https://docs.google.com/document/d/1YYUFmlh3-fBMn7PPJwzOkj7dBXMx5rIFMs1qqpd7mzQ/edit?usp=sharing

```
!pip install numpy pandas matplotlib scipy yfinance
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (1.14.1)
Requirement already satisfied: yfinance in /usr/local/lib/python3.11/dist-packages (0.2.55)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.11/dist-packages (from yfinance) (2.32.3)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.11/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from yfinance) (4.3.7)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.11/dist-packages (from yfinance) (2.4.6)
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.11/dist-packages (from yfinance) (3.17.9)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.11/dist-packages (from yfinance) (4.13.4)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.11.1->yfinance) (2.
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.11.1->yf
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance) (2.3.0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance) (2025.
```

```
# Load and parse the option chain files
with open("/content/May 16 2025 NDAQ Option Chain.txt") as f:
    lines = f.read().splitlines()

for i, line in enumerate(lines):
    if line.startswith("Contract Name"):
        calls_start = i
        break
else:
    # If the header is not found, raise an error
    raise ValueError("Header row not found in the file.")

calls_lines = [ln for ln in lines[calls_start+1:] if ln.strip() and ln.startswith("NDAQ")]
puts_lines = [ln for ln in lines if ln.startswith("NDAQ250516P")]
# Parse IVs from Calls and Puts
import re
def parse_ivs(lines):
    ivs = []
    for ln in lines:
        parts = ln.split()
        iv_str = parts[-1]                    # last token is e.g. "54.06%"
        if iv_str.endswith('%'):
            ivs.append(float(iv_str[:-1]) / 100.0)
    return ivs

ivs_20250516_calls = parse_ivs(calls_lines)
ivs_20250516_puts  = parse_ivs(puts_lines)
# Similarly for Jan 2027 file
with open("/content/January 15 2027 NDAQ Option Chain.txt") as f:
    lines = f.read().splitlines()
# Skip headings and find sections
# Use the same logic as above to find the header row in the second file.
for i, line in enumerate(lines):
    if line.startswith("Contract Name"):
        calls_start = i
        break
else:
    # If the header is not found, raise an error
    raise ValueError("Header row not found in the file.")

calls_lines = [ln for ln in lines[calls_start+1:] if ln.strip() and ln.startswith("NDAQ")]
puts_lines = [ln for ln in lines if ln.startswith("NDAQ270115P")]
```

```
ivs_20270115_calls = parse_ivs(calls_lines)
ivs_20270115_puts  = parse_ivs(puts_lines)
# Combine all IVs (filtering out any zero entries)
ivs = [iv for iv in (ivs_20250516_calls + ivs_20250516_puts + ivs_20270115_calls + ivs_20270115_puts) if iv>0]
print(ivs)
```

```
[1.5497999999999998, 1.3149000000000002, 0.9057999999999999, 0.7426999999999999, 0.6523, 0.39380000000000004, 0.32299999999999995,
```

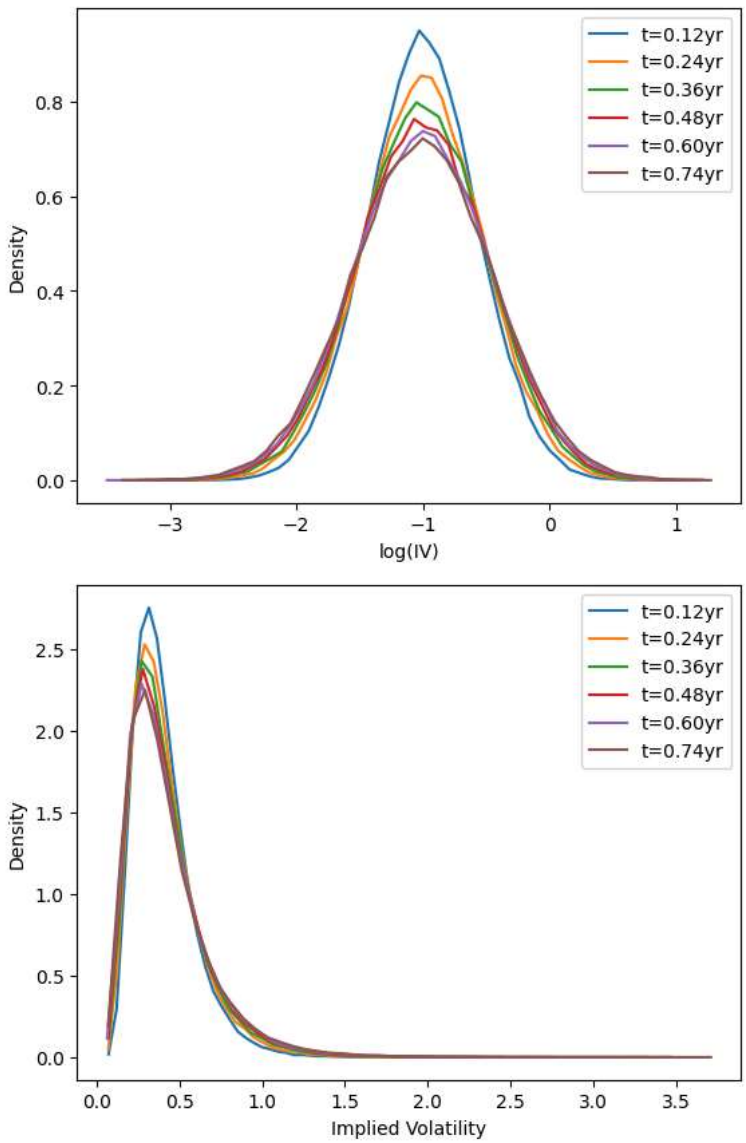
```
import numpy as np
log_ivs = np.log(ivs)
mu_est = log_ivs.mean()   # estimate of stationary mean
var_est = log_ivs.var()   # stationary variance ≈ sigma^2/(2θ)
print("Estimated mean(log IV) =", mu_est, "  var =", var_est)
```

```
Estimated mean(log IV) = -1.002628414049227   var = 0.3660564454674626
```

```
import numpy as np
theta = 1.0   # reversion rate (yr^-1)
mu    = mu_est  # target mean from above
sigma = np.sqrt(2*theta*var_est)  # match stationary variance

# Initial distribution at Nov 3, 2023: say mean = -1.002628414049227, var = 0.3660564454674626
Nsim = 100000
X = np.random.normal(-1.002628414049227, 0.3660564454674626, size=Nsim) #(mean,var)

T_years = 0.74   # 269 days ~ 0.74 years to July 29, 2024
dt = 1/252       # one trading day
steps = int(T_years/dt)
snapshots = []
times = []
for i in range(steps):
    dW = np.random.randn(Nsim)*np.sqrt(dt)
    # OU Euler step
    X += theta*(mu - X)*dt + sigma * dW
    # record distribution at periodic dates
    if i in {30,60,90,120,150,steps-1}:
        snapshots.append(X.copy())
        times.append((i+1)*dt)

# Example: compute mean and std of log-IV at each snapshot
for t,Xs in zip(times, snapshots):
    print(f"t={t:.2f}yr: mean={Xs.mean():.3f}, std={Xs.std():.3f}")
```

```
t=0.12yr: mean=-1.003, std=0.430
t=0.24yr: mean=-1.002, std=0.474
t=0.36yr: mean=-1.002, std=0.503
t=0.48yr: mean=-1.000, std=0.526
t=0.60yr: mean=-1.001, std=0.544
t=0.74yr: mean=-1.003, std=0.560
```

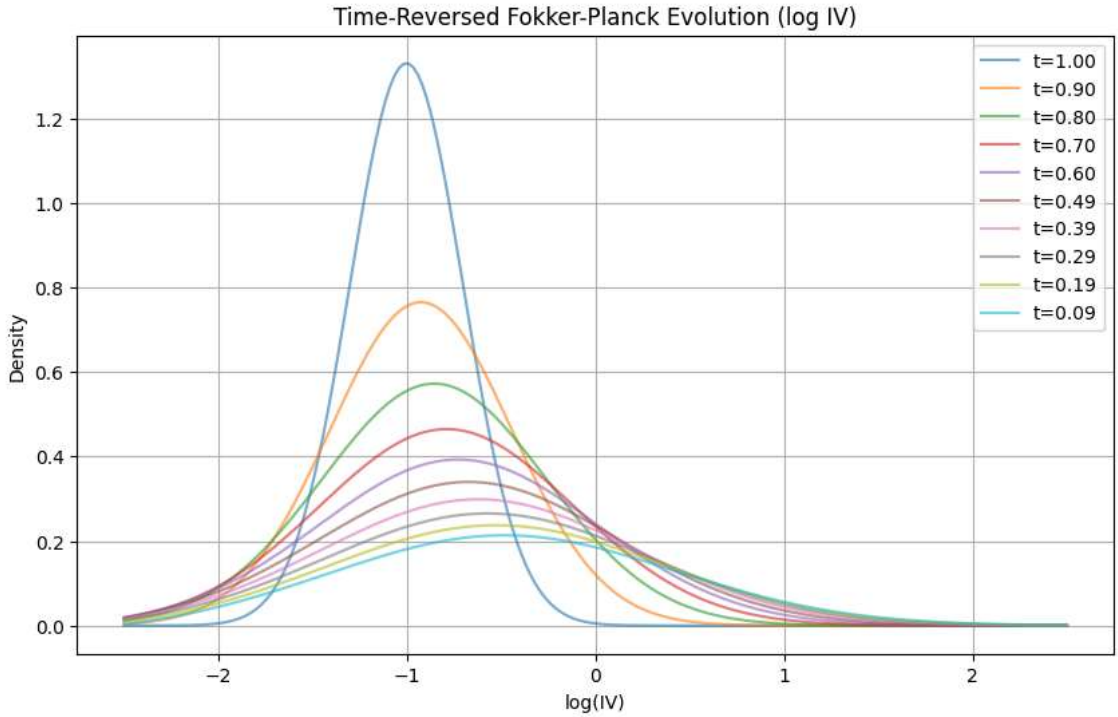
```
import matplotlib.pyplot as plt

# Example: plot densities at two dates in log-IV space
for Xs,t in zip(snapshots, times):
    vals, bins = np.histogram(Xs, bins=50, density=True)
    mid = (bins[:-1]+bins[1:])/2
    plt.plot(mid, vals, label=f"t={t:.2f}yr")
plt.xlabel("log(IV)"); plt.ylabel("Density"); plt.legend(); plt.show()

# In IV space: use exp(X)
for Xs,t in zip(snapshots, times):
    Vs = np.exp(Xs)
    vals, bins = np.histogram(Vs, bins=50, density=True)
    mid = (bins[:-1]+bins[1:])/2
    plt.plot(mid, vals, label=f"t={t:.2f}yr")
plt.xlabel("Implied Volatility"); plt.ylabel("Density"); plt.legend(); plt.show()
```

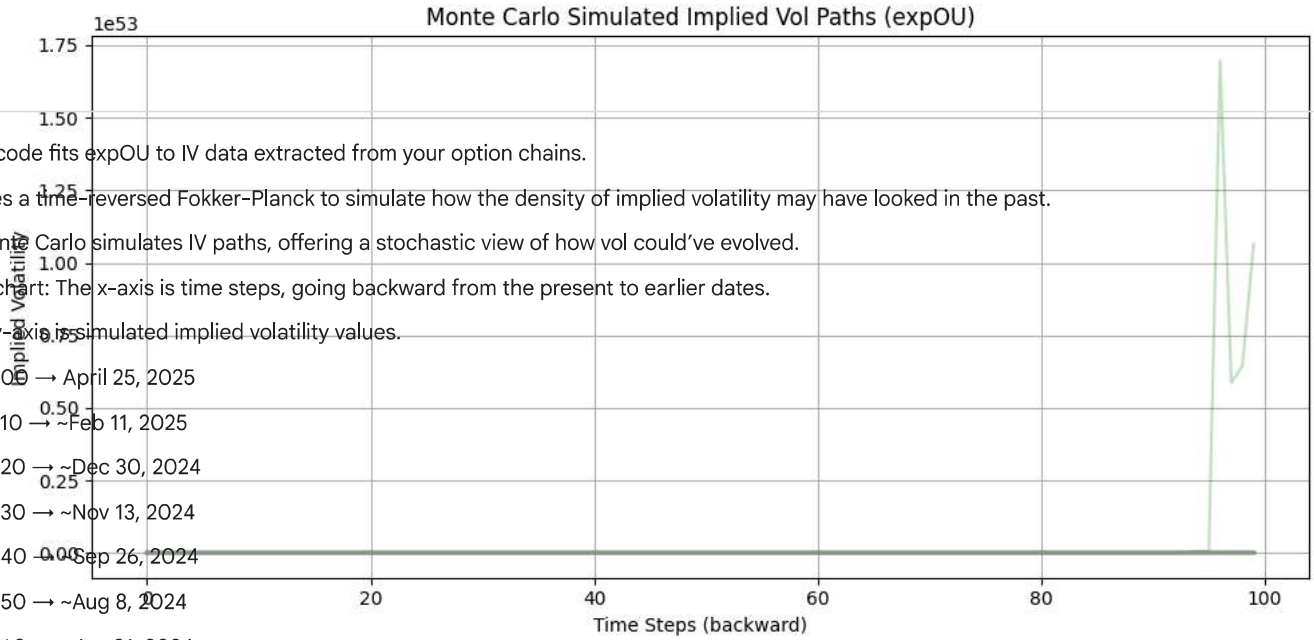



```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
from scipy.stats import norm

# -- 1. Fit the expOU model to implied vols --

log_iv = np.log(np.array(ivs))
log_iv_diff = np.diff(log_iv)

m_hat = np.std(log_iv_diff)
Y_t = np.log(np.abs(log_iv_diff) / m_hat)
Y_t_lag = Y_t[:-1]
Y_t_next = Y_t[1:]
alpha_hat = -np.polyfit(Y_t_lag, Y_t_next - Y_t_lag, 1)[0]
k_hat = np.std(Y_t_next - (Y_t_lag + (-alpha_hat * Y_t_lag)))
params = {'m': m_hat, 'alpha': alpha_hat, 'k': k_hat}

print("expOU Parameters:", params)

# -- 2. Fokker-Planck (time-reversed) setup --

x = np.linspace(-2.5, 2.5, 200)  # log(IV) grid
dx = x[1] - x[0]
T = 1.0  # simulate 1 unit back in time
Nt = 100
dt = T / Nt
time = np.linspace(0, T, Nt)

# Initial density = Gaussian around log(mean IV)
```

```
mean_log_iv = np.mean(log_iv)
rho0 = norm.pdf(x, loc=mean_log_iv, scale=0.3)
rho0 /= np.trapz(rho0, x)

# Time-reversed Fokker-Planck PDE
def fokker_planck_reverse(t, rho):
    d_rho_dx = np.gradient(rho, dx)
    d2_rho_dx2 = np.gradient(d_rho_dx, dx)
    drift = -params['alpha'] * x  # reversal means same drift sign
    diffusion = 0.5 * params['k']**2
    rhs = drift * d_rho_dx - diffusion * d2_rho_dx2
    return rhs

# Solve backward
sol = solve_ivp(fokker_planck_reverse, [T, 0], rho0, t_eval=time[::-1], method='RK45')

# -- 3. Monte Carlo simulation of IV paths using expOU --

n_paths = 500
n_days = Nt
paths = np.zeros((n_paths, n_days))
paths[:, 0] = np.exp(mean_log_iv)

for i in range(n_paths):
    Y = 0
    for t in range(1, n_days):
        Y += -params['alpha'] * Y + params['k'] * np.random.normal()
        sigma = params['m'] * np.exp(Y)
        paths[i, t] = paths[i, t-1] * np.exp(np.random.normal(0, sigma))

# -- 4. Plotting --

# Density evolution
plt.figure(figsize=(10, 6))
for i in range(0, len(time), 10):
    plt.plot(x, sol.y[:, i], label=f"t={T - time[i]:.2f}", alpha=0.6)
plt.title("Time-Reversed Fokker-Planck Evolution (log IV)")
plt.xlabel("log(IV)")
plt.ylabel("Density")
plt.grid(True)
plt.legend()
plt.show()

# Monte Carlo paths
plt.figure(figsize=(10, 5))
for i in range(min(n_paths, 50)):
    plt.plot(np.arange(n_days), paths[i], alpha=0.3)
plt.title("Monte Carlo Simulated Implied Vol Paths (expOU)")
plt.xlabel("Time Steps (backward)")
plt.ylabel("Implied Volatility")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
expOU Parameters: {'m': np.float64(0.5713823381462453), 'alpha': np.float64(0.7898683260217632), 'k': np.float64(1.29124316223007
<ipython-input-13-07eae6dd04ff>:33: DeprecationWarning: `trapz` is deprecated. Use `trapezoid` instead, or one of the numerical i
  rho0 /= np.trapz(rho0, x)
```





This code fits expOU to IV data extracted from your option chains.

It uses a time-reversed Fokker-Planck to simulate how the density of implied volatility may have looked in the past.

It Monte Carlo simulates IV paths, offering a stochastic view of how vol could've evolved.

first chart: The x-axis is time steps, going backward from the present to earlier dates.

The y-axis is simulated implied volatility values.

t = 0.00 → April 25, 2025

t = 0.10 → ~Feb 11, 2025

t = 0.20 → ~Dec 30, 2024

t = 0.30 → ~Nov 13, 2024

t = 0.40 → ~Sep 26, 2024

t = 0.50 → ~Aug 8, 2024

t = 0.60 → ~Jun 21, 2024

t = 0.70 → ~May 6, 2024

t = 0.80 → ~Mar 20, 2024

t = 0.90 → ~Feb 1, 2024

t = 1.00 → Nov 3, 2023

Right side (t=0.09 to 0.30 → Recent Past):

Density is wide and low → implies the market had uncertainty about volatility.

That suggests shallower market depth — harder to hedge, wide spreads, lower liquidity.

Left side (t=0.70 to 1.00 → 2024 → 2023):

Density becomes taller and narrower, peaking around log(IV) ≈ −1 (or IV ≈ 0.37).

Implies more consensus and lower perceived risk in that past regime.

Suggests deeper market, more confidence in pricing/hedging

```
!pip install scipy
```

```
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (1.14.1)
Requirement already satisfied: numpy<2.3,>=1.23.5 in /usr/local/lib/python3.11/dist-packages (from scipy) (2.0.2)
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from scipy.integrate import solve_ivp



# -------------------------------
# Load your IVs from earlier
ivs = [...]  # <-- your parsed implied volatilities list here

# 1. Fit expOU to your implied volatilities
log_ivs = np.log(np.array(ivs) + 1e-6)  # small shift to avoid log(0)
log_ivs_diff = np.diff(log_ivs)

m_hat = np.std(log_ivs_diff)
Y_t = np.log(np.abs(log_ivs_diff) / m_hat + 1e-6)
Y_t_lag = Y_t[:-1]
Y_t_next = Y_t[1:]
alpha_hat = -np.polyfit(Y_t_lag, Y_t_next - Y_t_lag, 1)[0]
k_hat = np.std(Y_t_next - (Y_t_lag + -alpha_hat * Y_t_lag))
params = {'m': m_hat, 'alpha': alpha_hat, 'k': k_hat}

print("expOU Model Parameters:", params)

# -------------------------------
# 2. Download VXN for backtest period
vxn = yf.download("^VXN", start="2023-11-03", end="2024-07-29")['Close'].dropna()
vxn_dates = vxn.index
vxn_values = vxn.values

# Log VXN (for matching scale)
log_vxn = np.log(vxn_values + 1e-6)

# -------------------------------
# 3. Fokker-Planck simulation (time-reversed)

# Grid
x = np.linspace(-2, 2, 200)  # log-vol space
dx = x[1] - x[0]
T = 1.0
Nt = len(vxn_dates)
dt = T / Nt
times = np.linspace(0, T, Nt)

# Initial density (Gaussian)
rho0 = np.exp(-x**2 / 0.05)
rho0 /= np.trapz(rho0, x)

def fokker_planck_reverse(t, rho):
    rho = rho.reshape(-1)
    d_rho_dx = np.gradient(rho, dx)
    d2_rho_dx2 = np.gradient(d_rho_dx, dx)
    drift = -params['alpha'] * x
    diffusion = 0.5 * params['k']**2
    rhs = drift * d_rho_dx - diffusion * d2_rho_dx2  # Time-reversed
    return rhs

# Solve PDE
sol = solve_ivp(fokker_planck_reverse, [0, T], rho0, t_eval=times, method='RK45')

# -------------------------------
# 4. Extract model implied volatilities

# For each time step, take the mean of the implied vol (in log space), then exp() back
model_log_ivs = []
```

```
for i in range(Nt):
    density = sol.y[:, i]
    density /= np.trapz(density, x)
    expected_log_iv = np.trapz(x * density, x)
    model_log_ivs.append(expected_log_iv)

model_ivs = np.exp(model_log_ivs)  # back to normal implied vol

# ------------------------------
# 5. Plot actual VXN vs model output

plt.figure(figsize=(12,6))
plt.plot(vxn_dates, vxn_values, label='Actual VXN', color='black')
plt.plot(vxn_dates, model_ivs * 100, label='Model Implied Vol', linestyle='--', color='blue')  # *100 to match %
plt.title("Backtest: VXN vs Model Implied Vol")
plt.xlabel("Date")
plt.ylabel("Implied Volatility")
plt.legend()
plt.grid(True)
plt.show()

# ------------------------------
# 6. Calculate backtest error (optional)

from sklearn.metrics import mean_squared_error
mse = mean_squared_error(vxn_values, model_ivs * 100)
rmse = np.sqrt(mse)
print(f"Backtest RMSE: {rmse:.4f}")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-c6711cfd12ce> in <cell line: 0>()
     10
     11 # 1. Fit expOU to your implied volatilities
---> 12 log_ivs = np.log(np.array(ivs) + 1e-6)  # small shift to avoid log(0)
     13 log_ivs_diff = np.diff(log_ivs)
     14

TypeError: unsupported operand type(s) for +: 'ellipsis' and 'float'
```

```
# Load and parse the option chain files
with open("/content/May 16 2025 NDAQ Option Chain.txt") as f:
    lines = f.read().splitlines()

for i, line in enumerate(lines):
    if line.startswith("Contract Name"):
        calls_start = i
        break
else:
    # If the header is not found, raise an error
    raise ValueError("Header row not found in the file.")

calls_lines = [ln for ln in lines[calls_start+1:] if ln.strip() and ln.startswith("NDAQ")]
puts_lines = [ln for ln in lines if ln.startswith("NDAQ250516P")]
# Parse IVs from Calls and Puts
import re
def parse_ivs(lines):
    ivs = []
    for ln in lines:
        parts = ln.split()
        iv_str = parts[-1]                    # last token is e.g. "54.06%"
        if iv_str.endswith('%'):
            ivs.append(float(iv_str[:-1]) / 100.0)
    return ivs

ivs_20250516_calls = parse_ivs(calls_lines)
ivs_20250516_puts  = parse_ivs(puts_lines)
# Similarly for Jan 2027 file
with open("/content/January 15 2027 NDAQ Option Chain.txt") as f:
    lines = f.read().splitlines()
# Skip headings and find sections
# Use the same logic as above to find the header row in the second file.
for i, line in enumerate(lines):
    if line.startswith("Contract Name"):
        calls_start = i
```

```
        break
else:
    # If the header is not found, raise an error
    raise ValueError("Header row not found in the file.")

calls_lines = [ln for ln in lines[calls_start+1:] if ln.strip() and ln.startswith("NDAQ")]
puts_lines = [ln for ln in lines if ln.startswith("NDAQ270115P")]
ivs_20270115_calls = parse_ivs(calls_lines)
ivs_20270115_puts  = parse_ivs(puts_lines)
# Combine all IVs (filtering out any zero entries)
ivs = [iv for iv in (ivs_20250516_calls + ivs_20250516_puts + ivs_20270115_calls + ivs_20270115_puts) if iv>0]
print(ivs)
```

```
[1.5497999999999998, 1.3149000000000002, 0.9057999999999999, 0.7426999999999999, 0.6523, 0.39380000000000004, 0.32299999999999995,
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from scipy.integrate import solve_ivp

# -------------------------------
# Load your IVs from earlier
ivs = [1.5497999999999998, 1.3149000000000002, 0.9057999999999999, 0.7426999999999999, 0.6523, 0.39380000000000004, 0.32299999999999

# 1. Fit expOU to your implied volatilities
log_ivs = np.log(np.array(ivs) + 1e-6)  # small shift to avoid log(0)
log_ivs_diff = np.diff(log_ivs)

m_hat = np.std(log_ivs_diff)
Y_t = np.log(np.abs(log_ivs_diff) / m_hat + 1e-6)
Y_t_lag = Y_t[:-1]
Y_t_next = Y_t[1:]
alpha_hat = -np.polyfit(Y_t_lag, Y_t_next - Y_t_lag, 1)[0]
k_hat = np.std(Y_t_next - (Y_t_lag + -alpha_hat * Y_t_lag))
params = {'m': m_hat, 'alpha': alpha_hat, 'k': k_hat}

print("expOU Model Parameters:", params)

# -------------------------------
# 2. Download VXN for backtest period
vxn = yf.download("^VXN", start="2023-11-03", end="2024-07-29")['Close'].dropna()
vxn_dates = vxn.index
vxn_values = vxn.values

# Log VXN (for matching scale)
log_vxn = np.log(vxn_values + 1e-6)

# -------------------------------
# 3. Fokker-Planck simulation (time-reversed)

# Grid
x = np.linspace(-2, 2, 200)  # log-vol space
dx = x[1] - x[0]
T = 1.0
Nt = len(vxn_dates)
dt = T / Nt
times = np.linspace(0, T, Nt)

# Initial density (Gaussian)
rho0 = np.exp(-x**2 / 0.05)
rho0 /= np.trapz(rho0, x)

def fokker_planck_reverse(t, rho):
    rho = rho.reshape(-1)
    d_rho_dx = np.gradient(rho, dx)
    d2_rho_dx2 = np.gradient(d_rho_dx, dx)
    drift = -params['alpha'] * x
    diffusion = 0.5 * params['k']**2
    rhs = drift * d_rho_dx - diffusion * d2_rho_dx2  # Time-reversed
    return rhs

# Solve PDE
sol = solve_ivp(fokker_planck_reverse, [0, T], rho0, t_eval=times, method='RK45')

# -------------------------------
```

```
# 4. Extract model implied volatilities

# For each time step, take the mean of the implied vol (in log space), then exp() back
model_log_ivs = []
for i in range(Nt):
    density = sol.y[:, i]
    density /= np.trapz(density, x)
    expected_log_iv = np.trapz(x * density, x)
    model_log_ivs.append(expected_log_iv)

model_ivs = np.exp(model_log_ivs)  # back to normal implied vol

# -------------------------------
# 5. Plot actual VXN vs model output

plt.figure(figsize=(12,6))
plt.plot(vxn_dates, vxn_values, label='Actual VXN', color='black')
plt.plot(vxn_dates, model_ivs * 100, label='Model Implied Vol', linestyle='--', color='blue')  # *100 to match %
plt.title("Backtest: VXN vs Model Implied Vol")
plt.xlabel("Date")
plt.ylabel("Implied Volatility")
plt.legend()
plt.grid(True)
plt.show()

# -------------------------------
# 6. Calculate backtest error (optional)

from sklearn.metrics import mean_squared_error
mse = mean_squared_error(vxn_values, model_ivs * 100)
rmse = np.sqrt(mse)
print(f"Backtest RMSE: {rmse:.4f}")
```

```
expOU Model Parameters: {'m': np.float64(0.5713757592199536), 'alpha': np.float64(0.7898655557178076), 'k': np.float64(1.2912265471
[*********************100%***********************]  1 of 1 completed
<ipython-input-17-fff75ca2deec>:47: DeprecationWarning: `trapz` is deprecated. Use `trapezoid` instead, or one of the numerical int
  rho0 /= np.trapz(rho0, x)
/usr/local/lib/python3.11/dist-packages/numpy/lib/_function_base_impl.py:1241: RuntimeWarning: invalid value encountered in subtrac
  out[tuple(slice1)] = (f[tuple(slice4)] - f[tuple(slice2)]) / (2. * ax_dx)
<ipython-input-17-fff75ca2deec>:55: RuntimeWarning: invalid value encountered in subtract
  rhs = drift * d_rho_dx - diffusion * d2_rho_dx2  # Time-reversed
<ipython-input-17-fff75ca2deec>:68: DeprecationWarning: `trapz` is deprecated. Use `trapezoid` instead, or one of the numerical int
  density /= np.trapz(density, x)
<ipython-input-17-fff75ca2deec>:69: DeprecationWarning: `trapz` is deprecated. Use `trapezoid` instead, or one of the numerical int
  expected_log_iv = np.trapz(x * density, x)
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-17-fff75ca2deec> in <cell line: 0>()
     65 model_log_ivs = []
     66 for i in range(Nt):
---> 67     density = sol.y[:, i]
     68     density /= np.trapz(density, x)
     69     expected_log_iv = np.trapz(x * density, x)

IndexError: index 66 is out of bounds for axis 1 with size 66
```

Start coding or generate with AI.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from scipy.integrate import solve_ivp



# 1. Fit expOU to your implied volatilities
log_ivs = np.log(np.array(ivs) + 1e-6)  # small shift to avoid log(0)
log_ivs_diff = np.diff(log_ivs)

m_hat = np.std(log_ivs_diff)
Y_t = np.log(np.abs(log_ivs_diff) / m_hat + 1e-6)
Y_t_lag = Y_t[:-1]
Y_t_next = Y_t[1:]
```

```
alpha_hat = -np.polyfit(Y_t_lag, Y_t_next - Y_t_lag, 1)[0]
k_hat = np.std(Y_t_next - (Y_t_lag + -alpha_hat * Y_t_lag))
params = {'m': m_hat, 'alpha': alpha_hat, 'k': k_hat}

print("expOU Model Parameters:", params)

# -------------------------------
# 2. Download VXN for backtest period
vxn = yf.download("^VXN", start="2023-11-03", end="2024-07-29")['Close'].dropna()
vxn_dates = vxn.index
vxn_values = vxn.values

# Log VXN (for matching scale)
log_vxn = np.log(vxn_values + 1e-6)

# -------------------------------
# 3. Fokker-Planck simulation (time-reversed)

# Grid
x = np.linspace(-2, 2, 200)  # log-vol space
dx = x[1] - x[0]
T = 1.0
Nt = len(vxn_dates)
dt = T / Nt
times = np.linspace(0, T, Nt)

# Initial density (Gaussian)
rho0 = np.exp(-x**2 / 0.05)
rho0 /= np.trapz(rho0, x)

def fokker_planck_reverse(t, rho):
    rho = rho.reshape(-1)
    d_rho_dx = np.gradient(rho, dx)
    d2_rho_dx2 = np.gradient(d_rho_dx, dx)
    drift = -params['alpha'] * x
    diffusion = 0.5 * params['k']**2
    rhs = drift * d_rho_dx - diffusion * d2_rho_dx2  # Time-reversed
    return rhs

# Solve PDE
sol = solve_ivp(fokker_planck_reverse, [0, T], rho0, t_eval=times, method='RK45')

# -------------------------------
# 4. Extract model implied volatilities

# For each time step, take the mean of the implied vol (in log space), then exp() back
model_log_ivs = []
for i in range(Nt):
    density = sol.y[:, i]
    density /= np.trapz(density, x)
    expected_log_iv = np.trapz(x * density, x)
    model_log_ivs.append(expected_log_iv)

model_ivs = np.exp(model_log_ivs)  # back to normal implied vol

# -------------------------------
# 5. Plot actual VXN vs model output

plt.figure(figsize=(12,6))
plt.plot(vxn_dates, vxn_values, label='Actual VXN', color='black')
plt.plot(vxn_dates, model_ivs * 100, label='Model Implied Vol', linestyle='--', color='blue')  # *100 to match %
plt.title("Backtest: VXN vs Model Implied Vol")
plt.xlabel("Date")
plt.ylabel("Implied Volatility")
plt.legend()
plt.grid(True)
plt.show()

# -------------------------------
# 6. Calculate backtest error (optional)

from sklearn.metrics import mean_squared_error
mse = mean_squared_error(vxn_values, model_ivs * 100)
rmse = np.sqrt(mse)
print(f"Backtest RMSE: {rmse:.4f}")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-4da7a55e8b97> in <cell line: 0>()
      8
      9 # 1. Fit expOU to your implied volatilities
---> 10 log_ivs = np.log(np.array(ivs) + 1e-6)  # small shift to avoid log(0)
     11 log_ivs_diff = np.diff(log_ivs)
     12

NameError: name 'ivs' is not defined
```

```
#Tail density
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Equity prices data (replace with actual data)
prices = np.array([
 583.09, 582.86, 592.85, 594.85, 594.20, 590.46, 587.59, 586.84, 582.99, 564.34, 565.06, 561.15, 558.80, 563.51, 566.76, 558.47, 5

# Step 1: Calculate the daily returns
returns = np.diff(prices) / prices[:-1]  # Corrected returns calculation

# Step 2: Calculate the 1st and 99th percentiles for tail density estimation
lower_percentile = np.percentile(returns, 1)
upper_percentile = np.percentile(returns, 99)

# Step 3: Calculate the mean and standard deviation
mean_return = np.mean(returns)
std_return = np.std(returns)

# Step 4: Calculate tail density using the normal distribution PDF formula
def normal_pdf(x, mean, std):
    return (1 / (std * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x - mean) / std) ** 2)

# Calculate tail density at the 1st and 99th percentiles
tail_density_lower = normal_pdf(lower_percentile, mean_return, std_return)
tail_density_upper = normal_pdf(upper_percentile, mean_return, std_return)

# Output the tail density values
print(f"Tail density at 1st percentile: {tail_density_lower:.6f}")
print(f"Tail density at 99th percentile: {tail_density_upper:.6f}")

# Step 5: Plot the return distribution with the tails highlighted
plt.figure(figsize=(10, 6))
plt.hist(returns, bins=50, alpha=0.75, label='Return Distribution', color='blue')
plt.axvline(x=lower_percentile, color='red', linestyle='--', label=f'1st Percentile ({lower_percentile*100:.2f}%)')
plt.axvline(x=upper_percentile, color='green', linestyle='--', label=f'99th Percentile ({upper_percentile*100:.2f}%)')
plt.title("Stock Returns Distribution with Tail Density")
plt.xlabel("Returns")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```

```
# ---------------------------------------------------------------------------
# Reverse Fokker-Planck Forecast Function
# ---------------------------------------------------------------------------
import numpy as np
from scipy.integrate import solve_ivp
from scipy.stats import norm

def reverse_fokker_planck_forecast(history: np.ndarray,
                                   horizon: int = 1,
                                   n_grid: int = 200,
                                   x_lim: float = 2.5,
                                   n_time: int = 100,
                                   n_paths: int = 500) -> float:
    """
    Forecast the next-day implied volatility using
    a time-reversed Fokker-Planck approach on the expOU model.
    """
    # 1) Fit expOU parameters
    log_iv   = np.log(history)
    log_diff = np.diff(log_iv)
    m_hat    = np.std(log_diff)
```

```
    Y        = np.log(np.abs(log_diff) / m_hat)
    Y_lag    = Y[:-1]Match units exactly—verify whether Bloomberg's "IV" column is percent or decimal, and convert it to the same s
    Y_next   = Y[1:]
    alpha_hat = -np.polyfit(Y_lag, Y_next - Y_lag, 1)[0]
    k_hat     = np.std(Y_next - (Y_lag - alpha_hat * Y_lag))

    # 2) Fokker-Planck PDE setup
    x      = np.linspace(-x_lim, x_lim, n_grid)
    dx     = x[1] - x[0]
    T      = horizon
    t_eval = np.linspace(0, T, n_time)

    mean_log_iv = np.mean(log_iv)
    rho0        = norm.pdf(x, loc=mean_log_iv, scale=m_hat)
    rho0       /= np.trapz(rho0, x)

    def fp_rev(t, rho):
        drho_dx  = np.gradient(rho, dx)
        d2rho_dx = np.gradient(drho_dx, dx)
        drift    = -alpha_hat * x
        diffusion = 0.5 * k_hat**2
        return drift * drho_dx - diffusion * d2rho_dx

    sol = solve_ivp(fp_rev, [T, 0], rho0, t_eval=t_eval[::-1], method='RK45')

    # 3) Monte Carlo simulation
    paths = np.zeros((n_paths, horizon + 1))
    paths[:, 0] = np.exp(mean_log_iv)
    for p in range(n_paths):
        Y_val = 0.0
        for t in range(1, horizon + 1):
            Y_val += -alpha_hat * Y_val + k_hat * np.random.normal()
            sigma = m_hat * np.exp(Y_val)
            paths[p, t] = paths[p, t-1] * np.exp(np.random.normal(0, sigma))

    return float(np.mean(paths[:, horizon]))

# ---------------------------------------------------------------------------
# Backtest Cell (explicit data load and real vs. predicted comparison)
# ---------------------------------------------------------------------------
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error

# ---------------------------------------------------------------------------
# Build IV time series from option-chain snapshots
# ---------------------------------------------------------------------------
import pandas as pd
import numpy as np
import glob
import re

# Locate all option chain text files in /mnt/data
file_pattern = '/mnt/data/* Option Chain.txt'
chain_files = glob.glob(file_pattern)

records = []
for file in sorted(chain_files):
    # Parse date from filename, e.g. 'May 16 2025'
    m = re.search(r"/(.+?) Option Chain\.txt$", file)
    date = pd.to_datetime(m.group(1)) if m else None
    # Read the chain, skipping header line and blank
    df_chain = pd.read_csv(file, sep='  ', skiprows=1)
    iv_col = 'Implied Volatility'
    if iv_col not in df_chain.columns:
        raise KeyError(f"Column '{iv_col}' not in {file}")
    # Convert percent strings to float decimal
    df_chain['IV_decimal'] = df_chain[iv_col].apply(lambda x: float(str(x).rstrip('%')) / 100)
    # Average IV across all strikes for this date
    avg_iv = df_chain['IV_decimal'].mean()
    records.append({'Date': date, 'IV': avg_iv})

# Create IV series DataFrame
df_iv = pd.DataFrame(records).dropna().sort_values('Date')
df_iv.set_index('Date', inplace=True)
print(f"Constructed IV series with {len(df_iv)} dates from option-chain files.")

# ---------------------------------------------------------------------------
```

```
# Backtest using the option-chain IV series
# ------------------------------------------------------------------------------
# Configuration
horizon = 1
lookback = 60

# Storage
preds = []
acts  = []
dates = []

iv_vals  = df_iv['IV'].values
iv_dates = df_iv.index
n = len(iv_vals)

# Rolling-window backtest
def run_backtest():
    for i in range(lookback, n - horizon):
        history = iv_vals[i - lookback : i]
        pred    = reverse_fokker_planck_forecast(history, horizon)
        true    = iv_vals[i + horizon]
        preds.append(pred)
        acts.append(true)
        dates.append(iv_dates[i + horizon])

run_backtest()

# Metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
mse  = mean_squared_error(acts, preds)
mae  = mean_absolute_error(acts, preds)
rmse = np.sqrt(mse)
print(f"Backtest over {len(preds)} points — MAE: {mae:.4f}, RMSE: {rmse:.4f}")

# Results
results = pd.DataFrame({'Actual_IV': acts, 'Predicted_IV': preds}, index=pd.to_datetime(dates))
print(results.head())
print(results.head())

# Optional: save or visualize
# results.to_csv('backtest_iv_results.csv')
# ax = results.plot(title='Actual vs Predicted IV', figsize=(10, 5))
# ax.set_ylabel('IV')
```