# Advanced Equity Derivatives

## Homework 2

Ju Hyung Kang

jk8448

Use the files “Option Prices” and “Gatheral SVI” to produce a local volatility surface on S&P500 index. This can be implemented in Python or another programming language. Discuss your choices and observations on quality of the fit, as well as on arbitrages in the surface.

• To use SPX option prices please refer to https://www.cboe.com/delayed_quotes/spx/quote_table

• Choose maturities (suggest one day a month) called slices

• For each slice, use OTM options: calls with strikes higher than spot, puts with strikes lower

• Use USD rates from https://www.global-rates.com/en/ (interpolate for time slices)

• For dividends use Put Call Parity: takes ATM (closest strike to spot) puts and calls and

Call(T) – Put(T) = Spot . exp{- dividend . T} - Strike . exp{-rate . T}

In [1]:

```
def get_rate(url):
    response = requests.get(url)

    if response.status_code == 200:
        soup = BeautifulSoup(response.text, "html.parser")

        date = soup.select("div > div > section:nth-child(2) > div > div > div:nth-child
        rate = soup.select("div > div > section:nth-child(2) > div > div > div:nth-child
        rate = float(rate)/100

        return rate, date
    else:
        print("Failed to fetch the webpage. Status code:", response.status_code)
```

In [2]:

```
url_1m = "https://www.global-rates.com/en/interest-rates/libor/american-dollar/19/usd-li
url_3m = "https://www.global-rates.com/en/interest-rates/libor/american-dollar/21/usd-li
url_6m = "https://www.global-rates.com/en/interest-rates/libor/american-dollar/24/usd-li
```

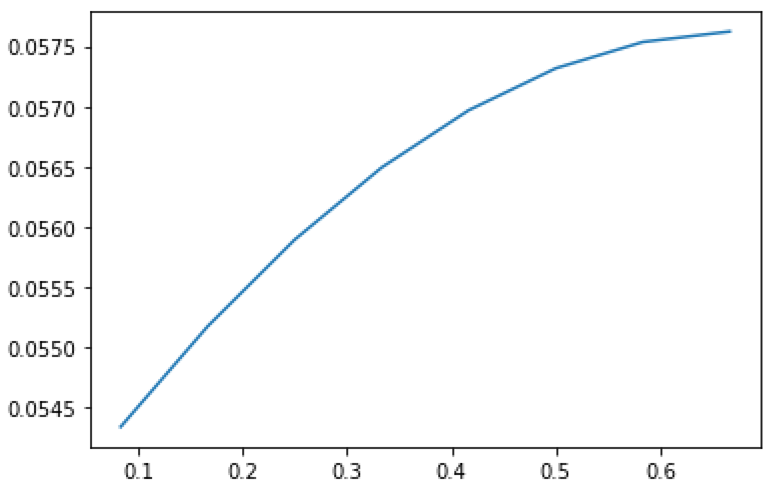
In [3]:

```
import requests
from bs4 import BeautifulSoup

rate_1m, date_1m = get_rate(url_1m)
rate_3m, date_3m = get_rate(url_3m)
rate_6m, date_6m = get_rate(url_6m)

print(date_1m, rate_1m)
print(date_3m, rate_3m)
print(date_6m, rate_6m)
```

```
04-15-2024 0.0543072
04-15-2024 0.055786800000000004
04-15-2024 0.057101
```

In [4]:
```
# 04-12-2024 us rates
rate_1m = 0.0543377
rate_3m = 0.0558927
rate_6m = 0.0573163
```

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt

def rate(x, x1=1/12, y1=rate_1m, x2=3/12, y2=rate_3m, x3=6/12, y3=rate_6m):
    # Log interpolation
    log_y1 = np.log(y1)
    log_y2 = np.log(y2)
    log_y3 = np.log(y3)

    # Fit a quadratic function in logarithmic space
    A = np.array([[1, x1, x1**2], [1, x2, x2**2], [1, x3, x3**2]])
    b = np.array([log_y1, log_y2, log_y3])
    coeffs = np.linalg.solve(A, b)

    # Calculate the interpolated value at x
    log_y = coeffs[0] + coeffs[1]*x + coeffs[2]*x**2
    return np.exp(log_y)

x = [i/12 for i in range(1,9)]
y = [rate(i) for i in x]
plt.plot(x,y)
plt.show()
```




In [6]:
```
import pandas as pd

option_chain_04 = pd.read_csv("option_prices/202404.csv", skiprows=3)
option_chain_05 = pd.read_csv("option_prices/202405.csv", skiprows=3)
option_chain_06 = pd.read_csv("option_prices/202406.csv", skiprows=3)
option_chain_07 = pd.read_csv("option_prices/202407.csv", skiprows=3)
option_chain_08 = pd.read_csv("option_prices/202408.csv", skiprows=3)
option_chain_09 = pd.read_csv("option_prices/202409.csv", skiprows=3)
option_chain_10 = pd.read_csv("option_prices/202410.csv", skiprows=3)
```

In [7]:
```
from datetime import datetime

TODAY = pd.read_csv("option_prices/202404.csv", skiprows=2, nrows=1, header=None)[0][0].
TODAY = datetime.strptime(TODAY, "%B %d, %Y at %I:%M %p")
TODAY
```

Out[7]:
```
datetime.datetime(2024, 4, 12, 11, 54)
```

In [8]:
```
NUM_DAYS_IN_YEAR = 252
```

In [9]:
```
S0 = float(pd.read_csv("option_prices/202404.csv", skiprows=1, nrows=1, header=None)[1][
S0
```
Out[9]:
```
5151.75
```
In [10]:
```
from scipy.stats import norm

def N(x):
    return norm.cdf(x)
```
```
C:\Users\kangj\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumPy v
ersion >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.2
6.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```
In [11]:
```
from scipy.optimize import newton

# Call implied volatiltiy
def call_iv(C, S, tau, K, r, q):
    def equation(sigma):
        d1 = (np.log(S / K) + (r - q + sigma**2 / 2) * tau)
        d2 = d1 - sigma * np.sqrt(tau)

        return C - (S * np.exp(-q * tau) * N(d1) - K * np.exp(-r * tau) * N(d2))

    iv = newton(equation, x0=0.2, tol=1e-10, maxiter=1000)

    return iv

# Put implied volatiltiy
def put_iv(P, S, tau, K, r, q):
    def equation(sigma):
        d1 = (np.log(S / K) + (r - q + sigma**2 / 2) * tau)
        d2 = d1 - sigma * np.sqrt(tau)

        return P - (K * np.exp(-r * tau) * N(-d2) - S * np.exp(-q * tau) * N(-d1))

    iv = newton(equation, x0=0.2, tol=1e-10, maxiter=1000)

    return iv

# Calculate iv array
def iv(price, S, tau, K, r, q, mode):
    n = len(price)
    iv = np.zeros(n)

    if mode=="C":
        for i in range(n):
            iv[i] = call_iv(price[i], S, tau, K[i], r, q)
    elif mode=="P":
        for i in range(n):
            iv[i] = put_iv(price[i], S, tau, K[i], r, q)

    return iv
```
In [12]:
```
import pandas as pd
from datetime import datetime
from scipy.optimize import curve_fit


def raw(k, a, b, rho, m, sigma):
    return a + b * (rho * (k - m) + np.sqrt((k-m)**2 + sigma**2))

def natural(k, Delta, mu, rho, w, zeta):
```

```python
    return Delta + w / 2 * (1 + zeta * rho * (k - mu) + np.sqrt((zeta * (k - mu) + rho)*

def option_chain_process(option_chain):
    grouped = option_chain.groupby('Expiration Date')

    grouped_dfs = {}

    for group_name, group_data in grouped:
        try:
            # Make dataframe
            group_df = pd.DataFrame(group_data).reset_index(drop=True)
            group_df = group_df[group_df['Calls'].str.startswith('SPXW')]

            date_obj = datetime.strptime(group_name, "%a %b %d %Y")
            formatted_date_str = date_obj.strftime("%m%d%y")

            # Get expiration date
            expiration_date_str = group_df['Expiration Date'].iloc[0]

            # Parse the date string into a datetime object
            expiration_date = datetime.strptime(expiration_date_str, "%a %b %d %Y")

            # Set the time to 4:00 PM (16:00)
            expiration_date = expiration_date.replace(hour=16, minute=0, second=0)

            # tau (T-0)
            tau = (expiration_date - TODAY).total_seconds() / 3600 / 24 / NUM_DAYS_IN_YE

            # r
            r = rate(tau)

            # F
            F = S0*np.exp(r*tau)

            ### Dividend
            K = np.array(group_df['Strike'])

            # ATM index
            atm_K_idx = np.abs(K - S0).argmin()

            # Get price of ATM call and put
            atm_C = (group_df['Bid'].iloc[atm_K_idx] + group_df['Ask'].iloc[atm_K_idx])
            atm_P = (group_df['Bid.1'].iloc[atm_K_idx] + group_df['Ask.1'].iloc[atm_K_id

            atm_K = K[atm_K_idx]

            # q
            q = np.log((atm_C - atm_P + atm_K * np.exp(-r * tau)) / S0) / (-tau)

            # Implied volatility
            n = len(group_df)

            call_bid = np.array(group_df['Bid'])
            put_bid = np.array(group_df['Bid.1'])

            call_ask = np.array(group_df['Ask'])
            put_ask = np.array(group_df['Ask.1'])

            put_bid_iv = iv(put_bid[K <= F], S0, tau, K[K <= F], r, q, "P")
            call_bid_iv = iv(call_bid[K > F], S0, tau, K[K > F], r, q, "C")

            bid_iv = np.append(put_bid_iv, call_bid_iv)

            put_ask_iv = iv(put_ask[K <= F], S0, tau, K[K <= F], r, q, "P")
            call_ask_iv = iv(call_ask[K > F], S0, tau, K[K > F], r, q, "C")
```

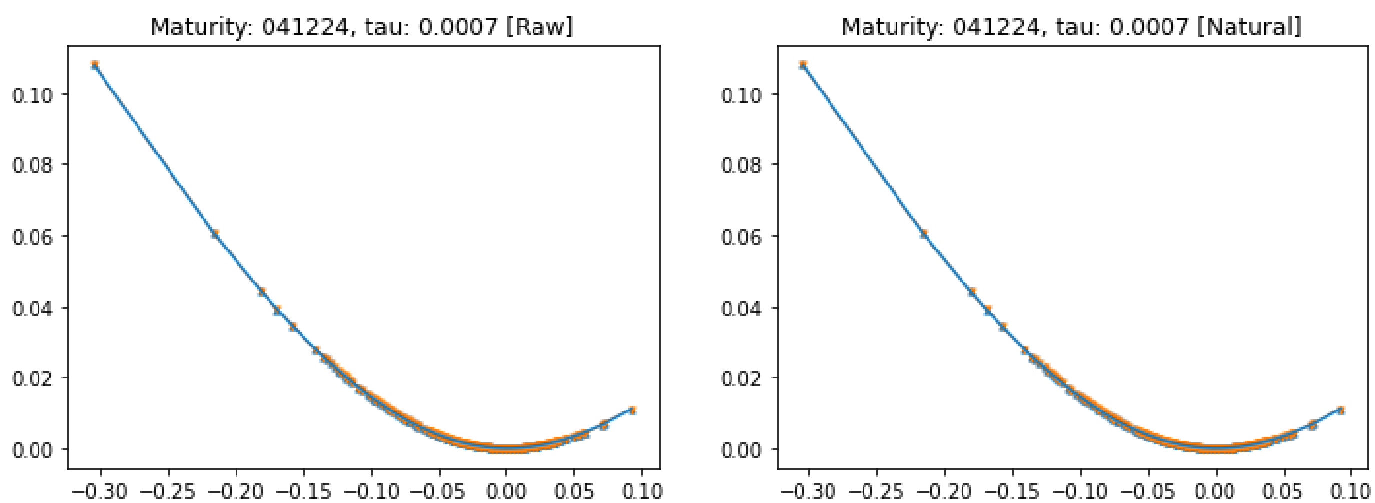
```
            ask_iv = np.append(put_ask_iv, call_ask_iv)

            group_df['bid_iv'] = bid_iv
            group_df['ask_iv'] = ask_iv

            bid_w = bid_iv**2 * tau
            ask_w = ask_iv**2 * tau

            group_df['bid_w'] = bid_w
            group_df['ask_w'] = ask_w

            # k (log strike)
            strike = np.array(group_df['Strike'])
            k = np.log(strike/F)
            group_df['k'] = k

            # raw
            x_data = np.append(k, k)
            y_data = np.append(bid_w, ask_w)

            lower_bounds = [-np.inf, 0, -1, -np.inf, 1e-9]
            upper_bounds = [np.inf, np.inf, 1, np.inf, np.inf]

            raw_params, covariance = curve_fit(raw, x_data, y_data, bounds=(lower_bounds

            a, b, rho, m, sigma = raw_params

            raw_w = raw(k, a, b, rho, m, sigma)

            group_df['raw_w'] = raw_w

            # natural
            lower_bounds = [-np.inf, -np.inf, -1, 0, 1e-9]
            upper_bounds = [np.inf, np.inf, 1, np.inf, np.inf]

            natural_params, covariance = curve_fit(natural, x_data, y_data, bounds=(lowe

            Delta, mu, rho, w, zeta = natural_params

            natural_w = natural(k, Delta, mu, rho, w, zeta)

            group_df['natural_w'] = natural_w

            # SVI-JW
            a, b, rho, m, sigma = raw_params

            v_t = (a + b * (-rho * m + np.sqrt(m**2 + sigma**2))) / tau
            w_t = v_t * tau
            psi_t = 1 / np.sqrt(w_t) * b / 2 * (-m / np.sqrt(m**2 + sigma**2) + rho)
            p_t = 1 / np.sqrt(w_t) * b * (1 - rho)
            c_t = 1 / np.sqrt(w_t) * b * (1 + rho)
            v_tilde_t = (a + b * sigma * np.sqrt(1 - rho**2)) / tau

            svi_jw_parmas = [v_t, psi_t, p_t, c_t, v_tilde_t]

            # Filter columns
            group_df = group_df[['k', 'Strike', 'Bid', 'Bid.1', 'Ask', 'Ask.1', 'IV', 'I
#               print(group_df)

            grouped_dfs[formatted_date_str] = [tau, raw_params, natural_params, svi_jw_p
        except:
            print(group_name, "passed")

    return grouped_dfs
```

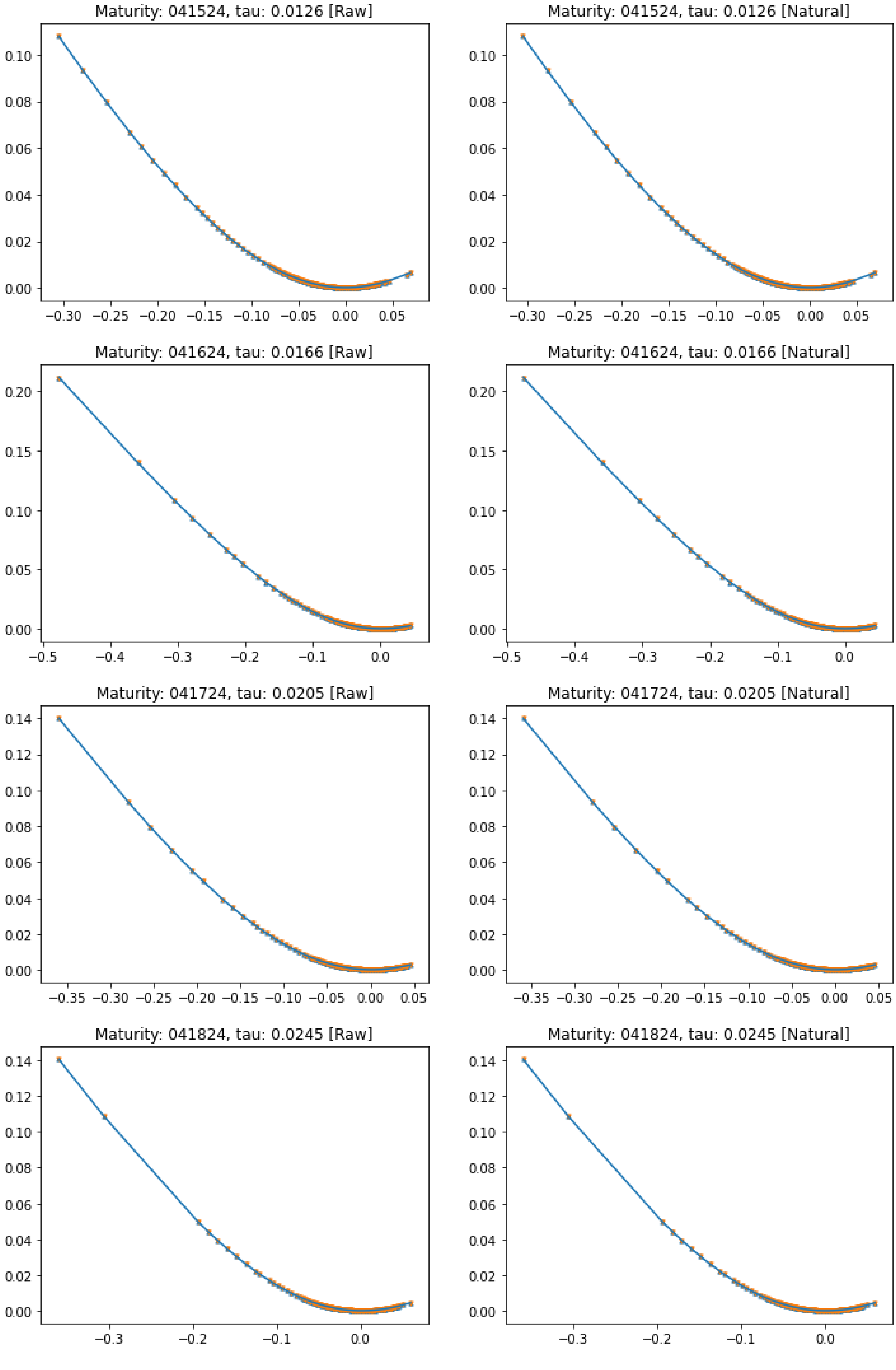
In [13]:
```
processed_04 = option_chain_process(option_chain_04)
processed_05 = option_chain_process(option_chain_05)
processed_06 = option_chain_process(option_chain_06)
processed_07 = option_chain_process(option_chain_07)
processed_08 = option_chain_process(option_chain_08)
processed_09 = option_chain_process(option_chain_09)
```

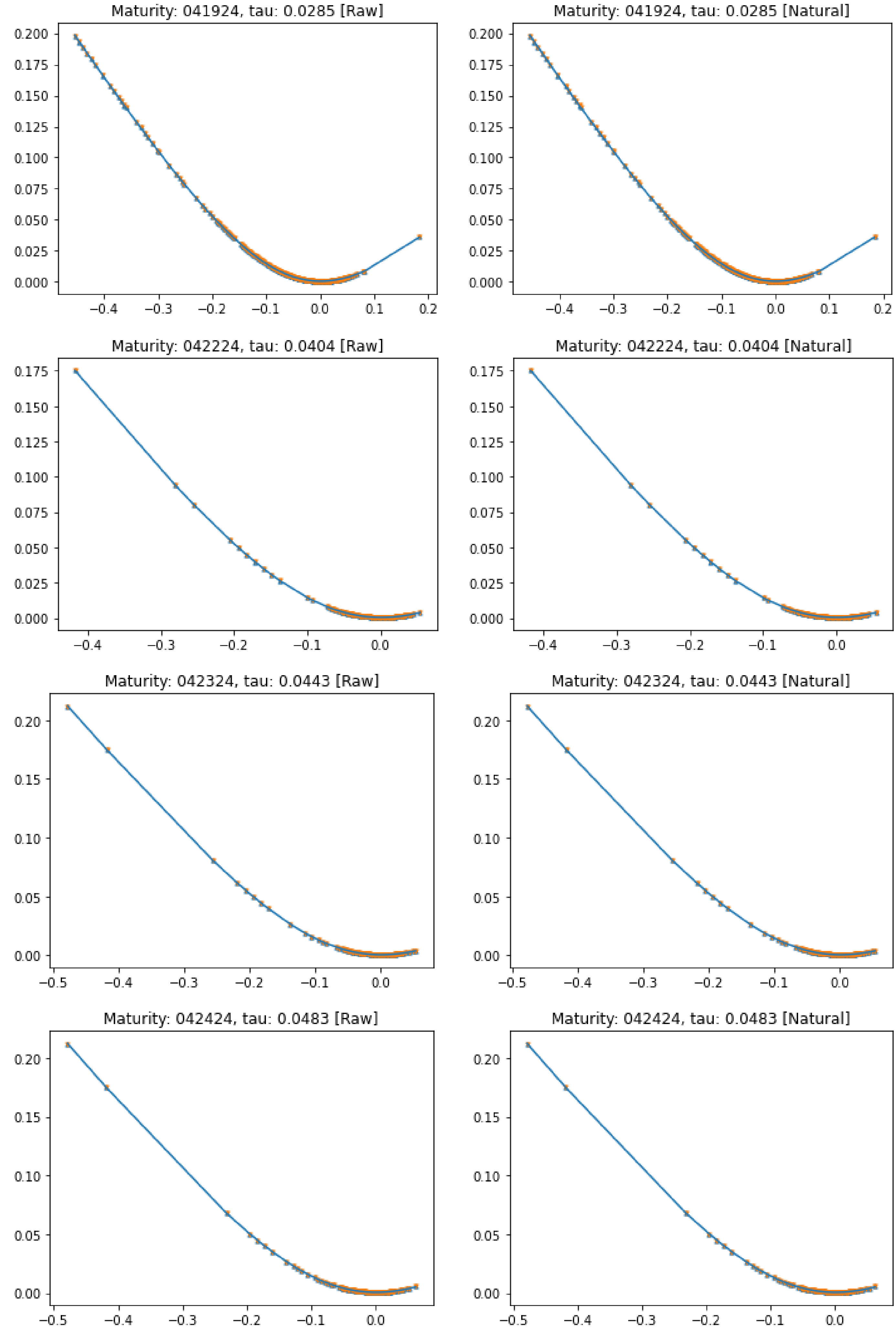
```
Mon May 20 2024 passed
Tue May 07 2024 passed
Tue May 21 2024 passed
Fri Sep 20 2024 passed
```

In [14]:
```
def plot(option_dict):
    keys = option_dict.keys()
    options_li = sorted(option_dict.items())

    for key, option in options_li:
        df = option[-1]
        tau = option[0]

        fig, axs = plt.subplots(1, 2, figsize=(12, 4))

        axs[0].scatter(df['k'], df['bid_w'], marker='^', s=10)
        axs[0].scatter(df['k'], df['ask_w'], marker='v', s=10)
        axs[0].plot(df['k'], df['raw_w'])
        axs[0].set_title(f'Maturity: {key}, tau: {np.round(tau,4)} [Raw]')

        axs[1].scatter(df['k'], df['bid_w'], marker='^', s=10)
        axs[1].scatter(df['k'], df['ask_w'], marker='v', s=10)
        axs[1].plot(df['k'], df['natural_w'])
        axs[1].set_title(f'Maturity: {key}, tau: {np.round(tau,4)} [Natural]')

        plt.show()
```

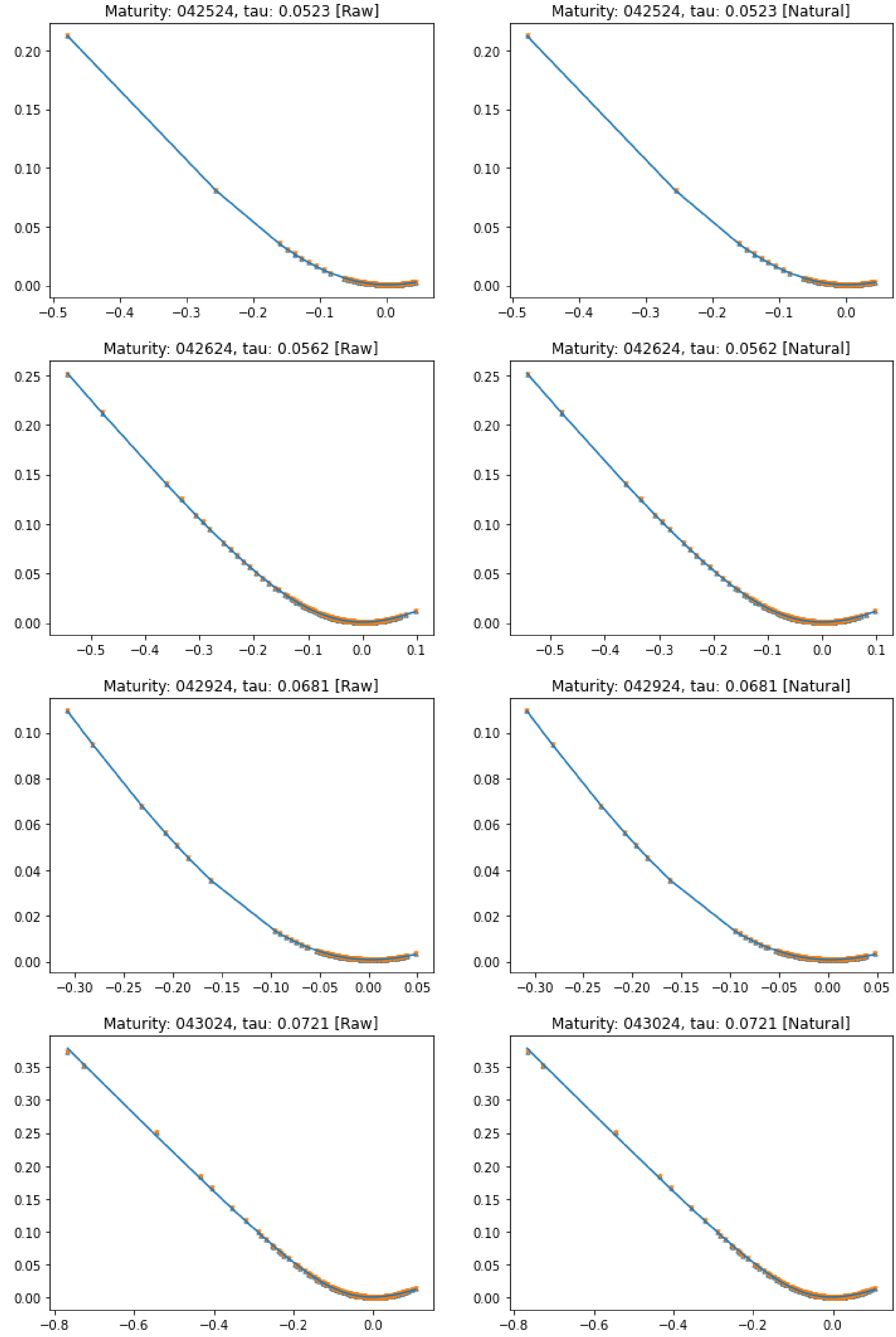
In [15]:
```
plot(processed_04)
plot(processed_05)
plot(processed_06)
plot(processed_07)
plot(processed_08)
plot(processed_09)
```

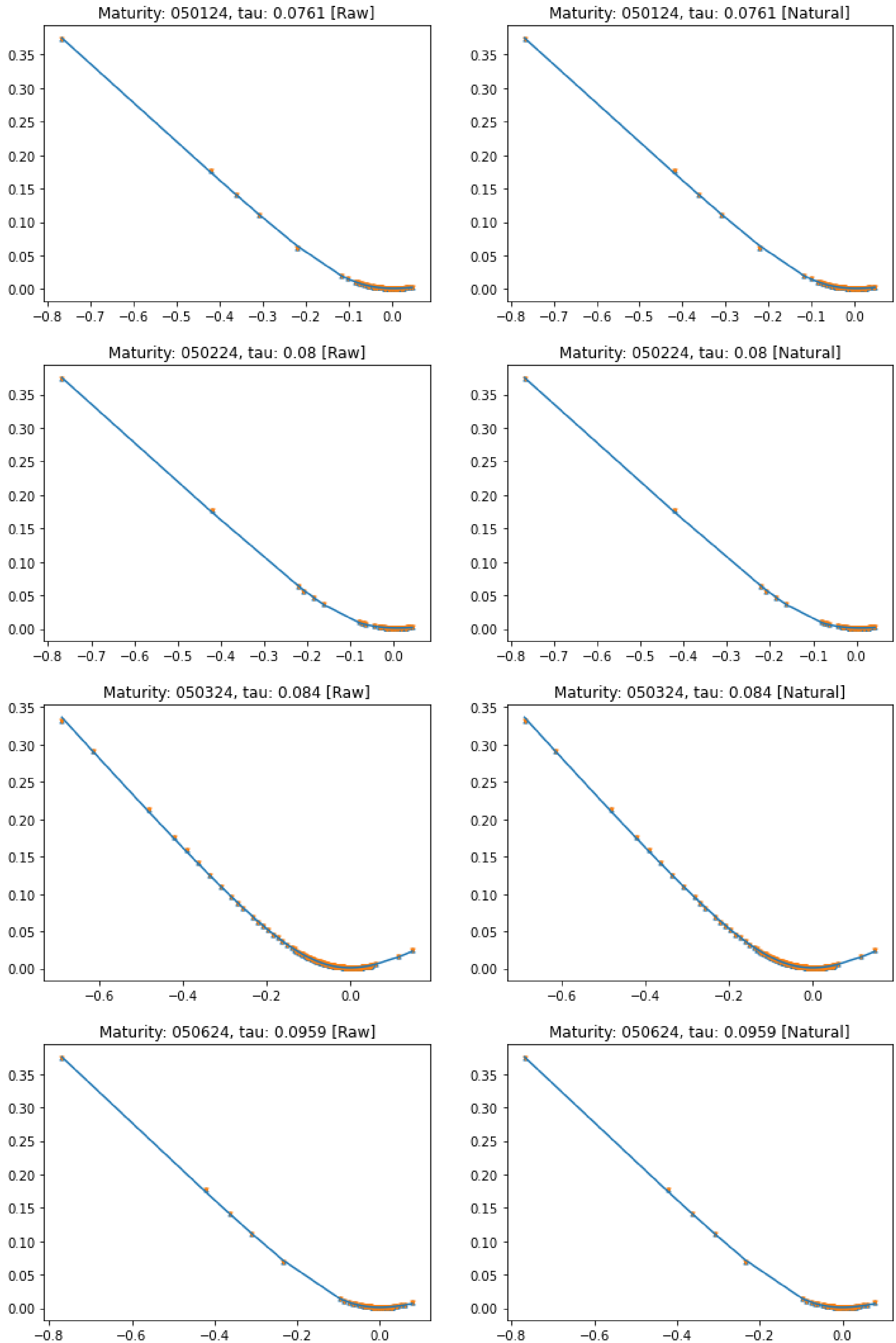


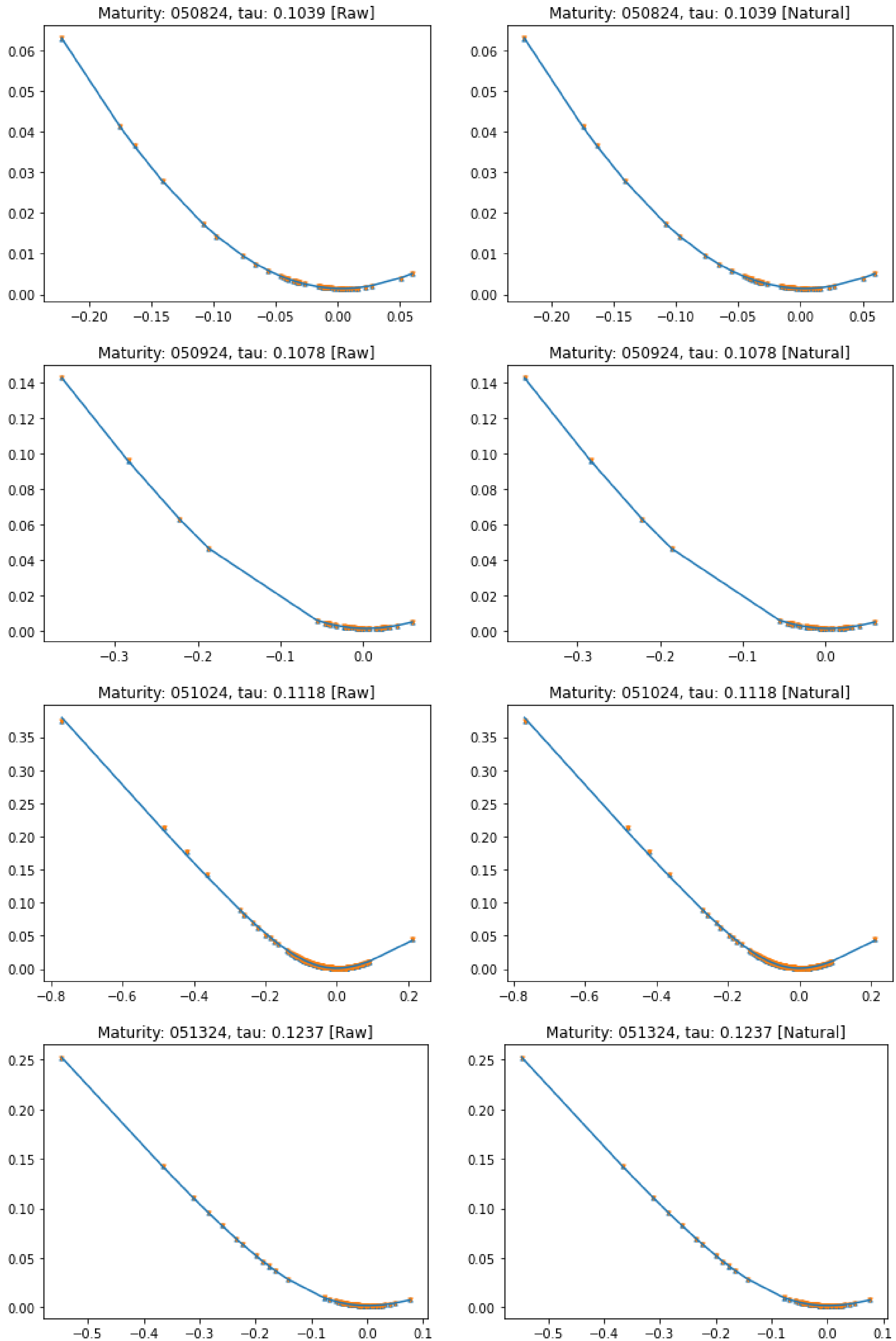

Maturity: 041524, tau: 0.0126 [Raw]
0.10
0.08
0.06
0.04
0.02
0.00
−0.30 −0.25 −0.20 −0.15 −0.10 −0.05 0.00 0.05
Maturity: 041524, tau: 0.0126 [Natural]
0.10
0.08
0.06
0.04
0.02
0.00
−0.30 −0.25 −0.20 −0.15 −0.10 −0.05 0.00 0.05
Maturity: 041624, tau: 0.0166 [Raw]
0.20
0.15
0.10
0.05
0.00
−0.5 −0.4 −0.3 −0.2 −0.1 0.0
Maturity: 041624, tau: 0.0166 [Natural]
0.20
0.15
0.10
0.05
0.00
−0.5 −0.4 −0.3 −0.2 −0.1 0.0
Maturity: 041724, tau: 0.0205 [Raw]
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−0.35 −0.30 −0.25 −0.20 −0.15 −0.10 −0.05 0.00 0.05
Maturity: 041724, tau: 0.0205 [Natural]
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−0.35 −0.30 −0.25 −0.20 −0.15 −0.10 −0.05 0.00 0.05
Maturity: 041824, tau: 0.0245 [Raw]
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−0.3 −0.2 −0.1 0.0
Maturity: 041824, tau: 0.0245 [Natural]
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−0.3 −0.2 −0.1 0.0

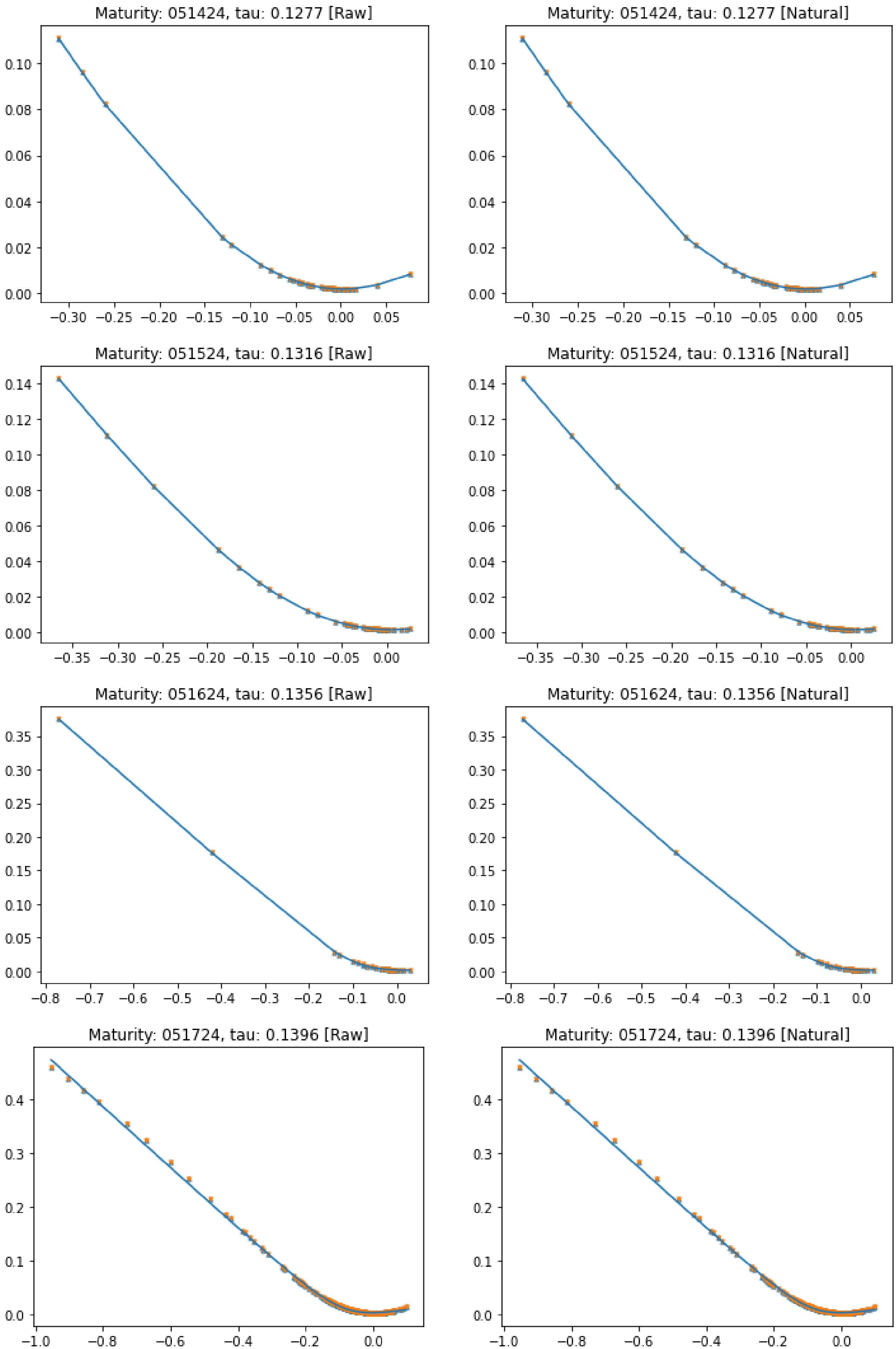


Maturity: 041924, tau: 0.0285 [Raw]
Maturity: 041924, tau: 0.0285 [Natural]
Maturity: 042224, tau: 0.0404 [Raw]
Maturity: 042224, tau: 0.0404 [Natural]
Maturity: 042324, tau: 0.0443 [Raw]
Maturity: 042324, tau: 0.0443 [Natural]
Maturity: 042424, tau: 0.0483 [Raw]
Maturity: 042424, tau: 0.0483 [Natural]

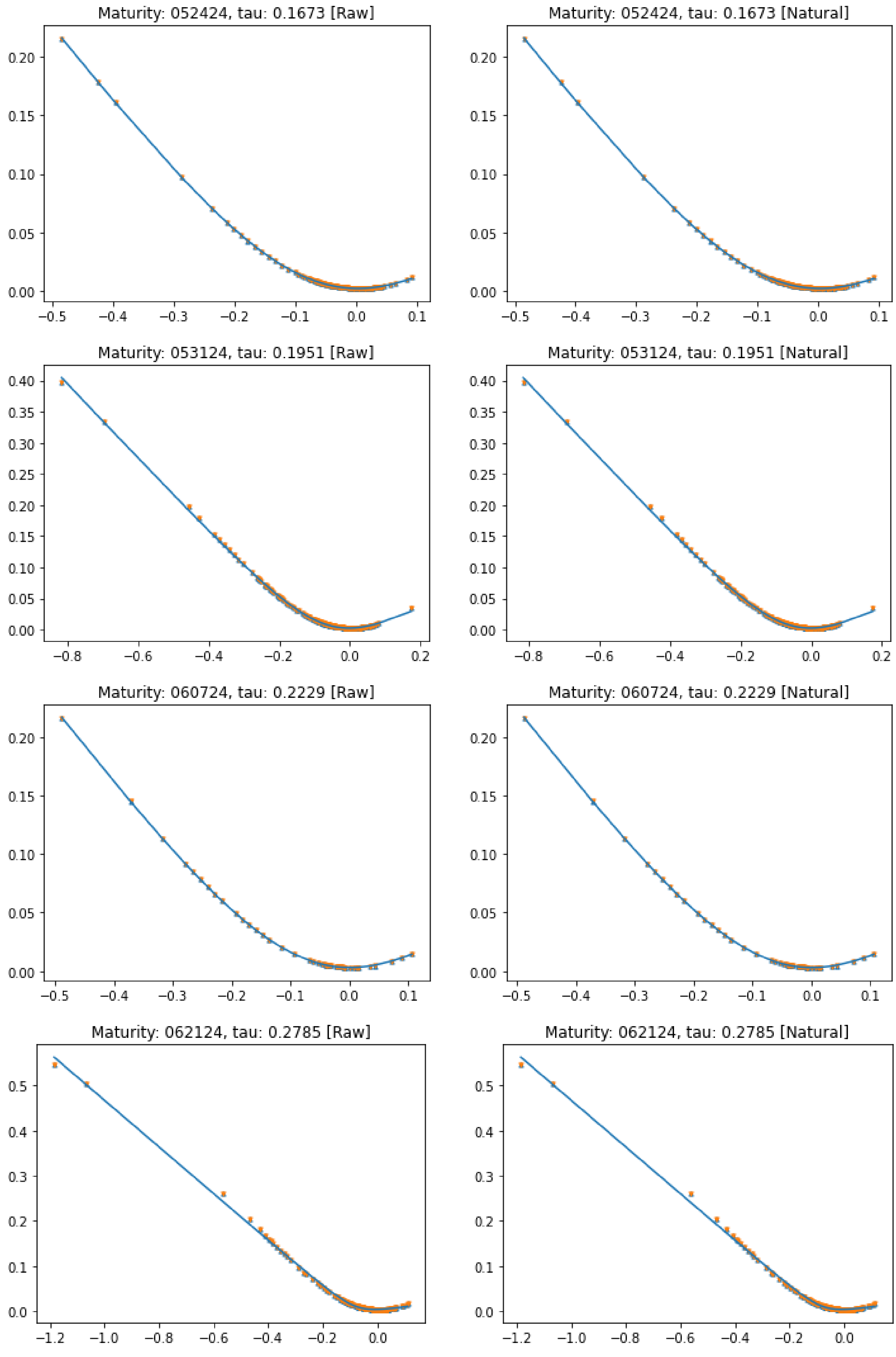

Maturity: 042524, tau: 0.0523 [Raw]
Maturity: 042524, tau: 0.0523 [Natural]
0.20
0.15
0.10
0.05
0.00
−0.5
−0.4
−0.3
−0.2
−0.1
0.0
Maturity: 042624, tau: 0.0562 [Raw]
Maturity: 042624, tau: 0.0562 [Natural]
0.25
0.20
0.15
0.10
0.05
0.00
−0.5
−0.4
−0.3
−0.2
−0.1
0.0
0.1
Maturity: 042924, tau: 0.0681 [Raw]
Maturity: 042924, tau: 0.0681 [Natural]
0.10
0.08
0.06
0.04
0.02
0.00
−0.30
−0.25
−0.20
−0.15
−0.10
−0.05
0.00
0.05
Maturity: 043024, tau: 0.0721 [Raw]
Maturity: 043024, tau: 0.0721 [Natural]
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−0.8
−0.6
−0.4
−0.2
0.0

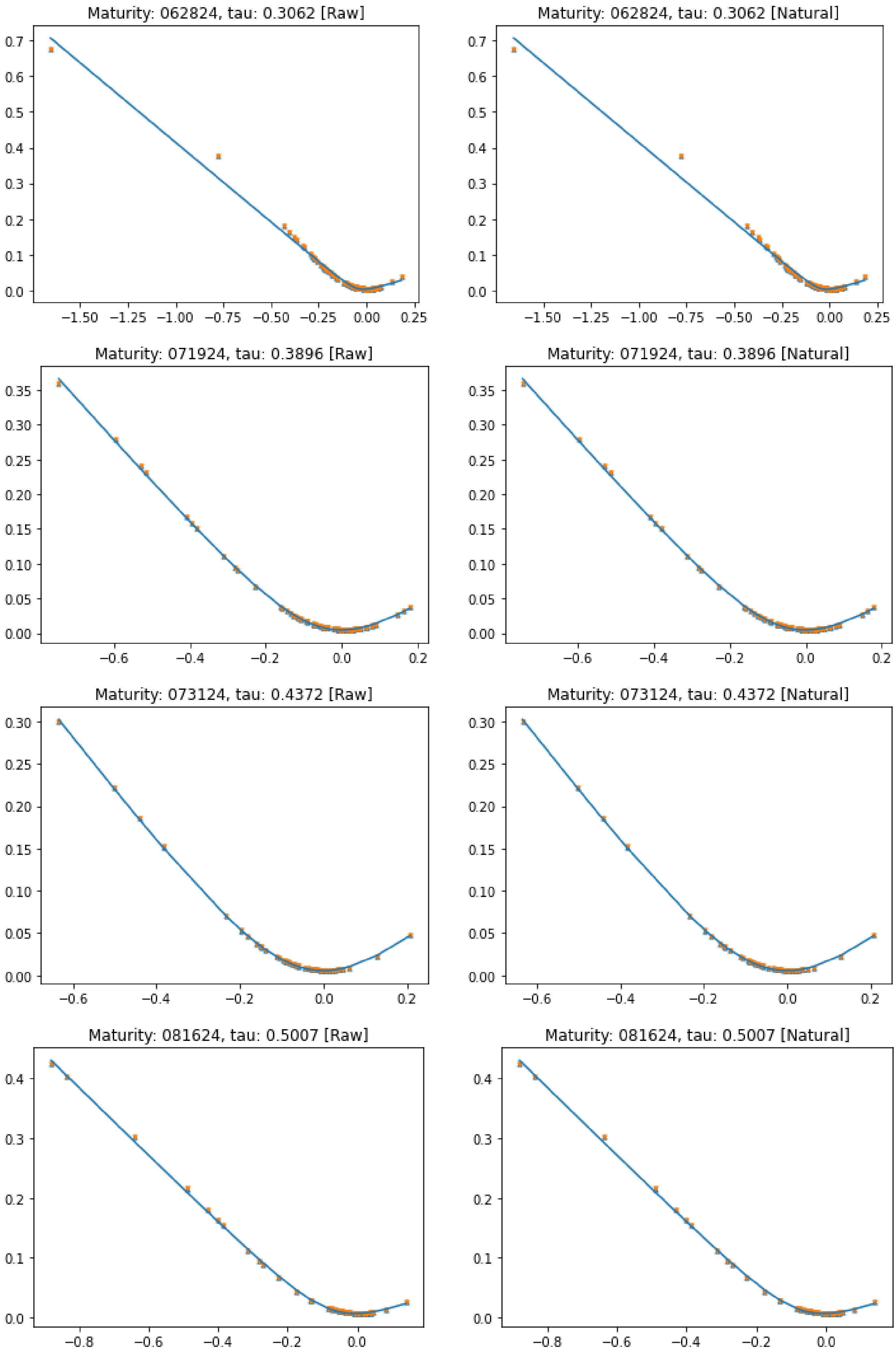

Maturity: 050124, tau: 0.0761 [Raw]
Maturity: 050124, tau: 0.0761 [Natural]
Maturity: 050224, tau: 0.08 [Raw]
Maturity: 050224, tau: 0.08 [Natural]
Maturity: 050324, tau: 0.084 [Raw]
Maturity: 050324, tau: 0.084 [Natural]
Maturity: 050624, tau: 0.0959 [Raw]
Maturity: 050624, tau: 0.0959 [Natural]
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35
−0.8 −0.7 −0.6 −0.5 −0.4 −0.3 −0.2 −0.1 0.0
−0.6 −0.4 −0.2 0.0
−0.8 −0.6 −0.4 −0.2 0.0

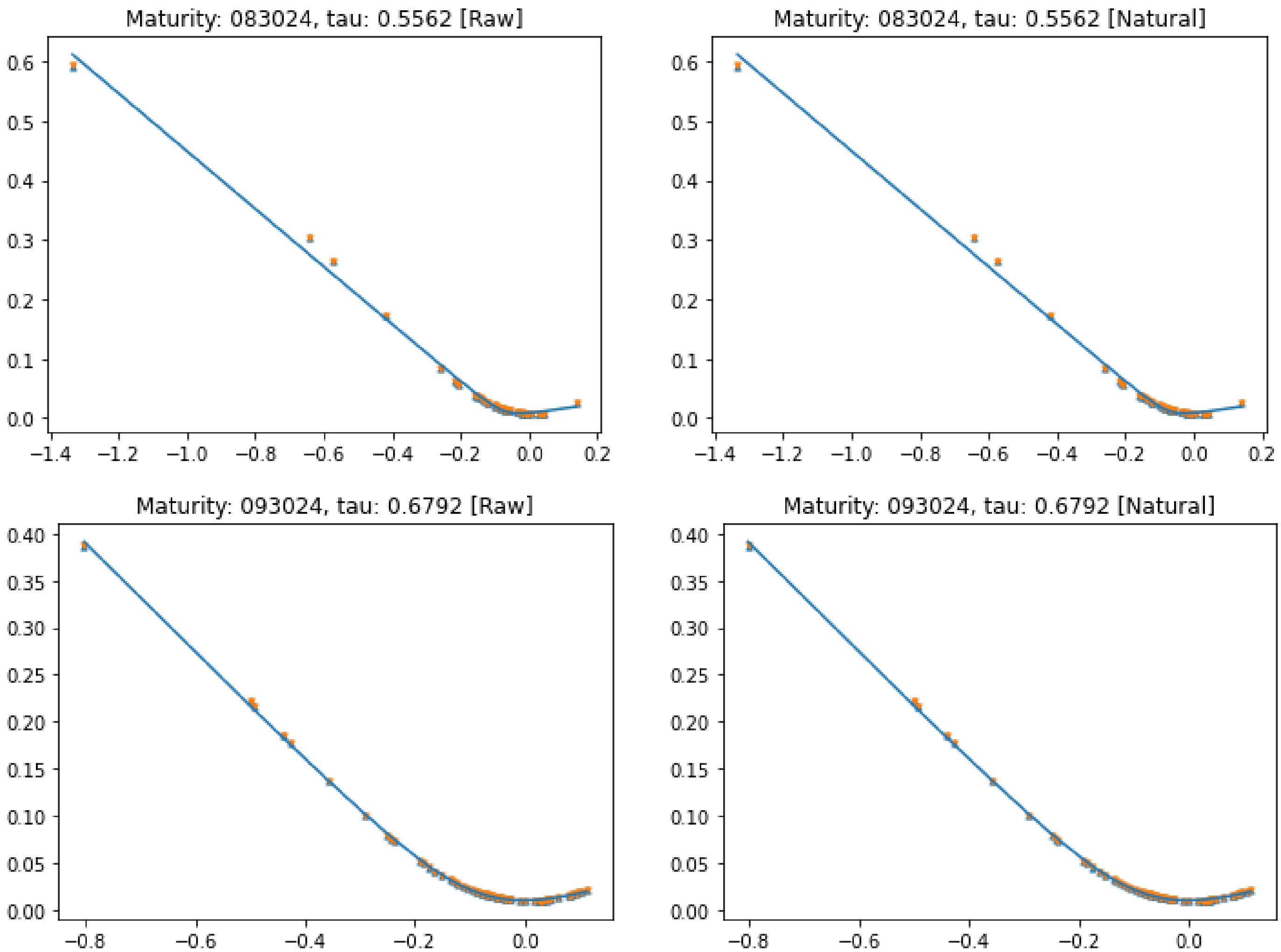

Maturity: 050824, tau: 0.1039 [Raw]
Maturity: 050824, tau: 0.1039 [Natural]
Maturity: 050924, tau: 0.1078 [Raw]
Maturity: 050924, tau: 0.1078 [Natural]
Maturity: 051024, tau: 0.1118 [Raw]
Maturity: 051024, tau: 0.1118 [Natural]
Maturity: 051324, tau: 0.1237 [Raw]
Maturity: 051324, tau: 0.1237 [Natural]



Maturity: 051424, tau: 0.1277 [Raw]
Maturity: 051424, tau: 0.1277 [Natural]
0.00 0.02 0.04 0.06 0.08 0.10
−0.30 −0.25 −0.20 −0.15 −0.10 −0.05 0.00 0.05
Maturity: 051524, tau: 0.1316 [Raw]
Maturity: 051524, tau: 0.1316 [Natural]
0.00 0.02 0.04 0.06 0.08 0.10 0.12 0.14
−0.35 −0.30 −0.25 −0.20 −0.15 −0.10 −0.05 0.00
Maturity: 051624, tau: 0.1356 [Raw]
Maturity: 051624, tau: 0.1356 [Natural]
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35
−0.8 −0.7 −0.6 −0.5 −0.4 −0.3 −0.2 −0.1 0.0
Maturity: 051724, tau: 0.1396 [Raw]
Maturity: 051724, tau: 0.1396 [Natural]
0.0 0.1 0.2 0.3 0.4
−1.0 −0.8 −0.6 −0.4 −0.2 0.0


Maturity: 052424, tau: 0.1673 [Raw]
Maturity: 052424, tau: 0.1673 [Natural]
Maturity: 053124, tau: 0.1951 [Raw]
Maturity: 053124, tau: 0.1951 [Natural]
Maturity: 060724, tau: 0.2229 [Raw]
Maturity: 060724, tau: 0.2229 [Natural]
Maturity: 062124, tau: 0.2785 [Raw]
Maturity: 062124, tau: 0.2785 [Natural]


Maturity: 062824, tau: 0.3062 [Raw]
Maturity: 062824, tau: 0.3062 [Natural]
Maturity: 071924, tau: 0.3896 [Raw]
Maturity: 071924, tau: 0.3896 [Natural]
Maturity: 073124, tau: 0.4372 [Raw]
Maturity: 073124, tau: 0.4372 [Natural]
Maturity: 081624, tau: 0.5007 [Raw]
Maturity: 081624, tau: 0.5007 [Natural]




The raw SVI parametrization and the natural SVI parameterization showcases no big difference in their fit. Based on the plots, both parametrizations fit well with the bid/ask implied variances.

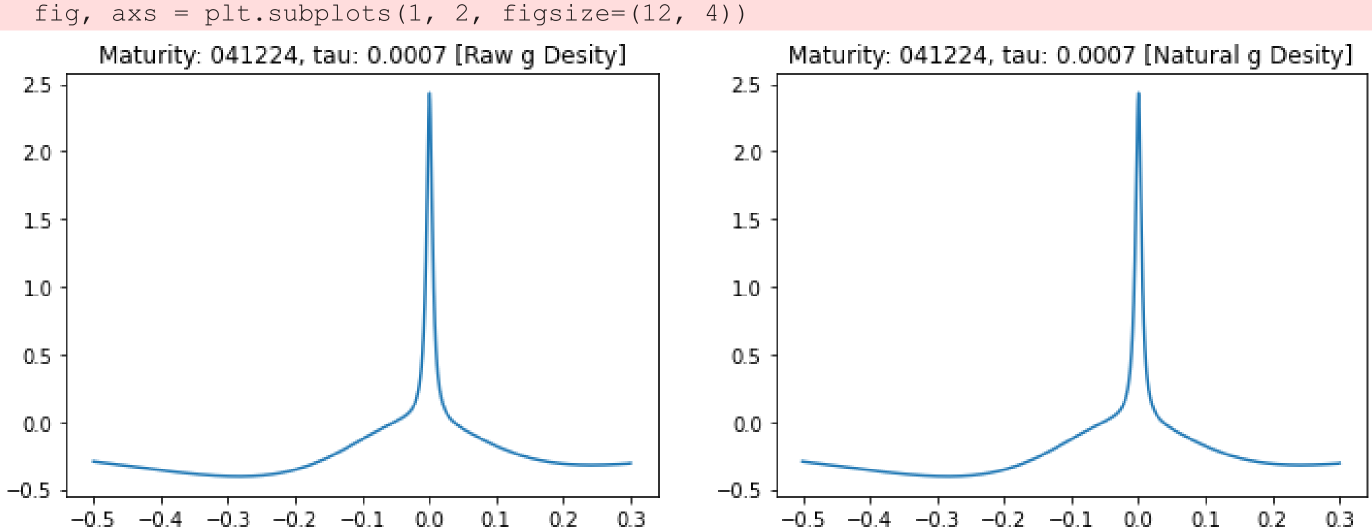
In [16]:

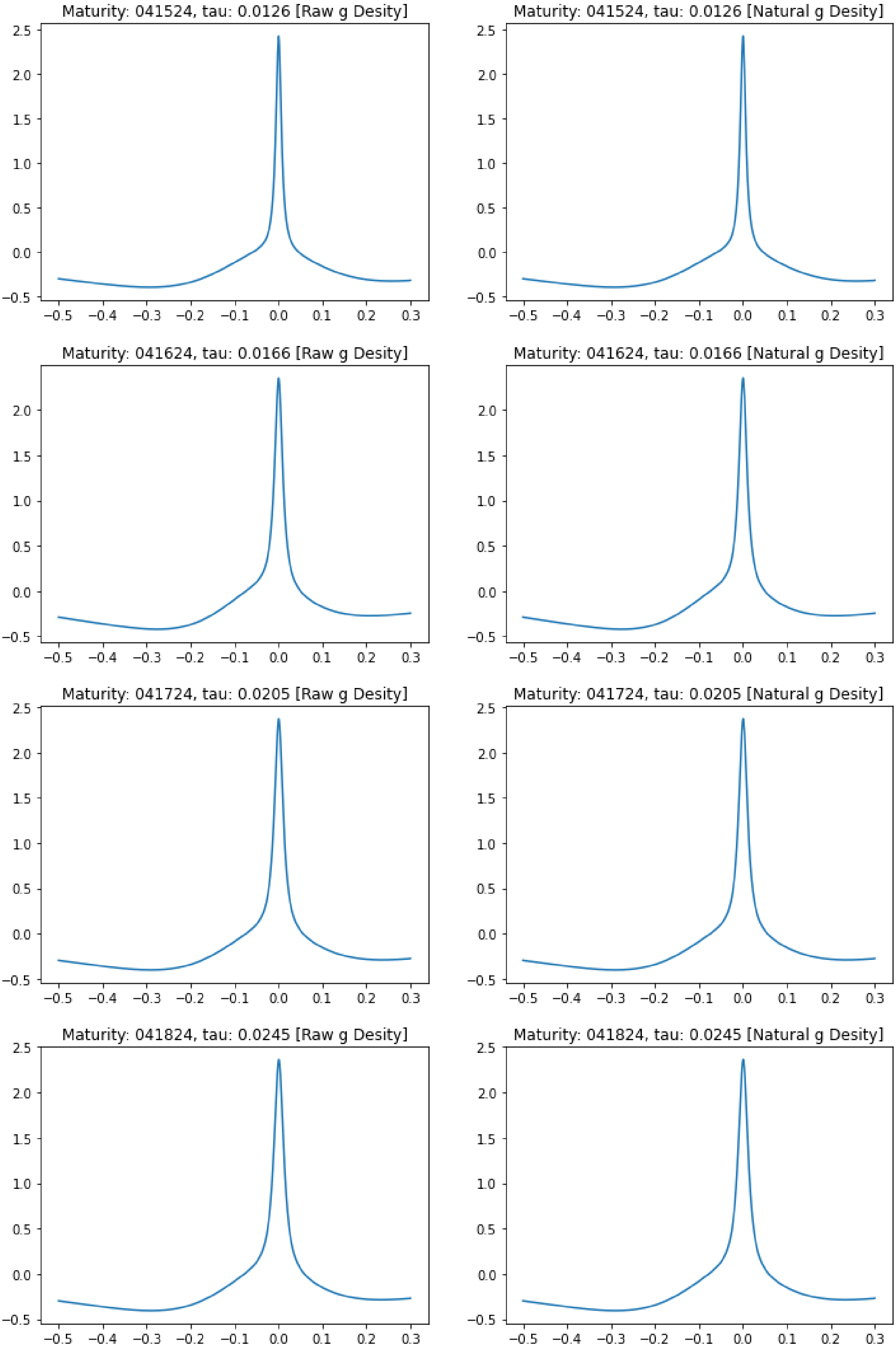
```
def g(k, w):
    n = len(k)
    h = k[1]-k[0]

    w_1 = np.zeros(n)
    w_2 = np.zeros(n)

    for i in range(n):
        if i == 0:
            w_1[i] = (w[i+1] - w[i]) / h
        elif 0 < i < n-1:
            w_1[i] = (w[i+1] - w[i-1]) / (2 * h)
        elif i == n-1:
            w_1[i] = (w[i] - w[i-1]) / h

    for i in range(n):
        if i == 0:
            w_2[i] = (w[i+2] - 2 * w[i+1] + w[i]) / h**2
        elif 0 < i < n-1:
            w_2[i] = (w[i+1] - 2 * w[i] + w[i-1]) / h**2
        elif i == n-1:
            w_2[i] = (w[i-2] - 2 * w[i-1] + w[i]) / h**2

    g = (1 - (k * w_1) / (2 * w))**2\
     - (w_1**2) / 4 * (1 / w + 1 / 4) + w_2 / 2

    return g
```

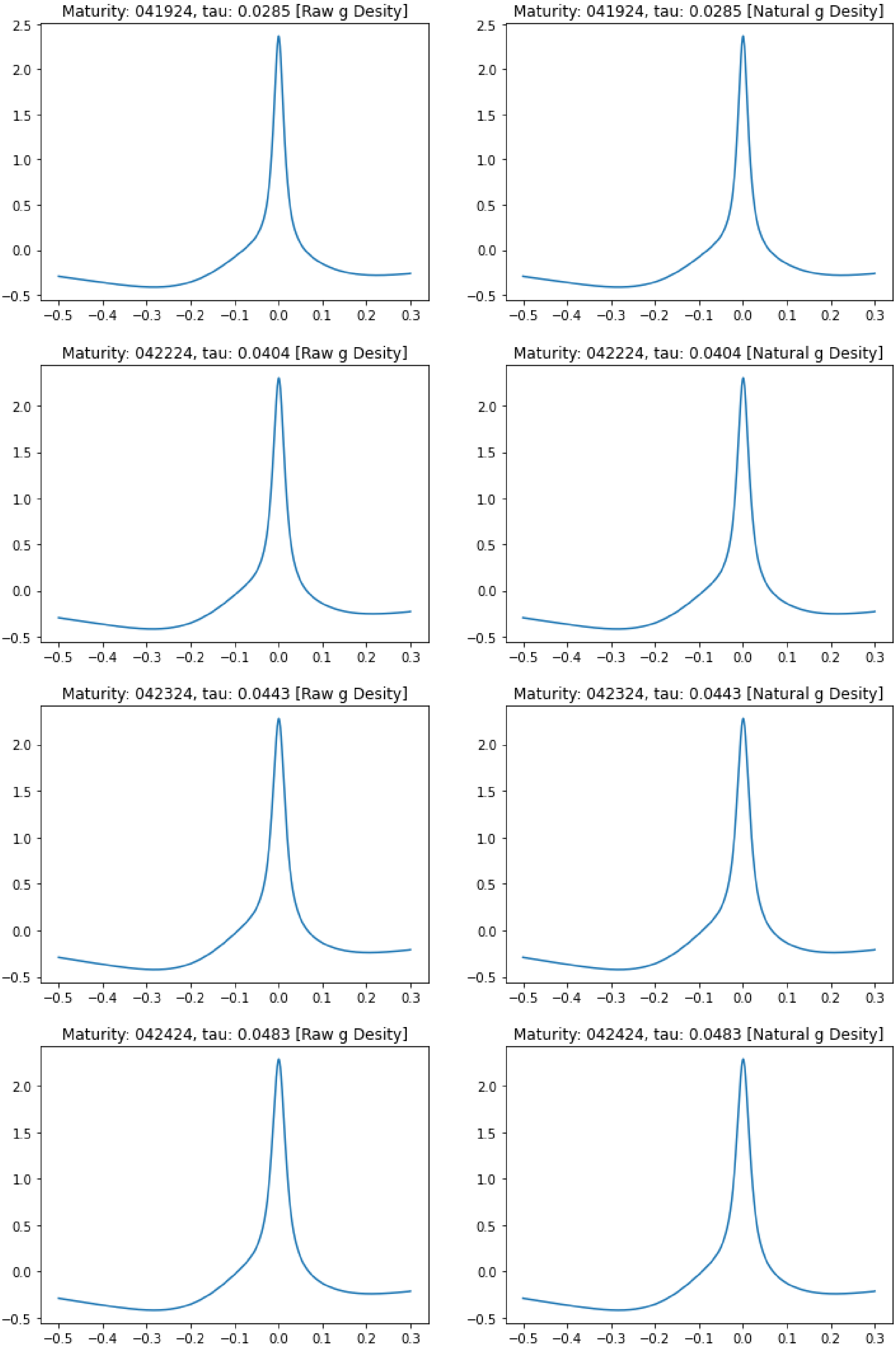
In [17]:
```
def butterfly_arbitrage(option_dict):
    g_dict = {}

    keys = option_dict.keys()
    options_li = sorted(option_dict.items())

    k_arr = np.linspace(-0.5, 0.3, 1000)

    for key, option in options_li:
        df = option[-1]
        tau = option[0]
        raw_params = option[1]
        natural_params = option[2]

        a, b, rho, m, sigma = raw_params
        raw_w = raw(k_arr, a, b, rho, m, sigma)

        Delta, mu, rho, w, zeta = natural_params
        natural_w = natural(k_arr, Delta, mu, rho, w, zeta)

        raw_g = g(k_arr, raw_w)
        natural_g = g(k_arr, natural_w)

        g_dict[key] = {'raw_g': raw_g, 'natural_g': natural_g}

        fig, axs = plt.subplots(1, 2, figsize=(12, 4))

        axs[0].plot(k_arr, raw_g)
        axs[0].set_title(f'Maturity: {key}, tau: {np.round(tau,4)} [Raw g Desity]')

        axs[1].plot(k_arr, natural_g)
        axs[1].set_title(f'Maturity: {key}, tau: {np.round(tau,4)} [Natural g Desity]')

    return g_dict
```

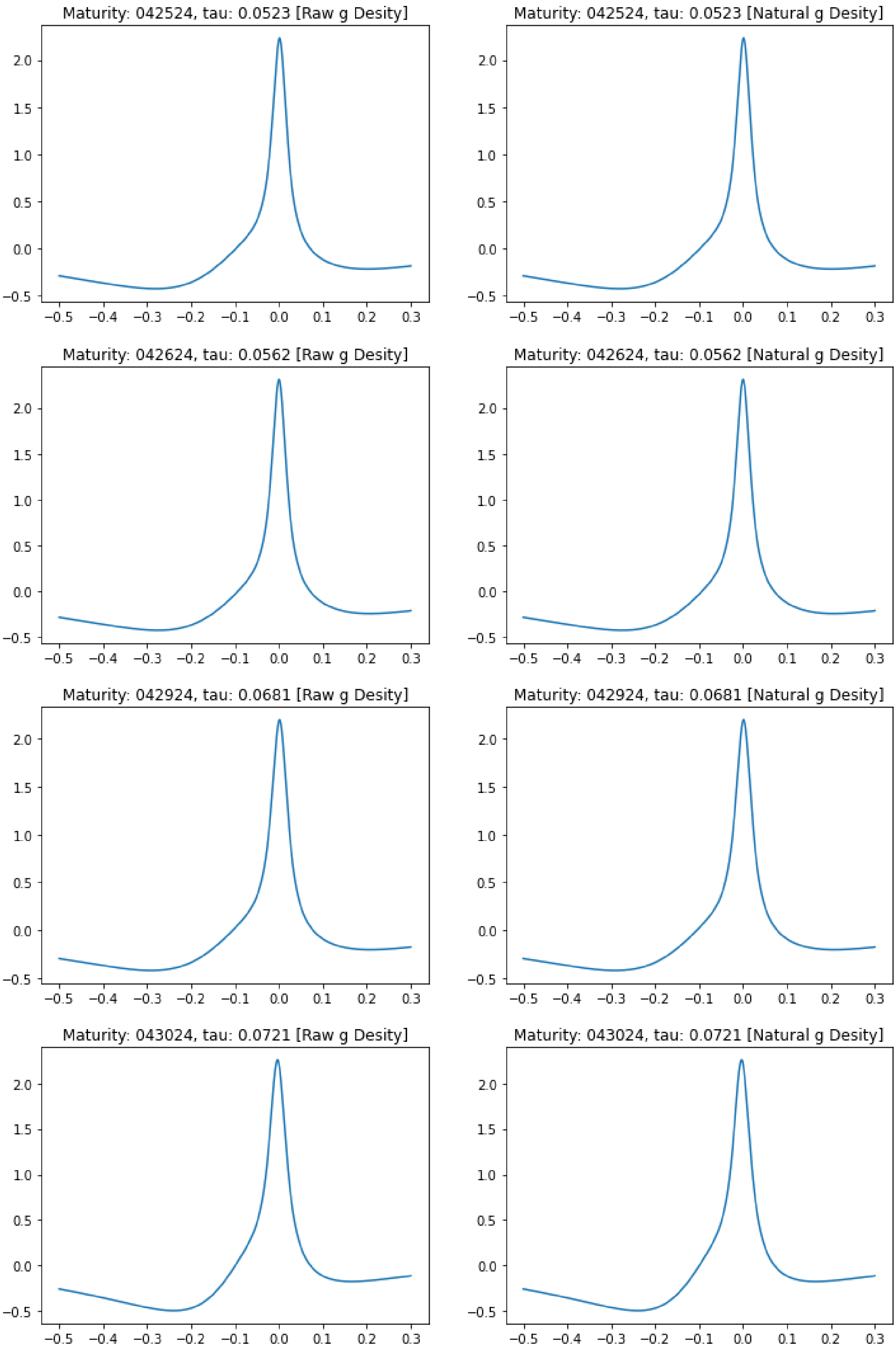
In [18]:
```
g_dict_04 = butterfly_arbitrage(processed_04)
g_dict_05 = butterfly_arbitrage(processed_05)
g_dict_06 = butterfly_arbitrage(processed_06)
g_dict_07 = butterfly_arbitrage(processed_07)
g_dict_08 = butterfly_arbitrage(processed_08)
g_dict_09 = butterfly_arbitrage(processed_09)
```

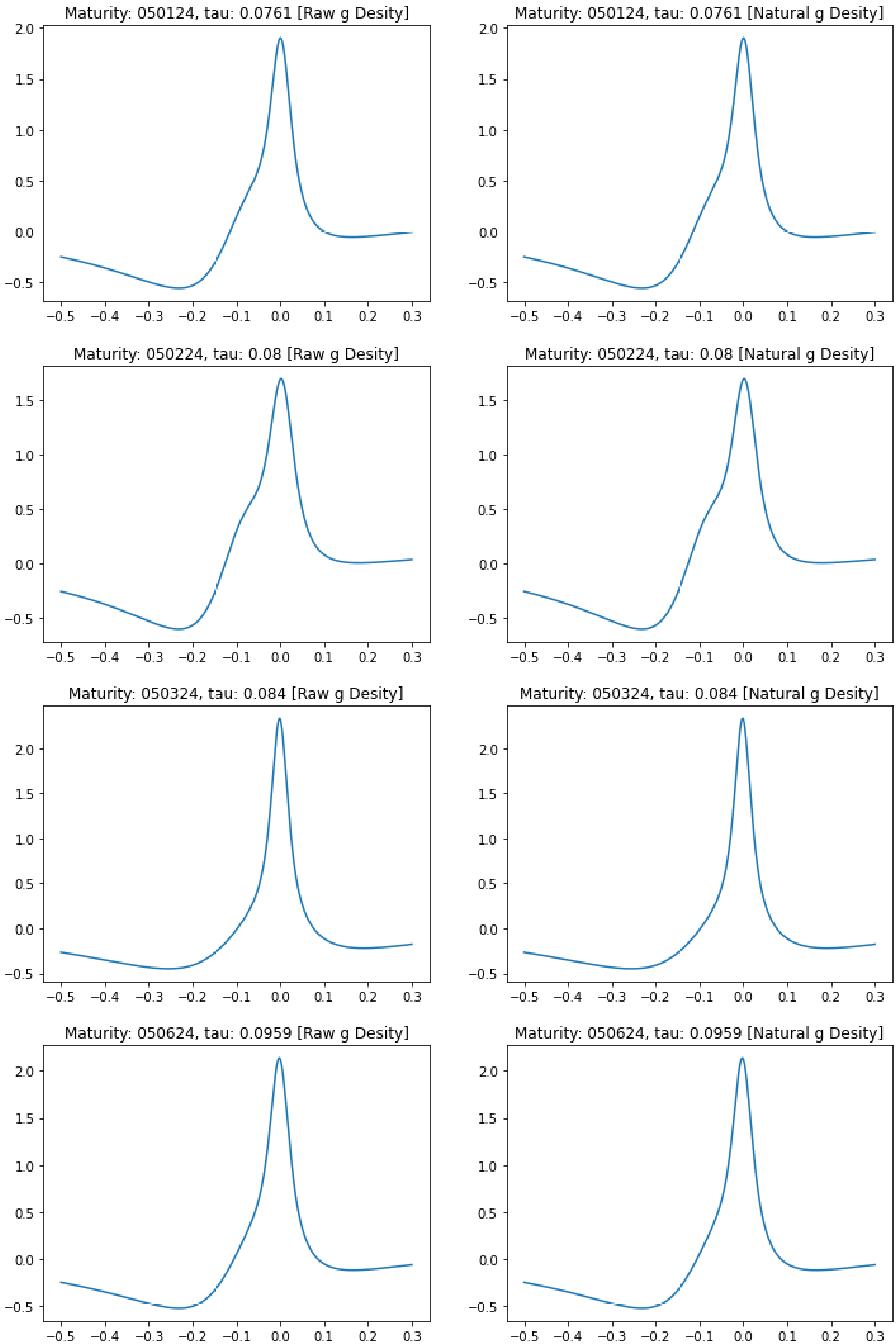
```
C:\Users\kangj\AppData\Local\Temp\ipykernel_34988\3238590254.py:26: RuntimeWarning: More
than 20 figures have been opened. Figures created through the pyplot interface (`matplot
lib.pyplot.figure`) are retained until explicitly closed and may consume too much memor
y. (To control this warning, see the rcParam `figure.max_open_warning`). Consider using
`matplotlib.pyplot.close()`.
  fig, axs = plt.subplots(1, 2, figsize=(12, 4))
```

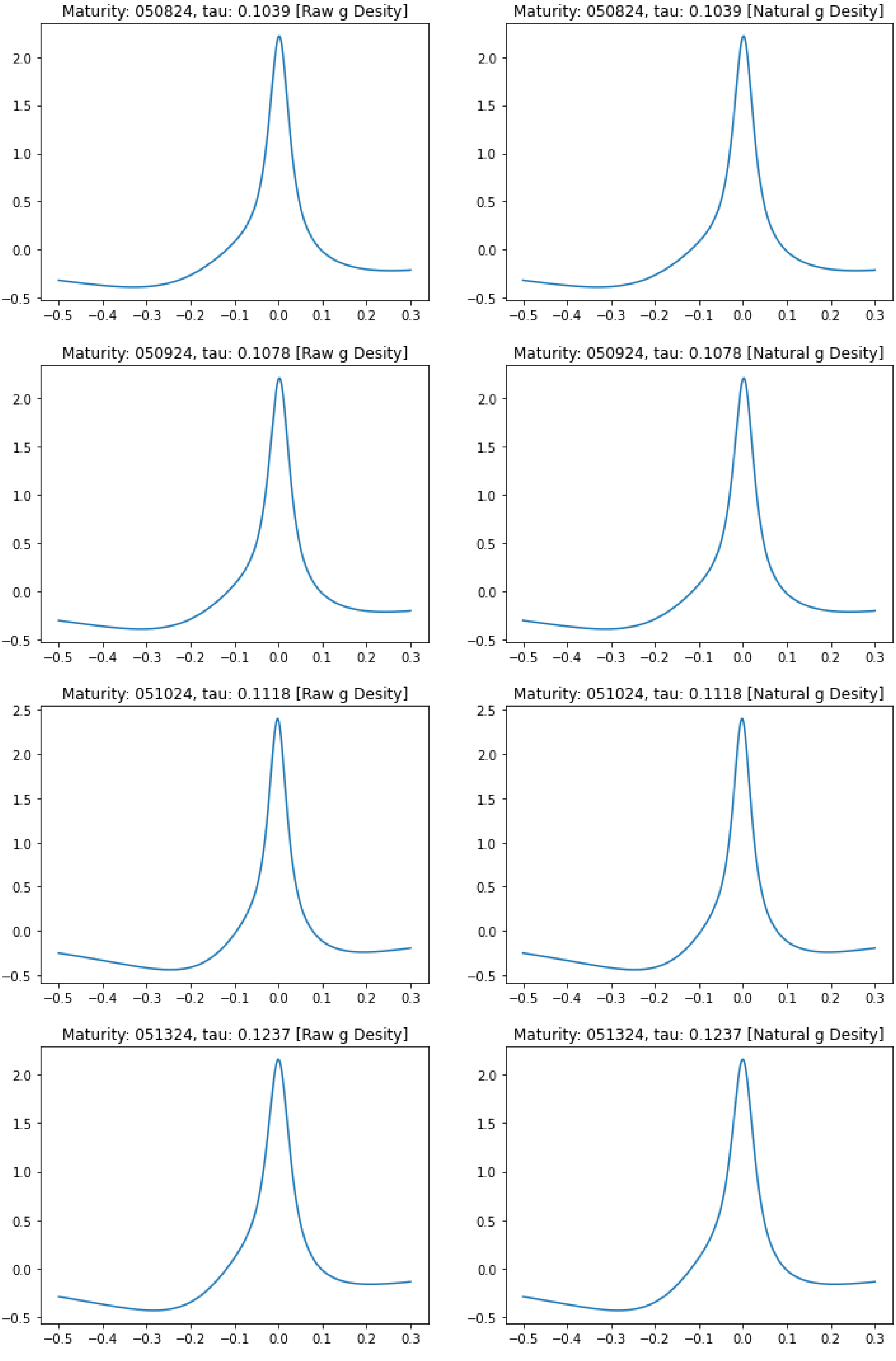



Maturity: 041524, tau: 0.0126 [Raw g Desity]
Maturity: 041524, tau: 0.0126 [Natural g Desity]
Maturity: 041624, tau: 0.0166 [Raw g Desity]
Maturity: 041624, tau: 0.0166 [Natural g Desity]
Maturity: 041724, tau: 0.0205 [Raw g Desity]
Maturity: 041724, tau: 0.0205 [Natural g Desity]
Maturity: 041824, tau: 0.0245 [Raw g Desity]
Maturity: 041824, tau: 0.0245 [Natural g Desity]

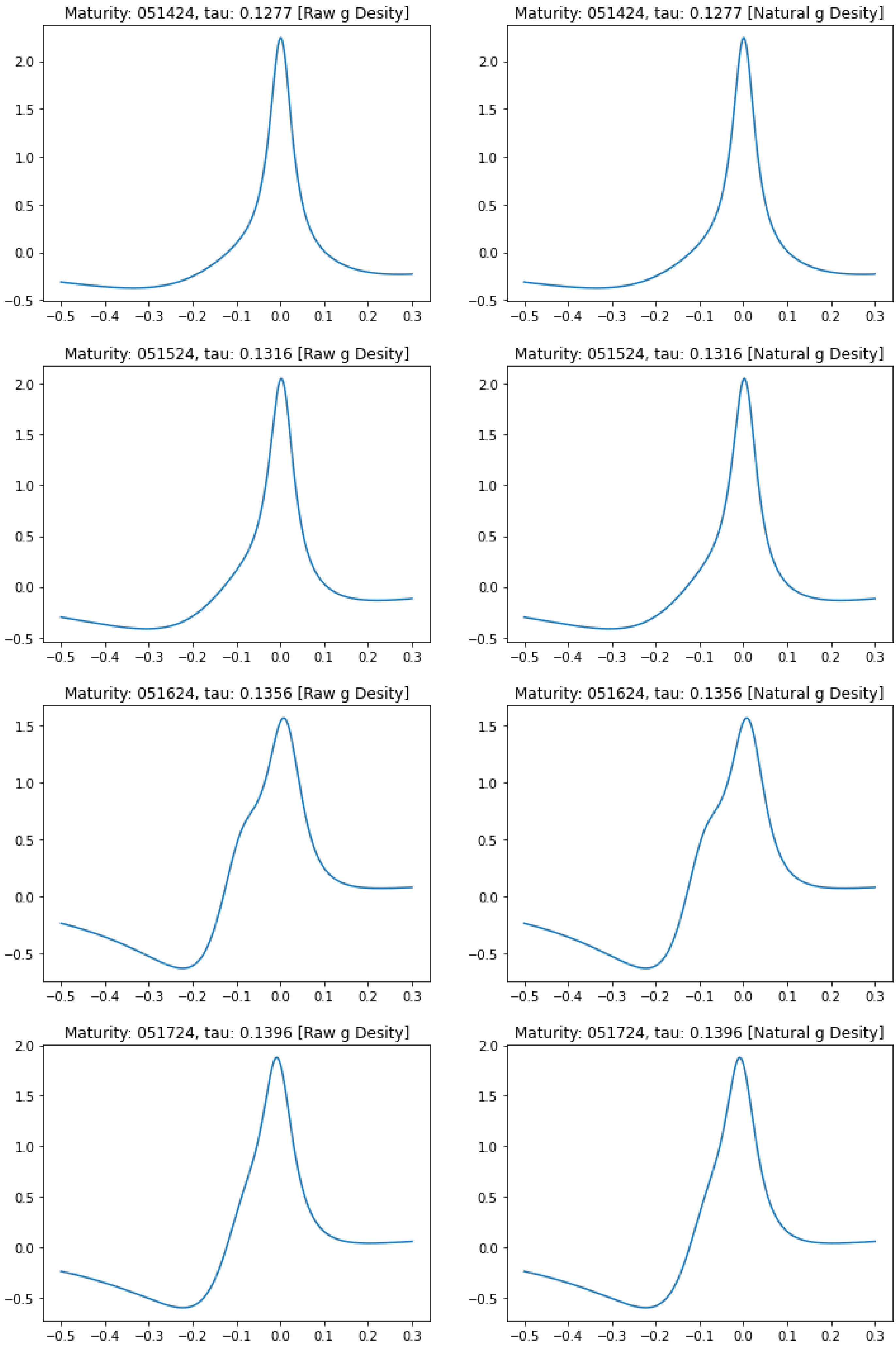

Maturity: 041924, tau: 0.0285 [Raw g Desity]
Maturity: 041924, tau: 0.0285 [Natural g Desity]
Maturity: 042224, tau: 0.0404 [Raw g Desity]
Maturity: 042224, tau: 0.0404 [Natural g Desity]
Maturity: 042324, tau: 0.0443 [Raw g Desity]
Maturity: 042324, tau: 0.0443 [Natural g Desity]
Maturity: 042424, tau: 0.0483 [Raw g Desity]
Maturity: 042424, tau: 0.0483 [Natural g Desity]
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−0.5 −0.4 −0.3 −0.2 −0.1 0.0 0.1 0.2 0.3

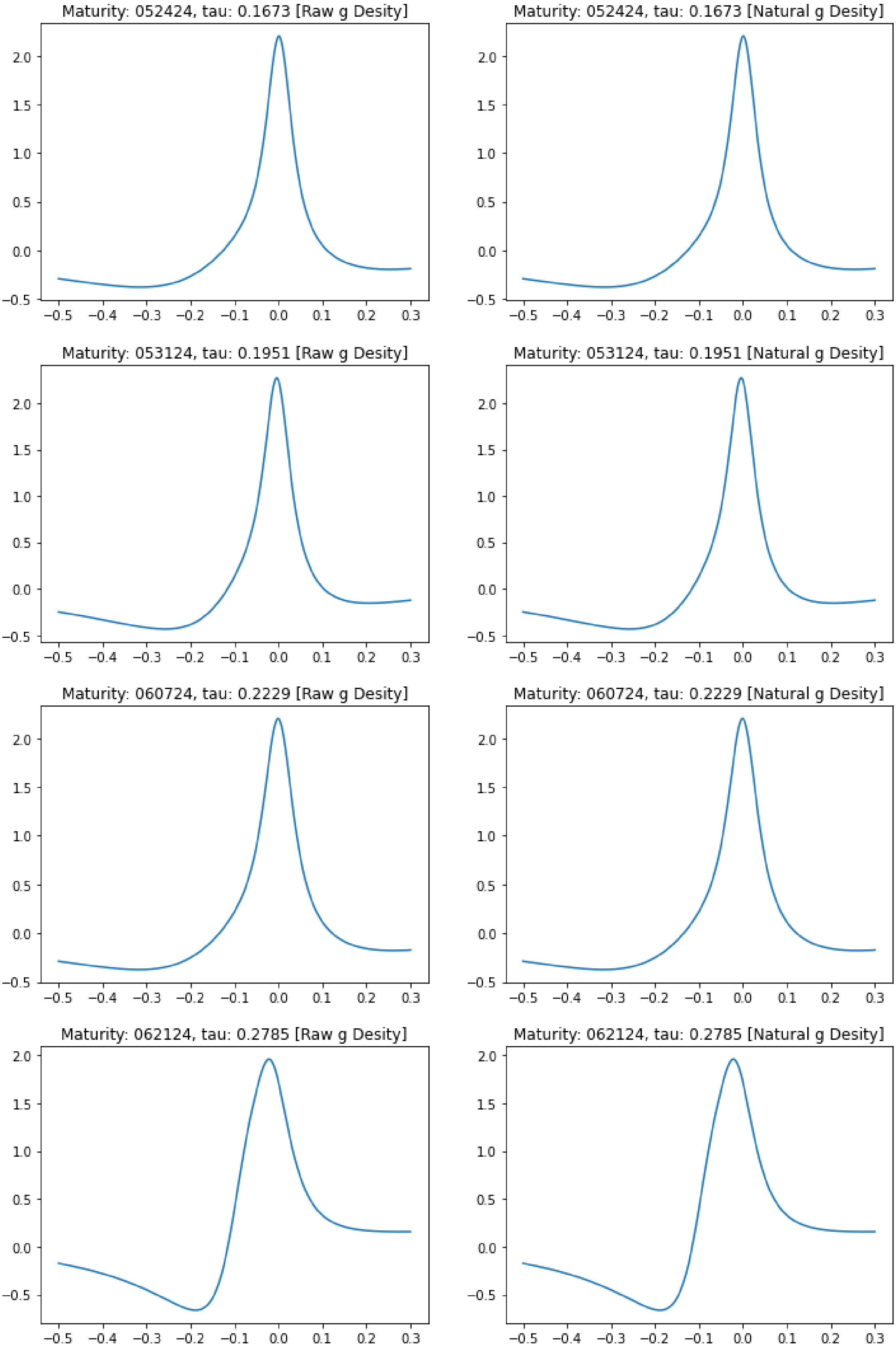

Maturity: 042524, tau: 0.0523 [Raw g Desity]
Maturity: 042524, tau: 0.0523 [Natural g Desity]
Maturity: 042624, tau: 0.0562 [Raw g Desity]
Maturity: 042624, tau: 0.0562 [Natural g Desity]
Maturity: 042924, tau: 0.0681 [Raw g Desity]
Maturity: 042924, tau: 0.0681 [Natural g Desity]
Maturity: 043024, tau: 0.0721 [Raw g Desity]
Maturity: 043024, tau: 0.0721 [Natural g Desity]

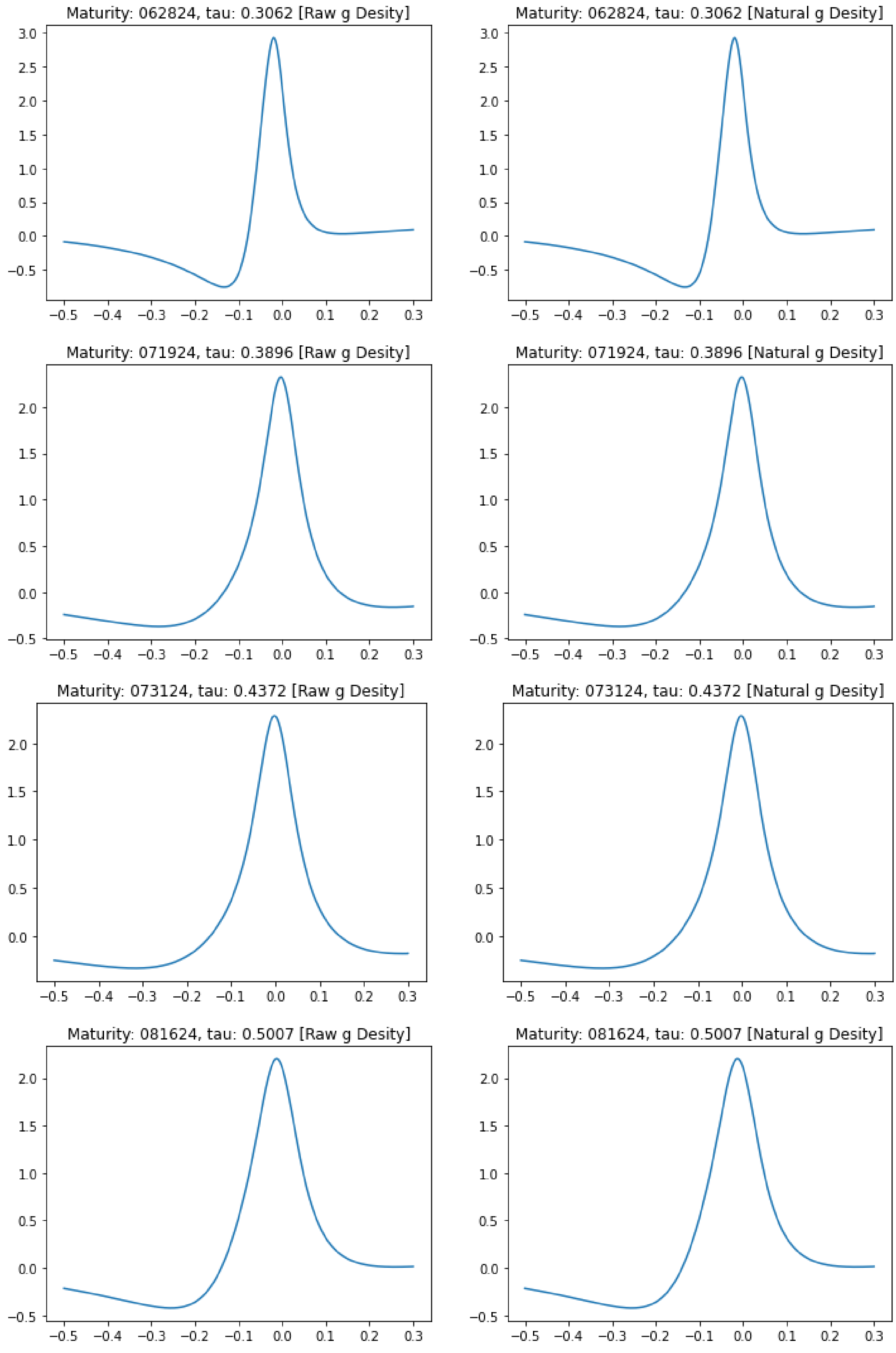

Maturity: 050124, tau: 0.0761 [Raw g Desity]
Maturity: 050124, tau: 0.0761 [Natural g Desity]
Maturity: 050224, tau: 0.08 [Raw g Desity]
Maturity: 050224, tau: 0.08 [Natural g Desity]
Maturity: 050324, tau: 0.084 [Raw g Desity]
Maturity: 050324, tau: 0.084 [Natural g Desity]
Maturity: 050624, tau: 0.0959 [Raw g Desity]
Maturity: 050624, tau: 0.0959 [Natural g Desity]

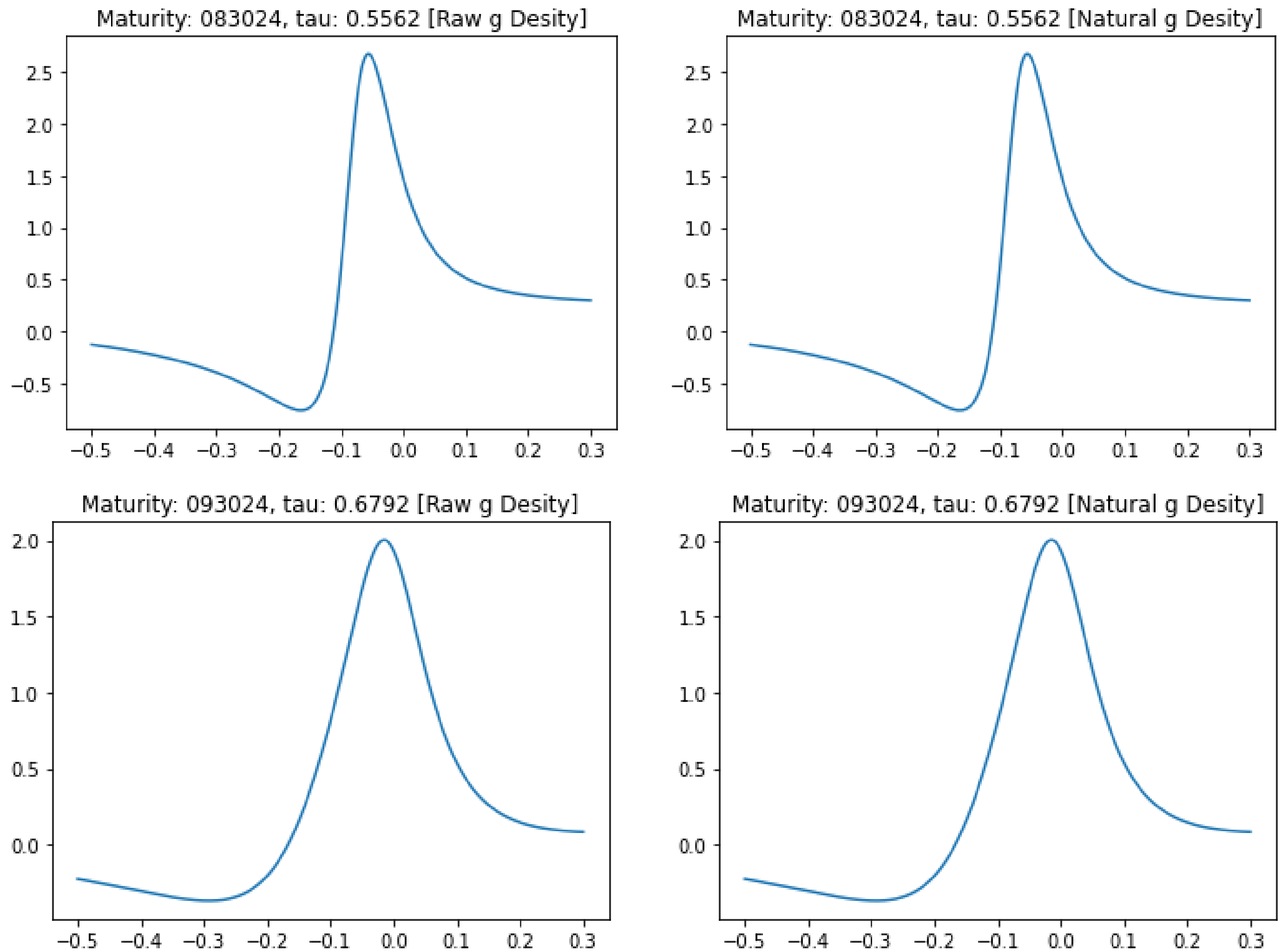


Maturity: 050824, tau: 0.1039 [Raw g Desity]
Maturity: 050824, tau: 0.1039 [Natural g Desity]
Maturity: 050924, tau: 0.1078 [Raw g Desity]
Maturity: 050924, tau: 0.1078 [Natural g Desity]
Maturity: 051024, tau: 0.1118 [Raw g Desity]
Maturity: 051024, tau: 0.1118 [Natural g Desity]
Maturity: 051324, tau: 0.1237 [Raw g Desity]
Maturity: 051324, tau: 0.1237 [Natural g Desity]


Maturity: 051424, tau: 0.1277 [Raw g Desity]
Maturity: 051424, tau: 0.1277 [Natural g Desity]
Maturity: 051524, tau: 0.1316 [Raw g Desity]
Maturity: 051524, tau: 0.1316 [Natural g Desity]
Maturity: 051624, tau: 0.1356 [Raw g Desity]
Maturity: 051624, tau: 0.1356 [Natural g Desity]
Maturity: 051724, tau: 0.1396 [Raw g Desity]
Maturity: 051724, tau: 0.1396 [Natural g Desity]
−0.5 −0.4 −0.3 −0.2 −0.1 0.0 0.1 0.2 0.3
−0.5 0.0 0.5 1.0 1.5 2.0


Maturity: 052424, tau: 0.1673 [Raw g Desity]
Maturity: 052424, tau: 0.1673 [Natural g Desity]
Maturity: 053124, tau: 0.1951 [Raw g Desity]
Maturity: 053124, tau: 0.1951 [Natural g Desity]
Maturity: 060724, tau: 0.2229 [Raw g Desity]
Maturity: 060724, tau: 0.2229 [Natural g Desity]
Maturity: 062124, tau: 0.2785 [Raw g Desity]
Maturity: 062124, tau: 0.2785 [Natural g Desity]


Maturity: 062824, tau: 0.3062 [Raw g Desity]
Maturity: 062824, tau: 0.3062 [Natural g Desity]
Maturity: 071924, tau: 0.3896 [Raw g Desity]
Maturity: 071924, tau: 0.3896 [Natural g Desity]
Maturity: 073124, tau: 0.4372 [Raw g Desity]
Maturity: 073124, tau: 0.4372 [Natural g Desity]
Maturity: 081624, tau: 0.5007 [Raw g Desity]
Maturity: 081624, tau: 0.5007 [Natural g Desity]




It is observable that the density $g$ is negative for negative $\tau$'s. Since $g < 0$ for some $k$, slices are not free of butterfly arbitrage.

In [19]:

```
def column_adder(df):
    cols = df.columns

    minval = np.inf
    for i in range(len(cols)-1):
        minval = min(minval, cols[i+1] - cols[i])

    new_arr = []
    for i in range(len(cols) - 1):
        start = cols[i]
        end = cols[i+1]
        num_values = int((end - start) / minval * 2)
        if num_values > 1:
            interpolated_values = np.linspace(start, end, num_values - 1)[1:-2]
            new_arr.extend(interpolated_values)

    return new_arr

def interpolate_row(row):
    return row.astype(float).interpolate()
```

In [20]:

```
def surface(option_batch_li):
    k_arr =  np.linspace(-0.5, 0.3, 1000)
    raw_surface = pd.DataFrame(index=k_arr)
    natural_surface = pd.DataFrame(index=k_arr)

    for option_batch in option_batch_li:
```

```python
        for key in option_batch.keys():
            df = option_batch[key][-1]
            tau = option_batch[key][0]
            raw_params = option_batch[key][1]
            natural_params = option_batch[key][2]

            a, b, rho, m, sigma = raw_params
            raw_w = raw(k_arr, a, b, rho, m, sigma)

            Delta, mu, rho, w, zeta = natural_params
            natural_w = natural(k_arr, Delta, mu, rho, w, zeta)

            raw_surface[tau] = raw_w
            natural_surface[tau] = natural_w


    raw_surface = raw_surface.sort_index().sort_index(axis=1)
    new_columns = column_adder(raw_surface)
    new_df = pd.DataFrame(columns=new_columns)
    raw_surface = pd.concat([raw_surface, new_df], axis=1)
    raw_surface = raw_surface.sort_index().sort_index(axis=1)
    raw_surface = raw_surface.astype(float).interpolate(axis=1)
    raw_surface = raw_surface.astype(float).interpolate(axis=0)
    raw_surface = raw_surface.astype(float)

    natural_surface = natural_surface.sort_index().sort_index(axis=1)
    new_columns = column_adder(natural_surface)
    new_df = pd.DataFrame(columns=new_columns)
    natural_surface = pd.concat([natural_surface, new_df], axis=1)
    natural_surface = natural_surface.sort_index().sort_index(axis=1)
    natural_surface = natural_surface.astype(float).interpolate(axis=1)
    natural_surface = natural_surface.astype(float).interpolate(axis=0)
    natural_surface = natural_surface.astype(float)

    return raw_surface, natural_surface
```

In [21]:

```python
raw_surface, natural_surface = surface([processed_04, processed_05, processed_06, proces
```

In [22]:

```python
raw_surface
```

Out[22]:

| | 0.000678 | 0.003654 | 0.006630 | 0.012583 | 0.016551 | 0.020519 | 0.024487 | 0.028456 | 0.031432 | 0.03440 |
|---|---|---|---|---|---|---|---|---|---|---|
| **-0.500000** | 0.228676 | 0.229062 | 0.229449 | 0.229835 | 0.226177 | 0.228614 | 0.228624 | 0.227295 | 0.227187 | 0.22708 |
| **-0.499199** | 0.228157 | 0.228541 | 0.228925 | 0.229308 | 0.225669 | 0.228093 | 0.228105 | 0.226782 | 0.226675 | 0.22656 |
| **-0.498398** | 0.227638 | 0.228019 | 0.228400 | 0.228782 | 0.225162 | 0.227573 | 0.227586 | 0.226270 | 0.226162 | 0.22605 |
| **-0.497598** | 0.227119 | 0.227498 | 0.227876 | 0.228255 | 0.224654 | 0.227053 | 0.227066 | 0.225757 | 0.225650 | 0.22554 |
| **-0.496797** | 0.226600 | 0.226976 | 0.227352 | 0.227729 | 0.224147 | 0.226533 | 0.226548 | 0.225244 | 0.225138 | 0.22503 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **0.296797** | 0.086410 | 0.087366 | 0.088321 | 0.089276 | 0.074233 | 0.080924 | 0.079105 | 0.077734 | 0.075469 | 0.07320 |
| **0.297598** | 0.086777 | 0.087738 | 0.088698 | 0.089659 | 0.074537 | 0.081265 | 0.079437 | 0.078056 | 0.075781 | 0.07350 |
| **0.298398** | 0.087144 | 0.088110 | 0.089076 | 0.090043 | 0.074842 | 0.081606 | 0.079769 | 0.078379 | 0.076093 | 0.07380 |
| **0.299199** | 0.087512 | 0.088483 | 0.089455 | 0.090426 | 0.075147 | 0.081947 | 0.080102 | 0.078701 | 0.076405 | 0.07411 |
| **0.300000** | 0.087879 | 0.088856 | 0.089834 | 0.090811 | 0.075452 | 0.082288 | 0.080434 | 0.079024 | 0.076718 | 0.07441 |

1000 rows × 277 columns

In [23]:
```
natural_surface
```

Out[23]:

| | 0.000678 | 0.003654 | 0.006630 | 0.012583 | 0.016551 | 0.020519 | 0.024487 | 0.028456 | 0.031432 | 0.03440 |
|---|---|---|---|---|---|---|---|---|---|---|
| **-0.500000** | 0.228676 | 0.229062 | 0.229449 | 0.229835 | 0.226177 | 0.228614 | 0.228624 | 0.227295 | 0.227187 | 0.22708 |
| **-0.499199** | 0.228157 | 0.228541 | 0.228925 | 0.229308 | 0.225669 | 0.228093 | 0.228105 | 0.226782 | 0.226675 | 0.22656 |
| **-0.498398** | 0.227638 | 0.228019 | 0.228400 | 0.228782 | 0.225162 | 0.227573 | 0.227586 | 0.226270 | 0.226162 | 0.22605 |
| **-0.497598** | 0.227119 | 0.227498 | 0.227876 | 0.228255 | 0.224654 | 0.227053 | 0.227066 | 0.225757 | 0.225650 | 0.22554 |
| **-0.496797** | 0.226600 | 0.226976 | 0.227352 | 0.227729 | 0.224147 | 0.226533 | 0.226548 | 0.225244 | 0.225138 | 0.22503 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **0.296797** | 0.086410 | 0.087366 | 0.088321 | 0.089276 | 0.074233 | 0.080924 | 0.079105 | 0.077734 | 0.075469 | 0.07320 |
| **0.297598** | 0.086777 | 0.087738 | 0.088698 | 0.089659 | 0.074537 | 0.081265 | 0.079437 | 0.078056 | 0.075781 | 0.07350 |
| **0.298398** | 0.087144 | 0.088110 | 0.089076 | 0.090043 | 0.074842 | 0.081606 | 0.079769 | 0.078379 | 0.076093 | 0.07380 |
| **0.299199** | 0.087512 | 0.088483 | 0.089455 | 0.090426 | 0.075147 | 0.081947 | 0.080102 | 0.078701 | 0.076405 | 0.07411 |
| **0.300000** | 0.087879 | 0.088856 | 0.089834 | 0.090811 | 0.075453 | 0.082288 | 0.080434 | 0.079024 | 0.076718 | 0.07441 |

1000 rows × 277 columns

In [24]:
```python
def calendar_spread_arbitrage(surface):
    n = surface.shape[1]
    cols = surface.columns

    cal_df = pd.DataFrame(columns=cols)

    arb =[]

    for i in range(n):
        if i < n-1:
            val = (surface.iloc[:, i+1] - surface.iloc[:, i]) / (cols[i+1]-cols[i])
            if sum(val < 0) > 0:
                arb.append(pd.DataFrame({cols[i]: val[(val < 0)]}))
            else:
                print(f"No arbitrage for {cols[i]}")

            cal_df[cols[i]] = val
        else:
            val = (surface.iloc[:, i-1] - surface.iloc[:, i]) / (cols[i]-cols[i-1])

            if len(val.index[val < 0]) > 0:
                arb.append(pd.DataFrame({cols[i]: val[(val < 0)]}))

            cal_df[cols[i]] = val

    if arb == []:
        print("There is no calendar spread arbitrage.")
    else:
        cnt = 0
        for i in range(len(arb)):
            cnt += len(arb[i])
        print(f"There are {cnt} / {cal_df.shape[0]*cal_df.shape[1]} calendar spread arbi

    return arb, cal_df
```

In [25]:
```python
def brighten(array, x=0.35):
```

```python
    for i in range(array.shape[0]):
        for j in range(array.shape[1]):
            if array[i, j] > 0:
                # Down
                try:
                    array[i-1, j] = min(1, array[i-1, j] + array[i, j] * x)
                except:
                    pass

                # Up
                try:
                    array[i+1, j] = min(1, array[i+1, j] + array[i, j] * x)
                except:
                    pass

                # Right
                try:
                    array[i, j+1] = min(1, array[i, j+1] + array[i, j] * x)
                except:
                    pass

                # Left
                try:
                    array[i, j-1] = min(1, array[i, j-1] + array[i, j] * x)
                except:
                    pass
    return array
```

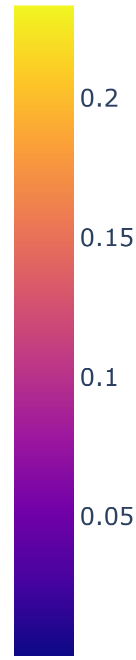
In [26]:
```python
def plot_surface(df, caption):
    # Create meshgrid from DataFrame
    x, y = np.meshgrid(df.index, df.columns)
    z = df.values.T  # Transpose to match x and y dimensions

    arb, cal_df = calendar_spread_arbitrage(df)

    col_map = cal_df.copy()

    col_map[cal_df>0] = 0

    col_map = -col_map

    col_map = (col_map - col_map.min().min()) / (col_map.max().max() - col_map.min().min

    colormap = brighten(brighten(brighten(col_map.values)))

    # Create a colormap using colorscale
    colorscale = [[0, 'green'], [1.0, 'red']]

    # Create a 3D surface plot
    fig = go.Figure(data=[go.Surface(z=z, x=x, y=y)])
    fig.update_layout(title=f'w_t Surface {caption}', scene=dict(
                        xaxis_title='k (log Strike)',
                        yaxis_title='t',
                        zaxis_title='w_t'))
    fig.show()

    fig = go.Figure(data=[go.Surface(z=z, x=x, y=y, surfacecolor=colormap, colorscale=co
    fig.update_layout(title=f'w_t Surface Calendar Arbitrage, neg. derivative=red {capti
                        xaxis_title='k (log Strike)',
                        yaxis_title='t',
                        zaxis_title='w_t'))
    fig.show()
```

In [27]:
```python
import plotly.graph_objects as go
```

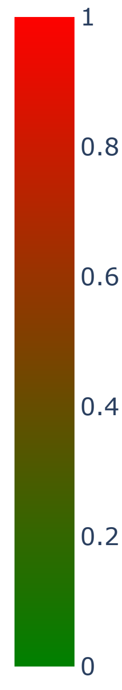
```python
plot_surface(raw_surface, '[raw]')
plot_surface(natural_surface, '[natural]')
```

```
There are 111049 / 277000 calendar spread arbitrage points where \partial_t w(k,t) < 0.
```

w_t Surface [raw]

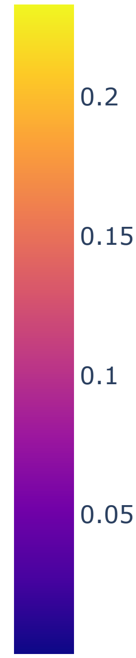



w_t Surface Calendar Arbitrage, neg. derivative=red [raw]

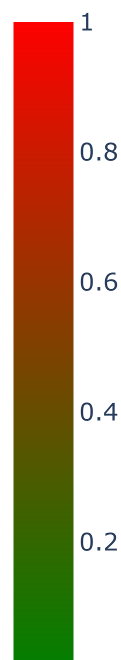


```
There are 111074 / 277000 calendar spread arbitrage points where \partial_t w(k,t) < 0.
```

w_t Surface [natural]




w_t Surface Calendar Arbitrage, neg. derivative=red [natural]




In [28]:
```python
arb, cal_df = calendar_spread_arbitrage(raw_surface)

print(arb[0])
print(cal_df)
```

```
There are 111049 / 277000 calendar spread arbitrage points where \partial_t w(k,t) < 0.
           0.000678
-0.345445 -0.000375
-0.344645 -0.000835
-0.343844 -0.001293
-0.343043 -0.001748
-0.342242 -0.002201
...             ...
-0.059560 -0.000754
-0.058759 -0.000586
-0.057958 -0.000420
-0.057157 -0.000255
-0.056356 -0.000093

[362 rows x 1 columns]
           0.000678  0.003654  0.006630  0.012583  0.016551  0.020519  \
-0.500000  0.129830  0.129830  0.064915 -0.921995  0.614102  0.002663
-0.499199  0.128982  0.128982  0.064491 -0.917130  0.610874  0.002921
-0.498398  0.128134  0.128134  0.064067 -0.912275  0.607652  0.003177
-0.497598  0.127289  0.127289  0.063644 -0.907428  0.604435  0.003434
-0.496797  0.126444  0.126444  0.063222 -0.902590  0.601223  0.003689
...             ...       ...       ...       ...       ...       ...
 0.296797  0.320900  0.320900  0.160450 -3.790769  1.686121 -0.458278
 0.297598  0.322752  0.322752  0.161376 -3.810603  1.695226 -0.460509
 0.298398  0.324607  0.324607  0.162304 -3.830470  1.704348 -0.462742
 0.299199  0.326467  0.326467  0.163233 -3.850370  1.713485 -0.464979
 0.300000  0.328331  0.328331  0.164165 -3.870301  1.722638 -0.467218

           0.024487  0.028456  0.031432  0.034408  ...  0.658747  0.660797  \
-0.500000 -0.334862 -0.036211 -0.036211 -0.018105  ...  0.087479  0.087479
-0.499199 -0.333238 -0.036115 -0.036115 -0.018057  ...  0.086927  0.086927
-0.498398 -0.331615 -0.036019 -0.036019 -0.018009  ...  0.086375  0.086375
-0.497598 -0.329995 -0.035923 -0.035923 -0.017962  ...  0.085823  0.085823
-0.496797 -0.328377 -0.035829 -0.035829 -0.017914  ...  0.085271  0.085271
...             ...       ...       ...       ...  ...       ...       ...
 0.296797 -0.345575 -0.760947 -0.760947 -0.380473  ...  0.133373  0.133373
 0.297598 -0.348011 -0.764429 -0.764429 -0.382215  ...  0.134005  0.134005
 0.298398 -0.350452 -0.767916 -0.767916 -0.383958  ...  0.134638  0.134638
 0.299199 -0.352899 -0.771406 -0.771406 -0.385703  ...  0.135271  0.135271
 0.300000 -0.355352 -0.774900 -0.774900 -0.387450  ...  0.135905  0.135905

           0.662847  0.664897  0.666948  0.668998  0.671048  0.673099  \
-0.500000  0.087479  0.087479  0.087479  0.087479  0.087479  0.087479
-0.499199  0.086927  0.086927  0.086927  0.086927  0.086927  0.086927
-0.498398  0.086375  0.086375  0.086375  0.086375  0.086375  0.086375
-0.497598  0.085823  0.085823  0.085823  0.085823  0.085823  0.085823
-0.496797  0.085271  0.085271  0.085271  0.085271  0.085271  0.085271
...             ...       ...       ...       ...       ...       ...
 0.296797  0.133373  0.133373  0.133373  0.133373  0.133373  0.133373
 0.297598  0.134005  0.134005  0.134005  0.134005  0.134005  0.134005
 0.298398  0.134638  0.134638  0.134638  0.134638  0.134638  0.134638
 0.299199  0.135271  0.135271  0.135271  0.135271  0.135271  0.135271
 0.300000  0.135905  0.135905  0.135905  0.135905  0.135905  0.135905
```

```
           0.675149  0.679249
-0.500000  0.043740 -0.043740
-0.499199  0.043463 -0.043463
-0.498398  0.043187 -0.043187
-0.497598  0.042911 -0.042911
-0.496797  0.042636 -0.042636
...             ...       ...
 0.296797  0.066687 -0.066687
 0.297598  0.067003 -0.067003
 0.298398  0.067319 -0.067319
 0.299199  0.067636 -0.067636
 0.300000  0.067953 -0.067953

[1000 rows x 277 columns]
```

In [29]:
```
arb, cal_df = calendar_spread_arbitrage(natural_surface)

print(arb[0])
print(cal_df)
```

```
There are 111074 / 277000 calendar spread arbitrage points where \partial_t w(k,t) < 0.
           0.000678
-0.345445 -0.000375
-0.344645 -0.000835
-0.343844 -0.001293
-0.343043 -0.001748
-0.342242 -0.002201
...             ...
-0.059560 -0.000754
-0.058759 -0.000586
-0.057958 -0.000420
-0.057157 -0.000255
-0.056356 -0.000093

[362 rows x 1 columns]
           0.000678  0.003654  0.006630  0.012583  0.016551  0.020519  \
-0.500000  0.129830  0.129830  0.064915 -0.921994  0.614102  0.002663
-0.499199  0.128981  0.128981  0.064491 -0.917130  0.610874  0.002921
-0.498398  0.128134  0.128134  0.064067 -0.912274  0.607652  0.003178
-0.497598  0.127288  0.127288  0.063644 -0.907427  0.604435  0.003434
-0.496797  0.126444  0.126444  0.063222 -0.902589  0.601223  0.003689
...             ...       ...       ...       ...       ...       ...
 0.296797  0.320900  0.320900  0.160450 -3.790765  1.686117 -0.458277
 0.297598  0.322752  0.322752  0.161376 -3.810599  1.695222 -0.460507
 0.298398  0.324607  0.324607  0.162304 -3.830466  1.704344 -0.462741
 0.299199  0.326467  0.326467  0.163233 -3.850365  1.713481 -0.464977
 0.300000  0.328330  0.328330  0.164165 -3.870297  1.722634 -0.467217

           0.024487  0.028456  0.031432  0.034408  ...  0.658747  0.660797  \
-0.500000 -0.334862 -0.036211 -0.036211 -0.018106  ...  0.087492  0.087492
-0.499199 -0.333237 -0.036115 -0.036115 -0.018057  ...  0.086940  0.086940
-0.498398 -0.331615 -0.036019 -0.036019 -0.018009  ...  0.086387  0.086387
-0.497598 -0.329995 -0.035924 -0.035924 -0.017962  ...  0.085836  0.085836
-0.496797 -0.328377 -0.035829 -0.035829 -0.017914  ...  0.085284  0.085284
...             ...       ...       ...       ...  ...       ...       ...
 0.296797 -0.345577 -0.760947 -0.760947 -0.380473  ...  0.133335  0.133335
 0.297598 -0.348013 -0.764429 -0.764429 -0.382215  ...  0.133967  0.133967
 0.298398 -0.350455 -0.767916 -0.767916 -0.383958  ...  0.134599  0.134599
 0.299199 -0.352902 -0.771406 -0.771406 -0.385703  ...  0.135232  0.135232
 0.300000 -0.355354 -0.774900 -0.774900 -0.387450  ...  0.135866  0.135866

           0.662847  0.664897  0.666948  0.668998  0.671048  0.673099  \
-0.500000  0.087492  0.087492  0.087492  0.087492  0.087492  0.087492
-0.499199  0.086940  0.086940  0.086940  0.086940  0.086940  0.086940
-0.498398  0.086387  0.086387  0.086387  0.086387  0.086387  0.086387
```

```
-0.497598  0.085836  0.085836  0.085836  0.085836  0.085836  0.085836
-0.496797  0.085284  0.085284  0.085284  0.085284  0.085284  0.085284
...             ...       ...       ...       ...       ...       ...
 0.296797  0.133335  0.133335  0.133335  0.133335  0.133335  0.133335
 0.297598  0.133967  0.133967  0.133967  0.133967  0.133967  0.133967
 0.298398  0.134599  0.134599  0.134599  0.134599  0.134599  0.134599
 0.299199  0.135232  0.135232  0.135232  0.135232  0.135232  0.135232
 0.300000  0.135866  0.135866  0.135866  0.135866  0.135866  0.135866

           0.675149  0.679249
-0.500000  0.043746 -0.043746
-0.499199  0.043470 -0.043470
-0.498398  0.043194 -0.043194
-0.497598  0.042918 -0.042918
-0.496797  0.042642 -0.042642
...             ...       ...
 0.296797  0.066667 -0.066667
 0.297598  0.066983 -0.066983
 0.298398  0.067300 -0.067300
 0.299199  0.067616 -0.067616
 0.300000  0.067933 -0.067933

[1000 rows x 277 columns]
```

There are some points where $\partial_t w(k,t) < 0$. This indicates that there are calendar spread arbitrage opportunities.

In [ ]: