Turning ORE into a LiveRisk Service for Linear FX Products with AADC

ORE (Open Risk Engine) built on top of QuantLib presents a formidable challenge for quants and developers. While powerful, its complexity can make even seemingly simple tasks like setting up a Monte Carlo model quite daunting. This blog post demonstrates how we can transform ORE into a streamlined LiveRisk service for basic FX linear products using AADC.

The Challenge

When working with ORE, we often face a steep learning curve: - Complex C++ class hierarchies - Intricate XML configuration requirements - Elaborate initialization sequences - Interconnected objects that must be precisely orchestrated

Our goal is simple: create a black box where market rates go in and trade prices come out as quickly as possible, with first-order sensitivities to all market rates via AAD.

Finding the Input/Output Points

Creating a TodaysMarket object in ORE typically requires extensive setup code. However, for our purposes, we need to identify where market data actually enters the system and where trade valuations exit.

After investigation, we found that market data enters through `CSVLoader::loadFile` with this simple pattern:

```
const string& key = tokens[1];
Real value = parseReal(tokens[2]);
```

This is where we can hook into AADC by marking these values as inputs:

```
if (idouble::isRecording()) {
    auto aadc_arg = value.markAsInput();
    AADCKernelInputOutputRegistryInstance().inputs.push_back({key, aadc_arg});
    AADCKernelInputOutputRegistryInstance().input_values.push_back(value.val);
}
```

For outputs, the `ReportWriter::writeNpv` method provides access to trade NPVs:

```
Real npv = trade->instrument()->NPV();
```

Which we can mark as outputs:

```
auto npv_res = (npv * fx).markAsOutput();
AADCKernelInputOutputRegistryInstance().outputs.push_back({trade->id(), npv_res});
```

Handling Control Flow Branches

When recording with AADC, it's crucial to identify any branches that depend on input values, as these can create discontinuities leading to unstable risk calculations and PnL jumps.

In our setup, AADC identified a potential issue in `Rounding::operator()`:

```
Decimal Rounding::operator()(Decimal value) const {
    if (type_ == None)
        return value;

    Real mult = std::pow(10.0,precision_);
    bool neg = AAD_PASSIVE((value < 0.0));
    // ...
}
```

To resolve this, we modified the method to bypass rounding when running under AADC:

```
if (idouble::isRecording()) return value;
```

After these changes, the AADC report confirms clean recording:

```
Number active to passive conversions: 0 while recording OREApp::run()
```

Performance Results

We tested the solution with 1 million randomly generated FX linear trades:

Operation	Time (seconds)
Standard ORE NPV calculation	98
AADC recording	350
NPV recalculation after recording	0.40
NPV + all portfolio delta risks	<1.0

The LiveRisk Approach

The beauty of this approach is that:

1. Recording can be done before the trading day begins
2. The computational graph can be serialized and cached
3. The resulting AADC kernel can be deployed to a cloud API
4. Each market data update triggers rapid recalculation (<1 second)
5. You get both prices AND sensitivities almost instantaneously
6. For new trades only incremental recording is needed
7. For trade cancellations, weight parameters can be adjusted without need for re-recording

Although this demonstration uses ORE/QL, the same technique can be applied to proprietary quant libraries. AADC's AVX2 vectorization capabilities (not leveraged in this example) can further accelerate calculations for delta ladders or what-if scenarios.

This approach effectively turns the complex ORE/QL infrastructure into a high-performance risk engine capable of handling real-time market data updates with comprehensive sensitivity and scenario analysis.