```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

import sys
import os
sys.path.append(os.path.abspath(os.path.join('..')))
from methods.portfolio import Portfolio, Derivative, Call, Put, Forward
```

Demonstrating Derivative and Portfolio Classes: Payoff Profiles and Spreads

In this notebook, I will demonstrate the functionality of the `Derivatives` and `Portfolio` classes defined in `methods.portfolio`. The goal of these classes is to help visualise portfolio payoff profiles at different maturities, easily customise a portfolio and finally visualise spreads.

Contents

Building a Portfolio

I will first demonstrate how to use the `Derivative` class. It has some subclasses such as `Call`, `Put` and `Forward` for easy use but one can also define a custom payoff function. As payoff profiles are plotted only at distinct maturity times I only include European style derivatives, which can be exercised at maturity.

```
time1 = 1
time2 = 2
call = Call(100, time1)
put = Put(120, time1)
forward = Forward(105, time2)

# We can assign these directly to the portfolio with or without a position.
# To define a long/short position for each derivative input a list of '+' and
# '-' respectively
portfolio = Portfolio([call, put])
# or
portfolio = Portfolio([call, put], ['+', '+'])
```

```
# If we want to add more derivatives:
# Long
portfolio + [forward]
# Short
portfolio - [forward]

# To remove derivatives one can look at the derivatives attribute and remove the
# desired function by index.

portfolio.remove(-1)
```

```
portfolio.derivatives
```

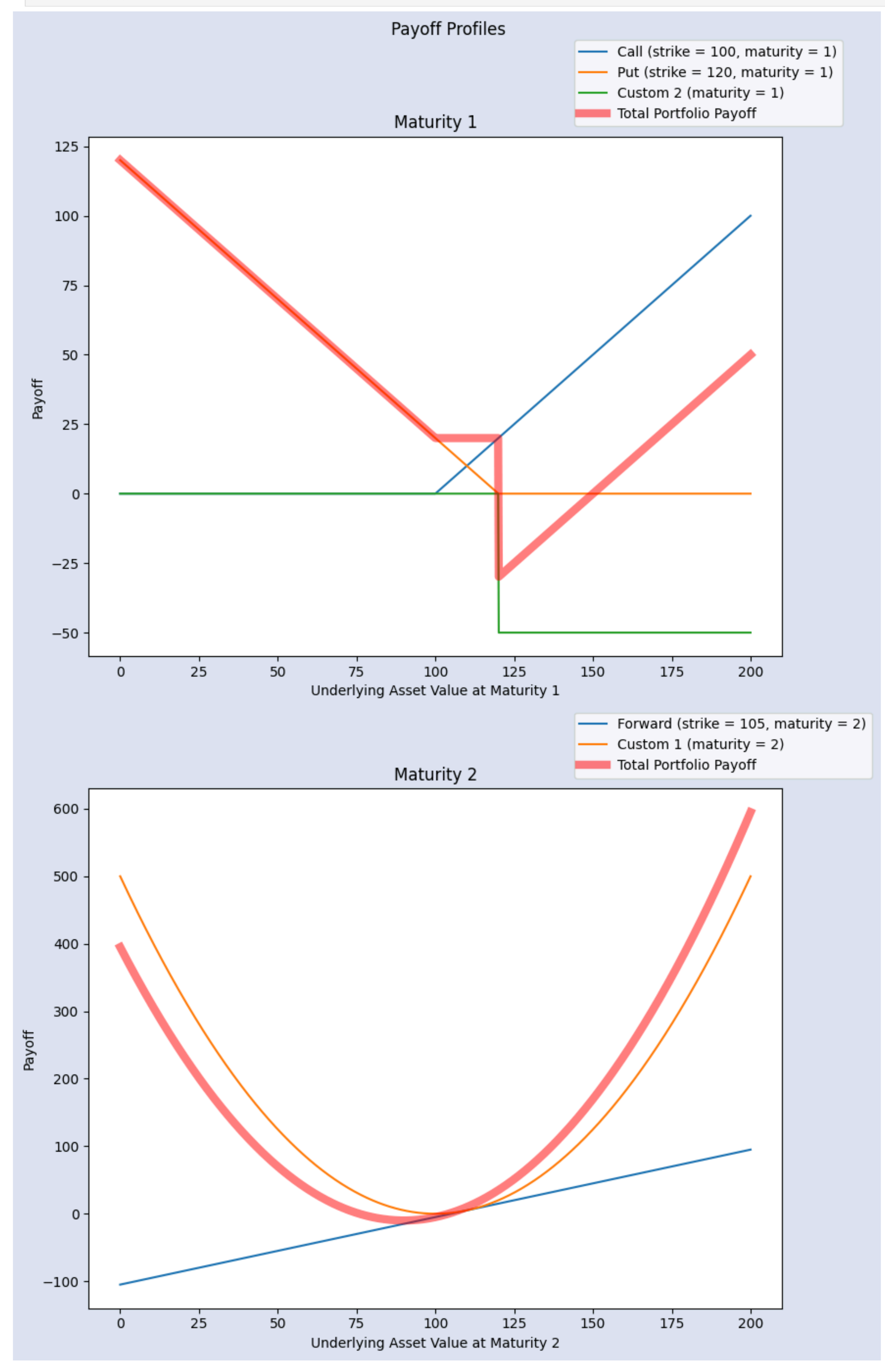
```
[Call (strike = 100, maturity = 1),
 Put (strike = 120, maturity = 1),
 Forward (strike = 105, maturity = 2)]
```

We can define a custom derivative payoff (having the underlying asset price as input).

```
custom_strike1 = 100
custom_payoff1 = lambda S: (S - custom_strike1)**(2)/20
custom1 = Derivative(custom_payoff1, 2, name='1')

custom_cutoff2 = 120
custom_payoff2 = lambda S: 50*(S> custom_cutoff2)
custom2 = Derivative(custom_payoff2, 1, name='2')

portfolio + [custom1]
portfolio - [custom2]

portfolio.derivatives
```

```
[Call (strike = 100, maturity = 1),
 Put (strike = 120, maturity = 1),
 Forward (strike = 105, maturity = 2),
 Custom 1 (maturity = 2),
 Custom 2 (maturity = 1)]
```

```
portfolio.payoff()
```



One can change the range in the plots either by directly changing the entries in the `low` and `high` attributes or by using the `range` method.

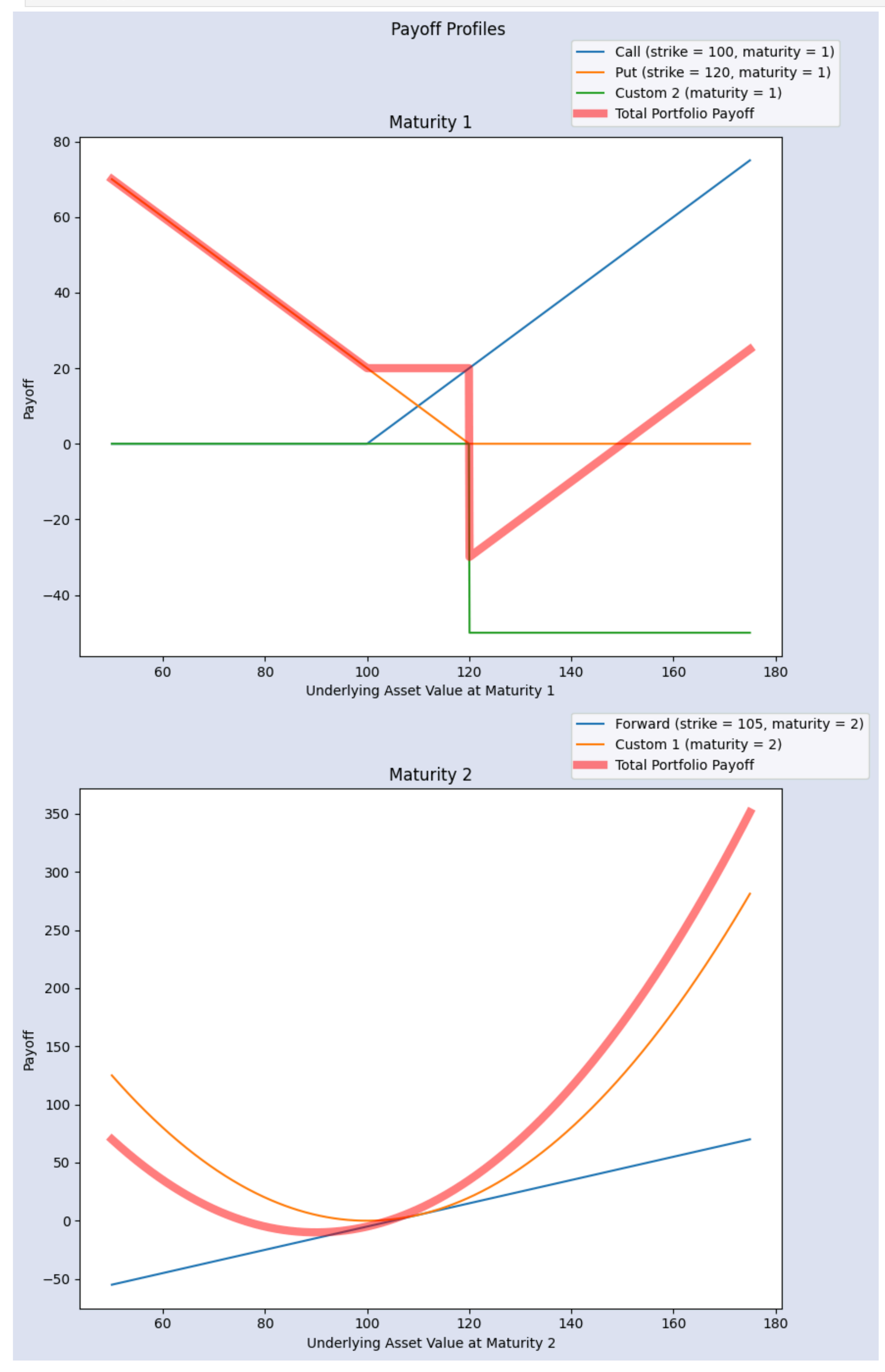
```
print(Portfolio.__doc__)
```

```
    Represents a portfolio of derivatives.

    The Portfolio class allows users to manage and analyze multiple derivative
    payoffs with different maturities.

    Attributes:
        derivatives (list): List of Derivative objects held in the portfolio.

        position (list): List of long/short positions.

        maturities (list): Unique maturities of the derivatives in the
        portfolio.

        low (dict): Lower bounds of the asset price range for each maturity
        for payoff plotting.

        high (dict): Upper bounds of the asset price range for each maturity
        for payoff plotting.
```

```
print(Portfolio.range.__doc__)
```

```
        Sets the price range (low and high) for plotting the payoff of
        derivatives at a specific maturity or all maturities.

        Args:
            low (float): The lower bound of the price range.
            high (float): The upper bound of the price range.
            maturity (float, optional): The maturity for which to set the
            range. If None, sets range for all maturities.
```

```
portfolio.range(50, 175)
portfolio.payoff()
```



Pricing

Given the current asset price, volatility and dividend yield q, an assumed constant risk-free interest rate and time to maturity, we can use the Black Scholes formulae to price the European calls and puts considered to have a better understanding of the payoff profiles. We can also use the interest rate to price the forward contracts.

For custom derivatives one can use numerical pricing procedures, some of which I have implemented in another project: (https://github.com/lr1021/Option_Pricing_Numerical_Methods).

```
# Example Values
# Time to maturity
T = 0.3333
# Current asset price and strike price/delivery price
S = 100
X = 100

# Risk free interest rate and asset volatility (we assume these are constant up
# to maturity).
r = 0.08
q = r
sigma = 0.4

def price_call(S, X, T, r, q, sigma):
    d1 = (np.log(S/X) + (r - q + sigma**2/2)*T)/(sigma*np.sqrt(T))
    d2 = (np.log(S/X) + (r - q - sigma**2/2)*T)/(sigma*np.sqrt(T))
    c = S*np.exp(-q*T)*norm.cdf(d1) - X*np.exp(-r*T)*norm.cdf(d2)
    return c

def price_put(S, X, T, r, q, sigma):
    d1 = (np.log(S/X) + (r - q + sigma**2/2)*T)/(sigma*np.sqrt(T))
    d2 = (np.log(S/X) + (r - q - sigma**2/2)*T)/(sigma*np.sqrt(T))
    c = -S*np.exp(-q*T)*norm.cdf(-d1) + X*np.exp(-r*T)*norm.cdf(-d2)
    return c

def price_forward(S, X, T, r, q):
    return (S*np.exp((r-q)*T) - X)*np.exp(-r*T)


price_put(S, X, T, r, q, sigma)
```

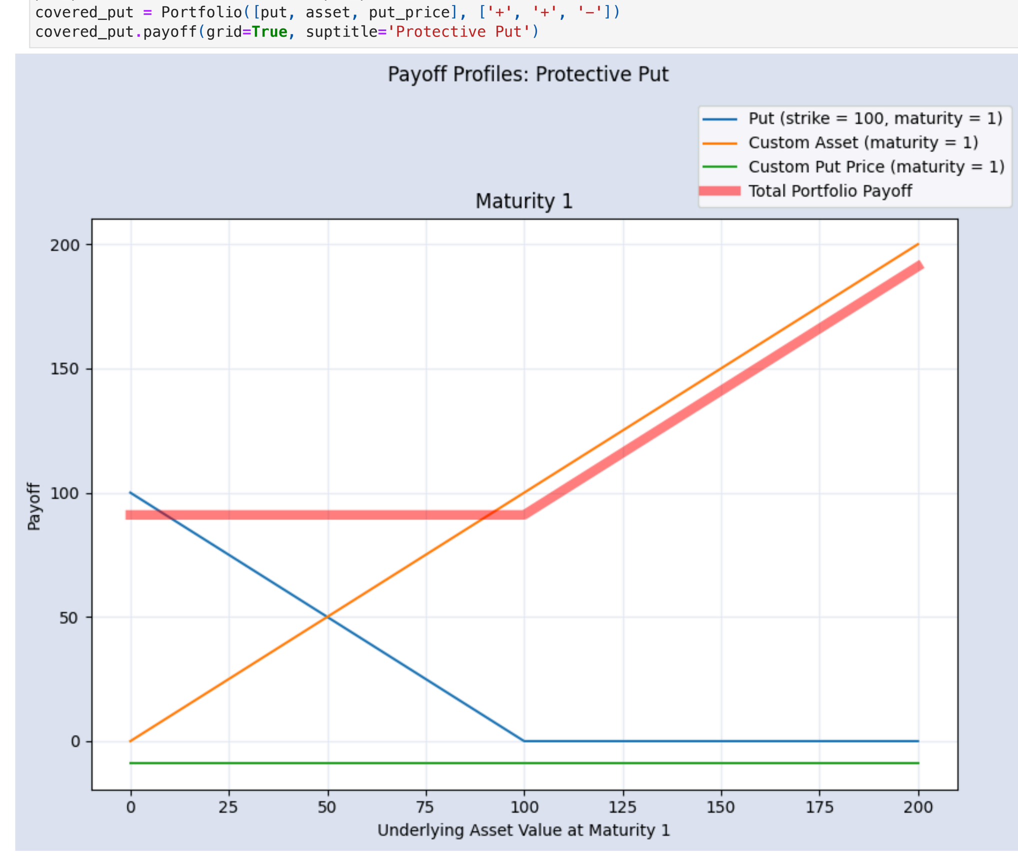
```
np.float64(8.950422591972334)
```

Spreads

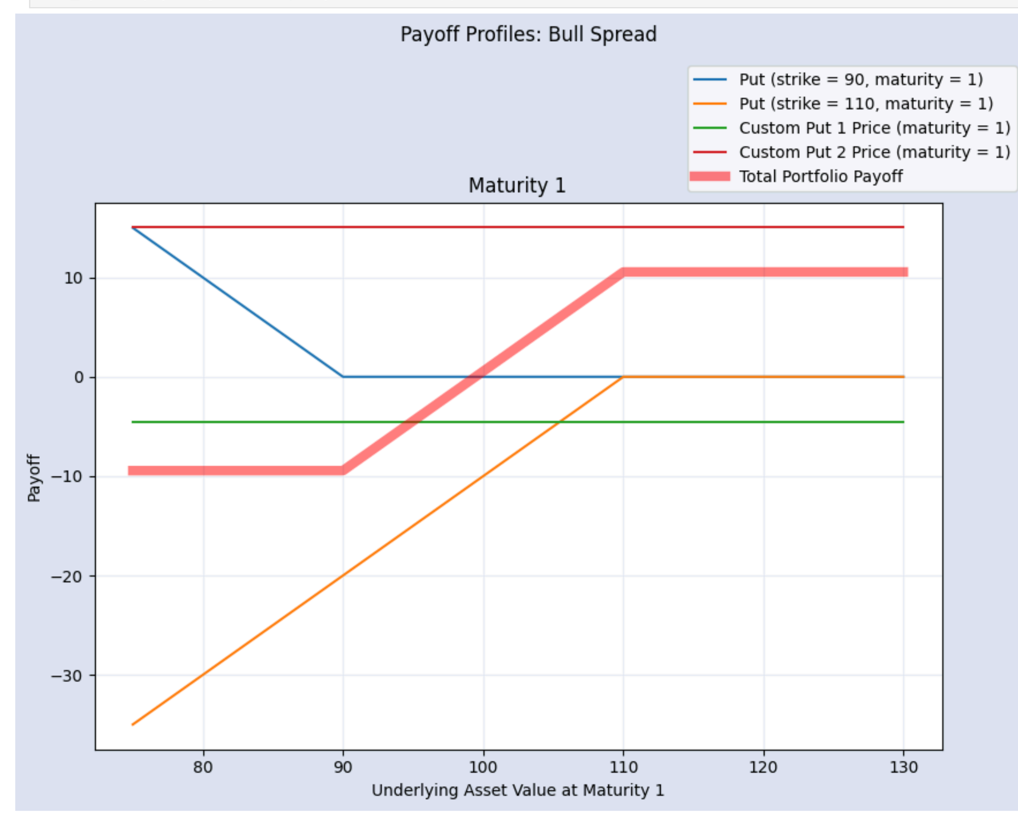
We can now use this infrastructure to easilty visualise different types of spreads.

```
asset = Derivative(lambda S: S, 1, name='Asset')

## Covered Call
X0 = 100
call = Call(X0, 1)
call_price_val = price_call(S, X, T, r, q, sigma)
call_price = Derivative(lambda S:call_price_val, 1, name='Call Price')
covered_call = Portfolio([call, asset, call_price], ['-', '+', '+'])
covered_call.payoff(grid=True, suptitle='Covered Call')
```

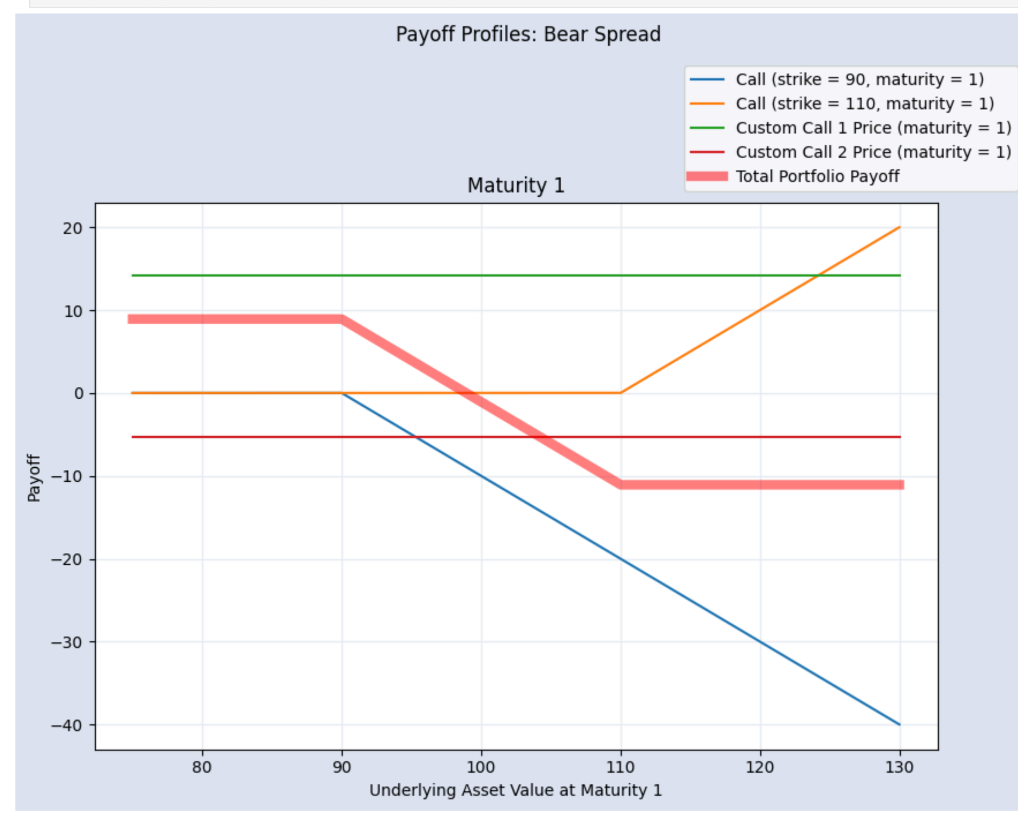

```
## Protective Put
X0 = 100
put = Put(X0, 1)
put_price_val = price_put(S, X, T, r, q, sigma)
put_price = Derivative(lambda S:put_price_val, 1, name='Put Price')
covered_put = Portfolio([put, asset, put_price], ['+', '+', '-'])
covered_put.payoff(grid=True, suptitle='Protective Put')
```




```
# Bull Spread (with Puts)
X1 = 90
X2 = 110
put_price_val1 = price_put(S, X1, T, r, q, sigma)
put_price_val2 = price_put(S, X2, T, r, q, sigma)
put_price1 = Derivative(lambda S:put_price_val1, 1, name='Put 1 Price')
put_price2 = Derivative(lambda S:put_price_val2, 1, name='Put 2 Price')
put1 = Put(X1, 1)
put2 = Put(X2, 1)

bull_spread = Portfolio([put1, put2, put_price1, put_price2], ['+', '-', '-', '+'])
bull_spread.range(75, 130)
bull_spread.payoff(grid=True, suptitle='Bull Spread')
```




```
# Bear Spread (with Calls)
X1 = 90
X2 = 110
call_price_val1 = price_call(S, X1, T, r, q, sigma)
call_price_val2 = price_call(S, X2, T, r, q, sigma)
call_price1 = Derivative(lambda S:call_price_val1, 1, name='Call 1 Price')
call_price2 = Derivative(lambda S:call_price_val2, 1, name='Call 2 Price')
call1 = Call(X1, 1)
call2 = Call(X2, 1)

bear_spread = Portfolio([call1, call2, call_price1, call_price2], ['-', '+', '+', '-'])
bear_spread.range(75, 130)
bear_spread.payoff(grid=True, suptitle='Bear Spread')
```




```
# Butterfly Spread
X1 = 80
X3 = 120
X2 = (X1 + X3)/2
call_price_val1 = price_call(S, X1, T, r, q, sigma)
call_price_val2 = price_call(S, X2, T, r, q, sigma)
call_price_val3 = price_call(S, X3, T, r, q, sigma)
call_price1 = Derivative(lambda S:call_price_val1, 1,
                         name='Call 1 Price', alpha=0.5)
call_price2 = Derivative(lambda S:call_price_val2, 1,
                         name='Call 2 Price', alpha=0.5)
call_price3 = Derivative(lambda S:call_price_val3, 1,
                         name='Call 3 Price', alpha=0.5)
call1 = Call(X1, 1)
call2 = Call(X2, 1)
call3 = Call(X3, 1)

butterfly_spread = Portfolio(
    [call1, call2, call2, call3, call_price1, call_price2, call_price2, call_price3],
    ['+', '-', '-', '+', '-', '+', '+', '-'])
butterfly_spread.range(70, 130)
butterfly_spread.payoff(grid=True, suptitle='Butterfly Spread')
```