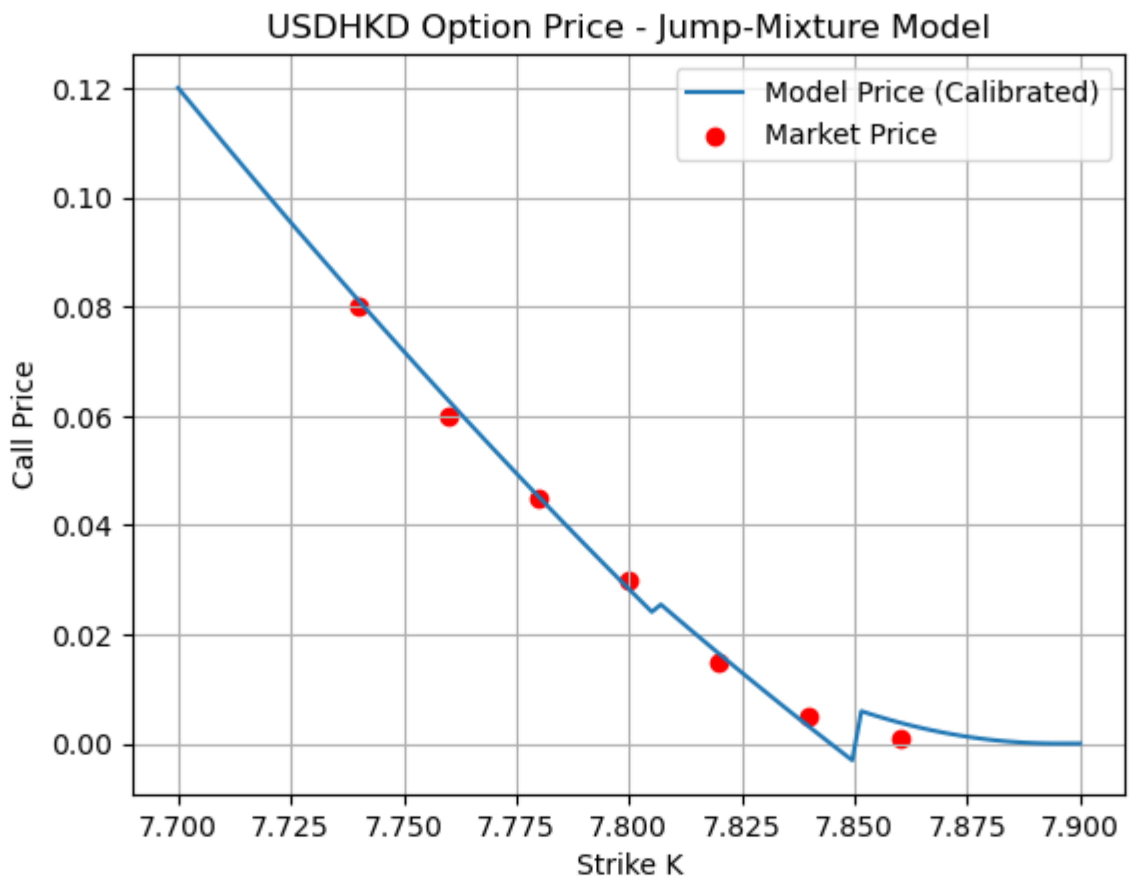
In [2]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize_scalar

# === PARAMETERS ===
a, b = 7.75, 7.85
S0, F = 7.80, 7.82
alpha = np.log(F / S0)
ratio = 1  # q = ratio * p

# === CORE FUNCTIONS ===
def determine_case(K, d):
    bounds = [a * np.exp(-d), b * np.exp(-d), a, b, a * np.exp(d), b * np
    if K < bounds[0]: return 1
    elif bounds[0] <= K < bounds[1]: return 2
    elif bounds[1] <= K < bounds[2]: return 3
    elif bounds[2] <= K < bounds[3]: return 4
    elif bounds[3] <= K < bounds[4]: return 5
    elif bounds[4] <= K < bounds[5]: return 6
    else: return 7

def call_price_closed_form(K, p):
    if p <= 0 or p >= 1 / (1 + 2 * ratio): return np.nan
    d = alpha / (1 - p * (1 + 2 * ratio))
    q = ratio * p
    w0 = p
    w1 = 1 - p * (1 + ratio)
    ax_neg, bx_neg = a * np.exp(-d), b * np.exp(-d)
    Dx_neg = bx_neg - ax_neg
    ax_0, bx_0 = a, b
    Dx_0 = bx_0 - ax_0
    ax_pos, bx_pos = a * np.exp(d), b * np.exp(d)
    Dx_pos = bx_pos - ax_pos
    case = determine_case(K, d)

    if case == 1:
        phi_m1 = (ax_neg + bx_neg) / 2 - K
        phi_0 = (ax_0 + bx_0) / 2 - K
        phi_p1 = (ax_pos + bx_pos) / 2 - K
    elif case == 2:
        phi_m1 = (bx_neg - K)**2 / (2 * Dx_neg)
        phi_0 = (ax_0 + bx_0) / 2 - K
        phi_p1 = (ax_pos + bx_pos) / 2 - K
    elif case == 3:
        phi_m1 = 0.0
        phi_0 = (ax_0 + bx_0) / 2 - K
        phi_p1 = (ax_pos + bx_pos) / 2 - K
    elif case == 4:
        phi_m1 = 0.0
        phi_0 = (bx_0 - K)**2 / (2 * Dx_0)
        phi_p1 = (ax_pos + bx_pos) / 2 - K
    elif case == 5:
        phi_m1 = 0.0
        phi_0 = 0.0
        phi_p1 = (ax_pos + bx_pos) / 2 - K
    elif case == 6:
        phi_m1 = 0.0
        phi_0 = 0.0
        phi_p1 = (bx_pos - K)**2 / (2 * Dx_pos)
```


USDHKD Option Price - Jump-Mixture Model
Model Price (Calibrated)
Market Price
Call Price
Strike K
0.12
0.10
0.08
0.06
0.04
0.02
0.00
7.700
7.725
7.750
7.775
7.800
7.825
7.850
7.875
7.900

In [5]:

```
import numpy as np
from scipy.optimize import minimize_scalar

# --- Known Inputs ---
a, b = 7.75, 7.85
S0, F = 7.80, 7.82
K_market = 7.74
call_market = 0.08
ratio = 1
alpha = np.log(F / S0)

# --- Compute d from p ---
def compute_d(p, ratio, alpha):
    return alpha / (1 - p * (1 + ratio))

# --- Case 1 pricing function (K < ae^{-d}) ---
def call_price_case1(p, K, a, b, ratio, alpha):
    if p <= 0 or p >= 1 / (1 + ratio):
        return np.nan
    d = compute_d(p, ratio, alpha)
    q = ratio * p
    w0 = p
    w1 = 1 - p * (1 + ratio)

    term1 = q * ((a * np.exp(-d) + b * np.exp(-d)) / 2 - K)
    term2 = w0 * ((a + b) / 2 - K)
    term3 = w1 * ((a * np.exp(d) + b * np.exp(d)) / 2 - K)
    return term1 + term2 + term3

# --- Loss function to calibrate p ---
def loss(p):
    model = call_price_case1(p, K_market, a, b, ratio, alpha)
    return (model - call_market)**2 if not np.isnan(model) else np.inf

# --- Minimize loss to find p ---
res = minimize_scalar(loss, bounds=(1e-4, 1 / (1 + ratio) - 1e-4), method=
p_star = res.x
d_star = compute_d(p_star, ratio, alpha)
q_star = ratio * p_star
w1_star = 1 - p_star - q_star

# --- Output results ---
import pandas as pd
params = pd.DataFrame({
    'Parameter': ['p', 'q', '1-p-q', 'd', 'ln(F/S)'],
    'Value': [p_star, q_star, w1_star, d_star, alpha]
})
```

In [6]:
```python
# --- Define Case 2 to Case 6 pricing formulas ---
def call_price_case2(p, K, a, b, ratio, alpha):
    if p <= 0 or p >= 1 / (1 + ratio): return np.nan
    d = compute_d(p, ratio, alpha)
    q = ratio * p
    w0 = p
    w1 = 1 - p * (1 + ratio)
    ae_neg, be_neg = a * np.exp(-d), b * np.exp(-d)
    Dx = be_neg - ae_neg
    term1 = q * ((be_neg - K)**2 / (2 * Dx))
    term2 = w0 * ((a + b) / 2 - K)
    term3 = w1 * ((a * np.exp(d) + b * np.exp(d)) / 2 - K)
    return term1 + term2 + term3

def call_price_case3(p, K, a, b, ratio, alpha):
    if p <= 0 or p >= 1 / (1 + ratio): return np.nan
    d = compute_d(p, ratio, alpha)
    w0 = p
    w1 = 1 - p * (1 + ratio)
    term2 = w0 * ((a + b) / 2 - K)
    term3 = w1 * ((a * np.exp(d) + b * np.exp(d)) / 2 - K)
    return term2 + term3

def call_price_case4(p, K, a, b, ratio, alpha):
    if p <= 0 or p >= 1 / (1 + ratio): return np.nan
    d = compute_d(p, ratio, alpha)
    w0 = p
    w1 = 1 - p * (1 + ratio)
    Dx_0 = b - a
    term2 = w0 * ((b - K)**2 / (2 * Dx_0))
    term3 = w1 * ((a * np.exp(d) + b * np.exp(d)) / 2 - K)
    return term2 + term3

def call_price_case5(p, K, a, b, ratio, alpha):
    if p <= 0 or p >= 1 / (1 + ratio): return np.nan
    d = compute_d(p, ratio, alpha)
    w1 = 1 - p * (1 + ratio)
    term3 = w1 * ((a * np.exp(d) + b * np.exp(d)) / 2 - K)
    return term3

def call_price_case6(p, K, a, b, ratio, alpha):
    if p <= 0 or p >= 1 / (1 + ratio): return np.nan
    d = compute_d(p, ratio, alpha)
    w1 = 1 - p * (1 + ratio)
    ae_pos, be_pos = a * np.exp(d), b * np.exp(d)
    Dx = be_pos - ae_pos
    term3 = w1 * ((be_pos - K)**2 / (2 * Dx))
    return term3

# --- Unified dispatcher based on K ---
def unified_call_price(p, K, a, b, ratio, alpha):
    d = compute_d(p, ratio, alpha)
    bounds = [a * np.exp(-d), b * np.exp(-d), a, b, a * np.exp(d), b * np
    if K < bounds[0]:  # Case 1
        return call_price_case1(p, K, a, b, ratio, alpha)
    elif bounds[0] <= K < bounds[1]:  # Case 2
        return call_price_case2(p, K, a, b, ratio, alpha)
    elif bounds[1] <= K < bounds[2]:  # Case 3
        return call_price_case3(p, K, a, b, ratio, alpha)
    elif bounds[2] <= K < bounds[3]:  # Case 4
```

Out[6]:

	Parameter	Value
0	p	0.000106
1	q	0.000106
2	1-p-q	0.999788
3	d	0.002561
4	ln(F/S)	0.002561

Pegged USDHKD Option Model –

Peg + Jump

- Spot Return: $R = R_{\text{uniform}} + R_{\text{jump}}$
- $R_{\text{uniform}} \sim \mathcal{U}[a, b]$ $a=7.75$, $b=7.85$
- $R_{\text{jump}} \in \{-d, +d\}$ Jump asymmetric

Jump Type	
Jump Down	$q = \text{ratio} \cdot p$
No Jump	p
Jump Up	$1 - p - q = 1 - p(1+\text{ratio})$

Forward Spot

return forward

$$ \ln\left(\frac{F}{S_0}\right) = d \cdot (1 - p(1 + \text{ratio})) $$

$$ d = \frac{\ln(F/S_0)}{1 - p(1 + \text{ratio})} $$

Call Option Pricing (Closed Form)

strike K 7

Case 1: $K < ae^{-d}$

$$ C(K) = q \cdot \left(\frac{ae^{-d} + be^{-d}}{2} - K \right) + p \cdot \left(\frac{a + b}{2} - K \right) p(1 + \text{ratio})) \cdot \left(\frac{ae^d + be^d}{2} - K \right) $$

Case 2: $ae^{-d} \le K < be^{-d}$

$$ C(K) = q \cdot \frac{(be^{-d} - K)^2}{2(be^{-d} - ae^{-d})} + p \cdot \left(\frac{a + b}{2} - K \right) p(1 + \text{ratio})) \cdot \left(\frac{ae^d + be^d}{2} - K \right) $$

Case 3: $be^{-d} \le K < a$

Pegged USDHKD Option Pricing

Spot return

$$ R = R_{\text{uniform}} + R_{\text{jump}} $$

- $R_{\text{uniform}} \sim \mathcal{U}[a, b]$ peg
- $R_{\text{jump}} \in \{-d, 0, +d\}$

Jump Down	$q = \text{ratio} \cdot p$
No Jump	p
Jump Up	$1 - p - q = 1 - p(1 + \text{ratio})$

Forward Spot

$$ \mathbb{E}[e^{R}] = \frac{F}{S_0} \Rightarrow \ln\left(\frac{F}{S_0}\right) = \mathbb{E}[R] = \m
[R_{\text{jump}}] $$

$$ \mathbb{E}[R_{\text{jump}}] = -d \cdot q + 0 \cdot p + d \cdot (1 - p - q) = d \cdot (1 - p(1 + \tex
$$

$$ \boxed{ d = \frac{\ln(F/S_0)}{1 - p(1 + \text{ratio})} } \tag{1} $$

Call Option

jump spot return

- Jump Down: $[a e^{-d}, b e^{-d}]$
- No Jump: $[a, b]$
- Jump Up: $[a e^{d}, b e^{d}]$

Call price

```
In [ ]:
```