```
#P1
import yfinance as yf
import pandas as pd
import datetime as dt
#P2
#given tickers
symbols = {
   "TSLA": "TSLA",
   "SPY": "SPY",
   "VIX": "^VIX"
}
Data_equity ={}
for symbol in symbols:
   ticker = yf.Ticker(symbol)

   hist = ticker.history(period = "1y")
   Data_equity[symbol] = hist
#consecutive days
day1 = "2026-02-12"
day2 = "2026-02-13"

def download_intraday(symbol, date):
   ticker = yf.Ticker(symbol)
   data = ticker.history(
      start=date,
      end=(pd.to_datetime(date) + pd.Timedelta(days=1)).strftime('%Y-%m-%d'),
```

```
        interval="5m"
    )
    return data
#Spot when downloaded
def get_spot_price(symbol):
    ticker = yf.Ticker(symbol)
    data = ticker.history(period="1d", interval="1m")
    return data["Close"].iloc[-1]
#3rd Friday of Month
def get_next_three_third_fridays(symbol):
    ticker = yf.Ticker(symbol)
    expirations = ticker.options

    third_fridays = []

    for exp in expirations:
        date = pd.to_datetime(exp)
        if date.weekday() == 4 and 15 <= date.day <= 21:
            third_fridays.append(exp)

    return third_fridays[:3]

#Options
def download_option_chain(symbol, expirations):
    ticker = yf.Ticker(symbol)
    option_data = {}
```

```
    for exp in expirations:
        opt = ticker.option_chain(exp)
        option_data[exp] = {
            "calls": opt.calls,
            "puts": opt.puts
        }

    return option_data

DATA1 = {}

for name, sym in symbols.items():
    DATA1[name] = {}
    DATA1[name]["intraday"] = download_intraday(sym, day1)
    DATA1[name]["spot_at_download"] = get_spot_price(sym)

# Options
DATA1["TSLA"]["expirations"] = get_next_three_third_fridays("TSLA")
DATA1["SPY"]["expirations"] = get_next_three_third_fridays("SPY")
DATA1["VIX"]["expirations"] = get_next_three_third_fridays("VIX")

DATA1["TSLA"]["options"] = download_option_chain("TSLA", DATA1["TSLA"]["expirations"])
DATA1["SPY"]["options"] = download_option_chain("SPY", DATA1["SPY"]["expirations"])
DATA1["VIX"]["options"] = download_option_chain("VIX", DATA1["VIX"]["expirations"])
```

```
DATA2 = {}

for name, sym in symbols.items():
    DATA2[name] = {}
    DATA2[name]["intraday"] = download_intraday(sym, day2)
    DATA2[name]["spot_at_download"] = get_spot_price(sym)

DATA2["TSLA"]["expirations"] = get_next_three_third_fridays("TSLA")
DATA2["SPY"]["expirations"] = get_next_three_third_fridays("SPY")
DATA2["VIX"]["expirations"] = get_next_three_third_fridays("VIX")

DATA2["TSLA"]["options"] = download_option_chain("TSLA", DATA2["TSLA"]["expirations"])
DATA2["SPY"]["options"] = download_option_chain("SPY", DATA2["SPY"]["expirations"])
DATA2["VIX"]["options"] = download_option_chain("VIX", DATA2["VIX"]["expirations"])

#P4
print("DATA1 Spot Prices:")
print("TSLA:", DATA1["TSLA"]["spot_at_download"])
print("SPY :", DATA1["SPY"]["spot_at_download"])
print("VIX :", DATA1["VIX"]["spot_at_download"])

print("\nDATA2 Spot Prices:")
print("TSLA:", DATA2["TSLA"]["spot_at_download"])
print("SPY :", DATA2["SPY"]["spot_at_download"])
print("VIX :", DATA2["VIX"]["spot_at_download"])
```

```
def compute_ttm(data_dict):

    # download
    download_time = data_dict["intraday"].index[-1]

    # timezone
    if download_time.tzinfo is not None:
        download_time = download_time.tz_localize(None)

    ttm_results = {}

    for exp in data_dict["expirations"]:

        expiration_date = pd.to_datetime(exp)

        # time of expire
        expiration_datetime = expiration_date.replace(hour=16, minute=0)

        # Compute TTM in years
        T = (expiration_datetime - download_time).total_seconds() / (365 * 24 * 60 * 60)

        ttm_results[exp] = T

    return ttm_results
```

```
print("TIME TO MATURITY (DATA1)")

for asset in ["TSLA", "SPY", "VIX"]:

    ttm_results = compute_ttm(DATA1[asset])

    print(f"\n{asset}:")
    for exp, T in ttm_results.items():
        print(f"Expiration {exp} -> TTM = {T:.6f} years")



print("TIME TO MATURITY (DATA2)")

for asset in ["TSLA", "SPY", "VIX"]:

    ttm_results = compute_ttm(DATA2[asset])

    print(f"\n{asset}:")
    for exp, T in ttm_results.items():
        print(f"Expiration {exp} -> TTM = {T:.6f} years")
#P5
import numpy as np
from scipy.stats import norm

#r = 3.64% based on link
#BS
```

```
def black_scholes_price(S0, K, T, r, sigma, option_type):
    """
    Inputs:
        S0 : stock price at download
        K  : strike price
        T  : time to maturity (in years)
        r  : annual risk-free rate (decimal)
        sigma : annual volatility (decimal)
        option_type : 'call' or 'put'
    """
    d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    # Call or Put
    if option_type.lower() == "call":
        price = S0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    else option_type.lower() == "put":
        price = K * np.exp(-r * T) * norm.cdf(-d2) - S0 * norm.cdf(-d1)
    return price
#P6
def bisection_iv(S, K, T, r, market_price, option_type="call",
                 tol=1e-6, max_iter=1000):

    low = 1e-6
    high = 5.0
```

```
    for _ in range(max_iter):
        #calc mid price
        mid = (low + high) / 2
        price = black_scholes_price(S, K, T, r, mid, option_type)

        diff = price - market_price

        if abs(diff) < tol:
            return mid

        if diff > 0:
            high = mid
        else:
            low = mid

    return mid

#IV using mid price
def compute_iv_for_asset(asset_name):

    exp = DATA1[asset_name]["expirations"][0]
    chain = DATA1[asset_name]["options"][exp]["calls"].copy()
    S0 = DATA1[asset_name]["spot_at_download"]
    T = compute_ttm(DATA1[asset_name])[exp]
    r = 0.0364
```

```
    iv_list = []
    for _, row in chain.iterrows():
        #calc mid, add strike, calc iv
        mid_price = (row["bid"] + row["ask"]) / 2
        K = row["strike"]
        iv = bisection_iv(S0, K, T, r, mid_price, "call")
        iv_list.append((K, iv))

    iv_df = pd.DataFrame(iv_list, columns=["strike","IV"])
    # Closest Strike to Spot
    atm_strike = iv_df.iloc[(iv_df["strike"] - S0).abs().argsort()[:1]]
    atm_iv = atm_strike["IV"].values[0]
    # Average
    avg_iv = iv_df["IV"].mean()
    return atm_iv, avg_iv, iv_df

atm_iv_tsla, avg_iv_tsla, iv_df_tsla = compute_iv_for_asset("TSLA")
atm_iv_spy, avg_iv_spy, iv_df_spy = compute_iv_for_asset("SPY")

print("IMPLIED VOLATILITY RESULTS\n")

print("TSLA ATM IV :", round(atm_iv_tsla,6))
print("TSLA Avg IV :", round(avg_iv_tsla,6))
print()

print("SPY ATM IV  :", round(atm_iv_spy,6))
```

```
print("SPY Avg IV  :", round(avg_iv_spy,6))
#P7
def vega(S, K, T, r, sigma):

    d1 = (np.log(S/K) + (r + 0.5*sigma**2)*T) / (sigma*np.sqrt(T))
    return S * norm.pdf(d1) * np.sqrt(T)

def newton_iv(S, K, T, r, market_price,
              option_type="call",
              tol=1e-6,
              max_iter=100,
              sigma_init=0.3):

    sigma = sigma_init

    for _ in range(max_iter):

        price = black_scholes_price(S, K, T, r, sigma, option_type)
        diff = price - market_price

        if abs(diff) < tol:
            return sigma


        v = vega(S, K, T, r, sigma)
```

```
        if v == 0:
            break

        sigma = sigma - diff / v

        if sigma <= 0:
            sigma = 1e-6

    return sigma

import time
def compare_methods(asset_name):

    exp = DATA1[asset_name]["expirations"][0]
    chain = DATA1[asset_name]["options"][exp]["calls"].copy()

    S0 = DATA1[asset_name]["spot_at_download"]
    T = compute_ttm(DATA1[asset_name])[exp]
    r = 0.0364

    results = []
    #time to derive
    for _, row in chain.iterrows():

        mid = (row["bid"] + row["ask"]) / 2
        K = row["strike"]
```

```
        # Bisection
        start_bis = time.time()
        iv_bis = bisection_iv(S0, K, T, r, mid, "call")
        time_bis = time.time() - start_bis

        # Newton
        start_new = time.time()
        iv_new = newton_iv(S0, K, T, r, mid, "call")
        time_new = time.time() - start_new

        results.append([K, iv_bis, iv_new, time_bis, time_new])

    df = pd.DataFrame(results,
                columns=["Strike","IV_Bisection","IV_Newton",
                    "Time_Bisection","Time_Newton"])

    return df

df_tsla = compare_methods("TSLA")
df_spy  = compare_methods("SPY")

print("TSLA Timing (seconds):")
print("Avg Bisection:", df_tsla["Time_Bisection"].mean())
print("Avg Newton   :", df_tsla["Time_Newton"].mean())
```

```
print("\nSPY Timing (seconds):")
print("Avg Bisection:", df_spy["Time_Bisection"].mean())
print("Avg Newton   :", df_spy["Time_Newton"].mean())
#P8
def compute_full_iv_table(asset_name):

    results = []
    S0 = DATA1[asset_name]["spot_at_download"]
    r = 0.0364

    for exp in DATA1[asset_name]["expirations"]:

        T = compute_ttm(DATA1[asset_name])[exp]

        for option_type in ["calls", "puts"]:

            chain = DATA1[asset_name]["options"][exp][option_type].copy()

            iv_list = []

            for _, row in chain.iterrows():
                #without for some reason gives divide by zero
                if (row["bid"] > 0 and row["ask"] > 0 and
                    row["volume"] > 0 and
                    abs(row["strike"]/S0 - 1) < 0.3):  # within ±30%
```

```
        mid = (row["bid"] + row["ask"]) / 2
        K = row["strike"]

        try:
            iv = newton_iv(
                S0, K, T, r, mid,
                option_type="call" if option_type=="calls" else "put"
            )

            # Remove unrealistic IVs
            if 0 < iv < 3:
                iv_list.append((K, iv))

        except:
            continue

    if len(iv_list) == 0:
        continue

    iv_df = pd.DataFrame(iv_list, columns=["strike", "IV"])

    # ATM IV
    atm_row = iv_df.iloc[
        (iv_df["strike"] - S0).abs().argsort()[:1]
    ]
    atm_iv = atm_row["IV"].values[0]
```

```
        # Average IV
        avg_iv = iv_df["IV"].mean()

        results.append([
          asset_name,
          exp,
          option_type,
          atm_iv,
          avg_iv
        ])

  return pd.DataFrame(
    results,
    columns=["Asset", "Maturity", "OptionType", "ATM_IV", "Average_IV"]
  )



iv_table_tsla = compute_full_iv_table("TSLA")
iv_table_spy  = compute_full_iv_table("SPY")

final_iv_table = pd.concat([iv_table_tsla, iv_table_spy])

print(final_iv_table)

vix_value = DATA1["VIX"]["spot_at_download"]
```

```
print("Current VIX:", vix_value)
#P9
import numpy as np
import pandas as pd

def put_call_parity_test(asset_name, K_min=400, K_max=700):

    results = []

    S0 = DATA1[asset_name]["spot_at_download"]
    r = 0.0364

    ttm_map = compute_ttm(DATA1[asset_name])

    for exp in DATA1[asset_name]["expirations"]:

        T = ttm_map[exp]
        discount = np.exp(-r * T)

        calls = DATA1[asset_name]["options"][exp]["calls"]
        puts  = DATA1[asset_name]["options"][exp]["puts"]

        # Merge call and put on strike
        merged = pd.merge(
            calls, puts,
            on="strike",
```

```
    suffixes=("_call", "_put")
)

# 400 to 700
merged = merged[
    (merged["strike"] >= K_min) &
    (merged["strike"] <= K_max)
]

for _, row in merged.iterrows():

    if (row["bid_call"] > 0 and row["ask_call"] > 0 and
        row["bid_put"] > 0 and row["ask_put"] > 0):

        K = row["strike"]

        call_mid = (row["bid_call"] + row["ask_call"]) / 2
        put_mid  = (row["bid_put"] + row["ask_put"]) / 2

        # Put-Call Parity/ calc values
        synthetic_call = put_mid + S0 - K * discount
        synthetic_put  = call_mid - S0 + K * discount

        results.append([
            asset_name,
            exp,
```

```
                K,
                call_mid,
                synthetic_call,
                row["bid_call"],
                row["ask_call"],
                put_mid,
                synthetic_put,
                row["bid_put"],
                row["ask_put"]
            ])

    df = pd.DataFrame(
        results,
        columns=[
            "Asset","Maturity","Strike",
            "Call_Mid","Call_Calc",
            "Call_Bid","Call_Ask",
            "Put_Mid","Put_Calc",
            "Put_Bid","Put_Ask"
        ]
    )


    return df
parity_tsla = put_call_parity_test("TSLA", 400, 700)
parity_spy  = put_call_parity_test("SPY", 400, 700)
```

```
parity_table = pd.concat([parity_tsla, parity_spy], ignore_index=True)
print(parity_table)
#P10
import matplotlib.pyplot as plt
#Smile
def get_iv_smile(asset_name):

  S0 = DATA1[asset_name]["spot_at_download"]
  r = 0.0364

  smile_data = {}
  ttm_map = compute_ttm(DATA1[asset_name])

  for exp in DATA1[asset_name]["expirations"]:

    T = ttm_map[exp]
    calls = DATA1[asset_name]["options"][exp]["calls"].copy()

    strikes = []
    ivs = []

    for _, row in calls.iterrows():
      #non empty
      if (row["bid"] > 0 and row["ask"] > 0 and
        row["volume"] > 0 and
```

```
            abs(row["strike"]/S0 - 1) < 0.3):

            mid = (row["bid"] + row["ask"]) / 2
            K = row["strike"]

            try:
                iv = newton_iv(
                    S0, K, T, r, mid,
                    option_type="call"
                )

                if 0 < iv < 3:
                    strikes.append(K)
                    ivs.append(iv)

            except:
                continue

        smile_data[exp] = {
            "T": T,
            "strikes": strikes,
            "ivs": ivs
        }

    return smile_data
```

```
def plot_3_maturities(asset_name):

    smile_data = get_iv_smile(asset_name)

    # Sort expirations
    sorted_exps = sorted(
        smile_data.keys(),
        key=lambda exp: smile_data[exp]["T"]
    )

    # 3 maturities
    closest_3 = sorted_exps[:3]

    plt.figure(figsize=(10, 6))

    for exp in closest_3:
        strikes = smile_data[exp]["strikes"]
        ivs = smile_data[exp]["ivs"]
        T = smile_data[exp]["T"]

        if len(strikes) > 0:
            plt.scatter(strikes, ivs, label=f"{exp} (T={T:.3f})")

    plt.xlabel("Strike (K)")
    plt.ylabel("Implied Volatility")
```

```
    plt.title(f"{asset_name} Implied Volatility vs Strike (3 Closest Maturities)")
    plt.legend()
    plt.grid(True)
    plt.show()



plot_3_maturities("TSLA")
plot_3_maturities("SPY")
#11
import numpy as np
import pandas as pd
from scipy.stats import norm
#BS formula
def bs_greeks(S, K, T, r, sigma):
    if T <= 0 or sigma <= 0:
        return np.nan, np.nan, np.nan

    d1 = (np.log(S/K) + (r + 0.5*sigma**2)*T) / (sigma*np.sqrt(T))

    delta = norm.cdf(d1)
    gamma = norm.pdf(d1) / (S * sigma * np.sqrt(T))
    vega  = S * norm.pdf(d1) * np.sqrt(T)

    return delta, gamma, vega

#Partials
```

```
def numerical_greeks(S, K, T, r, sigma):
    hS = 0.01 * S
    hV = 1e-4

    C0 = black_scholes_price(S, K, T, r, sigma, "call")

    C_up = black_scholes_price(S + hS, K, T, r, sigma, "call")
    C_down = black_scholes_price(S - hS, K, T, r, sigma, "call")

    delta_num = (C_up - C_down) / (2 * hS)
    gamma_num = (C_up - 2 * C0 + C_down) / (hS**2)

    C_vol_up = black_scholes_price(S, K, T, r, sigma + hV, "call")
    C_vol_down = black_scholes_price(S, K, T, r, sigma - hV, "call")

    vega_num = (C_vol_up - C_vol_down) / (2 * hV)

    return delta_num, gamma_num, vega_num


def greek_table(asset_name, K_min=400, K_max=700):

    exp = DATA1[asset_name]["expirations"][1]
    S0 = DATA1[asset_name]["spot_at_download"]
    T = compute_ttm(DATA1[asset_name])[exp]
    r = 0.0364
```

```
chain = DATA1[asset_name]["options"][exp]["calls"].copy()

results = []

for _, row in chain.iterrows():

    K = row["strike"]

    #400 to 700
    if (K >= K_min and K <= K_max and
        row["bid"] > 0 and row["ask"] > 0 and
        row["volume"] > 0 and
        abs(K / S0 - 1) < 0.3):

        mid = (row["bid"] + row["ask"]) / 2

        try:
            iv = newton_iv(S0, K, T, r, mid, option_type="call")
        except:
            continue

        # limit variability
        if not (0 < iv < 3):
            continue
```

```
        # Use IV
        delta_a, gamma_a, vega_a = bs_greeks(S0, K, T, r, iv)
        delta_n, gamma_n, vega_n = numerical_greeks(S0, K, T, r, iv)

        results.append([
          K, iv,
          delta_a, delta_n,
          gamma_a, gamma_n,
          vega_a, vega_n
        ])

    return pd.DataFrame(
      results,
      columns=[
        "Strike", "IV",
        "Delta_BSl","Delta_Numerical",
        "Gamma_BS","Gamma_Numerical",
        "Vega_BS","Vega_Numerical"
      ]
    ).sort_values("Strike").reset_index(drop=True)


greeks_tsla = greek_table("TSLA", 400, 700)
greeks_spy  = greek_table("SPY", 400, 700)

print("TSLA Greeks (400 ≤ K ≤ 700):")
```

```
print(greeks_tsla.head())


print("\nSPY Greeks (400 ≤ K ≤ 700):")
print(greeks_spy.head())
#12
import numpy as np
import pandas as pd

def get_iv_surface_from_data1(asset_name, K_min=400, K_max=700):

    iv_dict = {}

    S0 = DATA1[asset_name]["spot_at_download"]
    r = 0.0364

    ttm1 = compute_ttm(DATA1[asset_name])

    for exp in DATA1[asset_name]["expirations"]:

        T = ttm1[exp]
        calls = DATA1[asset_name]["options"][exp]["calls"]

        iv_dict[exp] = {}

        for _, row in calls.iterrows():
```

```
        K = row["strike"]

        # 400 to 700
        if not (K_min <= K <= K_max):
            continue

        if row["bid"] > 0 and row["ask"] > 0 and row["volume"] > 0:

            mid = (row["bid"] + row["ask"]) / 2

            try:
                sigma = newton_iv(S0, K, T, r, mid, option_type="call")
            except Exception:
                continue

            # sigma bound
            if 0 < sigma < 3:
                iv_dict[exp][K] = sigma

    return iv_dict

#price for data2
def price_data2_with_old_iv(asset_name, r_data2, K_min=400, K_max=700):

    iv_surface = get_iv_surface_from_data1(asset_name, K_min=K_min, K_max=K_max)
```

```
S2 = DATA2[asset_name]["spot_at_download"]

results = []

ttm2 = compute_ttm(DATA2[asset_name])

for exp in DATA2[asset_name]["expirations"]:



    T2 = ttm2[exp]
    calls2 = DATA2[asset_name]["options"][exp]["calls"]

    for _, row in calls2.iterrows():

        K = row["strike"]

        # 400 to 700
        if not (K_min <= K <= K_max):
            continue

        if K in iv_surface[exp]:

            sigma_old = iv_surface[exp][K]

            model_price = black_scholes_price(
                S2, K, T2, r_data2, sigma_old, "call"
```

```
            )

            market_mid = (row["bid"] + row["ask"]) / 2

            results.append([
                asset_name,
                exp,
                K,
                sigma_old,
                model_price,
                market_mid
            ])

    return pd.DataFrame(
        results,
        columns=[
            "Asset","Maturity","Strike",
            "IV_from_DATA1",
            "ModelPrice_DATA2",
            "MarketMid_DATA2"
        ]
    )


r_data2 = 0.0364
```

```
pricing_tsla = price_data2_with_old_iv("TSLA", r_data2, K_min=400, K_max=700)
pricing_spy  = price_data2_with_old_iv("SPY",  r_data2, K_min=400, K_max=700)

final_pricing_table = pd.concat([pricing_tsla, pricing_spy], ignore_index=True)
print(final_pricing_table)
#Part 3
#x_t -> BTC reserves at time t
#y_t -> USDC reserves at time t
#p_t -> USDC per BTC
#s_t -> external market price
#gamma -> fee rate
#constant product rule -> x_t+1 * y_t+1 = x_t * y_t = k
#trade mechanic -> (x_t + (1-gamma)*(delta_x)) * (y_t - (delta_y)) = k



#case 1
#S_t+1 > p_t * (1/(1-gamma)) BTC cheaper in pool

#case 2
#S_t+1 < p_t * (1-gamma) BTC cheaper outside

#a Results on PDF
#b
import numpy as np
from scipy.stats import norm
```

```
#Givens
xt = 1000
yt = 1000
k = xt * yt
Pt = 1
St = 1
dt = 1/365
sigma = 0.2
gamma = 0.003



def lognormal_pdf(s):
   mu = -0.5 * sigma**2 * dt
   var = sigma**2 * dt
   return (1 / (s * np.sqrt(2*np.pi*var))) * \
        np.exp(-(np.log(s) - mu)**2 / (2*var))
#results from part a
def delta_y(s):
   return np.sqrt(k * (1-gamma) * s) - yt


def delta_x(s):
   return np.sqrt(k * ((1-gamma) / s)) - xt


upper_trigger = 1/(1-gamma)
lower_trigger = 1-gamma
```

```
s_max = 3
N = 10000
s_grid = np.linspace(1e-6, s_max, N)

integrand = np.zeros_like(s_grid)

#cases
for i, s in enumerate(s_grid):
  if s > upper_trigger:
    integrand[i] = gamma * delta_y(s) * lognormal_pdf(s)
  elif s < lower_trigger:
    integrand[i] = gamma * delta_x(s) * s * lognormal_pdf(s)
  else:
    integrand[i] = 0

expected_revenue = np.trapz(integrand, s_grid)

print("Expected one-step fee revenue =", expected_revenue)
#3.C
import numpy as np
import pandas as pd



# Givens
xt = 1000.0
yt = 1000.0
```

```
k = xt * yt


dt = 1/365
sigmas = [0.2, 0.6, 1.0]
gammas = [0.001, 0.003, 0.01]


s_max = 3.0
N = 10000
s_grid = np.linspace(1e-6, s_max, N)



def lognormal_pdf(s, sigma, dt):
   mu = -0.5 * sigma**2 * dt
   var = sigma**2 * dt
   return (1.0 / (s * np.sqrt(2*np.pi*var))) * np.exp(-(np.log(s) - mu)**2 / (2*var))


def delta_y(s, gamma):
   # from part (a)
   return np.sqrt(k * (1 - gamma) * s) - yt


def delta_x(s, gamma):
   # from part (a)
   return np.sqrt(k * ((1 - gamma) / s)) - xt


def expected_one_step_fee_revenue(sigma, gamma):
   upper_trigger = 1.0 / (1.0 - gamma)
```

```
    lower_trigger = 1.0 - gamma

    pdf_vals = lognormal_pdf(s_grid, sigma, dt)

    # piecewise integrand
    integrand = np.zeros_like(s_grid)

    mask_up = s_grid > upper_trigger
    mask_dn = s_grid < lower_trigger

    # R = 1{s>Pt/(1-g)} * g*delta_y + 1{s<Pt(1-g)} * g*delta_x*s
    integrand[mask_up] = gamma * delta_y(s_grid[mask_up], gamma) * pdf_vals[mask_up]
    integrand[mask_dn] = gamma * delta_x(s_grid[mask_dn], gamma) * s_grid[mask_dn] *
pdf_vals[mask_dn]

    return np.trapz(integrand, s_grid)

rows = []
for sigma in sigmas:
    for gamma in gammas:
        rev = expected_one_step_fee_revenue(sigma, gamma)
        rows.append([sigma, gamma, rev])

df = pd.DataFrame(rows, columns=["sigma", "gamma", "ExpectedFeeRevenue"])
print(df.to_string(index=False))
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import minimize_scalar

# Givens
xt = 1000.0
yt = 1000.0
k  = xt * yt
dt = 1/365

s_max = 3.0
N = 8000
s_grid = np.linspace(1e-6, s_max, N)

# Model pieces
def lognormal_pdf_grid(sigma):
   mu = -0.5 * sigma**2 * dt
   var = sigma**2 * dt
   return (1.0 / (s_grid * np.sqrt(2*np.pi*var))) * np.exp(-(np.log(s_grid) - mu)**2 / (2*var))

def delta_y_grid(gamma):
   return np.sqrt(k * (1.0 - gamma) * s_grid) - yt

def delta_x_grid(gamma):
   return np.sqrt(k * ((1.0 - gamma) / s_grid)) - xt
```

```
def expected_fee_revenue_given_pdf(pdf_vals, gamma):
    upper_trigger = 1.0 / (1.0 - gamma)
    lower_trigger = 1.0 - gamma

    integrand = np.zeros_like(s_grid)

    mask_up = s_grid > upper_trigger
    mask_dn = s_grid < lower_trigger

    # R = 1{s>1/(1-g)} * g*delta_y + 1{s<1-g} * g*delta_x*s
    integrand[mask_up] = gamma * delta_y_grid(gamma)[mask_up] * pdf_vals[mask_up]
    integrand[mask_dn] = gamma * delta_x_grid(gamma)[mask_dn] * s_grid[mask_dn] *
pdf_vals[mask_dn]

    return np.trapz(integrand, s_grid)

sigmas = np.arange(0.10, 1.00 + 1e-12, 0.01)

# 0 to 10%
gamma_lo = 1e-6
gamma_hi = 0.10

gamma_star = []

for sigma in sigmas:
```

```
    pdf_vals = lognormal_pdf_grid(sigma)

    # maximize revenue
    obj = lambda g: -expected_fee_revenue_given_pdf(pdf_vals, g)

    res = minimize_scalar(obj, bounds=(gamma_lo, gamma_hi), method="bounded")
    g_star = float(res.x)

    gamma_star.append(g_star)

out = pd.DataFrame({"sigma": sigmas, "gamma_star": gamma_star,})
print(out.head())
print("\nLast rows:")
print(out.tail())

#Plot
plt.figure(figsize=(9, 5))
plt.plot(out["sigma"], out["gamma_star"], marker="o")
plt.xlabel("Volatility σ")
plt.ylabel("Optimal fee γ*(σ)")
plt.title("volatility σ vs Optimal fee γ*(σ) ")
plt.grid(True)
plt.show()
```