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 H T H H H H T H H H 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.
- Hotbits: http://www.fourmilab.ch/hotbits/
  - Radioactive decay
- Random.org: http://www.random.org/
  - Atmospheric noise
- Lavarand: US Patent 5,732,138
  - "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/
  - 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*"

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

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 nbits)
  - Linear congruential method
  - The low order 1 <= nbits <= 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()         // Uniform over all possible values
long     nextLong()        // Uniform over all possible values
```

A Coin Flipper

```
class CoinFlipper {
	private Random rn = new Random();
	public boolean flip() { return rn.nextBoolean(); }
}
```

A Dice Roller

- Be careful to avoid bias (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;
        }
}
```

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 $120 on November 1, 2010.

- 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 "T_0"

- 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

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

$S_i = S(t_i)$

$\mu_{eff} = (\mu - ½\, \sigma \times \sigma) / \Delta T$

$\sigma_{eff} = \sigma / \text{sqrt}(\Delta T)$

normal(0,1) = a normally distributed random variable
with mean 0 and standard deviation 1.

ΔT = the unit of time corresponding to the rates r, μ and σ,
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 => $r_{cc} = \ln(1.06) = 5.8268$ % CC

Volatility

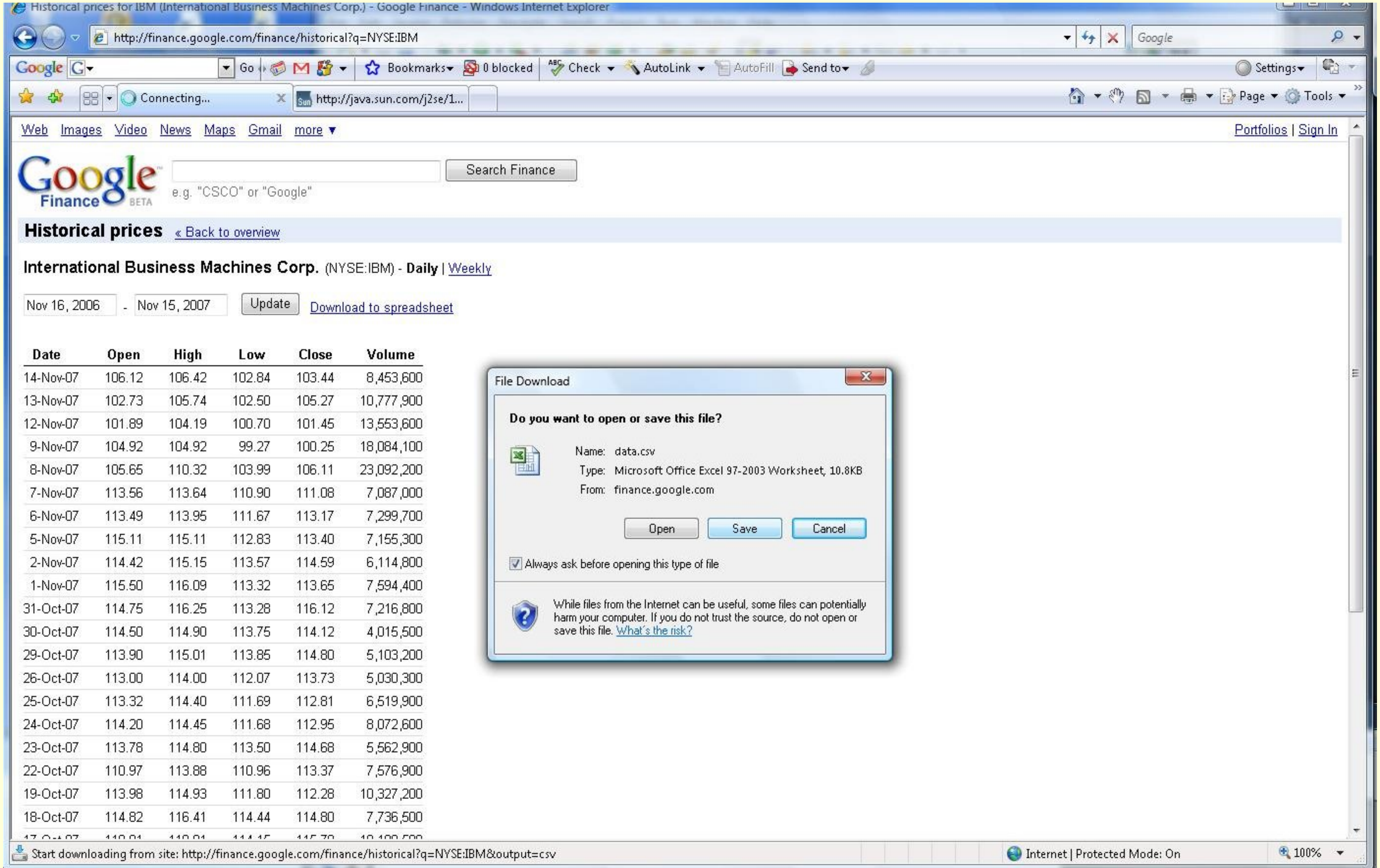
- Based on a series of price observations over a past window.

- Compute std deviation of $\ln\left((S_i + D_{i+1})/S_{i-1}\right)$ 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



Computing Volatility – The Main Event

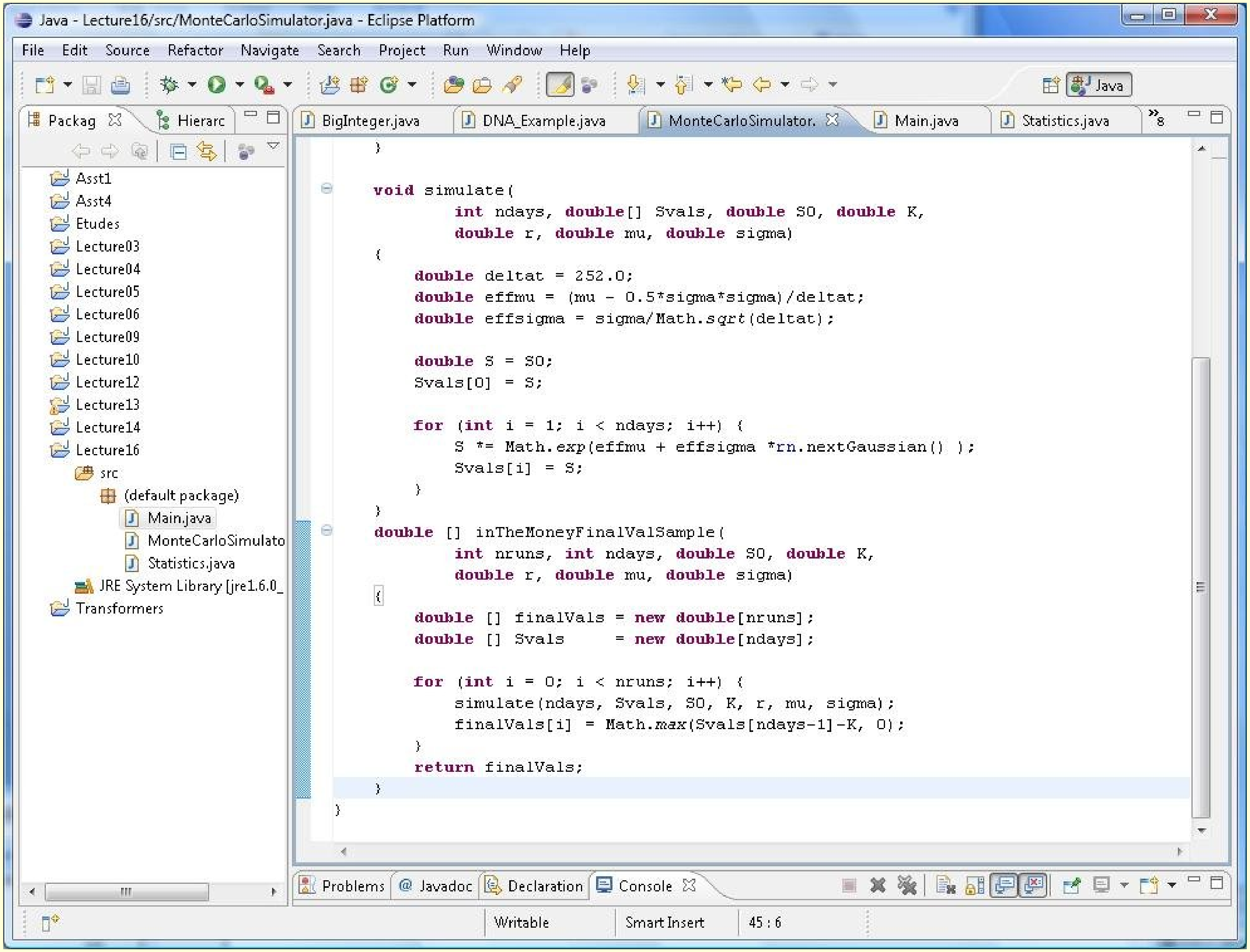
IBM_Stock_Data.csv - Microsoft Excel

I11 =STDEV(H$6:H11)*SQRT($A$1)

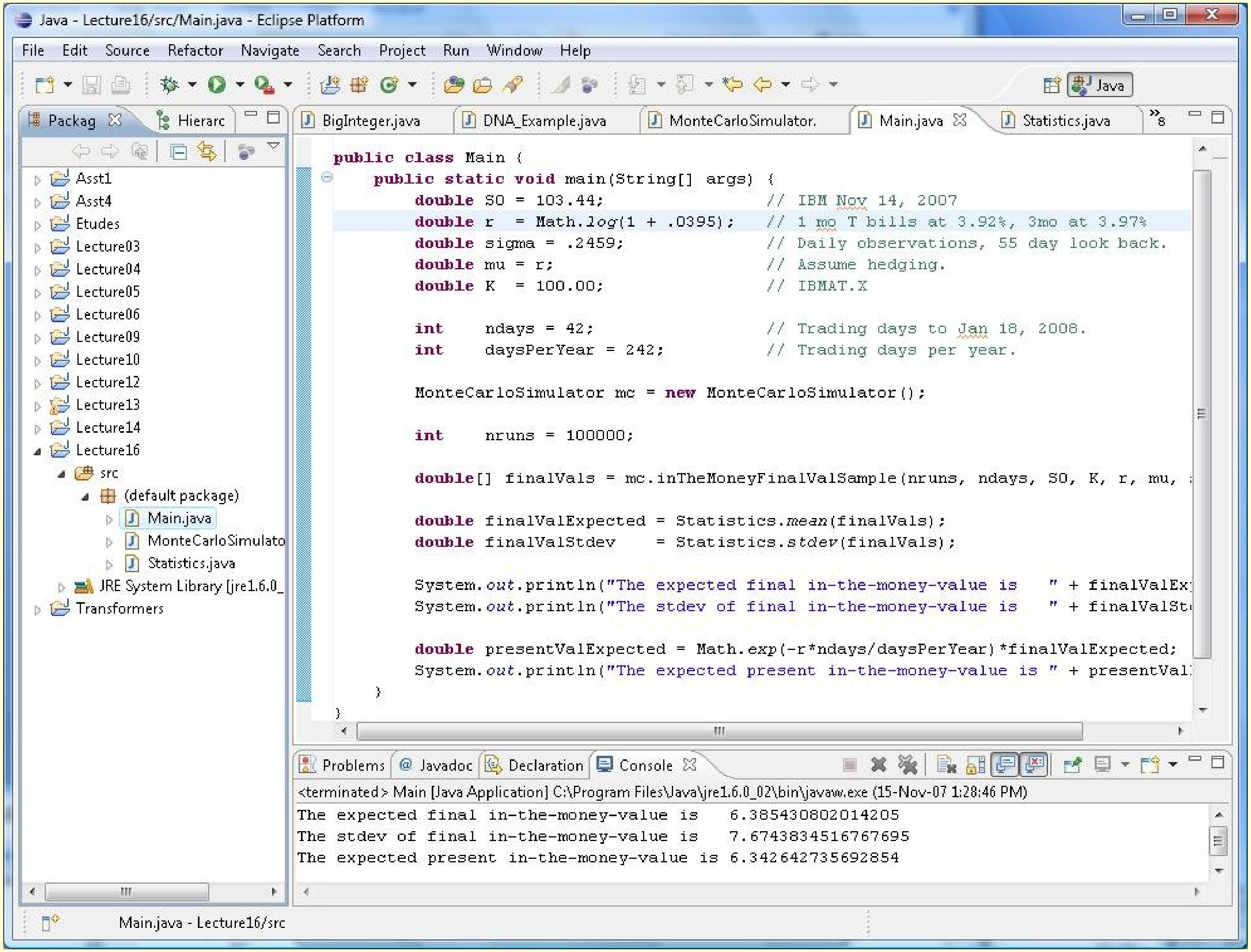
	A	B	C	D	E	F	G	H	I
1	252	Observations/yr			*CC Ret* = Continuously compounded rate of return = ln((S[i]+Div[i+1])/S[i-1])				
2					*Ann Vol* = Annualized historical volatility				= StDev(CC Ret)*sqrt(N obs/yr)
3									
4	Date	Open	High	Low	Close = S	Dividend	Volume	CC Ret	Ann Vol
5						-			
6	14-Nov-07	106.12	106.42	102.84	103.44		8453600	-0.0175367	
7	13-Nov-07	102.73	105.74	102.50	105.27		10777900	0.0369624	61.18%
8	12-Nov-07	101.89	104.19	100.70	101.45		13553600	0.0118990	43.30%
9	09-Nov-07	104.92	104.92	99.27	100.25		18084100	-0.0568092	64.03%
10	08-Nov-07	105.65	110.32	103.99	106.11		23092200	-0.0457744	62.11%
11	07-Nov-07	113.56	113.64	110.90	111.08	0.40	7087000	-0.0186405	55.62%
12	06-Nov-07	113.49	113.95	111.67	113.17		7299700	0.0014980	51.73%
13	05-Nov-07	115.11	115.11	112.83	113.40		7155300	-0.0104391	47.91%
14	02-Nov-07	114.42	115.15	113.57	114.59		6114800	0.0082370	46.12%
15	01-Nov-07	115.50	116.09	113.32	113.65		7594400	-0.0215006	43.86%
16	31-Oct-07	114.75	116.25	113.28	116.12		7216800	0.0173736	43.80%
17	30-Oct-07	114.50	114.90	113.75	114.12		4015500	-0.0059410	41.78%
18	29-Oct-07	113.90	115.01	113.85	114.80		5103200	0.0093643	40.76%
252	21-Nov-06	93.01	93.43	92.85	93.08		3067800	-0.0018247	19.30%
253	20-Nov-06	93.60	93.80	93.01	93.25		5010600	-0.0059874	19.27%
254	17-Nov-06	93.42	94.05	93.31	93.81		5251000	0.0036309	19.24%
255	16-Nov-06	93.07	93.60	92.60	93.47		4171500		
256									

IBM_Stock_Data

Monte Carlo Simulator



Main Program – Test with IBM Stock



In-the-money by $6.34

Compare to Current Options Market

Quotes for IBM - Yahoo! Finance - Windows Internet Explorer

http://finance.yahoo.com/q/os?s=IBM&m=2008-01-18

Options

View By Expiration: Nov 07 | Dec 07 | **Jan 08** | Apr 08 | Jan 09 | Jan 10

Options Expiring Fri, Jan 18, 2008

Calls							Strike Price	Puts						
Symbol	Last	Change	Bid	Ask	Volume	Open Int		Symbol	Last	Change	Bid	Ask	Volume	Open Int
IBMAK.X	46.30	0.00	50.40	50.70	27	1,308	55.00	IBMMK.X	0.05	0.00	N/A	0.10	60	3,016
IBMAL.X	43.40	0.00	45.40	45.80	31	1,008	60.00	IBMML.X	0.05	0.00	N/A	0.10	57	4,084
IBMAM.X	36.10	0.00	40.60	40.80	33	1,843	65.00	IBMMM.X	0.15	0.00	0.05	0.15	2	8,471
IBMAN.X	33.50	0.00	35.70	35.90	2	1,747	70.00	IBMMN.X	0.20	0.00	0.15	0.25	75	6,843
IBMAO.X	30.90	↑ 4.10	30.80	31.10	2	2,279	75.00	IBMMO.X	0.26	0.00	0.25	0.35	15	9,871
IBMAP.X	22.50	0.00	26.10	26.40	114	5,386	80.00	IBMMP.X	0.50	0.00	0.45	0.60	31	16,096
IBMAQ.X	20.20	↓ 0.70	21.30	21.70	13	9,612	85.00	IBMMQ.X	0.85	↑ 0.05	0.80	0.90	59	16,097
IBMAR.X	17.50	↑ 1.10	17.00	17.30	28	10,342	90.00	IBMMR.X	1.40	↑ 0.05	1.30	1.45	16	15,449
IBMAS.X	12.70	↓ 0.24	12.90	13.10	36	16,871	95.00	IBMMS.X	2.21	↓ 0.29	2.10	2.30	197	10,412
IBMAT.X	9.10	↑ 1.00	9.20	9.40	110	12,085	100.00	IBMMT.X	3.50	↓ 0.60	3.40	3.60	159	21,872
IBMAA.X	6.10	↑ 0.80	6.10	6.40	98	10,978	105.00	IBMMA.X	5.51	↓ 0.81	5.30	5.50	30	15,302
IBMAB.X	3.90	↑ 0.60	3.80	4.00	157	14,268	110.00	IBMMB.X	8.10	↑ 0.30	7.90	8.20	171	17,162
IBMAC.X	2.15	↑ 0.25	2.20	2.40	665	13,282	115.00	IBMMC.X	11.50	0.00	11.40	11.60	17	7,008
IBMAD.X	1.25	↓ 0.05	1.20	1.35	214	19,381	120.00	IBMMD.X	16.00	↑ 1.00	15.40	15.80	40	4,934

Internet | Protected Mode: On 125%

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% threshhold, we would replace

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

with something like:

```
double thisFinalVal = 0;
for (int j = 0; j < ndays; j++)
 if (Svals[j] > 1.25 * K) {
        // Threshhold 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;
```