but not from the normal.

Acceptance-Rejection Method Example: Standard Normal Distribution



- The next step is to choose c such that $f(x) < cg(x)$. To do this, let's start by defining $h(x)$ as:

$$h(x) = \frac{f(x)}{g(x)} = \frac{2}{\sqrt{2\pi}} e^{x - \frac{x^2}{2}} \tag{10}$$

- We need to find an upper bound for $h(x)$.
- To that end, we maximize $h(x)$ and set $c = h(x^*)$, where x^* is the maximizer of $h(x)$.
- The exponent in (10) reaches a maximum at $x = 1$. Therefore, we can set $c = \sqrt{\frac{2e}{\pi}}$, which then makes

$$h(x) = e^{-\frac{(x-1)^2}{2}} \tag{11}$$

Acceptance-Rejection Method Example: Standard Normal Distribution

- Note that in this case we needed to generate 3 random samples in order to generate a single standard normal:
  - An exponential random sample
  - A $\mathcal{U}(0,1)$ to determine whether to accept or reject the exponential sample.
  - A $\mathcal{U}(0,1)$ to determine the sign of the standard normal sample.

Acceptance-Rejection Method Example: Standard Normal Distribution

Putting it all together, here are the steps we need to take in order to generate a standard normal via acceptance-rejection.

1. Generate X from an exponential distribution via the inverse transform technique.
2. Generate u_1 from $\mathcal{U}(0, 1)$
3. Accept the sample if $u_1 < e^{-\frac{(X-1)^2}{2}}$. Set $|Z| = X$. Otherwise reject and go back to the beginning.
4. Generate u_2 from $\mathcal{U}(0, 1)$. If $u_2 < 0.5$ then $Z = |Z|$. Otherwise set $Z = -|Z|$.

Generating Standard Normal Random Variables

- The acceptance-rejection method is one of many methods for generating draws from a standard normal distribution.
- Other methods include:
  - Rational Approximation Method
  - Box-Muller Method
  - Marsaglia's Polar Method
- We won't discuss these methods in class, and in practice we can rely on pre-built algorithms to apply these methods, however, they are documented in Hirsa's book for those interested.

Generating Standard Normal Random Variables

- We can easily transform standard normal random variables to normal random variables with mean μ and standard deviation σ using the following equation:

$$X = \mu + \sigma Z \tag{12}$$

  where $Z \sim \mathcal{N}(0, 1)$ and therefore, $X \sim \mathcal{N}(\mu, \sigma)$

- This equation helps us transform a 1D standard normal random variable into a 1D normal random variable. But what if want to transform to draws from a **joint normal distribution**? need to correlated them, (risk analysis)

Random Number Generation: Simulating Multivariate Normals

- At this point we know how to generate uncorrelated standard normals.
- This helps us to generate a full path of dW's as each increment is independent. It does not help us simulate the path of multiple correlated assets.
  - One example would be stochastic volatility models (e.g. Heston) which may have correlation between volatility and the asset itself.
  - Another example would be anytime that we had more than one underlying asset.
- In order to do this, we need to know how to transform a series of independent standard normal random variables into a series of correlated random variables that follow a joint normal distribution.

Random Number Generation: Simulating Multivariate Normals

- Let's start with the case of 2 correlated normals with means μ_1 and μ_2, variances σ_1^2 and σ_2^2 and correlation ρ.
- First, we generate two uncorrelated standard normals, Z_1 and Z_2.
- We can generate the 1^{st} sample the same as before, that is:

$$x_1 = \mu_1 + \sigma_1 Z_1 \tag{13}$$

- For the 2^{nd} sample, we need to ensure the correlation with x_1 is ρ. So it needs to be part Z_1 and part Z_2.
- In particular, if we choose: $p(x_2|x_1)\,p(x_1)$

$$x_2 = \mu_2 + \sigma_2 \left(\rho Z_1 + \sqrt{1-\rho^2} Z_2\right) \tag{14}$$

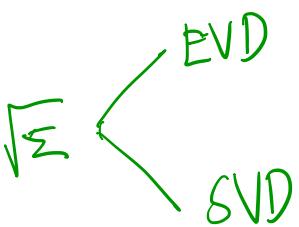
then we can see that x_2 has the desired mean and variance properties, as well as the appropriate correlation to x_1.

Random Number Generation: Simulating Multivariate Normals

- If we generalize this to the case with N correlated normal random variables, we will have:

$$X = \mu + \sqrt{\Sigma} Z \tag{15}$$

where μ is a vector of means and Σ is the covariance matrix of the underlying correlated normals.



Simulating Multivariate Normals: Computing the Square Root of the Covariance Matrix

Note that this implies that creating standard normals requires calculating the square root of a matrix. The two most common methods for doing so are:

- Cholesky Decomposition: requires covariance matrix to be positive definite. need full rank
- Eigenvalue Decomposition: only requires covariance matrix to be positive semi-definite. 0 but non negative eigen values. (bad data : miss matched)
- As a result, if the covariance matrix we are working with is not full rank, Eigenvalue Decomposition is the preffered method.

Quasi Random Number Generation & Low-Discrepancy Sequences (put notes efficiently and, if time, convergence)

- Quasi-random numbers are an alternative to pseudorandom numbers.
- The premise is that they attempt to sample more efficiently than uniformly distributed, uncorrelated random points.
- Remember that simulation methods randomly sample points from some space in order to calculate some expectation or integral.
- But if all we are doing is trying to compute some underlying integral or expectation, then why choose random nodes instead of placing the nodes in some optimal spaces?
  - For example, if we are simulating a $\mathcal{U}(0,1)$, a quasi-random sequence might observe that most of the previous numbers had been below one-half, for example, and give more weight to numbers above one-half for the rest of the sample.

Sobol Sequences

- Sobol Sequences are perhaps the most common quasi-random number / low-discrepancy sequence.
- Intuitively, we can think of them doing this efficiently by choosing nodes intelligently.
  - Recall how the impact of our node placement had a significant impact on the efficiency of our quadrature approximations.
- Most programming languages have packages that include the capability to generate a Sobol sequence.
- We won't get into the internals of these algorithms in this class, but they are worth knowing about as they can help make our simulations more efficient.

Summary

- The main building block of any simulation algorithm is a set of random uniform numbers which we must then transform to more realistic distributions.
- We now know how to generate these numbers and transform them into some of the most common distributions in finance.
- In particular, we learned how to simulate single and multivariate normal random variables. These will be the key to simulating any process that contains one or more underlying Brownian Motions.
- We also saw how to simulate times to default for a process with a constant default intensity by generating samples from an exponential distribution.

Summary

- Once we have these random numbers, we need to know how to:
  - Turn a series of random numbers into a path for the asset(s). This often involves discretizing the SDE in cases when the final pdf is unknown.
  - Calculate and average the payoffs across all paths.
- In the next lecture, we will discuss these steps along with the practicalities of simulating some of the more commonly used SDE's