Random Methods

CS 1025 Computer Science Fundamentals I

Stephen M. Watt

University of Western Ontario
What are Random Numbers?

- A sequence of numbers is random if there is no short pattern that can describe it.

- Given a sequence of numbers, can we tell if it is random?

- Maybe we just cannot see the pattern.

- Random numbers in nature (as far as we can tell).
  - Flipping coins
  - Time of quantum events

- Are these random?
  - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ...
  - H T H H T H H T H H H T H H H H T H H T ... 
  - 50 25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4
Using Random Numbers

• Random numbers are useful for:
  – Simulating processes
  – Computer games
  – Computer graphics
  – Testing software
  – Gambling
  – Cryptography

• Simulations:
  – If a model is too complex to model exactly.
  – To observe a system’s evolution.
  – For fun.
  – Aquarium screen savers and QCD Monte Carlo methods.
Can We Generate Random Numbers?

• We *cannot* provide an arithmetic algorithm to produce sequences of truly random numbers.

• We *can* provide an algorithm to produce numbers having certain statistical properties.
  – Frequency of individual numbers
  – Frequency of pairs
  – Average distance between recurring numbers
  – Bit transition tests
  – Autocorrelation tests
True Random Numbers

- Physically generated. Quantum events or chaotic systems.

  - Radioactive decay

  - Atmospheric noise

  - "Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of a chaotic system."

- Quantum Random Number Generator: [http://qrbg.irb.hr/](http://qrbg.irb.hr/)
  - USB2 12Mb/s
Pseudorandom Numbers

• Fast vs Good
• A “complicated” mathematical function often fails the statistical tests.
• Early random number generators were bad and led to flawed simulations.

• A reasonably fast and good method is the “linear congruential method”

\[ X_{[n+1]} = (a \cdot X_{[n]} + b) \mod m \]

for certain choices of \( a, b, m, X_{[0]} \).
(Pseudo) Random Numbers in Java

- Random() // Creates a new random number generator
- Random(long seed) // Creates a random no generator with a seed.

- Protected int next(int nb)ts
  - Linear congruential method
  - The low order $1 \leq n \leq 32$ of the result are pseudorandom bits
  - Can over-ride this in subclasses

- boolean nextBoolean()

  double nextDouble() // Uniform over $[0, 1]$
  double nextGaussian() // Normal with mean 0, std-dev 1

  int nextInt(int n) // Uniform integers over 0..n-1
  int nextInt() // Uniform over all possible values
  long nextLong() // Uniform over all possible values
class CoinFlipper {
    private Random rn = new Random();
    public boolean flip() { return rn.nextBoolean(); } 
}
Avoiding Bias

• Suppose we didn’t have `nextInt(int n)`

• Note `maxint mod 6 != 0`

class DiceRoller {
    private Random rn = new Random();

    // Return a number in 1..6
    public int roll() {
        int n;
        do {
            n = rn.next(3) & 0x7; // n in 0..7
        } while (n == 0 || n == 7);
        return n;
    }
}
An unfair coin

• Suppose you are not sure whether a coin is fair.
• How could you use it anyway to generate random bits? (That is bits with equal probability of being 0 or 1?)
Stock Option Pricing

• You should know about stock options if you are going to be offered some as part of your compensation.

• A stock option is a contract that allows you to buy (or sell) a share of a particular stock in a particular time window for a particular price.

  E.g. An option to buy IBM stock at $185 on November 18, 2011.

• What are options worth? We can use Monte Carlo simulation to find out.
Terminology

- Using the option to buy (or sell) the share is called “exercising” the option.
- The price at which you can buy (or sell) the share is called the “strike price.”
- The last day they can be exercised is the “expiry date”.
- If the exercise is to buy, it is called a “call”. If it is to sell then it is a “put”.
- Options that may be exercised at any time up to the expiry date are called “American” options. Options that may be exercised only on the expiry date are called “European” options.
- Some options, notably employee stock options, are “granted” on one date (when the parameters are fixed) and “vest” according to a schedule. Once they vest, they are locked in.
Some Notation

- The price of the stock at a point in time “S(t)”
- The strike price “K”
- The expiry date “T”
- The grant date “T₀”
- The “risk-free” rate “r” (e.g. bond yields for same period)
- The “drift” rate “μ” (= r, because you are hedging)
- The “volatility” “σ”

All of r, μ, σ are usually given as “annualized” rates.
Simulating One Time Step

\[ S_{i+1} = S_i \times \exp (\mu_{\text{eff}} + \sigma_{\text{eff}} \times \text{normal}(0,1)) \]

\[
S_i = S(t_i) \\
\mu_{\text{eff}} = (\mu - \frac{1}{2} \sigma \times \sigma) / \Delta T \\
\sigma_{\text{eff}} = \sigma / \sqrt{\Delta T} \\
\text{normal}(0,1) = \text{a normally distributed random variable with mean 0 and standard deviation 1.} \\
\Delta T = \text{the unit of time corresponding to the rates } r, \mu \text{ and } \sigma, \text{ typically 252 (trading days per year).} \]
Continuously Compounded Interest

• Outside the world of financial analysis, people think about annual percentage rates. E.g. 6% APR.

• If compounded monthly we have a monthly interest rate $r_{mo}$, given by $1 + r_{yr} = (1 + r_{mo})^{12}$ or a daily interest rate $r_{dy}$, given by $1 + r_{yr} = (1 + r_{dy})^{365}$.

• After $t$ days, a value will have grown by $(1 + r_{dy})^t$.

• Financial mathematicians use a “continuously compounded” interest rate $r_{cc}$ such that $\exp(r_{cc} t) = (1 + r_{yr})^t$ e.g. for $t$ in years.

So $\exp(r_{cc}) = 1 + r_{yr}$. 6% APR => $r_{cc} = \ln(1.06) = 5.8268 \% \text{ CC}$
Volatility

• Based on a series of price observations over a past window.

• Compute std deviation of \( \ln\left(\frac{S_i + D_{i+1}}{S_{i-1}}\right) \) divided by the sqrt of the # of observations per year.

That is, the annualized std dev of the continuously compounded growth.
Computing Volatility – Getting Data
Computing Volatility – Getting Data

International Business Machines Corp. historical prices

<table>
<thead>
<tr>
<th>Date</th>
<th>Open</th>
<th>High</th>
<th>Low</th>
<th>Close</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov 2, 2011</td>
<td>184.00</td>
<td>185.29</td>
<td>183.03</td>
<td>183.92</td>
<td>4,136,703</td>
</tr>
<tr>
<td>Nov 1, 2011</td>
<td>191.55</td>
<td>183.29</td>
<td>190.74</td>
<td>181.35</td>
<td>5,883,336</td>
</tr>
<tr>
<td>Oct 31, 2011</td>
<td>185.53</td>
<td>186.91</td>
<td>184.63</td>
<td>184.53</td>
<td>4,709,968</td>
</tr>
<tr>
<td>Oct 28, 2011</td>
<td>186.27</td>
<td>187.07</td>
<td>185.26</td>
<td>187.45</td>
<td>5,009,820</td>
</tr>
<tr>
<td>Oct 27, 2011</td>
<td>184.99</td>
<td>186.00</td>
<td>183.18</td>
<td>186.34</td>
<td>7,057,876</td>
</tr>
<tr>
<td>Oct 26, 2011</td>
<td>181.74</td>
<td>182.28</td>
<td>179.93</td>
<td>181.97</td>
<td>3,587,883</td>
</tr>
<tr>
<td>Oct 25, 2011</td>
<td>191.66</td>
<td>192.97</td>
<td>190.01</td>
<td>180.36</td>
<td>5,374,064</td>
</tr>
<tr>
<td>Oct 24, 2011</td>
<td>191.51</td>
<td>193.39</td>
<td>190.62</td>
<td>182.25</td>
<td>5,691,258</td>
</tr>
<tr>
<td>Oct 21, 2011</td>
<td>178.11</td>
<td>181.67</td>
<td>178.76</td>
<td>181.63</td>
<td>8,056,801</td>
</tr>
<tr>
<td>Oct 20, 2011</td>
<td>178.13</td>
<td>179.81</td>
<td>177.25</td>
<td>177.25</td>
<td>7,514,283</td>
</tr>
<tr>
<td>Oct 19, 2011</td>
<td>176.68</td>
<td>179.81</td>
<td>177.01</td>
<td>177.39</td>
<td>8,708,371</td>
</tr>
<tr>
<td>Oct 18, 2011</td>
<td>178.00</td>
<td>179.38</td>
<td>176.25</td>
<td>178.90</td>
<td>15,729,689</td>
</tr>
<tr>
<td>Oct 17, 2011</td>
<td>185.71</td>
<td>190.16</td>
<td>185.90</td>
<td>186.59</td>
<td>8,750,278</td>
</tr>
<tr>
<td>Oct 14, 2011</td>
<td>188.19</td>
<td>190.83</td>
<td>187.73</td>
<td>190.53</td>
<td>5,368,616</td>
</tr>
<tr>
<td>Oct 13, 2011</td>
<td>184.97</td>
<td>187.30</td>
<td>184.50</td>
<td>186.82</td>
<td>4,398,267</td>
</tr>
<tr>
<td>Oct 12, 2011</td>
<td>186.08</td>
<td>188.00</td>
<td>185.85</td>
<td>188.12</td>
<td>5,338,149</td>
</tr>
<tr>
<td>Oct 11, 2011</td>
<td>184.59</td>
<td>184.11</td>
<td>184.11</td>
<td>184.39</td>
<td>5,330,581</td>
</tr>
<tr>
<td>Oct 10, 2011</td>
<td>183.00</td>
<td>186.63</td>
<td>182.90</td>
<td>186.62</td>
<td>5,786,341</td>
</tr>
<tr>
<td>Oct 7, 2011</td>
<td>182.32</td>
<td>183.72</td>
<td>181.64</td>
<td>182.39</td>
<td>6,842,567</td>
</tr>
<tr>
<td>Oct 6, 2011</td>
<td>176.81</td>
<td>181.59</td>
<td>176.66</td>
<td>181.69</td>
<td>7,161,124</td>
</tr>
<tr>
<td>Oct 5, 2011</td>
<td>174.57</td>
<td>177.30</td>
<td>172.88</td>
<td>178.85</td>
<td>5,851,858</td>
</tr>
<tr>
<td>Oct 4, 2011</td>
<td>171.54</td>
<td>175.67</td>
<td>168.98</td>
<td>174.74</td>
<td>9,176,346</td>
</tr>
<tr>
<td>Oct 3, 2011</td>
<td>174.36</td>
<td>177.92</td>
<td>173.04</td>
<td>173.29</td>
<td>9,043,214</td>
</tr>
</tbody>
</table>

Historical chart

Export
Download to spreadsheet
### Computing Volatility – The Main Event

- **CC Vol** = Continuously compounded rate of return = ln((S[t] + Div[t+1])/S[t-1])
- **Ann Vol** = Annualized historical volatility = \( \text{StDev}(CC \text{ Ret}) \times \sqrt{N \text{ obs/yr}} \)

<table>
<thead>
<tr>
<th>Date</th>
<th>Open</th>
<th>High</th>
<th>Low</th>
<th>Close</th>
<th>Dividend</th>
<th>Volume</th>
<th>CC Ret</th>
<th>Ann Vol</th>
</tr>
</thead>
<tbody>
<tr>
<td>14-Nov-07</td>
<td>106.12</td>
<td>106.42</td>
<td>102.84</td>
<td>103.44</td>
<td>8453600</td>
<td>-0.0175367</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13-Nov-07</td>
<td>102.73</td>
<td>105.74</td>
<td>102.50</td>
<td>105.27</td>
<td>10777900</td>
<td>0.0369624</td>
<td>61.18%</td>
<td></td>
</tr>
<tr>
<td>12-Nov-07</td>
<td>101.89</td>
<td>104.19</td>
<td>100.70</td>
<td>101.45</td>
<td>13553600</td>
<td>0.0118990</td>
<td>43.30%</td>
<td></td>
</tr>
<tr>
<td>09-Nov-07</td>
<td>104.92</td>
<td>104.92</td>
<td>99.27</td>
<td>100.25</td>
<td>18084100</td>
<td>-0.0568092</td>
<td>64.03%</td>
<td></td>
</tr>
<tr>
<td>08-Nov-07</td>
<td>105.65</td>
<td>110.32</td>
<td>103.99</td>
<td>106.11</td>
<td>23092200</td>
<td>-0.0457744</td>
<td>62.11%</td>
<td></td>
</tr>
<tr>
<td>07-Nov-07</td>
<td>113.56</td>
<td>113.64</td>
<td>110.90</td>
<td>111.08</td>
<td>7087000</td>
<td>-0.0186405</td>
<td><strong>55.62%</strong></td>
<td></td>
</tr>
<tr>
<td>06-Nov-07</td>
<td>113.49</td>
<td>113.95</td>
<td>111.67</td>
<td>113.17</td>
<td>7297700</td>
<td>0.0014980</td>
<td>51.73%</td>
<td></td>
</tr>
<tr>
<td>05-Nov-07</td>
<td>115.11</td>
<td>115.11</td>
<td>112.83</td>
<td>113.40</td>
<td>7155300</td>
<td>-0.0104391</td>
<td>47.91%</td>
<td></td>
</tr>
<tr>
<td>02-Nov-07</td>
<td>114.42</td>
<td>115.15</td>
<td>113.57</td>
<td>114.59</td>
<td>6114800</td>
<td>0.0082370</td>
<td>46.12%</td>
<td></td>
</tr>
<tr>
<td>01-Nov-07</td>
<td>115.50</td>
<td>116.09</td>
<td>113.32</td>
<td>113.65</td>
<td>7594400</td>
<td>-0.0215006</td>
<td>43.86%</td>
<td></td>
</tr>
<tr>
<td>31-Oct-07</td>
<td>114.75</td>
<td>116.25</td>
<td>113.28</td>
<td>116.12</td>
<td>7216800</td>
<td>0.0173736</td>
<td>43.80%</td>
<td></td>
</tr>
<tr>
<td>30-Oct-07</td>
<td>114.50</td>
<td>114.90</td>
<td>113.75</td>
<td>114.12</td>
<td>4015500</td>
<td>-0.0059410</td>
<td>41.78%</td>
<td></td>
</tr>
<tr>
<td>29-Oct-07</td>
<td>113.90</td>
<td>115.01</td>
<td>113.85</td>
<td>114.80</td>
<td>5103200</td>
<td>0.0093643</td>
<td>40.76%</td>
<td></td>
</tr>
<tr>
<td>252</td>
<td>21-Nov-06</td>
<td>93.01</td>
<td>93.43</td>
<td>92.85</td>
<td>93.08</td>
<td>3067800</td>
<td>-0.0018247</td>
<td>19.30%</td>
</tr>
<tr>
<td>253</td>
<td>20-Nov-06</td>
<td>93.60</td>
<td>93.80</td>
<td>93.01</td>
<td>93.25</td>
<td>5010600</td>
<td>0.0059874</td>
<td>19.27%</td>
</tr>
<tr>
<td>254</td>
<td>17-Nov-06</td>
<td>93.42</td>
<td>94.05</td>
<td>93.31</td>
<td>93.81</td>
<td>5251000</td>
<td>0.0036309</td>
<td>19.24%</td>
</tr>
<tr>
<td>255</td>
<td>16-Nov-06</td>
<td>93.07</td>
<td>93.60</td>
<td>92.60</td>
<td>93.47</td>
<td>4171500</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Monte Carlo Simulator

```java
void simulate(
    int ndays, double[] Svals, double SO, double K,
    double r, double mu, double sigma
) {
    double delt at = 252.0;
    double effmu = (mu - 0.5*sigma*sigma)/deltat;
    double effsigma = sigma/Math.sqrt(deltat);
    double S = SO;
    Svals[0] = S;
    for (int i = 1; i < ndays; i++) {
        S *= Math.exp(effmu + effsigma * cn.nextGaussian());
        Svals[i] = S;
    }
}

double[] inTheMoneyFinalValSample(
    int nruns, int ndays, double SO, double K,
    double r, double mu, double sigma
) {
    double[] finalVals = new double[nruns];
    double[] Svals = new double[ndays];
    for (int i = 0; i < nruns; i++) {
        simulate(ndays, Svals, SO, K, r, mu, sigma);
        finalVals[i] = Math.max(Svals[ndays-1]-K, 0);
    }
    return finalVals;
```
Main Program – Test with IBM Stock

In-the-money by $6.34

```java
public class Main {
    public static void main(String[] args) {
        double S0 = 103.44; // IBM Nov 14, 2007
        double r = Math.log(1 + 0.0355); // 1 year T-bills at 3.92%, 5 year at 3.97%
        double sigma = 0.2459; // Daily observations, 55 day look back.
        double mu = r; // Assume hedging.
        double K = 100.00; // IBMAT.X

        int ndays = 42; // Trading days to Jan 18, 2008.
        int daysPerYear = 242; // Trading days per year.

        MonteCarloSimulator mc = new MonteCarloSimulator();

        int nruns = 100000;

        double[] finalVals = mc.inTheMoneyFinalValSample(nruns, ndays, S0, K, r, mu, sigma);

        double finalValExpected = Statistics.mean(finalVals);
        double finalValStdev = Statistics.stdev(finalVals);

        System.out.println("The expected final in-the-money-value is "+finalValExpected);
        System.out.println("The stdev of final in-the-money-value is "+finalValStdev);

        double presentValExpected = Math.exp(-r*ndays/daysPerYear)*finalValExpected;
        System.out.println("The expected present in-the-money-value is "+presentValExpected);
    }
}
```
Compare to Current Options Market

The market has these options trading at $9.10, which is overvalued according to our simulation.
Modelling More Complex Situations

• It turns out that the simulations we have just done can be calculated analytically using the Black-Scholes model.

• More complex situations cannot be modelled exactly so easily, which is the real reason to use Monte Carlo methods.

• E.g. Suppose an investor is nervous and will cash out the moment the investment reaches 125% of the strike price.

Then we would modify the method `inTheMoneyFinalValSample` to compute the final value from the array for each run differently.
Modelling More Complex Situations

• For our nervous investor with the 125% threshold, we would replace

```java
finalVals[i] = Math.max(Svals[ndays-1]-K, 0);
```

with something like:

```java
double thisFinalVal = 0;
for (int j = 0; j < ndays; j++)
    if (Svals[j] > 1.25 * K) {
        // Threshold met. Exercise now.
        thisFinalVal = Svals[j] - K;
        break;
    }
if (thisFinalVal == 0) {
    // Reached end with no early exercise. Exercise at expiry.
    thisFinalVal = Math.max(Svals[ndays-1]-K, 0);
}
finalVals[i] = thisFinalVal;
```