```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import norm
from scipy.optimize import brentq, fsolve, minimize_scalar
```

```
# Parameters
F = 70
K = 85
r = 0.05
tau = 30/365

a, b, c = 45.0, -1.0, 0.01  # equivalent to sigma(f): f -> 0.01*(f-50)**2 + 20
```

```
# Bachelier (1900)

def BC_formula(vol, option_type):
   '''
   Bachelier formula
   return call/put option price
   '''
   m = (F - K) / (vol * tau**0.5)
   if option_type == 'call':
     return np.exp(-r * tau) * ((F - K) * norm.cdf(m) + vol * tau**0.5 * norm.pdf(m))
   elif option_type == 'put':
     return np.exp(-r * tau) * ((K - F) * (1 - norm.cdf(m)) + vol * tau**0.5 * norm.pdf(m))


def BC_SimulateF(vol, paths = 100000, steps = 2):
   '''
   Monte Carlo simulation
   return underlying futures price and time vector
   '''
   # initialize vectors
   arr_F = np.zeros((paths, steps))
   arr_F[:,0] = F

   # generate Brownian motion
   arr_w = np.random.standard_normal(size = (paths, steps-1))

   # compute the corresponding price
   t_vec, dt = np.linspace(0, tau, steps, retstep=True)
   for t in range(0, steps-1):
     arr_F[:,t+1] = arr_F[:,t] + vol * dt**0.5 * arr_w[:,t]

   return arr_F, t_vec
```

```
# verify Bachelier implemenation


vol_bachelier = 30
# Bachelier formula
bachelier_call, bachelier_put = BC_formula(vol_bachelier, 'call'), BC_formula(vol_bachelier, 'put')
print(f'Bachelier formula:')
print(f'call is {bachelier_call:.4f}')
print(f'put is {bachelier_put:.4f}')


# Monte Carlo
paths = 10000000
arr_F, _ = BC_SimulateF(vol = vol_bachelier, paths = paths)

arr_V = np.exp(-r*tau) * np.maximum(arr_F[:,-1] - K, 0)
mean_call, ste_call = arr_V.mean(), arr_V.std() / paths**0.5
arr_V = np.exp(-r*tau) * np.maximum(K - arr_F[:,-1], 0)
mean_put, ste_put = arr_V.mean(), arr_V.std() / paths**0.5

print(f'\nMonte Carlo simulation:')
print(f'call is {mean_call:.4f} with standard error {ste_call:.6f}')
print(f'put is {mean_put:.4f} with standard error {ste_put:.6f}')
```

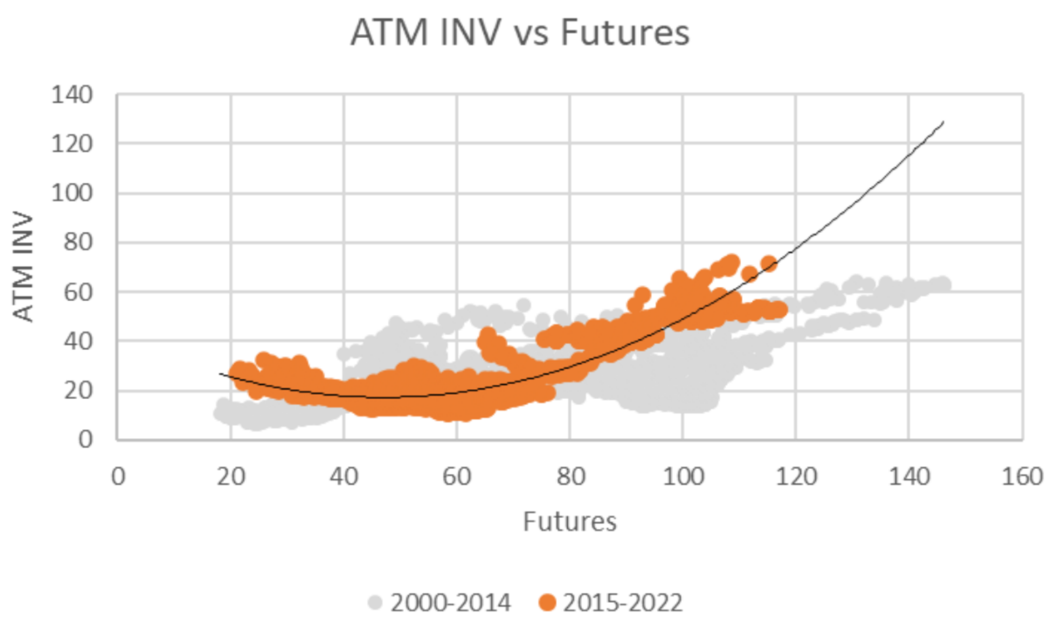
```
Bachelier formula:
call is 0.1406
put is 15.0791

Monte Carlo simulation:
call is 0.1410 with standard error 0.000294
put is 15.0785 with standard error 0.002613
```

Quadratic Normal Model (Bouchouev, 2023)

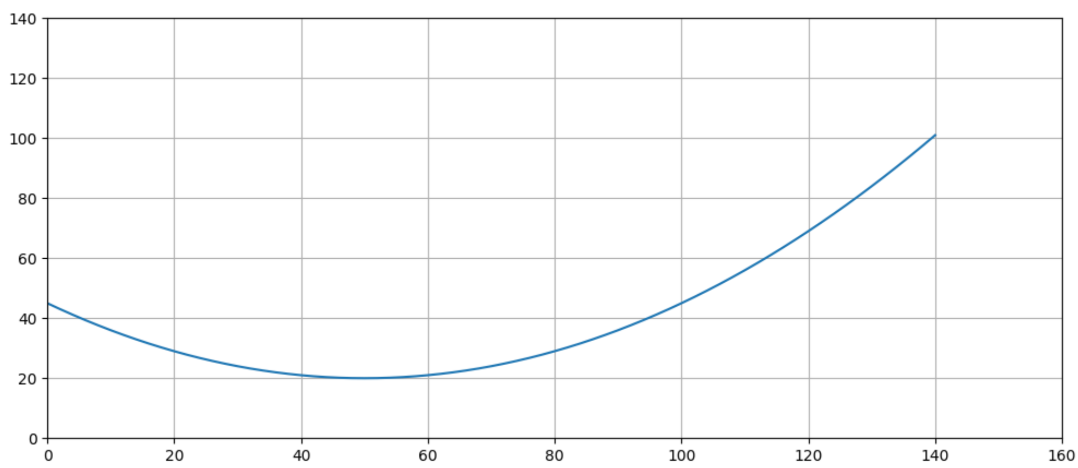
$$dF_t = \sigma(F_t)dW_t = (a + bF_t + cF_t^2)dW_t$$

$$C(F, K, a, b, c, r, \tau) = C_{BC}(F, K, \sigma_a = a, r, \tau) + e^{-r\tau}U$$
$$P(F, K, a, b, c, r, \tau) = P_{BC}(F, K, \sigma_a = a, r, \tau) + e^{-r\tau}U$$

where
C_{BC} : Bachelier call formula
P_{BC} : Bachelier put formula
$$U = \sqrt{\tau} n\left(\frac{F-K}{a\sqrt{\tau}}\right)\left[\frac{b}{2}(F+K) + \frac{c}{3}\left(F^2 + FK + K^2 + \frac{a^2\tau}{2}\right)\right]$$




```
# local vol given the above a, b, c

sigma_qnm = lambda f : a + b*f + c*f**2
arr_f = np.arange(0,141,2)

plt.figure(figsize=(12,5))
plt.plot(arr_f, sigma_qnm(arr_f))

plt.xlim(0,160)
plt.ylim(0,140)
plt.grid()

plt.show()
```




```
def QNM_formula(option_type):
    '''
    The method of linearization
    return call/put option price, Bachelier component, skew correction component
    '''
    m = (F - K)/(a * tau**0.5)
    BC = BC_formula(a, option_type)
    U = tau**0.5 * norm.pdf(m) * (b*(F + K)/2 + c*(F**2 + F*K + K**2 + 0.5*a**2*tau)/3)

    return BC + U*np.exp(-r * tau), BC, U*np.exp(-r * tau)

qnm_call, qnm_call_BC, qnm_call_skew = QNM_formula('call')
qmm_put, qnm_put_BC, qnm_put_skew = QNM_formula('put')
```

```
print(f'Quadratic Normal Model formula:')
print(f'QNM call is {qnm_call:.4f}, Bachelier component is {qnm_call_BC:.4f}, skew correction component is {qnm_call_skew:.4f}')
print(f'QNM put is {qmm_put:.4f}, Bachelier component is {qnm_put_BC:.4f}, skew correction component is {qnm_put_skew:.4f}')
```

```
Quadratic Normal Model formula:
QNM call is -0.2057, Bachelier component is 0.7777, skew correction component is -0.9834
QNM put is 14.7328, Bachelier component is 15.7162, skew correction component is -0.9834
```

按兩下 (或按 Enter 鍵) 即可編輯