Kernel: Python 3 (system-wide)

In [6]:
```
import os
os.environ["MOSEKLM_LICENSE_FILE"] = "/home/user/.mosek/mosek.lic"
```

In [7]:
```
# Task 1

import pandas as pd
import numpy as np
import statsmodels.api as sm


# Read VIX ODS file
vix_ods = pd.read_excel("./VIX-daily-opening-prices.ods",
engine="odf")


# Data preprocessing
vix_ods["DATE"] = pd.to_datetime(vix_ods["DATE"], format="%m/%d/%Y")
vix_ods = vix_ods.sort_values("DATE")
vix_ods["LOG_VIX"] = np.log(vix_ods["OPEN"])

# Construct lagged and incremental terms
xi_t = vix_ods["LOG_VIX"].values[1:]
xi_tm1 = vix_ods["LOG_VIX"].values[:-1]
delta_xi = xi_t - xi_tm1

# Construct regression: Δξ_t = β0 + β1 ξ_{t-1} + ε
X = sm.add_constant(xi_tm1)
model = sm.OLS(delta_xi, X).fit()

# Extract regression parameters
beta0 = model.params[0]
beta1 = model.params[1]

# Strictly derive OU model parameters
a_hat = -beta1
xi_bar_hat = -beta0 / beta1
sigma_hat = np.std(model.resid, ddof=1)

# Output result
print(f"Estimated a: {a_hat:.6f}")
print(f"Estimated ξ̄: {xi_bar_hat:.6f}")
print(f"Estimated σ: {sigma_hat:.6f}")

# Result table
result_table = pd.DataFrame({
    'a': [a_hat],
    'ξ̄': [xi_bar_hat],
    'σ': [sigma_hat]
})
result_table
```

Out[7]:
```
Estimated a: 0.024498
Estimated ξ̄: 2.910105
Estimated σ: 0.076589
```

	a	ξ̄	σ
0	0.024498	2.910105	0.076589

```python
# Verification
mu_bar=sum(xi_t)/len(xi_t)
print(mu_bar)
```

Out[8]:
```
2.90889863254407
```

```python
# Task 2

# Initial log-VIX value (2024-06-21)
xi_0 = vix_ods.loc[vix_ods["DATE"] == "2024-06-21",
"LOG_VIX"].values[0]

# Take observation data between 6/21 and 7/17
start_date = pd.to_datetime("2024-06-21")
end_date = pd.to_datetime("2024-07-17")
mask = (vix_ods["DATE"] >= start_date) & (vix_ods["DATE"] <=
end_date)
subset = vix_ods.loc[mask]

# Actual trading days
T = len(subset) - 1   # If there are n points, there will be n-1 day
intervals

# Calculate mean and standard deviation using OU model formula
E_xi_T = xi_bar_hat + (1 - a_hat)**T * (xi_0 - xi_bar_hat)
Var_xi_T = (sigma_hat**2 / (1 - (1 - a_hat)**2)) * (1 - (1 -
a_hat)**(2*T))
Std_xi_T = np.sqrt(Var_xi_T)


# Calculate median VIX
median_vix_T = np.exp(E_xi_T)


# Output result
print(f"E[ξ_T] = {E_xi_T:.6f}")
print(f"Std[ξ_T] = {Std_xi_T:.6f}")
print(f"Median(VIX_T) = {median_vix_T:.6f}")


# Result table
distribution_stats = pd.DataFrame({
    "E[ξT]": [E_xi_T],
    "σ(ξT)": [Std_xi_T],
    "Median VIX": [median_vix_T]
})

distribution_stats
```

Out[9]:
```
E[ξ_T] = 2.699982
Std[ξ_T] = 0.267539
Median(VIX_T) = 14.879464
```

	E[ξT]	σ(ξT)	Median VIX
0	2.699982	0.267539	14.879464

In [10]:
```
# Task 3

import cvxpy as cp

# Simulate data
M = 100  # quadrature points
n = 10   # number of assets
lambda_ = 2
w_hat = 100000

# Simulate payoff 和 weights
np.random.seed(0)
A = np.random.randn(M, n)
c = np.random.randn(M)
p = np.ones(M) / M  # uniform weights
delta_xi = 1 / M

# Define variable
x = cp.Variable(n)
expr_list = [lambda_ / w_hat * (c[i] - A[i] @ x) for i in range(M)]
log_weights = np.log(p * delta_xi)

# Define the objective function
objective = cp.Minimize((1 / lambda_) *
cp.log_sum_exp(cp.hstack(expr_list) + log_weights))

# Example constraints: x ∈ [-5, 5]
constraints = [x >= -5, x <= 5]

# Construct problems and validate DCP
prob = cp.Problem(objective, constraints)
assert prob.is_dcp(), "The problem is not DCP-compliant!"
print("The optimization problem is convex and DCP-compliant.")
```

Out[10]: The optimization problem is convex and DCP-compliant.

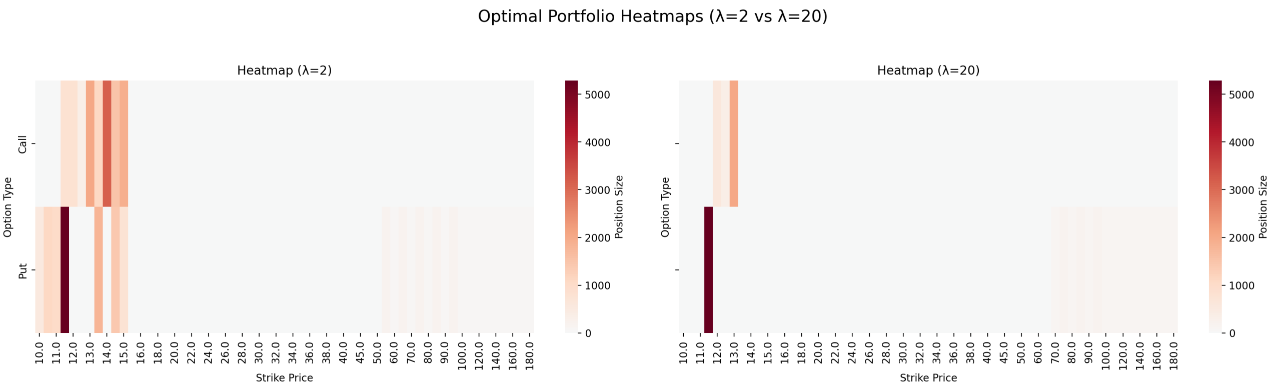
```
# Task 4

from scipy.stats import norm
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore", category=UserWarning,
module="mosek")


# Construct quadrature
M = 100
xi_grid = np.linspace(E_xi_T - 9 * Std_xi_T, E_xi_T + 9 * Std_xi_T,
M)
```

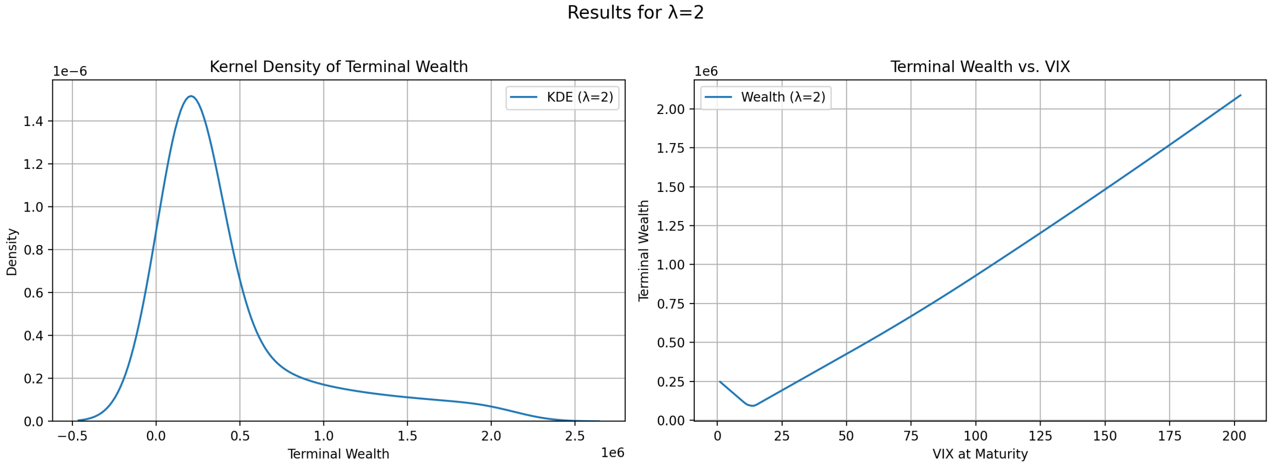
```
delta_xi = (xi_grid[1] - xi_grid[0])
p_xi = norm.pdf(xi_grid, loc=E_xi_T, scale=Std_xi_T)

# Read data
vix_options_data = pd.read_csv("VIX_Data_MScFM_project.csv")

# Construct asset information
assets = [{"type": "cash", "bid": 1.0, "ask": 1.0, "q_bid": 1e6,
"q_ask": 1e6}]
calls = vix_options_data[vix_options_data["option_type"] ==
"C"].sort_values("strike")
puts = vix_options_data[vix_options_data["option_type"] ==
"P"].sort_values("strike")

for _, row in calls.iterrows():
    assets.append({
        "type": "call", "strike": float(row["strike"]),
        "bid": float(row["bid_1545"]), "ask":
float(row["ask_1545"]),
        "q_bid": float(row["bid_size_1545"]), "q_ask":
float(row["ask_size_1545"])
    })
for _, row in puts.iterrows():
    assets.append({
        "type": "put", "strike": float(row["strike"]),
        "bid": float(row["bid_1545"]), "ask":
float(row["ask_1545"]),
        "q_bid": float(row["bid_size_1545"]), "q_ask":
float(row["ask_size_1545"])
    })

payoff_matrix = []
for asset in assets:
    if asset["type"] == "cash":
        payoff = np.ones_like(xi_grid)
    elif asset["type"] == "call":
        payoff = np.maximum(np.exp(xi_grid) - asset["strike"], 0)
    elif asset["type"] == "put":
        payoff = np.maximum(asset["strike"] - np.exp(xi_grid), 0)
    payoff_matrix.append(payoff)
payoff_matrix = np.array(payoff_matrix).T


def solve_portfolio(lambda_val, w_init, xi_grid, p_xi,
payoff_matrix, assets, w_hat=100000):
    M = len(xi_grid)
    n_assets = len(assets)
    delta_xi = (xi_grid[1] - xi_grid[0])
    log_weights = np.log(p_xi * delta_xi)

    # Define variable
    x_var = cp.Variable(n_assets)

    # Construct log-sum-exp objective function
    expr_list = []
    for i in range(M):
        terminal_cf = - (payoff_matrix[i] @ x_var)  # 因为 c(ξ)=0
        expr_list.append((lambda_val / w_hat) * terminal_cf)
    objective = cp.Minimize((1 / lambda_val) *
cp.log_sum_exp(cp.hstack(expr_list) + log_weights))
```

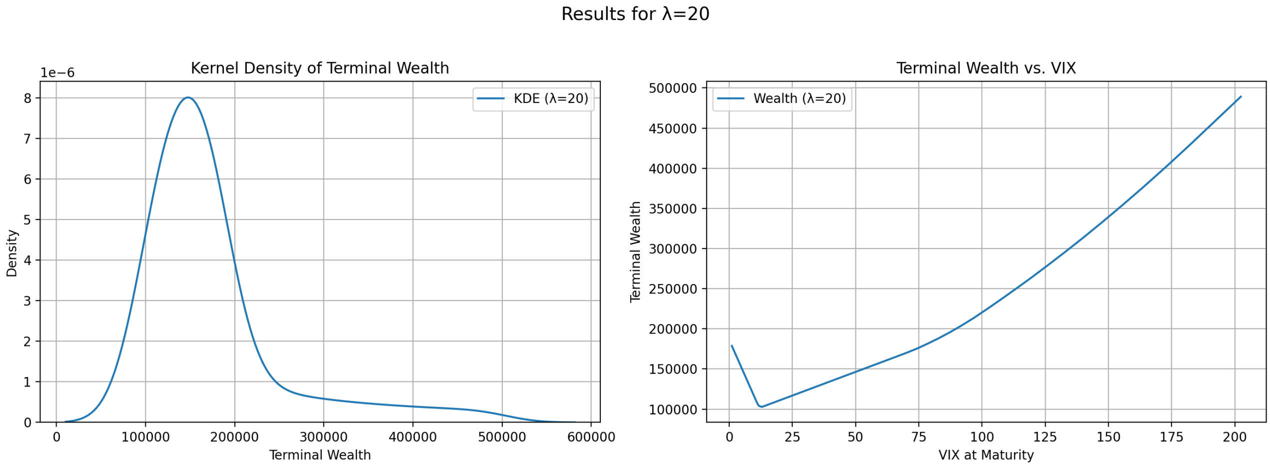
```
    # Construct budget constraints
    constraints = []
    cost_expr = 0
    for j, asset in enumerate(assets):
        xj = x_var[j]
        if asset["type"] == "cash":
            cost_expr += xj
        else:
            xj_pos = cp.pos(xj)
            xj_neg = cp.neg(xj)
            cost_expr += asset["ask"] * xj_pos + asset["bid"] *
xj_neg
            constraints += [
                xj_pos <= asset["q_ask"],
                xj_neg <= asset["q_bid"]
            ]
    constraints += [cost_expr <= w_init]

    # Solution
    prob = cp.Problem(objective, constraints)
    result = prob.solve(solver=cp.MOSEK)

    # Output result
    opt_x = x_var.value
    terminal_wealth = payoff_matrix @ opt_x
    return opt_x, terminal_wealth

# Input preparation (completed)
# xi_grid: quadrature point
# p_xi: probability density
# payoff_matrix: The return of each asset under each ξ
# assets: dict list (including type, price, quantity limit, etc.)

# Run two scenarios of λ=2 and λ=20
w_init = 105000  # 105 * 100000
opt_x_2, wealth_2 = solve_portfolio(2, w_init, xi_grid, p_xi,
payoff_matrix, assets)
opt_x_20, wealth_20 = solve_portfolio(20, w_init, xi_grid, p_xi,
payoff_matrix, assets)


# Visualization
vix_grid = np.exp(xi_grid)

# Draw a heat map of a single optimal combination (λ fixed)
def plot_heatmap(opt_x, assets, label, tol=1e-6, ax=None):



    data = []
    for x, a in zip(opt_x, assets):
        if abs(float(x)) <= tol:
            continue
        if a["type"] in ["call", "put"]:
            data.append({"Strike": a["strike"], "Type":
a["type"].capitalize(), "Position": float(x)})

    if not data:
        ax.set_visible(False)
        return
```

```
    df = pd.DataFrame(data)
    pivot_table = df.pivot_table(index="Type", columns="Strike",
values="Position", fill_value=0)

    sns.heatmap(pivot_table, cmap="RdBu_r", center=0, ax=ax,
                cbar_kws={"label": "Position Size"})
    ax.set_title(f"Heatmap (λ={label})")
    ax.set_xlabel("Strike Price")
    ax.set_ylabel("Option Type")

def plot_two_heatmaps(opt_x_2, opt_x_20, assets):

    fig, axes = plt.subplots(1, 2, figsize=(18, 5), sharey=True)
    plot_heatmap(opt_x_2, assets, label=2, ax=axes[0])
    plot_heatmap(opt_x_20, assets, label=20, ax=axes[1])
    fig.suptitle("Optimal Portfolio Heatmaps (λ=2 vs λ=20)",
fontsize=16, y=1.05)
    plt.tight_layout()
    plt.show()

def plot_results_together(wealth, label):
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # Left chart: KDE
    sns.kdeplot(wealth, label=f"KDE (λ={label})", ax=axes[0])
    axes[0].set_xlabel("Terminal Wealth")
    axes[0].set_title("Kernel Density of Terminal Wealth")
    axes[0].legend()
    axes[0].grid(True)

    # Right chart: Wealth vs VIX
    axes[1].plot(vix_grid, wealth, label=f"Wealth (λ={label})")
    axes[1].set_xlabel("VIX at Maturity")
    axes[1].set_ylabel("Terminal Wealth")
    axes[1].set_title("Terminal Wealth vs. VIX")
    axes[1].legend()
    axes[1].grid(True)

    fig.suptitle(f"Results for λ={label}", fontsize=14, y=1.02)
    plt.tight_layout()
    plt.show()


# Present
plot_two_heatmaps(opt_x_2, opt_x_20, assets)
plot_results_together(wealth_2, label=2)
plot_results_together(wealth_20, label=20)
```

Out[41]:





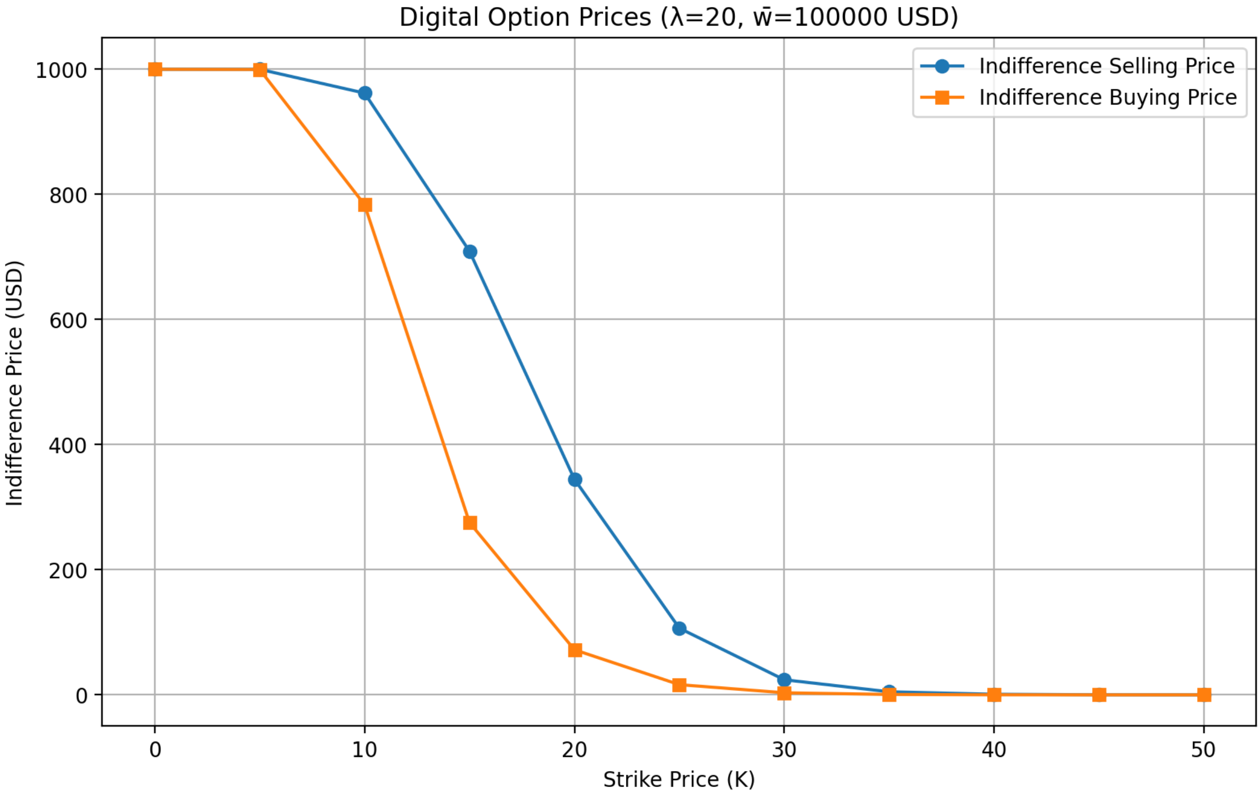



```
# Task 6

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize_scalar

# Corrected Parameters
initial_cash = 1e5        # w̄ = 100,000 USD
initial_liability = 0     # c̄ = 0
risk_aversion = 20        # λ = 20
payoff = 1000             # Digital option pays 1,000 USD

# VIX simulation (lognormal distribution)
np.random.seed(42)
n_sim = 100000

vix_terminal = np.random.lognormal(mean=E_xi_T, sigma=Std_xi_T,
size=n_sim)
```

```python
# Numerically stable utility function
def utility(x):
    scale = 1e-4  # scaling factor
    x_scaled = x * scale
    x_scaled = np.clip(x_scaled, -100, 100)  # Prevent overflow
    return -np.exp(-risk_aversion * x_scaled) / scale

# Expected utility calculation
def rho(w, c):
    terminal_wealth = w - c
    return np.mean(utility(terminal_wealth))

# No difference price calculation (with optimized fault tolerance)
def indifference_price(w0, c0, c_new, is_selling=True):
    u0 = rho(w0, c0)
    def objective(w):
        if is_selling:
            return abs(rho(w0 + w, c0 + c_new) - u0)
        else:
            return abs(rho(w0 - w, c0 - c_new) - u0)

    # Optimize the search scope to [0, payoff]
    res = minimize_scalar(
        objective,
        bounds=(0, payoff),
        method='bounded',
        options={'xatol': 1e-6, 'maxiter': 1000}
    )
    return res.x if res.success else np.nan

# Calculate the indifference price for different execution prices K
strikes = np.arange(0, 55, 5)  # K = 0, 5, 10, ..., 50
sell_prices = []
buy_prices = []

for K in strikes:
    digital_payoff = np.where(vix_terminal > K, payoff, 0)

    # Sell price: the minimum compensation after accepting the option
    sell_price = indifference_price(initial_cash, initial_liability, digital_payoff, is_selling=True)
    sell_prices.append(sell_price)

    # Purchase price: The maximum amount willing to pay when purchasing an option
    buy_price = indifference_price(initial_cash, initial_liability, digital_payoff, is_selling=False)
    buy_prices.append(buy_price)

# Present results
plt.figure(figsize=(10, 6))
plt.plot(strikes, sell_prices, 'o-', label="Indifference Selling Price")
plt.plot(strikes, buy_prices, 's-', label="Indifference Buying Price")
plt.xlabel("Strike Price (K)")
plt.ylabel("Indifference Price (USD)")
plt.title(f"Digital Option Prices (λ={risk_aversion}, w̄={initial_cash:.0f} USD)")
```

```
plt.legend()
plt.grid(True)
plt.show()
```

Out[22]:

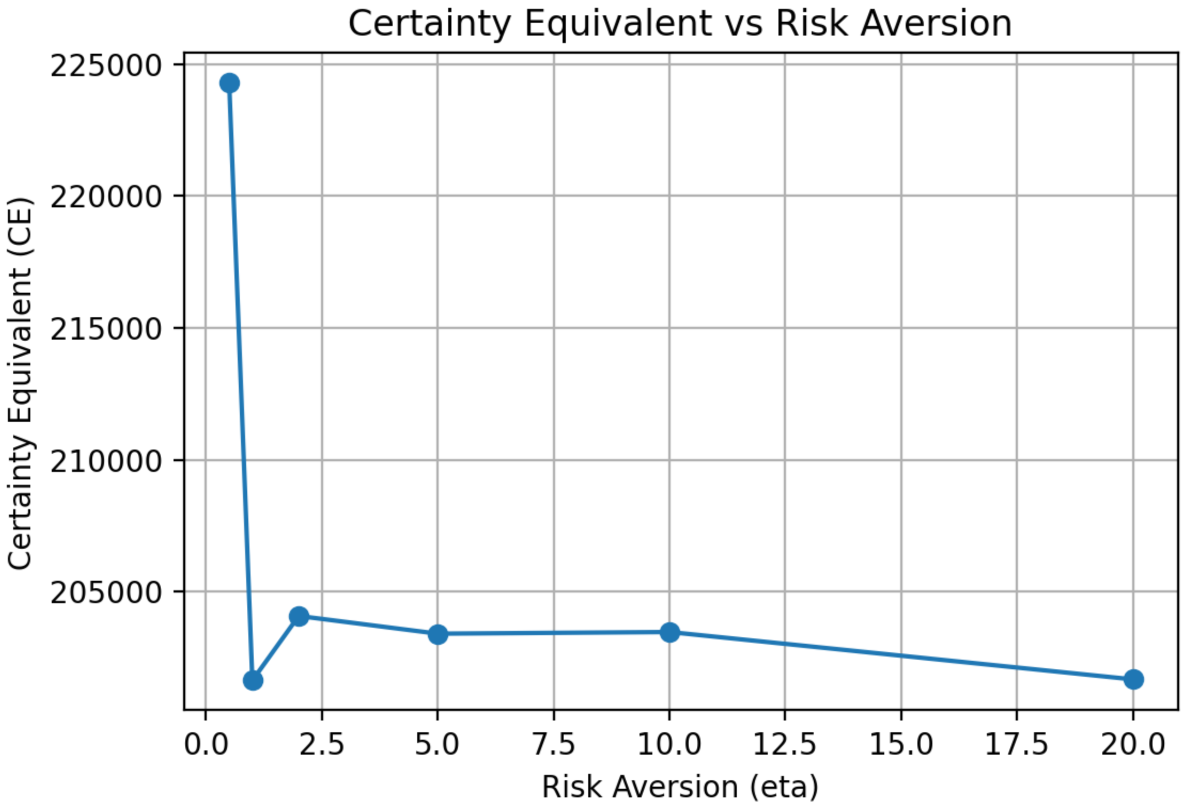



```
# Chapter 4.2

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from scipy.optimize import minimize


# Simulate VIX distribution
M = 2000

np.random.seed(42)
xi_samples = np.random.normal(E_xi_T, Std_xi_T, M)
VIX_T = np.exp(xi_samples)


# Read VIX option data
vix_options_data = pd.read_csv("VIX_Data_MScFM_project.csv")
calls = vix_options_data[vix_options_data["option_type"] ==
"C"].sort_values("strike")
puts = vix_options_data[vix_options_data["option_type"] ==
"P"].sort_values("strike")

assets = [{"type": "cash", "bid": 1.0, "ask": 1.0}]
for _, row in pd.concat([calls, puts]).iterrows():
    assets.append({
        "type": row["option_type"],
        "strike": row["strike"],
        "bid": row["bid_1545"],
        "ask": row["ask_1545"]
```

```python
    })
J = len(assets)

# payoff matrix
payoffs = []
for asset in assets:
    if asset["type"] == "cash":
        payoff = np.ones_like(VIX_T)
    elif asset["type"] == "C":
        payoff = np.maximum(VIX_T - asset["strike"], 0)
    elif asset["type"] == "P":
        payoff = np.maximum(asset["strike"] - VIX_T, 0)
    payoffs.append(payoff)
payoffs = np.array(payoffs).T  # (M, J)


# Power utility function + certainty equivalent
def certainty_equivalent(portfolio_payoff, eta, w0=100000):
    W = w0 + portfolio_payoff
    W = np.maximum(W, 1e-8)  # Ensure positive wealth
    if eta != 1:
        EU = np.mean(W**(1-eta)/(1-eta))
        CE = ((1-eta)*EU) ** (1/(1-eta))
    else:
        EU = np.mean(np.log(W))
        CE = np.exp(EU)
    return CE

def obj(x, eta, w0=100000):
    return -certainty_equivalent(payoffs @ x, eta, w0)

def budget_constraint(x, w0=100000):
    cost = 0
    for j, asset in enumerate(assets):
        if asset["type"] == "cash":
            cost += x[j]
        else:
            cost += asset["ask"] * max(x[j], 0) - asset["bid"] *
min(x[j], 0)
    return w0 - cost

def optimize_portfolio(eta):
    x0 = np.zeros(J)
    cons = {"type": "ineq", "fun": lambda x: budget_constraint(x)}
    res = minimize(obj, x0, args=(eta,), constraints=[cons],
method="SLSQP")
    return res.x, -res.fun



# Run different η
etas = [0.5 ,1 ,2 , 5, 10, 20]
results = []
for eta in etas:
    x_opt, ce = optimize_portfolio(eta)
    results.append({"eta": eta, "CE": ce, "x": x_opt})


# Plot η vs CE
plt.figure(figsize=(6,4))
plt.plot([r["eta"] for r in results], [r["CE"] for r in results],
```

```
marker="o")
plt.xlabel("Risk Aversion (eta)")
plt.ylabel("Certainty Equivalent (CE)")
plt.title("Certainty Equivalent vs Risk Aversion")
plt.grid(True)
plt.show()
```

Out[33]:




```
# Chapter 4.3

def compute_metrics(WT, VIX_T, eta, w0=100000):
    metrics = {}

    # 1. Basic Statistics
    metrics["mean"] = np.mean(WT)
    metrics["std"] = np.std(WT)
    metrics["median"] = np.median(WT)

    # 2. Lower tail risk
    alpha = 0.05
    var_level = np.quantile(WT, alpha)  # VaR_5%
    metrics["VaR_5%"] = var_level
    metrics["ES_5%"] = WT[WT <= var_level].mean()  # Expected
Shortfall

    # 3. Certainty equivalent (CE)
    WT_safe = np.maximum(WT, 1e-8)
    if eta != 1:
        EU = np.mean(WT_safe**(1-eta)/(1-eta))
        CE = ((1-eta)*EU) ** (1/(1-eta))
    else:
        EU = np.mean(np.log(WT_safe))
        CE = np.exp(EU)
    metrics["CE"] = CE

    # 4. Upward elasticity (fitting slope in the larger range of
```

```
VIX)
    high_idx = VIX_T > np.quantile(VIX_T, 0.75)  # Take the upper
quartile of VIX
    slope = np.polyfit(VIX_T[high_idx], WT[high_idx], 1)[0]
    metrics["upside_slope"] = slope

    # 5. Convexity index (approximated by second-order difference in
the middle interval)
    order = np.argsort(VIX_T)
    V_sorted, W_sorted = VIX_T[order], WT[order]
    # Second order difference
    second_diff = np.diff(W_sorted, 2)
    metrics["convexity_mean"] = np.mean(second_diff)

    return metrics



# Calculate metrics for multiple eta
etas = [0.5, 1, 2, 5, 10, 20]
results = []
w0=100000

for eta in etas:

    WT = w0 + payoffs @ optimize_portfolio(eta)[0]
    metrics = compute_metrics(WT, VIX_T, eta, w0=w0)
    metrics["eta"] = eta
    results.append(metrics)

df_metrics = pd.DataFrame(results)
print(df_metrics)
```

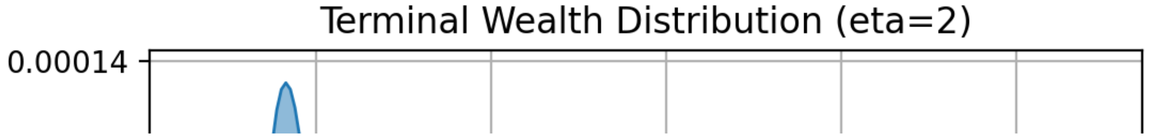
Out[30]:

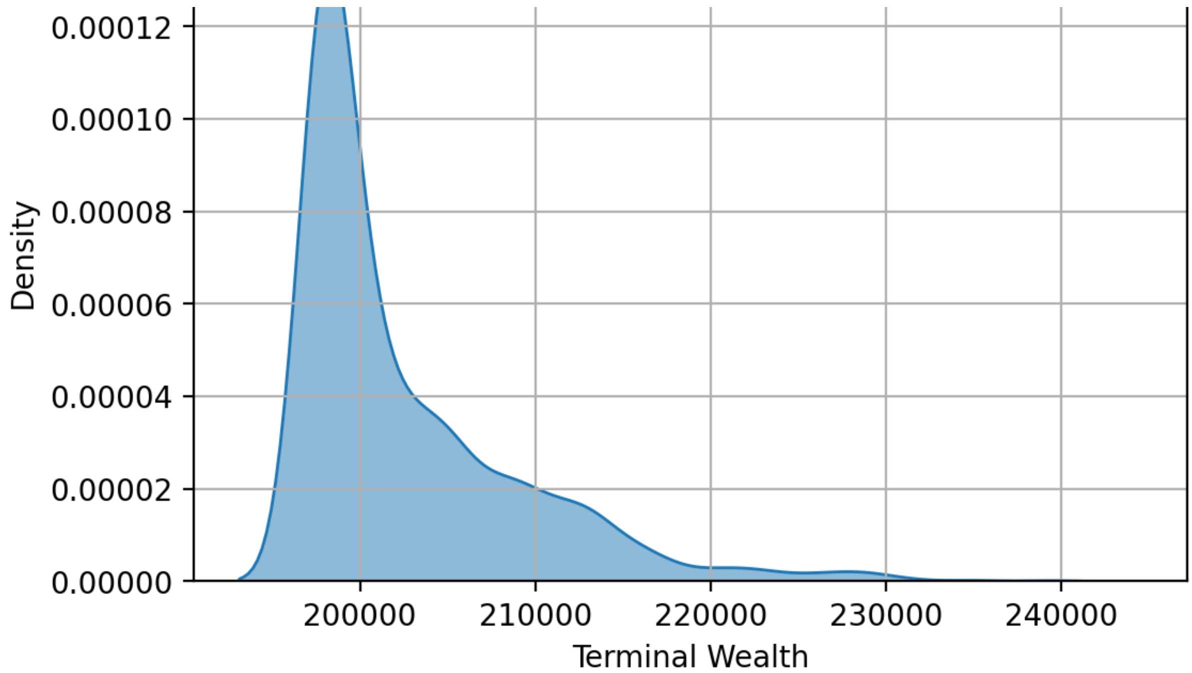
```
            mean           std          median           VaR_5%
ES_5%  \
0  230542.606325  83913.101005  200398.542192  159979.947023
159749.757308
1  201664.851574   3554.452585  200131.280885  198877.914590
198875.647782
2  204752.338613  12309.358196  199939.835769  194642.423501
194622.691194
3  205457.466304  14410.728954  200043.792681  193470.434722
193451.749054
4  208014.961353  17243.364829  202214.507921  193365.577757
193255.294966
5  202810.435158   5523.795354  200992.755832  197831.119540
197807.413635

              CE  upside_slope  convexity_mean   eta
0  224278.604494  22457.597603       88.298515   0.5
1  201634.190447    962.957367        3.010091   1.0
2  204077.870606   3192.059323        9.807034   2.0
3  203407.012505   3806.559086       12.968462   5.0
4  203469.442814   5362.706095       24.046099  10.0
5  201674.726534   1525.201931        6.706481  20.0
```

```
# Chapter 4.4

eta_idx = 2  # Select the third, which is eta=2
x_opt = results[eta_idx]["x"]
```

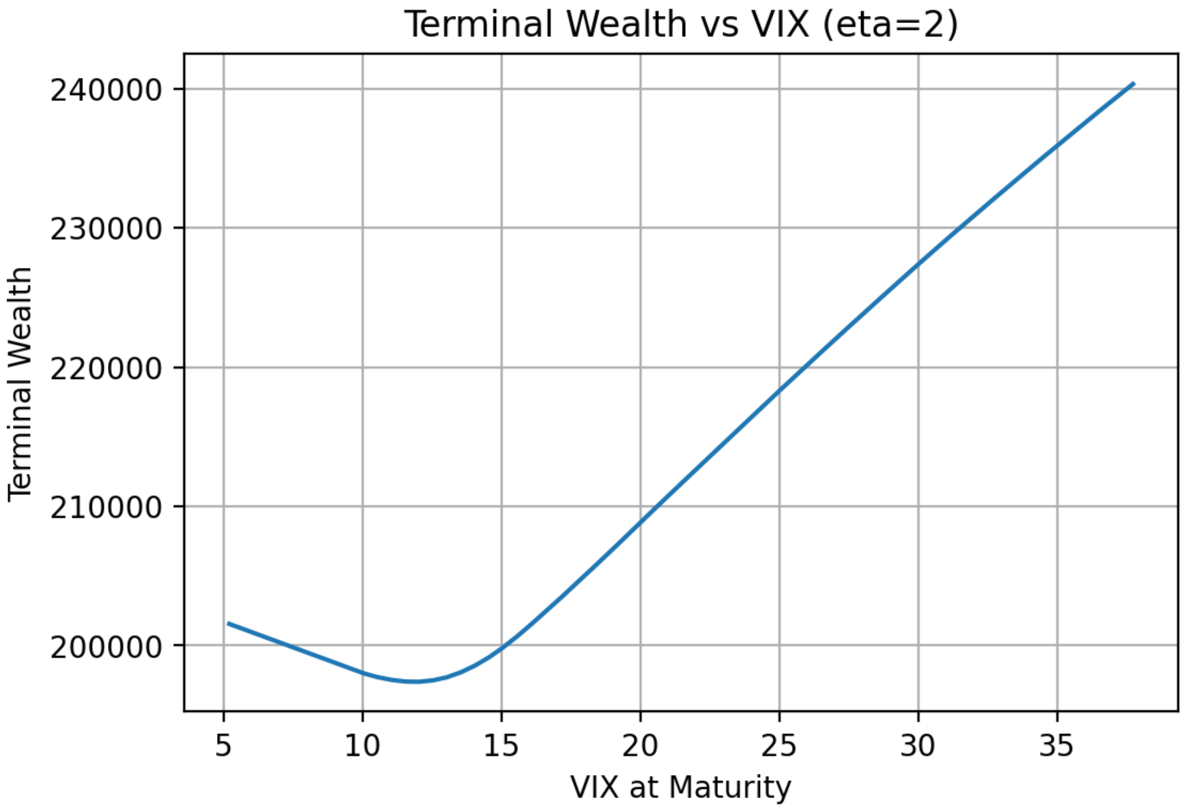
```python
# Distribution of terminal wealth KDE (eta=2)
portfolio_payoff = payoffs @ x_opt
W_T = 100000 + portfolio_payoff

plt.figure(figsize=(6,4))
sns.kdeplot(W_T, fill=True, alpha=0.5)
plt.xlabel("Terminal Wealth")
plt.ylabel("Density")
plt.title(f"Terminal Wealth Distribution (eta={results[eta_idx]['eta']})")
plt.grid(True)
plt.show()


# Terminal wealth vs VIX (eta=2)
# Sort by VIX_T
idx = np.argsort(VIX_T)
plt.figure(figsize=(6,4))
plt.plot(VIX_T[idx], W_T[idx], lw=1.5)
plt.xlabel("VIX at Maturity")
plt.ylabel("Terminal Wealth")
plt.title(f"Terminal Wealth vs VIX (eta={results[eta_idx]['eta']})")
plt.grid(True)
plt.show()


eta_idx = 5  # Select the sixth, which is eta=20
x_opt = results[eta_idx]["x"]

# Distribution of terminal wealth KDE (eta=20)
portfolio_payoff = payoffs @ x_opt
W_T = 100000 + portfolio_payoff

plt.figure(figsize=(6,4))
sns.kdeplot(W_T, fill=True, alpha=0.5)
plt.xlabel("Terminal Wealth")
plt.ylabel("Density")
plt.title(f"Terminal Wealth Distribution (eta={results[eta_idx]['eta']})")
plt.grid(True)
plt.show()


# Terminal wealth vs VIX (eta=20)
# Sort by VIX_T
idx = np.argsort(VIX_T)
plt.figure(figsize=(6,4))
plt.plot(VIX_T[idx], W_T[idx], lw=1.5)
plt.xlabel("VIX at Maturity")
plt.ylabel("Terminal Wealth")
plt.title(f"Terminal Wealth vs VIX (eta={results[eta_idx]['eta']})")
plt.grid(True)
plt.show()
```

Out[16]:

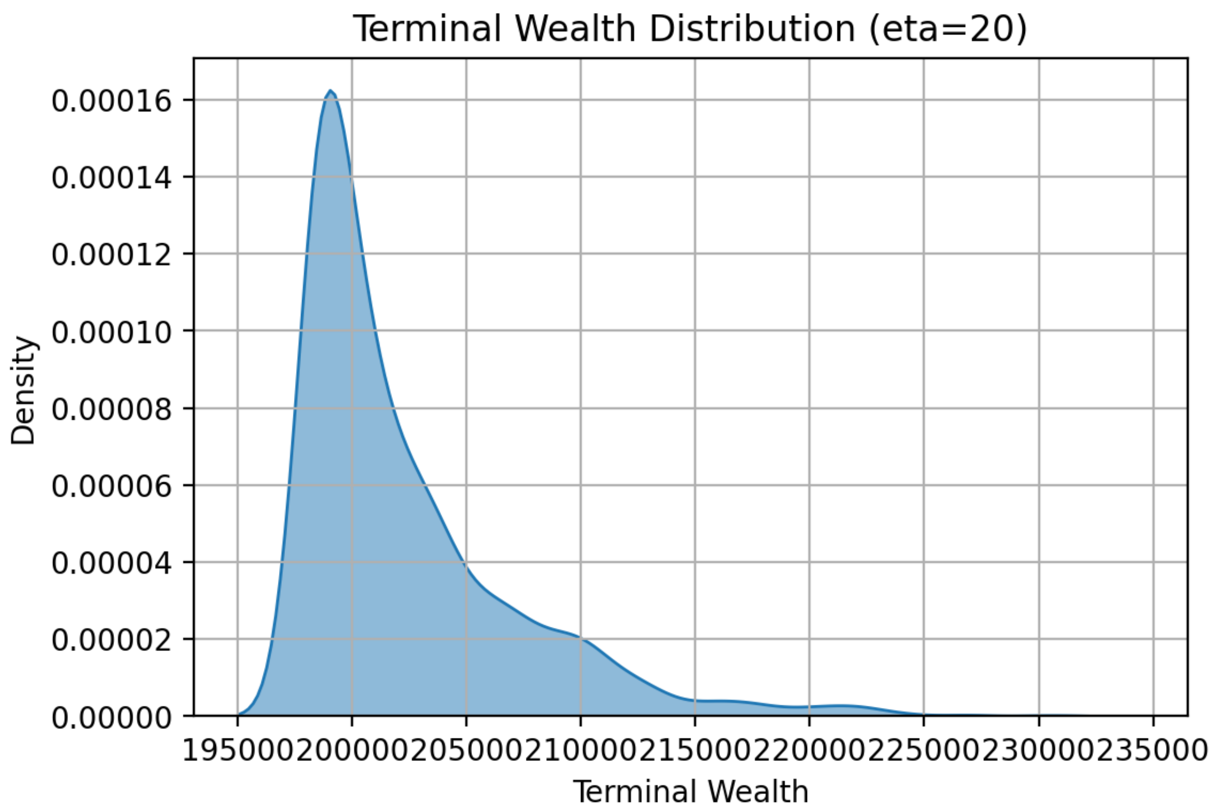




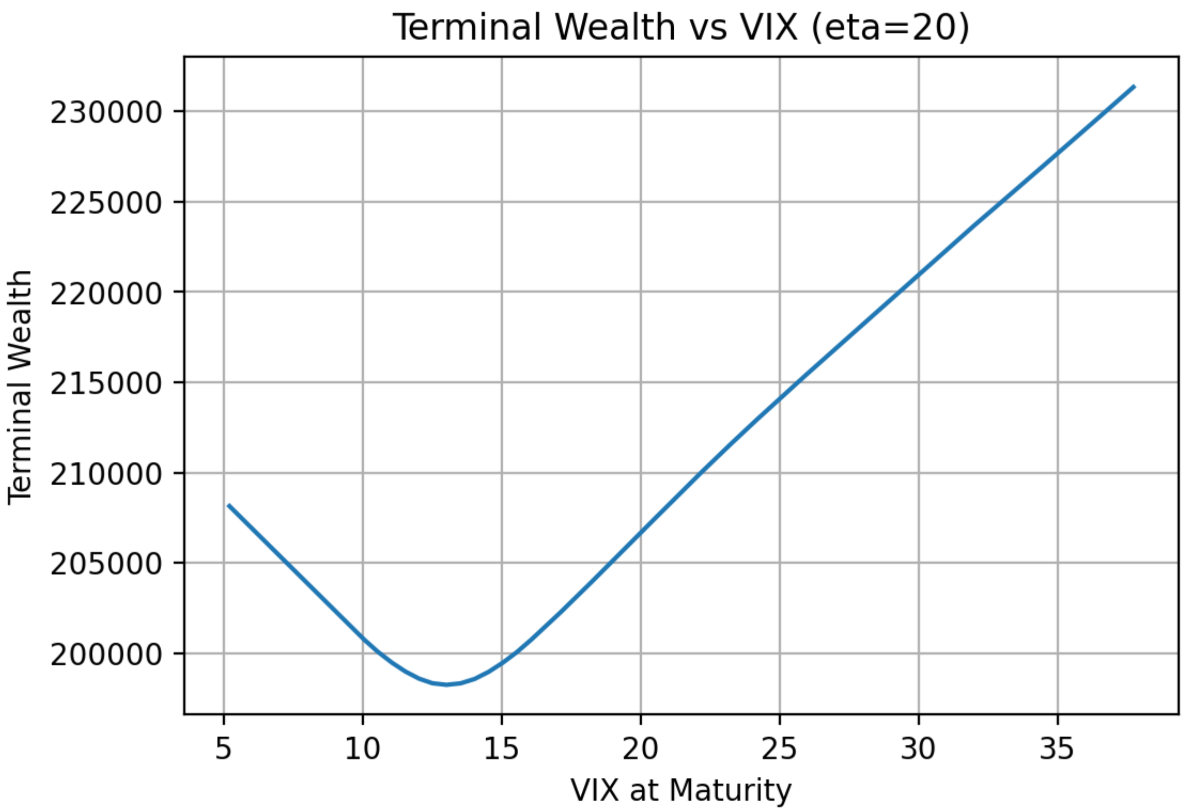
Density
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
200000
210000
220000
230000
240000
Terminal Wealth



Terminal Wealth vs VIX (eta=2)
Terminal Wealth
240000
230000
220000
210000
200000
5
10
15
20
25
30
35
VIX at Maturity



Terminal Wealth Distribution (eta=20)
Density
0.00016
0.00014
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
195000
200000
205000
210000
215000
220000
225000
230000
235000
Terminal Wealth



Terminal Wealth vs VIX (eta=20)
Terminal Wealth
230000
225000
220000
215000
210000
205000
200000
5
10
15
20
25
30
35
VIX at Maturity