- LYU JIAMING

Payoff Function 1

$$V_T(S_T) = \sqrt{S_T}$$

$$\frac{\partial^2 V_T}{\partial S_T^2} = -\frac{1}{4} \times S_T^{-1.5}$$

$$V_0 = \sqrt{F_0(T)} - \int_0^{F_0(T)} Put(K,T)\frac{K^{-1.5}}{4}dK - \int_{F_0(T)}^{\infty} Call(K,T)\frac{K^{-1.5}}{4}dK$$

Payoff Function 2

$$V_T(S_T) = S_T^3$$

$$\frac{\partial^2 V_T}{\partial S_T^2} = 6S_T$$

$$V_0 = (F_0(T))^3 + 6\int_0^{F_0(T)} Put(K,T)KdK + 6\int_{F_0(T)}^{\infty} Call(K,T)KdK$$

In [1]:
```
import numpy as np
import scipy.integrate as integrate
from enum import Enum
import math
import pandas as pd
```

In [2]:
```
def ivol_helper(K):
    return 0.510 - 0.591*K + 0.376*K**2 - 0.105*K**3 + 0.011*K**4
```

In [3]:
```
def ivol_HW6(K):
    if K>=3:
        return ivol_helper(3)
    else:
        return ivol_helper(K)
```

In [4]:
```
class PayoffType(str, Enum):
    Call = 'Call'
    Put = 'Put'
```

In [5]:
```
def cnorm(x):
    return (1.0 + math.erf(x / math.sqrt(2.0))) / 2.0
```

In [6]:
```
def Black(f, r, vol, T, strike, payoffType):
    stdev = vol * math.sqrt(T)
    d1 = math.log(f / strike) / stdev + stdev / 2
    d2 = d1 - stdev
    if payoffType == PayoffType.Call:
        return math.exp(-r * T) * (f * cnorm(d1) - cnorm(d2) * strike)
    elif payoffType == PayoffType.Put:
        return math.exp(-r * T) * (strike * cnorm(-d2) - cnorm(-d1) * f)
```

In [7]:
```
def black_with_smile(f,r,k,T,df,callorput):
    vol = ivol_HW6(k)
    return Black(f, r, vol, T, k, callorput) * df
```

In [8]:
```
def numerical_integration_HW6Q1(S0, r, q, T, SD):
    DF = np.exp(-r*T)
    DivF = np.exp(-q*T)
    f = S0*DivF/DF
    vol_for_range = ivol_HW6(f)
    maxS = f * np.exp(vol_for_range * SD * np.sqrt(T))
    forward_part = np.sqrt(f) * DF
    integrand_put = lambda y: y**(-1.5)/4 * black_with_smile(f, r, y, T, DF, PayoffType.Put)
    put_part, error = integrate.quad(integrand_put, 0.0001, f)
    integrand_call = lambda x: x**(-1.5)/4 * black_with_smile(f, r, x, T, DF, PayoffType.Call)
    call_part, error = integrate.quad(integrand_call, f, maxS)
    return forward_part - put_part - call_part
```

In [9]:
```
def numerical_integration_HW6Q2(S0, r, q, T, SD):
    DF = np.exp(-r*T)
    DivF = np.exp(-q*T)
    f = S0*DivF/DF
    vol_for_range = ivol_HW6(f)
    maxS = f * np.exp(vol_for_range * SD * np.sqrt(T))
    forward_part=f*f*f*DF
    integrand_put = lambda y: 6 * y * black_with_smile(f, r, y, T, DF, PayoffType.Put)
    put_part, error = integrate.quad(integrand_put, 0, f)
    integrand_call = lambda x: 6 * x * black_with_smile(f, r, x, T, DF, PayoffType.Call)
    call_part, error = integrate.quad(integrand_call, f, maxS)
    return forward_part + put_part + call_part
```

In [10]:
```
q=0.0;r=0.0;T=4;S0=1
SDs = np.linspace(1, 6, 6)
Q1numIntResults = [numerical_integration_HW6Q1(S0, r, q, T, sd) for sd in SDs]
Q2numIntResults = [numerical_integration_HW6Q2(S0, r, q, T, sd) for sd in SDs]
```

In [11]:
```
Combined_Results = pd.DataFrame(list(zip(Q1numIntResults,Q2numIntResults)),\
                   columns = ['PayoffFunction1','PayoffFunction2'], index = [1,2,3,4,5,6])
Combined_Results
```

Out[11]:

	PayoffFunction1	PayoffFunction2
1	0.974453	1.456347
2	0.973792	1.515702
3	0.973763	1.522669
4	0.973762	1.523054
5	0.973762	1.523059
6	0.973762	1.523059