A crash course in (Adjoint) Algorithmic Differentiation

Luca Capriotti

Guest Lecture for the Master of Finance
Baruch College, New York, December 14, 2017

What is Algorithmic Differentiation? (cont'd)

- Algorithmic or (Automatc) Differentiation (AD) is a set of programming techniques first introduced in the early 60's aimed at computing accurately and efficiently the derivatives of a function given in the form of a computer program.
- The main idea underlying AD is that any such program can be interpreted as the composition of functions each of which is in turn a composition of basic arithmetic (addition, multiplication etc.), and intrinsic operations (logarithm, exponential, etc.).
- Hence, it is possible to calculate the derivatives of the outputs of the program with respect to its inputs by applying mechanically the rules of differentiation.
- This makes it possible to generate *automatically* a computer program that evaluates efficiently and with machine precision accuracy the derivatives of the function [3].

What is Algorithmic Differentiation? (cont'd)

- AD is distinct from
  - *Symbolic differentiation*, which works by the symbolic manipulation of closed-form expressions using rules of differential calculus.
  - Numerical differentiation (the method of finite differences).
- What makes AD particularly attractive when compared to the methods above is its computational efficiency.
- In fact, AD aims at exploiting the information on the structure of the computer function, and on the dependencies between its various parts, in order to optimize the calculation of the sensitivities.
- AD comes in two main flavors, Tangent and Adjoint mode, which are characterized by different properties in different (complementary) computational complexity.

Algorithmic Differentiation: Tangent mode

- Consider a function
$$Y = \texttt{FUNCTION}(X)$$
mapping a vector X in $\mathbb{R}^n$ in a vector Y in $\mathbb{R}^m$.
- The execution time of its Tangent counterpart
$$\dot{X} = \texttt{FUNCTION\_d}(X, \dot{X})$$
(with suffix _d for "dot") calculating the linear combination of the columns of the Jacobian of the function:
$$\dot{Y}_j = \sum_{i=1}^{n} \dot{X}_i \frac{\partial Y_j}{\partial X_i},$$
with $j = 1, \ldots, m$, is bounded by
$$\frac{\mathrm{Cost}[\texttt{FUNCTION\_d}]}{\mathrm{Cost}[\texttt{FUNCTION}]} \leq \omega_T$$
with $\omega_T \in [2, 5/2]$.

Algorithmic Differentiation: Adjoint mode

- The execution time of the Adjoint counterpart of

$$Y = \texttt{FUNCTION}(X),$$

namely,

$$\bar{X} = \texttt{FUNCTION\_b}(X, \bar{Y})$$

(with suffix _b for "backward" or "bar") calculating the linear combination of the rows of the Jacobian of the function:

$$\bar{X}_i = \sum_{j=1}^{m} \bar{Y}_j \frac{\partial Y_j}{\partial X_i},$$

with $i = 1, \ldots, n$, is bounded by

$$\frac{\mathrm{Cost}[\texttt{FUNCTION\_b}]}{\mathrm{Cost}[\texttt{FUNCTION}]} \leq \omega_A$$

with $\omega_A \in [3, 4]$.

Algorithmic Differentiation: Tangent vs Adjoint mode

Given the results above:

- The Tangent mode is particularly well suited for the calculation of (linear combinations of) the columns of the Jacobian matrix.
- Instead, the Adjoint mode is particularly well-suited for the calculation of (linear combinations of) the rows of the Jacobian matrix .
- In particular, the Adjoint mode provides the full gradient of a scalar ($m = 1$) function at a cost which is just a small constant times the cost of evaluating the function itself. Remarkably such relative cost is *independent* of the number of components of the gradient.
- When the full Jacobian is required, the Adjoint mode is likely to be more efficient than the Tangent mode when the number of independent variables is significantly larger than the number of the dependent ones ($m \ll n$). Or viceversa.

Tangent mode: Propagating Forwards

- Imagine that the function $Y = \texttt{FUNCTION}(X)$ is implemented by means of a sequence of steps

$$X \rightarrow \ldots \rightarrow U \rightarrow V \rightarrow \ldots \rightarrow Y,$$

  where the real vectors U and V represent intermediate variables used in the calculation and each step can be a distinct high-level function or even an individual instruction.
- Define the Tangent of any intermediate variable U_k as

$$\dot{U}_k = \sum_{i=1}^{n} \dot{X}_i \frac{\partial U_k}{\partial X_i}.$$

Tangent mode: Propagating Forwards

- Using the chain rule we get,

$$\dot{V}_j = \sum_{i=1}^{n} \dot{X}_i \frac{\partial V_j}{\partial X_i} = \sum_{i=1}^{n} \dot{X}_i \sum_k \frac{\partial V_j}{\partial U_k} \frac{\partial U_k}{\partial X_i} = \sum_k \frac{\partial V_j}{\partial U_k} \dot{U}_k,$$

  which corresponds to the Tangent mode equation for the intermediate step represented by the function $V = V(U)$

$$\dot{V}_j = \sum_k \dot{U}_k \frac{\partial V_j}{\partial U_k},$$

  namely a function of the form $\dot{V} = \dot{V}(U, \dot{U})$.

Tangent mode: Propagating Forwards

- Hence the computation of the Tangents can be executed in the same direction of the original function

$$\dot{X} \rightarrow \ldots \rightarrow \dot{U} \rightarrow \dot{V} \rightarrow \ldots \rightarrow \dot{Y}.$$

This can be executed simultaneously with the original function, since at each intermediate step $U \rightarrow V$ one can compute the derivatives $\partial V_j(U)/\partial U_k$ and execute the Tangent forward propagation

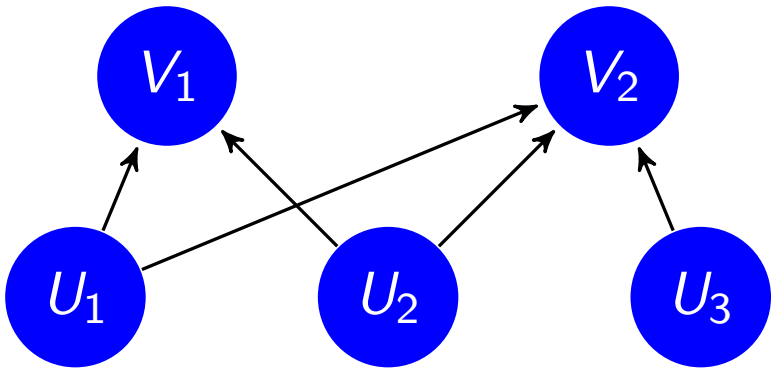
$$\dot{U} \rightarrow \dot{V} \qquad \dot{V}_j = \sum_k \dot{U}_k \frac{\partial V_j}{\partial U_k}.$$

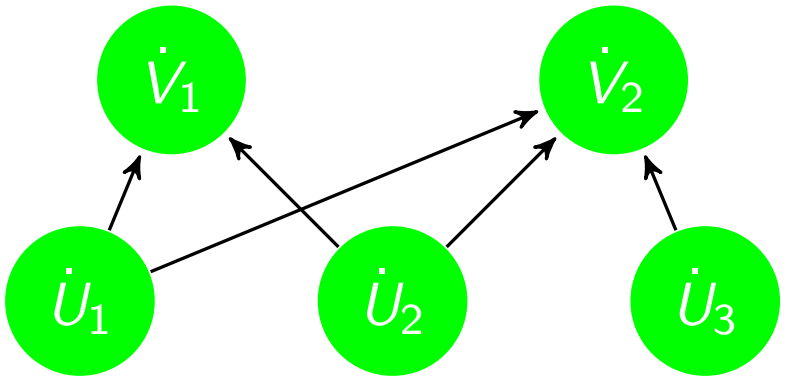
- The Tangent of the output obtained with this forward recursion is by definition:

$$\dot{Y}_k = \sum_{i=1}^{n} \dot{X}_i \frac{\partial Y_k}{\partial X_i},$$

i.e., in a single forward sweep one can produce a linear combination of the columns of the Jacobian $\partial Y/\partial X$.

Tangent mode: Propagating Forwards





$$V_1 = V_1(U_1, U_2)$$

$$\dot{V}_1 = \frac{\partial V_1}{\partial U_1}\dot{U}_1 + \frac{\partial V_1}{\partial U_2}\dot{U}_2$$

$$V_2 = V_2(U_1, U_2, U_3)$$

$$\dot{V}_2 = \frac{\partial V_2}{\partial U_1}\dot{U}_1 + \frac{\partial V_2}{\partial U_2}\dot{U}_2 + \frac{\partial V_2}{\partial U_3}\dot{U}_3$$

Tangent mode: Propagating Forwards

- The Tangent mode produces the linear combination of columns of the Jacobian

$$\dot{Y}_k = \sum_{i=1}^{n} \dot{X}_i \frac{\partial Y_k}{\partial X_i},$$

where $\dot{X}$ is an arbitrary vector in $\mathbb{R}^n$.

- By initializing in turn $\dot{X}$ with each vector of the canonical basis in $\mathbb{R}^n$, $(e_1, \ldots, e_n)$ with

$$e_j = (\underbrace{0, \ldots, 1}_{j}, 0, \ldots, 0)$$

one can obtain the partial derivatives of all the outputs with respect to each of the inputs $\dot{Y}_k = \partial Y_k / \partial X_i$, thus resulting in a cost that is n times the cost of a single forward Tangent sweep.

Tangent mode: Propagating Forwards

- It is not difficult to realize that the cost of computing each single step $\dot{U} \to \dot{V}$ is just a small multiple of the cost of executing $U \to V$.
- Consider for instance the example:

$$V_1 = \cos(U_1)U_1 + U_2 \exp(U_2)$$

the corresponding Tangents read

$$\dot{V}_1 = \dot{U}_1(-\sin(U_1)U_1 + cos(U_1)) + \dot{U}_2(U_2 + 1)\exp(U_2),$$

- Computing $\dot{V}$ (3 intrinsic operations, 4 multiplication and 3 additions) has the same computational complexity of computing the original function (2 intrinsic operations, 2 multiplications and 1 additions). Assuming that all the operations have the same cost it would be twice as expensive.

Tangent mode: Propagating Forwards

- Extending to the whole computation one can see how keeping into account of the relative cost of different types of operation one can arrive to the result [3]:

$$\frac{\mathrm{Cost}[\texttt{FUNCTION\_d}]}{\mathrm{Cost}[\texttt{FUNCTION}]} \leq \omega_T$$

  with $\omega_T \in [2, 5/2]$.
- By performing simultaneously the calculation of all the components of the gradient one can optimize the calculation by reusing a certain amount of computations (for instance the arc derivatives). This leads to a more efficient implementation also known as *Tangent Multimode*. The constant ω_T for these implementations is generally smaller than in the standard Tangent mode. However, the cost of computing the full gradient of a scalar function is still *proportional to the number of inputs*.

Adjoint mode: Propagating Backwards

- Let's consider again the function $Y = \texttt{FUNCTION}(X)$ implemented by means of a sequence of steps

$$X \rightarrow \ldots \rightarrow U \rightarrow V \rightarrow \ldots \rightarrow Y.$$

- Define the Adjoint of any intermediate variable V_k as

$$\bar{V}_k = \sum_{j=1}^{m} \bar{Y}_j \frac{\partial Y_j}{\partial V_k},$$

where $\bar{Y}$ is vector in $\mathbb{R}^m$.

Adjoint mode: Propagating Backwards

- Using the chain rule we get,

$$\bar{U}_i = \sum_{j=1}^{m} \bar{Y}_j \frac{\partial Y_j}{\partial U_i} = \sum_{j=1}^{m} \bar{Y}_j \sum_k \frac{\partial Y_j}{\partial V_k} \frac{\partial V_k}{\partial U_i},$$

  which corresponds to the Adjoint mode equation for the intermediate step represented by the function $V = V(U)$

$$\bar{U}_i = \sum_k \bar{V}_k \frac{\partial V_k}{\partial U_i},$$

  namely a function of the form $\bar{U} = \bar{V}(U, \bar{V})$.

Adjoint mode: Propagating Backwards

- Starting from the Adjoint of the outputs, $\bar{Y}$, we can apply this rule to each step in the calculation, working from right to left,

$$\bar{X} \leftarrow \ldots \leftarrow \bar{U} \leftarrow \bar{V} \leftarrow \ldots \leftarrow \bar{Y}$$

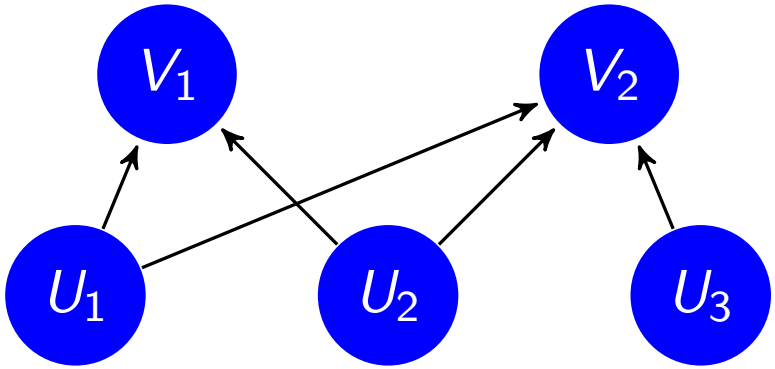
  until we obtain $\bar{X}$, i.e., the following linear combination of the rows of the Jacobian $\partial Y / \partial X$

$$\bar{X}_i = \sum_{j=1}^{m} \bar{Y}_j \frac{\partial Y_j}{\partial X_i},$$

  with $i = 1, \ldots, n$.

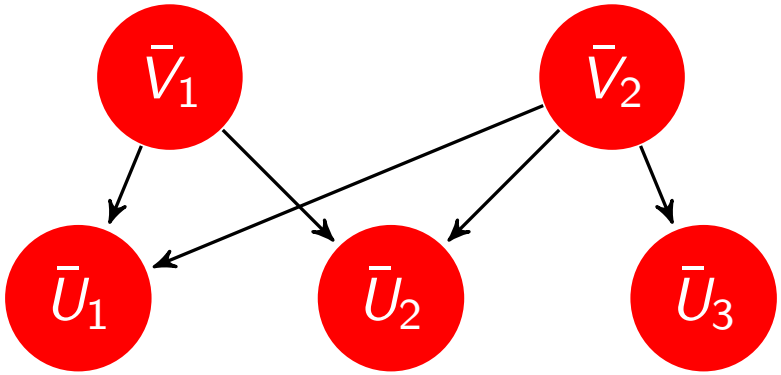
- Contrary to the Tangent mode, the backward propagation can start only after the calculation of the function an the intermediate variables have been computed and stored.

Adjoint mode: Propagating Backwards



$$V_1 = V_1(U_1, U_2)$$

$$V_2 = V_2(U_1, U_2, U_3)$$



$$\bar{U}_1 = \frac{\partial V_1}{\partial U_1}\bar{V}_1 + \frac{\partial V_2}{\partial U_1}\bar{V}_2$$

$$\bar{U}_2 = \frac{\partial V_1}{\partial U_2}\bar{V}_1 + \frac{\partial V_2}{\partial U_2}\bar{V}_2$$

$$\bar{U}_3 = \frac{\partial V_2}{\partial U_3}\bar{V}_2$$

Adjoint mode: Propagating Backwards

- Consider as before the example:

$$V_1 = \cos(U_1)U_1 + U_2 \exp(U_2)$$

the corresponding Adjoints read

$$\bar{U}_1 = \bar{V}_1(-\sin(U_1)U_1 + \cos(U_1)),$$
$$\bar{U}_2 = \bar{V}_1(U_2 + 1)\exp(U_2))$$

- Computing $\bar{U}$ (3 intrinsic operations, 4 multiplications and 2 additions) has the same computational complexity of computing the original function (2 intrinsic operations, 2 multiplications and 1 addition). Assuming that all the operations have the same cost it would be about twice as expensive.

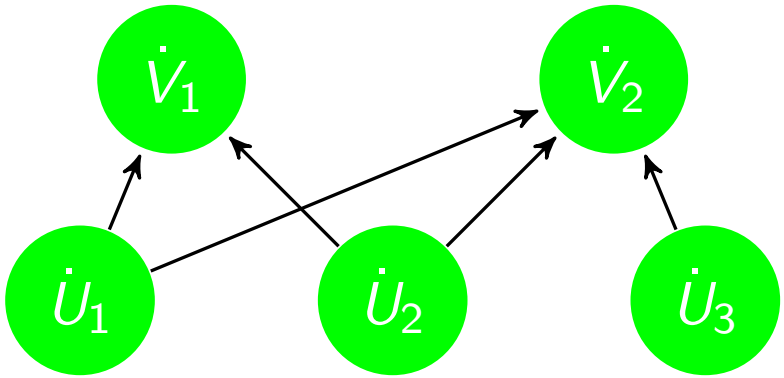
Adjoint mode: Propagating Backwards

- Extending to the whole computation one can see how keeping into account of the relative cost of different types of operation one can arrive to the result [3]:

$$\frac{\mathrm{Cost}[\texttt{FUNCTION\_b}]}{\mathrm{Cost}[\texttt{FUNCTION}]} \leq \omega_A$$

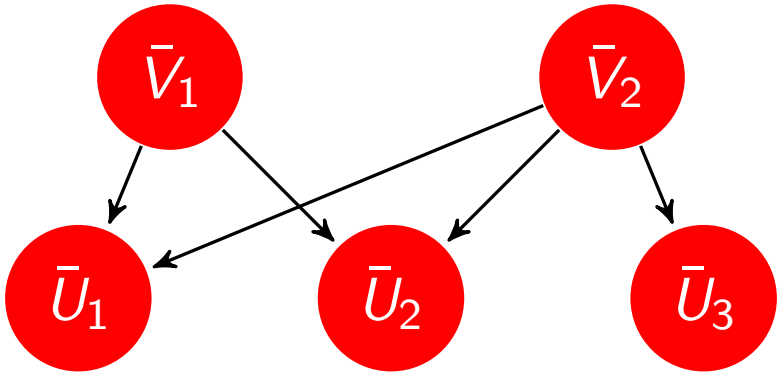
  with $\omega_A \in [3, 4]$.
- This result is based on the number of arithmetic operations which must be performed. It also includes the cost of memory operations, but assumes a uniform cost for these, irrespective of the total amount of memory used. This assumption is violated in practice due to the cache hierarchy in modern computers.
- Nevertheless, it remains true in practice that one can obtain the sensitivity of a single output, or a linear combination of outputs, to an unlimited number of inputs for only a little more work than the original calculation.

Recap: Forwards vs Backwards



$$\dot{V}_1 = \frac{\partial V_1}{\partial U_1}\dot{U}_1 + \frac{\partial V_1}{\partial U_2}\dot{U}_2$$

$$\dot{V}_2 = \frac{\partial V_2}{\partial U_1}\dot{U}_1 + \frac{\partial V_2}{\partial U_2}\dot{U}_2 + \frac{\partial V_2}{\partial U_3}\dot{U}_3$$

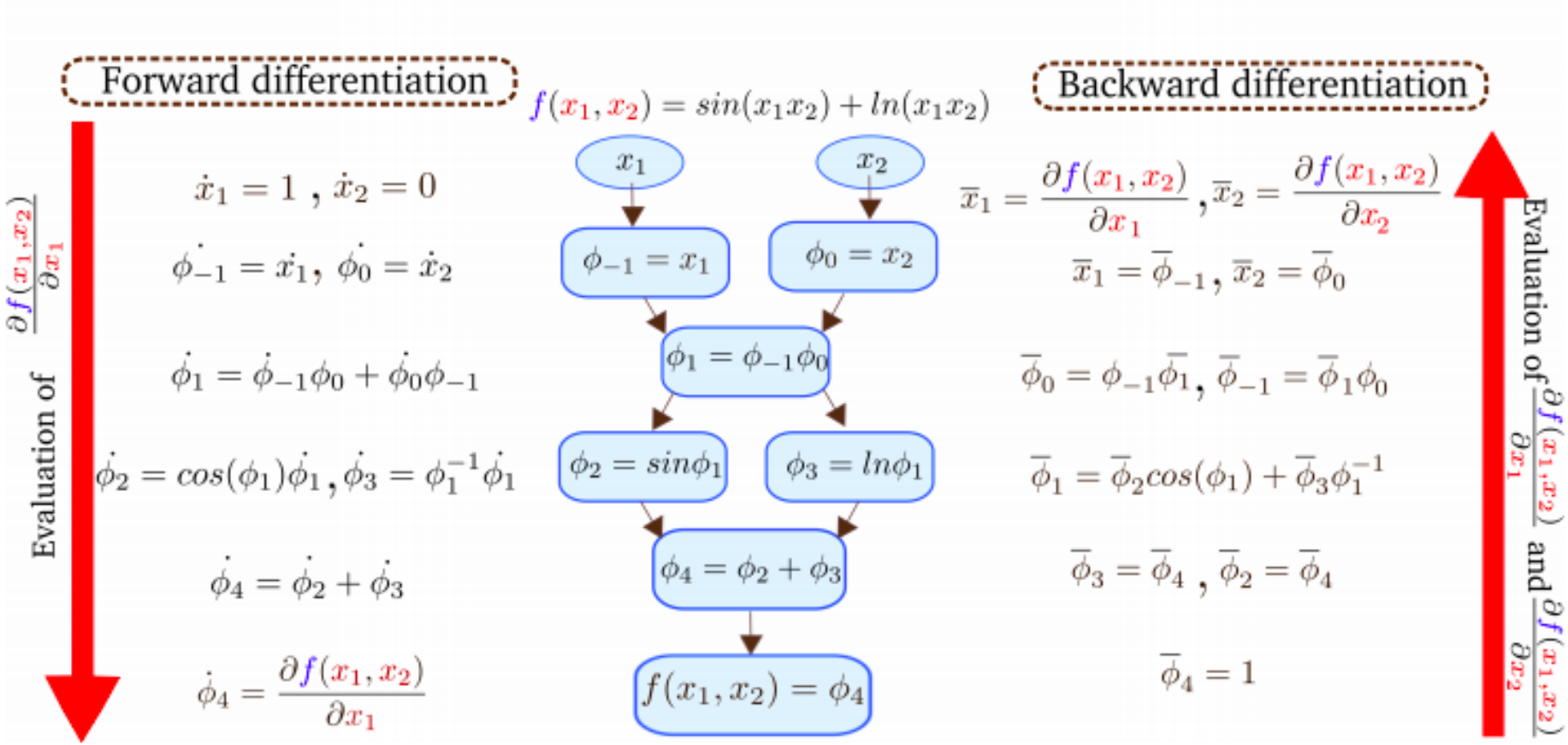


$$\bar{U}_1 = \frac{\partial V_1}{\partial U_1}\bar{V}_1 + \frac{\partial V_2}{\partial U_1}\bar{V}_2$$

$$\bar{U}_2 = \frac{\partial V_1}{\partial U_2}\bar{V}_1 + \frac{\partial V_2}{\partial U_2}\bar{V}_2$$

$$\bar{U}_3 = \frac{\partial V_2}{\partial U_3}\bar{V}_2$$

Simple Example: Forwards vs Backwards



First Financial Examples: Derivatives of Payoff Functions

- As a first example let's consider the Payoff of a Basket Option

$$P(X(T)) = e^{-rT}\left(\sum_{i=1}^{N} w_i X_i(T) - K\right)^+ ,$$

where $X(T) = (X_1(T), \ldots, X_N(T))$ represent the value of a set of N underlying assets, say a set of equity prices, at time T, w_i, $i = 1, \ldots, N$, are the weights defining the composition of the basket, K is the strike price, and r is the risk free yield for the considered maturity.

- For this example, we are interested in the calculation of the sensitivities with respect to r and the N components of the state vector X so that the other parameters, i.e., strike and maturity, are seen here as dummy constants.

Pseudocode of the Basket Option

```
(P)= payout (r, X[N]){

  B = 0.0;
  for (i = 1 to N)
   B += w[i]* X[i];

  x = B - K;
  D = exp(-r * T);
  P = D * max(x, 0.0);
};
```

From Ref. [2]

Pseudocode of the Tangent Payoff for the Basket Option

```
(P, P_d)= payout_d(r, X[N], r_d, X_d[N]){

  B = 0.0;
  for (i = 1 to N) {
   B += w[i]*X[i];
   B_d += w[i]*X_d[i];
  }

  x = B - K;
  x_d = B_d;

  D = exp(-r * T);
  D_d = -T * D * r_d;

  P = D * max(x, 0.0);
  P_d = 0;
  if(x > 0)
   P_d = D_d*x + D*x_d;

};
```

From Ref. [2]

- The computational cost of the Tangent payoff is of the same order of the original Payoff.
- To get all the components of the gradient of the payoff, the Tangent payoff code must be run $N + 1$ times, setting in turn one component of the Tangent input vector $I = (\dot{r}, \dot{X})^t$ to one and the remaining ones to zero.

Pseudocode of the MultimodeTangent Payoff for the Basket Option

```
(P, P_d[Nd]) = payout_dv(r, X[N], r_d[Nd], X_d[N,Nd]){

  B = 0.0;
  for (id = 1 to Nd)
     B_d[id] = 0.0;

  for (i = 1 to N) {
    B += w[i]*X[i];
    for (j = 1 to Nd)
      B_d[j] += w[i]*X_d[i,j];
  }
  x = B - K;
  for (j = 1 to Nd)
    x_d[j] = B_d[j];

  D = exp(-r * T);
  for (j = 1 to Nd)
    D_d[j] = -T * D * r_d[j];

  P = D * max(x, 0.0);
  P_d = 0;
  if(x > 0){
     for (j = 1 to Nd)
        P_d[j] = D_d[j]*x + D*x_d[j];
  }
};
```

From Ref. [2]

- To get all the components of the gradient of the payoff, the Tangent payoff code must be run only once.
- The computational cost of the Multimode Tangent payoff still scales as N times the cost of the original Payoff.

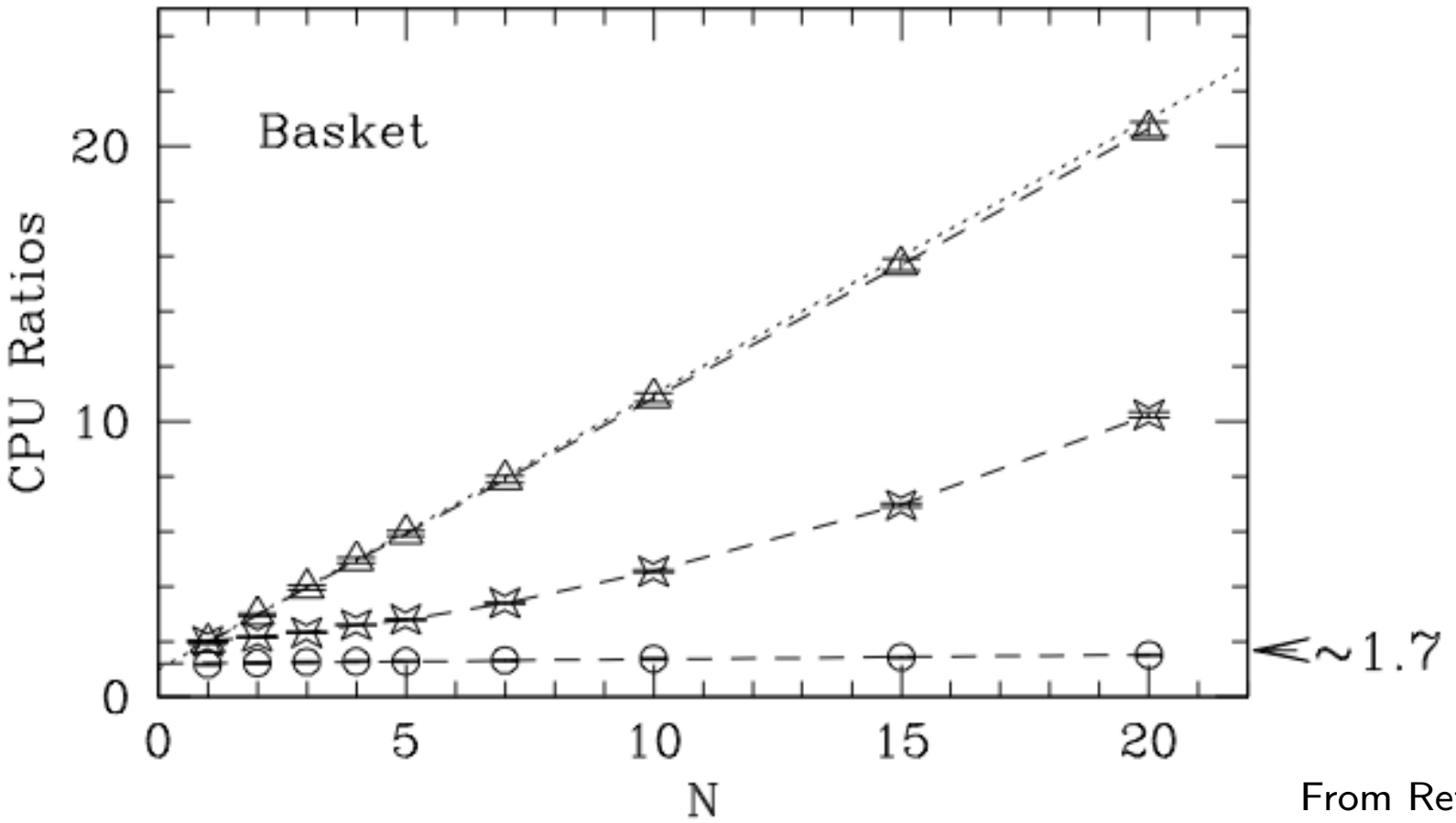
Pseudocode of the Adjoint Payoff for the Basket Option

```
(P, r_b, X_b[N]) = payout_b(r, X[N], P_b){

                                // Forward sweep
  B = 0.0;
  for (i = 0 to N)
   B += w[i] * X[i];

  x = B - K;
  D = exp(-r * T);
  P = D * max(x, 0.0);
                                // Backward sweep
  D_b = max(x, 0.0) * P_b;

  x_b = 0.0;
  if (x > 0)
   x_b = D * P_b;

  r_b = - D * T * D_b;
  B_b = x_b;

  for (i = 0 to N)
   X_b[i] = w[i] * B_b;
};
```

From Ref. [2]

- The Adjoint payoff contains a forward sweep.
- The computational cost of the Adjoint payoff is of the same order of the original Payoff.
- All the components of the gradient of the payoff, are obtained by running the Adjoint payoff only once setting $\bar{P} = 1$.

Tangent vs Adjoint



From Ref. [2]

- The Tangent payoff performs similarly to bumping (much better for the Multimode version) and has a computational complexity that scales with the number of inputs.
- In the Adjoint mode the calculation of all the derivatives of the payoff requires an extra overhead of just 70% with respect to the calculation of the payoff itself for *any* number of inputs.

Tangent vs Adjoint

- In general we are interested in computing the sensitivities of a derivative or of a portfolio of derivatives with respect to a large number of risk factors.
- The Adjoint model of Algorithmic Differentiation is therefore the one best suited for the task.
- In some applications, however, one is also interested in computing the sensitivities of a multiplicity of derivatives individually. In those cases one can effectively combine the Adjoint and Tangent mode. See e.g. [2].
- In the following we will concentrate on the Adjoint mode of Algorithmic Differentiation (AAD) as it is the one of wider applicability.

Section 1

Differentiating Programs

AAD as a Design Principle

- The propagation of the Adjoints according to the steps, being mechanical in nature, can be automated.
- Several AD tools are available that given a procedure of the form:

$$Y = \texttt{FUNCTION}(X),$$

generate the Adjoint function:

$$\bar{X} = \texttt{FUNCTION\_b}(X, \bar{Y}).$$

- An excellent source of information can be found at www.autodiff.org.

AAD as a Design Principle

- The principles of AD can be used as a programming paradigm for any algorithm.
- An easy way to illustrate the Adjoint design paradigm is to consider again the arbitrary computer function

$$Y = \texttt{FUNCTION}(X),$$

  and to imagine that this represents a certain high level algorithm that we want to differentiate.
- By appropriately defining the intermediate variables, any such algorithm can be abstracted in general as a composition of functions like

$$X \rightarrow \ldots \rightarrow U \rightarrow V \rightarrow \ldots \rightarrow Y.$$

AAD as a Design Principle

- However, the actual calculation graph might have a more complex structure. For instance the step $U \to V$ might be implemented in terms of two computer functions of the form

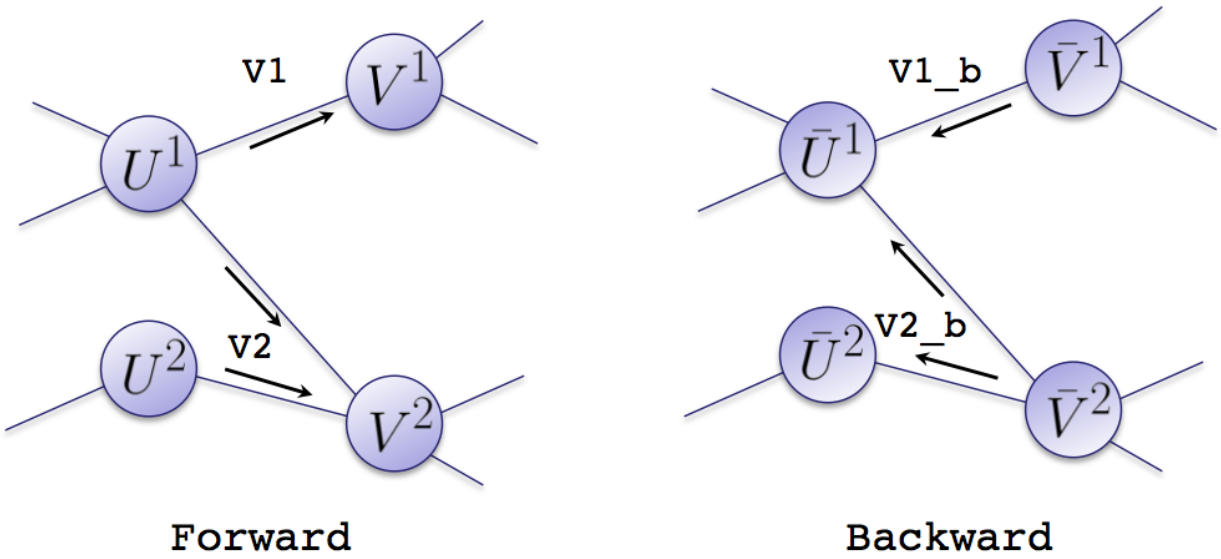
$$\begin{aligned} V^1 &:= \mathbb{V}1(U^1) \,, \\ V^2 &:= \mathbb{V}2(U^1, U^2) \,, \end{aligned}$$

  with $U = (U^1, U^2)^t$ and $V = (V^1, V^2)^t$. Here the notation $W = (W^1, W^2)^t$ simply indicates a specific partition of the components of the vector W in two sub-vectors.
- A natural way to represent the step $\bar{U} \leftarrow \bar{V}$ in

$$\bar{X} \leftarrow \ldots \leftarrow \bar{U} \leftarrow \bar{V} \leftarrow \ldots \leftarrow \bar{Y}$$

  i.e., the function $\bar{U} = \bar{V}(U, \bar{V})$, can be given in terms of an Adjoint calculation graph.

AAD as a Design Principle



- The Adjoint graph has the same structure of the original graph with each node/variable representing the Adjoint of the original node/variable, and it is executed in opposite direction with respect to the original one.

AAD as a Design Principle

- The relation between the Adjoint nodes is defined by the correspondence between $Y = \texttt{FUNCTION}(X)$ and $\bar{X} = \texttt{FUNCTION\_b}(X, \bar{Y})$., e.g., in the specific example

$$\begin{aligned} (\bar{U}^1, \bar{U}^2)^t &:= \texttt{V2\_b}(U^1, U^2, \bar{V}^2) \,, \\ \bar{U}^1 &:= \bar{U}^1 + \texttt{V1\_b}(U^1, \bar{V}^1) \,. \end{aligned}$$

- This can be understood as it follow: the variable U^1 is an input of two distinct functions so that, by applying the definition of Adjoint for the variable U^1 as an input of the function $V = V(U^1, U^2) = (V^1(U^1), V^2(U^1, U^2))^t$, we get

$$\bar{U}^1 = \sum_j \bar{V}_j \frac{\partial V_j}{\partial U^1} = \sum_k \bar{V}^1_k \frac{\partial V^1_k}{\partial U^1} + \sum_k \bar{V}^2_k \frac{\partial V^2_k}{\partial U^1}$$

where we have simply partitioned the components of the vector V as $(V^1, V^2)^t$ for the second equality.

AAD as a Design Principle

- Similarly, one has for $\bar{U}^2$

$$\bar{U}^2 = \sum_j \bar{V}_j \frac{\partial V_j}{\partial U^2} = \sum_k \bar{V}_k^2 \frac{\partial V_k^2}{\partial U^2},$$

  where we have used the fact that V^1 has no dependence on U^2.

- Therefore, one can realize that the Adjoint calculation graph implementing the instructions in

$$\begin{aligned} (\bar{U}^1, \bar{U}^2)^t &:= \mathtt{V2\_b}(U^1, U^2, \bar{V}^2) \ , \\ \bar{U}^1 &:= \bar{U}^1 + \mathtt{V1\_b}(U^1, \bar{V}^1) \ . \end{aligned}$$

  indeed produces the Adjoint $\bar{U} = (\bar{U}^1, \bar{U}^2)^t$.

Forward and Backward Sweeps

- The Adjoint instructions

$$\begin{aligned}(\bar{U}^1, \bar{U}^2)^t &:= \mathtt{V2\_b}(U^1, U^2, \bar{V}^2) \ , \\ \bar{U}^1 &:= \bar{U}^1 + \mathtt{V1\_b}(U^1, \bar{V}^1) \ .\end{aligned}$$

  depend on the variables U^1 and U^2.
- As a result, the Adjoint algorithm can be executed only after the original instructions

$$X \ \rightarrow \ \ldots \ \rightarrow \ U \ \rightarrow \ V \ \rightarrow \ \ldots \ \rightarrow \ Y.$$

  have been executed and the necessary intermediate results have been computed and stored.
- This is the reason why, as note before, the Adjoint of a given algorithm generally contains a *forward sweep*, which reproduces the steps of the original algorithm, plus a *backward sweep*, which propagates the Adjoints.

Section 2

Computing Risk in Monte Carlo

Option Pricing Problems

- Option pricing problems can be typically formulated in terms of the calculation of expectation values of the form

$$V = \mathbb{E}_{\mathbb{Q}}\Big[P(X(T_1), \ldots, X(T_M))\Big].$$

- Here $X(t)$ is a N-dimensional vector and represents the value of a set of underlying market factors (e.g., stock prices, interest rates, foreign exchange pairs, etc.) at time t.
- $P(X(T_1), \ldots, X(T_M))$ is the discounted payout function of the priced security, and depends in general on M observations of those factors.
- In the following, we will indicate the collection of such observations with a $d = N \times M$ dimensional state vector

$$X = (X(T_1), \ldots, X(T_M))^t.$$

Monte Carlo Sampling of the Payoff Estimator

- The expectation value above can be estimated by means of Monte Carlo (MC) by sampling a number N_{MC} of random replicas of the underlying state vector $X[1], \ldots, X[N_{\mathrm{MC}}]$, sampled according to the distribution $\mathbb{Q}(X)$, and evaluating the payout $P(X)$ for each of them.
- This leads to the estimate of the option value V as

$$V \simeq \frac{1}{N_{\mathrm{MC}}} \sum_{i_{\mathrm{MC}}=1}^{N_{\mathrm{MC}}} P\left(X[i_{\mathrm{MC}}]\right),$$

with standard error $\Sigma/\sqrt{N_{\mathrm{MC}}}$, where

$$\Sigma^2 = \mathbb{E}_{\mathbb{Q}}[P\left(X\right)^2] - \mathbb{E}_{\mathbb{Q}}[P\left(X\right)]^2$$

is the variance of the sampled payout.

Pathwise Derivative Method

- The Pathwise Derivative Method allows the calculation of the sensitivities of the option price V with respect to a set of N_θ parameters $\theta = (\theta_1, \ldots, \theta_{N_\theta})$, with a single set of N_{MC} simulations.
- This can be achieved by noticing that, whenever the payout function is regular enough, e.g., Lipschitz-continuous, and under additional conditions that are often satisfied in financial pricing (see, e.g., [1]), one can write the sensitivity $\langle \bar{\theta}_k \rangle \equiv dV/d\theta_k$ as

$$\langle \bar{\theta}_k \rangle = \mathbb{E}_{\mathbb{Q}}\left[\frac{dP_\theta(X)}{d\theta_k}\right].$$

- In the context of MC simulations, this equation can be easily understood by thinking the random sampling of the state vector X as performed in terms of a mapping of the form, $X = X(\theta; Z)$, where Z is a random vector *independent* of θ. In fact, after this mapping, the expectation value $\mathbb{E}_{\mathbb{Q}}[\ldots]$ can be expressed as an average over the probability distribution of Z, $\mathbb{Q}(Z)$, which is independent of θ.

Pathwise Derivative Method: Interpretation

- The calculation of $\langle \bar{\theta}_k \rangle$ can be performed by applying the chain rule, and averaging on each MC sample the so-called Pathwise Derivative Estimator

$$\bar{\theta}_k \equiv \frac{dP_\theta(X)}{d\theta_k} = \sum_{j=1}^{d} \frac{\partial P_\theta(X)}{\partial X_j} \times \frac{\partial X_j}{\partial \theta_k} + \frac{\partial P_\theta(X)}{\partial \theta_k}.$$

- The matrix of derivatives of each state variable, or *Tangent state vector*, is by definition given by

$$\frac{\partial X_j}{\partial \theta_k} = \lim_{\Delta\theta \to 0} \frac{X_j(\theta_1, \ldots, \theta_k + \Delta\theta, \ldots, \theta_{N_\theta}) - X_j(\theta)}{\Delta\theta}.$$

- This gives the intuitive interpretation of $\partial X_j / \partial \theta_k$ in terms of the difference between the sample of the j-th component of the state vector obtained after an infinitesimal 'bump' of the k-th parameter, $X_j(\theta_1, \ldots, \theta_k + \Delta\theta, \ldots, \theta_{N_\theta})$, and the base sample $X_j(\theta)$, both calculated on *the same random realization*.

Pathwise Derivative Method: Diffusions

- Consider the case for instance in which the state vector X is a path of a N-dimensional diffusive process,

$$dX(t) = \mu(X(t), t, \theta)\, dt + \sigma(X(t), t, \theta)\, dW_t,$$

with $X(t_0) = X_0$. Here the drift $\mu(X, t, \theta)$ and volatility $\sigma(X, t, \theta)$ are N-dimensional vectors and W_t is a N-dimensional Brownian motion with instantaneous correlation matrix $\rho(t)$ defined by $\rho(t)\, dt = \mathbb{E}_{\mathbb{Q}}\left[dW_t dW_t^T\right]$.

- The Pathwise Derivative Estimator may be rewritten as

$$\bar{\theta}_k = \sum_{l=1}^{M} \sum_{j=1}^{N} \frac{\partial\, P(X(T_1), \ldots, X(T_M))}{\partial X_j(T_l)} \frac{\partial X_j(T_l)}{\partial \theta_k} + \frac{\partial P_\theta(X)}{\partial \theta_k}$$

where we have relabeled the d components of the state vector X grouping together different observations $X_j(T_1), \ldots, X_j(T_M)$ of the same (j-th) asset.

Pathwise Derivative Method: Diffusions

- In particular, the components of the Tangent vector for the k-th sensitivity corresponding to observations at times $(T_1, \ldots, T_M)$ along the path of the j-th asset, say,

$$\Delta_{jk}(T_l) = \frac{\partial X_j(T_l)}{\partial \theta_k}$$

with $l = 1, \ldots, M$, can be obtained by solving a stochastic differential equation

$$\begin{aligned} d\Delta_{jk}(t) = \sum_{i=1}^{N} \left[\frac{\partial \mu_j(X(t), t; \theta)}{\partial X_i(t)}\, dt + \frac{\partial \sigma_j(X(t), t; \theta)}{\partial X_i(t)}\, dW_t^j \right] \Delta_{ik}(t) \\ + \left[\frac{\partial \mu_j(X(t), t; \theta)}{\partial \theta_k}\, dt + \frac{\partial \sigma_j(X(t), t; \theta)}{\partial \theta_k}\, dW_t^j \right], \end{aligned}$$

with the initial condition $\Delta_{jk}(0) = \partial X_j(0)/\partial \theta_k$.

Pathwise Derivative Method: Is it worth the trouble?

- The Pathwise Derivative Estimators of the sensitivities are mathematically equivalent to the estimates obtained by standard finite differences approaches when using the same random numbers in both simulations and for a vanishing small perturbation. In this limit, the Pathwise Derivative Method and finite differences estimators provide exactly the same estimators for the sensitivities, i.e., estimators with the same expectation value, *and* the same MC variance.
- As a result, the implementation effort associated with the Pathwise Derivative Method is generally justified if the computational cost of the Pathwise Estimator is significantly less than the corresponding finite differences one.
- This is the case for instance in very simple models but difficult to achieve for those used in the financial practice.

Section 3

AAD and the Pathwise Derivative Method

AAD and the Pathwise Derivative Method

- AAD provides a general design and programming paradigm for the efficient implementation of the Pathwise Derivative Method.
- This stems from the observation that the Pathwise Estimator in

$$\bar{\theta}_k \equiv \frac{dP_\theta(X)}{d\theta_k} = \sum_{j=1}^{d} \frac{\partial P_\theta(X)}{\partial X_j} \times \frac{\partial X_j}{\partial \theta_k} + \frac{\partial P_\theta(X)}{\partial \theta_k},$$

  is a l.c. of the rows of the Jacobian of the map $\theta \to X(\theta)$, with weights given by the X gradient of the payout function $P_\theta(X)$, plus the derivatives of the payout function with respect to θ.
- Both the calculation of the derivatives of the payout and of the linear combination of the rows of $\partial X/\partial\theta$ are tasks that can be performed efficiently by AAD.
- We know now that we can compute all the Pathwise sensitivities with respect to θ, $\bar{\theta}$, at a cost that is at most roughly 4 times the cost of calculating the payout estimator itself.

AAD enabled Monte Carlo Engines: Forward Sweep

- In a typical MC simulation, in order to generate each sample $X[i_{\mathrm{MC}}]$, the evolution of the process X is usually simulated, possibly by means of an approximate discretization scheme, by sampling $X(t)$ on a discrete grid of points, $0 = t_0 < t_1 < \ldots < t_n < \cdots < t_{N_s}$, a superset of the observation times $(T_1, \ldots, T_M)$.
- The state vector at time t_{n+1} is obtained by means of a function of the form

$$X(t_{n+1}) = \mathtt{PROP}_n[\{X(t_m)\}_{m \leq n}, Z(t_n), \theta],$$

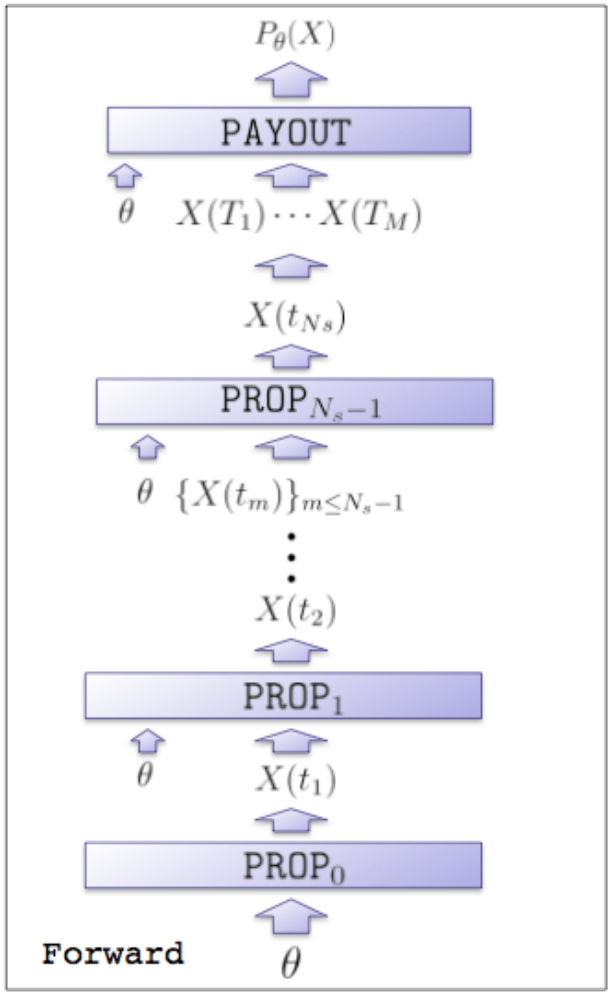
mapping the set of state vector values on the discretization grid up to t_n, $\{X(t_m)\}_{m \leq n}$, into the value of the state vector at time $t_n + 1$.

AAD enabled Monte Carlo Engines: Forward Sweep

- Note that in $X(t_{n+1}) = \texttt{PROP}_n[\{X(t_m)\}_{m\leq n}, Z(t_n), \theta]$:
  a. The propagation method is a function of the model parameters θ and of the particular time step considered.
  b. $Z(t_n)$ indicates the vector of uncorrelated random numbers which are used for the MC sampling in the step $n \to n+1$.
  c. The initial values of the state vector $X(t_0)$ are known quantities and they can be considered as components of θ so that the $n = 0$ step is of the form, $X(t_1) = \texttt{PROP}_0[Z(t_0), \theta]$.
- Once the full set of state vector values on the simulation time grid $\{X(t_m)\}_{m\leq N_s}$ is obtained, the subset of values corresponding to the observation dates is passed to the the payout function, evaluating the payout estimator $P_\theta(X)$ for the specific random sample $X[i_{\mathrm{MC}}]$

$$(X(T_1), \ldots, X(T_M)) \to P_\theta(X(T_1), \ldots, X(T_M)).$$

AAD enabled Monte Carlo Engines: Forward Sweep



Schematic illustration of the orchestration of a MC engine.

AAD enabled Monte Carlo Engines: Backward Sweep

- The evaluation of a MC sample of a Pathwise Estimator can be seen as an algorithm implementing a function of the form $\theta \to P(\theta)$.
- As a result, it is possible to design its Adjoint counterpart $(\theta, \bar{P}) \to (P, \bar{\theta})$ which gives (for $\bar{P} = 1$) the Pathwise Derivative Estimator $dP/d\theta_k$.
- The backward sweep can be simply obtained by reversing the flow of the computations, and associating to each function its Adjoint counterpart.

AAD enabled Monte Carlo Engines: Backward Sweep

- In particular, the first step of the Adjoint algorithm is the Adjoint of the payout evaluation $P = P(X, \theta)$. This is a function of the form

$$(\bar{X}, \bar{\theta}) = \bar{P}(X, \theta, \bar{P}),$$

where $\bar{X} = (\bar{X}(T_1), \ldots, \bar{X}(T_M))$ is the Adjoint of the state vector on the observation dates, and $\bar{\theta}$ is the Adjoint of the model parameters vector, respectively (for $\bar{P} = 1$)

$$\begin{aligned} \bar{X}(T_m) &= \frac{\partial P_\theta(X)}{\partial X(T_m)}, \\ \bar{\theta} &= \frac{\partial P_\theta(X)}{\partial \theta}, \end{aligned}$$

for $m = 1, \ldots, M$. The Adjoint of the state vector on the simulation dates corresponding to the observation dates are initialized at this stage. The remaining ones are initialized to zero.

AAD enabled Monte Carlo Engines: Backward Sweep

- The Adjoint state vector is then propagated backwards in time through the Adjoint of the propagation method, namely

$$(\{\bar{X}(t_m)\}_{m\leq n}, \bar{\theta}) \mathrel{+}= \texttt{PROP\_b}_n[\{X(t_m)\}_{m\leq n}, Z(t_n), \theta, \bar{X}(t_{n+1})],$$

for $n = N_s - 1, \ldots, 1$, giving

$$\bar{X}(t_m) \mathrel{+}= \sum_{j=1}^{N} \bar{X}_j(t_{n+1}) \frac{\partial X_j(t_{n+1})}{\partial X(t_m)},$$

with $m = 1, \ldots, n$,

$$\bar{\theta} \mathrel{+}= \sum_{j=1}^{N} \bar{X}_j(t_{n+1}) \frac{\partial X_j(t_{n+1})}{\partial \theta}.$$

AAD enabled Monte Carlo Engines: Backward Sweep

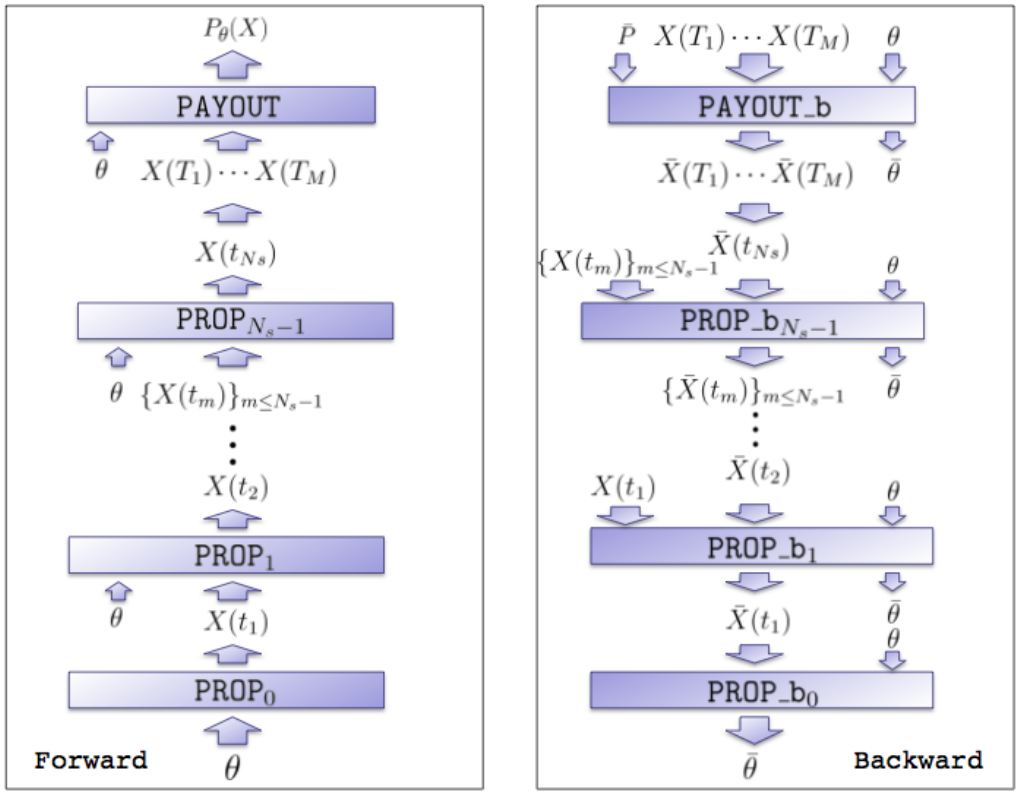
- Here, according to the principles of AAD, the Adjoint of the propagation method takes as arguments the inputs of its forward counterpart, namely the state vectors up to time t_n, $\{X(t_m)\}_{m\leq n}$, the vector of random variates $Z(t_n)$, and the θ vector. The additional input is the Adjoint of the state vector at time t_{n+1}, $\bar{X}(t_{n+1})$.
- The return values of $\texttt{PROP\_b}_n$ are the contributions associated with the step $n+1 \to n$ to the Adjoints of
  - i) the state vector $\{\bar{X}(t_m)\}_{m\leq n}$;
  - ii) the model parameters $\bar{\theta}_k$, $k = 1, \ldots, N_\theta$.
- The final step of the backward propagation corresponds to the Adjoint of $X(t_1) = \texttt{PROP}_0[Z(t_0), \theta]$, giving

$$\bar{\theta} \mathrel{+}= \texttt{PROP\_b}_0[X(t_0)\, Z(t_0), \theta, \bar{X}(t_1)],$$

  i.e., the final contribution to the Adjoints of the model parameters.
- It is easy to verify that the final result is the Pathwise Derivative Estimator $dP/d\theta_k$ for all k's on the given MC path.

AAD enabled Monte Carlo Engines: The complete blueprint

- The resulting algorithm can be illustrated as follows:



Schematic illustration of the orchestration of an AAD enabled MC engine.

Diffusion Processes and Euler Discretization

- As a first example, consider the case in which the underlying factors follow multi dimensional diffusion processes introduced in slide 3

$$dX(t) = \mu(X(t), t, \theta)\, dt + \sigma(X(t), t, \theta)\, dW_t.$$

- In this case, the evolution of the process X is usually approximated by sampling $X(t)$ on a discrete grid of points by means, for instance, of an Euler scheme, so that the propagation function

$$X(t_{n+1}) = \texttt{PROP}_n[\{X(t_m)\}_{m \leq n}, Z(t_n), \theta]$$

implements the rule

$$X(t_{n+1}) = X(t_n) + \mu(X(t_n), t_n, \theta)\, h_n + \sigma(X(t_n), t_n, \theta)\, \sqrt{h_n}\, Z(t_n),$$

where $h_n = t_{n+1} - t_n$, and $Z(t_n)$ is a N-dimensional vector of correlated unit normal random variables.

Diffusion Processes and Euler Discretization: Forward Sweep

- In particular, given the state vector at time t_n, $X(t_n)$, and the vector $Z(t_n)$, one can implement the method $\texttt{PROP}_n$ according to the following steps:

Step 1. Compute the drift vector, by evaluating the function:

$$\mu(t_n) = \mu(X(t_n), t_n, \theta) \ .$$

Step 2. Compute the volatility vector, by evaluating the function:

$$\sigma(t_n) = \sigma(X(t_n), t_n, \theta) \ .$$

Step 3. Compute the function

$$(X(t_n), \mu(t_n), \sigma(t_n), Z(t_n), \theta) \to X(t_{n+1}) \ ,$$

defined by

$$X(t_{n+1}) = X(t_n) + \mu(t_n)h_n + \sigma(t_n)\sqrt{h_n} \cdot Z(t_n) \ .$$

Diffusion Processes and Euler Discretization: Backward Sweep

- The corresponding Adjoint method $\texttt{PROP\_b}_n$ is executed from time step t_{n+1} to t_n and consists of the Adjoint counterpart of each of the steps above executed in reverse order, namely:

Step $\bar{3}$. Compute the Adjoint of the function defined by Step 3. This is a function

$$(X(t_n), \mu(t_n), \sigma(t_n), Z(t_n), \bar{X}(t_{n+1})) \to \bar{X}(t_n)$$

defined by the instructions

$$\begin{aligned}
&\bar{X}(t_n) \mathrel{+}= \bar{X}(t_{n+1}), \\
&\bar{\mu}(t_n) = 0, && \bar{\mu}(t_n) \mathrel{+}= \bar{X}(t_{n+1}) h_n, \\
&\bar{\sigma}(t_n) = 0, && \bar{\sigma}(t_n) \mathrel{+}= \bar{X}(t_{n+1}) \sqrt{h_n} \cdot Z(t_n), \\
&\bar{Z}(t_n) = 0, && \bar{Z}(t_n) \mathrel{+}= \bar{X}(t_{n+1}) \sqrt{h_n} \cdot \sigma(t_n).
\end{aligned}$$

Diffusion Processes and Euler Discretization: Backward Sweep

- And:

Step $\bar{2}$. Compute the Adjoint of the volatility function in Step 2, namely

$$\bar{X}_i(t_n) \mathrel{+}= \sum_{j=1}^{N} \bar{\sigma}_j(t_n) \frac{\partial \sigma_j(t_n)}{\partial X_i}, \qquad \bar{\theta}_k \mathrel{+}= \sum_{j=1}^{N} \bar{\sigma}_j(t_n) \frac{\partial \sigma_j(t_n)}{\partial \theta_k},$$

for $i = 1, \ldots, N$ and $k = 1, \ldots, N_\theta$.

Step $\bar{1}$. Compute the Adjoint of the drift function in Step 1, namely

$$\bar{X}_i(t_n) \mathrel{+}= \sum_{j=1}^{N} \bar{\mu}_j(t_n) \frac{\partial \mu_j(t_n)}{\partial X_i(t_n)}, \qquad \bar{\theta}_k \mathrel{+}= \sum_{j=1}^{N} \bar{\mu}_j(t_n) \frac{\partial \mu_j(t_n)}{\partial \theta_k},$$

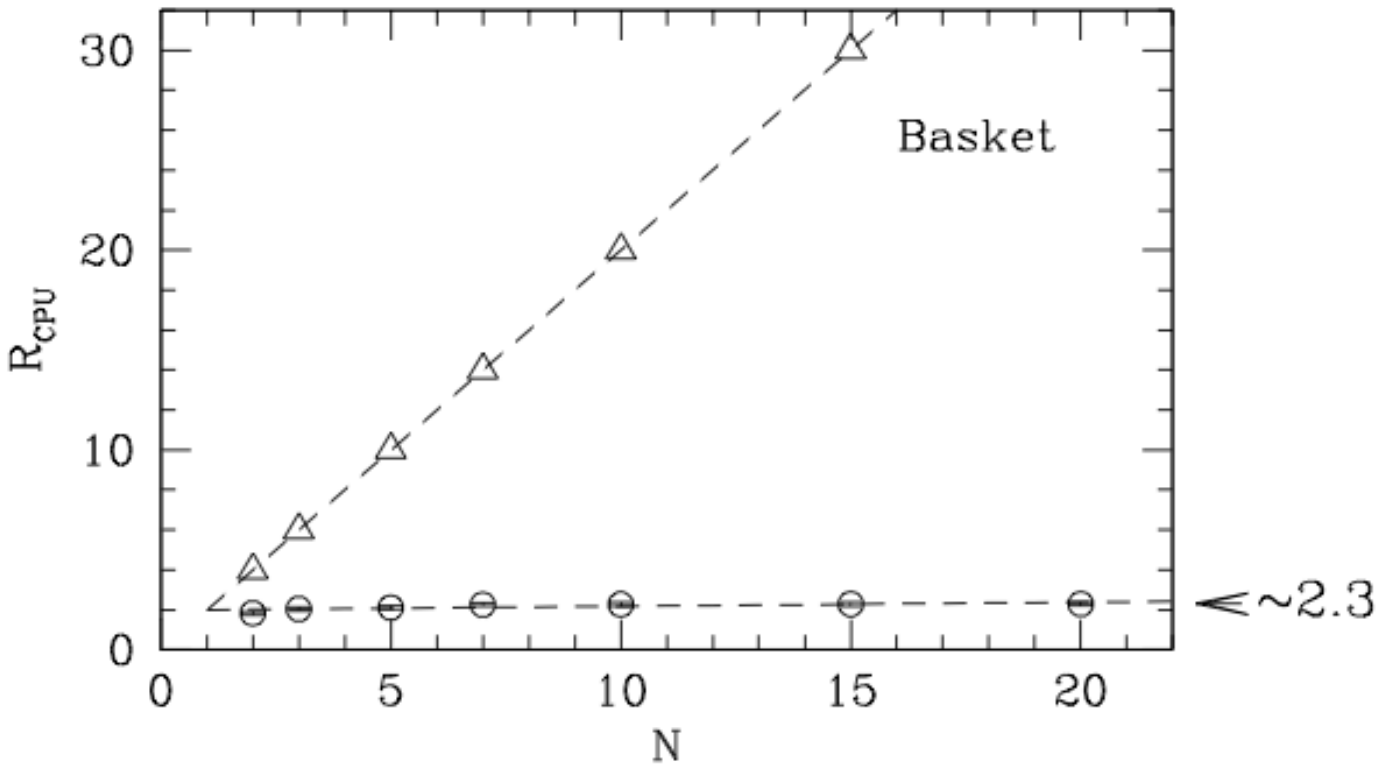
for $i = 1, \ldots, N$ and $k = 1, \ldots, N_\theta$.

Diffusion Processes and Euler Discretization: Backward Sweep

- Note that, the variables $\bar{X}(t_{n+1})$, $\bar{X}(t_n)$ and $\bar{\theta}$ typically contain on input the derivatives of the payout function. During the backward propagation $\bar{X}(t_n)$ (resp. $\bar{\theta}$) accumulate several contributions, one for each Adjoint of an instruction in which $X(t_n)$ (resp. θ) is on the right hand side of the assignment operator in the forward sweep (Steps 1-3).
- The implementation of the Adjoint of the drift and volatility functions in Step $\bar{2}$ and Step $\bar{1}$ is problem dependent. In many cases, the drift and volatility may be represented by computer routines self-contained enough to be processed by means of an automatic differentiation tool, thus facilitating the implementation.

Basket Options: Results

- Let's consider again the Basket Option example introduced earlier for the Payoff.



CPU time ratios for the calculation of Delta and Vega Risk as a function of the number of underlying assets, N: circles (AAD), triangles (Bumping).

Basket Options: Comments

- The performance of the AAD implementation of the Pathwise Derivative Method in this setup is well within the expected bounds.
- In particular, the computation of the $2 \times N$ sensitivities for the N assets requires a very small overhead (of about 130%) with respect to the calculation of the option value itself. This is true for any number of underlying assets.
- This is in stark contrast with the relative cost of evaluating the same sensitivities by means of finite-differences, scaling linearly with the number of assets.
- For typical applications this clearly results in remarkable speedups with respect to bumping.

Section 4

Conclusions

Conclusions

- We have shown how Adjoint Algorithmic Differentiation (AAD) can be used to implement the Adjoint calculation of price sensitivities in a straightforward manner and in complete generality.

- The proposed method allows the calculation of the complete risk at a computational cost which is at most 4 times the cost of calculating the P&L of the portfolio itself, resulting in remarkable computational savings with respect to standard finite differences approaches.
- Although we have covered only a simple MC application, AAD can be applied to Regression Monte Carlo for American Options, as well as Partial Differential Equations and Calibration methods.

References I

[1] P. Glasserman, *Monte Carlo Methods in Financial Engineering*, Springer, New York (2004).

[2] L. Capriotti, *Fast Greeks by Algorithmic Differentiation*, J. of Computational Finance, **14**, 3 (2011).

[3] A. Griewank, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Frontiers in Applied Mathematics, Philadelphia, 2000.

[4] L. Capriotti and M. Giles, *Algorithmic Differentiation: Adjoint Greeks Made Easy*, Risk, Risk **25**, 92 (2012).

[5] L. Capriotti and M. Giles, *Fast Correlation Greeks by Adjoint Algorithmic Differentiation*, Risk **23**, 79 (2010).

[6] L. Capriotti, Jacky Lee, and Matthew Peacock, *Real Time Counterparty Credit Risk Management in Monte Carlo*, Risk **24**, 86 (2011).

[7] L. Capriotti and J. Lee, *Adjoint Credit Risk Management*, Risk Magazine, **27**, 90 (2014).

[8] L. Capriotti, Yupeng Jiang and Andrea Macrina, *Real-Time Risk Management: An AAD-PDE Approach*, Int. J. of Fin. Engineering, **2** 1550039 (2015).

[9] L. Capriotti, Yupeng Jiang and Andrea Macrina, *AAD and least-square Monte Carlo: Fast Bermudan-style options and XVA Greeks*, Algorithmic Finance, **6** 35 (2017).

References II

See also:

- My Publications' Page

Disclaimer

This material has been prepared by the Quantitative Strategy Group ("Quant Strategists") which is part of the institutional trading desk of Credit Suisse and/or its affiliates (collectively "CS"). Quant Strategists are not part of the Research Department and the written materials disseminated by Quant Strategists are not research reports. The views of CS' Quant Strategists may differ materially from the views of the Research Department and other divisions at CS. CS has a number of policies in place designed to ensure the independence of CS' Research Department from CS' Trading Desks, including policies relating to the front-running of research reports. These policies do not apply to the materials provided by Quant Strategists. This material may have previously been communicated to the CS trading desk or other CS clients. You should assume that the CS' trading desk makes markets and/or currently maintains positions in the securities (or related securities) or derivatives thereof that are the subject of these materials, and such positions may be inconsistent with the materials provided by Quant Strategists.

This material is being provided to you solely for information purposes, is intended for your use and does not constitute an offer or commitment, a solicitation of an offer or commitment, or any advice or recommendation to enter into a transaction (whether on the indicative terms or any other). This has been prepared based on assumptions and parameters determined in good faith. Calculations in these materials may reflect CS's theoretical (i.e., model) prices only and may not take into account other potentially significant factors. It is important that you (recipient) understand that those assumptions and parameters are not the only ones that might have reasonably been selected or that could apply in the preparation of these materials or in an assessment of any transaction. A variety of other assumptions or parameters, or other market factors or conditions, could result in different contemporaneous good faith analysis or assessment of a transaction. Past performance should not be taken as an indication or guarantee of future performance, and no warranty or representation, expressed or implied is made regarding future performance. Opinions and estimates may be changed without notice. The information set forth above has been obtained from or based upon sources believed by CS to be reliable, but CS does not represent or warrant its accuracy or completeness. This material does not purport to contain all of the information that an interested party may desire. In all cases, interested parties should conduct their own investigation and analysis of the data set forth in these materials. Each person receiving these materials should make an independent assessment of the merits of pursuing a transaction described in these materials and should consult their own professional advisors. CS may from time to time, participate or invest in other financing transactions with the issuers of securities referred to herein, perform services for or solicit business from such issuers, and/or have a position or effect transactions in the securities of derivatives thereof. The market value of any security may be affected by changes in economic, financial, and political factors (including, but not limited to, spot and forward interest and exchange rates), time to maturity, market conditions and volatility and the credit quality of any issuer or reference issuer. Any investor interested in purchasing a product should conduct its own investigation and analysis of the product and consult with its own professional advisors as to the risk involved in making such a purchase.

CS is not qualified to give tax or accounting advice. This document is not to be relied upon in substitution for the exercise of independent judgment and for consultation of an external tax or accounting advisor. CS may have issued other documents that are inconsistent with, and reach different conclusions from, the information presented in this document. Those documents may reflect different assumptions, views and analytical methods of the analysts who prepared them. This document may not be reproduced in whole or in part or made available without the written consent of CS. The distribution of this information may be restricted by local law or regulation in certain jurisdictions.

CS may provide various services to US municipal entities or obligated persons ("municipalities"), including suggesting individual transactions or trades and entering into such transactions. Any services CS provides to municipalities are not viewed as "advice" within the meaning of Section 975 of the Dodd-Frank Wall Street Reform and Consumer Protection Act. CS is providing any such services and related information solely on an arm's length basis and not as an advisor or fiduciary to the municipality. In connection with the provision of the any such services, there is no agreement, direct or indirect, between any municipality (including the officials, management, employees or agents thereof) and CS for CS to provide advice to the municipality. Municipalities should consult with their financial, accounting and legal advisors regarding any such services provided by CS. In addition, CS is not acting for direct or indirect compensation to solicit the municipality on behalf of an unaffiliated broker, dealer, municipal securities dealer, municipal advisor, or investment adviser for the purpose of obtaining or retaining an engagement by the municipality for or in connection with Municipal Financial Products, the issuance of municipal securities, or of an investment adviser to provide investment advisory services to or on behalf of the municipality.