

ChatGPT

Previous 7 Days

Portfolio VaR Calculation

Consider Asset Correlation Vasicek

Vasicek Model Variables Summary

Previous 30 Days

Equal Scaling with ScalarFormatter

Grid Plot for PDs

Compact Python Code Loops

Modify Vasicek Simulation Code

Fixing Vasicek Simulation

Portfolio Loss Simulation Enhancements

Fix Scaling X-Axis

Portfolio Modeling Metrics Overview

Graphs Illustrate Loan Simulation

Choose subtitles for 4 people

Video Project Script Draft

Expected Loss Values

Number Formatting with Locale

Loss Simulation using Vasicek

Dynamic Asset Allocation Model

Vasicek Model Adjustment

Integrate Business PD Values

Continuous LGD using Beta

Airbnb Fragen für Vermietern

Grundstück Beratungsgespräch

Multiple simulations with varying parameters

Loan PD Average: Positive

Loan Amount Scaling

Thinner Histogram Lines

Credit Loss Projection Model

Good Boy Prize

Excel Value Lookup: Criteria1 & C2

In writing a vasicek portfolio simulation in a jupyter notebook, in this modeling technique rho is a value that is used to describe the correlation between assets within a portfolio. I am using two asset classes, personal and business loans, for these two asset classes I use two separate rho values and I run my simulations 5 pairs of these rho values, however before I made this modification to my code I was running the simulation with just 1 pair of rho values and the simulation therefore only resulted in two data frames for losses data. now I instead get two dictionaries with loss values. what I need you to do now is modify my graphs and calculations that use these results so that instead of giving just one output I want 5 now for each pair of rho values.

the changed simulation code which also results in all the output data for calculations is as follows:

```

# Dictionary to store data frames for each rho value
personal_losses_dict = {}
business_losses_dict = {}

# Vasicek parameters
Y = np.random.normal(0, 1, n_simulations)
Z = np.random.normal(0, 1, (n_simulations, n_assets))

# Simulation for each pair of rho values
for rho_personal, rho_business in zip(rho_personal_values,
                                       rho_business_values):
    losses_personal = np.zeros(n_simulations)
    losses_business = np.zeros(n_simulations)

    for i in range(n_simulations):
        # Compute X_personal and X_business for the subset of loan types
        X_personal = (np.sqrt(rho_personal**2) * Y[i] + (np.sqrt(1 - rho_personal**2)) * Z[i])[loan_types == 0]
        X_business = (np.sqrt(rho_business**2) * Y[i] + (np.sqrt(1 - rho_business**2)) * Z[i])[loan_types == 1])

        # Compute default events using norm.ppf(pd) for each PD
        values
        default_events_personal = X_personal < np.array([norm.ppf(pd)
        for pd in personal_PDs])
        default_events_business = X_business < np.array([norm.ppf(pd)
        for pd in business_PDs])

        # Calculate losses
        loss_personal = np.sum(default_events_personal * LGD_personal *
        loan_amounts_pe)
        loss_business = np.sum(default_events_business * LGD_business *
        loan_amounts_business)

        losses_personal[i] = loss_personal
        losses_business[i] = loss_business

    # Create data frames and store them in the dictionaries
    personal_losses_dict[rho_personal] = pd.DataFrame({'Loss': losses_personal})
    business_losses_dict[rho_business] = pd.DataFrame({'Loss': losses_business})

and the code I need you to modify is this:
# Combine losses for calculations
total_losses = losses_personal + losses_business

# Set locale to use dot as thousands separator and comma as decimal separator
locale.setlocale(locale.LC_ALL, 'en_de.UTF-8')

# Portfolio Value
PL = np.sum(loan_amounts_personal)
BL = np.sum(loan_amounts_business)
PV = PL + BL
print(f'Portfolio Value (PV): {locale.format_string("%.2f", PV,
grouping=True)}')
print(f'Personal Loans: {locale.format_string("%.2f", PL, grouping=True)}')
print(f'Business Loans: {locale.format_string("%.2f", BL, grouping=True)}')
print(f'€')

# Expected Loss (EL)
EL = np.mean(total_losses)
print(f'Expected Loss (EL): {locale.format_string("%.2f", EL,
grouping=True)}')
print(f'{(np.round(EL/PV*100.2))}%')

# Unexpected Loss (UL)
UL = np.std(total_losses)+EL
print(f'Unexpected Loss (UL): {locale.format_string("%.2f", UL,
grouping=True)}')
print(f'{(np.round(UL/PV*100.2))}%')

# Value at Risk (VaR) 99%
VaR_99 = np.percentile(total_losses, 99)
print(f'Value at Risk (VaR) 99%: {locale.format_string("%.2f", VaR_99,
grouping=True)}')
print(f'{(np.round(VaR_99/PV*100.2))}%')

# Value at Risk (VaR) 95%
VaR_95 = np.percentile(total_losses, 95)
print(f'Value at Risk (VaR) 95%: {locale.format_string("%.2f", VaR_95,
grouping=True)}')
print(f'{(np.round(VaR_95/PV*100.2))}%')

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.hist(personal_PD, bins=30, color='blue', alpha=0.7,
edgecolor='black')
plt.title('Distribution of Personal Loan PD Values')
plt.xlabel('Probability of Default')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
plt.hist(business_PD, bins=30, color='red', alpha=0.7,
edgecolor='black')
plt.title('Distribution of Business Loan PD Values')
plt.xlabel('Probability of Default')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
from matplotlib.ticker import ScalarFormatter

plt.figure(figsize=(12, 6))

# Plot for Personal Loan Amounts
plt.subplot(1, 2, 1)
plt.hist(loan_amounts_personal, bins=30, color='blue', alpha=0.7,
edgecolor='black')
plt.title('Distribution of Personal Loan Amounts')
plt.xlabel('Loan Amount (€)')
plt.ylabel('Frequency')

# Use ScalarFormatter for the x-axis
plt.gca().xaxis.set_major_formatter(ScalarFormatter(useOffset=False))

# Plot for Business Loan Amounts
plt.subplot(1, 2, 2)
plt.hist(loan_amounts_business, bins=30, color='red', alpha=0.7,
edgecolor='black')
plt.title('Distribution of Business Loan Amounts')
plt.xlabel('Loan Amount (€)')
```

```

plt.ylabel('Frequency')

# Use ScalarFormatter for the x-axis
plt.gca().xaxis.set_major_formatter(ScalarFormatter(useOffset=False))

plt.tight_layout()
plt.show()

# Data points to highlight
points = [
    "Expected Loss (EL)": EL,
    "Unexpected Loss (UL)": UL,
    "Value at Risk (VaR) 99%": VaR_99,
    "Value at Risk (VaR) 95%": VaR_95
]

# Colors for the vertical lines
colors = [
    "Expected Loss (EL)": 'blue',
    "Unexpected Loss (UL)": 'green',
    "Value at Risk (VaR) 99%": 'magenta',
    "Value at Risk (VaR) 95%": 'cyan'
]

plt.figure(figsize=(10, 6))
plt.hist(total_losses, bins=30, color='purple', alpha=0.7,
        edgecolor='black')
plt.title('Overall Portfolio Loss Distribution')
plt.xlabel('Total Loss (€)')
plt.ylabel('Frequency')

# Add vertical lines for each point
for label, value in points.items():
    plt.axvline(x=value, color=colors[label], linestyle='--', linewidth=1,
                label=f'[label]: {value:.2f}€')

# Add legend to the plot
plt.legend(loc='upper right')

plt.tight_layout()
plt.show()

from numpy.polynomial.polynomial import Polynomial

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

# Sort the losses and compute the cumulative distribution
sorted_losses = np.sort(total_losses)
cumulative = np.cumsum(sorted_losses) / np.sum(sorted_losses)

# Fit a polynomial curve to the cumulative distribution
degree = 5 # Degree of the polynomial
p = Polynomial.fit(sorted_losses, cumulative, degree)

# Calculate the polynomial derivative
dp = p.deriv()

# Find the x-value corresponding to the steepest point (maximum derivative)
x_steepest = sorted_losses[np.argmax(dp(sorted_losses))]
y_steepest = p(x_steepest)

# Plot the cumulative distribution and the fitted polynomial
plt.plot(sorted_losses, cumulative, color='green', label='Cumulative Distribution')
plt.plot(sorted_losses, p(sorted_losses), color='blue', linestyle='--',
        label='Fitted Curve', alpha=0.5)

# Highlight the steepest point
plt.scatter([x_steepest], [y_steepest], color='red')

# Additional plot details
plt.title('Cumulative Loss Distribution')
plt.xlabel('Total Loss (€)')
plt.ylabel('Cumulative Probability')
plt.grid(True)
plt.legend(loc='best')

plt.tight_layout()
plt.show()

```

To modify your code to handle multiple pairs of rho values and produce outputs for each pair, you'll need to iterate over each pair, compute the necessary metrics (such as EL, UL, VaR), and then modify the plotting routines to handle multiple datasets. Here's how you can modify your existing code:

1. Compute Metrics for Each Pair of Rho Values:

Instead of computing metrics using `'total_losses'`, which combined losses from all pairs, you should compute metrics separately for each pair stored in `'personal_losses_dict'` and `'business_losses_dict'`.

```

python Copy code

# Initialize lists to store metrics for each pair
EL_list = []
UL_list = []
VaR_99_list = []
VaR_95_list = []

for rho_personal, rho_business in zip(rho_personal_values, rho_business_values):
    # Retrieve losses for the current pair of rho values
    losses_personal = personal_losses_dict['rho_personal_(rho_personal)']['loss']
    losses_business = business_losses_dict['rho_business_(rho_business)']['loss']

    # Combine losses for calculations
    total_losses = losses_personal + losses_business

    # Calculate metrics
    EL = np.mean(total_losses)
    UL = np.std(total_losses) + EL
    VaR_99 = np.percentile(total_losses, 99)
    VaR_95 = np.percentile(total_losses, 95)

    # Append metrics to lists
    EL_list.append(EL)
    UL_list.append(UL)
    VaR_99_list.append(VaR_99)
    VaR_95_list.append(VaR_95)

```

2. Modify Plotting Routines:

Adjust the plotting routines to handle multiple datasets (one for each pair of rho values). Here's an example for the histogram and vertical lines plot:

```

python Copy code

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

for i, (rho_personal, rho_business) in enumerate(zip(rho_personal_values, rho_business_values)):
    total_losses = personal_losses_dict['rho_personal_(rho_personal)']['loss'] + business_losses_dict['rho_business_(rho_business)']['loss']
    plt.hist(total_losses, bins=30, alpha=0.5, edgecolor='black', label=f'Rho Pair [{i}]')

plt.title('Overall Portfolio Loss Distribution')
plt.xlabel('Total Loss (€)')
plt.ylabel('Frequency')
plt.legend(loc='upper right')

plt.tight_layout()

```

```
plt.show()
```

Adjust similar modifications for other plots like cumulative distribution and polynomial fitting.

3. Output Metrics for Each Pair:

Instead of printing a single set of metrics, iterate over your lists of metrics ("EL_list", "UL_list", "Var_99_list", "Var_95_list") to print out each one in any further analysis or reporting.

```
for i, (rho_personal, rho_business) in enumerate(zip(rho_personal_values, rho_business_values)):
    print(f"Metrics for Rho Pair ({i+1}):")
    print(f"Expected loss (EL): {locale.format_string('.%2f', EL_list[i], grouping=True)}")
    print(f"Unexpected loss (UL): {locale.format_string('.%2f', UL_list[i], grouping=True)}")
    print(f"Value at Risk (Var 99%): {locale.format_string('.%2f', Var_99_list[i], grouping=True)}")
    print(f"Value at Risk (Var 95%): {locale.format_string('.%2f', Var_95_list[i], grouping=True)}")
```

This will ensure that you get separate metrics for each pair of rho values used in your simulation.

By following these steps, you should be able to adapt your Jupyter notebook to handle multiple pairs of rho values, compute appropriate metrics for each pair, and visualize the results accordingly. Adjust the details (like plot titles, axis labels, formatting) to suit your specific requirements and preferences.

can you give me new code for all the plots i sent you?

Certainly! Here's how you can modify the plotting code for each of the plots you've described: histogram plots, cumulative distribution plot, and polynomial fitting plot. We'll iterate over each pair of rho values and plot the corresponding data.

Histogram Plots

```
plt.figure(figsize=(12, 6))

for i, (rho_personal, rho_business) in enumerate(zip(rho_personal_values, rho_business_values)):
    total_losses = personal_losses_dict[f"rho_personal_{(rho_personal)}"]['loss'] + business_losses_dict[f"rho_business_{(rho_business)}"]['loss']

    plt.subplot(2, 5, i+1)
    plt.hist(total_losses, bins=10, alpha=0.7, edgecolor='black')
    plt.title(f'Rho Pair ({i+1})')
    plt.xlabel('Total Loss (€)')
    plt.ylabel('Frequency')
```

Cumulative Distribution Plot

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

for i, (rho_personal, rho_business) in enumerate(zip(rho_personal_values, rho_business_values)):
    total_losses = personal_losses_dict[f"rho_personal_{(rho_personal)}"]['loss'] + business_losses_dict[f"rho_business_{(rho_business)}"]['loss']

    sorted_losses = np.sort(total_losses)
    cumulative = np.cumsum(sorted_losses) / np.sum(sorted_losses)

    # Fit a polynomial curve to the cumulative distribution
    degree = 5 # Degree of the polynomial
    p = Polynomial.fit(sorted_losses, cumulative, degree)

    plt.plot(sorted_losses, cumulative, label=f'Rho Pair ({i+1})')

plt.title('Cumulative loss Distribution')
plt.xlabel('Total Loss (€)')
plt.ylabel('Cumulative Probability')
plt.grid(True)
plt.legend(loc='best')

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

Polynomial Fitting Plot

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

for i, (rho_personal, rho_business) in enumerate(zip(rho_personal_values, rho_business_values)):
    total_losses = personal_losses_dict[f"rho_personal_{(rho_personal)}"]['loss'] + business_losses_dict[f"rho_business_{(rho_business)}"]['loss']

    sorted_losses = np.sort(total_losses)
    cumulative = np.cumsum(sorted_losses) / np.sum(sorted_losses)

    # Fit a polynomial curve to the cumulative distribution
    degree = 5 # Degree of the polynomial
    p = Polynomial.fit(sorted_losses, cumulative, degree)

    # Plot the cumulative distribution and the fitted polynomial
    plt.plot(sorted_losses, cumulative, color='C1', label=f'Rho Pair ({i+1})')
    plt.plot(sorted_losses, p(sorted_losses), color='C1', linestyle='--', alpha=0.5)

plt.title('Cumulative loss Distribution with Polynomial Fit')
plt.xlabel('Total Loss (€)')
plt.ylabel('Cumulative Probability')
plt.grid(True)
plt.legend(loc='best')

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

Explanation:

- Histogram Plots:** We iterate over each pair of rho values ("rho_personal_values" and "rho_business_values"), retrieve the corresponding total losses from "personal_losses_dict" and "business_losses_dict", and plot histograms for each pair in a row of subplots.
- Cumulative Distribution Plot:** Similarly, iterate over each pair, compute cumulative distribution using "np.cumsum", and plot the results.
- Polynomial Fitting Plot:** For each pair, fit a polynomial of degree 5 to the cumulative distribution and plot both the cumulative distribution and the fitted polynomial curve.

These modifications should adapt your existing plotting code to handle multiple pairs of rho values, reflecting the results from your simulation for each pair. Adjust colors ("C1") and other styling details as needed to match your preferences or to distinguish between different pairs clearly.

Q: A: P: Q: A:

Is this conversation helpful so far? ⌂ ⌂ X

