AUTOMATIC DIFFERENTIATION CASE STUDY: AMERICAN OPTION FINITE DIFFERENCE PDE PRICING WITH CPPAD, ADOL-C AND OPENAD/F

P. CASPERS

first version July 15, 2015 - this version July 15, 2015
preliminary draft, has issues

Abstract. This is a case study in computing adjoint derivatives with (a) operator overloading using CppAD [1] and ADOL-C [2] and (b) source code transformation using OpenAD/F [3] for an american option priced with a finite difference pde.

Contents

1. Sample Problem

We consider a local volatility asset process

$$dS = \sigma(S,t) S dW \tag{1.1}$$

$$d\log(S) = -\frac{1}{2}\sigma(S,t)^2 dt + \sigma(S,t) dW \tag{1.2}$$

Date: July 15, 2015.

and the valuation of an american call option with maturity T and strike K in this model. The call price $c = c(t, X), X := \log(S)$ obeys

$$dc = \left(c_t + \frac{1}{2}\sigma(S,t)^2(c_{XX} - c_X) \right) dt + c_X \sigma(S,t) dW \tag{1.3}$$

from where we get the backward PDE

$$c_t + \frac{1}{2}\sigma(S,t)^2(c_{XX} - c_X) = 0 \tag{1.4}$$

with initial condition $c(T, X(T)) = \max\{e^{X(T)} - K, 0\}$ and update scheme $c(t, X(t)) \to \max\{c(t, X(t)), e^{X(t)} - K\}$.

We aim to compute the price using a simple explicit Euler finite difference scheme and the sensitivity to $\sigma(S,t)$ (vega) using the forward and reverse modes of [1], [2] and [3].

2. Test setup

We implement a solution to the problem using the high level objects of [4], in particular the finite difference framework and an american option pricing engine. This is solely to generate a reference price against we can check the other ad-hoc implementations.

Next we implement a low level C++ solution from scratch without any external libraries. The test code allows for three modes which are

(1) `PLAIN` compute the option price using the standard `double` type
(2) `CPPAD` compute the option price and reverse (forward) mode sensitivities to $\sigma(S,t)$ using CppAD
(3) `ADOLC` compute the option price and reverse mode sensitivities to $\sigma(S,t)$ using ADOL-C

The last implementation is in Fortran90 which can be fed into the source code transformation engine of OpenAD/F to generate a differentiated version of the pricing code. The implementation is separated into a core which is used from three drivers

(1) `driverf90_plain.f90` to compute the option price using the standard `double precision` type
(2) `driverf90_forward.f90` to compute forward mode sensitivities to $\sigma(S,t)$
(3) `dirverf90.f90` to compute reverse mode sensitivities to $\sigma(S,t)$

The parameters for the problem are chosen to be $T = 10$, $S(0) = 100$ and $K = 120$. The PDE uses 5000 time steps and 101 grid points to discretize $\log(S)$ in the interval $[1.07555172964, 8.13478864234]$ (these values are copied from the QuantLib engine we use which determines the bounds automatically). We use a uniform grid in time and spatial direction, while QuantLib uses a concentrating mesher around the strike for the asset grid, a minor difference which we ignore and which might explain the slight pricing differences we observe below.

We use a flat volatility $\sigma(S,t) = 0.2$ for the pricing, but interpret the volatility to be piecewise constant on a grid of 500 buckets (i.e. each bucket covers 10 time steps in the pricing scheme) and compute the sensitivities to each of these buckets. This is to get a realistic test setup with a lot of sensitivities to compute, which is the main use case of the reverse mode of AD tools. Although we compute the bucketed vega we report only the sum of these bucket vegas in the results below.

For the forward mode tests we compute only the total vega (i.e. only one forward sweep is executed where the dot product of the partial derivatives and $(1, \dots, 1)$ is computed).

3. Test enviroment

The test enviroment's main characteristics are:

(1) CPU: Intel(R) Core(TM) i7-2760QM CPU @ 2.40GHz, single threaded,
(2) OS: Ubuntu 14.04
(3) Compiler: gcc (g++, gfortran) 4.9.2, optimization O3
(4) AD tools: ADOL-C-2.5.2, CppAD git repository 02a00a, Open AD/F svn revision 247

4. Test results

Table 1 shows the results for the price and vega produced in QuantLib (reference) and in the C++ and Fortran code that was written from scratch. The pricing is close and there are differences in the pde solver that might explain the difference, see above, so that we conclude that the code written from scratch is working correctly.

Table 1. Pricing results Quantlib, from scratch C++, Fortran

Code	NPV	Total Vega
QuantLib (pricing reference)	18.4072	na
from scratch C++ / CppAD	18.3746	126.0181
from scrach C++ / ADOL-C	18.3746	126.0181
Fortran / OpenAD/F	18.3746	126.0181

Table 2 shows the computation time for a single pricing and with AD enabled in both forward and reverse mode. The results are estimated using 50 outer loops over the test programs.

Table 2. Computation times (loop over 50 pde solutions) in milliseconds

Test	QuantLib	CppAD	ADOL-C	OpenAD/F
Plain computation	37.4	7.0		7.1
Forward	na	575.5	528.3	15.6
Reverse	na	540.9	544.8	31.3

While table 2 suggests that OpenAD/F works close to the theoretical optimum speed (factor 4 in computation time for reverse mode), we observe that this optimum is only reached when a bigger number of outer loops is executed, while for one or only a few executions the computation time is much higer, see table 3. To be clarified.

Table 3. Computation time OpenAD/F averaged over different loop sizes in milliseconds

Loop size	OpenAD/F Reverse
1	216.0
2	125.0
3	86.7
4	76.5
5	66.6
10	47.9
50	31.3
100	30.1
500	28.7
1000	28.4
5000	28.3

5. Tape size in OpenAD/F

Some code changes (like changing the computation of `d1` to the usual one, which is commented out in the source, or like changing the number of time steps `sizet` from 5000 to 7000 produces runtime exceptions. This is due to hard coded maximum tape sizes defined in `OAD_tape.f90` (cf. Krishna's email 15-Jul-2015).

References

[1] https://projects.coin-or.org/CPPAD
[2] https://projects.coin-or.org/ADOL-C
[3] http://www.mcs.anl.gov/OpenAD
[4] QuantLib A free/open-source library for quantitative finance, http://www.quantlib.org

Appendix A. Source Code

In this section we list the full source code used in this paper. Since the OpenAD/F framework requires non-standard build steps we list the used make files as well.

A.1. **CppAD and ADOL-C C++ test code.** File `testcpp.cpp`

```
// define to enable AD
//#define CPPAD

// define to enable ADOL-C
#define ADOLC

// define for plain computation
// #define PLAIN

#include <iostream>
#include <vector>
#include <cmath>

#ifdef CPPAD
#include <cppad/cppad.hpp>
using CppAD::AD;
```

```
typedef AD<double> dbl;
#endif

#ifdef ADOLC
#include <adolc/adolc.h>
typedef adouble dbl;
#endif

#ifdef PLAIN
typedef double dbl;
#endif

// problem data
const dbl S0 = std::log(100.0);
const double T = 10.0;
const dbl K = std::log(120.0);
const unsigned int n = 500; // sigma grid

// PDE parameters
const dbl Smin = 1.07555172964;
const dbl Smax = 8.13478864234;
const unsigned int sizeS = 2 * 50 + 1;
const unsigned int sizeT = 500 * 10;

// solution grid
dbl loc[sizeS + 1], c[2][sizeS], exerciseValue[sizeS];

int main() {

    for (unsigned int testrun = 0; testrun < 1; ++testrun) {

#ifdef PLAIN
        std::vector<dbl> implVol(n, 0.20);
#endif

#ifdef CPPAD
        std::vector<dbl> implVol(n, 0.20);
        CppAD::Independent(implVol);
#endif

#ifdef ADOLC
        int tag = 1, keep = 1;
        adouble implVol[n];
        trace_on(tag, keep);
        for (unsigned int i = 0; i < n; ++i)
            implVol[i] <<= 0.20;
#endif

        unsigned int swap = 0;

        // initial values
        const dbl h = (Smax - Smin) / static_cast<dbl>(sizeS - 1);
        const dbl hq = h * h;
        for (unsigned int j = 0; j < sizeS; ++j) {
            loc[j] = Smin + h * static_cast<double>(j);
            c[swap][j] = exerciseValue[j] =
                std::max<dbl>(exp(loc[j]) - exp(K), 0.0);
        }

        // PDE solver
        const double dt = T / static_cast<double>(sizeT);
        for (unsigned int i = 0; i < sizeT; ++i) {
            // rollback
            for (unsigned int j = 0; j < sizeS; ++j) {
                const dbl v = implVol[static_cast<int>(
                    static_cast<double>(i) * static_cast<double>(n) /
                    static_cast<double>(sizeT))];
                dbl d1, d2;
                if (j == 0 || j == sizeS - 1) {
                    d2 = 0.0;
                } else {
                    d2 = (c[swap][j + 1] - 2.0 * c[swap][j] + c[swap][j - 1]) /
                         hq;
                }
                if (j == 0) {
                    d1 = (c[swap][j + 1] - c[swap][j]) / h;
                } else {
                    if (j == sizeS - 1) {
                        d1 = (c[swap][j] - c[swap][j - 1]) / h;
                    } else {
                        d1 = (c[swap][j + 1] - c[swap][j - 1]) / (2.0 * h);
                    }
                }
                // Euler
                c[1 - swap][j] = c[swap][j] + 0.5 * dt * v * v * (d2 - d1);
            }
            // update prices
            for (unsigned int j = 0; j < sizeS; ++j) {
                c[1 - swap][j] = std::max(c[1 - swap][j], exerciseValue[j]);
            }
            swap = 1 - swap;
        }

        // solution output
       std::clog.precision(12);
       std::clog << "c(0,0) = " << c[swap][(sizeS - 1) / 2] << std::endl;

#ifdef CPPAD
        std::vector<dbl> y(1);
        y[0] = c[swap][(sizeS - 1) / 2];
        CppAD::ADFun<double> f(implVol, y);
        // std::vector<double> w(1, 1.0);
        // std::vector<double> vega(n);
        // vega = f.Reverse(1, w);
        double sum = 0.0;
        // for (unsigned int i = 0; i < n; ++i) {
```

```
        //     sum += vega[i];
        // }
        std::vector<double> x0(n, 1.0);
        sum = f.Forward(1, x0)[0];
        std::clog << "vega =" << sum << std::endl;
#endif

#ifdef ADOLC
        double yout;
        c[swap][(sizeS - 1) / 2] >>= yout;
        trace_off();
        double u[1];
        u[0] = 1.0;
        double vega[n];
        reverse(tag, 1, n, 0, u, vega);
        double sum = 0.0;
        for (unsigned int i = 0; i < n; ++i) {
            sum += vega[i];
        }
        // double x0[n], x1[n], y0[1], y1[1];
        // for (unsigned int i = 0; i < n; ++i) {
        //     x0[i] = 0.2;
        //     x1[i] = 1.0;
        // }
        // fos_forward(tag, 1, n, 0, x0, x1, y0, y1);
        // sum = y1[0];
        std::clog << "vega =" << sum << std::endl;
#endif
    }

} // main
```

A.2. **QuantLib C++ reference code.** File `testql.cpp`:

```
#include <ql/quantlib.hpp>

#include <boost/make_shared.hpp>

using namespace QuantLib;

int main() {

    Date evalDate(13, July, 2015);
    Settings::instance().evaluationDate() = evalDate;
    Date maturity = evalDate + 10 * Years;

    const Real S0 = 100.0;
    const Real K = 120.0;
    const Real vol = 0.20;

    const Size tGrid = 500 * 10;
    const Size xGrid = 2 * 50 + 1;

    const Handle<Quote> S0_q(boost::make_shared<SimpleQuote>(S0));
    const Handle<BlackVolTermStructure> blackVol(
        boost::make_shared<BlackConstantVol>(evalDate, NullCalendar(), vol,
                                             Actual365Fixed()));
    Handle<YieldTermStructure> zeroyts(
        boost::make_shared<FlatForward>(evalDate, 0.0, Actual365Fixed()));

    boost::shared_ptr<StrikedTypePayoff> payoff =
        boost::make_shared<PlainVanillaPayoff>(Option::Call, K);
    boost::shared_ptr<Exercise> exercise =
        boost::make_shared<AmericanExercise>(maturity);
    boost::shared_ptr<VanillaOption> option =
        boost::make_shared<VanillaOption>(payoff, exercise);

    boost::shared_ptr<BlackScholesProcess> p =
        boost::make_shared<BlackScholesProcess>(S0_q, zeroyts, blackVol);

    boost::shared_ptr<FdBlackScholesVanillaEngine> engine =
        boost::make_shared<FdBlackScholesVanillaEngine>(
            p, tGrid, xGrid, 0, FdmSchemeDesc::ExplicitEuler());

    option->setPricingEngine(engine);

    std::clog << "c = " << option->NPV() << std::endl;
}
```

A.3. **OpenAD/F calculation Fortran code.** File testf90.f90:

```
subroutine toy_pde(impliedVol,price)
implicit none

integer, parameter:: n = 500

double precision:: impliedVol(0:n-1), price

double precision, parameter:: s0=4.605170185988092d0, k=4.787491742782046, t=10.0d0
double precision, parameter:: smin=1.07555172964d0, smax=8.13478864234d0
integer, parameter:: sizes = 101, sizet = 5000

double precision:: loc(0:sizes), c(0:1,0:sizes-1), exerciseValue(0:sizes-1)
double precision:: h, hq, dt, d1, d2, v
integer:: swap, i, j, ind

!£openad INDEPENDENT(impliedVol)

swap = 0

! initial values
h = (smax-smin)/dble(sizes-1)
hq = h*h
do j=0,sizeS-1,1
    loc(j) = smin + h*dble(j)
    c(swap,j) = max(dexp(loc(j))-dexp(K),0.0d0)
    exerciseValue(j) = c(swap,j)
end do
```

```
! PDE solver
dt = t / dble(sizeT)
do i=0,sizeT-1,1
    do j=0,sizeS-1,1
       ind = int(dble(n)*dble(i)/dble(sizeT))
       v = impliedVol(ind)
       if(j==0.or.j==sizeS-1) then
          d2=0
          d1=0 ! we do not really want this ...
       else
          d2=(c(swap,j+1)-2.0d0*c(swap,j)+c(swap,j-1))/hq
          d1 = (c(swap,j+1)-c(swap,j-1))/(2.0d0*h)
       endif
       ! ... but the following code does not build with OpenAD:
       ! if(j==0) d1 = (c(swap,j+1)-c(swap,j))/h
       ! if(j==sizeS-1) d1 = (c(swap,j)-c(swap,j-1))/h
       ! if(j>0.and.j<sizeS-1) d1 = (c(swap,j+1)-c(swap,j-1))/(2.0d0*h)
       c(1-swap,j)=c(swap,j) + 0.5 * dt * v*v * (d2-d1)
     end do
     do j=0,sizeS-1,1
        c(1-swap,j)=max(c(1-swap,j),exerciseValue(j))
     end do
     swap = 1 -swap
end do

! return solution
price = c(swap,(sizeS-1)/2)

!£openad DEPENDENT(price)

end subroutine toy_pde
```

A.4. **OpenAD/F plain mode driver Fortran code.** File `driverf90_plain.f90`:

```
program driver
implicit none
external toy_pde

integer,parameter::n=500
integer::i
double precision::impliedVol(0:n-1),price

do i=0,n-1,1
   impliedVol(i) = 0.20d0
end do

call toy_pde(impliedVol, price)

write(*,*) 'c = ', price

end program driver
```

A.5. **OpenAD/F forward mode driver Fortran code.** File `driverf90_forward.f90`:

```
program driver
use OAD_active
implicit none
external toy_pde

integer, parameter:: n = 500

type(active)::impliedVol(0:n-1),price

double precision::vega
integer::i

do i=0,n-1,1
   impliedVol(i)%v=0.20d0
   impliedVol(i)%d=1.0
end do

call toy_pde(impliedVol, price)

vega = price%d

write(*,*) 'c = ', price%v, ' dprice/dvol = ', vega

end program driver
```

A.6. **OpenAD/F reverse mode driver Fortran code.** File `driverf90.f90`:

```
program driver
use OAD_active
use OAD_rev
implicit none
external toy_pde

integer, parameter:: n = 500

type(active)::impliedVol(0:n-1),price

double precision::vega
integer::i

do i=0,n-1,1
   impliedVol(i)%v=0.20d0
end do

price%d=1.0d0
our_rev_mode%tape=.true.

call toy_pde(impliedVol, price)

do i=0,n-1,1
   vega = vega + impliedVol(i)%d
end do

write(*,*) 'c = ', price%v, ' dprice/dvol = ', vega
```

```
end program driver
```

A.7. **OpenAD/F makefile plain mode.** File `Makefile_plain`:

```
ifndef F90C
F90C=gfortran
endif
driverf90_plain: driverf90_plain.o testf90.o
        ${F90C} -O3 -o $@ $^
testf90.f90, driverf90_plain.f90:
%.o : %.f90
        ${F90C} -g -O3 -o $@ -c $<
clean:
        rm -f testf90.o driverf90_plain.o
.PHONY: clean toolChain
```

A.8. **OpenAD/F makefile forward mode.** File `Makefile_forward`:

```
ifndef F90C
F90C=gfortran
endif
RTSUPP=w2f__types OAD_active
driverf90_forward: $(addsuffix .o, $(RTSUPP)) driverf90_forward.o
                                              testf90.pre.xb.x2w.w2f.post.o
        ${F90C} -O3 -o $@ $^
testf90.pre.xb.x2w.w2f.post.f90 $(addsuffix .f90, $(RTSUPP)) : toolChain
toolChain : testf90.f90
        openad -c -m f $<
%.o : %.f90
        ${F90C} -g -O3 -o $@ -c $<
clean:
        rm -f ad_template* OAD_* w2f__* iaddr*
        rm -f testf90.pre* *.B *.xaif *.o *.mod driver driverE *~
.PHONY: clean toolChain
```

A.9. **OpenAD/F makefile reverse mode.** File `Makefile`:

```
ifndef F90C
F90C=gfortran
endif
ifndef CC
CC=gcc
endif
RTSUPP=w2f__types OAD_active OAD_cp OAD_tape OAD_rev
driverf90: $(addsuffix .o, $(RTSUPP)) driverf90.o testf90.pre.xb.x2w.w2f.post.o
        ${F90C} -O3 -o $@ $^
testf90.pre.xb.x2w.w2f.post.f90 $(addsuffix .f90, $(RTSUPP)) iaddr.c : toolChain
toolChain : testf90.f90
        openad -c -m rj $<
%.o : %.f90
        ${F90C} -g -O3 -o $@ -c $<
%.o : %.c
        ${CC} -g -O3 -o $@ -c $<
clean:
        rm -f ad_template* OAD_* w2f__* iaddr*
        rm -f testf90.pre* *.B *.xaif *.o *.mod driver driverE *~
.PHONY: clean toolChain
```

E-mail address, P. Caspers: `pcaspers1973@googlemail.com`