MF728_hw3

March 28, 2024

0.1 MF 728: Fixed Income

0.1.1 Problem set # 3

1. Swaption Pricing and Risk Management under the SABR Model Consider the following table of normal swaption volatilities and corresponding par swap rates:

Expiry	Tenor	F0	ATM-50	ATM-25	ATM-5	ATM+5	ATM+25	ATM+50
1Y	5Y	117.45	58.31	51.51	49.28	48.74	41.46	37.33
2Y	5Y	120.60	51.72	46.87	43.09	42.63	38.23	34.55
3Y	5Y	133.03	46.29	44.48	43.61	39.36	35.95	32.55
4Y	5Y	152.05	45.72	41.80	38.92	38.19	34.41	31.15
5Y	5Y	171.85	44.92	40.61	37.69	36.94	33.36	30.21

NOTE: All numbers in the table are reported in bps.

(a) Calculate the constant instantaneous forward rate for each swap that will lead to the par swap rates listed. You may use a different instantaneous forward rate for each swap and are not required to go through an entire bootstrapping exercise.

[16]:
```
from math import log

# Par swap rates from the table (F0), in basis points, converted to percentage
par_swap_rates = [117.45, 120.60, 133.03, 152.05, 171.85]
par_swap_rates_percent = [rate / 10000 for rate in par_swap_rates]  # Convert␣
 ↪basis points to decimal

# Tenor for all swaps
T = 5

# Calculating the constant instantaneous forward rate f for each swap
instantaneous_forward_rates = []
for S in par_swap_rates_percent:
    f = -(1 / T) * log(1 - S*T)
    instantaneous_forward_rates.append(f)

instantaneous_forward_rates
```

```
[16]: [0.012103987959157759,
       0.012438920718623556,
       0.013766076937729256,
       0.015814052796707372,
       0.017968530800338 15]
```

(b) Using the rates obtained above, calculate the current annuity value for each swap in the above table.

```
[22]: from math import exp

      annuity_values = []
      for f in instantaneous_forward_rates:
          annuity_value = sum([exp(-f * t) for t in range(1, T+1)])
          annuity_values.append(annuity_value)

      annuity_values
```

```
[22]: [4.82240347685909,
       4.817599966733417,
       4.7986238491522055,
       4.769520733811614,
       4.739137539924281]
```

(c) Calculate a table of premiums for each swaption in the table using the Bachelier pricing formula and the annuities computed above.

```
[24]: import numpy as np
      import pandas as pd

      # Given par swap rates and normal volatilities in basis points
      F0_bps = np.array([117.45, 120.60, 133.03, 152.05, 171.85])
      normal_volatilities_bps = {
          'ATM-50': [58.31, 51.72, 46.29, 45.72, 44.92],
          'ATM-25': [51.51, 46.87, 44.48, 41.80, 40.61],
          'ATM-5': [49.28, 43.09, 43.61, 38.92, 37.69],
          'ATM+5': [48.74, 42.63, 39.36, 38.19, 36.94],
          'ATM+25': [41.46, 38.23, 35.95, 34.41, 33.36],
          'ATM+50': [37.33, 34.55, 32.55, 31.15, 30.21]
      }
      # Convert rates from bps to percentage
      F0 = F0_bps / 10000
      normal_volatilities = {k: np.array(v) / 10000 for k, v in␣
       ↪normal_volatilities_bps.items()}

      # Annuity values from previous calculation
      annuity_values = np.array(annuity_values)

      # Time to expiry in years
```

```
T_expiry = np.array([1, 2, 3, 4, 5])

# Calculate premiums using the Bachelier (normal) model formula
premiums = {}
for adjustment, volatilities in normal_volatilities.items():
    strike_adjustment = int(adjustment.replace('ATM', '')) / 10000  # Convert␣
 ↪adjustment to decimal
    K = F0 + strike_adjustment  # Adjusted strike rates
    d1 = (F0 - K) / (volatilities * np.sqrt(T_expiry[:, None]))  # d1 formula␣
 ↪adjustment, reshape T_expiry for broadcasting
    premiums[adjustment] = annuity_values[:, None] * volatilities * np.
 ↪sqrt(T_expiry[:, None]) * (d1 * norm.cdf(d1) + norm.pdf(d1))

# Convert the premiums dictionary to a DataFrame for display
premiums_df_columns = ['Expiry'] + list(normal_volatilities.keys())
premiums_df = pd.DataFrame(columns=premiums_df_columns)
premiums_df['Expiry'] = [f"{x}Y" for x in T_expiry]

for adjustment in normal_volatilities.keys():
    premiums_df[adjustment] = np.diagonal(premiums[adjustment])

premiums_df.set_index('Expiry', inplace=True)
premiums_df
```

[24]:
```
          ATM-50    ATM-25    ATM-5     ATM+5     ATM+25    ATM+50
Expiry
1Y      0.027163  0.017083  0.010735  0.008221  0.003356  0.000753
2Y      0.029264  0.019657  0.012956  0.010422  0.005461  0.001876
3Y      0.030237  0.021517  0.015691  0.011886  0.006870  0.002783
4Y      0.031861  0.022575  0.016034  0.013372  0.007988  0.003556
5Y      0.033144  0.023739  0.017147  0.014461  0.008964  0.004271
```

(d) For each option expiry, find the set of SABR parameters that best matches the quoted normal volatilities. Utilize the asymptotic approximation formula to calculate the normal volatility for a given set of SABR parameters and look for a solution that minimizes the distance between market and model volatilities.

[26]:
```
from scipy.stats import norm

def sabr_normal_volatility(alpha, beta, rho, nu, F, K, T):
    # Calculate SABR parameters
    F_avg = (F + K) / 2
    z = (nu / alpha) * ((F * K) ** ((1 - beta) / 2)) * np.log(F / K)
    x = np.log((np.sqrt(1 - 2 * rho * z + z ** 2) + z - rho) / (1 - rho))

    # Calculate gamma1 and gamma2
    gamma1 = beta / F_avg
    gamma2 = beta * (beta - 1) / F_avg ** 2
    # Calculate SABR normal volatility
    factor = alpha / ((F * K) ** ((1 - beta) / 2))
    term1 = ((2 * gamma2 - gamma1 ** 2) / 24) * (nu * x / alpha) ** 2
    term2 = (rho * gamma1 / 4) * (nu * x / alpha)
    term3 = (2 - 3 * rho ** 2) / 24
    normal_vol = factor * (z / x) * (1 + (term1 + term2 + term3) * T)
    return normal_vol

def objective_function(params, F, strikes, T, market_vols):
    alpha, beta, rho, nu = params
    model_vols = [sabr_normal_volatility(alpha, beta, rho, nu, F, K, T) for K␣
 ↪in strikes]
    return np.sum((np.array(model_vols) - np.array(market_vols)) ** 2)

# Initialize empty lists to store calibrated SABR parameters and model␣
 ↪volatilities
calibrated_params = []
model_volatilities = []

for i in range(len(T_expiry)):
    F = F0[i]
    strikes = F + np.array([-0.0050, -0.0025, -0.0005, 0.0005, 0.0025, 0.0050])
    market_vols = [normal_volatilities[k][i] for k in normal_volatilities.
 ↪keys()]

    # Define initial guess for SABR parameters
    init_params = [0.01, 0.5, -0.2, 0.2]  # Example initial guess, you may need␣
 ↪to adjust

    # Perform optimization to find calibrated SABR parameters
    result = minimize(objective_function, init_params, args=(F, strikes,␣
 ↪T_expiry[i], market_vols), method='Nelder-Mead')

    calibrated_params.append(result.x)
    model_vols = [sabr_normal_volatility(*result.x, F, K, T_expiry[i]) for K in␣
 ↪strikes]
    model_volatilities.append(model_vols)

# Print the calibrated SABR parameters and model volatilities for each expiry
for i in range(len(T_expiry)):
    print(f"Expiry: {T_expiry[i]}Y")
    print(f"Calibrated SABR Parameters: alpha={calibrated_params[i][0]:.4f},␣
 ↪beta={calibrated_params[i][1]:.4f}, rho={calibrated_params[i][2]:.4f},␣
 ↪nu={calibrated_params[i][3]:.4f}")
```

```
    print(f"Model Volatilities: {[f'{vol:.4f}' for vol in␣
↪model_volatilities[i]]}")
    print()
```

```
Expiry: 1Y
Calibrated SABR Parameters: alpha=0.0043, beta=0.9922, rho=-0.2859, nu=0.0001
Model Volatilities: ['0.0048', '0.0048', '0.0048', '0.0048', '0.0048', '0.0048']

Expiry: 2Y
Calibrated SABR Parameters: alpha=0.0047, beta=1.0431, rho=-0.4611, nu=0.0000
Model Volatilities: ['0.0042', '0.0043', '0.0043', '0.0043', '0.0043', '0.0043']

Expiry: 3Y
Calibrated SABR Parameters: alpha=0.0041, beta=1.0361, rho=-0.5138, nu=0.0000
Model Volatilities: ['0.0040', '0.0040', '0.0040', '0.0040', '0.0041', '0.0041']

Expiry: 4Y
Calibrated SABR Parameters: alpha=0.0030, beta=1.0011, rho=-0.3690, nu=0.0000
Model Volatilities: ['0.0038', '0.0038', '0.0038', '0.0038', '0.0038', '0.0038']

Expiry: 5Y
Calibrated SABR Parameters: alpha=0.0062, beta=1.1947, rho=-0.3977, nu=0.0000
Model Volatilities: ['0.0036', '0.0037', '0.0037', '0.0037', '0.0038', '0.0038']
```

(e) Comment on the relationship of the calibrated parameters as a function of expiry
Based on the calibrated SABR parameters for different expiries,below are the observations:

1. Alpha ( ): The alpha parameter represents the initial volatility level. It does not show a clear trend with increasing expiry. It starts at 0.0043 for the 1Y expiry, increases to 0.0047 for the 2Y expiry, then decreases for the 3Y and 4Y expiries, and finally increases again to 0.0062 for the 5Y expiry. This suggests that the initial volatility level does not have a strong dependence on the expiry.

2. Beta ( ): The beta parameter determines the relationship between the underlying asset price and volatility. A beta close to 1 indicates a nearly lognormal model, while a beta close to 0 indicates a normal model. In the calibrated parameters, beta is close to 1 for all expiries, ranging from 0.9922 to 1.1947. This suggests that the underlying asset price and volatility have a nearly lognormal relationship, and this relationship remains relatively consistent across different expiries.

3. Rho ( ): The rho parameter represents the correlation between the underlying asset price and volatility. A negative rho indicates a negative correlation, meaning that as the underlying price increases, the volatility tends to decrease. The calibrated rho values are all negative, ranging from -0.2859 to -0.5138. The absolute value of rho generally increases with expiry, indicating a stronger negative correlation for longer expiries.

4. Nu ( ): The nu parameter represents the volatility of volatility. In the calibrated parameters, nu is very close to zero for all expiries except for the 1Y expiry, where it is 0.0001. This

suggests that the volatility of volatility is relatively low and does not vary significantly with expiry.

(f) Using these calibrated SABR parameters, calculate the price and normal volatility of swaptions with strikes equal to ATM - 75 and ATM + 75.

[30]:
```python
# Function to calculate swaption price using Bachelier formula
def bachelier_swaption_price(F, K, T, normal_vol, annuity):
    d1 = (F - K) / (normal_vol * np.sqrt(T))
    price = annuity * normal_vol * np.sqrt(T) * (d1 * norm.cdf(d1) + norm.
 ↪pdf(d1))
    return price

# Calculate prices and normal volatilities for swaptions with strikes ATM - 75␣
 ↪and ATM + 75
prices_atm_minus_75 = []
prices_atm_plus_75 = []
normal_vols_atm_minus_75 = []
normal_vols_atm_plus_75 = []

for i in range(len(T_expiry)):
    F = F0[i]
    alpha, beta, rho, nu = calibrated_params[i]
    annuity = annuity_values[i]

    strike_atm_minus_75 = F - 0.0075
    strike_atm_plus_75 = F + 0.0075

    normal_vol_atm_minus_75 = sabr_normal_volatility(alpha, beta, rho, nu, F,␣
 ↪strike_atm_minus_75, T_expiry[i])
    normal_vol_atm_plus_75 = sabr_normal_volatility(alpha, beta, rho, nu, F,␣
 ↪strike_atm_plus_75, T_expiry[i])

    price_atm_minus_75 = bachelier_swaption_price(F, strike_atm_minus_75,␣
 ↪T_expiry[i], normal_vol_atm_minus_75, annuity)
    price_atm_plus_75 = bachelier_swaption_price(F, strike_atm_plus_75,␣
 ↪T_expiry[i], normal_vol_atm_plus_75, annuity)

    prices_atm_minus_75.append(price_atm_minus_75)
    prices_atm_plus_75.append(price_atm_plus_75)
    normal_vols_atm_minus_75.append(normal_vol_atm_minus_75)
    normal_vols_atm_plus_75.append(normal_vol_atm_plus_75)

# Create a DataFrame to store the results
results_df = pd.DataFrame({
    'Expiry': T_expiry,
    'ATM - 75 Price': prices_atm_minus_75,
    'ATM - 75 Normal Volatility': normal_vols_atm_minus_75,
    'ATM + 75 Price': prices_atm_plus_75,
    'ATM + 75 Normal Volatility': normal_vols_atm_plus_75
})

# Set the 'Expiry' column as the index
results_df.set_index('Expiry', inplace=True)
results_df
```

[30]:
```
        ATM - 75 Price  ATM - 75 Normal Volatility  ATM + 75 Price  \
Expiry
1             0.036759                    0.004806        0.000572
2             0.037551                    0.004208        0.001583
3             0.038323                    0.003985        0.002500
4             0.038958                    0.003838        0.003188
5             0.038972                    0.003532        0.004364

        ATM + 75 Normal Volatility
Expiry
1                         0.004773
2                         0.004338
3                         0.004074
4                         0.003839
5                         0.003866
```

(g) Calculate the equivalent Black volatilities for each option in the table above.

[10]:
```
# Calculate equivalent Black volatilities for ATM - 75 and ATM + 75 using the␣
 ↪approximation
black_volatilities_minus_75 = {}
black_volatilities_plus_75 = {}

for expiry in normal_volatilities_df.index:
    T = int(expiry.strip('Y'))
    F = par_swap_rates_percent[T-1]
    K_minus = K_minus_75[T-1]
    K_plus = K_plus_75[T-1]

    sigma_normal_minus = normal_volatilities_df.loc[expiry, 'Normal Volatility␣
 ↪(ATM - 75)']
    sigma_normal_plus = normal_volatilities_df.loc[expiry, 'Normal Volatility␣
 ↪(ATM + 75)']

    sigma_black_minus = sigma_normal_minus * ((F + K_minus) / (2 * np.sqrt(F *␣
 ↪K_minus)))
    sigma_black_plus = sigma_normal_plus * ((F + K_plus) / (2 * np.sqrt(F *␣
 ↪K_plus)))
```

```
    black_volatilities_minus_75[expiry] = sigma_black_minus
    black_volatilities_plus_75[expiry] = sigma_black_plus

# Convert to DataFrame
black_volatilities_df = pd.DataFrame({
    'Black Volatility (ATM - 75)': black_volatilities_minus_75,
    'Black Volatility (ATM + 75)': black_volatilities_plus_75
})

black_volatilities_df
```

[10]:
```
    Black Volatility (ATM - 75)  Black Volatility (ATM + 75)
1Y                     0.011488                     0.007145
2Y                     0.016656                     0.010623
3Y                     0.014259                     0.009764
4Y                     0.012135                     0.008924
5Y                     0.010803                     0.008342
```

(h) Calculate the delta of each options under Black's model

[31]:
```
from scipy.stats import norm

def black_swaption_delta(F, K, T, normal_vol, annuity):
    d1 = (np.log(F / K) + 0.5 * normal_vol**2 * T) / (normal_vol * np.sqrt(T))
    delta = annuity * norm.cdf(d1)
    return delta

# Calculate deltas for each swaption
deltas = []

for i in range(len(T_expiry)):
    F = F0[i]
    alpha, beta, rho, nu = calibrated_params[i]
    annuity = annuity_values[i]

    strikes = F + np.array([-0.0075, -0.0050, -0.0025, -0.0005, 0.0005, 0.0025,␣
 ↪0.0050, 0.0075])

    expiry_deltas = []
    for strike in strikes:
        normal_vol = sabr_normal_volatility(alpha, beta, rho, nu, F, strike,␣
 ↪T_expiry[i])
        delta = black_swaption_delta(F, strike, T_expiry[i], normal_vol,␣
 ↪annuity)
        expiry_deltas.append(delta)

    deltas.append(expiry_deltas)
```

```
# Create a DataFrame to store the deltas
deltas_df = pd.DataFrame(deltas, columns=['ATM - 75', 'ATM - 50', 'ATM - 25',␣
↪'ATM - 5', 'ATM + 5', 'ATM + 25', 'ATM + 50', 'ATM + 75'])
deltas_df.index = T_expiry
deltas_df
```

```
[31]:    ATM - 75  ATM - 50  ATM - 25   ATM - 5       ATM + 5       ATM + 25  \
      1  4.822403  4.822403  4.822403  4.822403  7.151365e-18   0.000000e+00
      2  4.817600  4.817600  4.817600  4.817600  5.776791e-11  3.670711e-209
      3  4.798624  4.798624  4.798624  4.798624  3.423258e-07  2.487824e-132
      4  4.769521  4.769521  4.769521  4.769490  6.081602e-05   4.522849e-87
      5  4.739138  4.739138  4.739138  4.738230  1.472949e-03   1.261053e-57

              ATM + 50  ATM + 75
      1   0.000000e+00       0.0
      2   0.000000e+00       0.0
      3   0.000000e+00       0.0
      4  1.042557e-299       0.0
      5  2.449928e-195       0.0
```

(i) Estimate a SABR smile adjusted delta for each option by calculating the expected implied shift in the volatility, 0 for a given shift in F0. Use this to create a shift of F0 and 0 and use this shift to approximate delta. Compare the delta you obtain using this methodology to the delta you obtained via Black's model. Comment on any differences you observe.

```
[32]: def sabr_shifted_volatility(alpha, beta, rho, nu, F, K, T, F_shift):
          shifted_F = F + F_shift
          shifted_vol = sabr_normal_volatility(alpha, beta, rho, nu, shifted_F, K, T)
          return shifted_vol

      # Calculate SABR smile adjusted deltas for each swaption
      sabr_deltas = []
      F_shift = 0.0001  # Shift in F0 for approximating delta

      for i in range(len(T_expiry)):
          F = F0[i]
          alpha, beta, rho, nu = calibrated_params[i]
          annuity = annuity_values[i]

          strikes = F + np.array([-0.0075, -0.0050, -0.0025, -0.0005, 0.0005, 0.0025,␣
       ↪0.0050, 0.0075])

          expiry_sabr_deltas = []
          for strike in strikes:
              normal_vol = sabr_normal_volatility(alpha, beta, rho, nu, F, strike,␣
       ↪T_expiry[i])
```

```
        shifted_normal_vol = sabr_shifted_volatility(alpha, beta, rho, nu, F,␣
↪strike, T_expiry[i], F_shift)

        delta_approx = (bachelier_swaption_price(F + F_shift, strike,␣
↪T_expiry[i], shifted_normal_vol, annuity) -
                        bachelier_swaption_price(F, strike, T_expiry[i],␣
↪normal_vol, annuity)) / F_shift

        expiry_sabr_deltas.append(delta_approx)

    sabr_deltas.append(expiry_sabr_deltas)

# Create DataFrames to store the SABR deltas and Black's model deltas
sabr_deltas_df = pd.DataFrame(sabr_deltas, columns=['ATM - 75', 'ATM - 50',␣
↪'ATM - 25', 'ATM - 5', 'ATM + 5', 'ATM + 25', 'ATM + 50', 'ATM + 75'])
sabr_deltas_df.index = T_expiry

black_deltas_df = deltas_df  # Rename for clarity

# Display the DataFrames
print("SABR Smile Adjusted Deltas:")
sabr_deltas_df
```

```
SABR Smile Adjusted Deltas:
```

[32]:
```
   ATM - 75  ATM - 50  ATM - 25   ATM - 5   ATM + 5  ATM + 25  ATM + 50  \
1  4.541649  4.116011  3.386989  2.629493  2.228156  1.464628  0.722088
2  4.334510  3.867968  3.215839  2.604285  2.288292  1.676987  1.023744
3  4.151594  3.692080  3.101194  2.568383  2.295302  1.762710  1.171564
4  3.994312  3.552506  3.006229  2.521909  2.274260  1.788651  1.238751
5  3.989341  3.550511  3.030937  2.583834  2.358047  1.916980  1.412053

   ATM + 75
1  0.285179
2  0.551558
3  0.709460
4  0.792075
5  0.985817
```

[33]:
```
print("\nBlack's Model Deltas:")
black_deltas_df
```

```
Black's Model Deltas:
```

[33]:
```
   ATM - 75  ATM - 50  ATM - 25   ATM - 5       ATM + 5       ATM + 25  \
1  4.822403  4.822403  4.822403  4.822403  7.151365e-18   0.000000e+00
2  4.817600  4.817600  4.817600  4.817600  5.776791e-11  3.670711e-209
```

```
3  4.798624  4.798624  4.798624  4.798624  3.423258e-07  2.487824e-132
4  4.769521  4.769521  4.769521  4.769490  6.081602e-05   4.522849e-87
5  4.739138  4.739138  4.739138  4.738230  1.472949e-03   1.261053e-57

        ATM + 50  ATM + 75
1   0.000000e+00       0.0
2   0.000000e+00       0.0
3   0.000000e+00       0.0
4  1.042557e-299       0.0
5  2.449928e-195       0.0
```

Differences:

1. The SABR smile adjusted deltas vary across different strike levels (from ATM-75 to ATM+75) for each expiry, while Black's model deltas remain constant across strikes for a given expiry.
2. The SABR smile adjusted deltas are more symmetrical around the at-the-money (ATM) strike compared to Black's model deltas. For example, at the 1-year expiry, the SABR delta for ATM-75 is 4.541649, and for ATM+75 is 0.285179, showing a more gradual change. In contrast, Black's model delta for ATM-75 is 4.822403, and for ATM+75 is 0.0, indicating a sharp drop-off.
3. Black's model deltas are larger for in-the-money options (from ATM-75 to ATM-5) compared to the SABR smile adjusted deltas. This suggests that Black's model overestimates the sensitivity of option prices to changes in the underlying forward rate for in-the-money options.

The main reason for these differences is that the SABR model incorporates the implied volatility smile, which varies across strike levels and expiries. The SABR model captures the non-constant relationship between implied volatility and the underlying forward rate, leading to a more accurate representation of option price sensitivity (delta).

On the other hand, Black's model assumes constant volatility across all strike levels for a given expiry. It does not take into account the implied volatility smile, resulting in deltas that are constant across strikes and may not accurately reflect the true sensitivity of option prices to changes in the underlying forward rate.

```
[ ]:
```