FINM3405 Derivatives and Risk Management

Week 7: Binomial model and Monte Carlo pricing

Dr Godfrey Smith

THE UNIVERSITY OF QUEENSLAND AUSTRALIA

August 31, 2024

Contents

Introduction

So far we've covered the basic of options, the Black-Scholes model and its associated Greeks, delta hedging, implied volatility and trading strategies. This week we take the first steps into the world of numerical option pricing methods, which are needed to price complex, exotic options and for using more advanced, complex and hopefully accurate option pricing models than the Black-Scholes model. This week we cover the binomial model and the famous Monte Carlo approach, which is one of the most powerful and practical numerical methods for pricing options.

- Readings: Chapters 13.1-4, 21.1, 21.6 of Hull.

Introduction

For an alternative perspective and some additional material, I highly recommend considering Chapters 21 and 22 on the binomial model, and Chapter 26 on Monte Carlo simulation, of the alternative textbook Cuthbertson, Nitzsche and O'Sullivan, Derivatives: theory and practice.

Question: Why do we need numerical option pricing techniques?

Numerical methods

European calls and puts are relatively simple plain vanilla derivatives.

- In this simple context we can derive a nice, clean Black-Scholes equation or formula for their premiums:

$$C = S\mathcal{N}(d_1) - Ke^{-rT}\mathcal{N}(d_2), \qquad P = Ke^{-rT}\mathcal{N}(-d_2) - S\mathcal{N}(-d_1),$$

$$d_1 = \frac{\log \frac{S}{K} + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}} \qquad \text{and} \qquad d_2 = d_1 - \sigma\sqrt{T}.$$

The Black-Scholes European call and put option pricing model results in an analytical or closed-form solution equation or formula.

Numerical methods

But this is rarely the case in general.

Numerical methods

1. When applying the Black-Scholes framework to derive pricing models for more complex, exotic derivatives (even just for American options), we immediately run into the scenario of being unable to derive closed-form, analytical solution equations.
2. We may also want to apply more complex derivative security pricing frameworks than the Black-Scholes framework (say the Heston stochastic volatility, or the Merton jump-diffusion, frameworks), but again we immediately run into the same problem.

Numerical methods

Question: How do we proceed in these scenarios of trying to

1. price complex derivative securities and
2. use more complex derivative security pricing frameworks

when we can't derive a closed-form solution equation or formula?

Answer: We use numerical approximation or computational techniques:

- We use computers and iterative algorithms.

Remark

These numerical algorithms are similar to Newton's method for the IRR of a bond or implied vol of an option.

Numerical methods

Three main **numerical** or **computation approaches**:

1. Lattice type methods like the binomial and trinomial models.
2. Monte Carlo methods:
   - Motivated by the risk-neutral approach to derivative security pricing.
3. Partial differential equation numerical solution methods:
   - Motivated by the partial differential equation (PDE) approach to derivative security pricing (we avoid it in FINM3405).

In FINM3405 we focus on the binomial and Monte-Carlo methods.

- Numerically solving PDE is a too messy mathematically.

Numerical methods

Question: Why do we require more complex derivative security pricing frameworks than the Black-Scholes framework?

Answer: When calibrated to market data, the Black-Scholes framework does not accurately price observed options trading in the market.

Remark

We see this from the volatility smile and term structure:

- The volatility parameter σ we need to use to get the Black-Scholes model to match observed option premiums is not constant across the range of strike prices and expiries.

Numerical methods

Consequently traders use:

- Rules of thumb, intuition and market conventions and experience to determine the σ parameter to use in the Black-Scholes model.
- Other, more complex and accurate option pricing models.

Numerical methods

Remark

Calibrating an option pricing model means estimating the unobserved input parameters of the model using statistical techniques. In the Black-Scholes model the only unobservable parameter is the volatility σ. Due to the assumption of geometric Brownian motion, the maximum likelihood estimator of σ is the annualised standard deviation of the underlying asset's historical returns. But using this estimate for σ in the Black-Scholes model results in inaccurate option prices, as we've seen.

Numerical methods

The reason that the Black-Scholes model, when using the maximum likelihood estimator of its unknown parameter σ, does not accurately price observed options in the market is that the Black-Scholes model is derived based on a large list of very restrictive and unrealistic assumptions. The two main assumptions important to us are:

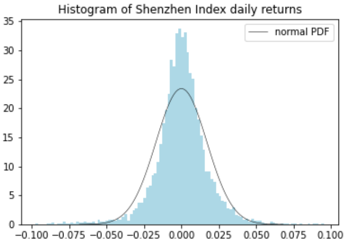
1. The underlying asset's returns are normally distributed, or equivalently, the asset's price is log-normally distributed.
2. The volatility parameter σ is constant over the life of an option.

These two assumptions <u>do not</u> hold in reality:

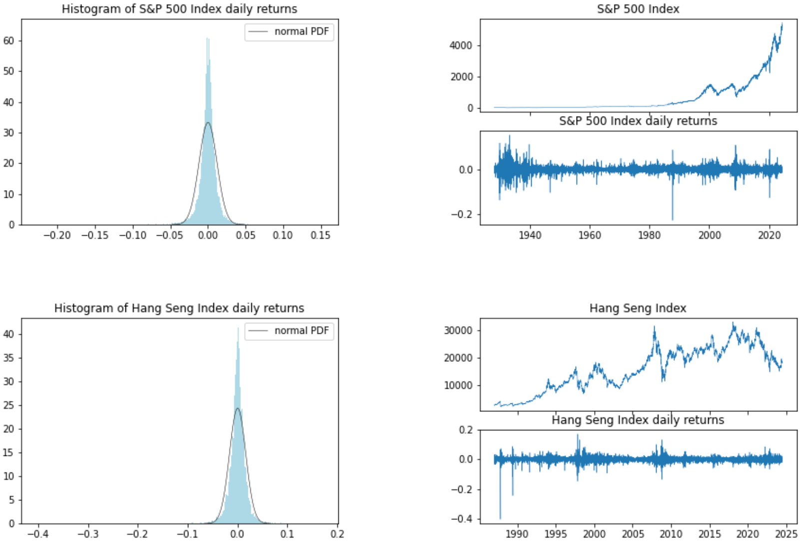
Numerical methods

The below plots display common **stylised features** of financial returns:

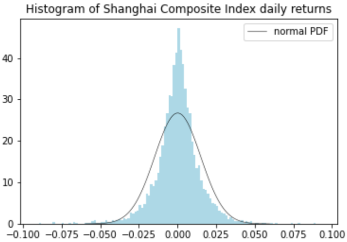
- Non-normality:
  - Spiked mean.
  - Narrow shoulders.
  - Fat tails.
  - Skewness, or at least extreme negative return events.
- Volatility clustering or regimes (non-constant volatility).

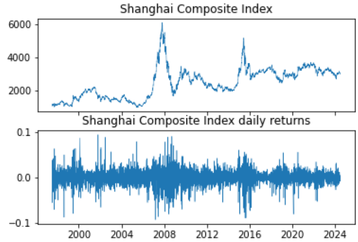


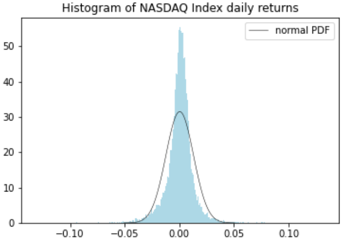
Numerical methods

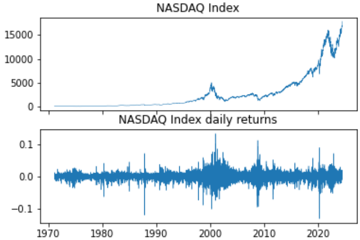


Numerical methods









Numerical methods

Python code that produced the above Shenzhen Index plots:

```python
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import yfinance as yf
df = yf.download("399001.SZ")
ret = np.log(df["Adj Close"]).diff(1).dropna().rename("returns")
# time series plots
fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, sharex=True)
ax1.plot(df["Adj Close"], linewidth=0.5)
ax1.set_title("Shenzhen Index")
ax2.plot(ret, linewidth=0.5)
ax2.set_title("Shenzhen Index daily returns")
plt.show()
# histogram with fitted normal pdf
plt.hist(ret, density=True, bins="auto")
mu, std = stats.norm.fit(ret)
x = np.linspace(-4*std, 4*std, 100)
plt.plot(x, npdf, color="k", linewidth=0.5, label="normal PDF")
plt.title("Histogram of Shenzhen Index daily returns")
plt.legend()
plt.show()
```

Numerical methods

A significant amount of research in quantitative finance involves developing (i) more complex and accurate option pricing models and (ii) models that are able to price more complex, exotic options.

- Relating to (i), a particular focus of this research is deriving option pricing models in which the underlying asset is assumed to follow a random process that matches or models observed asset price returns more accurately than geometric Brownian motion.
  - The Heston stochastic volatility model allows non-constant volatility.
  - The Merton jump-diffusion model allows jumps in asset prices.

More complex option pricing models nearly always require numerical pricing methods, so we start with basic numerical methods this week.

Numerical methods

Today we start the journey of numerical or computational option pricing methods by presenting the basic binomial model and Monte Carlo numerical approaches in the simple context of pricing plain vanilla European calls and puts, in which we have closed-form solution formula.

- Next week we extend these numerical approaches to American and other exotic options, for which we don't have analytical solutions.

We start with the 1-period binomial model in order to "get the idea" and then generalise it to the multi-period binomial model.

1-period

As per the Black-Scholes model, the binomial model is a pricing framework defined by a set of simplifying assumptions:

Assumptions of the 1-period binomial model

- One trading period of length T years to the option's expiry.
- Starting at S, two possible outcomes for the underlying asset:
  1. Up to $S_u = Su$, where u is the asset's **up factor**.
  2. Down to $S_d = Sd$, where d is the asset's **down factor**.
- A risk-free rate r satisfying $d < e^{rT} < u$.
- All the other "usual" assumptions.

1-period

The call option's **payoffs** in each possible outcome of the price of the underlying asset are:

$$C_u = \max\{0, S_u - K\}$$

and

$$C_d = \max\{0, S_d - K\}.$$

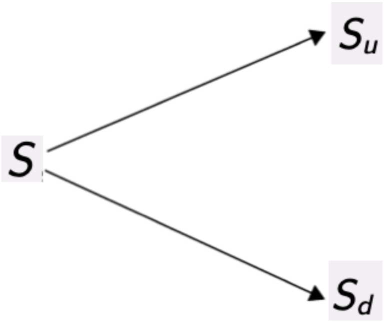


Figure: Possible outcomes of the price of the underlying asset.

1-period

We now use arbitrage arguments to derive an equation

$$C = e^{-rT}\left[qC_u + (1-q)C_d\right]$$

for the price C of a call option in the 1-period binomial framework, where

$$q = \frac{e^{rT} - d}{u - d}$$

is a probability distribution also derived below and that has important interpretations in quantitative finance: It is risk-neutral.

1-period

We set up a **replicating portfolio** R by investing:

- ϕ units (dollars) in the risk-free rate.
- Δ units (hence ΔS dollars) in the underlying asset.

The replicating portfolio's payoffs in each outcome of the underlying are

$$R_u = \phi e^{rT} + \Delta S_u \qquad \text{and} \qquad R_d = \phi e^{rT} + \Delta S_d,$$

and by "replicating" we mean $R_u = C_u$ and $R_d = C_d$.

Remark

We initially invest $R = \phi + \Delta S$ dollars in the replicating portfolio.

1-period

We can show that there is a unique replicating portfolio given by

$$\phi = \frac{uC_d - dC_u}{(u-d)e^{rT}} \qquad \text{and} \qquad \Delta = \frac{C_u - C_d}{S(u-d)}.$$

Remark

This replicating portfolio can be found by writing the conditions $R_u = C_u$ and $R_d = C_d$ as and then solving the linear system

$$\begin{bmatrix} e^{rT} & Su \\ e^{rT} & Sd \end{bmatrix} \begin{bmatrix} \phi \\ \Delta \end{bmatrix} = \begin{bmatrix} C_u \\ C_d \end{bmatrix}.$$

1-period

The payoffs of the replicating portfolio equal those of the call option, so:

Law of one price: The replicating portfolio and call option must have the same price, otherwise an arbitrage opportunity exists:

$$C = \phi + \Delta S.$$

Inserting the above values for ϕ and Δ and then rearranging, we get

$$C = e^{-rT}\left[qC_u + (1-q)C_d\right],$$

where $q = \dfrac{e^{rT} - d}{u - d}$ and $1 - q = \dfrac{u - e^{rT}}{u - d}$.

1-period

Remark

Here q is a probability distribution and we can write

$$\begin{aligned} C &= e^{-rT}\mathbb{E}^q[C_T] \\ &= e^{-rT}\left[qC_u + (1-q)C_d\right]. \end{aligned}$$

The option payoff C_T is a random variable with outcomes:

- $C_u = \max\{0, S_u - K\}$ (up state) with probability q.
- $C_d = \max\{0, S_d - K\}$ (down state) with probability $1 - q$.

Here, the price of a call option is the present value, discounted at the risk-free rate, of its expected future payoff under q.

1-period

By the **risk-neutral world** we mean quantifying the probability of outcomes in financial markets via a risk-neutral probability q.

- The value of every asset is the present value, discounted at the risk-free rate r, of its expected future cashflows under q.

We discount the expected future cashflows with r:

- We don't add a risk-premium to the discount rate.

This also holds for the underlying asset: We can also show that $S = e^{-rT}\mathbb{E}^q[S_T]$, where S_T is a random variable with outcomes S_u with probability q and S_d with probability $1 - q$.

1-period

Example

Let $S = 50$, $K = 50$, $r = 0.05$, $T = \frac{1}{2}$ and $u = 1.1$ with $d = \frac{1}{u}$.

- We calculate that

$$q = \frac{e^{rT} - d}{u - d} = 0.6088, \quad S_u = Su = 55, \quad S_d = Sd = 45.4545,$$

$$C_u = \max\{0, S_u - K\} = 5, \quad \text{and} \quad C_u = \max\{0, S_u = K\} = 0.$$

- The call and put option values are

$$C = e^{-rT}\mathbb{E}^q[C_T] = e^{-rT}\left[qC_u + (1-q)C_d\right] = 2.9688$$

$$P = e^{-rT}\mathbb{E}^q[P_T] = e^{-rT}\left[qP_u + (1-q)P_d\right] = 1.7343.$$

1-period

Example (Continued)

Some Python code:

```
import numpy as np
S = 50; K = 50; r = 0.05; T = 1/2
u = 1.1; d = 1/u
q = (np.exp(r*T)-d)/(u-d)
Su = S*u; Sd = S*d
Cu = max(0, Su-K); Cd = max(0, Sd-K)
Pu = max(0, K-Su); Pd = max(0, K-Sd)
C = np.exp(-r*T)*(q*Cu + (1-q)*Cd)
P = np.exp(-r*T)*(q*Pu + (1-q)*Pd)
```

1-period

Question: Where do we get the up u and down d factors from?

1. The **Cox, Ross, Rubinstein** (**CRR**) scheme derives

$$u = e^{\sigma\sqrt{T}} \qquad \text{and} \qquad d = \frac{1}{u} = e^{-\sigma\sqrt{T}},$$

   where σ is the volatility parameter.

2. The **Jarrow-Rudd** (**JR**) scheme derives

$$u = e^{(r-\sigma^2/2)T+\sigma\sqrt{T}} \qquad \text{and} \qquad d = e^{(r-\sigma^2/2)T-\sigma\sqrt{T}}.$$

More general and complex schemes have also been proposed.

1-period

Example

Continuing with the previous example, suppose $\sigma = 0.25$. Then:

- CRR: $u = 1.1934$, $d = 0.838$ and

$$C = 4.9708 \qquad \text{and} \qquad P = 3.7363.$$

- JR: $u = 1.2046$, $d = 0.84586$ and

$$C = 4.99 \qquad \text{and} \qquad P = 3.757.$$

Note that the Black-Scholes prices are $C = 4.13$ and $P = 2.896$.

The multi-period binomial model improves these prices.

1-period

Example (Continued)

Some Python code for the JR scheme:

```
import numpy as np
S = 50; K = 50; r = 0.05; T = 1/2
u = np.exp((r - 0.5*sigma**2)*T + sigma*np.sqrt(T))
d = np.exp((r - 0.5*sigma**2)*T - sigma*np.sqrt(T))
q = (np.exp(r*T)-d)/(u-d)
Su = S*u; Sd = S*d
Cu = max(0, Su-K); Cd = max(0, Sd-K)
Pu = max(0, K-Su); Pd = max(0, K-Sd)
C = np.exp(-r*T)*(q*Cu + (1-q)*Cd)
P = np.exp(-r*T)*(q*Pu + (1-q)*Pd)
```

Multi-period

For the multi-period binomial model we simply:

1. Discretise the interval $[0, T]$ into $N + 1$ equally-spaced dates $\{t_0, t_1, \ldots, t_N\}$ with $t_0 = 0$, $t_N = T$ and spacing $\mathrm{d}t = \frac{T}{N}$.
2. Build an asset price tree starting at S as per the next slide.
3. Calculate the option premium by recursively stepping backwards in time in the tree and using the 1-period pricing formula each step.

To build the asset price tree, we note that the asset price can go up by a factor of u or down by a factor of d at each time step.

- We build the asset price tree as follows:

Multi-period

- At expiry (time T) there is $N+1$ underlying asset prices

  $S_{iN} = Su^i d^{N-i} \quad \text{for } i = 0, 1, \ldots, N.$

  Asset price S_{iN} took i up steps and thus $N-i$ down steps.
- At time step j there is $j+1$ asset prices

  $S_{ij} = Su^i d^{j-i} \quad \text{for } i = 0, 1, \ldots, j.$

  Asset price S_{ij} took i up steps and thus $j-i$ down steps.

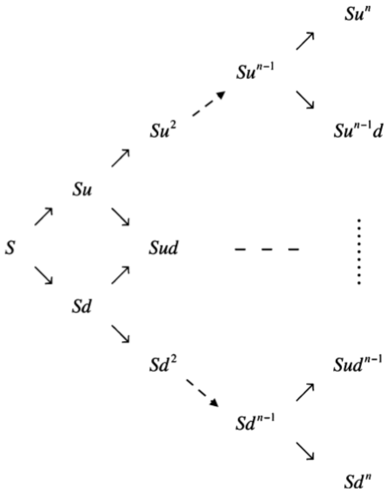


Figure: Underlying asset price tree.

Multi-period

We calculate call prices as follows:

- The call option payoffs at expiry (time T) are

$$C_{iN} = \max\{0, S_{iN} - K\}$$

  for $i = 0, 1, \ldots, N$.

- We let C_{ij} denote the call price at time step j when the underlying asset took i up steps (and thus $j - i$ down steps).

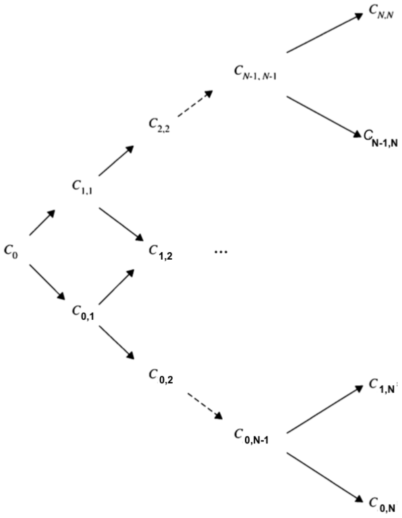


Figure: Call option price tree.

Multi-period

- Then working backwards, starting with the expiry payoffs C_{iN} for $i = 0, 1, \ldots, N$, we recursively calculate the call prices C_{ij} using the 1-period binomial pricing formula as follows:
  - Suppose we want to calculate the call price C_{ij} at time step j.
  - From the previous iteration in the recursion, we know the call prices $C_{i+1,j+1}$ (up step) and $C_{i,j+1}$ (down step) at time step $j+1$.
  - We calculate C_{ij} as follows:

$$C_{ij} = e^{-rdt}\left[qC_{i+1,j+1} + (1-q)C_{i,j+1}\right],$$

where the risk-neutral probability is given by

$$q = \frac{e^{rdt} - d}{u - d}.$$

Multi-period

Remark

1. In the CRR scheme, the up and down factors are
$$u = e^{\sigma\sqrt{\mathrm{d}t}} \qquad \text{and} \qquad d = \frac{1}{u} = e^{-\sigma\sqrt{\mathrm{d}t}}.$$
2. In the Jarrow-Rudd scheme, they are given by
$$u = e^{(r-\sigma^2/2)\mathrm{d}t+\sigma\sqrt{\mathrm{d}t}} \qquad \text{and} \qquad d = e^{(r-\sigma^2/2)\mathrm{d}t-\sigma\sqrt{\mathrm{d}t}}.$$

Multi-period

Example

We continue with the previous example, where the time to expiry was 6 months. Let's use daily time steps so that $N = 180$ and thus $\mathrm{d}t = \frac{T}{180} = 0.00278$. The CRR scheme results in

$$u = e^{\sigma\sqrt{\mathrm{d}t}} = 1.01326, \quad d = \frac{1}{u} = 0.9869, \quad q = \frac{e^{r\mathrm{d}t} - d}{u - d} = 0.502.$$

The below Python calculates premiums of

$$C = 4.125 \qquad \text{and} \qquad P = 2.891,$$

closer to the Black-Scholes prices of $C = 4.13$ and $P = 2.8955$.

Multi-period

Example (Continued)

If we use the Jarrow-Rudd scheme, we get

$$C = 4.134 \qquad \text{and} \qquad P = 2.899.$$

If we let $N = 1,000$, the JR scheme gives

$$C = 4.13 \qquad \text{and} \qquad P = 2.8956.$$

Some Python code:

Multi-period

Example

```python
import numpy as np
from scipy.stats import norm
S = 50; K = 50; r = 0.05; T = 1/2; sigma = 0.25
N = 1000; dt = T/N
#u = np.exp(sigma*np.sqrt(dt)); d = 1/u # CRR
u = np.exp((r - 0.5*sigma**2)*dt + sigma*np.sqrt(dt)) # JR
d = np.exp((r - 0.5*sigma**2)*dt - sigma*np.sqrt(dt)) # JR
q = (np.exp(r*dt)-d)/(u-d)
C = np.zeros([N+1, N+1])
P = np.zeros([N+1, N+1])
for i in range(N+1):
    St = S*(u**i)*d**(N-i)
    C[i,N] = max(0, St-K)
    P[i,N] = max(0, K-St)
for j in reversed(range(N)):
   for i in range(j+1):
        C[i,j] = np.exp(-r*dt)*(q*C[i+1, j+1] + (1-q)*C[i, j+1])
        P[i,j] = np.exp(-r*dt)*(q*P[i+1, j+1] + (1-q)*P[i, j+1])
C0 = C[0,0] # call premium
P0 = P[0,0] # put premium
```

Monte Carlo method

The binomial model is useful in some option pricing contexts, such as for American options (although the PDE approach is preferred here), but the Monte Carlo approach is beneficial in other option pricing contexts.

Monte Carlo method

The Monte Carlo method is motivated by the risk-neutral approach:

- The underlying asset follows geometric Brownian motion

$$S_t = S e^{(r-\frac{1}{2}\sigma^2)t+\sigma\sqrt{t}Z} \qquad \text{for } 0 \leq t \leq T,$$

  where Z is a standard normal random variable.

- Option prices are given by

$$C = e^{-rT}\mathbb{E}\big[\max\{0, S_T - K\}\big] \quad \text{and} \quad P = e^{-rT}\mathbb{E}\big[\max\{0, K - S_T\}\big],$$

where S_T is log-normally distributed as per geometric Brownian motion.

- To price options via the Monte Carlo method we:

Monte Carlo method

- Simulate N sample paths of geometric Brownian motion of length $M+1$ as per next slide to get N asset prices S_{iM} for $i = 1, \ldots, N$.
- Option prices are then given by

$$C = e^{-rT}\frac{1}{N}\sum_{i=1}^{N}\max\{0, S_{iM} - K\}$$

$$P = e^{-rT}\frac{1}{N}\sum_{i=1}^{N}\max\{0, K - S_{iM}\}.$$

Monte Carlo method

Recall that we simulate geometric Brownian motion (GBM) as follows:

- Discretise the interval $[0, T]$ into $M + 1$ equally-spaced dates $\{t_0, t_1, \ldots, t_M\}$ with $t_0 = 0$, $t_M = T$ and spacing $\mathrm{d}t = \frac{T}{M}$.
- Let S_{ij} be the underlying's price at date t_j for path i.
- Iteratively simulate GBM over each subinterval $[t_{j-1}, t_j]$ to create path i by starting at $S_{i0} = S$ and setting

$$\boxed{S_{ij} = S_{i,j-1} e^{(r - \frac{1}{2}\sigma^2)\mathrm{d}t + \sigma\sqrt{\mathrm{d}t} Z_{ij}} \qquad \text{for } j = 1, \ldots, M,}$$

  with Z_{ij} independent standard normal random variables.

Monte Carlo method

Example

We again continue with the same example.

```python
import numpy as np
from scipy.stats import norm
S = 50; K = 50; r = 0.05; T = 1/2; sigma = 0.25
N = 1000; M = 1000; dt = T/M
St = np.zeros([N, M+1]); St[:,0] = S # rows=paths, columns=time steps
CT = np.zeros(N)
PT = np.zeros(N)
for i in range(N):
    for j in range(1, M+1):
        Z = norm.rvs()
        St[i,j] = St[i,j-1]*np.exp((r-0.5*sigma**2)*dt + sigma*np.sqrt(dt)*Z)
    CT[i] = max(0, St[i,M]-K)
    PT[i] = max(0, K-St[i,M])
C = np.exp(-r*T)*np.mean(CT)
P = np.exp(-r*T)*np.mean(PT)
```

This code gives $C = 4.22$ and $P = 2.92$.

Monte Carlo method

Example (Continued)

Increasing the number of sample paths N and time steps M to $N = 15,000$ and $M = 2000$ yields $C = 4.07$ and $P = 2.887$.

Remark

There's a number of variance reduction techniques (and vectorised and parallel coding) that make Monte Carlo methods much more accurate, powerful and computationally efficient.

Summary