Computations in Finance Assessment 1

R

February 11, 2026

Contents

Contents

1 Question 1

We have to implement a function that inputs the typical inputs of a path independent binomial market for a European option. In this case, `S_0, r, sigma, T, K, M`. These variables all need to be positive integers or floats, and M specifically must be greater than or equal to 1. We also need to ensure that booleans are not allowed through, since these are treated as integers by Python. After validating that all inputs are numerical and within bounds, we can check for the presence of arbitrage (for continuously compounded data). In order to benefit the user, it is best that every bad entry has a specific warning, instead of the user being forced to retry the module for every bad input. As such, we use basic if checks and a `_fail` check that returns an error code `-1`.

```
def input_data (S_0, r, sigma, T, K, M):
  """Validates variable entries for Binomial Model
  Expects initial price, interest rate, volatility,
  time till maturity, strike price, and number of
  time periods."""
  _fail = False

  if not (isinstance(S_0, (int, float)) and not isinstance(S_0,
       bool)) or (S_0 <= 0):
      print ("Initial stock price must take a positive
         numerical value,")
      _fail = True
  if not (isinstance(r, (int, float)) and not isinstance(r,
      bool)) or (r <= 0):
      print ("Interest rate must take a positive numerical
          value,")
      _fail = True

  if not (isinstance(sigma, (int, float)) and not isinstance(
      sigma, bool)) or (sigma <= 0):
      print ("Volatility can only be a positive numerical,")
      _fail = True

  if not (isinstance(T, (int, float)) and not isinstance(T,
      bool)) or (T <= 0):
      print ("Time period must be some positive multiple of 1,"
          )
      _fail = True

  if not (isinstance(K, (int, float)) and not isinstance(K,
      bool)) or (K <= 0):
      print ("An option can only have a positive numerical
          value,")
      _fail = True

  if not (isinstance(M, (int, float)) and not isinstance(M,
      bool)) or (M < 1):
      print ("Must be at least one time period,")
      _fail = True

  if _fail:
      print("Invalid Entry, please amend arguments.")
      return -1

  # Handling arbitrage
  delta_t = math.exp(math.log(T) - math.log(M))
  cont_rate = math.exp(r * delta_t)

  u = 1 + sigma
  d = 1 - sigma

  if  cont_rate <= d or cont_rate >= u:
    print("Model has arbitrage")
    print("Invalid Entry, please amend arguments")
    return -1

  return 0

print(input_data(4, 5, 4, 5, True, 1))
```
Listing 1: input_data function

```
An option can only have a positive numerical value,
Invalid Entry, please amend arguments.
-1
```

In order to test this concretely, I also specified a set of pytest unit tests that run on `input_data` specifically, including cases with multiple bad entries and arbitrages (Listing 2). These all passed, suggesting the function effectively guards against inputs that would lead to undefined behaviour or a crash at runtime. Since Python lacks switch statements, the function cannot be optimised much further, but using the Python `Math` library and the properties of exponents stabilises the calculation of Δt since `M` could be very large while `T` could be small.

```
@pytest.mark.parametrize("args", invalid_inputs)
def test_1(args):
    assert input_data (*args) == -1

@pytest.mark.parametrize("args", arbitrage_cases)
def test_4 (args):
  assert input_data(*args) == -1

@pytest.mark.parametrize("args", valid_inputs)
def test_2(args):
    assert input_data (*args) == 0
```

Listing 2: Unit tests on `input_data`

2 Question 2

We are tasked with making a basic path independent binomial model for a European put. We can start by copying our `input_data` function to validate inputs.

3 Algorithm

3.1 Optimisations

To start with, we won't use any lists in
A major bottleneck to complexity in the model is evaluating the function:

$$\binom{M}{i} = \frac{M!}{i!(M-i)!}$$

Which is essentially an $\mathcal{O}(M^2)$ problem since it grows in M. However, we can observe that the distribution of combinations follows Pascal's triangle, so the first value is 1, and the second is *M*. We can also show that the next term in the sequence follows the recursive relation:

$$f_n \cdot \frac{M-n}{n+1} = f_{n+1}$$

Which is linear in M. Since we are calculating a path independent European put, we only have to do the section of the tree where the strike price exceeds the value of the stock, and, further, since the combination function is symmetrical, we only have to calculate a hard maximum of 1/2 of the tree nodes. Simplifying our recursive function to only use addition:

$$f_{n+1} = \exp(\log(f_n) + \log(M-n) - \log(n+1))$$

List indexing with an address is $\mathcal{O}(1)$ so there are optimisations we can do assigning values at two points in an array for a dynamically programmed sequence.

4 Proofs

4.1 Recursive Combination Calculation

If we consider the function:

$$\binom{M}{i} = \frac{M!}{i!(M-i)!}$$

Clearly:

$$\begin{aligned} i! &= i \times (i-1) \times (i-2) \times ... \times 2 \times 1 \\ &= \prod_{j=0}^{i-1}(i-j) \end{aligned}$$

And:

$$\frac{M!}{(M-i)!} = M \times (M-1) \times (M-2) \times ... \times (M-1+2) \times (M-i+1)$$

$$= \prod_{a=1}^{i}(M-i+a)$$

Therefore:

$$\frac{M!}{i!(M-i)!} = \frac{\prod_{a=1}^{i}(M-i+a)}{\prod_{j=0}^{i-1}(i-j)}$$

We can rewrite the RHS as:

$$\begin{aligned} \exp\left(\log\left(\prod_{a=1}^{i}(M-i+a)\right) - \log 3\left(\prod_{j=0}^{i-1}(i-j)\right)\right) \\ = \exp\left(\sum_{a=1}^{i}\log(M-i+a) - \sum_{j=0}^{i-1}\log(i-j)\right) \end{aligned}$$

We then make the contention that this defines a recurrence relation for functions of i:

$$f_n \cdot \exp\left(\log(M-n) - \log(n+1)\right) = f_{n+1}$$

where;

$$= \exp\left(\sum_{a=1}^{n}\log(M-n+a) - \sum_{j=0}^{n-1}\log(n-j)\right)$$

This can easily be proven for $n \in \mathbb{N}$. First, consider $i = 1$:

$$f_1 \cdot \frac{M-1}{1+1} = f_{1+1}$$

For the first term:

$$f_1 = \exp\left(\sum_{a=1}^{1} \log(M-1+a) - \sum_{j=0}^{0} \log(n-j)\right)$$

$$= \exp\left(\log(M)\right)$$

$$f_1 = M$$

For the second term:

$$\exp\left(\log(M-n) - \log(n+1)\right) = \frac{M-1}{1+1} = \frac{M-1}{2}$$

For the third term:

$$f_2 = \exp\left(\sum_{a=1}^{2} \log(M-2+a) - \sum_{j=0}^{1} \log(2-j)\right)$$

$$= \exp\left(\log(M-1) + \log(M) - \log 2\right)$$

$$f_2 = \frac{M(M-1)}{2}$$

Returning these into the original function:

$$f_1 \cdot \frac{M-n}{n+1} = f_2$$

$$M \cdot \frac{M-1}{2} = \frac{M(M-1)}{2}$$

Which verifies the relation in the case where $n = 1$. If we then show that the relation holds between *n* and $n+1$, then it is proven for all $n \in \mathbb{N}$. Observe that:

$$f_n = \exp\left(\sum_{a=1}^{n} \log(M-n+a) - \sum_{j=0}^{n-1} \log(n-j)\right)$$

$$f_{n+1} = \exp\left(\sum_{a=1}^{n+1} \log(M-n-1+a) - \sum_{j=0}^{n} \log(n+1-j)\right)$$

If we then evaluate:

$$f_n \cdot \exp(\log(M-n) - \log(n+1))$$

$$= \exp\left(\sum_{a=1}^{n} \log(M-n+a) + \log(M-n) - \sum_{j=0}^{n} \log(n-j) - \log(n+1)\right)$$

Taking the first sum:

$$\sum_{a=1}^{n} \log(M-n+a) + \log(M-n) = \log(M-n+1) + \log(M-n+2) + ... + \log(M-n+n-1)$$

$$+\log(M-n+n)+\log(M-n)$$

$$=\log(M-n)+\log(M-n+1)+...+\log(M-1)+\log(M)$$

$$=\sum_{a=1}^{n+1}\log(M-n-1+a)$$

Taking the second:

$$\sum_{j=0}^{n}\log(n-j)+\log(n+1)$$

$$\log(n-0)+\log(n-1)+...+\log(n-n+1)+\log(n-n)+\log(n+1)$$

$$\log(n+1)+\log(n)+...+\log(1)+\log(0)$$

$$\sum_{j=0}^{n}\log(n+1-j)$$

Which means that:

$$f_n\cdot\exp\left(\log(M-n)-\log(n+1)\right)$$

$$=\exp\left(\sum_{a=1}^{n+1}\log(M-n-1+a)-\sum_{j=0}^{n}\log(n+1-j)\right)$$

Which, we can notice, is exactly the definition of f_2. $\triangle$