Section 3: Markov Chain simulations

PSTAT 176/276: Advanced Mathematical Finance

Cosmin Borsa

April 13, 2024

The following section content is provided courtesy of Professor Mike Ludkovski.

Constructing a matrix with all 256 Head/Tail scenarios for N=8

- The code below demonstrates how to enumerate all the 2^N scenarios in a coin toss probability space with N tosses. Below '1' represents Heads and '0' Tails.

```r
w <- expand.grid(rep(list(0:1), 8))
w[111,1:8]
```

```
    Var1 Var2 Var3 Var4 Var5 Var6 Var7 Var8
111    0    1    1    1    0    1    1    0
```

- As illustration, in scenario no. 111, there are 5 Heads and 3 Tails

```r
sum(w[111,] == 1)  # Heads
```

```
[1] 5
```

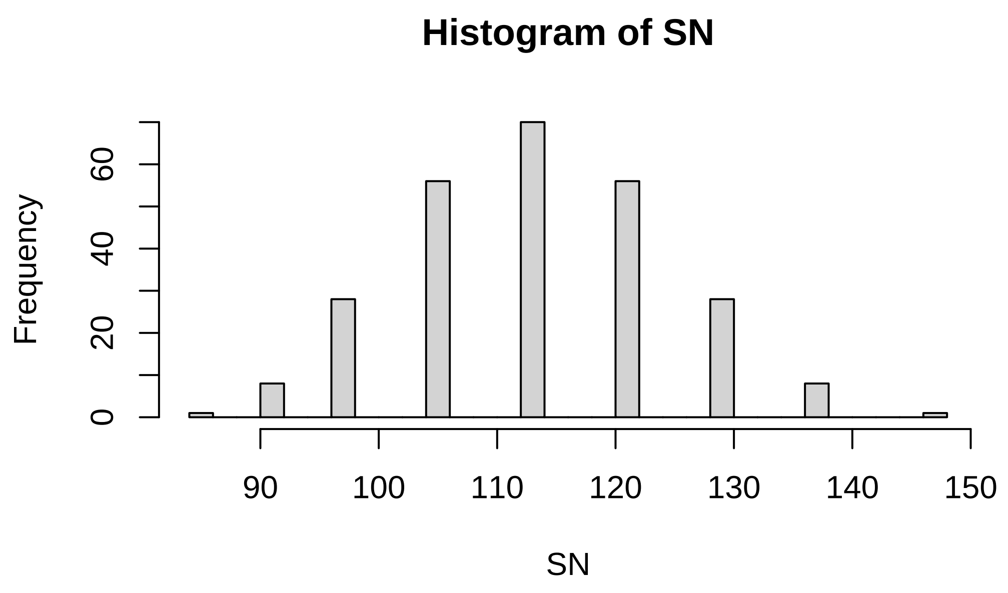
```r
sum(w[111,] == 0)  # Tails
```

```
[1] 3
```

Example computations with above

We can now compute for example S_N and histogram it (i.e fix N, vary omegas)

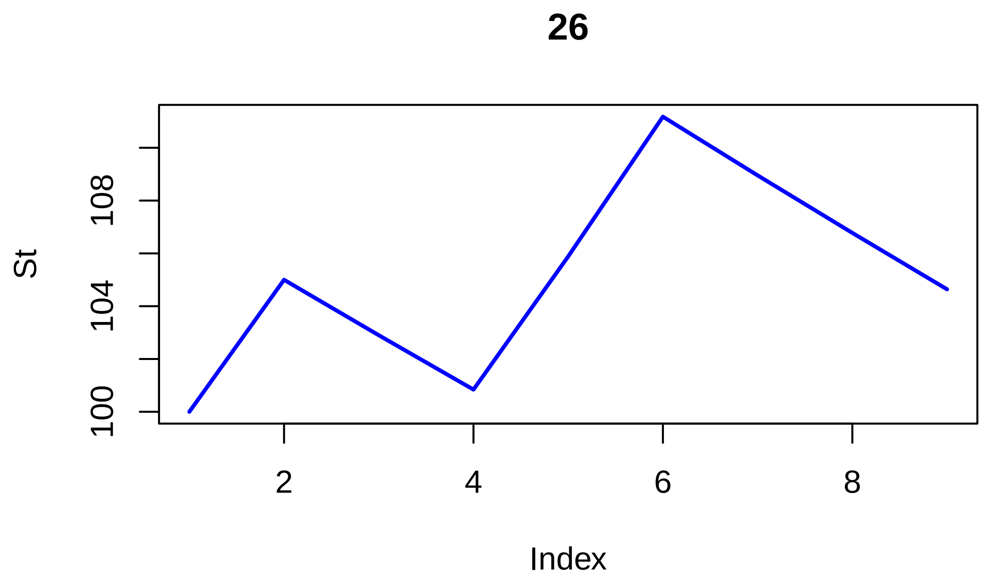
```r
u <- 1.05; d <- 0.98; S0 <- 100
SN <- rep(0, 256)
for (i in 1:256) {
  numH <- sum(w[i,] == 1)
  numT <- sum(w[i,] == 0)

  SN[i] <- S0*u^numH*d^numT
}
hist(SN, 30)
```



Note that to compute expectations, you will also have to compute the (risk-neutral) probability of each scenario, $\mathbb{Q}(\omega)$.

Or we can plot a particular trajectory of S_t along a scenario (ie fix ω, vary t)

```r
St <- rep(0, 9)
scen <- round(runif(1)*256)
St[1] <- S0
for (j in 1:8) {
  St[j+1] <- St[j]*ifelse(w[scen,j]==1,u,d)  # multiply by u (if YES), or by d (if NO)
}
plot(St, lwd=2, type="l", col="blue", main=scen )  # 5 ups, 3 downs
```



Note that in the last loop, St[1] is actually the initial value, namely S_0; `St[2]` is the value after the first coin toss, i.e S_1. Therefore the last value, S_8 is stored in `St[9]`.

Probability measures

- In the binomial tree our sample space is fixed at Ω which has 2^N scenarios.
- There are LOTS of different probability measures on this sample space.
- The main example are binomial probability measures that have a fixed probability a of Heads and therefore
- $\mathbb{P}(\omega_1 \dots \omega_N) = a^{numH(\omega)}(1-a)^{numT(\omega)}$
- It can be verified that this gives a probability measure for any $a \in (0,1)$
- Notationally, we identify $\mathbb{P}$ (a function on the sample space mapping scenarios to numbers between 0 and 1) with its probability of an up-move a

---

FTAPs

Theoretically the first module of the course is about the Fundamental Theorems of Asset Pricing.

Define: $\mathbb{Q}$ is a *martingale measure*, if $\mathbb{E}^{\mathbb{Q}}_n[S_{n+1}] = (1+r)S_n$

- For any portfolio X, $(X_n/(1+r)^n)$ is a $\mathbb{Q}$-martingale
- $X_0 = \mathbb{E}^{\mathbb{Q}}[\frac{X_N}{(1+r)^N}]$
- **FTAP 1**: There is a martingale measure if and only if the model admits no-arbitrage
- Existence of $\mathbb{Q}$ rules out arbitrage, i.e. the possibility that $X_0 = 0$ and $X_N \geq 0$ with a strictly positive wealth in at least one scenario. Indeed arbitrage is impossible: the latter statement implies that

$$\mathbb{E}^{\mathbb{Q}}[\frac{X_N}{(1+r)^N}] = \sum_{\omega} \mathbb{Q}(\omega) X_N(\omega)(1+r)^{-N} > 0 = X_0,$$

  because the random variable inside the expectation is non-negative and sometimes strictly positive.
- The converse is harder to prove: no-arbitrage implies existence of $\mathbb{Q}$
- In the binomial tree, we have a formula for the coefficients $q_n(\omega_1 \dots \omega_n)$ which yields the corresponding martingale measure $\mathbb{Q}$. For the formula to work, it reduces to the requirement that $u_n > 1 + r_n > d_n$ in all sub-trees
- if above condition does not hold, then q will not be in (0,1) and so $\mathbb{Q}$ would not be well-defined.

FTAP 2: Risk-neutral pricing

- If X replicates some payoff V, ie $X_N = V_N$ in all scenarios, then by the law of one price, $X_0 = V_0$ which combined with above implies

$$V_0 = \mathbb{E}^{\mathbb{Q}}[\frac{V_N}{(1+r)^N}].$$

- More generally, if X super-replicates V, i.e $X_N(\omega) \geq V_N(\omega)$ in ALL scenarios, then $X_0 \geq V_0$.
- Super-replication price is the lowest upper bound:

$$\bar{V}_0 = \min_{X_N \text{ super-repl. } V_N} X_0.$$

- Sub-replication price is the greatest lower bound:

$$\underline{V}_0 = \max_{X_N \text{ sub-repl. } V_N} X_0.$$

- FTAP 2: All fair values of V_0 must lie in this range: $\underline{V}_0 \le V_0 \le \bar{V}_0$
- A replicating portfolio both super- and sub-replicates which gives $\underline{V} = \bar{V} = X_0$ – the range collapses to a **unique** no-arbitrage price for V

State Price Density

- By definition the discounted asset price is a $\mathbb{Q}$-martingale. This means on “average” grows at the risk-free rate.
- This does **NOT** make sense in real-life: assets are risky, so investors must be compensated for taking on risk by having additional return
- In other words, $\mathbb{E}^P_n[S_{n+1}] > (1+r)S_n$ (under the **real** probability P, (S_n) is a sub-martingale)
- How to reconcile real-world P with the pricing mechanism we derived above through replication arguments?

Arrow-Debreu Securities: Building Blocks of Finance

- Consider a contract which pays $1 if the scenario is all Heads: HHH…H, and nothing otherwise (we’re back in the binomial tree with constant q and p and $N = 8$ periods). Also assume zero interest rates for now.
- In real-world, this contract will pay with probability p^N, so its expected payoff is $\mathbb{E}[V_N] = \sum_\omega P(\omega)V_N(\omega) = p^N$ since the sum has a single non-zero term (the very first one corresponding to HHHHHHHH)
- However, its no-arbitrage price is $V_0 = \sum_\omega \mathbb{Q}(\omega)V_N(\omega) = q^N$.
- How to explain this?

```
u <- 1.05; d <- 0.98; r <- 0; S0 <- 100; N <- 8
p <- 0.6;
q <- (1+r-d)/(u-d)
q^N/(1+r)^N
```

```
[1] 4.440743e-05
```

```
p^N/(1+r)^N
```

```
[1] 0.01679616
```

Not All Scenarios Are Created Equal

- Scenario HHHHHHHH is less “important” under $\mathbb{Q}$ than under P
- Getting paid in that scenario is less rewarding, so worth less relative to its likelihood
- Let us define the ratio of the likelihoods for each scenario: $Z(\omega) := \frac{\mathbb{Q}(\omega)}{P(\omega)}$
- Known as the Radon-Nikodym derivative between $\mathbb{Q}$ and P

```
ZN <- rep(0, 256)
for (i in 1:256) {
  numH <- sum(w[i,] == 1)
  numT <- sum(w[i,] == 0)

  ZN[i] <- (q/p)^numH*((1-q)/(1-p))^numT
}
```

- For scenario 111 again:

```
w[111,1:8]
```

```
    Var1 Var2 Var3 Var4 Var5 Var6 Var7 Var8
111    0    1    1    1    0    1    1    0
```

```
ZN[111]
```

```
[1] 0.1394246
```

- For $\omega = HHHHHHHH$, scenario probability is 0.0167962 but worth only 4.4407431×10^{-5}
- We say that the state price density at that scenario is 0.0026439
- Useful facts: $Z_N(\omega) \geq 0$, $\mathbb{E}[Z_N] = 1$
- Using conditional expectation, can extend Z_N to the Radon-Nikodym process (Z_n):

$$Z_n = \mathbb{E}_n[Z_N].$$

  In other words $Z_n(\omega_1 \dots \omega_n) = pZ_{n+1}(\omega_1 \dots \omega_n H) + (1-p)Z_{n+1}(\omega_1 \dots \omega_n T)$, defined recursively in the tree.
- Always have $\mathbb{E}[Z_n] = 1$ for all n. $Z_0 = 1$. In fact, (Z_n) is a martingale under P.

Pricing under P

- For the plain binomial tree $Z_N(\omega) = \frac{q}{p}^{numH(\omega)} \frac{(1-q)}{(1-p)}^{numT(\omega)}$
- Let $\xi_n := Z_n/(1+r)^n$ – incorporate discounting
- Now can use ξ to do "normal" discounted payoff under P:

$$V_0 = \mathbb{E}^P[\xi_N V_N] = \sum_\omega P(\omega)\xi_N(\omega)V_N(\omega)$$

- *multiply* each scenario payoff by its state-price density
- Compute ξ_N once and for all. Then can consistently price any contract using physical-P probabilities
- If the model is scenario-dependent (varying r, u, d), that will be reflected in ξ [but nothing to do with P!]
- Can use as a calibration procedure to "real-world": $\xi_N(\omega)$ is the multiplier needed to price the Arrow-Debreu security that pays 1 dollar in scenario ω – can learn ξ_N (and hence eventually $\mathbb{Q}$) from surveying observed prices

---

- State prices are related to the risk premium in finance and the risk loading in insurance
- **Naming**: we distinguish Z_N which is the *Radon-Nikodym derivative* between $\mathbb{Q}$ and P
- ξ_N which is the *State Price Density* (includes discounting)

Multiple martingale measures

- There can be multiple martingale measures
- Example: **trinomial** tree. Trinomial probability measures are parameterized by *two* parameters (p^u, p^m) – prob of up- and middle-move.
- Suppose up-move is from S to uS; mid-move is from S to mS and down-move is from S to dS.
- A trinomial measure $Q(q^u, q^m)$ is a martingale measure if $q^u u + q^m m + (1 - q^u - q^m)d = (1 + r)$

- One equation for two unknowns + the constraints $q^u \in (0,1), q^m \in (0,1), 1 - q^u - q^m \in (0,1)$.
- Yields a continuum of feasible Q's
- From above, any value of the form $\mathbb{E}^Q\left[\frac{V_N}{(1+r)^N}\right]$ for **some** Q, would appear to be a fair price for V.
- No longer a unique answer/price!

FTAP part 3

- Is this consistent with FTAP part 2?
- The answer is YES (the proof is long): the range of $\mathbb{E}^Q\left[\frac{V_N}{(1+r)^N}\right]$ as Q changes is *exactly* the above interval $[\underline{V}, \bar{V}]$

$$\min_{Q-mart} \mathbb{E}^Q\left[\frac{V_N}{(1+r)^N}\right] = \max_{X_N \text{ sub-repl. } V_N} X_0$$

$$\max_{Q-mart} \mathbb{E}^Q\left[\frac{V_N}{(1+r)^N}\right] = \min_{X_N \text{ super-repl. } V_N} X_0$$

- Note that for any portfolio X, the martingale property is the same irrespective of what martingale measure we choose: so for any Q^1, Q^2 we have

$$\mathbb{E}^{Q_1}\left[\frac{X_N}{(1+r)^N}\right] = X_0 = \mathbb{E}^{Q_2}\left[\frac{X_N}{(1+r)^N}\right]$$

- So the risk-neutral pricing formula is still consistent across different Q's for any *actual* portfolio.
- Hence, if V can be replicated, i.e $V_N = X_N$ for some portfolio X then V must have a unique no-arbitrage price, namely X_0. This is consistent with part 2 above.
- Conversely, if V cannot be replicated then we must have a **range** of no-arbitrage prices.

- Take-away: lots of prices can be consistent with no-arbitrage and hence be "fair"
- In the trinomial tree to solve the replication problem leads to 3 equations in 2 unknowns – usually no solutions. I.e cannot replicate V.
- By above get a range of fair values for price of V, which is consistent with having a continuum of martingale measures Q
- Replicating portfolio exists <-> **Unique** no-arbitrage price
- Multiple no-arbitrage prices <-> no replicating portfolio

Complete Markets

- **Complete** market: can replicate *ANY* payoff
- Example: binomial model is a complete market (always have 2 equations in 2 unknowns, the two equations are linearly independent, hence guaranteed a solution)
- **FTAP 3b**: the market is complete *IF AND ONLY IF* there is a unique martingale measure Q
- Check: in the trinomial trees there are many Q's, so cannot replicate everything and expect to see a range of prices (we already knew this)
- Real-world markets are **incomplete**.

Markov Property

- In general, contract value right now is $V_n(\omega_1 \dots \omega_n)$. Should be thought of as a vector indexed by the scenario.
- If there are many scenarios, there are many values to store.
- Hard to visualize scenarios in aggregate
- Solution: convert random variables into **functions**. Instead of $V_n(\omega)$ we shall write $v_n(S_n)$
- Captures the idea that option contract is a function of the underlying. This feature works if: (i) stock prices form a Markov chain (no memory); (ii) contract payoff is path-independent and depends only on S_N.
- True for Calls, Puts, digitals,...
- The conversion allows for example to *graph* $v_n(s)$ – eg Call price is increasing in underlying. This is how it was all done in PSTAT 170
- Reduce from 2^n scenarios in Ω to $n+1$ function values over $\mathbb{R}$
- Replace probabilistic recursion

  $$V_n(\omega_1 \dots \omega_n) = qV_{n+1}(\omega_1 \dots \omega_n H) + (1-q)V_{n+1}(\omega_1 \dots \omega_n T)$$

  with a functional recursion:

  $$v_n(s) = qv_{n+1}(us) + (1-q)v_{n+1}(ds)$$

- The above would not work for path-dependent contracts which have some memory about the scenario so far, beyond what is "saved" in S_n
- Try to find *sufficient statistics* to recover the Markov property. Eg for Asian option want to store the sum $\sum_k S_k$. Give it a name Y and then expand the inputs of v to become $v_n(s, y)$ – see textbook
- For barrier options, can store whether hit the barrier or not $b \in \{0, 1\}$. For lookback option can store the maximum. For box options can store ......

Conclusion

- Different parts of FTAP cover the role of martingale measures for pricing/replication of contracts
- Theory extends nicely to **many assets** and **continuous time**
- Can think of pricing probabilistically (scenario by scenario) or functionally (as a function of sufficient statistics with a recursion formula)
- These are the two pricing paradigms: Monte Carlo and PDEs
- Read Chapter 1, 2, 3.1-3.2

Pricing Review

- We consider a binomial tree with 20 periods
- We have constant u,r,d and a European Call with strike $K = 103$
- There are 2^{20} scenarios; using the Markov property the payoff is a function of S_T so only 21 different payoffs
- So we could easily get the exact answer. But we will now try a different strategy
- Will use **simulation**, which is a scalable method that can handle much more complicated pricing problems.

Mean —> Average

- Let's apply Monte Carlo that you've seen in PSTAT 160A
- Sample a few scenarios and compute the **empirical average**
- Idea is to approximate risk-neutral pricing $\mathbb{E}^Q[\frac{V_N}{(1+r)^N}]$ = mean by the average:

$$\frac{1}{\sum_{m=1}^M Q(\omega^m)} \sum_{m=1}^M Q(\omega^m) V_N(\omega^m) \frac{1}{(1+r)^N}$$

- Scenarios selected are $\omega^m, m = 1, \ldots, M$, sampled uniformly from the 1048576 available scenarios
- $\frac{1}{\sum_{m=1}^M Q(\omega^m)}$ is the normalizing constant, the total "weight" of the scenarios picked
- still use the **q's** to give them the correct weight; also don't forget discounting

```r
S0 <- 100
u <- 1.01
d <- 0.99
r <- 0.002
N <- 20
K <- 103

q <- (1+r-d)/(u-d)
```

- Risk neutral q=0.6

Sample a few scenarios

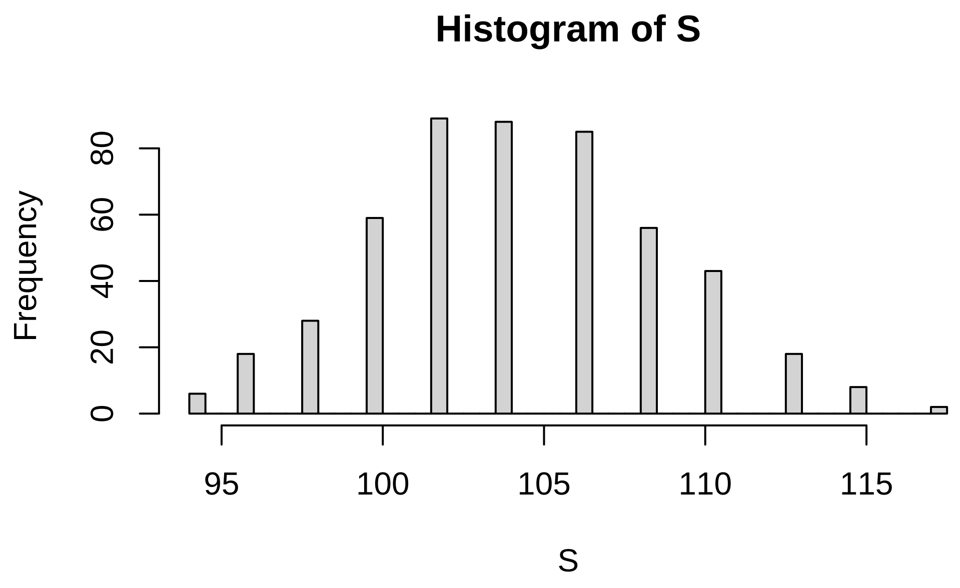
```r
M <- 500
SN <- qProb <- Payoff1 <- rep(0,M)
for (i in 1:M) {
   ndx <- sample.int( 2^N, 1)
   scen <- as.integer(rev(intToBits(ndx)[1:N]))  # scenario Heads/Tail sequence
   numH <- sum(scen == 1)
   numT <- sum(scen == 0)

   SN[i] <- S0*u^numH*d^numT
   qProb[i] <- q^numH*(1-q)^numT
   Payoff1[i] <- pmax(SN[i]-K,0)
}
sum( qProb*Payoff1/(1+r)^N )/sum (qProb)
```

```
[1] 2.501231
```

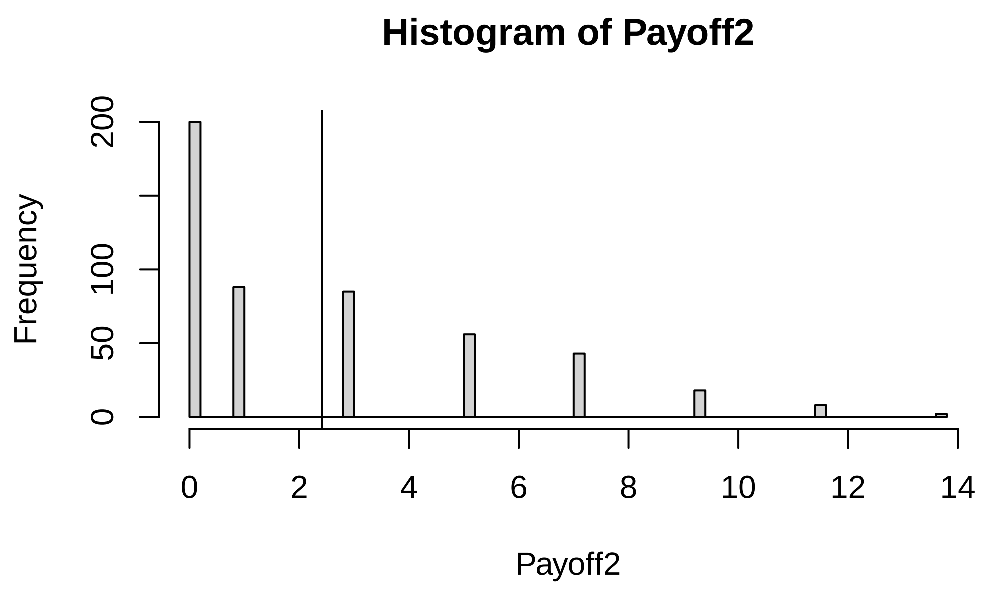
Dynamic Scenario Generation

- An equivalent way is to generate scenarios step-by-step through the coin tosses
- We will directly toss coins using q-probabilities, which will ensure that scenarios with higher Q-weights get picked more often
- (In the first method each outcome is equally likely to be picked, but their q-probabilities are highly nonuniform)
- Recall number of up-moves is Binomial $numH \sim Bin(N, q)$ and $S_N = S_0 \cdot u^{numH} d^{N-numH} = S_0(u/d)^{numH} d^N$

```
M <- 500
U <- runif(M)
S <- Payoff2 <- array(0, dim=c(M,1))

for (j in 1:M) {
  numH <- rbinom(1,N,q)
  S[j] <- S0*(u/d)^numH*d^N
  VN <- pmax(S[j]-K,0)  # Call option
  #S[j] <- S0*(u/d)^qbinom( U[j], N, q)*d^N
  Payoff2[j] <- 1/(1+r)^N*VN  # Discounted payoff to be averaged
}
hist(S, 50)
```



```
hist(Payoff2, 50)
abline(v=mean(Payoff2))
```



```
mean(Payoff2)
```

```
[1] 2.413035
```

```
sd(Payoff2)
```

```
[1] 2.990371
```

```
mean(Payoff2)-1.96*sd(Payoff2)/sqrt(M)
```

```
[1] 2.150917
```

```
mean(Payoff2)+1.96*sd(Payoff2)/sqrt(M)
```

```
[1] 2.675152
```

- Above we histogram the terminal values $S_N(\omega^m)$ across the $M = 500$ random scenarios
- Note that now scenarios are about the random variable S_N which under Q is $Bin(N, q)$-distributed
- The above is Monte Carlo applied to the Markovian reprsentation $V_0 = \mathbb{E}^Q[\frac{f(S_N}{(1+r)^N}]$ where $f(s)$ is the option payoff – now viewed as a function of the r.v. S_N.
- Empirical average is 2.4130347 – this is our **approximation** $\bar{V}$ for option value

---

Monte Carlo error

- The above is an **estimate** . Need *error bars* around the answer to understand how confident we are in our estimate
- Recall the idea of standard error of an estimator from your statistics classes
- The analogy is: true mean=true price of the contract; samples=scenario payoffs; estimator=scenario sample average.
- Have an i.i.d average of M terms. Variance of the sample average $\bar{V}$ is σ^2/M

- i.e a confidence interval for the population mean is

$$[\bar{V} - z\sigma/\sqrt{M}, \bar{V} + z\sigma/\sqrt{M}]$$

- Using *Central limit theorem*, the empirical average has an approximately Gaussian sampling distribution, so z is based on the quantiles of standard normal.
- For example if we want 95% CI: $z = 1.96 = \Phi^{-1}(0.975)$, Φ is the standard Gaussian CDF

Monte Carlo variance

- What is σ^2? This is the *population variance* of the i.i.d samples (for us the variance of the payoff)
- We don't know the true price and certainly not the payoff variance
- Plug-in the *empirical* standard deviation: $\text{StDev} = \text{sd}(\text{Payoff2}) = 2.9903714 \simeq \sigma$
- Answer is [2.1509171, 2.6751523]
- Don't forget to divide by $\sqrt{M}$!

Monte Carlo uncertainty

- The above price estimate is **random** – if we were to re-run the whole experiment a second time, we would get a different answer:

```
M <- 500
U <- runif(M)
S <- Payoff <- array(0, dim=c(M,1))

for (j in 1:M) {
  S[j] <- S0*(u/d)^qbinom( U[j], N, q)*d^N
  Payoff[j] <- 1/(1+r)^N*pmax(S[j]-K, 0)
}
print(c(mean(Payoff)-1.96*sd(Payoff)/sqrt(M), mean(Payoff)+1.96*sd(Payoff)/sqrt(M)))
```

```
[1] 2.062772 2.573459
```

- The confidence interval tells us that based on our samples, with 95% chance the truth should be in that range.
- Note that since samples are i.i.d., standard theory tells us that the empirical mean is unbiased, so our estimate has zero bias, i.e. is centered on the true answer (hence symmetric CI).
- True answers:

```
allS <- S0*(u/d)^(0:N)*d^N
truePrice <- sum( pmax( allS-K,0)*dbinom(0:N,N,q) )/(1+r)^N
truePriceSq <- sum( pmax( allS-K,0)^2*dbinom(0:N,N,q) )/(1+r)^(2*N)
trueSd <- sqrt( truePriceSq - truePrice^2)
# true mean and sd
print( truePrice)
```

```
[1] 2.330635
```

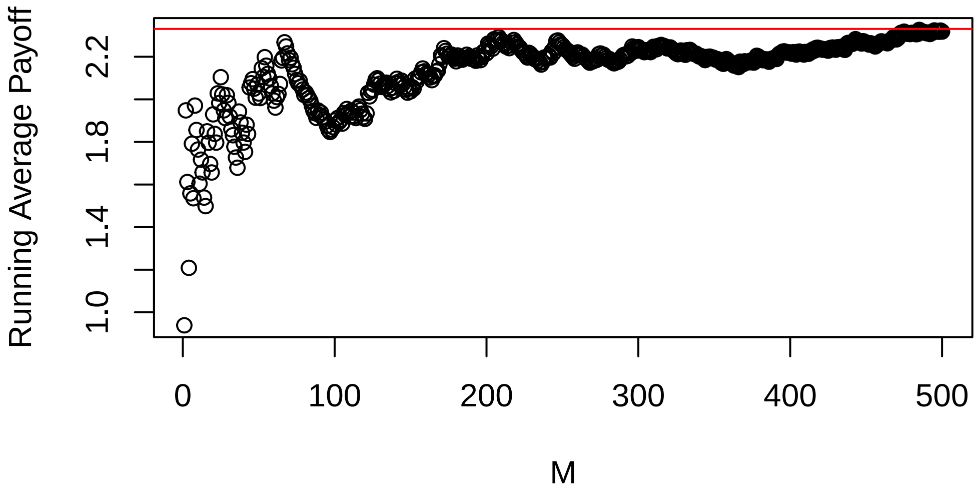
```
print(trueSd)
```

```
[1] 2.901832
```

Number of Monte Carlo samples

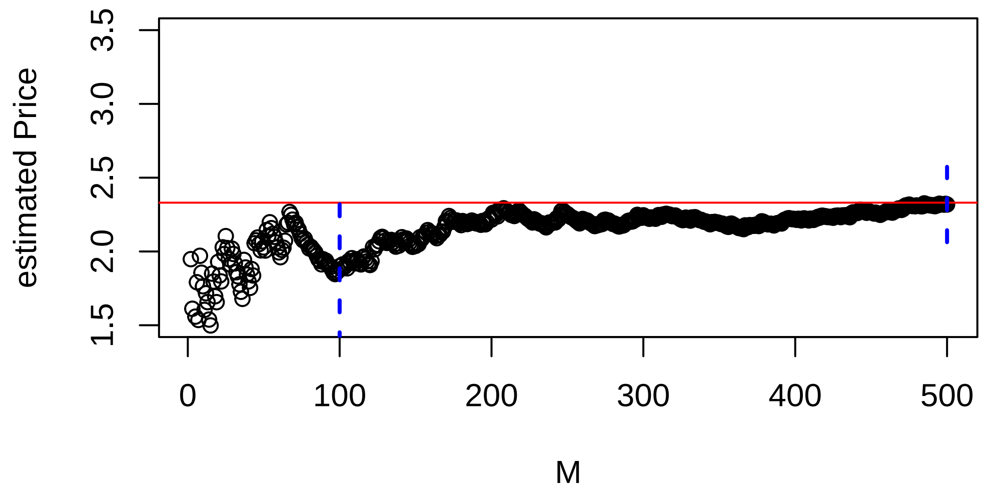
- As the number of samples M grows, the answer becomes more accurate. Confidence interval gets narrower.
- By the Strong Law of Large Numbers, guaranteed to converge to the true risk-neutral expectation
- Convergence rate is $1/\sqrt{M}$
- We can visualize the convergence of the MC estimates to the truth as M increases

```r
plot(cumsum(Payoff)/(1:M), xlab='M', ylab='Running Average Payoff')
abline(h=truePrice,col="red")
```



- And the shrinking CI's:

```r
s500 <- sd( Payoff[1:500])
ci500 <- c(mean(Payoff[1:500])-1.96*s500/sqrt(500),mean(Payoff[1:500])+1.96*s500/sqrt(500))
plot(cumsum(Payoff)/(1:M), ylim=c(1.5,3.5), xlab='M', ylab='estimated Price')
abline(h=truePrice,col="red")
lines( c(500,500), ci500, col="blue", lwd=2, lty=2)

s100 <- sd( Payoff[1:100])
ci100 <- c(mean(Payoff[1:100])-1.96*s100/10,mean(Payoff[1:100])+1.96*s100/10)
lines( c(100,100), ci100, col="blue", lwd=2, lty=2)
```



Some statistical concepts

- Estimator is: $\bar{V} = \frac{1}{M}\sum_{m=1}^{M} f(S_T^{(m)})$ (e.g $f(s) = e^{-rT}(s-K)_+$)
- Estimator MSE: $E[(\bar{V} - E[f(S_T)])^2] = bias^2 +$ variance
- Unbiased due to iid samples: $E[\bar{V}] = E[f(S_T)]$
- Estimator variance $Var(\bar{V}) = \frac{Var(f(S_T))}{M}$
- LLN: $\bar{V}$ is strongly consistent as $M \to \infty$
- CLT: $\bar{V} - E[f(S_T)] \sim \frac{StDev(f(S_T))}{\sqrt{M}} Z$
- Actually have exact expression using the t distribution with $M-1$ degrees of freedom
- Also have statistical information about the sample estimators for $StDev(f(S_T))$

Sampling Uniforms

- When talking about vanilla European options, payoff is a function of S_N, i.e $V_N = v(S_N)$
- Sufficient to just generate samples of $S_N(\omega)$ for representative ω's
- We "flatten" all the scenarios to think about the sample space being $\Omega = [0,1]$. This means that samples are just drawing Uniforms from the unit interval – the basic computer operation.
- Map $u \in [0,1]$ interpreted as an outcome $\omega = u$ to $S_N(u)$ and finally to $v(S_N(u))$.
- Intuitively we reduce to aggregating scenarios by their respective S_N values and sum the Q-probabilities
- Exactly 1 scenario is HHH…HHH – has probability q^N, so if $U \in [0, q^N]$ then $S_N = S_0 u^N$
- Exactly N scenarios that have a single T and the rest H: probability is $Nq^{N-1}(1-q)$ – corresponds to another small interval for U
- This is exactly the **inverse CDF** simulation method you've seen in PSTAT 160:

$$S_N = F_{S_N}^{-1}(U), \qquad U \sim Unif(0,1)$$

- In the binomial tree, the risk-neutral distribution of S_N is related to a Binomial distribution with parameters N, q
- Use the **quantile** function, like qbinom or qnorm to apply inverse CDF of a known family.

```
U <- runif(0,1)
S <- S0*(u/d)^qbinom( U, N, q)*d^N
```

- the number of Heads is binomial; the number of tails is N - numH; qbinom is the quantile function, aka inverse CDF of the Binomial distribution

---

Monte Carlo for Black-Scholes Model

- Can generalize the above idea to any other model for the stock price
- e.g. let us price a Call via Monte Carlo in a **Black-Scholes** model with maturity T (continuous-time)
- S_T is log-normal in the Black-Scholes model. Recall that risk-neutral pricing in B-S is based on

$$S_T = S_0 \exp((r - \sigma^2/2)T + \sigma Z_T), \qquad Z_T \sim N(0,T)$$

- drift of the stock is $r - \sigma^2/2$ **under Q** rather than the true rate of return α (under P)
- discount factor $\exp(-rT)$
- True price is an **integral** against the log-normal density of S_T – replace with an average
- With Monte Carlo, the key step is simulating Z_T using *rnorm*: $Z_T \sim \sqrt{T} \cdot N(0,1)$

```
S0 <- 50
r <- 0.02
T <- 1/2
sigma <- 0.25
K <- 55

M <- 100
U <- runif(M) # standard Gaussian
S <- Payoff <- array(0, dim=c(M,1))

for (j in 1:M) {
    S[j] <- S0*exp( (r-sigma^2/2)*T + sigma*sqrt(T)*qnorm(U[j]))
    Payoff[j] <- exp(-r*T)*pmax(S[j]-K, 0)
}
MC.est <- mean(Payoff)
MC.sd <- sd(Payoff)

c(MC.est - 1.96*MC.sd/sqrt(M), MC.est, MC.est+1.96*MC.sd/sqrt(M))  # conf interval
```

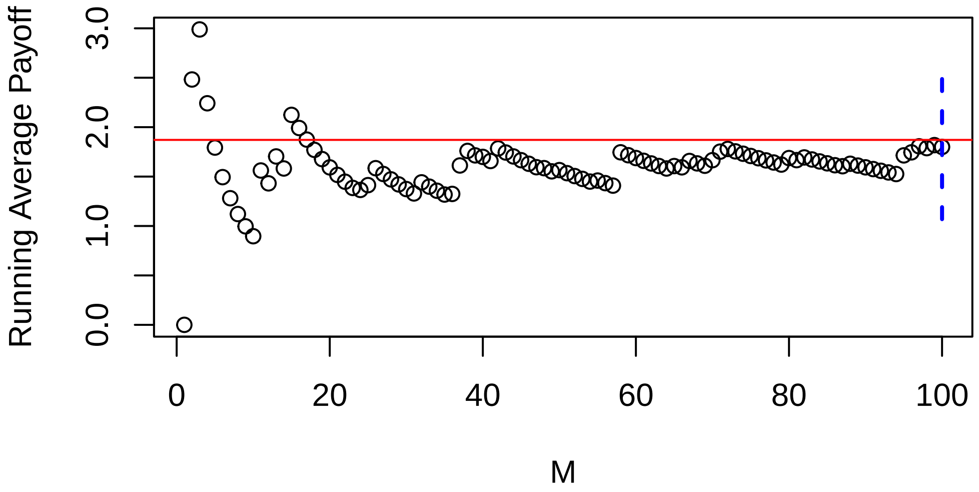
```
[1] 1.068913 1.799829 2.530745
```

- True answer is 1.871 using the Black-Scholes formula

Convergence again as number of simulations grows

```
plot(cumsum(Payoff)/(1:M), xlab='M', ylab='Running Average Payoff')
abline(h=1.871,col="red")

s100 <- sd( Payoff[1:100])
```

```
ci100 <- c(mean(Payoff[1:100])-1.96*s100/10,mean(Payoff[1:100])+1.96*s100/10)
lines( c(100,100), ci100, col="blue", lwd=2, lty=2)
```



Recap of the Monte Carlo framework

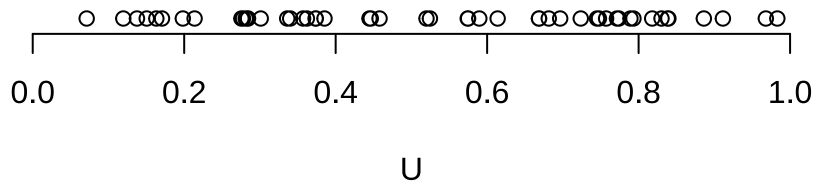
- Monte Carlo approximation consists of replacing a theoretical mean (2^N terms for binomial tree, integral for B-S model) with empirical average based on M simulations
- Sample $\omega^1, \ldots \omega^M$ (with replacement)
- Use empirical mean/empirical SD to compute the estimated option value and confidence interval
- As $M \to \infty$ converge to the true answer (Strong Law of Large Numbers).
- Can handle any model/payoff: just need to be able to simulate V_N.
- Examples: path-dependent options (simulate paths of (S_t))
- Examples: options on multiple stocks (simulate multivariate Brownian Motion)
- Background reference: Glasserman Ch 1, pp. 1-12 (ignore pp 13-19), see GS
- Up next: convergence is slow, so seek ways to speed it up

Improving Monte Carlo

- Monte Carlo tends to be slow. Means that need a large M to have an accurate estimate.
- To improve accuracy from say 5% to 1%, need 25 times more simulations
- In real life will often do millions of simulations
- **Variance reduction** is a key concept – how to shrink the Confidence Interval to increase accuracy.
- Throughout we need to keep in mind bias-variance trade-off.
- Plain Monte Carlo has ZERO bias but high variance

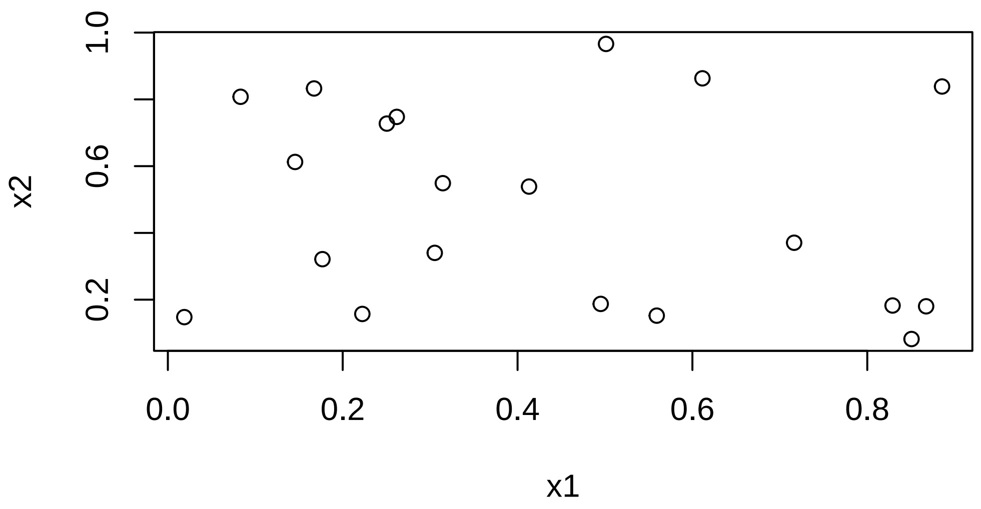
Uniforms are not so uniform

```
U <- runif(50)
plot(U, rep(0,50), bty='n',yaxt='n', xlim=c(0,1), ylab='')
```



Looks even worse in higher dimensions:

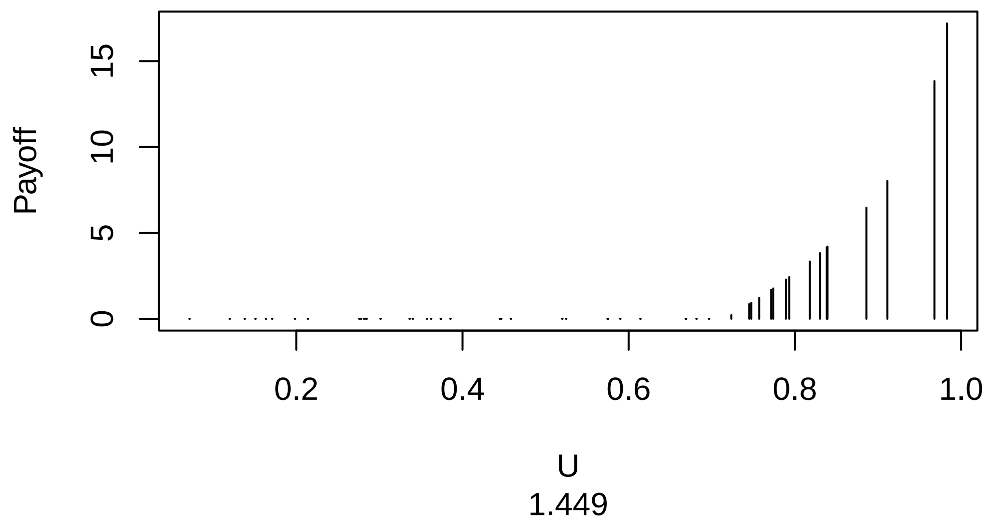
```
x1 <- runif(20)
x2 <- runif(20)
plot(x1, x2)
```



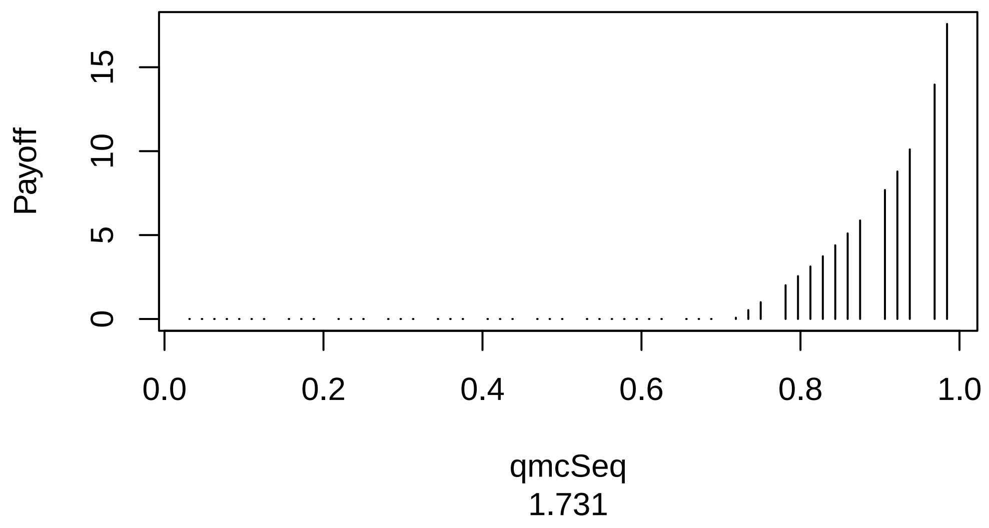
Impact of this “non-uniformity” in pricing a Black-Scholes Call:

```
S <- rep(0,50)
Payoff <- rep(0,50)
S0 <- 50; K <- 55; T<- 0.5; r <- 0.02; sigma <- 0.25
for (j in 1:50) {
  S[j] <- S0*exp( (r-sigma^2/2)*T + sigma*sqrt(T)*qnorm(U[j]))
  Payoff[j] <- exp(-r*T)*pmax(S[j]-K, 0)
}
```

```r
plot(U, Payoff, 'h', sub=round(mean(Payoff),3))
```



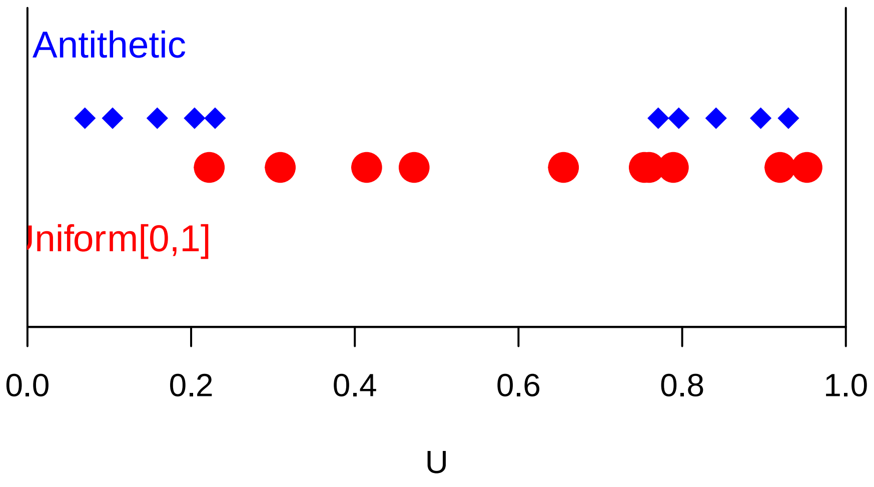
```r
library(randtoolbox)
qmcSeq <- sobol(50,scrambling=1)
for (j in 1:50) {
  S[j] <- S0*exp( (r-sigma^2/2)*T + sigma*sqrt(T)*qnorm(qmcSeq[j]))
  Payoff[j] <- exp(-r*T)*pmax(S[j]-K, 0)
}
plot(qmcSeq, Payoff, 'h', sub=round(mean(Payoff),3))
```



Antithetic MC

- At its basic level MC relies on sampling Uniforms U_i
- Our goal is to make them *more uniform*
- Rather than sample them one-by-one, we will sample them in pairs
- Namely, instead of U_1, U_2, use $U_1, 1 - U_1$
- This introduces dependence but makes sure that the samples are **better spread-out**

```
U <- runif(10)
AT <- runif(5)
AT2 <- 1-AT
plot( U, rep(0,10), pch=19, cex=2, col='red', xlim=c(0,1), ylim=c(-0.3, 0.3),
       xaxs="i", ylab='',yaxt='n',bty='u')
points( c(AT, AT2), rep(0.1,10), pch=18, cex=1.5, col='blue')
text(0.1,y=-0.15,"Uniform[0,1]", col="red", cex=1.2)
text(0.1,y=0.25, "Antithetic", col="blue", cex=1.2)
```



- Antithetic generates "twinned'' simulations
- For the GBM model, simulation is based on drawing Gaussians Z_j
- antithetic is natural: use Z and $-Z$
- Key idea: preserve the marginal (i.e don't alter the identical distribution of the scenarios), modify the joint (i.e drop the independence of scenarios).
- In fact, antithetic tries to induce negative correlation in scenarios which ought to reduce variance
- It's a very *simple* and easy-to-implement variance reduction technique
- Not guaranteed to work. Even when works, the gain might not be so large.

Example of using AT for pricing in a GBM model

```r
# Antithetic for a Black-Scholes model: use Z and -Z for increments
K <- 48

AT.est <- AT.sd <- rep(0,100)

# Repeat 100 times to see the sampling distribution of the estimator
for (Runs in 1:100) {
  M <- 25
  Z <- rnorm(M)
  S <- Payoff <- array(0, dim=c(2*M,1))

  for (j in 1:M) {  # create twins from Z, -Z
    S[2*j] <- S0*exp( (r-sigma^2/2)*T + sigma*sqrt(T)*Z[j])
    S[2*j+1] <- S0*exp( (r-sigma^2/2)*T + sigma*sqrt(T)*(-Z[j]))
    Payoff[2*j] <- exp(-r*T)*pmax(S[2*j]-K, 0)
    Payoff[2*j+1] <- exp(-r*T)*pmax(S[2*j+1]-K, 0)
  }

  AT.est[Runs] <- mean(Payoff)  # point estimate for this run
  AT.sd[Runs] <- sd(Payoff)
}
mean(AT.est)
```

```
[1] 4.829736
```

```r
sd(AT.est)
```

```
[1] 0.6620041
```

```r
# Sampling distribution of the naive MC
# to see how much larger the variance is
REG.est <- REG.sd <- rep(0,100)

for (Runs in 1:100) {
M <- 50
Z <- rnorm(50)
S <- Payoff <- array(0, dim=c(M,1))

for (j in 1:M) {
  S[j] <- S0*exp( (r-sigma^2/2)*T + sigma*sqrt(T)*Z[j])
  Payoff[j] <- exp(-r*T)*pmax(S[j]-K, 0)
}

REG.est[Runs] <- mean(Payoff)
REG.sd[Runs] <- sd(Payoff)
}
mean(REG.est)
```

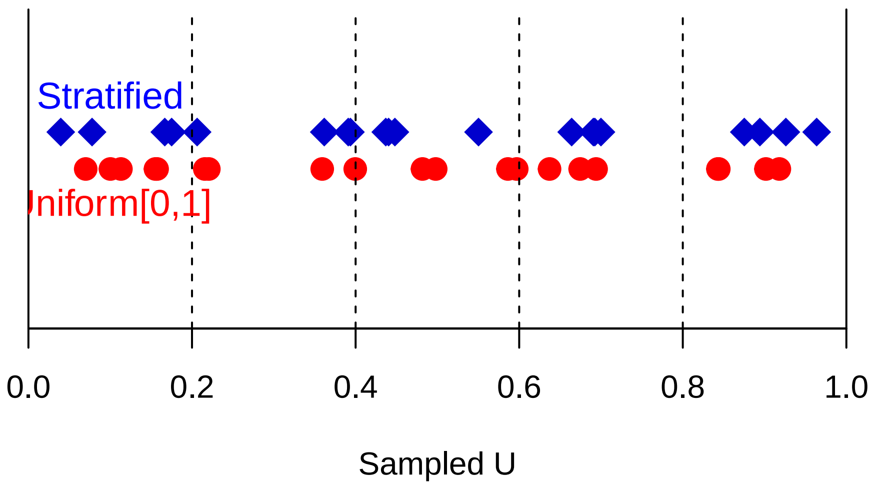
```
[1] 4.821549
```

```r
sd(REG.est)
```

```
[1] 0.957178
```

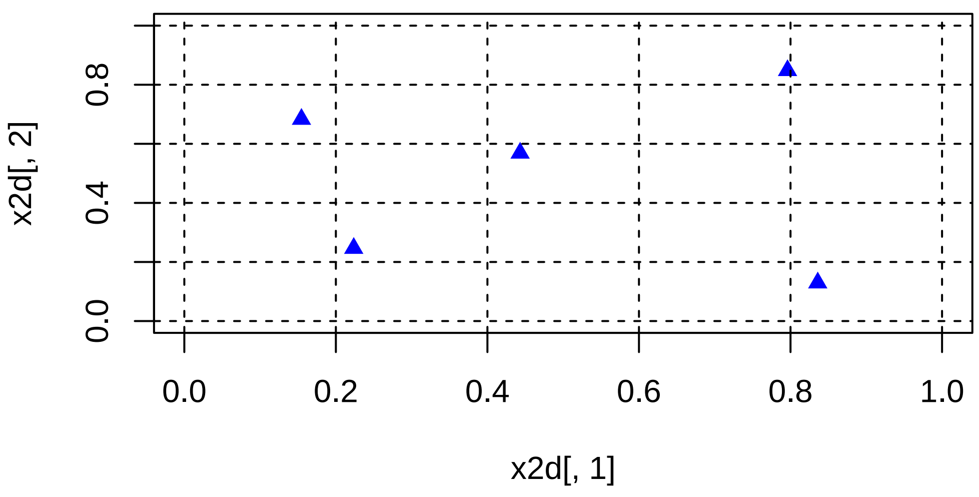
Stratification

```
U1 <- runif(4)*0.2  # in [0, 0.2]
U2 <- runif(4)*0.2+0.2 # in [0.2, 0.4]
U3 <- runif(4)*0.2+0.4 # in [0.4, 0.6]
U4 <- runif(4)*0.2+0.6 # in [0.6, 0.8]
U5 <- runif(4)*0.2+0.8 # in [0.8, 1]
bigU <- runif(20)
plot( bigU, rep(0,20), pch=19, cex=1.5, col='red', xlim=c(0,1), ylim=c(-0.4,0.4),
      xaxs='i', ylab='',yaxt='n',bty='u', xlab='Sampled U')
points( c(U1, U2, U3, U4, U5), rep(0.1,20), pch=18, cex=2, col='blue3')
abline(v=c(0.2,0.4, 0.6,0.8), lty=2)
text(0.1,y=-0.1,"Uniform[0,1]", col="red", cex=1.2)
text(0.1,y=0.2, "Stratified", col="blue", cex=1.2)
```

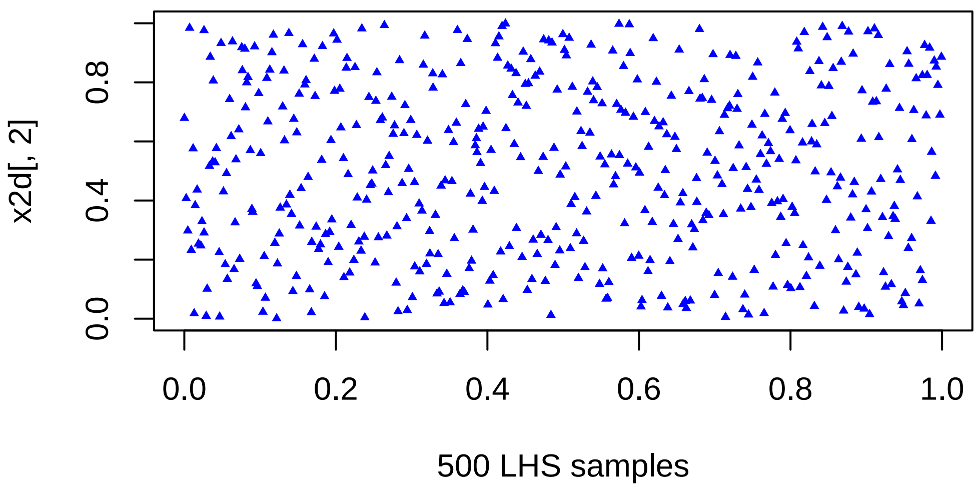


- 5 strata with 4 samples per strata = 20 samples (red dots are 20 Uniforms(0,1))
- Now the samples are NOT identical
- Stratification makes sure there is a diversity of scenarios – some very low ones, some middle ones, some high ones
- Guaranteed to reduce variance
- Can have non-equal strata of length p_i
- One generalization of stratification to higher dimensions is called **Latin Hypercube Sampling**
- Here is an LHS sample of size 5 in 2 dimensions

```
library(tgp)
x2d <- lhs(5, array(c(0,0,1,1),dim=c(2,2)))
plot(x2d[,1], x2d[,2], xlim=c(0,1), ylim=c(0,1), pch=17, col="blue")
abline(v=c(0,0.2, 0.4, 0.6, 0.8, 1), lty=2)
abline(h=c(0,0.2, 0.4, 0.6, 0.8, 1), lty=2)
```



```
x2d <- lhs(500, array(c(0,0,1,1),dim=c(2,2)))
plot(x2d[,1], x2d[,2], xlim=c(0,1), ylim=c(0,1), pch=17, col="blue", cex=0.5, xlab='500 LHS samples')
```

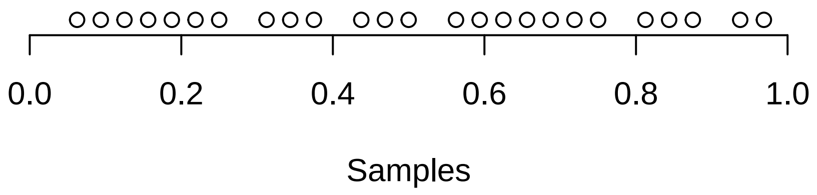


Quasi Monte Carlo

- A special **pre-computed** sequence of numbers
- Designed to have strong *uniformity* properties
- Use it to replace U_i's in Monte Carlo
- The QMC is deterministic, thus the resulting approximation is no longer a probabilistic estimate
- No confidence interval, no randomness of the estimator

- Tends to converge very fast
- Example: Sobol sequence

```r
sob <- sobol(25)
plot(sob, rep(0,25), bty='n', yaxt='n', xlab='Samples', ylab='',
     ylim=c(-0.1,0.1),xlim=c(0,1))
```



We **scramble** it to remove the above pattern. Then feed into the regular MC engine for option pricing

```r
K <- 55
qmcSeq <- sobol(500,scrambling=1)
qmcPayoff <- rep(0,500)
for (j in 1:500) {
  S[j] <- S0*exp( (r-sigma^2/2)*T + sigma*sqrt(T)*qnorm(qmcSeq[j]))
  qmcPayoff[j] <- exp(-r*T)*pmax(S[j]-K, 0)
}
mean(qmcPayoff)
```

```
[1] 1.845414
```

- This is very close to the exact 1.871 and only required 500 samples
- However, must accept its accuracy on faith (again, no confidence interval available)

Recap

- Efficiency of Monte Carlo is driven by its variance
- Higher variance = wider conf interval = less precision.
- Thus, requires more samples to achieve the target precision. So high variance → slower to converge/run.
- Antithetic MC:
  - Easy to code
  - Tends to reduce variance a bit
- Stratified MC:
  - Guaranteed to reduce variance
  - Often a significant speed-up
  - Still unbiased
- Quasi Monte Carlo
  - Zero variance

- Non-zero bias