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

from mpl_toolkits.mplot3d import Axes3D
```

In [ ]:
```
import numpy as np

from scipy.stats import norm

from collections.abc import Callable

def root_bisection(func:Callable, a:float, b:float, epsilon:float=1e-6, log_
    """
    Use the bisection method to find the root of a continuous function
    @param func: Objective function
    @param a: Initial bound (a and b must bracket a root, that is f(a)f(b) <
    @param b: Initial bound (a and b must bracket a root, that is f(a)f(b) <
    @param epsilon: convergence tolerance (>0)
    """
    fa = func(a)
    fb = func(b)

    if fa * fb > 0:
        # print(f"Warning: Supplied bounds {a} and {b} do not bracket a root
        if log_iter:
            return np.nan, np.nan
        else:
            return np.nan

    i = 0

    while abs(a - b) > epsilon:
        i += 1
        mid = (a + b) / 2
        fmid = func(mid)

        if fmid == 0:
            a = mid
            break
        elif fa * fmid < 0:
            fb = fmid
            b = mid
        else:
            fa = fmid
            a = mid

    if log_iter:
        return a, i
    else:
        return a


def root_newton(func:Callable, derivative:Callable, x:float, epsilon:float=1
```

```
    """
    Newton method for root-finding of continuous, differentiable function
    @param func: objective function
    @param derivative: derivative of objective function
    @param x: initial guess
    @param epsilon: convergence tolerance
    @param max_iter: max iterations before giving up
    @param log_iter: Log/output number of iterations to converge
    """

    for i in range(max_iter):
        fx = func(x)

        if abs(fx) < epsilon:
            if log_iter:
                return x, i + 1
            else:
                return x

        fpx = derivative(x)

        x = x - fx / fpx

    if log_iter:
        return np.nan, np.nan
    else:
        return np.nan


if __name__ == "__main__":
    """
    Brief tests for each function
    """
    print(root_bisection(lambda x: (x-2)**3, 0, 5))
    print(root_newton(lambda x: (x-2)**3, lambda x: 3 * (x-2) ** 2, 5))

class BlackScholes:
    @staticmethod
    def _d1_d2(S, K, t, r, sigma):
        """
        d1 and d2 helper function
        """
        d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * t) / (sigma * np.sqrt(t
        d2 = d1 - sigma * np.sqrt(t)
        return d1, d2

    @staticmethod
    def call(S, K, t, r, sigma):
        d1, d2 = BlackScholes._d1_d2(S, K, t, r, sigma)
        return S * norm.cdf(d1) - K * np.exp(-r * t) * norm.cdf(d2)

    @staticmethod
    def put(S, K, t, r, sigma):
        d1, d2 = BlackScholes._d1_d2(S, K, t, r, sigma)
        return K * np.exp(-r * t) * norm.cdf(-d2) - S * norm.cdf(-d1)
```

```
    @staticmethod
    def delta_call(S, K, t, r, sigma):
        d1, _ = BlackScholes._d1_d2(S, K, t, r, sigma)
        return norm.cdf(d1)

    @staticmethod
    def delta_put(S, K, t, r, sigma):
        d1, _ = BlackScholes._d1_d2(S, K, t, r, sigma)
        return norm.cdf(d1) - 1

    @staticmethod
    def vega(S, K, t, r, sigma):
        d1, _ = BlackScholes._d1_d2(S, K, t, r, sigma)
        return S * norm.pdf(d1) * np.sqrt(t)

    @staticmethod
    def gamma(S, K, t, r, sigma):
        d1, _ = BlackScholes._d1_d2(S, K, t, r, sigma)
        return norm.pdf(d1) / (S * sigma * np.sqrt(t))

    # finite difference approximations of the greeks
    @staticmethod
    def delta_call_fd(S, K, t, r, sigma, h=0.001):
        up = BlackScholes.call(S + h, K, t, r, sigma)
        down = BlackScholes.call(S - h, K, t, r, sigma)
        return (up - down) / (2 * h)

    @staticmethod
    def delta_put_fd(S, K, t, r, sigma, h=0.001):
        up = BlackScholes.put(S + h, K, t, r, sigma)
        down = BlackScholes.put(S - h, K, t, r, sigma)
        return (up - down) / (2 * h)

    @staticmethod
    def gamma_fd(S, K, t, r, sigma, h=0.001):
        delta_up = BlackScholes.delta_call_fd(S + h, K, t, r, sigma)
        delta_down = BlackScholes.delta_call_fd(S - h, K, t, r, sigma)
        return (delta_up - delta_down) / (h * 2)

    @staticmethod
    def vega_fd(S, K, t, r, sigma, h=0.0001):
        up = BlackScholes.call(S, K, t, r, sigma + h)
        down = BlackScholes.call(S, K, t, r, sigma - h)
        return (up - down) / (2 * h)


    @staticmethod
    def iv_call_bisection(S, K, t, r, mkt_price, log_iter=False):
        # create objective function to find root
        def call_objective(sigma):
            return BlackScholes.call(S, K, t, r, sigma) - mkt_price

        # providing conservative bounds to ensure vol is bracketed
        return root_bisection(call_objective, 0.000001, 20, log_iter=log_ite
```

```python
    @staticmethod
    def iv_put_bisection(S, K, t, r, mkt_price, log_iter=False):
        # create objective function to find root
        def put_objective(sigma):
            return BlackScholes.put(S, K, t, r, sigma) - mkt_price

        # providing conservative bounds to ensure vol is bracketed
        return root_bisection(put_objective, 0.000001, 20, log_iter=log_iter


    @staticmethod
    def iv_call_newton(S, K, t, r, mkt_price, log_iter=False):
        def call_objective(sigma):
            return BlackScholes.call(S, K, t, r, sigma) - mkt_price

        def call_derivative(sigma):
            return BlackScholes.vega(S, K, t, r, sigma)

        return root_newton(call_objective, call_derivative, 1, log_iter=log_


    @staticmethod
    def iv_put_newton(S, K, t, r, mkt_price, log_iter=False):
        def put_objective(sigma):
            return BlackScholes.put(S, K, t, r, sigma) - mkt_price

        def put_derivative(sigma):
            return BlackScholes.vega(S, K, t, r, sigma)

        return root_newton(put_objective, put_derivative, 1, log_iter=log_it



if __name__ == "__main__":
    print(BlackScholes.iv_call_bisection(100, 100, 1, 0.05, 10.45, log_iter=
    print(BlackScholes.iv_put_bisection(100, 100, 1, 0.05, 5.57))
    print(BlackScholes.iv_call_newton(100, 100, 1, 0.05, 10.45))
    print(BlackScholes.iv_put_newton(100, 100, 1, 0.05, 5.57))
```

FE621 Homework 1

Cian Gahan

NOTE: When setting up and running my data gathering script, I misread the assignment instructions and downloaded option/price data for the SPX index rather than the SPY ETF. I did not realize this until after Thursday's trading hours and therefore could not switch out the data for the correct ticker. I anticipate the results will be very similar. Vol/open interest relative to the contract size being 10x as much seems in line with SPY on yahoo finance when I checked, potentially smaller. I would anticipate liquidity is at worst similar since the granularity of contracts is higher on SPX (every 5 index points at the money, where SPY trades around 1/10th the value and has every $1-spaced options at the money), and SPY mostly does a good job tracking SPX so I don't think this will have a major impact on analysis or results.

Moving forward in the assignment I will use SPX in place of SPY.

Part 1: Data Gathering Component

1.1: Data Gathering Function & Bonus

- See `scripts/hw1_yf_ingest.py` for raw ingestion
- See `scripts/hw1_yf_ingest_cleaner.py` for cleaning/combination

1.2:

- Why do additional maturities exist? Given that TSLA, SPX, and VIX are all highly liquid, popular tickers with a lot of speculative interest, traders have demand for products that speculate on volatility/short term movements with finer control over the time window than the traditional monthly schedule, especially as the maturity date approaches. Therefore exchanges/market makers introduce options with more maturities as these dates approach, meeting demand without fragmenting long-term maturity liquidity (eg a month or two out, weekly options may be introduced, and a week or so out, daily options may be introduced in the case of SPX).

In [2]:
```
# load in options and price data
options = pd.read_csv("../data/cleaned/options.csv", index_col=0)
print(options.head())
```

```
        contractSymbol  strike   bid   ask optionType  expiration underlying  \
0  VIX260218C00010000    10.0  9.75  10.4       call  2026-02-18       ^VIX
1  VIX260218C00010500    10.5  9.25   9.9       call  2026-02-18       ^VIX
2  VIX260218C00011000    11.0  8.75   9.4       call  2026-02-18       ^VIX
3  VIX260218C00011500    11.5  8.25   8.9       call  2026-02-18       ^VIX
4  VIX260218C00012000    12.0  7.75   8.4       call  2026-02-18       ^VIX

    data_date  daysToMaturity  underlyingPrice  fedFunds
0  2026-02-12               6        19.879999    0.0364
1  2026-02-12               6        19.879999    0.0364
2  2026-02-12               6        19.879999    0.0364
3  2026-02-12               6        19.879999    0.0364
4  2026-02-12               6        19.879999    0.0364
```

1.3:

- SPX: Index published by S&P consisting of a market-cap weighted average of ~500 of the largest companies trading in the US stock market. Purpose is to provide a measure of overall US stock market performance

- SPY: ETF that tracks the SPX index, managed by State Street. Purpose is to gain easy exposure to the entire US stock market
- VIX: Index published by CBOE that estimates 30 day market-implied volatility using a weighted combination of implied volatilities calculated from out-of-the-money SPX call and put options between 23 and 37 days to maturity. Purpose is to provide a measure for expected volatility and allow for trading/speculation on it directly
- TSLA: Tesla Stock. Purpose is to invest in Tesla
- Example option symbol decomposition:
  - TICKERYYMMDD{C/P}STRIKE (strike listed as XXXXX.XXX fixed-point)
  - eg TSLA260220C00450000 - tesla call maturing on Feb 20th with strike $450.
  - Note on SPX options: Options may be listed with tickers as SPX or SPXW in raw data. SPXW are typically found in the finer-grain weekly maturities but may also exist at certain prices in the traditional maturity dates. The mechanical difference is that SPXW expire/are settled using VWAP near the end of the trading day on Friday, while SPX expire at the opening auction on Friday. To simplify analysis I removed this distinction while cleaning the data and am measuring time to maturity as the number of days

1.4:

- For short term interest rate I used the Fed Funds (effective) rate from the Fed website which was 3.64% for both days in my data

Part 2: Analysis of the Data

2.5:

- Black-Scholes Implementation: See `FE621/pricing/black_scholes.py`

2.6:

- Bisection Method: See `FE621/utils.py`
- Note: I found that implied volatility would not converge for much of the options (could not find clear pattern and option quotes seemed off from intrinsic value when checked so presumably stale quotes/data issue) so I replaced non converging values with nan here

In [3]:
```python
def row_iv(row):
    if row["optionType"] == "call":
        return BlackScholes.iv_call_bisection(
            row["underlyingPrice"],
            row["strike"],
            row["daysToMaturity"] / 365,
            np.log1p(row["fedFunds"]),
            (row["bid"] + row["ask"]) / 2
```

```
        )
    else:
        return BlackScholes.iv_put_bisection(
            row["underlyingPrice"],
            row["strike"],
            row["daysToMaturity"] / 365,
            np.log1p(row["fedFunds"]),
            (row["bid"] + row["ask"]) / 2
        )

options["impliedVolatilityB"] = options.apply(row_iv, axis=1)
```

In [4]:
```
options["moneyness"] = options["underlyingPrice"] / options["strike"]
options["abs_moneyness"] = (options["underlyingPrice"] - options["strike"]).

def timeval(row):
    if row["optionType"] == "call":
        return (row["bid"] + row["ask"]) / 2 - np.max(row["underlyingPrice"]
    else:
        return (row["bid"] + row["ask"]) / 2 - np.max(row["strike"] - row["u

options["timeval"] = options.apply(timeval, axis=1)

data1 = options[options["data_date"] == "2026-02-12"]

spx_first = data1[(data1["underlying"] == "^SPX") & (data1["expiration"] == 
tsla_first = data1[(data1["underlying"] == "TSLA") & (data1["expiration"] ==

spx_atm = spx_first[spx_first["abs_moneyness"] == spx_first["abs_moneyness"]
tsla_atm = tsla_first[tsla_first["abs_moneyness"] == tsla_first["abs_moneyne

spx_range_atm = spx_first[(spx_first["moneyness"] >= 0.95) & (spx_first["mon
tsla_range_atm = tsla_first[(tsla_first["moneyness"] >= 0.95) & (tsla_first[

spx_atm_iv = spx_atm["impliedVolatilityB"].mean()
tsla_atm_iv = tsla_atm["impliedVolatilityB"].mean()

spx_range_atm_iv = spx_range_atm["impliedVolatilityB"].mean()
tsla_range_atm_iv = tsla_range_atm["impliedVolatilityB"].mean()

print("SPX at the money: " + str(spx_atm_iv))
print("SPX around the money: " + str(spx_range_atm_iv))
print("TSLA: " + str(tsla_atm_iv))
print("TSLA around the money: " + str(tsla_range_atm_iv))
```

```
SPX at the money: 0.16649494394962489
SPX around the money: 0.16960305650260124
TSLA: 0.41606047321558
TSLA around the money: 0.42085042494567787
```

2.7:

- Newton Method: See `FE621/utils.py`

In [5]:
```
def row_iv_newton(row):
    if row["optionType"] == "call":
        return BlackScholes.iv_call_newton(
            row["underlyingPrice"],
            row["strike"],
            row["daysToMaturity"] / 365,
            np.log1p(row["fedFunds"]),
            (row["bid"] + row["ask"]) / 2
        )
    else:
        return BlackScholes.iv_put_newton(
            row["underlyingPrice"],
            row["strike"],
            row["daysToMaturity"] / 365,
            np.log1p(row["fedFunds"]),
            (row["bid"] + row["ask"]) / 2
        )

options["impliedVolatilityN"] = options.apply(row_iv_newton, axis=1)
```

```
/Users/ciangahan/Documents/Stevens/FE621/FE621/utils.py:68: RuntimeWarning: 
divide by zero encountered in scalar divide
  x = x - fx / fpx
/Users/ciangahan/Documents/Stevens/FE621/FE621/pricing/black_scholes.py:13: 
RuntimeWarning: invalid value encountered in scalar divide
  d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * t) / (sigma * np.sqrt(t))
/Users/ciangahan/Documents/Stevens/FE621/FE621/pricing/black_scholes.py:13: 
RuntimeWarning: overflow encountered in scalar power
  d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * t) / (sigma * np.sqrt(t))
/Users/ciangahan/Documents/Stevens/FE621/FE621/utils.py:68: RuntimeWarning: 
overflow encountered in scalar divide
  x = x - fx / fpx
```

In [6]:
```
print("Maximum ivol difference (Bisection vs Newton): " + str((options["impl
```

```
Maximum ivol difference (Bisection vs Newton): 1.441229731091731e-05
```

We can see that the implied volatility converges to essentially the same values for both newton and bisection methods

In [7]:
```
def iter_bisection(row):
    if row["optionType"] == "call":
        _, i = BlackScholes.iv_call_bisection(
            row["underlyingPrice"],
            row["strike"],
            row["daysToMaturity"] / 365,
            np.log1p(row["fedFunds"]),
            (row["bid"] + row["ask"]) / 2,
            log_iter = True
        )
        return i
    else:
        _, i = BlackScholes.iv_put_bisection(
            row["underlyingPrice"],
            row["strike"],
```

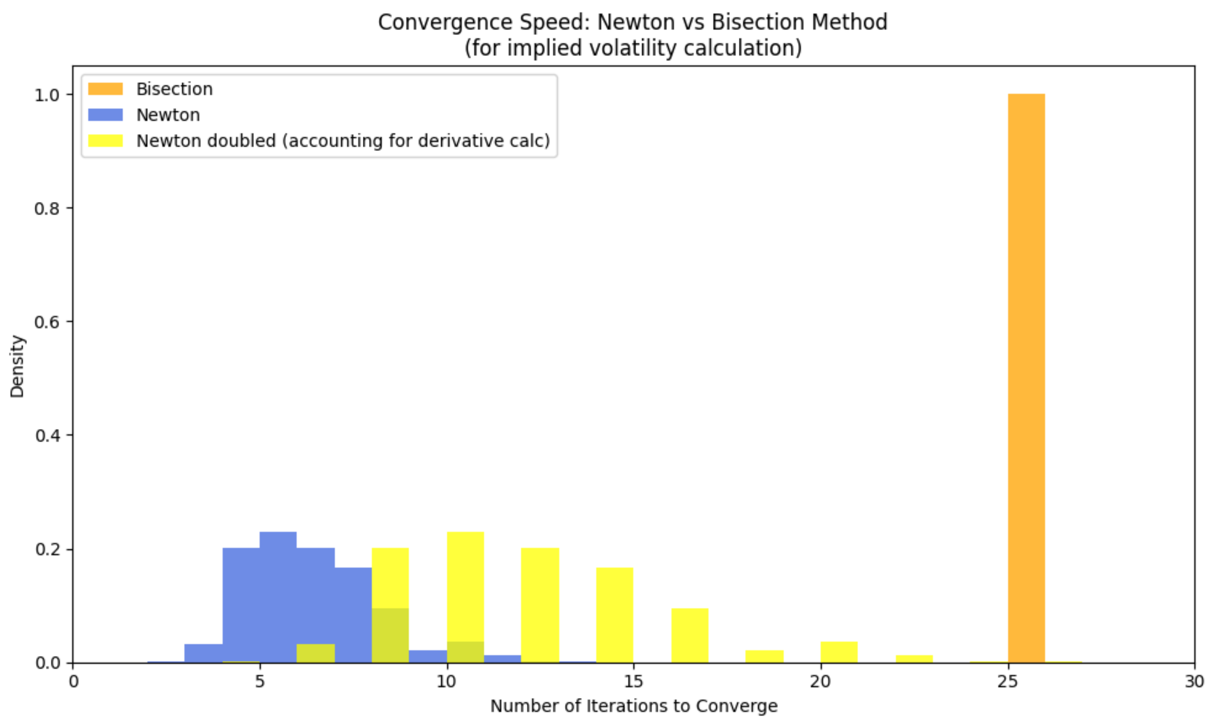
```
            row["daysToMaturity"] / 365,
            np.log1p(row["fedFunds"]),
            (row["bid"] + row["ask"]) / 2,
            log_iter = True
        )
        return i

def iter_newton(row):
    if row["optionType"] == "call":
        _, i = BlackScholes.iv_call_newton(
            row["underlyingPrice"],
            row["strike"],
            row["daysToMaturity"] / 365,
            np.log1p(row["fedFunds"]),
            (row["bid"] + row["ask"]) / 2,
            log_iter=True
        )
        return i
    else:
        _, i = BlackScholes.iv_put_newton(
            row["underlyingPrice"],
            row["strike"],
            row["daysToMaturity"] / 365,
            np.log1p(row["fedFunds"]),
            (row["bid"] + row["ask"]) / 2,
            log_iter=True
        )
        return i

iters_bisection = options.apply(iter_bisection, axis=1)
iters_newton = options.apply(iter_newton, axis=1)
```

```
/Users/ciangahan/Documents/Stevens/FE621/FE621/utils.py:68: RuntimeWarning:
divide by zero encountered in scalar divide
  x = x - fx / fpx
/Users/ciangahan/Documents/Stevens/FE621/FE621/pricing/black_scholes.py:13:
RuntimeWarning: invalid value encountered in scalar divide
  d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * t) / (sigma * np.sqrt(t))
/Users/ciangahan/Documents/Stevens/FE621/FE621/pricing/black_scholes.py:13:
RuntimeWarning: overflow encountered in scalar power
  d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * t) / (sigma * np.sqrt(t))
/Users/ciangahan/Documents/Stevens/FE621/FE621/utils.py:68: RuntimeWarning:
overflow encountered in scalar divide
  x = x - fx / fpx
```

In [50]:

```
fig, ax = plt.subplots(figsize=(10, 6))

ax.hist(iters_bisection,
        bins=np.arange(0, 30, 1),
        alpha=0.75,
        label='Bisection',
        color='orange',
        density=True)

ax.hist(iters_newton[iters_newton != 100],
        bins=np.arange(0, 30, 1),
```

```
        alpha=0.75,
        label='Newton',
        color='royalblue',
        density=True)

ax.hist(iters_newton[iters_newton * 2 <= 100] * 2,
        bins=np.arange(0, 30, 1),
        alpha=0.75,
        label='Newton doubled (accounting for derivative calc)',
        color='yellow',
        density=True)

ax.set_xlabel('Number of Iterations to Converge')
ax.set_ylabel('Density')
ax.set_title('Convergence Speed: Newton vs Bisection Method\n(for implied vo

ax.legend()
ax.set_xlim(0, 30)

plt.tight_layout()
plt.show()
```



The Newton method clearly converges faster with essentially all cases below 15 iterations, even though it has more variability than bisection. The bisection iterations are constant since the initial interval is the same, binary search within a certain tolerance should take the same number of iterations, validating the result. Shrinking the bisection interval to a less conservative estimate of implied volatility would not likely change the results by much, as the interval could be halved or quartered but this would only speed up by 1-2 iterations given the halving nature of the algorithm. It is worth noting the Newton method requires derivative calculations which double the computations per iteration, but even after doubling the iterations to account for this, the distribution still clearly outperforms bisection.

2.8:

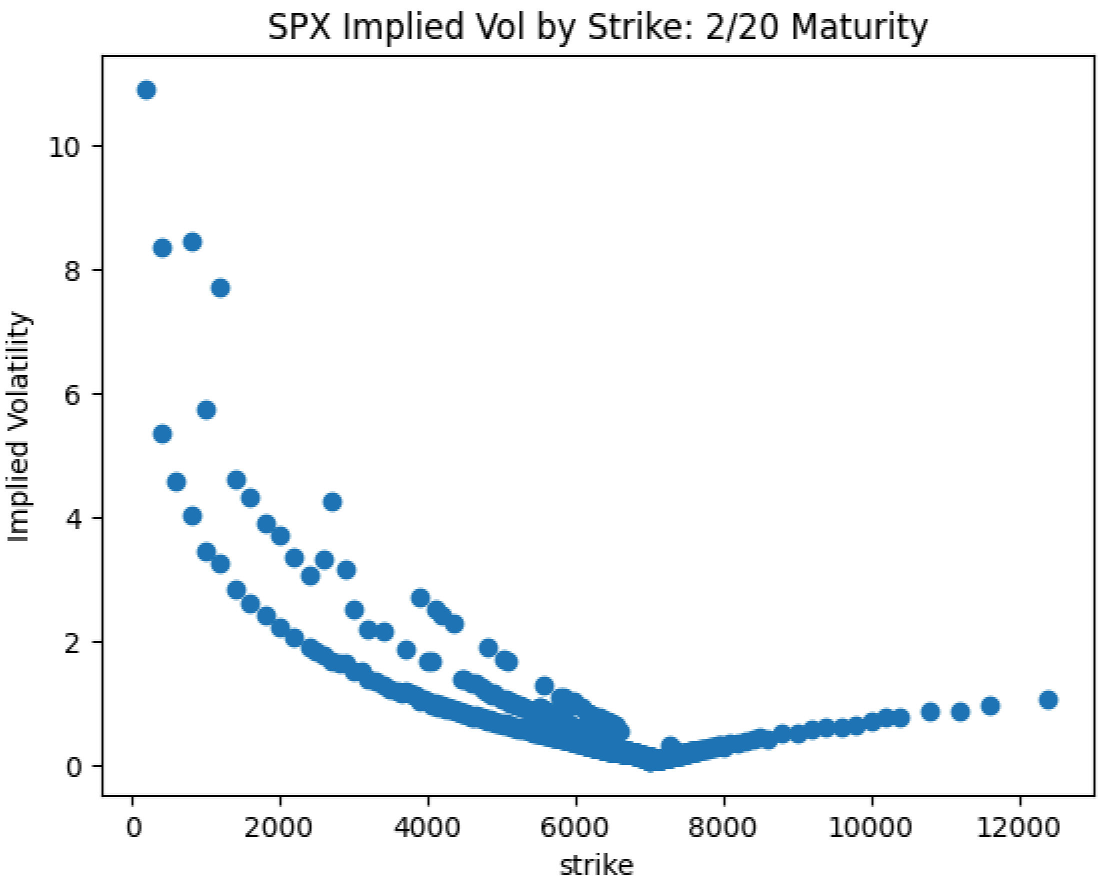
- To avoid skewing results with high OTM/ITM implied vols, I used the same moneyness thresholds from earlier for each stock/option type/maturity combination and averaged the results of each category's range for the table. I am using the bisection implied volatilities here

In [23]:
```
data1_atm = data1[(data1["moneyness"] >= 0.95) & (data1["moneyness"] <= 1.05
avg_ivs = data1_atm.groupby(
    ['expiration', 'optionType', 'underlying'],
    as_index=False
)['impliedVolatilityB'].mean()
avg_ivs
```

Out[23]:

	expiration	optionType	underlying	impliedVolatilityB
0	2026-02-20	call	TSLA	0.430960
1	2026-02-20	call	^SPX	0.170339
2	2026-02-20	put	TSLA	0.410741
3	2026-02-20	put	^SPX	0.168539
4	2026-03-20	call	TSLA	0.445434
5	2026-03-20	call	^SPX	0.168052
6	2026-03-20	put	TSLA	0.434499
7	2026-03-20	put	^SPX	0.159492
8	2026-04-17	call	TSLA	0.455664
9	2026-04-17	call	^SPX	0.166487
10	2026-04-17	put	TSLA	0.446099
11	2026-04-17	put	^SPX	0.162410

In [25]:
```
plt.scatter(spx_first["strike"], spx_first["impliedVolatilityB"])
plt.xlabel("strike")
plt.ylabel("Implied Volatility")
plt.title("SPX Implied Vol by Strike: 2/20 Maturity")
plt.show()
```



We can see that the implied volatility for TSLA options is consistently significantly higher than that of SPX, around 43-45% on average compared to 16-17%. As the maturity increases, the implied volatility for SPX options decreases slightly while TSLA options seem not to exhibit much of a pattern; both are relatively stable. We can see from the data (illustrated with an example of SPX 2/20 maturity options above) that the implied vol increases (sometimes significantly) as the option moves further from at the money in either direction. We can also see that the implied volatility of SPX options is lower than the current VIX value (16-17% vs 0.19-0.2) for all maturities, which could indicate an issue in my data or implied volatility calculations (as SPX options within this three-month window are used to calculate the VIX).

2.9:

- To save time I just aggregated all options with a corresponding pair using a left join on a call-filtered version of the dataframe and calculated put-call parity for each

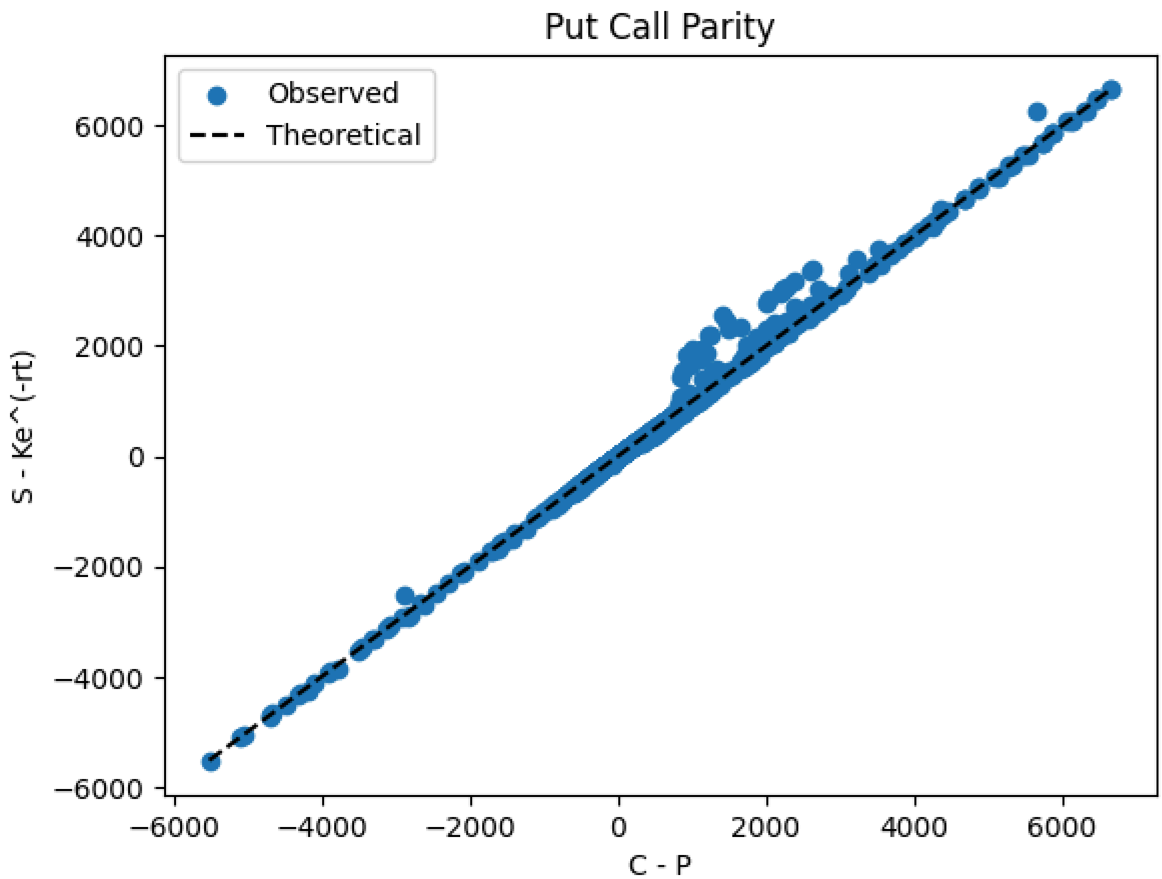
In [26]:
```
data1.head()
```

Out[26]:

	contractSymbol	strike	bid	ask	optionType	expiration	underlying	data_d
0	VIX260218C00010000	10.0	9.75	10.4	call	2026-02-18	^VIX	2026-
1	VIX260218C00010500	10.5	9.25	9.9	call	2026-02-18	^VIX	2026-
2	VIX260218C00011000	11.0	8.75	9.4	call	2026-02-18	^VIX	2026-
3	VIX260218C00011500	11.5	8.25	8.9	call	2026-02-18	^VIX	2026-
4	VIX260218C00012000	12.0	7.75	8.4	call	2026-02-18	^VIX	2026-

In [43]:

```
calls = data1[data1["optionType"] == "call"]
calls["midprice"] = (calls["bid"] + calls["ask"]) / 2
puts = data1[data1["optionType"] == "put"]
puts["midprice"] = (puts["bid"] + puts["ask"]) / 2
matched = pd.merge(calls, puts, how="inner", on=["strike", "underlying", "ex
matched = matched.rename(columns={"midprice_x": "call_price", "midprice_y":
matched["parity"] = matched["underlyingPrice"] - matched["strike"] * np.exp(

plt.scatter(matched["call_price"] - matched["put_price"], matched["parity"],
plt.plot(
    np.linspace(min(matched["call_price"] - matched["put_price"]), max(match
    np.linspace(min(matched["call_price"] - matched["put_price"]), max(match
)
plt.xlabel("C - P")
plt.ylabel("S - Ke^(-rt)")
plt.title("Put Call Parity")
plt.legend()
plt.show()
```



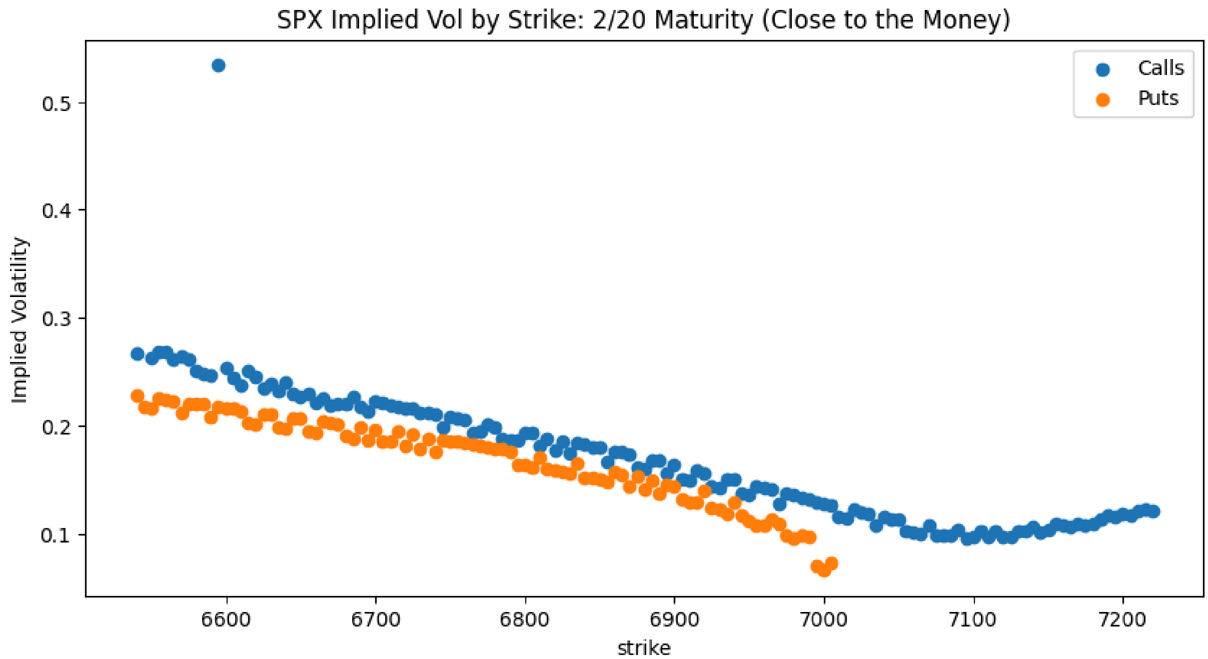
We can see that most of the option values roughly track with what is expected in put-call parity, though there are some near the money (where the stock price is slightly above the discounted strike) that have a higher than expected C - P difference. This could be due to them being American options or potential issues with the data (delays between stock price and option prices, stale quotes etc)

2.10:

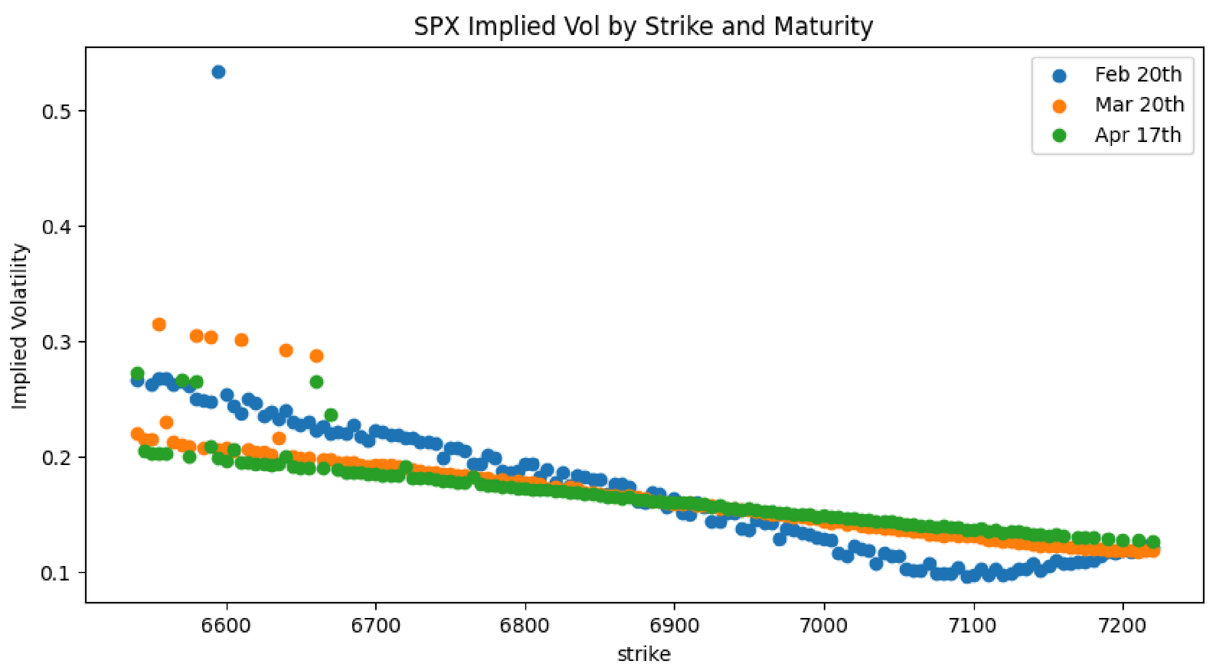
- Using SPX options here for the (ostensibly) best result as asset is not specified. I believe there is a data issue causing puts to have negative time value at a certain strike above the money, hence the dropoff and disappearance of those
- For the cross-maturity comparison I chose to just use calls to avoid confusion in the chart with six different series (or with different-shaped implied vol curves). I recognize this might be a data issue and not be relevant in practice but in this case I think it improves the quality of the chart

In [56]:
```
spx_first_atm = spx_first[(spx_first["moneyness"] >= 0.95) & (spx_first["mor
plt.figure(figsize=(10,5))
plt.scatter(spx_first_atm[spx_first_atm["optionType"] == "call"]["strike"],
plt.scatter(spx_first_atm[spx_first_atm["optionType"] == "put"]["strike"], s
plt.xlabel("strike")
plt.ylabel("Implied Volatility")
plt.title("SPX Implied Vol by Strike: 2/20 Maturity (Close to the Money)")
```

```
plt.legend()
plt.show()
```

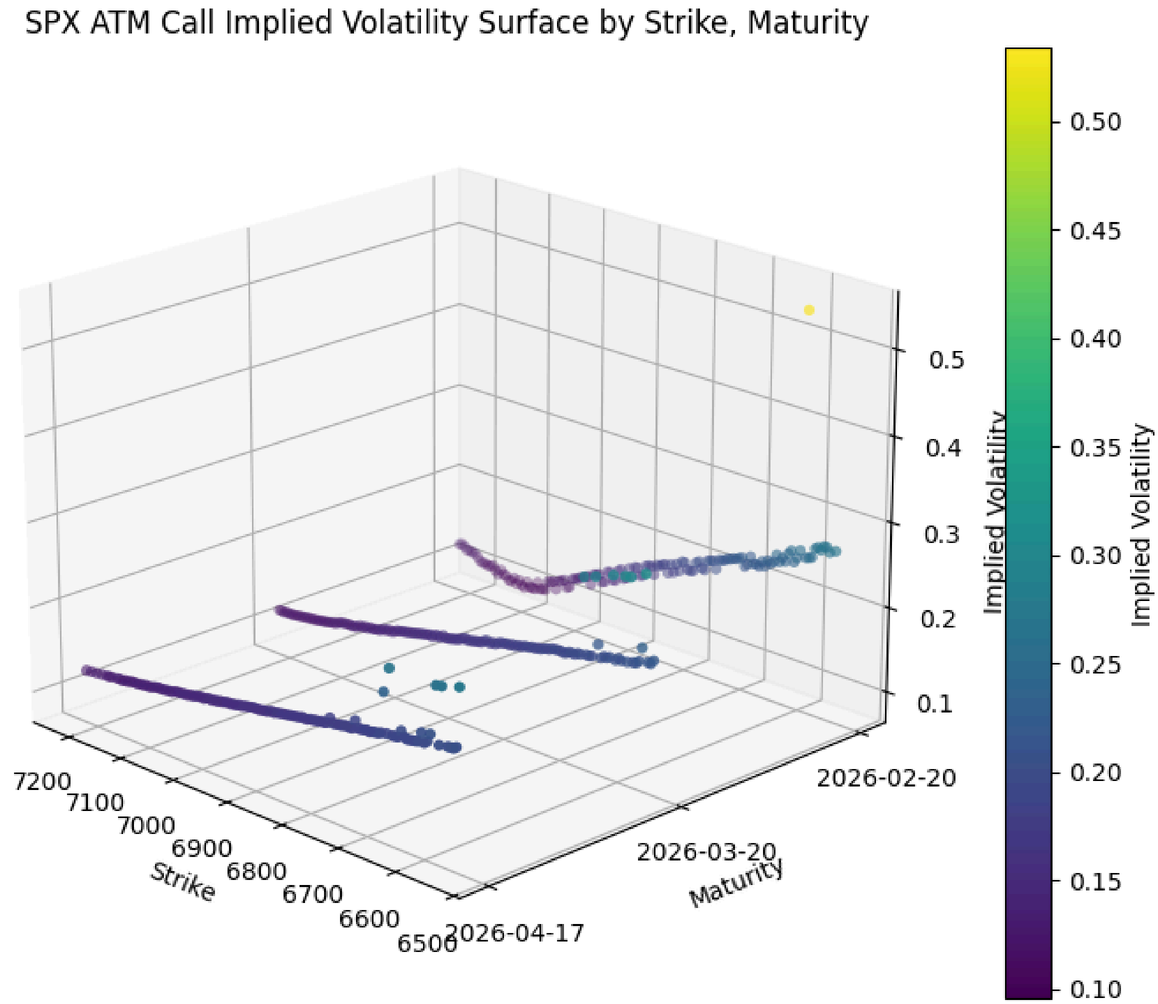


In [60]:
```
spx_atm = data1_atm[(data1_atm["underlying"] == "^SPX") & (data1_atm["optior
plt.figure(figsize=(10,5))
plt.scatter(spx_atm[spx_atm["expiration"] == "2026-02-20"]["strike"], spx_at
plt.scatter(spx_atm[spx_atm["expiration"] == "2026-03-20"]["strike"], spx_at
plt.scatter(spx_atm[spx_atm["expiration"] == "2026-04-17"]["strike"], spx_at
plt.xlabel("strike")
plt.ylabel("Implied Volatility")
plt.title("SPX Implied Vol by Strike and Maturity")
plt.legend()
plt.show()
```



Bonus:

In [85]:
```
maturities = ['2026-02-20', '2026-03-20', '2026-04-17']

# need to swap maturity strings for numbers so it will show on the chart
maturity_to_idx = {mat: i for i, mat in enumerate(maturities)}
spx_atm["mat_idx"] = spx_atm["expiration"].map(maturity_to_idx)

fig = plt.figure(figsize=(12, 7))
ax = fig.add_subplot(111, projection='3d')

# will use colors with colorbar for better interpretability (hard to read ot
sc = ax.scatter(
    spx_atm["strike"],
    spx_atm["mat_idx"],
    spx_atm["impliedVolatilityB"],
    c=spx_atm["impliedVolatilityB"],
    cmap='viridis',
    edgecolor='none'
)

ax.set_xlabel("Strike")
ax.set_ylabel("Maturity")
ax.set_zlabel("Implied Volatility")

ax.set_yticks(range(len(maturities)))
ax.set_yticklabels(maturities)

ax.set_title("SPX ATM Call Implied Volatility Surface by Strike, Maturity")

cbar = fig.colorbar(sc, ax=ax)
cbar.set_label("Implied Volatility")

# tilting the chart to see all points
ax.view_init(elev=20, azim=135)

ax.grid()
plt.show()
```



2.11:

- See `FE621/pricing/black_scholes.py` for implementation (includes theoretical and finite difference approximation)

In [ ]:
```
# TODO: choose options subset to calculate greeks on; compare results for ea
```

2.12:

- I used an inner join to add all existing data1 implied vols onto data2 using merge function, discarding data2 entries where data1 implied vol didn't converge. From there I used black-scholes and fetched remaining params from data2

In [ ]:
```
def calc_price(row):
    if row["optionType"] == "call":
        return BlackScholes.call(row["underlyingPrice"], row["strike"], row[
    else:
        return BlackScholes.put(row["underlyingPrice"], row["strike"], row["

data2 = pd.merge(options[options["data_date"] == "2026-02-13"], data1[["cont
data2 = data2.rename(columns={"impliedVolatilityB_y": "impliedVolatility"})
```

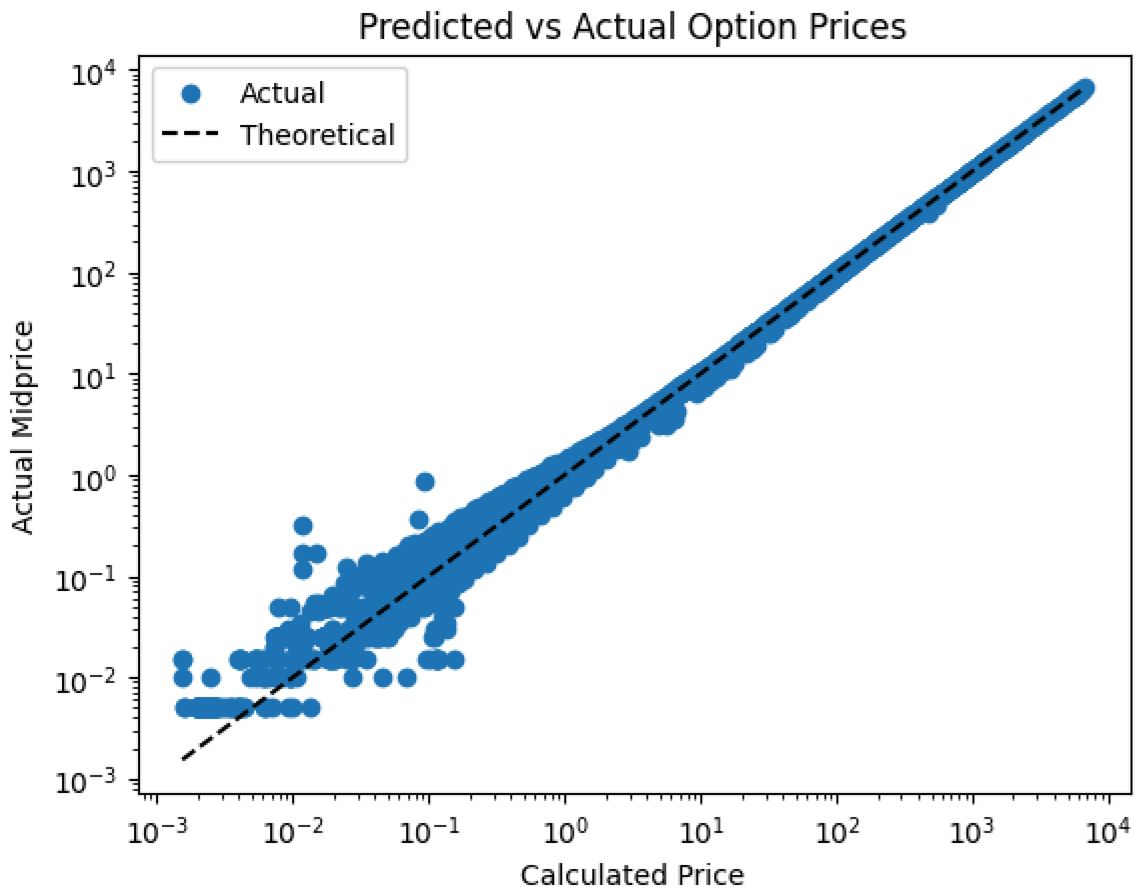
```
data2["midprice"] = (data2["bid"] + data2["ask"]) / 2
data2["calcprice"] = data2.apply(calc_price, axis=1)
data2.head()
```

Out[ ]:

	contractSymbol	strike	bid	ask	optionType	expiration	underlying	data_da
0	VIX260218C00010000	10.0	9.5	10.1	call	2026-02-18	^VIX	2026-0:
1	VIX260218C00010500	10.5	9.0	9.6	call	2026-02-18	^VIX	2026-0:
2	VIX260218C00011000	11.0	8.5	9.1	call	2026-02-18	^VIX	2026-0:
3	VIX260218C00011500	11.5	8.0	8.6	call	2026-02-18	^VIX	2026-0:
4	VIX260218C00012000	12.0	7.5	8.1	call	2026-02-18	^VIX	2026-0:

In [76]:

```
plt.scatter(data2["calcprice"], data2["midprice"], label="Actual")
plt.plot(
    np.linspace(min(data2["calcprice"]), max(data2["calcprice"])),
    np.linspace(min(data2["calcprice"]), max(data2["calcprice"])), "k--", la
)
plt.xscale('log')
plt.yscale('log')
plt.xlabel("Calculated Price")
plt.ylabel("Actual Midprice")
plt.title("Predicted vs Actual Option Prices")
plt.legend()
plt.show()
```



I used a log-log plot to better zoom in on differences between options with lower prices as these are the most sensitive/potentially interesting. It appears the calculated prices did a decent job of predicting what the actual prices would end up as, but there were still notable discrepancies, particularly for lower-priced options. Implied volatility likely changed for many of these options day-to-day.

Part 3: Numerical Integration of AMM Arbitrage Fee Revenue

Note volatility = 0.2, fee rate = 0.003

P_{t+1} = \frac{y_t - \Delta y}{x_t + (1 - \gamma)\Delta x} = S_{t+1}(1-\gamma)

I will begin with Case 1. First solving for Δx:

$$(x_t + (1-\gamma)\Delta x)(y_t - \Delta y) = k \quad (1)$$

$$(x_t + (1-\gamma)\Delta x) = \frac{k}{(y_t - \Delta y)} \quad (2)$$

$$(1-\gamma)\Delta x = \frac{k}{(y_t - \Delta y)} - x_t \quad (3)$$

$$\Delta x = (\frac{k}{(y_t - \Delta y)} - x_t)(1-\gamma) \quad (4)$$

Next, solving for Δy and plugging in for Δx:

$$\begin{align}
\frac{y_t - \Delta y}{x_t + (1-\gamma)\Delta x} &= S_{t+1}(1-\gamma) \tag{5}\\
\frac{y_t - \Delta y}{x_t + (1-\gamma)^2(\frac{k}{(y_t-\Delta y)} - x_t)} &= S_{t+1}(1-\gamma) \tag{6}\\
\frac{(y_t - \Delta y)}{x_t(1-(1-\gamma)^2) + \frac{k(1-\gamma)^2}{(y_t-\Delta y)}} &= S_{t+1}(1-\gamma) \tag{7}\\
\frac{(y_t - \Delta y)^2}{x_t(1-(1-\gamma)^2)(y_t-\Delta y) + k(1-\gamma)^2} &= S_{t+1}(1-\gamma) \tag{8}\\
(y_t - \Delta y)^2 = S_{t+1}(1-\gamma)(x_t(1-(1-\gamma)^2)(y_t-\Delta y) + k(1-\gamma)^2) \tag{9}\\
(y_t - \Delta y)^2 - S_{t+1}(1-\gamma)x_t(1-(1-\gamma)^2)(y_t-\Delta y) - S_{t+1}k(1-\gamma)^3 = 0 \tag{10}
\end{align}$$

Using the quadratic formula:

$$y_t - \Delta y = \frac{S_{t+1}(1-\gamma)x_t(1-(1-\gamma)^2) \pm \sqrt{(S_{t+1}(1-\gamma)x_t(1-(1-\gamma)^2))^2 + }}{2}$$

Since $y_{t+1} > 0$ and the inside of the square root is greater than the first term in the fraction numerator, we can discard the minus as extraneous solution:

$$\Delta y = y_t - \frac{S_{t+1}(1-\gamma)x_t(1-(1-\gamma)^2) + \sqrt{(S_{t+1}(1-\gamma)x_t(1-(1-\gamma)^2))^2 + }}{2}$$

Next Case 2. Can find Δx as function of above Δy:

```
In [ ]:
```

Was unable to finish this section on time unfortunately

Part 4 Bonus: Numerical Integration

$$f_1(x,y) = xyf_2(x,y) = e^{x+y} \tag{13}$$

4.1: Analytically solve the following integral for f_1 and f_2:

$$
\begin{aligned}
I_1 &= \int_0^1 \left(\int_0^3 xy\,dy \right) dx && (14)\\
&= \int_0^1 x \left[\frac{y^2}{2} \right]_0^3 dx && (15)\\
&= \int_0^1 x \cdot \frac{9}{2}\,dx && (16)\\
&= \frac{9}{2} \int_0^1 x\,dx && (17)\\
&= \frac{9}{2} \left[\frac{x^2}{2} \right]_0^1 && (18)\\
&= \frac{9}{2} \cdot \frac{1}{2} = \frac{9}{4} && (19)\\
I_2 &= \int_0^1 \int_0^3 e^{x+y}\,dydx && (20)\\
&= \int_0^1 e^x \int_0^3 e^y\,dydx && (21)\\
&= \int_0^1 e^x [e^y]_0^3 dx && (22)\\
&= \int_0^1 e^x (e^3 - e^0)\,dx && (23)\\
&= (e^3 - 1) \int_0^1 e^x\,dx && (24)\\
&= (e^3 - 1)[e^x]_0^1 && (25)\\
&= (e^3 - 1)(e^1 - e^0) && (26)\\
&= (e^3 - 1)(e - 1) && (27)
\end{aligned}
$$

In [87]:
```
print("First Integral: " + str(9/4))
print("Second Integral: " + str((np.exp(3) - 1)*(np.exp(1) - 1)))
```

```
First Integral: 2.25
Second Integral: 32.79433128149753
```

Was unable to finish this section on time, which would have included debugging the below function (which has some syntax issue presumably)

In [89]:
```
def trapezoid_integrate(f, x0, x1, n, y0, y1, m):
    dx = (x1 - x0) / n
    dy = (y1 - y0) / m

    # set up discrete points
    x = np.linspace(x0, x1, n + 1)
    y = np.linspace(y0, y1, m + 1)

    res = 0
```

```
    for xi in x:
        for yi in y:
            res += dx * dy / 16 * (f(xi, yi) + f(xi, yi + dy) + f(xi + dx, y
                                   2*(f(xi + dx/2, yi) + f(xi + dx/2, yi + c
                                   4*f(x+dx/2, y+dy/2))

    return res
```

In [90]:
```
def f1(x, y):
    return x * y

def f2(x, y):
    return np.exp(x + y)
```

In [91]:
```
pairs = [(10, 10), (50, 50), (100, 100), (500, 500)]

for pair in pairs:
    n, m = pair
    print("First Integral Approx: " + str(trapezoid_integrate(f1, 0, 1, n, 0
    print("Second Integral Approx: " + str(trapezoid_integrate(f2, 0, 1, n,
```

```
First Integral Approx: [2.477475 2.531925 2.640825 2.804175 3.021975 3.29422
5 3.620925 4.002075
 4.437675 4.927725 5.472225]
Second Integral Approx: [40.52234749 41.06749729 41.88076524 43.09401845 44.
90397956 47.60412423
 51.63226675 57.64154924 66.60634528 79.98024941 99.9317699 ]
First Integral Approx: [1.82683836 1.82871108 1.83245652 1.83807468 1.845565
56 1.85492916
 1.86616548 1.87927452 1.89425628 1.91111076 1.92983796 1.95043788
 1.97291052 1.99725588 2.02347396 2.05156476 2.08152828 2.11336452
 2.14707348 2.18265516 2.22010956 2.25943668 2.30063652 2.34370908
 2.38865436 2.43547236 2.48416308 2.53472652 2.58716268 2.64147156
 2.69765316 2.75570748 2.81563452 2.87743428 2.94110676 3.00665196
 3.07406988 3.14336052 3.21452388 3.28755996 3.36246876 3.43925028
 3.51790452 3.59843148 3.68083116 3.76510356 3.85124868 3.93926652
 4.02915708 4.12092036 4.21455636]
Second Integral Approx: [27.8501885  27.91782964 27.99110442 28.07048204 28.
15647079 28.24962129
 28.35053002 28.45984315 28.57826064 28.70654078 28.845505   28.99604314
 29.15911916 29.3357773  29.52714878 29.73445903 29.95903555 30.20231638
 30.46585936 30.75135207 31.06062262 31.39565141 31.75858376 32.15174369
 32.57764876 33.03902621 33.53883043 34.08026188 34.66678757 35.30216327
 35.99045754 36.73607782 37.54379864 38.41879215 39.3666613  40.3934757
 41.50581045 42.71078831 44.01612524 45.43017985 46.96200693 48.62141539
 50.41903111 52.36636498 54.47588658 56.76110404 59.23665057 61.91837811
 64.82345887 67.97049529 71.37963914]
First Integral Approx: [1.75607665 1.75653569 1.75745378 1.75883092 1.760667
1  1.76296232
 1.76571659 1.76892991 1.77260227 1.77673367 1.78132412 1.78637362
 1.79188216 1.79784974 1.80427637 1.81116205 1.81850677 1.82631053
 1.83457334 1.8432952  1.8524761  1.86211604 1.87221503 1.88277307
 1.89379015 1.90526627 1.91720144 1.92959566 1.94244892 1.95576122
 1.96953257 1.98376297 1.99845241 2.01360089 2.02920842 2.045275
 2.06180062 2.07878528 2.09622899 2.11413175 2.13249355 2.15131439
 2.17059428 2.19033322 2.2105312  2.23118822 2.25230429 2.27387941
 2.29591357 2.31840677 2.34135902 2.36477032 2.38864066 2.41297004
 2.43775847 2.46300595 2.48871247 2.51487803 2.54150264 2.5685863
 2.596129   2.62413074 2.65259153 2.68151137 2.71089025 2.74072817
 2.77102514 2.80178116 2.83299622 2.86467032 2.89680347 2.92939567
 2.96244691 2.99595719 3.02992652 3.0643549  3.09924232 3.13458878
 3.17039429 3.20665885 3.24338245 3.28056509 3.31820678 3.35630752
 3.3948673  3.43388612 3.47336399 3.51330091 3.55369687 3.59455187
 3.63586592 3.67763902 3.71987116 3.76256234 3.80571257 3.84932185
 3.89339017 3.93791753 3.98290394 4.0283494  4.0742539 ]
Second Integral Approx: [26.56923691 26.60109097 26.63424501 26.6687521  26.
70466745 26.74204853
 26.78095516 26.8214496  26.86359665 26.90746376 26.95312111 27.00064178
 27.0501018  27.10158032 27.15515973 27.21092575 27.26896762 27.32937823
 27.39225424 27.45769627 27.52580904 27.59670154 27.67048722 27.74728416
 27.82721523 27.91040836 27.99699666 28.0871187  28.18091868 28.27854672
 28.38015903 28.48591822 28.59599352 28.71056108 28.82980424 28.9539138
 29.08308836 29.21753464 29.35746778 29.50311169 29.65469945 29.81247362
 29.97668668 30.14760139 30.32549127 30.51064098 30.70334678 30.90391706
 31.11267277 31.32994796 31.55609032 31.79146173 32.03643882 32.29141362
 32.55679414 32.83300504 33.12048832 33.41970402 33.73113094 34.05526743
 34.39263219 34.74376506 35.10922794 35.48960564 35.88550685 36.2975651
```

```
 36.72643976 37.17281712 37.6374115  38.12096633 38.6242554  39.1480841
 39.69329064 40.26074749 40.85136269 41.46608136 42.10588717 42.77180395
 43.46489731 44.18627635 44.93709542 45.718556   46.5319086  47.37845474
 48.25954908 49.17660157 50.13107968 51.12451078 52.15848457 53.23465564
 54.35474608 55.52054827 56.73392776 57.9968262  59.31126451 60.67934606
 62.10326007 63.58528512 65.12779276 66.73325134 68.40422991]
First Integral Approx: [1.70104281 1.70106089 1.70109703 1.70115125 1.701223
53 1.70131389
 1.70142233 1.70154883 1.70169341 1.70185606 1.70203678 1.70223557
 1.70245243 1.70268737 1.70294038 1.70321146 1.70350061 1.70380784
 1.70413314 1.70447651 1.70483795 1.70521746 1.70561505 1.7060307
 1.70646443 1.70691624 1.70738611 1.70787406 1.70838007 1.70890416
 1.70944633 1.71000656 1.71058487 1.71118125 1.7117957  1.71242822
 1.71307881 1.71374748 1.71443422 1.71513903 1.71586191 1.71660287
 1.71736189 1.71813899 1.71893416 1.71974741 1.72057872 1.72142811
 1.72229557 1.7231811  1.7240847  1.72500638 1.72594613 1.72690395
 1.72787984 1.7288738  1.72988584 1.73091595 1.73196413 1.73303038
 1.7341147  1.7352171  1.73633757 1.73747611 1.73863272 1.73980741
 1.74100016 1.74221099 1.74343989 1.74468687 1.74595191 1.74723503
 1.74853622 1.74985548 1.75119281 1.75254822 1.7539217  1.75531325
 1.75672287 1.75815056 1.75959633 1.76106016 1.76254207 1.76404206
 1.76556011 1.76709624 1.76865043 1.7702227  1.77181305 1.77342146
 1.77504795 1.77669251 1.77835514 1.78003584 1.78173461 1.78345146
 1.78518638 1.78693937 1.78871043 1.79049957 1.79230678 1.79413206
 1.79597541 1.79783683 1.79971633 1.80161389 1.80352953 1.80546325
 1.80741503 1.80938488 1.81137281 1.81337881 1.81540288 1.81744503
 1.81950524 1.82158353 1.82367989 1.82579433 1.82792683 1.83007741
 1.83224606 1.83443278 1.83663757 1.83886043 1.84110137 1.84336038
 1.84563746 1.84793261 1.85024584 1.85257714 1.85492651 1.85729395
 1.85967946 1.86208305 1.8645047  1.86694443 1.86940224 1.87187811
 1.87437206 1.87688407 1.87941416 1.88196233 1.88452856 1.88711287
 1.88971524 1.8923357  1.89497422 1.89763081 1.90030548 1.90299822
 1.90570903 1.90843791 1.91118487 1.91394989 1.91673299 1.91953416
 1.92235341 1.92519072 1.92804611 1.93091957 1.9338111  1.9367207
 1.93964838 1.94259413 1.94555795 1.94853984 1.9515398  1.95455784
 1.95759395 1.96064813 1.96372038 1.9668107  1.9699191  1.97304557
 1.97619011 1.97935272 1.98253341 1.98573216 1.98894899 1.99218389
 1.99543687 1.99870791 2.00199703 2.00530422 2.00862948 2.01197281
 2.01533422 2.01871369 2.02211124 2.02552687 2.02896056 2.03241233
 2.03588216 2.03937007 2.04287605 2.04640011 2.04994224 2.05350243
 2.0570807  2.06067705 2.06429146 2.06792395 2.07157451 2.07524314
 2.07892984 2.08263461 2.08635746 2.09009838 2.09385737 2.09763443
 2.10142957 2.10524278 2.10907405 2.11292341 2.11679083 2.12067632
 2.12457989 2.12850153 2.13244124 2.13639903 2.14037488 2.14436881
 2.14838081 2.15241088 2.15645903 2.16052524 2.16460953 2.16871189
 2.17283232 2.17697083 2.18112741 2.18530205 2.18949478 2.19370557
 2.19793443 2.20218137 2.20644638 2.21072946 2.21503061 2.21934984
 2.22368714 2.2280425  2.23241595 2.23680746 2.24121705 2.2456447
 2.25009043 2.25455423 2.25903611 2.26353605 2.26805407 2.27259016
 2.27714432 2.28171656 2.28630686 2.29091524 2.29554169 2.30018622
 2.30484881 2.30952948 2.31422822 2.31894503 2.32367991 2.32843286
 2.33320389 2.33799299 2.34280016 2.34762541 2.35246872 2.35733011
 2.36220957 2.3671071  2.3720227  2.37695638 2.38190813 2.38687795
 2.39186584 2.3968718  2.40189584 2.40693795 2.41199813 2.41707638
 2.4221727  2.4272871  2.43241957 2.43757011 2.44273872 2.4479254
 2.45313016 2.45835299 2.46359389 2.46885286 2.47412991 2.47942503
```

```
 2.48473822 2.49006948 2.49541881 2.50078622 2.50617169 2.51157524
 2.51699686 2.52243656 2.52789432 2.53337016 2.53886407 2.54437605
 2.54990611 2.55545423 2.56102043 2.5666047  2.57220704 2.57782746
 2.58346594 2.5891225  2.59479713 2.60048984 2.60620061 2.61192946
 2.61767638 2.62344137 2.62922443 2.63502557 2.64084477 2.64668205
 2.6525374  2.65841083 2.66430232 2.67021189 2.67613953 2.68208524
 2.68804903 2.69403088 2.70003081 2.70604881 2.71208488 2.71813903
 2.72421124 2.73030153 2.73640989 2.74253632 2.74868083 2.7548434
 2.76102405 2.76722277 2.77343957 2.77967443 2.78592737 2.79219838
 2.79848746 2.80479461 2.81111984 2.81746313 2.8238245  2.83020394
 2.83660146 2.84301704 2.8494507  2.85590243 2.86237223 2.86886011
 2.87536605 2.88189007 2.88843216 2.89499232 2.90157056 2.90816686
 2.91478124 2.92141369 2.92806421 2.93473281 2.94141947 2.94812421
 2.95484702 2.96158791 2.96834686 2.97512389 2.98191899 2.98873216
 2.9955634  3.00241272 3.00928011 3.01616556 3.0230691  3.0299907
 3.03693038 3.04388812 3.05086394 3.05785783 3.0648698  3.07189983
 3.07894794 3.08601412 3.09309838 3.1002007  3.1073211  3.11445956
 3.1216161  3.12879072 3.1359834  3.14319416 3.15042299 3.15766989
 3.16493486 3.17221791 3.17951902 3.18683821 3.19417547 3.20153081
 3.20890421 3.21629569 3.22370524 3.23113286 3.23857855 3.24604232
 3.25352416 3.26102407 3.26854205 3.2760781  3.28363223 3.29120443
 3.2987947  3.30640304 3.31402946 3.32167394 3.3293365  3.33701713
 3.34471583 3.35243261 3.36016746 3.36792037 3.37569136 3.38348043
 3.39128756 3.39911277 3.40695605 3.4148174  3.42269682 3.43059432
 3.43850989 3.44644353 3.45439524 3.46236502 3.47035288 3.47835881
 3.48638281 3.49442488 3.50248502 3.51056324 3.51865953 3.52677389
 3.53490632 3.54305682 3.5512254  3.55941205 3.56761677 3.57583956
 3.58408043 3.59233936 3.60061637 3.60891145 3.61722461 3.62555583
 3.63390513 3.6422725  3.65065794 3.65906145 3.66748304 3.6759227
 3.68438043 3.69285623 3.7013501  3.70986205 3.71839207 3.72694016
 3.73550632 3.74409055 3.75269286 3.76131324 3.76995169 3.77860821
 3.7872828  3.79597547 3.80468621 3.81341502 3.8221619  3.83092686
 3.83970989 3.84851098 3.85733016 3.8661674  3.87502271 3.8838961
 3.89278756 3.90169709 3.9106247  3.91957037 3.92853412 3.93751594
 3.94651583 3.9555338  3.96456983]
Second Integral Approx: [25.58597553 25.59204795 25.59816915 25.60433952 25.
61055945 25.61682933
 25.62314958 25.62952059 25.63594277 25.64241654 25.6489423  25.65552048
 25.6621515  25.66883578 25.67557375 25.68236583 25.68921247 25.69611411
 25.70307117 25.71008412 25.7171534  25.72427946 25.73146275 25.73870374
 25.74600289 25.75336067 25.76077755 25.768254   25.7757905  25.78338754
 25.79104559 25.79876516 25.80654673 25.8143908  25.82229788 25.83026847
 25.83830307 25.84640222 25.85456641 25.86279618 25.87109205 25.87945456
 25.88788424 25.89638162 25.90494725 25.91358168 25.92228547 25.93105917
 25.93990333 25.94881854 25.95780535 25.96686434 25.9759961  25.98520121
 25.99448025 26.00383382 26.01326252 26.02276695 26.03234772 26.04200544
 26.05174074 26.06155423 26.07144655 26.08141832 26.09147018 26.10160278
 26.11181677 26.1221128  26.13249153 26.14295361 26.15349974 26.16413056
 26.17484678 26.18564907 26.19653813 26.20751464 26.21857932 26.22973288
 26.24097602 26.25230946 26.26373394 26.27525018 26.28685892 26.2985609
 26.31035687 26.32224759 26.33423381 26.34631631 26.35849586 26.37077324
 26.38314923 26.39562462 26.40820021 26.42087682 26.43365524 26.44653631
 26.45952083 26.47260964 26.48580359 26.49910351 26.51251026 26.52602469
 26.53964767 26.55338008 26.56722278 26.58117666 26.59524263 26.60942158
 26.62371441 26.63812204 26.6526454  26.66728541 26.68204301 26.69691914
 26.71191476 26.72703082 26.7422683  26.75762817 26.77311141 26.78871901
```

```
26.80445198 26.82031131 26.83629803 26.85241315 26.86865771 26.88503275
26.90153931 26.91817846 26.93495125 26.95185877 26.96890209 26.9860823
27.0034005  27.0208578  27.03845533 27.05619419 27.07407554 27.09210051
27.11027026 27.12858595 27.14704876 27.16565986 27.18442044 27.20333172
27.22239488 27.24161117 27.2609818  27.28050802 27.30019108 27.32003223
27.34003275 27.36019391 27.38051701 27.40100335 27.42165423 27.44247098
27.46345494 27.48460744 27.50592984 27.5274235  27.5490898  27.57093013
27.59294588 27.61513847 27.6375093  27.66005982 27.68279147 27.7057057
27.72880398 27.75208779 27.77555861 27.79921796 27.82306733 27.84710827
27.87134231 27.895771   27.9203959  27.94521859 27.97024065 27.9954637
28.02088934 28.0465192  28.07235493 28.09839816 28.12465058 28.15111386
28.1777897  28.2046798  28.23178588 28.25910968 28.28665295 28.31441745
28.34240495 28.37061725 28.39905616 28.42772349 28.45662108 28.48575078
28.51511444 28.54471396 28.57455123 28.60462815 28.63494665 28.66550867
28.69631617 28.72737112 28.75867551 28.79023133 28.82204061 28.85410539
28.88642771 28.91900965 28.95185329 28.98496074 29.0183341  29.05197552
29.08588716 29.12007117 29.15452976 29.18926512 29.22427947 29.25957507
29.29515416 29.33101902 29.36717196 29.40361528 29.44035131 29.47738241
29.51471095 29.55233932 29.59026991 29.62850518 29.66704754 29.70589949
29.7450635  29.78454207 29.82433774 29.86445306 29.90489058 29.9456529
29.98674263 30.02816239 30.06991484 30.11200265 30.15442851 30.19719514
30.24030528 30.28376168 30.32756713 30.37172442 30.41623639 30.46110588
30.50633577 30.55192895 30.59788834 30.64421688 30.69091753 30.73799329
30.78544716 30.83328219 30.88150143 30.93010797 30.97910493 31.02849543
31.07828264 31.12846975 31.17905996 31.23005652 31.28146268 31.33328175
31.38551703 31.43817187 31.49124964 31.54475373 31.59868757 31.65305461
31.70785834 31.76310225 31.81878988 31.87492481 31.93151061 31.98855092
32.04604938 32.10400967 32.1624355  32.22133062 32.28069878 32.3405438
32.40086949 32.46167973 32.5229784  32.58476942 32.64705676 32.70984439
32.77313634 32.83693666 32.90124942 32.96607876 33.0314288  33.09730374
33.1637078  33.23064522 33.29812029 33.36613732 33.43470067 33.50381473
33.57348391 33.64371269 33.71450555 33.78586702 33.85780168 33.93031412
34.00340898 34.07709096 34.15136475 34.22623511 34.30170684 34.37778476
34.45447375 34.53177872 34.6097046  34.68825639 34.76743911 34.84725784
34.92771767 35.00882377 35.09058132 35.17299555 35.25607174 35.33981521
35.42423131 35.50932544 35.59510306 35.68156966 35.76873076 35.85659195
35.94515884 36.03443711 36.12443247 36.21515069 36.30659756 36.39877894
36.49170072 36.58536887 36.67978936 36.77496825 36.87091162 36.96762562
37.06511643 37.16339029 37.26245351 37.3623124  37.46297337 37.56444286
37.66672736 37.76983342 37.87376763 37.97853665 38.08414719 38.190606
38.29791989 38.40609574 38.51514047 38.62506105 38.73586453 38.84755799
38.96014858 39.07364351 39.18805003 39.30337548 39.41962724 39.53681274
39.65493948 39.77401503 39.894047   40.01504308 40.13701101 40.2599586
40.38389371 40.50882428 40.6347583  40.76170384 40.88966901 41.01866201
41.14869109 41.27976458 41.41189086 41.54507839 41.6793357  41.81467137
41.95109407 42.08861252 42.22723554 42.36697199 42.50783081 42.64982103
42.79295172 42.93723205 43.08267125 43.22927863 43.37706358 43.52603554
43.67620407 43.82757875 43.9801693  44.13398546 44.28903709 44.4453341
44.60288651 44.76170439 44.92179791 45.08317731 45.24585293 45.40983517
45.57513453 45.74176159 45.90972702 46.07904155 46.24971604 46.42176139
46.59518863 46.77000885 46.94623325 47.12387309 47.30293975 47.48344469
47.66539946 47.8488157  48.03370517 48.22007967 48.40795116 48.59733165
48.78823325 48.98066819 49.17464879 49.37018746 49.56729671 49.76598916
49.96627752 50.16817461 50.37169336 50.5768468  50.78364804 50.99211032
51.20224699 51.4140715  51.6275974  51.84283836 52.05980816 52.27852067
52.4989899  52.72122996 52.94525507 53.17107957 53.39871791 53.62818466
```

```
53.8594945  54.09266225 54.32770282 54.56463126 54.80346272 55.04421251
55.28689601 55.53152877 55.77812644 56.0267048  56.27727977 56.52986737
56.78448378 57.0411453  57.29986833 57.56066946 57.82356536 58.08857286
58.35570893 58.62499066 58.89643528 59.17006016 59.44588283 59.72392092
60.00419225 60.28671473 60.57150646 60.85858566 61.14797071 61.43968012
61.73373257 62.03014687 62.32894199 62.63013707 62.93375137 63.23980432
63.54831552 63.85930471 64.17279178 64.48879682 64.80734003 65.12844181
65.4521227  65.77840343 66.10730488]
```