Finance 5350 - Homework Assignment 3

Tyler J. Brough

November 5, 2016

Assignment 3

For this assignment your job is the implement the American Binomial Optiong Pricing Model (ABOPM) within the context of the `dylan` option pricing module that we have been building.

The first step is the fork the repository as follows:

1. In your browser navigate to https://github.com/broughtj/dylan
2. In the top-right corner hit the *fork* button.
3. Check your github profile. You should now have a repository called `dylan`

The next step is to get a local clone of the repository in the usual way. Once you have obtained it you need to use the setup script to install the `dylan` module. Do this as follows:

```
$ git clone https://github.com/broughtj/dylan.git
$ cd dylan
$ python setup.py install
```

The above steps are for a Unix workflow, say using the *git bash* prompt. But you can do them using the gui for git and the Spyder IDE.

Next, test that the code installed properly by running the following test:

```
$ python test_european.py
```

You get the by-now-familiar result for the option pricing problem that we have been working on (i.e. $7.074).

The code for `test_european.py` is given below. You can think of it as a client file that imports the `dylan` option pricing module and uses it to price the call option.

```
from dylan.payoff import VanillaPayoff, call_payoff, put_payoff
from dylan.engine import BinomialPricingEngine, EuropeanBinomialPricer
from dylan.marketdata import MarketData
from dylan.option import Option
```

```
def main():
    spot = 41.0
    strike = 40.0
    rate = 0.08
    volatility = 0.30
    expiry = 1.0
    steps = 3
    dividend = 0.0

    the_call = VanillaPayoff(expiry, strike, call_payoff)
    the_bopm = BinomialPricingEngine(steps, EuropeanBinomialPricer)
    the_data = MarketData(rate, spot, volatility, dividend)

    the_option = Option(the_call, the_bopm, the_data)
    fmt = "The call option price is {0:0.3f}"
    print(fmt.format(the_option.price()))


if __name__ == "__main__":
    main()
```

Work through this code and make sure you understand each line. A few things to note:

- We use the ***Facade Pattern*** to simplify the client interface to price options. This is provided by the `Option` class.
- We use the ***Strategy Pattern*** in several different ways:
    - to provide the specific option payoffs (call and put) for the `VanillaPayoff` class.
    - to provice the specific option pricing model or numerical method within the class `PricingEngine`.
- We use the concept of an ***abstract base class (abc)*** to provide a consistent interface for our software objects `Payoff` and `PricingEngine`.
    - we leave it to concrete classes to implement the specific functionality desired.

The first step is to write a strategy function called `AmericanBinomialPricer` that is similar to the function `EuropeanBinomialPricer`, which is given below:

```
def EuropeanBinomialPricer(pricing_engine, option, data):
    """
    The binomial option pricing model for a plain vanilla European option.

    Args:
        pricing_engine (PricingEngine): a pricing method via the PricingEngine interface
        option (Payoff):                an option payoff via the Payoff interface
        data (MarketData):              a market data variable via the MarketData interface

    """

    expiry = option.expiry
    strike = option.strike
    (spot, rate, volatility, dividend) = data.get_data()
    steps = pricing_engine.steps
    nodes = steps + 1
    dt = expiry / steps
    u = np.exp((rate * dt) + volatility * np.sqrt(dt))
    d = np.exp((rate * dt) - volatility * np.sqrt(dt))
    pu = (np.exp(rate * dt) - d) / (u - d)
    pd = 1 - pu
    disc = np.exp(-rate * expiry)
    spotT = 0.0
    payoffT = 0.0

    for i in range(nodes):
        spotT = spot * (u ** (steps - i)) * (d ** (i))
        payoffT += option.payoff(spotT)  * binom.pmf(steps - i, steps, pu)
    price = disc * payoffT

    return price
```

Once you have your strategy function implemented, you should write a client script called `test_american.py` similar to the `test_european.py`.

Once you have you code implemented, push to your forked repository.