FINM3405 Derivatives and Risk Management

Week 8: Pricing American and path-dependent options

Dr Godfrey Smith

September 9, 2024

Contents

Introduction

Last week we introduced the binomial model and the Monte Carlo method as numerical or computational techniques for pricing European options. We did it in the context of plain vanilla calls and puts (and for binary options in the tutorial question), where we have Black-Scholes prices against which to compare the accuracy of the numerical pricing methods. The ideas behind numerical methods are to

1. use more advanced, complex option pricing models and
2. price more complex, exotic options and other derivatives.

We don't cover point 1 in FINM3405 since it involves doing some maths.

Introduction

This week we use the binomial and Monte Carlo methods to price:

1. American options (don't have closed-form Black-Scholes prices).
2. Some simple exotic path-dependent options (some actually do have closed-form solutions but we largely ignore them).

The binomial model and Monte Carlo method (and PDE approach) have strengths and weaknesses. One should always use the computational technique that is most tractable, practical, accurate, efficient and numerically stable for the pricing application at hand, and we'll do this.

- Readings: Chapters 13, 21.1, 21.6 and 26 of Hull.

We start with American options:

American options

An **American option** gives the holder the right (but not the obligation) to exercise it at any point up to and including the expiry date T.

- This **early exercise feature** actually introduces quite a bit of mathematical complexity into the pricing of American options.
- We immediately run into the problem of not having a closed-form solution for the price of an American option.

Remark

Recall that the price of American and European call options coincide on non-dividend-paying stocks.

- Can use the Black-Scholes model for these American calls.

American options

In these notes we'll use the binomial model for pricing American options since it's conceptually very simple and is also very accurate.

- Modifying the Monte Carlo approach for American options is conceptually and mathematically a bit of a stretch for FINM3405.
- The PDE approach is extensively used for American options but it's mathematically a bit beyond FINM3405.

American options

A very simple adjustment needs only be made to our previous binomial model approach of stepping backwards through the asset price tree in order to price American options, and it is motivated by some of the **pricing bounds** previously noted for American options:

- American options are worth at least as much as European options:

$$C^{\text{Am}} \geq C^{\text{Eu}} \qquad \text{and} \qquad P^{\text{Am}} \geq P^{\text{Eu}}.$$

- American options are worth at least as much as their intrinsic value:

$$C_t^{\text{Am}} \geq \max\{0, S_t - K\} \qquad \text{and} \qquad P_t^{\text{Am}} \geq \max\{0, K - S_t\}.$$

American options

The adjustment to price American options is:

A each node, set the American option price equal to the maximum of the "1-step European price" and the option's intrinsic value:

- Recall we discretise the interval $[0, T]$ into $N+1$ equally-spaced dates $\{t_0, \ldots, t_N\}$ with $t_0 = 0$, $t_N = T$ and spacing $\mathrm{d}t = \frac{T}{N}$.
- At date t_j there are $j+1$ asset prices $S_{ij} = Su^i d^{j-1}$ for $i = 0, \ldots, j$.
- The (American and European) option payoffs at expiry are

$$C_{iN}^{\mathrm{Am}} = \max\{0, S_{iN} - K\} \qquad \text{and} \qquad P_{iN}^{\mathrm{Am}} = \max\{0, K - S_{iN}\}.$$

American options

- The "1-step European" option prices at node ij on the tree are

$$C_{ij}^{\text{Eu}} = e^{-r dt}\left[q C_{i+1,j+1}^{\text{Am}} + (1-q) C_{i,j+1}^{\text{Am}}\right],$$
$$P_{ij}^{\text{Eu}} = e^{-r dt}\left[q P_{i+1,j+1}^{\text{Am}} + (1-q) P_{i,j+1}^{\text{Am}}\right].$$

- The call and put option intrinsic values at node ij are

$$C_{ij}^{\text{iv}} = \max\{0, S_{ij} - K\} \qquad \text{and} \qquad P_{ij}^{\text{iv}} = \max\{0, K - S_{ij}\}.$$

- Hence, the American option prices at node ij are

$$\boxed{C_{ij}^{\text{Am}} = \max\{C_{ij}^{\text{Eu}}, C_{ij}^{\text{iv}}\} \qquad \text{and} \qquad P_{ij}^{\text{Am}} = \max\{P_{ij}^{\text{Eu}}, P_{ij}^{\text{iv}}\}.}$$

American options

Example

Again let $S = 50$, $K = 50$, $r = 5\%$, $T = \frac{1}{2}$, $\sigma = 25\%$ and $q = 0$.

```python
import numpy as np
S = 50; K = 50; r = 0.05; T = 1/2; sigma = 0.25; N = 10000; dt = T/N
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)
# asset price tree
St = np.zeros([N+1, N+1])*np.nan
for j in range(N+1):
    for i in range(j+1):
        St[i,j] = S*(u**i)*d**(j-i)
# option premiums
Ct = np.zeros([N+1, N+1])*np.nan
Pt = np.zeros([N+1, N+1])*np.nan
for i in range(N+1): # option payoffs at expiry
    ST = St[i,N]
    Ct[i,N] = max(0, ST-K)
    Pt[i,N] = max(0, K-ST)
```

American options

Example (Continued)

```
for j in reversed(range(N)): # step backwards through the tree
   for i in range(j+1):
       Ceu = np.exp(-r*dt)*(q*Ct[i+1, j+1]+(1-q)*Ct[i, j+1]) # 1 step European call
       Peu = np.exp(-r*dt)*(q*Pt[i+1, j+1]+(1-q)*Pt[i, j+1]) # 1 step European put
       Civ = max(0, St[i,j]-K) # call intrinsic value
       Piv = max(0, K-St[i,j]) # put intrinsic value
       Ct[i,j] = max(Ceu, Civ) # American call
       Pt[i,j] = max(Peu, Piv) # American put
C = Ct[0,0]
P = Pt[0,0]
```

Thos code gives $C^{\text{Am}} = 4.13$ and $P^{\text{Am}} = 3.011$. Recall that the Black-Scholes European prices were $C = 4.13$ and $P = 2.8955$, which make sense since European and American call prices coincide on non-income paying assets.

Incorporating dividends

Let's consider a continuous dividend yield y (sorry for the notation).

- The European binomial model with a dividend yield is the same as without one except we set the risk-neutral probability to

$$\boxed{q = \frac{e^{(r-y)\mathrm{d}t} - d}{u - d}.}$$

- We also require the technical condition $d < e^{(r-y)\mathrm{d}t} < u$.
- The CRR scheme is unchanged with $u = e^{\sigma\sqrt{\mathrm{d}t}}$ and $d = \frac{1}{u}$.
- But the Jarrow-Rudd scheme becomes

$$\boxed{u = e^{(r-y-\frac{1}{2}\sigma^2)\mathrm{d}t+\sigma\sqrt{\mathrm{d}t}} \qquad \text{and} \qquad d = e^{(r-y-\frac{1}{2}\sigma^2)\mathrm{d}t-\sigma\sqrt{\mathrm{d}t}}.}$$

Incorporating dividends

Example

Use the same parameters but also let the continuously compounded dividend yield be $y = 6\%$. Making the required simple modifications to the above Python code, with $N = 10,000$ we get $C^{\text{Am}} = 3.34$ and $P^{\text{Am}} = 3.554$. We also calculate the Black-Scholes European option prices to be $C = 3.306$ and $P = 3.549$. Now both the American call and put prices are greater than that of their European counterparts.

Path-dependent options

We now numerically price some European path-dependent options.

- A European option is **path dependent** if its payoff is calculated from the underlying asset's path travelled over the option's life.

We consider the following path-dependent options:

- Chooser options.
- Lookback options.
- Barrier options.
- Asian options.

The binomial model was well suited for pricing American options.

- The binomial model is well suited for chooser options.
- Monte Carlo is well suited to the other path-dependent options.

Chooser options

A European **chooser option** is similar to a plain vanilla European option except that it allows the holder to choose at some date t_{choose} (the **choice date**) over the option's life if the option is a call or a put!

- Of course the choice date satisfies $0 < t_{\text{choose}} < T$.

On the choice date, it is rational for the holder to choose the option to be whichever out of the call or put has the highest value on that date.

Chooser options

Hence, to adapt the binomial European option pricing Python code to handle the choice date we do the following:

When working backwards recursively through the binomial tree to calculate the European call and put option prices, on each node of the choice date select the chooser option's value as the maximum of the call and put values. From there work backwards recursively as per normal for the chooser option, ignoring the call and put.

Python code to price a chooser option using the same parameters:

Chooser options

Example

```python
import numpy as np
import math
S = 50; K = 50; sigma = 0.25; r = 0.05; T = 1/2; N = 10000; dt = T/N
dates = np.linspace(0,T,N+1) # asset price tree dates
cdate = 1/4 # the choice date
cidx = math.ceil(N*cdate/T) # the index of the choice date
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)
# expiry payoffs
Ct = np.zeros([N+1 ,N+1])*np.nan
Pt = np.zeros([N+1, N+1])*np.nan
for i in range(N+1):
    ST = S*(u**i)*d**(N-i)
    Ct[i,N]=max(0, ST-K)
    Pt[i,N]=max(0, K-ST)

```

Chooser options

Example (Continued)

```
# chooser option value
Vt = np.zeros([N+1, N+1])*np.nan
for j in reversed(range(N)):
    if j>cidx: # after the choice date
        for i in range(j+1): # running value of both puts and calls
            Ct[i,j] = np.exp(-r*dt)*(q*Ct[i+1, j+1]+(1-q)*Ct[i, j+1])
            Pt[i,j] = np.exp(-r*dt)*(q*Pt[i+1, j+1]+(1-q)*Pt[i, j+1])
    if j==cidx: # at the choice date
        for i in range(j+1): # make the choice
            Ct[i,j] = np.exp(-r*dt)*(q*Ct[i+1, j+1]+(1-q)*Ct[i, j+1])
            Pt[i,j] = np.exp(-r*dt)*(q*Pt[i+1, j+1]+(1-q)*Pt[i, j+1])
            Vt[i,j] = max(Ct[i,j], Pt[i,j]) # take the max value on choice date
    if j<cidx: # before the choice date
        for i in range(j+1): # only calculate choose option price
            Vt[i,j] = np.exp(-r*dt)*(q*Vt[i+1, j+1]+(1-q)*Vt[i, j+1])
V = Vt[0,0]
```

Chooser options

Example

Note here that the expiry date is `T = 1/2` so 6 months, and the choice date is `cdate = 1/4` so 3 months. With the number of periods set to `N = 10000`, we calculate the choice index to be `cidx = 5,000` as expected. The above code gives the chooser option value as $V = 6.023$ and we recall that the Black-Scholes European option values are $C = 4.13$ and $P = 2.8956$.

Lookback options

A **lookback** option's payoff at expiry depends on the maximum or minimum prices of the underlying asset over the life of the option. We distinguish between two types of lookback options:

1. Fixed-strike lookback options.
2. Floating-strike lookback options.

Fixed-strike lookback options

European **fixed-strike** lookback options are very similar to plain vanilla European options except the "final price" of the underlying asset used in calculating the payoff is not the asset price S_T but instead the maximum $S_{\max}$ or minimum $S_{\min}$ asset price reached over the option's life:

$$\text{call payoff} = \max\{0, S_{\max} - K\},$$

$$\text{put payoff} = \max\{0, K - S_{\min}\}.$$

To price fixed-strike lookbacks via Monte Carlo we first simulate N asset price paths under geometric Brownian motion as per usual:

Fixed-strike lookback options

- 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 $\mathrm{d}t = \frac{T}{M}$.
- Calculate N asset price paths $\{S_{i0}, S_{i1}, S_{i2}, \ldots, S_{iM}\}$ for $i = 1, \ldots, N$, with price S_{ij} of path i at date t_j calculated by setting $S_{i0} = S$ and simulating geometric Brownian motion:

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

where Z_{ij} are independent standard normal random variables.

Fixed-strike lookback options

The maximum and minimum asset prices of path i are given by

$$S_{i,\max} = \max_{j=0,\ldots,M} S_{ij} \qquad \text{and} \qquad S_{i,\min} = \min_{j=0,\ldots,M} S_{ij}.$$

The fixed-strike lookback option payoffs for path i are given by

$$\text{call payoff}_i = \max\{0, S_{i,\max} - K\},$$

$$\text{put payoff}_i = \max\{0, K - S_{i,\min}\}.$$

Fixed-strike lookback options

The Monte Carlo option prices are then simply

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

We now calculate Monte Carlo prices of fixed-strike lookback options using the same parameters as we have been using:

Fixed-strike lookback options

Example

```python
import numpy as np
from scipy.stats import norm
S = 50; K = 50; sigma = 0.25; r = 0.05; T = 1/2
N = 1000; M = 1000; dt = T/M
St = np.zeros([N, M+1]) # rows=paths, columns=time steps
St[:,0] = S # each path starts at S
CT = np.zeros(N); PT = np.zeros(N)
for i in range(N):
    for j in range(1, M+1):
        St[i,j] = St[i,j-1]*np.exp((r-0.5*sigma**2)*dt + sigma*np.sqrt(dt)*norm.rvs())
    Smax = np.max(St[i, 0:M]) # maximum asset price of path i
    Smin = np.min(St[i, 0:M]) # minimum asset price of path i
    CT[i] = max(0, Smax-K) # fixed strike call payoff
    PT[i] = max(0, K-Smin) # fixed strike put payoff
C = np.exp(-r*T)*np.mean(CT)
P = np.exp(-r*T)*np.mean(PT)
```

We get $C = 8.21$ and $P = 5.822$.

Floating-strike lookback options

European **floating-strike** lookback options differ from fixed-strike options by instead setting the strike price K to be the maximum $S_{\max}$ or minimum $S_{\min}$ asset prices over the life of the option. Their payoffs are

$$\boxed{\begin{aligned} \text{call payoff} &= \max\{0, S_T - S_{\min}\}, \\ \text{put payoff} &= \max\{0, S_{\max} - S_T\}. \end{aligned}}$$

Floating-strike lookback options

So, after calculating the N asset price paths $\{S_{i0}, S_{i1}, \ldots, S_{iM}\}$ for $i = 1, \ldots, N$ by simulating geometric Brownian motion, and calculating the minimum $S_{i,\min}$ and maximum $S_{i,\max}$ prices for each path, the Monte Carlo floating-strike lookback option prices are then simply

$$
\begin{aligned}
C &= e^{-rT}\frac{1}{N}\sum_{i=1}^{N}\max\{0, S_{iM} - S_{i,\min}\},\\
P &= e^{-rT}\frac{1}{N}\sum_{i=1}^{N}\max\{0, S_{i,\max} - S_{iM}\},
\end{aligned}
$$

and the above Python code is only slightly modified for floating-strikes:

Floating-strike lookback options

Example

```python
import numpy as np
from scipy.stats import norm
S = 50; K = 50; sigma = 0.25; r = 0.05; T = 1/2
N = 1000; M = 1000; dt = T/M
St = np.zeros([N, M+1]) # rows=paths, columns=time steps
St[:,0] = S # each path starts at S
CT = np.zeros(N); PT = np.zeros(N)
for i in range(N):
    for j in range(1, M+1):
        St[i,j] = St[i,j-1]*np.exp((r-0.5*sigma**2)*dt + sigma*np.sqrt(dt)*norm.rvs())
    ST = St[i,M] # path i asset price at expiry
    Smax = np.max(St[i, 0:M]) # maximum asset price of path i
    Smin = np.min(St[i, 0:M]) # minimum asset price of path i
    CT[i] = max(0, ST-Smin) # floating strike call payoff
    PT[i] = max(0, Smax-ST) # floating strike put payoff
C = np.exp(-r*T)*np.mean(CT)
P = np.exp(-r*T)*np.mean(PT)
```

Here we get $C = 7.2385$ and $P = 6.2675$.

Barrier options

European barrier options are another interesting variation of the path-dependence theme. They are effectively plain vanilla European options whose payoffs “knock in” or “knock-out” if the price of the underlying asset hits a barrier B at some point over the life of the option:

1. **Knock-in** barrier options: The payoff “knocks in” if the barrier B is hit, meaning that the option starts its life deactivated and then is activated if B is hit at some point.
2. **Knock-out** barrier options: The payoff “knocks out” if the barrier B is hit, meaning that the option starts its life as activated and then is deactivated if B is hit at some point.

Knock-in barrier options

European **knock-in** options are activated if the price of the underlying asset hits the barrier B at some point of the option's life.

- The payoffs are then the usual $\max\{0, S_T - K\}$ for a call and $\max\{0, K - S_T\}$ for a put.

If the barrier is never hit, the option is never activated and expires worthless. There's two kinds of knock-in options depending on the relation between B and S (they typically start at-the-money):

1. **Up-and-in** options set $B > S$, noting that typically $K \approx S$.
2. **Down-and-in** options set $B < S$, noting that typically $K \approx S$.

The Monte Carlo pricing of knock-in options is a simple modification to the above, simply to check each path to see if the barrier was hit.

Knock-in barrier options

Example (Up-and-in barrier option)

```python
import numpy as np
from scipy.stats import norm
S = 50; K = 50; sigma = 0.25; r = 0.05; T = 1/2; B = 60 # set the barrier above S
N = 1000; M = 1000; dt = T/M
St = np.zeros([N, M+1]) # rows=paths, columns=time steps
St[:,0] = S # each path starts at S
CT = np.zeros(N); PT = np.zeros(N)
for i in range(N):
    for j in range(1, M+1):
        St[i,j] = St[i,j-1]*np.exp((r-0.5*sigma**2)*dt + sigma*np.sqrt(dt)*norm.rvs())
    ST = St[i,M] # final asset prices at expiry
    Smax = np.max(St[i, 0:M]) # maximum asset price of path i
    CT[i] = max(0,ST-K) if Smax>=B else 0 # payoff if asset hits the barrier
    PT[i] = max(0,K-ST) if Smax>=B else 0 # payoff if asset hits the barrier
C = np.exp(-r*T)*np.mean(CT)
P = np.exp(-r*T)*np.mean(PT)
```

Here we get $C = 3.2593$ and $P = 0.0701$.

Knock-in barrier options

Example (Down-and-in barrier option)

```python
import numpy as np
from scipy.stats import norm
S = 50; K = 50; sigma = 0.25; r = 0.05; T = 1/2; B = 40 # set the barrier below S
N = 1000; M = 1000; dt = T/M
St = np.zeros([N, M+1]) # rows=paths, columns=time steps
St[:,0] = S # each path starts at S
CT = np.zeros(N); PT = np.zeros(N)
for i in range(N):
    for j in range(1, M+1):
        St[i,j] = St[i,j-1]*np.exp((r-0.5*sigma**2)*dt + sigma*np.sqrt(dt)*norm.rvs())
    ST = St[i,M] # final asset prices at expiry
    Smin = np.min(St[i, 0:M]) # minimum asset price of path i
    CT[i] = max(0,ST-K) if Smin<=B else 0 # payoff if asset hits the barrier
    PT[i] = max(0,K-ST) if Smin<=B else 0 # payoff if asset hits the barrier
C = np.exp(-r*T)*np.mean(CT)
P = np.exp(-r*T)*np.mean(PT)
```

We get $C = 0.0102$ and $P = 1.9041$.

Knock-out barrier options

European **knock-out** options are deactivated if the price of the underlying asset hits the barrier B at some point of the option's life.

- The option then expires worthless.

If the barrier is never hit, the option stays alive and the payoffs are the usual $\max\{0, S_T - K\}$ for a call and $\max\{0, K - S_T\}$ for a put. There's two kinds of knock-ins depending on the relation between B and S:

1. **Up-and-out** options set $B > S$, noting that typically $K \approx S$.
2. **Down-and-out** options set $B < S$, noting that typically $K \approx S$.

The Monte Carlo pricing of knock-out options is a simple modification to the above for knock-in options, namely a reversal of the logical conditions on the payoffs if the barrier was hit.

Knock-out barrier options

Example (Up-and-out barrier option)

```python
S = 50; K = 50; sigma = 0.25; r = 0.05; T = 1/2; B = 60 # set the barrier above S
N = 1000; M = 1000; dt = T/M
St = np.zeros([N, M+1]) # rows=paths, columns=time steps
St[:,0] = S # each path starts at S
CT = np.zeros(N); PT = np.zeros(N)
for i in range(N):
    for j in range(1, M+1):
        St[i,j] = St[i,j-1]*np.exp((r-0.5*sigma**2)*dt + sigma*np.sqrt(dt)*norm.rvs())
    ST = St[i,M] # final asset prices at expiry
    Smax = np.max(St[i, 0:M]) # maximum asset price of path i
    CT[i] = 0 if Smax>=B else max(0,ST-K) # payoff unless asset hits the barrier
    PT[i] = 0 if Smax>=B else max(0,K-ST) # payoff unless asset hits the barrier
C = np.exp(-r*T)*np.mean(CT)
P = np.exp(-r*T)*np.mean(PT)
```

We get $C = 0.7665$ and $P = 3.10$.

Knock-out barrier options

Example (Down-and-out barrier option)

```python
S = 50; K = 50; sigma = 0.25; r = 0.05; T = 1/2; B = 40 # set the barrier below S
N = 1000; M = 1000; dt = T/M
St = np.zeros([N, M+1]) # rows=paths, columns=time steps
St[:,0] = S # each path starts at S
CT = np.zeros(N); PT = np.zeros(N)
for i in range(N):
    for j in range(1, M+1):
        St[i,j] = St[i,j-1]*np.exp((r-0.5*sigma**2)*dt + sigma*np.sqrt(dt)*norm.rvs())
    ST = St[i,M] # final asset prices at expiry
    Smin = np.min(St[i, 0:M]) # minimum asset price of path i
    CT[i] = 0 if Smin<=B else max(0,ST-K) # payoff unless asset hits the barrier
    PT[i] = 0 if Smin<=B else max(0,K-ST) # payoff unless asset hits the barrier
C = np.exp(-r*T)*np.mean(CT)
P = np.exp(-r*T)*np.mean(PT)
```

We get $C = 4.31$ and $P = 1.064$.

Asian options

A European **Asian** or **average option's** payoffs depend on the average underlying asset price $\bar{S}$ over the life of the option.

- They'e called "Asian" options simply because they were first developed in the foreign exchange markets of Japan.
- There is various methods of calculating the historical averages used:
  - Typically either continuous or discrete (say daily) averages.
  - We assume continuous averages, which get more accurately approximated as our numerical method takes smaller step sizes.

There's two general types of Asian options:

1. Fixed-strike Asian options.
2. Floating-strike Asian options.

Fixed-strike Asian options

European **fixed-strike** Asian options are very similar to plain vanilla European options except the “final price” of the underlying asset used in calculating an option’s payoff is not the asset price itself but the average asset price $\bar{S}$ over the life of the option. Their payoffs are

$$\text{call payoff} = \max\{0, \bar{S} - K\},$$
$$\text{put payoff} = \max\{0, K - \bar{S}\}.$$

It is simple to use Monte Carlo simulation to price them:

Fixed-strike Asian options

After calculating the N price paths $\{S_{i0}, S_{i1}, \ldots, S_{iM}\}$ for $i = 1, \ldots, N$ by simulating geometric Brownian motion, the **average price** of path i is

$$\bar{S}_i = \frac{1}{M} \sum_{j=0}^{M} S_{ij}$$

(arithmetic average). The payoffs for path i are

$$\text{call payoff}_i = \max\{0, \bar{S}_i - K\}$$
$$\text{put payoff}_i = \max\{0, K - \bar{S}_i\}.$$

Fixed-strike Asian options

Remark

The geometric average price of path i would be $\bar{S}_i = \left(\prod_{j=0}^{M} S_{ij}\right)^{\frac{1}{M}}$.

The Monte Carlo option prices are then simply

$$C = e^{-rT}\frac{1}{N}\sum_{i=1}^{N}\max\{0, \bar{S}_i - K\},$$

$$P = e^{-rT}\frac{1}{N}\sum_{i=1}^{N}\max\{0, K - \bar{S}_i\},$$

and again, the Python code is simple:

Fixed-strike Asian options

Example

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

Here we get $C = 2.382$ and $P = 1.694$.

Average-strike Asian options

European **average-strike** Asian options differ from fixed-strike Asian options by instead setting the strike price $K = \bar{S}$ to be the average asset price $\bar{S}$ over the life of the option. So, their payoffs are simply

$$\begin{aligned} \text{call payoff} &= \max\{0, S_T - \bar{S}\}, \\ \text{put payoff} &= \max\{0, \bar{S} - S_T\}. \end{aligned}$$

The payoffs for path i are

$$\begin{aligned} \text{call payoff}_i &= \max\{0, S_{iM} - \bar{S}_i\} \\ \text{put payoff}_i &= \max\{0, \bar{S}_i - S_{iM}\}. \end{aligned}$$

Average-strike Asian options

Example (Continued)

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

In this case we get $C = 2.589$ and $P = 1.75$.

Summary