SwaptionPricing V2

April 1, 2025

0.0.1 Swaptions under Hull & White model

[46]:
```
import numpy as np
import pandas as pd
from datetime import datetime
import matplotlib.pyplot as plt
import random
from scipy.interpolate import CubicSpline
import scipy.optimize as opt
from scipy.stats import norm
from scipy.interpolate import interp1d
from scipy.stats import norm
from dateutil.relativedelta import relativedelta
import re
```

0.0.2 Building discount Curves

[47]:
```
df = pd.read_excel('market_ata.xlsx', sheet_name = "instruments" )
df.T
```

[47]:
```
                 0        1        2        3        4        5        6  \
Maturity  0.003968     0.25      0.5      1.0      2.0      3.0      4.0
Type     Overnight  Euribor  Euribor  Euribor     Swap     Swap     Swap
Rate        0.0143   0.0169   0.0184  0.01908  0.02091  0.02184  0.02227

                7        8        9  …       23       24       25       26  \
Maturity      5.0      6.0      7.0  …     21.0     22.0     23.0     24.0
Type         Swap     Swap     Swap  …     Swap     Swap     Swap     Swap
Rate      0.02261  0.02295  0.02332  …  0.02582  0.02584  0.02585  0.02585

               27       28       29       30       31       32
Maturity     25.0     26.0     27.0     28.0     29.0     30.0
Type         Swap     Swap     Swap     Swap     Swap     Swap
Rate      0.02585  0.02584  0.02582  0.02581  0.02579  0.02578

[3 rows x 33 columns]
```

[48]:
```python
def extractZCCurve(maturities, rates):
    ZC = []
    for i in range(len(maturities)):
        if maturities[i] <=1:
            ZCvalue = 1 / (1 +  maturities[i] *  rates[i])
            ZC.append(ZCvalue)
        else:
            ZCvalue = (1.0 - rates[i] * sum(ZC[3:i])) / (1 + rates[i])
            ZC.append(ZCvalue)

    return np.array(ZC)

df['ZC_PRICE'] = extractZCCurve(df['Maturity'].values, df.Rate.values)
df.T
```

[48]:
```
                 0         1         2         3        4         5   \
Maturity   0.003968      0.25       0.5       1.0      2.0       3.0
Type      Overnight   Euribor   Euribor   Euribor     Swap      Swap
Rate         0.0143    0.0169    0.0184   0.01908  0.02091   0.02184
ZC_PRICE   0.999943  0.995793  0.990884  0.981277  0.95942  0.937148

                6         7         8         9   …        23        24  \
Maturity      4.0       5.0       6.0       7.0   …      21.0      22.0
Type         Swap      Swap      Swap      Swap   …      Swap      Swap
Rate      0.02227   0.02261   0.02295   0.02332   …   0.02582   0.02584
ZC_PRICE 0.915522  0.894018  0.872403  0.850512   …  0.581899  0.566926

                25        26        27       28        29        30        31  \
Maturity      23.0      24.0      25.0     26.0      27.0      28.0      29.0
Type          Swap      Swap      Swap     Swap      Swap      Swap      Swap
Rate       0.02585   0.02585   0.02585  0.02584   0.02582   0.02581   0.02579
ZC_PRICE  0.552477  0.538555  0.524985  0.51194  0.499422  0.487046  0.475188

                32
Maturity      30.0
Type          Swap
Rate       0.02578
ZC_PRICE  0.463444

[4 rows x 33 columns]
```

0.1 Hull White Calibration with black formula

0.1.1 Fit initial term structure

Le terme $\theta(t)$ est choisi pour que le modèle reproduise la courbe des taux zéro-coupon observée. Il est donné par :

$$\theta(t) = \frac{df(0,t)}{dt} + \alpha f(0,t) + \frac{\sigma^2}{2\alpha}\left(1 - e^{-2\alpha t}\right)$$

où : - $f(0,t) = -\frac{d}{dt}\ln P(0,t)$ est le taux instantané forward, - $P(0,t)$ est le prix d'une obligation zéro-coupon à l'instant 0 avec maturité t.

[49]:
```
def calibrate_theta_hull_white(alpha, sigma, zc_values, maturities):
    log_zc = np.log(zc_values)
    spline = CubicSpline(maturities, log_zc, bc_type='natural')
    fwd_rates = -spline.derivative()(maturities)
    dfdt = spline.derivative(nu=2)(maturities)
    theta = dfdt + alpha * fwd_rates + (sigma**2 / (2 * alpha)) * (1 - np.
 ↪exp(-2 * alpha * maturities))
    return theta
```

0.1.2 Price zero coupon with HullWhite

La formule fermée pour le prix d'une obligation zéro-coupon $P(t,T)$, qui verse 1 à l'instant T, est :

$$P(t,T) = A(t,T)e^{-B(t,T)r(t)}$$

avec :

$$B(t,T) = \frac{1 - e^{-\alpha(T-t)}}{\alpha}$$

$$A(t,T) = \exp\left((B - (T-t))\frac{\alpha^2\theta(T) - \frac{1}{2}\sigma^2}{\alpha^2} - \frac{\sigma^2 B^2}{4\alpha}\right)$$

[50]:
```
def price_zero_coupon_hw(alpha, sigma, T, spotRate, maturities, zc_values):
    log_zc_spline = CubicSpline(maturities, np.log(zc_values),␣
 ↪bc_type='natural')
    P0_T = np.exp(log_zc_spline(T))
    d_logP0S_dS = log_zc_spline.derivative()(0)
    B_0T = (1 - np.exp(-alpha * T)) / alpha
    exp_term = -B_0T * d_logP0S_dS - (sigma**2 / (4 * alpha**3)) * ((np.
 ↪exp(-alpha * T) - 1)**2 * (np.exp(0) - 1))
    A_0T = P0_T * np.exp(exp_term)
    return A_0T * np.exp(-B_0T * spotRate)
```

0.1.3 Price Swaption payer with Black

Dans le cadre du modèle de Black, la valeur d'un **swaption payer** est donnée par :

$$V_{\text{swaption}(t,T_0,T_n)} = N \sum_{i=T_0}^{T_n} P(t,T_i) \left(SwapRate(t,T_0,T_n) N(d_1) - K N(d_2) \right)$$

$$d_1 = \frac{\ln\left(\frac{SwapRate(t,T_0,T_n)}{K}\right) + \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

$$SwapRate(t,T_0,T_n) = \frac{P(t,T_0) - P(t,T_n)}{\sum_{i=T_0}^{T_n} P(t,T_i)}$$

[51]:
```
def swaption_price_black(kappa, sigma,start_date, end_date, strike, spotRate,␣
↪theta, maturities, zc_values):
    zc_po = price_zero_coupon_hw(kappa, sigma, start_date, spotRate,␣
↪maturities, zc_values)
    zc_pn = price_zero_coupon_hw(kappa, sigma, end_date, spotRate, maturities,␣
↪zc_values)
    payments_dates = [t for t in range(start_date+1, end_date+1)]

    annuity = np.sum([price_zero_coupon_hw(kappa, sigma, t, spotRate,␣
↪maturities, zc_values) for t in payments_dates])
    swap_rate = (zc_po - zc_pn) / annuity

    tau = end_date - start_date
    d1 = (np.log(swap_rate / strike) + 0.5 * sigma**2 * tau) / (sigma * np.
↪sqrt(tau))
    d2 = d1 - sigma * np.sqrt(tau)
    return annuity * (swap_rate * norm.cdf(d1) - strike * norm.cdf(d2))
```

0.1.4 Get Swpation implied volatility from market

[41]:
```
df_swaption_vol = pd.read_excel('market_ata.xlsx', sheet_name = "swaption_vol" )
df_swaption_vol
```

[41]:

	Unnamed: 0	1Y	2Y	3Y	4Y	5Y	6Y	7Y	8Y	\
0	1Y	0.8356	0.6660	0.5910	0.5300	0.4882	0.4402	0.4042	0.3761	
1	2Y	0.6795	0.5370	0.4789	0.4370	0.4080	0.3786	0.3548	0.3353	
2	3Y	0.5212	0.4420	0.4085	0.3801	0.3572	0.3372	0.3203	0.3062	
3	4Y	0.4326	0.3803	0.3561	0.3359	0.3186	0.3052	0.2938	0.2851	
4	5Y	0.3737	0.3345	0.3175	0.3029	0.2903	0.2811	0.2741	0.2687	
5	7Y	0.2964	0.2744	0.2644	0.2560	0.2499	0.2463	0.2437	0.2424	
6	10Y	0.2376	0.2286	0.2258	0.2239	0.2227	0.2239	0.2252	0.2269	

9Y 10Y

```
0   0.3534   0.3353
1   0.3192   0.3065
2   0.2948   0.2853
3   0.2778   0.2716
4   0.2646   0.2609
5   0.2423   0.2417
6   0.2290   0.2301
```

0.1.5 Make Calibration

[42]:
```
def error_objective(params, *args):
    kappa, sigma = params
    market_vol,maturities, zc_price, strike, spotRate  = args
    today = datetime.today()  # Remplace "today" par une autre date si besoin

    error = 0
    for i in range(len(market_vol)):
        for j in range(1,len(market_vol[i])):
            volatility = market_vol[i][j]
            option_maturity_years = int(re.match(r"(\d+)Y", market_vol[i][0]).
 ↪group(1))
            start_date =  option_maturity_years  #today +␣
 ↪relativedelta(years=option_maturity_years)
            end_date = option_maturity_years + j # start_date + ␣
 ↪relativedelta(years=j)


            theta_value = calibrate_theta_hull_white(kappa,␣
 ↪sigma,zc_price,maturities)
            #theta_value =  [0.03 for i in range(len(maturities))]
            theta_t = CubicSpline(maturities, theta_value, extrapolate=True)

            P_model = swaption_price_black(kappa, sigma,start_date, end_date,␣
 ↪strike, spotRate, theta_t,maturities, zc_price)
            P_market = swaption_price_black(kappa, volatility, start_date,␣
 ↪end_date, strike, spotRate, theta_t,maturities, zc_price)
            error += (P_model - P_market) ** 2
    return error
```

0.1.6 Application

[43]:
```
maturities = df['Maturity'].values
zc_price = df['ZC_PRICE'].values
rates = df['Rate'].values
zc_curve_t = interp1d(maturities,zc_price , kind='linear')
rate_curve_t = interp1d(maturities, rates, kind='linear')
```

```
strike = 0.02
spotRate = rates[0]
initial_guess = [0.01, 0.01]
bounds = [(0.0001, 0.9), (0.00001, 1.9)]
market_vol = df_swaption_vol.values
args = (market_vol,maturities, zc_price,strike, spotRate)
result = opt.minimize(error_objective, initial_guess, bounds=bounds, args=args,␣
↪method='Nelder-Mead')
kappa, sigma= result.x
print(f'kappa = {kappa:.6f}, sigma = {sigma:.6f}')
```

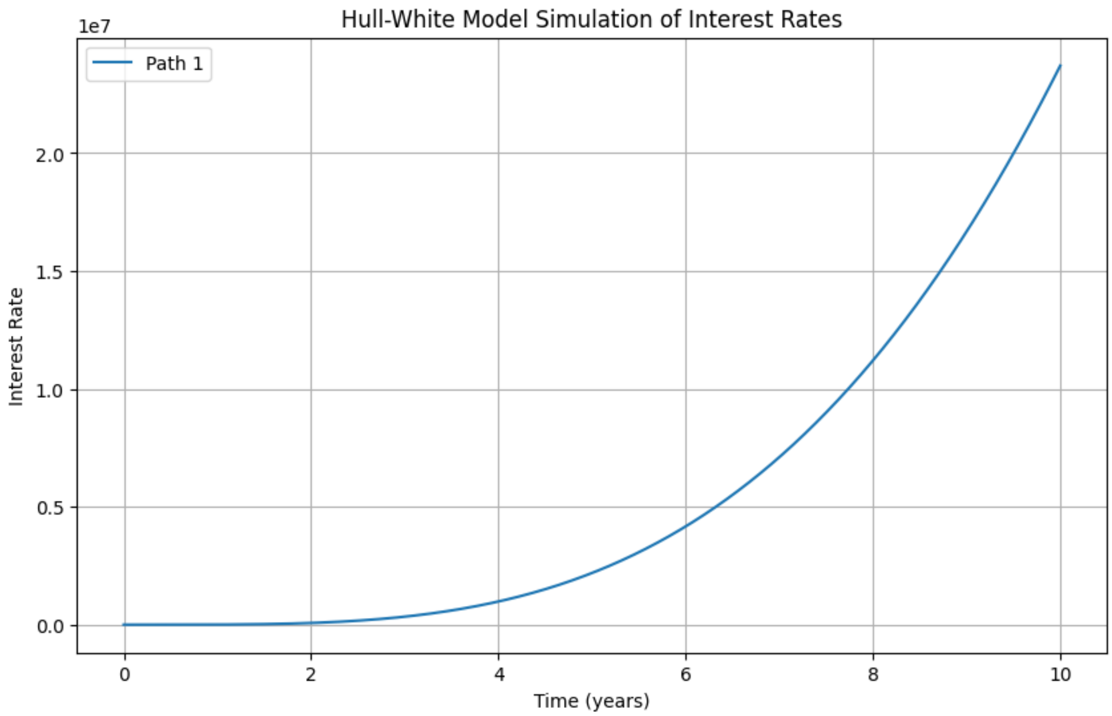
```
kappa = 0.900000, sigma = 0.298526
```

[44]:
```
theta_value = calibrate_theta_hull_white(kappa, sigma,zc_price,maturities)
theta_t = CubicSpline(maturities, theta_value, extrapolate=True)
theta_value
```

[44]:
```
array([0.01486623, 0.01555468, 0.0541141 , 0.05214227, 0.06974985,
       0.07056549, 0.07006264, 0.0705501 , 0.07074769, 0.07257608,
       0.07202585, 0.0732909 , 0.07322699, 0.07464724, 0.07433521,
       0.07356128, 0.07427334, 0.07430105, 0.07400588, 0.07462597,
       0.0739172 , 0.07483914, 0.07376651, 0.07379906, 0.07294958,
       0.07306091, 0.07225455, 0.07279633, 0.07262262, 0.07098977,
       0.07303147, 0.07092901, 0.07215981])
```

[45]:
```
import numpy as np
import matplotlib.pyplot as plt

theta = 0.03  # Long-term mean
# Simulation parameters
T = 10          # Time horizon (years)
dt = 1/252      # Time step (daily steps, assuming 252 trading days per year)
N = int(T / dt)   # Number of time steps
M = 1           # Number of simulated paths


# Generate Brownian motion
np.random.seed(42)  # For reproducibility
dW = np.sqrt(dt) * np.random.randn(M, N)

# Hull-White model simulation
r = np.zeros((M, N))
r[:, 0] = spotRate

for i in range(1, N):
    dr = kappa * (theta_t(i) - r[:, i-1]) * dt + sigma * dW[:, i-1]
    r[:, i] = r[:, i-1] + dr
```

```python
# Time axis for plotting
time = np.linspace(0, T, N)

# Plot the results
plt.figure(figsize=(10, 6))
for i in range(M):
    plt.plot(time, r[i], label=f'Path {i+1}')

plt.title("Hull-White Model Simulation of Interest Rates")
plt.xlabel("Time (years)")
plt.ylabel("Interest Rate")
plt.legend()
plt.grid(True)
plt.show()
```



0.2 Evaluation Swaption payer

[ ]:

```python
```