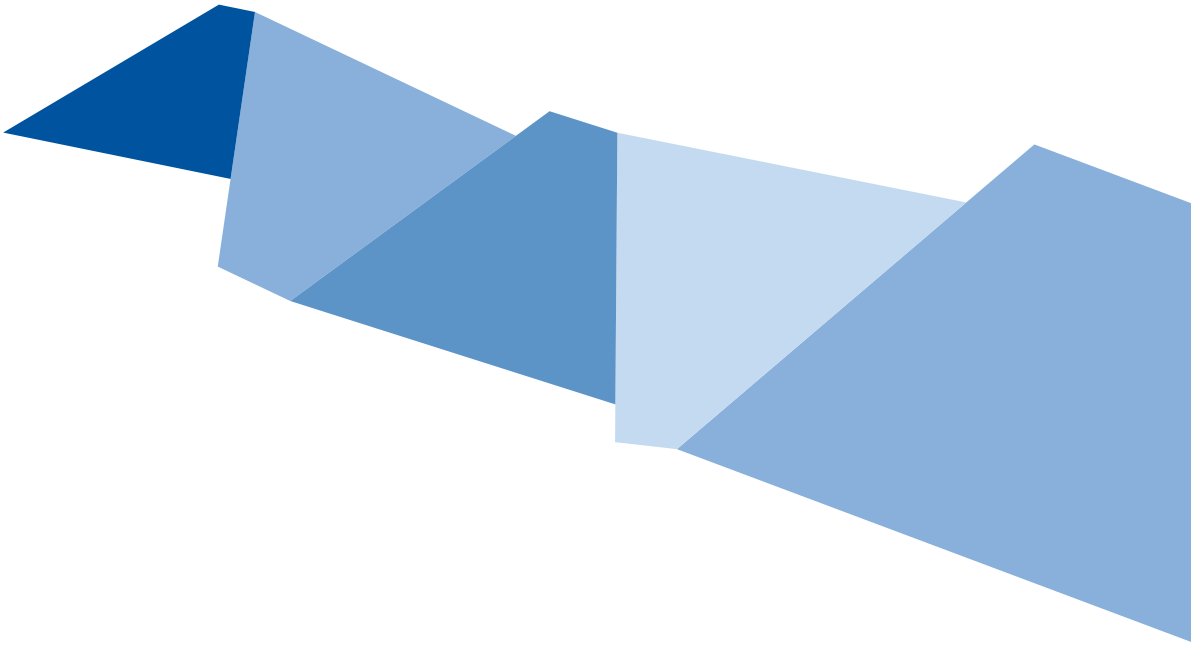


Fast Estimates of Greeks from American Options: A Case Study in Adjoint Algorithmic Differentiation

Jens Deussen
Viktor Mosenkis
Uwe Naumann

Department of Computer Science

Technical Report

Aachener Informatik-Berichte (AIB) | ISSN 0935-3232 | AIB-2018-02
RWTH Aachen University | Department of Computer Science | January 2018

The publications of the Department of Computer Science of *RWTH Aachen University* are in general accessible through the World Wide Web.

`http://aib.informatik.rwth-aachen.de/`

Fast Estimates of Greeks from American Options: A Case Study in Adjoint Algorithmic Differentiation

Jens Deussen*, Viktor Mosenkis and Uwe Naumann

LuFG Informatik 12, RWTH Aachen University
Software and Tools for Computational Engineering
52074 Aachen, Germany

Abstract

In this article algorithmic differentiation is applied to compute the sensitivities of the price of an American option, which is computed by the Longstaff-Schwartz algorithm. Adjoint algorithmic differentiation methods speed up the calculation and make the results more accurate and robust compared to a finite difference approximation. However, adjoint computations require more memory due to the storing of intermediate results. One possibility to reduce these memory requirements is to use a technique called checkpointing: Instead of storing all intermediate results the required values are recomputed. Another possibility is to apply a pathwise adjoint approach, which results from an analysis of the Longstaff-Schwartz algorithm and the exploitation of its features. The presented approach is embarrassing parallel and yields the same results as the other adjoint methods, but it reduces the computational effort as well as the memory requirements of the computation to the level of a single pricing calculation.

1. Introduction and Summary of the Results

There are several approaches to calculating the price of American-style options. Besides finite difference and binomial tree methods, simulations can be used to estimate option prices. This article focuses on the least-squares approach for American option pricing by Longstaff and Schwartz (Longstaff and Schwartz, 2001). Some further research in the area of American option pricing using simulation includes (Tilley, 1993; Carriere, 1996; Broadie et al., 1997; Rogers, 2002; Glasserman, 2003). We chose a representative case study in order to illustrate many features of problems considered in practice.

A naive ansatz for the estimation of the sensitivities of an American option is to use numerical differentiation with a finite difference approximation (FD). One disadvantage of this approach is its computational cost which grows with the number of input parameters. For each first-order sensitivity an additional function evaluation is required at least. The calculation of higher derivatives is even more expensive and the results are often inaccurate.

The preferred numerical method to derive the sensitivities of a computer program is Algorithmic Differentiation (AD) (Griewank and Walther, 2008; Naumann, 2012). This technique returns mathematically exact derivatives with machine accuracy up to an arbitrary order by exploiting elemental symbolic differentiation rules and the chain rule. AD distinguishes between two basic modes: the forward mode and the reverse mode. The forward mode builds a directional derivative (also: tangent) code for the computation of the sensitivities at a cost proportional to the number of input parameters. Similarly, the reverse mode builds an adjoint version of the program but it can compute the Jacobian at a cost that is proportional to the number of outputs. For that reason, the adjoint method is advantageous for problems with

*email: deussen@stce.rwth-aachen.de

a small set of output parameters, as it occurs frequently in many fields of Computational Science and Engineering, for example in fluid dynamics (Giles and Pierce, 2000) or optimization (Nikolova et al., 2004). Furthermore, adjoint methods should be suitable for an efficient calculation of the sensitivities in option pricing, due to the fact, that the algorithms have a lot of input variables but typically just a single output (the option price). The fact that AD can also be used for problems in finance has been shown repeatedly e.g. (Giles and Glasserman, 2006; Leclerc et al., 2009; Capriotti and Giles, 2010; Capriotti, 2011; Henrard, 2013). The adjoint computation comes along with potential high memory requirements due to the storage of the values of all variables. To reduce this load, required values can be recomputed instead of stored by using checkpointing schemes.

We developed another algorithm similar to the algorithm from (Leclerc et al., 2009) by exploiting some properties of the Longstaff-Schwartz algorithm and using a simple assumption, thus the memory requirements can be diminished further. This could be called a pathwise adjoint approach, in which the time and the path loops are interchanged such that the path loop is the outer loop and the time loop is the inner loop. Moreover, parallelization is enabled for this approach.

The aim of this paper is to show that AD is applicable to American option pricing by simulation, using the Longstaff-Schwartz algorithm as an example. Furthermore, finite differences, tangent and adjoint results are compared to each other with respect to time and memory requirements. We show that the run time as well as the memory requirements of the calculation of the Greeks can be reduced almost to the level of the pricer by using the parallel pathwise adjoint approach.

The paper is organized as follows: In section 2 there is a brief introduction to the problem, the test cases and the methods which are used in this paper, AD, checkpointing and the pathwise adjoint approach. Subsequently in section 3 the results are presented. First, the AD methods and the FD approach are compared to each other in terms of run time. Then, the memory requirements are reduced by using a checkpointing scheme or by computing the adjoints pathwise. The last section gives a conclusion and an outlook.

2. Problem Description and Methodology

For the computation of the sensitivities we applied a numerical differentiation approach with FD and diverse AD techniques. The finite differences are computed by using a central FD scheme of order one for first-order sensitivities and of order two for second-order sensitivities. The AD results are computed with the AD tool *dco/c++* (Lotz et al., 2011), which computes the derivatives of a function by operator and function overloading in C++. The employed AD methods are the tangent mode, the adjoint mode as well as a checkpointed version of the adjoint computation. For the checkpointing we picked a scheme with a recomputation of the cycles of the time step loop. Thus, only the values of one loop iteration need to be stored at once. The scheme is kept constant, because an analysis of this part is beyond the scope of this article. Furthermore, a pathwise computation of the adjoints is used by exploiting some properties of the Longstaff-Schwartz algorithm (LSA) (Longstaff and Schwartz, 2001).

2.1. American Option Pricing with the Longstaff-Schwartz Algorithm

For the comparison between the different methods, an American put option on a single asset is considered. The LSA is used to compute the American option prices, at which the paths are assumed to evolve as a geometric Brownian motion. Therefore, the stochastic differential equation of the Black-Scholes model (Black and Scholes, 1973) is applied to generate the stock prices. This model is chosen due to its easy comprehensibility and genericity of its implementation. Furthermore, the implementation is modular, such that this model can be

replaced by another one without great effort. The price of the underlying asset S_t under the risk neutral measure and without a dividend yield satisfies

$$\mathrm{d}S_t = rS_t\mathrm{d}t + \sigma S_t \mathrm{d}W_t \quad ,$$

where r is the risk free interest rate, W_t is a standard Brownian motion and σ denotes the volatility. The stock prices are then calculated as

$$S_t = S_0 \exp\left(\left(r - 0.5\sigma^2\right) t\mathrm{d}t + \sigma \sum_{i=1}^{t} Z_i\right) \quad .$$

Pseudo-random numbers are generated with a constant seed of the random number generator *rand()* from the C random library to make the results comparable. To obtain the required standard normal random numbers Z the pseudo-random numbers are converted with a Box-Muller transformation (Box and Muller, 1958).

2.2. Algorithmic Differentiation

AD is a technique which transforms a *primal function* or *primal code* by using the chain rule to compute additionally to the function value the derivative of that function with respect to a given set of input and intermediate variables.

Inputs of a Monte Carlo simulation for American option pricing are for example the initial stock price S_0, the volatility σ, the random numbers Z and the number of simulated paths N_P, whereas output variables are for example the calculated option price V and the cash flow matrix. For the determination of the option Greeks not all of the derivatives of the outputs with respect to the inputs are needed. There are just derivatives of the option price with respect to a specified set of the inputs required. The option price and this set of inputs are then called *active*, while the elements of the cash flow matrix or the random numbers are *passive*. A tilde is used to denote *passive* variables.

We consider that the given *primal code* has $n+\tilde{n}$ inputs and $m+\tilde{m}$ outputs. Furthermore, for simplicity it is assumed that the implementation only contains the calls of the subroutines f and g. Thus, the implementation represents the function dependency between the set of inputs x and the set of outputs y

$$x \quad \xrightarrow{f} \quad v \quad \xrightarrow{g} \quad y \quad ,$$

in which v denotes the set of intermediate variables of the computation. Then, the composition of the two functions f and g leads to the multivariate vector function

$$F : \mathbb{R}^{n+\tilde{n}} \to \mathbb{R}^{m+\tilde{m}}, \quad (y, \tilde{y}) = F(x, \tilde{x}) = (g \circ f)(x, \tilde{x}) \quad .$$

The local differentiability of the vector function F and its corresponding implementation up to the required order is necessary to compute the Jacobian

$$\mathrm{J}_F(x, \tilde{x}) = \left(\frac{\partial y_j}{\partial x_i}\right)_{i=0,\ldots,n-1}^{j=0,\ldots,m-1} \in \mathbb{R}^{m \times n} \quad .$$

Tangent Mode

The tangent model can simply derived by differentiating the function dependence, using the notation from (Naumann, 2012).

$$x^{(1)} \quad \xrightarrow{f^{(1)}} \quad v^{(1)} \quad \xrightarrow{g^{(1)}} \quad y^{(1)}$$

The superscript $^{(1)}$ stands for the first directional derivative of the variable, also called tangent. This leads to the functions

$$v^{(1)} = \frac{\partial v}{\partial x} x^{(1)} \ ,$$
$$y^{(1)} = \frac{\partial y}{\partial v} v^{(1)} \ ,$$

and hence

$$y^{(1)} = \frac{\partial y}{\partial v} \frac{\partial v}{\partial x} x^{(1)} \ .$$

This approach is also called the *forward* mode and can be interpreted as a linear mapping with the Jacobian $\mathsf{J}_F \in \mathbb{R}^{m \times n}$

$$y^{(1)} = \mathsf{J}_F(x) \cdot x^{(1)} \ . \tag{1}$$

For each evaluation of (1) with $x^{(1)}$ set to the i-th Cartesian basis vector in $\mathbb{R}^n$, one gets the i-th column of the Jacobian. To get all entries of the Jacobian by using this model, n evaluations are required, for each *active* input respectively. Thus, the costs of the derivation of the Jacobian by using a tangent model is proportional to the number of *active* input variables. Note that the costs of this method are similar to the FD costs but this method is more accurate.

Adjoint Mode

Instead of deriving the first derivatives with the *forward* mode, the elements of the Jacobian can be computed by using the adjoints of the particular variables. This mode is also called *reverse* mode, due to the reverse computation of the adjoints compared to the computation of the values. Therefore, a data-flow reversal of the program is required, such that additional information on the computation (e.g. partial derivatives) needs to be stored (see (Hascoët et al., 2005)), which potentially leads to high memory requirements. The data structure to store this additional information is often called *tape*.

Again following the notation of (Naumann, 2012), the first-order adjoints are denoted with a subscript $_{(1)}$ and they are defined as

$$x_{(1)} = \left(\frac{\partial v}{\partial x} \right)^\top v_{(1)} \quad ,$$
$$v_{(1)} = \left(\frac{\partial y}{\partial v} \right)^\top y_{(1)} \quad ,$$

which yields

$$x_{(1)} = \left(\frac{\partial v}{\partial x} \right)^\top \left(\frac{\partial y}{\partial v} \right)^\top y_{(1)} \quad .$$

The dependency of this model is visualized in Figure 1 in which the subroutines are represented by squares. A rightwarded arrow at the bottom of a square indicates a forward execution of the subroutine as well as the storage of additional information on the tape. Analogously a leftwarded arrow denotes an execution in reverse order and an interpretation of the tape. The order of execution is depth-first and from left to right.

Similar to the *forward* mode, the *reverse* mode describes a mapping with the transpose of the Jacobian

$$x_{(1)} = \mathsf{J}_F(x)^\top \cdot y_{(1)} \quad .$$

The vector $y_{(1)}$ is consecutively set equal to each of the Cartesian basis vectors in $\mathbb{R}^m$. After running the *reverse* code the vector $x_{(1)}$ contains one row of the Jacobian. The costs of this mode are then of order m, which is the number of *active* outputs.

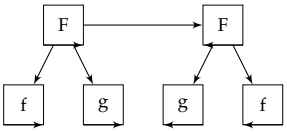


Figure 1: *Call tree of adjoint version with the dummy function* F

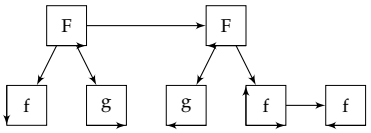


Figure 2: *Call tree of adjoint version with checkpointing*

Higher Derivatives To get second derivatives, the AD modes can be applied to an already differentiated program. It is incidental that four combinations are possible. While the complexity of the *forward-over-forward* mode is equal to that of the FD approach, respectively of order n^2, the other three combinations lead to a lower computational cost under the assumption that $m < n$. For example the *forward-over-reverse* mode is generated by applying the forward mode to a first-order adjoint code. This mode can compute the second derivatives at a cost of order $m \cdot n$. This procedure can be repeated for derivatives of arbitrary order.

Checkpointing

Checkpointing is a technique to reduce the memory requirements of the adjoint mode. Instead of storing the complete *forward* section on the tape, some of the values can be recomputed. In the following there is a minimal example for checkpointing with the above given function dependency.

Rather than taping f during its *forward* computation only its input arguments are recorded as a checkpoint. This is visualized by a downward pointing arrow on the left side of the square. Then, g is executed as before with storing the additional information on the tape, followed by its *reverse* computation. For the reverse section of f the additional information that was not stored is required, such that the checkpoint is restored and f is executed again. This time the execution is taped to make the additional information available. The re-storage is denoted by an arrow on the left side pointing to the top. Figure 2 represents this example as a call tree.

Checkpointing can also be applied recursively. The recomputation of the values is always feasible to save some memory at a computational cost, but since the computation should be as fast as possible, a trade-off between memory requirement and computational time is sought. Further information about checkpointing are given in (Volin and Ostrovskii, 1985; Griewank and Walther, 2000; Naumann, 2012).

Code Analysis

In the LSA (Algorithm 1) there is a decision for each path whether to exercise the option at that time or to wait and expect a higher cash flow in the future. Therefore, a predicted exercise boundary is computed by a regression of the future cash flows on the current stock prices. This exercise boundary is compared to the current cash flows of each path. A computational graph of this algorithm is given in Figure 3(a) in which the data dependency is represented with arrows. Due to the fact that the comparison behaves like a Heaviside step function, which is not differentiable at the step, the AD methods cannot catch the influence of the regression to the output.

Algorithm 1 Longstaff-Schwartz algorithm

In:

→ initial stock price S_0, strike price K, time to maturity T, volatility σ, risk-free interest rate r

→ number of paths N_P, number of time steps N_T

→ accumulated random numbers $Z_{p,t} = z + Z_{p,t-1}$ with random numbers z

→ function generating the stock price for a given time t and path p
$S_p = h(S_0, T, \sigma, r, t, N_T, Z_{p,t})$

→ function computing the exercise boundary b with a regression of the in the money paths I, their stock prices and current cash flows
$b = R(I, (S_i), (v_i))$

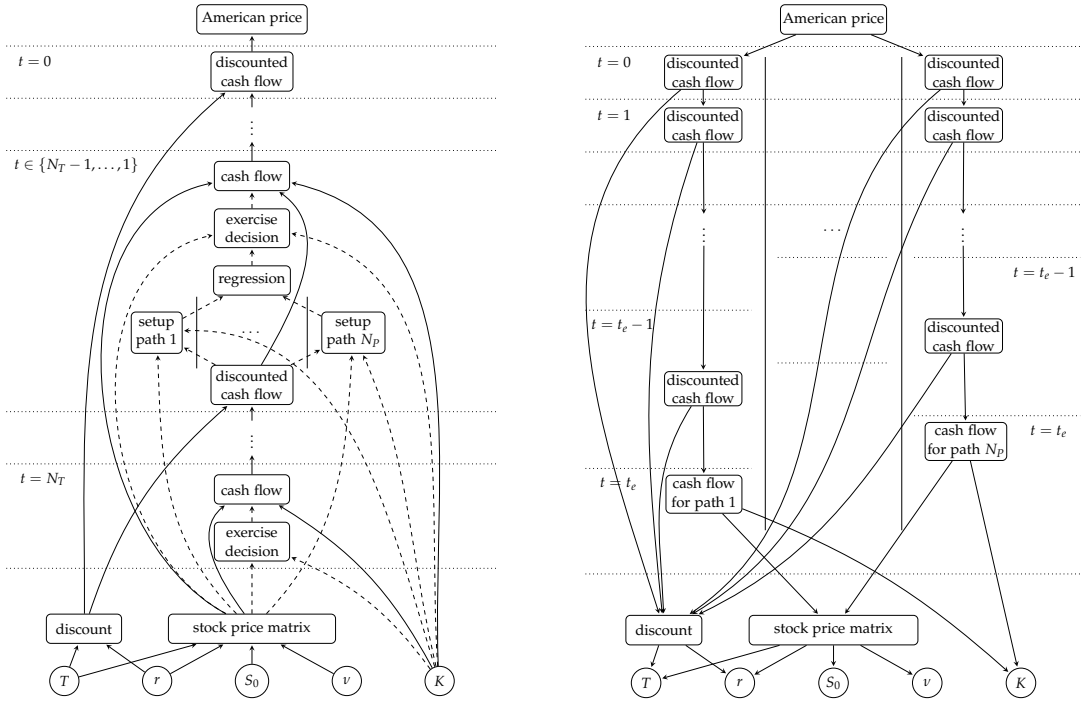
Out:

← option price: $V \in \mathbb{R}$

← exercise time for each path: $(\tilde{t}_p) \in \mathbb{N}^{N_P}$

Algorithm:

for $p = 1 \to N_P$ **do** ▷ Initialization of cash flow and exercise times for $t = N_T$
  $S_p = h(S_0, T, \sigma, r, N_T, N_T, Z_{p,N_T})$
  if $S_p < K$ **then**
    $v_p \leftarrow K - S_p$
    $\tilde{t}_p \leftarrow N_T$
  else
    $v_p \leftarrow 0$
  end if
end for
for $t = N_T - 1 \to 1$ **do** ▷ Time step loop
  $I \leftarrow \{\}$ ▷ Set for indices of in-the-money paths
  for $p = 1 \to N_P$ **do** ▷ Path loop
    $v_p \leftarrow v_p \cdot \exp(-r \cdot T/N_T)$
    $S_p = h(S_0, T, \sigma, r, t, N_T, Z_{p,t})$
    if $S_p < K$ **then**
      $I \leftarrow I \cup \{p\}$;
    end if
  end for
  $b = R(I, (S_i), (v_i))$ ▷ Computation of the exercise boundary
  for all $p \in I$ **do**
    if $S_p < b$ **then** ▷ Exercise decision
      $v_p \leftarrow K - S_p$
      $\tilde{t}_p \leftarrow t$
    end if
  end for
end for
$V \leftarrow 0$
for $p = 1 \to N_P$ **do**
  $V \leftarrow V + v_p$
end for
$V \leftarrow V \cdot \exp(-r \cdot T/N_T)\,/N_P$



(a) Basic Longstaff-Schwartz algorithm (b) Pathwise adjoint approach

Figure 3: *Computational graphs*

The LSA computes the option price according to

$$V = \frac{1}{N_P} \sum_p \left[K \exp\left(-r \frac{T}{N_T} t_p \right) - S_0 \exp\left(-0.5\sigma^2 \frac{T}{N_T} t_p + \sigma \cdot Z_p \right) \right]. \tag{2}$$

The exercise boundary t_p actually depends on the active input parameters. But in the code there is no assignment from an active parameter to t_p, such that t_p is an independent variable from the viewpoint of AD. Hence, differentiating (2) with respect to the initial stock price and the volatility, we get

$$\frac{\partial V}{\partial S_0} = \frac{1}{N_P} \sum_p \left[-\exp\left(-0.5\sigma^2 t_p \frac{T}{N_T} + \sigma \cdot Z_p \right) \right] \quad , \tag{3}$$

$$\frac{\partial V}{\partial \sigma} = \frac{1}{N_P} \sum_p \left[-S_0 \cdot \left(-\sigma t_p \frac{T}{N_T} + Z \right) \cdot \exp\left(-0.5\sigma^2 t_p \frac{T}{N_T} + \sigma \cdot Z_p \right) \right]. \tag{4}$$

From (3) and (4) we can see that some second-order sensitivities are computed to be zero by using AD methods, e.g. (3) differentiated with respect to the initial stock price.

The affected edges are dashed in the computational graph. The missing control-flow dependency could lead to problems in cases where the current cash flow is close to the exercise boundary.

Pathwise Adjoint Approach

One possibility of handling the discontinuity is to assume that the number of cases in which the stock price is close to the exercise boundary is negligible compared the number of simulated paths. Building the average over all paths should still allow the computation of the sensitivities.

Algorithm 2 Pathwise adjoint Longstaff-Schwartz algorithm

In:

→ active pricing parameters with an initial stock price $S_0 \in \mathbb{R}$, strike price $K \in \mathbb{R}$, time to maturity $T \in \mathbb{R}$, volatility $\sigma \in \mathbb{R}$, risk-free interest rate $r \in \mathbb{R}$

→ number of paths $N_P \in \mathbb{N}$, number of time steps $N_T \in \mathbb{N}$

→ accumulated random numbers $(Z_{p,t}) \in \mathbb{R}^{N_P \times N_T}$

→ implementation of the LSA for a set of active pricing parameters, a number of paths, a number of time steps and accumulated random numbers for computing the option price $V \in \mathbb{R}$ and the exercise times for each path $(\tilde{t}_p) \in \mathbb{N}^{N_P}$:
$f : \mathbb{R}^5 \times \mathbb{N}^2 \times \mathbb{R}^{N_P \times N_T} \to \mathbb{R} \times \mathbb{N}^{N_P},\ (V, (\tilde{t}_p)) = f(S_0, K, T, \sigma, r, N_P, N_T, (Z_{p,t}))$

→ function generating the stock price for a given time and path:
$h : \mathbb{R}^4 \times \mathbb{N}^2 \times \mathbb{R} \to \mathbb{R},\ S_p = h(S_0, T, \sigma, r, t, N_T, Z_{p,t})$

Out:

← option price: $V \in \mathbb{R}$

← gradient of option price w.r.t. active pricing parameters: $\mathbf{g} = \nabla_{(S_0,K,T,\sigma,r)} V$

Algorithm:

1: $(V, (\tilde{t}_p)) = f(S_0, K, T, \sigma, r, N_P, N_T, Z)$
2: $V \leftarrow 0$
3: $\mathbf{g} \leftarrow 0$
4: **for** $p = 1 \to N_P$ **do**
5: setup tape
6: $S_p = h(S_0, T, \sigma, r, \tilde{t}_p, N_T, Z_{p,\tilde{t}_p})$
7: $v \leftarrow (K - S_p) \cdot \exp\left(-r\tilde{t}_p T / N_T\right)$
8: $v_{(1)} \leftarrow 1$
9: interpret tape
10: $V \leftarrow V + v$
11: $\mathbf{g} \leftarrow \mathbf{g} + (S_{0,(1)}, K_{(1)}, T_{(1)}, \sigma_{(1)}, r_{(1)})^\top$
12: **end for**
13: $V \leftarrow V / N_P$
14: $\mathbf{g} \leftarrow \mathbf{g} / N_P$

It follows that the adjoints of the exercise boundary b are zero and therefore the functional dependency of the comparison on the option price (Algorithm 1 line 21) is negligible. Cutting this dependency of the exercise time on the option price should be legitimate, due to the assumption that the exercise time computed by the LSA maximizes the option price (see (Piterbarg, 2003)). This will cause the adjoints of the regression to be zero as well and it is no longer necessary to store these adjoints.

Algorithm 2 is developed to derive the sensitivities of the option with adjoint AD. A similar algorithm was already mentioned in (Leclerc et al., 2009). It is assumed that the exercise times are already computed and therefore it is not necessary to rerun the regression. Furthermore, the time and the path loop are interchanged such that each path can be computed separately. This has the advantage that the memory requirement can be diminished by computing local sensitivities for each path and average them to get the Greeks.

The average of the adjoints of a path in line 14 of Algorithm 2 leads to the Greeks because of

the sum rule in differentiation and due to the fact that the number of paths N_P is independent of the active variables. This relation is given in

$$\frac{\mathrm{d}\frac{\sum v_p(x)}{N_P}}{\mathrm{d}x} = \frac{\sum \frac{\mathrm{d}v_p(x)}{\mathrm{d}x}}{N_P} \quad .$$

The pathwise adjoint algorithm is also visualized as a computational graph in Figure 3(b). It can be seen that this approach is embarrassing parallelized, due to the fact that the values v_p of the paths are independent of each other.

3. Results

This section is organized as follows: First, the blackbox AD methods are compared to the FD approach in terms of accuracy, run time and memory requirements. Blackbox AD denotes the usage of the AD methods without exploiting any structure of the code. In subsection 3.2 the high memory requirement of the blackbox adjoint method is discussed.

The computation of the option price takes following inputs: $S_0 = 1$, $K = 1$, $T = 1$, $\sigma = 0.2$, $r = 0.04$ and $\alpha = 0.005$. The test cases contains the computation of the option price, five first-order sensitivities, respectively delta, vega, theta, rho and dual delta.

The timings and memory requirements of the test applications are generated using an architecture with two multi-core processors of type Intel®Xeon®Processor E5-2630 with a clock rate of 2.30 GHz. Each processor is built up of six CPUs and has access to a random-access memory of 64 GB.

3.1. Blackbox Algorithmic Differentiation

Although the FD approach and AD methods compute the same price for the American option there are differences in the values of the sensitivities. In general the FD results are more volatile than the results computed with the AD methods.

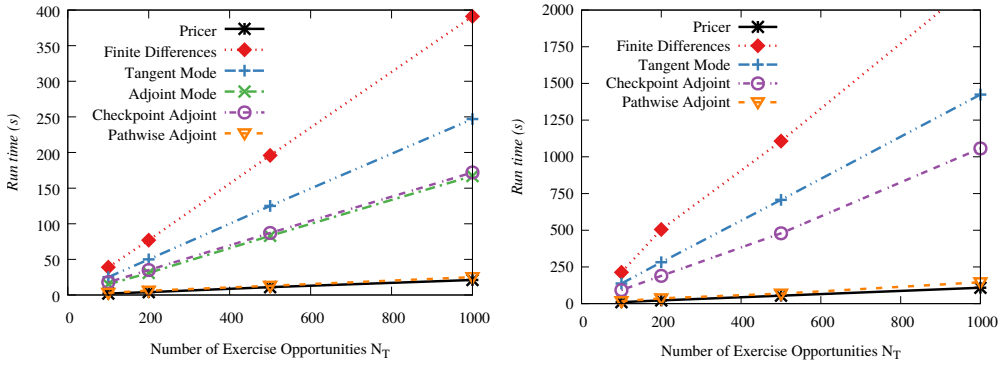
In (Geske and Johnson, 1984) an analytical reference value is given for delta, which is $\Delta_{\text{ref}} = -0.416$. We are not aware of reference values for the other sensitivities. In 50% of the test cases the FD approach leads to inaccurate values for the sensitivities with respect to S_0 and K and there are outlier with an absolute error up to 455. On the other hand, the computation of these sensitivities by using AD is accurate for all test cases. The results for the computation of the sensitivities of the basic LSA are identical for the different AD methods. All other first-order sensitivities are similar for FD and AD.

The timings and the memory requirements of the pricer and of the diverse methods for the sensitivity computation are visualized in Figure 4 and Figure 5, respectively. The results are strongly dependent on the specified test cases and it can be observed that the run time scales linearly by increasing the problem size, the number of paths and the number of exercise opportunities respectively. The blackbox adjoint method is more than twice as fast as the computation with FD, while the run times of the tangent mode computation is located between those two methods.

The computation with the blackbox adjoint method has high memory requirements such that the test cases with 500000 paths are omitted for this method. The other two methods need approximately the same amount of memory which is required for one evaluation.

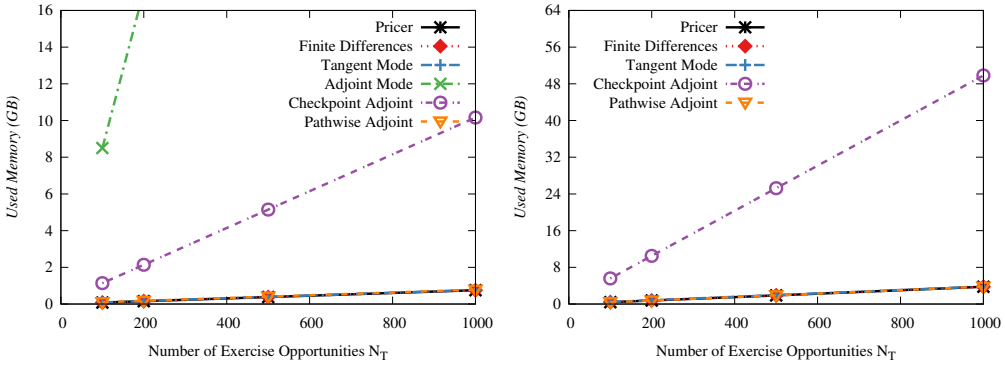
3.2. Reduction of Memory Requirements

To reduce the required memory of the blackbox adjoint method a checkpointing of the time-loop is applied, in which each cycle of this loop is recomputed. Another approach is to use the pathwise adjoint method for the computation. The checkpointed and the pathwise adjoint



(a) $N_P = 100000$ (b) $N_P = 500000$

Figure 4: *Visualization of the timings of the proposed methods*



(a) $N_P = 100000$ (b) $N_P = 500000$

Figure 5: *Visualization of the required amount of memory of the proposed methods*

method produce the same values as the other AD methods for the option price and for the sensitivities.

Again, the timings as well as the memory requirements are visualized in Figure 4 and Figure 5. Due to the recomputation of some values the adjoint method with a checkpointing scheme has a slightly higher computational effort than the blackbox adjoint method. The pathwise adjoint method is faster than the other methods. It has run times that are only 20% higher than the costs of the pricer for the specified test cases. Using parallelization can reduce the run times further such that the computational cost of the parallel approach is only 5% higher compared to the pricer.

By using a checkpointing scheme the high amount of memory which is used by the blackbox adjoint method can be decreased. Moreover, this enables the computation of the test cases with 500000 paths. The pathwise adjoint approach allows to compute the sensitivities using almost the original amount of memory.

4. Conclusion and Outlook

The goal of this paper was the computation of sensitivities of an American option, which is priced by the least squares approach of Longstaff and Schwartz.

The computation of the sensitivities was improved by using AD methods. The AD methods led to more accurate values and the computations were faster than the FD approach. The calculation of the adjoint mode with a checkpointing scheme made this method applicable to the used architecture by decreasing the memory requirements. The run time as well as the memory requirements of the computation were reduced almost to the level of the pricer by using the (parallel) pathwise approach. The value of delta was an accurate approximation of the analytical reference value. Because there were no analytical reference values for the other Greeks a check of their correctness was not possible.

This work can be extended in several directions. Alternative models and stochastic differential equations for the stock price evolution can be applied, for example with a local volatility. Furthermore, an optimal checkpointing scheme should be implemented by using the algorithm from (Griewank and Walther, 2000).

Moreover, the approach can be advanced by saving the exercise boundary, instead of the exercise times for each path. Then, the pathwise adjoint algorithm could check if the boundary is accurate, by evaluating the Greeks and the option price with a new set of random numbers and a new stock price matrix. After that, a comparison of the prices with two different sets of random numbers is possible. This approach was already mentioned by citepGarcia03.

At the end, the correctness of the pathwise approach for sensitivities should be checked and other test cases should be considered with an underlying of multiple assets.

References

Black, F. and Scholes, M. (1973). The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654.

Box, G. E. P. and Muller, M. E. (1958). A note on the generation of random normal deviates. *Ann. Math. Statist.*, 29(2):610–611.

Broadie, M., Glasserman, P., and Jain, G. (1997). Enhanced Monte Carlo estimates for American option prices. *The Journal of Derivatives*, 5(1):25–44.

Capriotti, L. (2011). Fast Greeks by algorithmic differentiation. *SSRN Electronic Journal*.

Capriotti, L. and Giles, M. B. (2010). Fast correlation Greeks by adjoint algorithmic differentiation. *SSRN Electronic Journal*.

Carriere, J. F. (1996). Valuation of the early-exercise price for options using simulations and nonparametric regression. *Insurance: Mathematics and Economics*, 19(1):19–30.

Geske, R. and Johnson, H. E. (1984). The American put option valued analytically. *The Journal of Finance*, 39(5):1511–1524.

Giles, M. and Glasserman, P. (2006). Smoking adjoints: Fast Monte Carlo Greeks. *Risk*, 19(1):88–92.

Giles, M. B. and Pierce, N. A. (2000). An introduction to the adjoint approach to design. *Flow, turbulence and combustion*, 65(3-4):393–415.

Glasserman, P. (2003). *Monte Carlo Methods in Financial Engineering*. Springer New York.

Griewank, A. and Walther, A. (2000). Algorithm 799: Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software*, 26(1):19–45.

Griewank, A. and Walther, A. (2008). *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial & Applied Mathematics (SIAM).

Hascoët, L., Naumann, U., and Pascual, V. (2005). "To Be Recorded" Analysis in Reverse-Mode Automatic Differentiation. *Future Generation Computer Systems*, 21(8):1401–1417.

Henrard, M. (2013). Calibration in finance: Very fast Greeks through algorithmic differentiation and implicit function. *Procedia Computer Science*, 18:1145–1154.

Leclerc, M., Liang, Q., and Schneider, I. (2009). Fast Monte Carlo Bermudan Greeks. *Risk*, 22(7):84.

Longstaff, F. A. and Schwartz, E. S. (2001). Valuing American options by simulation: A simple least-squares approach. *Rev. Financ. Stud.*, 14(1):113–147.

Lotz, J., Leppkes, K., and Naumann, U. (2011). dco/c++ - Derivative Code by Overloading in C++. Technical Report AIB-2011-06.

Naumann, U. (2012). *The Art of Differentiating Computer Programs: An Introduction to Algorithmic Differentiation*, volume 24. SIAM.

Nikolova, N., Safian, R., Soliman, E., Bakr, M., and Bandler, J. (2004). Accelerated gradient based optimization using adjoint sensitivities. *IEEE Transactions on Antennas and Propagation*, 52(8):2147–2157.

Piterbarg, V. (2003). Computing deltas of callable libor exotics in forward libor models. *SSRN Electronic Journal*.

Rogers, L. C. G. (2002). Monte Carlo valuation of American options. *Mathematical Finance*, 12(3):271–286.

Tilley, J. A. (1993). Valuing American options in a path simulation model. *Transactions of the Society of Actuaries*, 45(83):104.

Volin, Y. and Ostrovskii, G. (1985). Automatic computation of derivatives with the use of the multilevel differentiating technique—1. algorithmic basis. *Computers & Mathematics with Applications*, 11(11):1099–1114.

Aachener Informatik-Berichte

This list contains all technical reports published during the past three years. A complete list of reports dating back to 1987 is available from:

`http://aib.informatik.rwth-aachen.de/`

To obtain copies please consult the above URL or send your request to:

Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen,
Email: `biblio@informatik.rwth-aachen.de`

2015-01 *	Fachgruppe Informatik: Annual Report 2015
2015-02	Dominik Franke: Testing Life Cycle-related Properties of Mobile Applications
2015-05	Florian Frohn, Jürgen Giesl, Jera Hensel, Cornelius Aschermann, and Thomas Ströder: Inferring Lower Bounds for Runtime Complexity
2015-06	Thomas Ströder and Wolfgang Thomas (Editors): Proceedings of the Young Researchers' Conference "Frontiers of Formal Methods"
2015-07	Hilal Diab: Experimental Validation and Mathematical Analysis of Cooperative Vehicles in a Platoon
2015-08	Mathias Pelka, Jó Agila Bitsch, Horst Hellbrück, and Klaus Wehrle (Editors): Proceedings of the 1st KuVS Expert Talk on Localization
2015-09	Xin Chen: Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models
2015-11	Stefan Wüller, Marián Kühnel, and Ulrike Meyer: Information Hiding in the Public RSA Modulus
2015-12	Christoph Matheja, Christina Jansen, and Thomas Noll: Tree-like Grammars and Separation Logic
2015-13	Andreas Polzer: Ansatz zur variantenreichen und modellbasierten Entwicklung von eingebetteten Systemen unter Berücksichtigung regelungs- und softwaretechnischer Anforderungen
2015-14	Niloofar Safiran and Uwe Naumann: Symbolic vs. Algorithmic Differentiation of GSL Integration Routines
2016-01 *	Fachgruppe Informatik: Annual Report 2016
2016-02	Ibtissem Ben Makhlouf: Comparative Evaluation and Improvement of Computational Approaches to Reachability Analysis of Linear Hybrid Systems
2016-03	Florian Frohn, Matthias Naaf, Jera Hensel, Marc Brockschmidt, and JÃ$\frac{1}{4}$rgen Giesl: Lower Runtime Bounds for Integer Programs
2016-04	Jera Hensel, Jürgen Giesl, Florian Frohn, and Thomas Ströder: Proving Termination of Programs with Bitvector Arithmetic by Symbolic Execution
2016-05	Mathias Pelka, Grigori Goronzy, Jó Agila Bitsch, Horst Hellbrück, and Klaus Wehrle (Editors): Proceedings of the 2nd KuVS Expert Talk on Localization
2016-06	Martin Henze, René Hummen, Roman Matzutt, Klaus Wehrle: The SensorCloud Protocol: Securely Outsourcing Sensor Data to the Cloud

2016-07	Sebastian Biallas : Verification of Programmable Logic Controller Code using Model Checking and Static Analysis
2016-08	Klaus Leppkes, Johannes Lotz, and Uwe Naumann: Derivative Code by Overloading in C++ (dco/c++): Introduction and Summary of Features
2016-09	Thomas Ströder, Jürgen Giesl, Marc Brockschmidt, Florian Frohn, Carsten Fuhs, Jera Hensel, Peter Schneider-Kamp, and Cornelius Aschermann: Automatically Proving Termination and Memory Safety for Programs with Pointer Arithmetic
2016-10	Stefan Wüller, Ulrike Meyer, and Susanne Wetzel: Towards Privacy-Preserving Multi-Party Bartering
2017-01 *	Fachgruppe Informatik: Annual Report 2017
2017-02	Florian Frohn and Jürgen Giesl: Analyzing Runtime Complexity via Innermost Runtime Complexity
2017-04	Florian Frohn and Jürgen Giesl: Complexity Analysis for Java with AProVE
2017-05	Matthias Naaf, Florian Frohn, Marc Brockschmidt, Carsten Fuhs, and Jürgen Giesl: Complexity Analysis for Term Rewriting by Integer Transition Systems
2017-06	Oliver Kautz, Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe: CD2Alloy: A Translation of Class Diagrams to Alloy
2017-07	Klaus Leppkes, Johannes Lotz, Uwe Naumann, and Jacques du Toit: Meta Adjoint Programming in C++
2017-08	Thomas Gerlitz: Incremental Integration and Static Analysis of Model-Based Automotive Software Artifacts
2017-09	Muhammad Hamad Alizai, Jan Beutel, Jó Ágila Bitsch, Olaf Landsiedel, Luca Mottola, Przemyslaw Pawelczak, Klaus Wehrle, and Kasim Sinan Yildirim: Proc. IDEA League Doctoral School on Transiently Powered Computing
2018-01 *	Fachgruppe Informatik: Annual Report 2018

* These reports are only available as a printed version.
Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.