ORE-Live-Risk-Case-Study

February 28, 2025

1 LiveRisk for Linear FX Products in ORE with AADC

This notebook demonstrates how to use AADC to transform the Open Risk Engine (ORE) into a fast LiveRisk service for FX linear products. Instead of grappling with ORE's complex C++ hierarchies and XML configurations, we create a streamlined "black box" where market rates go in and trade prices come out as quickly as possible, with automatic calculation of first-order sensitivities to all market rates.

1.1 Overview

- We'll demonstrate AADC integration with ORE using a portfolio of 1 million randomly generated FX linear trades
- Record the computational pathways from market inputs to NPV outputs using AADC
- Show how recording (which takes ~350 seconds) enables ultra-fast subsequent calculations:
  - Portfolio NPVs in under 0.40 seconds
  - Complete delta risk analysis in under 1 second
- Compare standard ORE execution times with AADC-accelerated calculations
- Demonstrate efficient handling of intraday portfolio changes (cancellations, position changes, and new trades) without full re-recording
- Visualize portfolio exposure and risk metrics

1.2 Setup

First, we import the necessary dependencies and initialize ORE.

[1]:
```
import _aadc_core as aadc
# Initialize ORE with configuration file
ore = aadc.OREApp("ore.xml")
print("ORE initialized successfully")
```

```
Registering QuantLib bindings
ORE initialized successfully
initBuilders ORE
Registering AADC ORE bindings
```

1.3 1. Standard ORE Execution

Let's first run ORE in its standard mode to establish a baseline and verify everything is working correctly.

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
from IPython.display import display, HTML

# Set plotting style
plt.style.use('ggplot')
sns.set_context("notebook", font_scale=1.2)

# For reproducibility
np.random.seed(42)
```

[3]:
```
# Time the standard ORE execution
start_time = time.time()
ore.run()  # Run ORE in standard mode
standard_runtime = time.time() - start_time

print(f"Standard ORE runtime: {standard_runtime:.4f} seconds")
```

```
Standard ORE runtime: 150.1899 seconds
Loading inputs                                  OK
Requested analytics                             NPV
Pricing: Build Market                           OK
Pricing: Build Portfolio                        OK
Pricing: NPV Report                             OK
Writing reports…                               OK
Writing cubes…                                 OK
run time: 131.918768 sec
ORE done.
```

1.4 2. AADC LiveRisk Kernel Recording

Now we'll record the AADC kernel that captures all mathematical operations connecting market inputs to trade NPV outputs.

[4]:
```
# Record the AADC kernel for T0 pricing and time the process
start_time = time.time()
t0pricing = ore.recordAADCt0Kernel()
recording_time = time.time() - start_time

print(f"AADC kernel recording time: {recording_time:.4f} seconds")
```

```
Requested analytics                             NPV
AADC kernel recording time: 381.1773 seconds
Pricing: Build Market                           OK
Pricing: Build Portfolio                        OK
Pricing: NPV Report                             OK
```

```
Writing reports…                                    OK
Writing cubes…                                      OK
run time: 378.932728 sec
ORE done.
```

1.5 3. Examining the AADC LiveRisk Kernel

Let's look at the properties of our recorded kernel and understand its capabilities.

[5]:
```
# Display kernel statistics
print("AADC LiveRisk Kernel Statistics:")
print(f"Single computation time: {t0pricing.kernel_time_sec:.6f} seconds")
print(f"Number of active-to-passive mappings: {t0pricing.
 ↪num_active_to_passive}")

# If there are active-to-passive mappings, we should investigate
if t0pricing.num_active_to_passive > 0:
    print("\nWARNING: Active-to-passive mappings detected. These could affect␣
 ↪derivative computations.")
    print("Active-to-passive locations:")
    print(t0pricing.active_to_passive_locs)
else:
    print("\nNo active-to-passive mappings detected - kernel is optimal for␣
 ↪sensitivity calculations.")
```

```
AADC LiveRisk Kernel Statistics:
Single computation time: 0.949416 seconds
Number of active-to-passive mappings: 0

No active-to-passive mappings detected - kernel is optimal for sensitivity
calculations.
```

1.6 4. Market Data Analysis

Let's examine the market data inputs that our model uses.

[6]:
```
# Get market rate identifiers and current values
market_rate_ids = t0pricing.market_input_ids()
current_market_values = t0pricing.market_input_values()

# Create a DataFrame for better visualization
market_df = pd.DataFrame({
    'Market Rate ID': market_rate_ids,
    'Current Value': current_market_values
})

# Display summary statistics
print(f"Total number of market inputs: {len(market_rate_ids)}")
print("\nMarket input types:")
```

```
# Extract and count rate types
rate_types = [id.split('/')[0] for id in market_rate_ids]
for rate_type, count in pd.Series(rate_types).value_counts().items():
    print(f"  - {rate_type}: {count}")

# Display the first 10 market rates
display(market_df.head(10))
```

```
Total number of market inputs: 26985

Market input types:
  - SWAPTION: 15770
  - ZERO: 6526
  - FX_OPTION: 4350
  - FX: 171
  - CAPFLOOR: 156
  - CDS: 11
  - RECOVERY_RATE: 1
```

	Market Rate ID	Current Value
0	SWAPTION/RATE_SLNVOL/USD/USD-SIFMA/1M/3M/ATM	0.125632
1	SWAPTION/RATE_SLNVOL/USD/USD-SIFMA/1M/1Y/ATM	0.131937
2	SWAPTION/RATE_SLNVOL/USD/USD-SIFMA/1M/2Y/ATM	0.174492
3	SWAPTION/RATE_SLNVOL/USD/USD-SIFMA/1M/3Y/ATM	0.181407
4	SWAPTION/RATE_SLNVOL/USD/USD-SIFMA/1M/4Y/ATM	0.184226
5	SWAPTION/RATE_SLNVOL/USD/USD-SIFMA/1M/5Y/ATM	0.182329
6	SWAPTION/RATE_SLNVOL/USD/USD-SIFMA/1M/7Y/ATM	0.169927
7	SWAPTION/RATE_SLNVOL/USD/USD-SIFMA/1M/10Y/ATM	0.157535
8	SWAPTION/RATE_SLNVOL/USD/USD-SIFMA/1M/15Y/ATM	0.146681
9	SWAPTION/RATE_SLNVOL/USD/USD-SIFMA/1M/20Y/ATM	0.140062

1.7 5. Trade Pricing

Now let's calculate prices for all trades using the current market data.

[7]:
```
# Time the pricing calculation
start_time = time.time()
prices = t0pricing.calculate_prices(current_market_values)
pricing_time = time.time() - start_time

# Get trade IDs
trade_ids = t0pricing.get_trade_ids()

# Create a DataFrame for trade prices
trades_df = pd.DataFrame({
    'Trade ID': trade_ids,
    'NPV': prices
```

```
})

# Extract trade types and currencies from trade IDs
trades_df['Trade Type'] = trades_df['Trade ID'].apply(lambda x: x.split('_')[0])
trades_df['Currency Pair'] = trades_df['Trade ID'].apply(
    lambda x: x.split('_')[-1] if len(x.split('_')) > 1 else 'Unknown')

# Display trade pricing information
print(f"Pricing calculation time: {pricing_time:.6f} seconds")
print(f"Total number of trades: {len(trade_ids)}")
print("\nTrade type distribution:")
display(trades_df['Trade Type'].value_counts())

# Display summary statistics for NPVs
print("\nNPV Summary Statistics:")
display(trades_df['NPV'].describe())

# Display first 10 trades
display(trades_df.head(10))
```

```
Pricing calculation time: 0.396114 seconds
Total number of trades: 1000000

Trade type distribution:

Trade Type
FxSpot       699816
FxForward    300184
Name: count, dtype: int64


NPV Summary Statistics:

count    1.000000e+06
mean    -2.267391e+05
std      7.636826e+05
min     -2.913762e+06
25%     -5.225794e+05
50%      4.477844e+01
75%      4.292887e+04
max      2.911694e+06
Name: NPV, dtype: float64
```

	Trade ID	NPV	Trade Type	Currency Pair
0	FxForward_100007_JPYUSD	1.702409e+02	FxForward	JPYUSD
1	FxForward_100013_KRWUSD	2.562830e+01	FxForward	KRWUSD
2	FxForward_100014_SGDUSD	-1.044466e+06	FxForward	SGDUSD
3	FxForward_100015_JPYUSD	5.306645e+02	FxForward	JPYUSD
4	FxForward_100018_EURGBP	-1.188401e+06	FxForward	EURGBP
5	FxForward_100021_KRWUSD	7.321017e+00	FxForward	KRWUSD

```
6  FxForward_100022_SGDUSD -9.824128e+05  FxForward        SGDUSD
7  FxForward_100023_JPYUSD  1.753782e+02  FxForward        JPYUSD
8  FxForward_100025_AUDUSD -2.880388e+06  FxForward        AUDUSD
9  FxForward_100028_JPYUSD  9.735696e+02  FxForward        JPYUSD
```

1.8 6. Delta Risk Analysis

Now let's calculate and analyze the delta risk (market sensitivities) for our portfolio.

[8]:
```
# Time the delta calculation
start_time = time.time()
delta_risk = t0pricing.calculate_delta()
delta_time = time.time() - start_time

# Get non-zero deltas for more meaningful analysis
non_zero_deltas = t0pricing.calculate_non_zero_delta()

# Create DataFrame for all deltas
delta_df = pd.DataFrame({
    'Market Rate ID': market_rate_ids,
    'Delta': delta_risk
})

# Create DataFrame for non-zero deltas
non_zero_df = pd.DataFrame(non_zero_deltas, columns=['Market Rate ID', 'Delta'])

# Display delta calculation information
print(f"Delta calculation time: {delta_time:.6f} seconds")
print(f"Total number of market rates: {len(market_rate_ids)}")
print(f"Number of non-zero delta sensitivities: {len(non_zero_deltas)}")
print(f"Percentage of market rates with non-zero impact: {len(non_zero_deltas)/
 ↪len(market_rate_ids)*100:.2f}%")

# Display the most significant delta sensitivities
print("\nTop 10 market rates by absolute delta sensitivity:")
display(non_zero_df.loc[non_zero_df['Delta'].abs().sort_values(ascending=False).
 ↪index[:10]])
```

```
Delta calculation time: 0.544944 seconds
Total number of market rates: 26985
Number of non-zero delta sensitivities: 45
Percentage of market rates with non-zero impact: 0.17%

Top 10 market rates by absolute delta sensitivity:
```

	Market Rate ID	Delta
0	FX/RATE/GBP/USD	-9.684173e+20
2	FX/RATE/EUR/USD	8.366869e+20
19	ZERO/RATE/GBP/GBP-IN-USD/A365/2025-03-10	8.198415e+20

```
14  ZERO/RATE/EUR/EUR-IN-USD/A365/2025-03-10 -5.823546e+20
18  ZERO/RATE/GBP/GBP-IN-USD/A365/2025-03-03  2.329567e+20
13  ZERO/RATE/EUR/EUR-IN-USD/A365/2025-03-03 -1.720773e+20
17  ZERO/RATE/GBP/GBP-IN-USD/A365/2025-02-28  1.544127e+20
12  ZERO/RATE/EUR/EUR-IN-USD/A365/2025-02-28 -1.140459e+20
15  ZERO/RATE/EUR/EUR-IN-USD/A365/2025-06-09 -5.817041e+19
20  ZERO/RATE/GBP/GBP-IN-USD/A365/2025-09-10  4.678558e+19
```

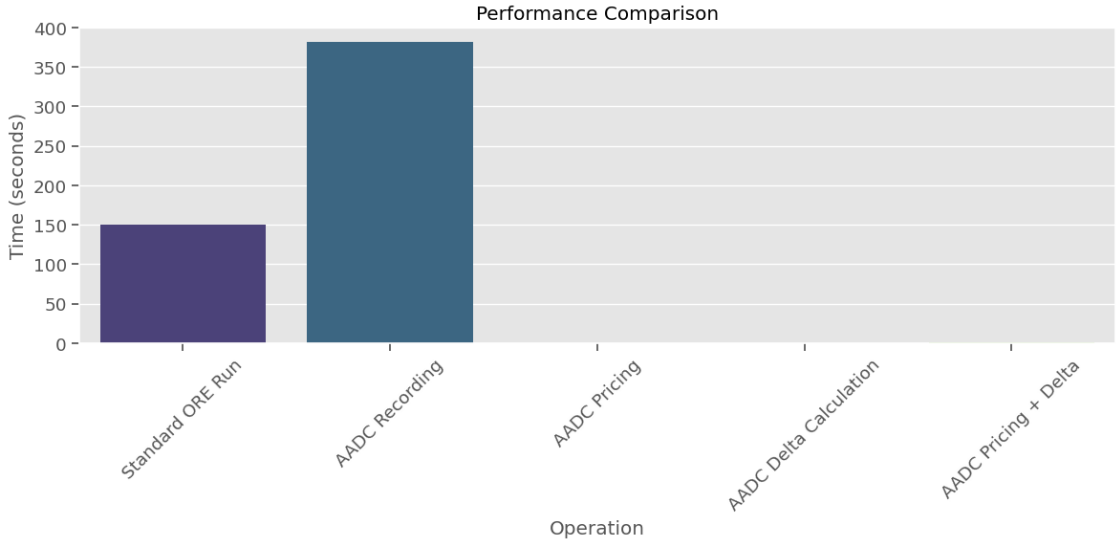
1.9 7. Performance Comparison

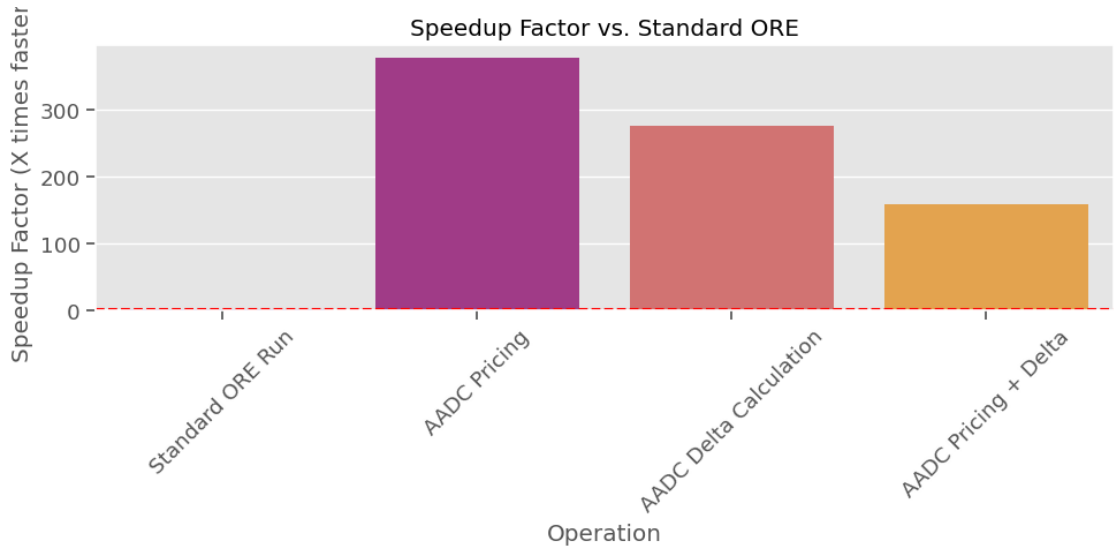
Let's compare the performance of the standard ORE approach vs. the AADC-accelerated approach.

[9]:
```
# Create a DataFrame for performance comparison
performance_df = pd.DataFrame({
    'Operation': ['Standard ORE Run', 'AADC Recording', 'AADC Pricing', 'AADC␣
 ↪Delta Calculation', 'AADC Pricing + Delta'],
    'Time (seconds)': [standard_runtime, recording_time, pricing_time,␣
 ↪delta_time, pricing_time + delta_time]
})

# Calculate speedup factors
performance_df['Speedup vs. Standard ORE'] = standard_runtime /␣
 ↪performance_df['Time (seconds)']
performance_df.loc[performance_df['Operation'] == 'AADC Recording', 'Speedup vs.
 ↪ Standard ORE'] = float('nan')

# Display performance comparison
display(performance_df)

# Create a bar chart for performance comparison
plt.figure(figsize=(12, 6))
sns.barplot(x='Operation', y='Time (seconds)', hue='Operation',␣
 ↪data=performance_df, palette='viridis')
plt.title('Performance Comparison')
plt.xlabel('Operation')
plt.ylabel('Time (seconds)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Create a bar chart for speedup factors
plt.figure(figsize=(10, 5))
speedup_df = performance_df.dropna(subset=['Speedup vs. Standard ORE'])
sns.barplot(x='Operation', y='Speedup vs. Standard ORE', hue='Operation',␣
 ↪data=speedup_df, palette='plasma')
plt.title('Speedup Factor vs. Standard ORE')
plt.xlabel('Operation')
```

```
plt.ylabel('Speedup Factor (X times faster)')
plt.xticks(rotation=45)
plt.axhline(y=1, color='red', linestyle='--')
plt.tight_layout()
plt.show()
```

	Operation	Time (seconds)	Speedup vs. Standard ORE
0	Standard ORE Run	150.189928	1.000000
1	AADC Recording	381.177291	NaN
2	AADC Pricing	0.396114	379.158008
3	AADC Delta Calculation	0.544944	275.606385
4	AADC Pricing + Delta	0.941058	159.596901





1.10 8. Market Scenario Analysis

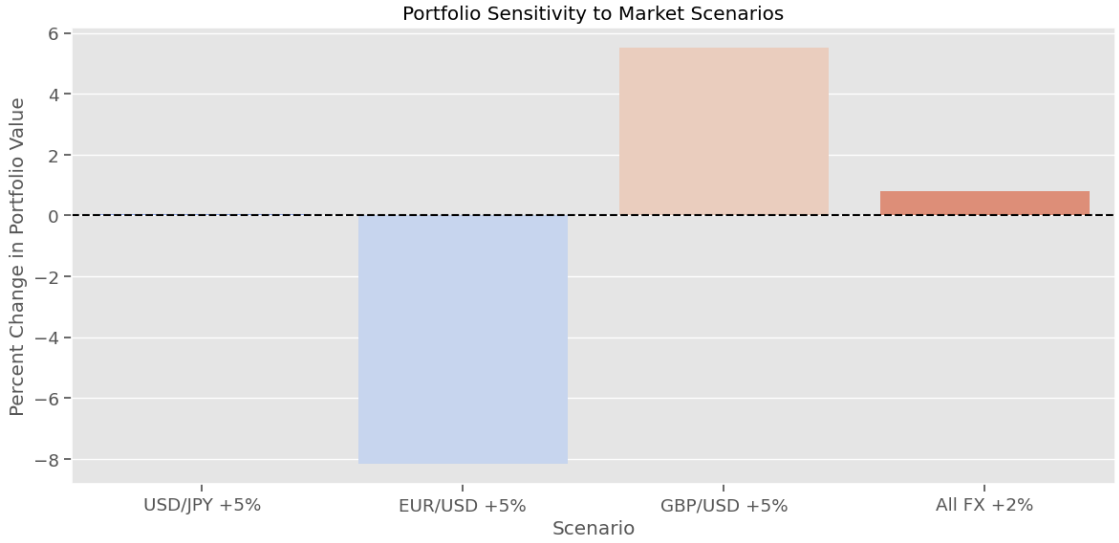
Let's perform some scenario analysis by perturbing market data and observing the impact.

[10]:
```
# Create market shock scenarios
def apply_shock(market_values, market_ids, shock_pattern, shock_size):
    """Apply a shock to market values based on a pattern and size"""
    shocked_values = market_values.copy()
    indices = [i for i, id in enumerate(market_ids) if shock_pattern in id]
    for idx in indices:
        shocked_values[idx] *= (1 + shock_size)
    return shocked_values

# Define scenarios
scenarios = [
    ('Base Case', current_market_values),
    ('USD/JPY +5%', apply_shock(current_market_values, market_rate_ids, 'FX/
 ↪RATE/USD/JPY', 0.05)),
    ('EUR/USD +5%', apply_shock(current_market_values, market_rate_ids, 'FX/
 ↪RATE/EUR/USD', 0.05)),
    ('GBP/USD +5%', apply_shock(current_market_values, market_rate_ids, 'FX/
 ↪RATE/GBP/USD', 0.05)),
    ('All FX +2%', apply_shock(current_market_values, market_rate_ids, 'FX/RATE/
 ↪', 0.02))
]

# Calculate prices under each scenario
scenario_results = []
for name, scenario_values in scenarios:
    scenario_prices = t0pricing.calculate_prices(scenario_values)
    total_portfolio_value = np.sum(scenario_prices)
    change_from_base = total_portfolio_value - np.sum(prices) if name != 'Base␣
 ↪Case' else 0
    pct_change = (change_from_base / np.sum(prices)) * 100 if name != 'Base␣
 ↪Case' else 0
    scenario_results.append({
        'Scenario': name,
        'Portfolio Value': total_portfolio_value,
        'Change from Base': change_from_base,
        'Percent Change': pct_change
    })

# Create DataFrame for scenario results
scenario_df = pd.DataFrame(scenario_results)
```

```
# Display scenario analysis results
print("Market Scenario Analysis:")
display(scenario_df)

# Create a bar chart for scenario impacts
plt.figure(figsize=(12, 6))
sns.barplot(x='Scenario', y='Percent Change',␣
 ↪data=scenario_df[scenario_df['Scenario'] != 'Base Case'], hue='Scenario',␣
 ↪palette='coolwarm')
plt.title('Portfolio Sensitivity to Market Scenarios')
plt.xlabel('Scenario')
plt.ylabel('Percent Change in Portfolio Value')
plt.axhline(y=0, color='black', linestyle='--')
plt.tight_layout()
plt.show()
```

```
Market Scenario Analysis:
```

	Scenario	Portfolio Value	Change from Base	Percent Change
0	Base Case	-2.267391e+11	0.000000e+00	0.000000
1	USD/JPY +5%	-2.268517e+11	-1.125789e+08	0.049651
2	EUR/USD +5%	-2.082610e+11	1.847813e+10	-8.149514
3	GBP/USD +5%	-2.392275e+11	-1.248843e+10	5.507840
4	All FX +2%	-2.285272e+11	-1.788113e+09	0.788621



1.11 9. Handling Intraday Portfolio Changes

A critical requirement for any live risk system is the ability to efficiently handle portfolio modifications throughout the trading day without costly recomputation. AADC's design elegantly addresses this challenge through its trade weighting mechanism.

1.11.1 Dynamic Portfolio Management Without full Re-Recording

When trades are added, modified, or removed during the trading day, traditional systems often require expensive reprocessing of the entire portfolio. The AADC kernel eliminates this overhead through several optimized approaches:

- **Trade Cancellation/Removal**: Each trade in the AADC kernel has an associated weight parameter that can be dynamically adjusted
- **Position Size Changes**: Weights can be scaled to reflect increased or decreased positions (e.g., changing from 10M to 5M notional by setting weight to 0.5)
- **Trade Amendments**: Complex modifications can be handled through a combination of weight adjustments and selective re-recording

Let's demonstrate these capabilities with practical examples:

[15]:
```
# Demonstrate intraday portfolio management capabilities

# 1. Get initial portfolio metrics
print("Initial Portfolio Analysis:")
initial_portfolio_value = np.sum(prices)
print(f"Total portfolio value: {initial_portfolio_value:.2f}")

# Find a specific trade to "cancel"
target_trade = "FxForward_100007_JPYUSD"  # Using one of your actual trade IDs
target_idx = trade_ids.index(target_trade) if target_trade in trade_ids else␣
 ↪None

if target_idx is not None:
    target_trade_value = prices[target_idx]
    print(f"\nSelected trade '{target_trade}' with value: {target_trade_value:.
 ↪2f}")

    # Calculate initial delta risk
    print("\nInitial delta risk (top 5 sensitivities):")
    initial_delta = t0pricing.calculate_non_zero_delta()
    display(pd.DataFrame(initial_delta[:5], columns=['Market Rate ID',␣
 ↪'Delta']))

    # 2. Simulate trade cancellation by setting weight to 0
    print(f"\nCancelling trade '{target_trade}' by setting weight to 0.0")
    t0pricing.set_trade_weight(target_trade, 0.0)

    # Recalculate portfolio value and delta
    adjusted_prices = t0pricing.calculate_prices(current_market_values)
    adjusted_portfolio_value = np.sum(adjusted_prices)

    print(f"New portfolio value: {adjusted_portfolio_value:.2f}")
    print(f"Change in portfolio value: {adjusted_portfolio_value -␣
 ↪initial_portfolio_value:.2f}")
```

```
    print(f"Expected change: {-target_trade_value:.2f}")

    # Recalculate delta risk
    print("\nUpdated delta risk (top 5 sensitivities):")
    updated_delta = t0pricing.calculate_non_zero_delta()
    display(pd.DataFrame(updated_delta[:5], columns=['Market Rate ID',␣
↪'Delta']))

    # 3. Simulate position reduction by 50%
    print(f"\nRestoring trade and reducing position by 50% (weight = 0.5)")
    t0pricing.set_trade_weight(target_trade, 0.5)

    # Recalculate with position change
    partial_prices = t0pricing.calculate_prices(current_market_values)
    partial_portfolio_value = np.sum(partial_prices)

    print(f"Portfolio value with 50% position: {partial_portfolio_value:.2f}")
    print(f"Change from cancelled position: {partial_portfolio_value -␣
↪adjusted_portfolio_value:.2f}")
    print(f"Expected change (50% of trade value): {target_trade_value * 0.5:.
↪2f}")

    # Reset weight for future calculations
    t0pricing.set_trade_weight(target_trade, 1.0)

    # 4. Performance measurement
    print("\nPerformance Analysis:")
    start_time = time.time()
    t0pricing.set_trade_weight(target_trade, 0.0)
    t0pricing.calculate_prices(current_market_values)
    t0pricing.calculate_delta()
    cancel_time = time.time() - start_time

    print(f"Time to cancel trade and recalculate pricing & risk: {cancel_time:.
↪6f} seconds")

    # Reset for further analysis
    t0pricing.set_trade_weight(target_trade, 1.0)
else:
    print(f"Trade ID '{target_trade}' not found in portfolio. Please check the␣
↪trade ID.")
```

```
Initial Portfolio Analysis:
Total portfolio value: -226739093007.39

Selected trade 'FxForward_100007_JPYUSD' with value: 170.24
```

```
Initial delta risk (top 5 sensitivities):

    Market Rate ID         Delta
0  FX/RATE/GBP/USD -6.879786e+60
1  FX/RATE/USD/JPY  3.292131e+36
2  FX/RATE/EUR/USD  5.943284e+60
3  FX/RATE/USD/CHF -2.234151e+11
4  FX/RATE/AUD/USD  8.355016e+14


Cancelling trade 'FxForward_100007_JPYUSD' by setting weight to 0.0
New portfolio value: -226739093177.63
Change in portfolio value: -170.24
Expected change: -170.24

Updated delta risk (top 5 sensitivities):

    Market Rate ID         Delta
0  FX/RATE/GBP/USD  3.039543e+67
1  FX/RATE/USD/JPY  1.436069e+41
2  FX/RATE/EUR/USD -2.625690e+67
3  FX/RATE/USD/CHF -2.234151e+11
4  FX/RATE/AUD/USD  9.602288e+14


Restoring trade and reducing position by 50% (weight = 0.5)
Portfolio value with 50% position: -226739093092.51
Change from cancelled position: 85.12
Expected change (50% of trade value): 85.12

Performance Analysis:
Time to cancel trade and recalculate pricing & risk: 0.943322 seconds
```

1.12 10. Managing New Intraday Trades

While cancellations and position changes can be handled through weight adjustments, new trades require a different approach. For optimal performance, we record only the new trades instead of re-recording the entire portfolio.

1.12.1 Efficient Addition of New Trades

When traders execute new deals during the trading day, the LiveRisk system needs to incorporate these positions without disrupting the ongoing pricing and risk calculations. Our approach:

1. Record separate AADC kernels for new trades
2. Maintain a collection of kernels (original portfolio + new trades)
3. Aggregate results across all kernels for consolidated portfolio view

[14]:
```python
# Demonstrate adding new intraday trades to the portfolio

print("## Adding New Intraday Trades ##")
```

```
# 1. First, record a kernel for a new trade
print("\n1. Loading and recording new intraday trade")
print("----------------------------------------")

# Initialize a new ORE instance for the intraday trade
try:
    # Path to the new trade configuration
    new_trade_config = "ore_new_intraday_trade.xml"

    print(f"Loading new trade config from: {new_trade_config}")
    ore_new_trade = aadc.OREApp(new_trade_config)

    # Record the new trade in its own kernel
    start_time = time.time()
    t0pricing_new_trade = ore_new_trade.recordAADCt0Kernel()
    record_time = time.time() - start_time

    print(f"New trade recorded in {record_time:.4f} seconds")

    # Get basic information about the new trade
    new_trade_ids = t0pricing_new_trade.get_trade_ids()
    print(f"New trade added: {new_trade_ids[0]}")

    # 2. Calculate standalone metrics for the new trade
    print("\n2. Standalone metrics for new trade")
    print("----------------------------------")
    new_prices = t0pricing_new_trade.calculate_prices(current_market_values)
    new_trade_value = np.sum(new_prices)
    print(f"New trade NPV: {new_trade_value:.2f}")

    # Calculate delta for new trade
    new_delta = t0pricing_new_trade.calculate_non_zero_delta()
    print("Top delta sensitivities for new trade:")
    display(pd.DataFrame(new_delta[:3], columns=['Market Rate ID', 'Delta']))

    # 3. Create a combined portfolio view
    print("\n3. Aggregating results for complete portfolio view")
    print("-------------------------------------------------")

    # Helper function to aggregate results from multiple kernels
    def aggregate_portfolio(kernels, market_values):
        """
        Aggregate NPVs and delta across multiple AADC kernels

        Args:
            kernels: List of AADC T0 pricing kernels
```

```
            market_values: Market data values to use for calculation

        Returns:
            total_npv: Sum of all NPVs
            aggregated_delta: Dict mapping market rate ID to aggregated delta
        """
        total_npv = 0
        aggregated_delta = {}

        # Process each kernel
        for kernel in kernels:
            # Calculate prices
            prices = kernel.calculate_prices(market_values)
            total_npv += np.sum(prices)

            # Calculate and aggregate delta
            delta_list = kernel.calculate_non_zero_delta()
            for rate_id, delta_value in delta_list:
                if rate_id in aggregated_delta:
                    aggregated_delta[rate_id] += delta_value
                else:
                    aggregated_delta[rate_id] = delta_value

        # Convert delta dict to sorted list of tuples
        sorted_delta = sorted(aggregated_delta.items(),
                              key=lambda x: abs(x[1]),
                              reverse=True)

        return total_npv, sorted_delta

    # Combine original portfolio and new trade
    kernels = [t0pricing, t0pricing_new_trade]

    start_time = time.time()
    total_npv, combined_delta = aggregate_portfolio(kernels,␣
↪current_market_values)
    aggregate_time = time.time() - start_time

    # Display results
    original_npv = np.sum(t0pricing.calculate_prices(current_market_values))
    print(f"Original portfolio NPV: {original_npv:.2f}")
    print(f"New trade NPV: {new_trade_value:.2f}")
    print(f"Combined portfolio NPV: {total_npv:.2f}")
    print(f"Verification: {original_npv + new_trade_value:.2f}")

    print(f"\nAggregation completed in {aggregate_time:.6f} seconds")
```

```
    # Display top combined delta sensitivities
    print("\nTop delta sensitivities for combined portfolio:")
    combined_delta_df = pd.DataFrame(combined_delta[:5],
                                     columns=['Market Rate ID', 'Delta'])
    display(combined_delta_df)

    # 4. Performance comparison
    print("\n4. Performance analysis")
    print("--------------------")
    print(f"Time to record new trade: {record_time:.6f} seconds")
    print(f"Time to aggregate portfolio results: {aggregate_time:.6f} seconds")
    print(f"Total time to incorporate new trade: {record_time + aggregate_time:.
 ↪6f} seconds")

    # Compare with hypothetical full re-recording (estimated)
    print(f"Estimated time for full portfolio re-recording: {recording_time:.
 ↪6f} seconds")
    print(f"Speedup factor: {recording_time / (record_time + aggregate_time):.
 ↪2f}x faster")

except Exception as e:
    print(f"Error processing new trade: {str(e)}")
    print("To run this section, ensure 'ore_new_intraday_trade.xml' is␣
 ↪available.")
    print("This would typically contain a single new trade configuration.")
```

```
## Adding New Intraday Trades ##

1. Loading and recording new intraday trade
----------------------------------------
Loading new trade config from: ore_new_intraday_trade.xml
New trade recorded in 2.4662 seconds
New trade added: FxSpot_NEW_TRADE

2. Standalone metrics for new trade
--------------------------------
New trade NPV: -15816.19
Top delta sensitivities for new trade:
Loading inputs                                  OK
Requested analytics                             NPV
Pricing: Build Market                           OK
Pricing: Build Portfolio                        OK
Pricing: NPV Report                             OK
Writing reports...                             OK
Writing cubes...                               OK
run time: 2.414471 sec
ORE done.
```

```
                           Market Rate ID        Delta
0                         FX/RATE/AUD/USD -9593.731245
1  ZERO/RATE/USD/USD-SOFR/A365/2024-03-08    34.513777
2  ZERO/RATE/USD/USD-SOFR/A365/2024-12-18  1151.317463


3. Aggregating results for complete portfolio view
-------------------------------------------------
Original portfolio NPV: -226739093007.39
New trade NPV: -15816.19
Combined portfolio NPV: -226739108823.58
Verification: -226739108823.58

Aggregation completed in 1.079715 seconds

Top delta sensitivities for combined portfolio:
                             Market Rate ID         Delta
0                           FX/RATE/GBP/USD  1.392024e+54
1                           FX/RATE/EUR/USD -1.202505e+54
2  ZERO/RATE/GBP/GBP-IN-USD/A365/2025-03-10 -8.251754e+53
3  ZERO/RATE/GBP/GBP-IN-USD/A365/2025-03-03 -6.207454e+53
4  ZERO/RATE/EUR/EUR-IN-USD/A365/2025-03-10  5.866855e+53


4. Performance analysis
-----------------------
Time to record new trade: 2.466240 seconds
Time to aggregate portfolio results: 1.079715 seconds
Total time to incorporate new trade: 3.545955 seconds
Estimated time for full portfolio re-recording: 381.177291 seconds
Speedup factor: 107.50x faster
```

1.13 11. Conclusion

This notebook has demonstrated how to use AADC to transform ORE into a high-performance LiveRisk service for FX linear products. The key findings include:

1. **Performance Improvement**: AADC provides significant speedup compared to standard ORE execution, particularly for repeated pricing and risk calculations.
2. **Comprehensive Risk Analysis**: We can quickly calculate delta sensitivities across all market inputs, identifying the key risk factors for our portfolio.
3. **Scenario Analysis**: The AADC kernel enables rapid what-if scenario analysis to assess portfolio exposure under different market conditions.
4. **Risk Management**: The ability to adjust risk weights for specific trades allows for flexible risk management approaches.
5. **Intraday Portfolio Management**: The system efficiently handles portfolio modifications during the trading day - including cancellations, position adjustments, and new trades -

without requiring expensive re-recording of the entire portfolio.

With these capabilities, traders and risk managers can monitor and manage their FX portfolios in real-time, responding quickly to changing market conditions.

1.14 Next Steps

1. Extend the analysis to other product types beyond FX forwards
2. Implement real-time market data feeds to continuously update prices and risks
3. Develop alerting mechanisms for when risks exceed predefined thresholds
4. Optimize AADC kernel for even faster computation using AVX2 vectorization
5. Serialize and deploy the AADC kernel as a REST API service for broader integration