QuantLibAdjoint News

Alexander Sokol
Head of Quant Research, CompatibL

*TapeScript and QuantLibAdjoint are open source
and will always remain free for commercial use
Download from `http://github.com/compatibl`

QuantLib User Conference, London, July 12, 2016

Possible ways to implement AAD

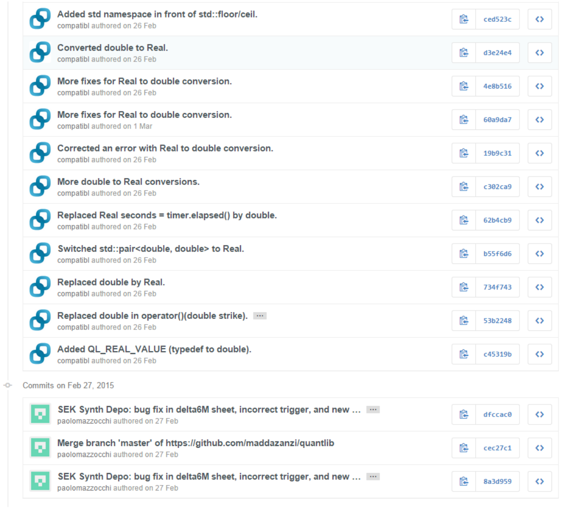
1. **Hand-coding** refers to the approach of manually adding AAD logic directly to the source code.
    - The speaker believes that hand-coding and hand-validating AAD is a Herculean task that would take unacceptably long, especially when also having to merge with the evolving code base.
2. **Automated source transformation** (e.g. Tapenade) refers to the approach where AAD logic is also inserted directly into the code, but unlike with hand-coding, it is done by an automated source code transformation.
    - Source transformation is not as labor intensive than hand-coding, however it is still mostly suitable for smaller, research-style code and will not work well for a large scale, modular quantitative analytics library with high degree of encapsulation via interfaces
3. **Operator overloading** (e.g. CppAD, ADOL-C, ADEPT, TapeScript) refers to the approach where the native double is replaced by a class with overloaded arithmetic operators.
    - The only approach that preserves the original code (except for defining Real as tdouble)

COMPATIBL

Using operator overloading approach in QuantLib

- Operator overloading approach in QuantLib works by defining Real as tdouble
- Back in year 2000, Luigi Ballabio and Ferdinando Ametrano decided to use Real instead of double throughout QuantLib to enable a compile time switch between single and double precision
- 16 years later this helped avoid the global replacement of double in QuantLib code to implement AAD
- By defining Real as tdouble, only 461 lines of the code had to be changed to make QuantLib AAD-capable (see github.com/compatibl/QuantLibAdjoint)
- These 461 lines of code changes are already merged into QuantLib master

Some of the changes in converting a QuantLib to AAD



Added std namespace in front of std::floor/ceil.
compatibl authored on 26 Feb — ced523c

Converted double to Real.
compatibl authored on 26 Feb — d3e24e4

More fixes for Real to double conversion.
compatibl authored on 26 Feb — 4e8b516

More fixes for Real to double conversion.
compatibl authored on 1 Mar — 60a9da7

Corrected an error with Real to double conversion.
compatibl authored on 26 Feb — 19b9c31

More double to Real conversions.
compatibl authored on 26 Feb — c302ca9

Replaced Real seconds = timer.elapsed() by double.
compatibl authored on 26 Feb — 62b4cb9

Switched std::pair<double, double> to Real.
compatibl authored on 26 Feb — b55f6d6

Replaced double by Real.
compatibl authored on 26 Feb — 734f743

Replaced double in operator()(double strike). …
compatibl authored on 26 Feb — 53b2248

Added QL_REAL_VALUE (typedef to double).
compatibl authored on 26 Feb — c45319b

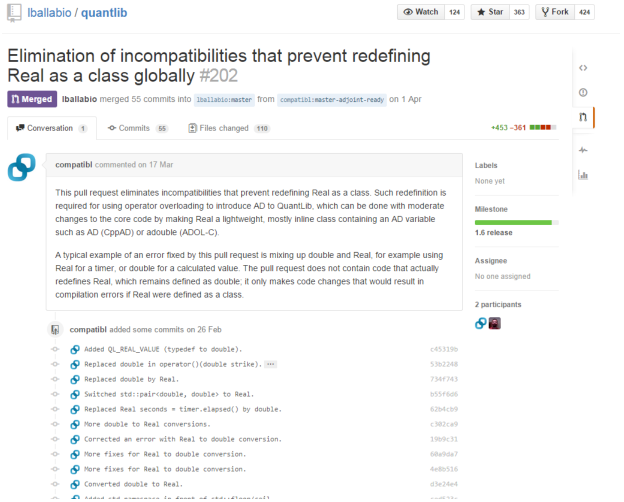
Commits on Feb 27, 2015

SEK Synth Depo: bug fix in delta6M sheet, incorrect trigger, and new … …
paolomazzocchi authored on 27 Feb — dfccac0

Merge branch 'master' of https://github.com/maddazanzi/quantlib
paolomazzocchi authored on 27 Feb — cec27c1

SEK Synth Depo: bug fix in delta6M sheet, incorrect trigger, and new … …
paolomazzocchi authored on 27 Feb — 8a3d959

QuantLib pull request: 461 lines of code changed



Elimination of incompatibilities that prevent redefining Real as a class globally #202

lballabio merged 55 commits into `lballabio:master` from `compatibl:master-adjoint-ready` on 1 Apr

compatibl commented on 17 Mar

This pull request eliminates incompatibilities that prevent redefining Real as a class. Such redefinition is required for using operator overloading to introduce AD to QuantLib, which can be done with moderate changes to the core code by making Real a lightweight, mostly inline class containing an AD variable such as AD (CppAD) or adouble (ADOL-C).

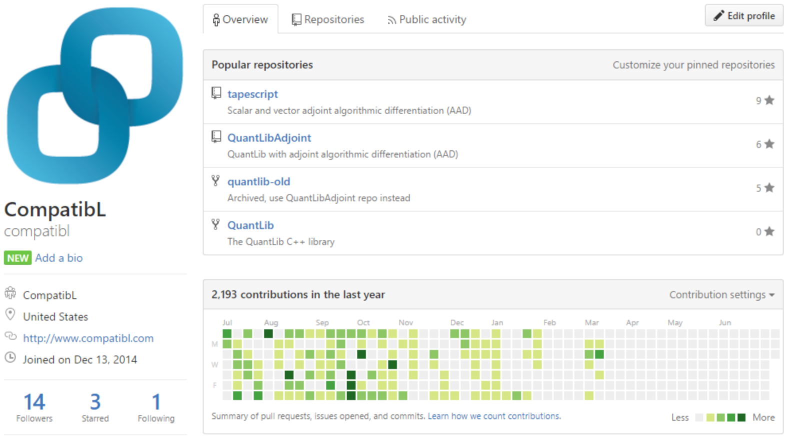
A typical example of an error fixed by this pull request is mixing up double and Real, for example using Real for a timer, or double for a calculated value. The pull request does not contain code that actually redefines Real, which remains defined as double; it only makes code changes that would result in compilation errors if Real were defined as a class.

compatibl added some commits on 26 Feb

```
Added QL_REAL_VALUE (typedef to double).                  c45319b
Replaced double in operator()(double strike). …           53b2248
Replaced double by Real.                                  734f743
Switched std::pair<double, double> to Real.               b55f6d6
Replaced Real seconds = timer.elapsed() by double.        62b4cb9
More double to Real conversions.                          c302ca9
Corrected an error with Real to double conversion.        19b9c31
More fixes for Real to double conversion.                 60a9da7
More fixes for Real to double conversion.                 4e8b516
Converted double to Real.                                 d3e24e4
Added std namespace in front of std::floor/ceil
```

COMPATIBL

GitHub page with converted QuantLib and TapeScript



Overview Repositories Public activity Edit profile

CompatibL
compatibl
NEW Add a bio

CompatibL
United States
http://www.compatibl.com
Joined on Dec 13, 2014

14 Followers 3 Starred 1 Following

Popular repositories — Customize your pinned repositories

tapescript	Scalar and vector adjoint algorithmic differentiation (AAD)	9 ★
QuantLibAdjoint	QuantLib with adjoint algorithmic differentiation (AAD)	6 ★
quantlib-old	Archived, use QuantLibAdjoint repo instead	5 ★
QuantLib	The QuantLib C++ library	0 ★

2,193 contributions in the last year — Contribution settings

Summary of pull requests, issues opened, and commits. Learn how we count contributions.

Less ⬜ 🟩 🟩 🟩 🟩 More

Using tcontext to register variables when the tape is already running

- Traditional API for AAD requires that the entire calculation is formulated as a function with all inputs in the beginning and all outputs at the end
- Most AAD libraries do not allow registering variables when the tape is already being recorded
- However in production quant libraries, this is exactly what we need
- For example, we want to get rate quotes from the database, register, build the curve, then get vol quotes from the database, register, calibrate the model, etc.
- The tcontext API makes it possible to do this, minimizing the changes to the existing code

DEMO OF REGISTERING VARIABLES WHEN THE TAPE IS ALREADY RUNNING

Selectively excluding classes from AAD

- Peter Caspers converted part of QuantLib to AAD using a template-based approach
- If each QuantLib class is converted into a template parametrized by either double or the adjoint variable (e.g. tdouble), the overhead of tdouble is avoided in classes where AAD is not needed
- However converting each QuantLib class into a template is a massive code change, and will slow down the compilation/increase object code bloat.
- We developed an alternative way to selectively exclude classes from AAD, based on using one of two different headers.
- One of the headers compiles code with tdouble, and the other with an inline wrapper that has no performance overhead relative to double as it is optimized away by the compiler.

DEMO OF SELECTIVE EXCLUSION OF CLASSES FROM AAD

Vector AAD

- The objective of Vector AAD is to reduce tape size and accelerate tape recording
- With vector tape, a single tape slot (address) can refer to either a single number or an array of numbers
    - When using operator overloading approach, the AD variable holds a discriminated union of a single number and an immutable array, and a single tape address.
- The calculation can begin and end as usual, with individual inputs (independent variables) and outputs (dependent variables) being single numbers, not arrays.
    - It is also possible to use array inputs and outputs. This is logically equivalent to working with individual elements.
- In the middle of the calculation, if encountering a segment where the same set of operations is performed for an array of values (e.g. Monte Carlo simulation), the individual numbers are merged into a tarray and from this moment on they have a shared tape
- Later, they can be split back into individual numbers for separate processing. This splits the shared tape into branches.

TapeScript API for Vector AAD

- The TapeScript Vector AD variable `tobject` holds a composite variable `tvalue` instead of a single number, and also a single tape address.
- This composite variable (`tvalue`) is a discriminated union of a single number (`double`) and an array of numbers (`double[]`). The array is immutable.
- The universal variable `tobject` can be converted to and from either `tdouble` or an immutable vector variable, `tarray`.
- Conversion from `tdouble[]` to `tarray` is done via constructor
- Conversion from `tarray` to `tdouble[]` is done via method split()

Vector Tape with TapeScript

Vector tape for $y = 2x_0 + x_1$:
Forward mode:

```
Op#  Var# Op    Operands     Calculated
1    1    Init               value = { 0.976, 1.86 }  fwd[1] = { 2.06, 7.16 }
2    2    Init               value = { 4.3, 6.89 }    fwd[1] = { 0.898, 6.95 }
3    3    *     2     var#1  value = { 1.95, 3.71 }   fwd[1] = { 4.11, 14.3 }
4    4    +     var#3 var#2  value = { 6.26, 10.6 }   fwd[1] = { 5.01, 21.3 }
5         End
```

Reverse mode:

```
Op#  Var# Op    Operands     Calculated
4    4    +     var#3  var#2  value = { 6.26, 10.6 }   rev[1] = { -1.53, 2.47 }
3    3    *     2      var#1  value = { 1.95, 3.71 }   rev[1] = { -1.53, 2.47 }
2    2    Init                value = { 4.3, 6.89 }    rev[1] = { -1.53, 2.47 }
1    1    Init                value = { 0.976, 1.86 }  rev[1] = { -3.05, 4.94 }
0         Begin
```

The Same Calculation with Scalar Tape

This is the scalar tape for the same function: Forward mode:

```
Op#  Var# Op    Operands      Calculated
1    1    Init                 value = 0.976     fwd[1] = 2.06
2    2    Init                 value = 1.86      fwd[1] = 7.16
3    3    Init                 value = 4.3       fwd[1] = 0.898
4    4    Init                 value = 6.89      fwd[1] = 6.95
5    5    *     2       var#1  value = 1.95      fwd[1] = 4.11
6    6    +     var#5   var#3  value = 6.26      fwd[1] = 5.01
7    7    *     2       var#2  value = 3.71      fwd[1] = 14.3
8    8    +     var#7   var#4  value = 10.6      fwd[1] = 21.3
9         End
```

Reverse mode:

```
Op#  Var# Op     Operands        Calculated
8    8    +      var#7    var#4   value = 10.6    rev[1] = 2.47
7    7    *      2        var#2   value = 3.71    rev[1] = 2.47
6    6    +      var#5    var#3   value = 6.26    rev[1] = -1.53
5    5    *      2        var#1   value = 1.95    rev[1] = -1.53
4    4    Init                    value = 6.89    rev[1] = 2.47
3    3    Init                    value = 4.3     rev[1] = -1.53
2    2    Init                    value = 1.86    rev[1] = 4.94
1    1    Init                    value = 0.976   rev[1] = -3.05
0         Begin
```

Tape size with Vector AAD vs. Scalar AAD

```
Vector Tape:
Monte Carlo Paths: 10,000
Tape size (bytes): 3,473

Scalar Tape:
Monte Carlo Paths: 10,000
Tape size (bytes): 9,364,122
```

Performance when not recording the tape with Vector AAD

Vector AD class wrapper has no measurable performance impact **when not recording the tape** in the following two examples implemented with TapeScript, because the array inside the composite variable is an array of native doubles that the compiler can optimize

- Calculation of the dot product of two vectors (size 100,000,000)

```
Time for tarray 10,920 ticks
Time for double[] 10,795 ticks
Relative difference  1.1\%
```

- Calculation using special functions:

```
Time for tarray 14,820 ticks
Time for double[] 14,680 ticks
Relative difference  0.9\%
```

Performance Improvement with Vector AAD

- As Peter Caspers observed during QuantLib User Meeting 2015, compiler does less optimization with the AAD variable compared to native double
- The worst case cited in Peter’s presentation is x80 slowdown compared to native double for a Bermudan swaption convolution engine
- Our testing shows that such slowdown is extremely rare in QuantLib, and typically the overhead is between x2 and x10
- In the example with multiple lattice methods we have just seen, it is close to x5
- This means that typically you still win overall compared to bump and reprice if you need more than 5-10 greeks
- It is enough to have a single curve with bucket sensitivities for AAD to win
- However even this moderate overhead can be reduced even further with Vector AAD, as an additional welcome side effect to other advantages of Vector AAD

Can we use Vector AAD in QuantLib?

- Vector AAD is effective in two cases:
    1. When the same operation is applied to each element of an array, for example in case of tree or Monte Carlo calculation with time slices
    2. When atomics operating on vectors are available, e.g. for American Monte Carlo
- The part of QuantLib code where Vector AAD is used must be converted to a new class RealArray instead of std::vector<Real>
- Properties of RealArray
    1. Immutable, cannot change neither size or value
    2. Construct RealArray from std::vector<Real>
    3. Convert RealArray to std::vector<Real> using split() function
- When AAD is not used, the change from std::vector<Real> to RealArray will slow down the original code somewhat due to additional deep copy operations involved operations
- Using expression templates can help minimize the impact
- CompatibL plans to first use it for adding new Monte Carlo functionality to QuantLib, and then retrofit it into the existing lattice code
- This feature will be in QuantLibAdjoint 3.0

How to download and compile QuantLibAdjoint

- Download QuantLibAdjoint from **github.com/compatibl**
  - Note: Master branch runs with Visual Studio 2013 and 2015. Support for Visual Studio 2012 and gcc is forthcoming.
- Compile and run test-suite and test-suite-adjoint
- Replace double with Real in your code
  - A number of refactoring tools is available to do this automatically, but even a simple global replacement in editor works surprisingly well if you are willing to tolerate some mangling of comments
- Register your independent and dependent variables and calculate sensitivities
- Contact team@compatibl.com in case of any problems

Summary

- **QuantLibAdjoint 1.0** is the stable release
  - Implements Scalar AAD
  - Already used in production by multiple firms
  - High performance with minimal use of RAM for calculations not involving trees or Monte Carlo
- **QuantLibAdjoint 2.0** implements the following new features
  - Registering variables when tape is being recorded (tcontext)
  - Selective use of AAD variable without templates
  - Supports Vector AAD mode with RealArray, however RealArray is not yet used as this would require changes to core QuantLib that require QuantLib community consensus
- **TapeLib** is a commercial extension of QuantLibAdjoint from CompatibL.
  - Parallel recording and playback of the tape
  - Serialization of tape and playback variables to disk
  - Adjoint Profiler for analyzing performance bottlenecks
  - Adjoint Pivot, an HTML5 front end for interactive drilldown from portfolio to trade level sensitivities using on the fly AAD calculation

Contact haining@compatibl.com for the details about TapeLib