Algorithmic Differentiation Cheat Sheet

Saïd Business School, University of Oxford, UK

Nicholas Burgess

nburgessx@gmail.com

May 2022

Abstract

Algorithmic Differentiation (AD), also known as automatic differentiation, computes the derivative(s) of computer code. It was pioneered by (Giles and Glasserman, 2006) and produces exact derivatives with low latency. AD is well presented in finance, see (Capriotti, 2010), (NAG, n.d.) and (Savine, 2018), where it can be used to compute financial risks such as Swap DV01, see (Burgess, 2022).

In this paper we summarize (Burgess, 2022) and provide a cheat sheet of how to perform AD by hand. Firstly we present algorithmic differentiation and how to compute the exact derivative(s) of computer code to machine precision. Secondly we give a summary of AD and illustrate how to perform AD by hand using tangent and adjoint calculation modes. Throughout we gave examples in C++ code, which are available for download.

Key words: Algorithmic Differentiation, Chain-Rule, Tangent Mode, Forwards, Adjoint Mode, Backwards, Reverse, Accuracy, Machine Precision, Low Latency, By Hand, Finance, Risks, Sensitivities

1. Algorithmic Differentiation Explained

Algorithmic Differentiation (AD), also known as automatic differentiation, is a mathematical, computer science technique for computing accurate sensitivities quickly. There are two main modes, namely tangent mode and adjoint mode, see (Burgess, 2022), (Capriotti, 2010). For many models, adjoint AD (AAD) can compute sensitivities 10s, 100s or even 1000s of times faster than numerical bumping and finite differences (NAG, n.d.). AD operates directly on analytics and each line of code is differentiated. AD can be applied to a single trade or a portfolio (or vector) of trades.

If for example we have a computer algorithm that computes y via multiple nested operations such as $y = h(g(f(x)))$ we could illustrate the series of operations as,

$$x \to f(x) \to g(\,f(x)\,) \to h(\,g(\,f(x)\,)\,) \to y \tag{1}$$

Working **forwards** from the input value x, we can compute the derivative of each operation and use the **'chain rule'** to compute the total derivative dy/dx as follows.

$$\frac{df}{dx}\cdot\frac{dg}{df}\cdot\frac{dh}{dg}\cdot\frac{dy}{dh}=\frac{\boldsymbol{dy}}{\boldsymbol{dx}} \tag{2}$$

We could also work **backwards** from the output value y to arrive at the same result,

$$\frac{dy}{dh}\cdot\frac{dh}{dg}\cdot\frac{dg}{df}\cdot\frac{df}{dx}=\frac{\boldsymbol{dy}}{\boldsymbol{dx}} \tag{3}$$

Furthermore AD can be used on systems of equations and matrices. Tangent mode works forward from the left and performs **matrix-matrix** multiplication followed by a final matrix-vector product,

$$\left(\left(\left(\frac{\partial x_1}{\partial y}\frac{\partial x_2}{\partial x_1}\right)\frac{\partial x_3}{\partial x_2}\right)\cdots\frac{\partial x_m}{\partial x_{m-1}}\right)\frac{\partial y}{\partial x_m} \tag{4}$$

With adjoint mode we work backwards from the right, however now everything is **matrix-vector** products, which is much faster.

$$\frac{\partial x_1}{\partial x}\left(\frac{\partial x_2}{\partial x_1}\left(\frac{\partial x_3}{\partial x_2}\cdots\left(\frac{\partial x_m}{\partial x_{m-1}}\frac{\partial y}{\partial x_m}\right)\right)\right) \tag{5}$$

1.1 Tangent Mode

In tangent mode we differentiate code working forwards starting with the trade inputs and follow the natural order of the original program. This method computes price sensitivities to one input at a time and we must call the tangent method several times, once for each input parameter.

Dot Notation:

When using tangent mode **'dot'** notation is used to denote derivatives being differentiated with respect to the function input. For example given y = f(x) then y dot would indicate $\dot{\boldsymbol{y}} = \boldsymbol{dy}/\boldsymbol{dx}$.

Consider the below simple function,

$$\text{Function:} \qquad y = 2x^2 \tag{6}$$

$$\text{Tangent:} \qquad \dot{y} = 4x.\dot{x} \tag{7}$$

In tangent mode $\dot{\boldsymbol{x}} = \boldsymbol{dx/dx}$ is specified as an input and used to enable/disable the tangent risk calculation. Setting $\dot{\boldsymbol{x}} = \mathbf{1}$ in (equation 7) above enables the derivative calculation giving dy/dx = 4x, however when $\dot{\boldsymbol{x}} = \mathbf{0}$ we have dy/dx = 0.

1.2 Tangent Mode Example

Next let us consider how to apply tangent AD code to a simple function comprising of a series of simple incremental operations.

$$\text{Function:} \qquad f(x_1, x_2) = 2x_1^2 + 3x_2 \tag{8}$$

$$\text{Solution:} \qquad \frac{df}{dx_1} = 4x_1 \text{ and } \frac{df}{dx_2} = 3 \tag{9}$$

When $x_1 = 2$ and $x_2 = 3$ we have,

$$\frac{df}{dx_1} = 8 \text{ and } \frac{df}{dx_2} = 3 \tag{10}$$

Let's write this function in C++ code[1] and implement (equation 8) as a series of operations spanning multiple lines of C++ code.

```
double function( double x1, double x2 )
{
        double a = x1*x1;        // Step 1:    a = x1^2
        double b = 2*a;          // Step 2:    b = 2x1^2
        double c = x2;           // Step 3:    c = x2
        double d = 3*c;          // Step 4:    d = 3x2
        double f = b + d;        // Step 5:    f = 2x1^2 + 3x2
        return f;
}
```

Code 1: **Simple Function: $\boldsymbol{f(x_1, x_x) = 2x_1^2 + 3x_2}$**

Source code: AD-Simple-Function.cpp
Available at: https://bit.ly/SwapCodeAAD
To run see: https://onlinegdb.com/kKqaS6hJT

[1] To run these examples in C++ perhaps use a free online web compiler such as https://www.onlinegdb.com/.

Now let’s add tangent AD code to this function, working forwards using dot notation.

```
01  double tangent( double x1, double x2, double x1_dot, double x2_dot )
02  {
03      double a = x1*x1;                  // Step 1:     a = x_1^2
04      double a_dot = 2*x1*x1_dot;        // Tangent:    \dot{a} = 2x_1 · \dot{x}_1        \dot{a} = 2x_1
05      double b = 2*a;                    // Step 2:     b = a
06      double b_dot = 2*a_dot;            // Tangent:    \dot{b} = 2 · \dot{a}             \dot{b} = 4x_1
07      double c = x2;                     // Step 3:     c = x_2
08      double c_dot = x2_dot;             // Tangent:    \dot{c} = \dot{x}_2               \dot{c} = 1
09      double d = 3*c;                    // Step 4:     d = 3c
10      double d_dot = 3*c_dot;            // Tangent:    \dot{d} = 3 · \dot{c}             \dot{d} = 3
11      double f = b + d;                  // Step 5:     f = 2x_1^2 + 3x_2
12      double f_dot = b_dot + d_dot;      // Tangent:    \dot{f} = \dot{b} + \dot{d}
13      return f_dot;                      // Result:     \dot{f} = 4x_1 + 3
14  }
```

Code 2: **Simple Function $f(x_1, x_x) = 2x_1^2 + 3x_2$ with Tangent Risk**

Source code: AD-Simple-Function.cpp
Available at: https://bit.ly/SwapCodeAAD
To run see: https://onlinegdb.com/kKqaS6hJT

Note the function f(x1, x2) takes two inputs, but we only have one f_dot risk output on (line 13). This means in tangent mode we can only get one risk output at a time. So to get the risk to each input we would have to call the tangent method several times, once per input variable as follows,

```
tangent(2.0, 3.0, 1.0, 0.0);    // Input: x1 = 2, x2 = 3, x1_d = 1, x2_d = 0     Output: 8
tangent(2.0, 3.0, 0.0, 1.0);    // Input: x1 = 2, x2 = 3, x1_d = 0, x2_d = 1     Output: 3
```

Code 3: **Function Derivatives using Tangent Mode**

As the function $f(x_1, x_2) = 2x_1^2 + 3x_2$ with derivatives $df/dx_1 = 4x_1$ and $df/dx_2 = 3$ as per (equations 8 and 9) then (code 3) returns output values of 8 and 3 as outlined in (equation 10).

1.3 Adjoint Mode

When using adjoint mode we differentiate code in reverse order, starting with function outputs. Adjoint mode follows the reverse order of the original program, consequently we must compute the function value first (**‘forward sweep’**) and store the intermediate values before applying adjoint AD in reverse (**‘back propagation’**). This method shifts one function **output** at a time and generates derivatives exactly to machine precision for all price inputs in one go.

Bar Notation:

Adjoint mode is computed using **'bar'** notation for derivatives to denote the variable is to be differentiated with respect to the function input. For example given y = f(x) and working **in reverse** order gives $\overline{\boldsymbol{x}} = \boldsymbol{dy/dx}$.

Once again let us consider same simple function from equation 8),

$$\text{Function:} \qquad y = 2x^2 \tag{11}$$

$$\text{Adjoint:} \qquad \bar{x} = 4x.\bar{y} \tag{12}$$

The adjoint bar notation in (equation 17) is equivalent to,

$$\text{Adjoint:} \qquad \frac{dy}{dx} = 4x.\frac{\boldsymbol{dy}}{\boldsymbol{dy}} \tag{13}$$

In adjoint mode $\overline{\boldsymbol{y}} = \boldsymbol{dy/dy}$ is specified as an input and allows us to enable/disable the adjoint risk calculation. Setting $\overline{\boldsymbol{y}} = \mathbf{1}$ in (equation 12) enables the risk calculation giving dy/dx = 4x and when $\overline{\boldsymbol{y}} = \mathbf{0}$ we have dy/dx = 0.

1.4 Adjoint Mode Example

Next let us consider how to add to add adjoint AD code to the same simple function from (8), requoted below for convenience.

$$\text{Function:} \qquad f(x_1, x_2) = 2x_1^2 + 3x_2 \tag{14}$$

$$\text{Solution:} \qquad \frac{df}{dx_1} = 4x_1 \text{ and } \frac{df}{dx_2} = 3 \tag{15}$$

When $x_1 = 2$ and $x_2 = 3$ we have,

$$\frac{df}{dx_1} = 8 \text{ and } \frac{df}{dx_2} = 3 \tag{16}$$

This function was implemented in (code 1) above let's apply adjoint AD (AAD) to this method, remembering that we are working backwards and in reverse. Consequently we must perform a forward sweep to evaluate the underlying function and store intermediate values for back-propagation and reverse calculation of risk as follows,

```
void adjoint( double x1, double x2, double f_bar )
{
```

```
03      // Forward Sweep
04      double a = x1*x1;                   // Step 1:    a = x1^2
05      double b = 2*a;                     // Step 2:    b = 2x1^2
06      double c = x2;                      // Step 3:    c = x2
07      double d = 3*c;                     // Step 4:    d = 3x2
10      double f = b + d;                   // Step 5:    f = 2x1^2 + 3x2
08
09      // Back Propagation
10      double b_bar = f_bar;               // Step 5:    b_bar  = 1     from input variable
11      double d_bar = f_bar;               // Step 5:    d_bar  = 1     from input variable
12      double c_bar = 3*d_bar;             // Step 4:    c_bar  = 3
13      double x2_bar = c_bar;              // Step 3:    x2_bar = 3     df/dx2 = 3
14      double a_bar = 2*b_bar;             // Step 2:    a_bar  = 2
15      double x1_bar = 2*x1*a_bar;         // Step 1:    x1_bar = 4x1   df/dx1 = 4x1
16
17      // Display Results
18      std::cout << "df/dx1: " << x1_bar << std::endl;      // x̄1 = df/dx1 = 4x1
19      std::cout << "df/dx2: " << x2_bar << std::endl;      // x̄2 = df/dx2 = 3
20  }
```

Code 4: **Simple Function $f(x_1, x_x) = 2x_1^2 + 3x_2$ with Adjoint Risk**

Source code: AD-Simple-Function.cpp
Available at: https://bit.ly/SwapCodeAAD
To run see: https://onlinegdb.com/kKqaS6hJT

Note the function f(x1, x2) takes two inputs and in adjoint mode we capture both the risk to x1 and x2, namely x1_bar (line 13) and x2_bar (line 15). This means that we capture a functions risk to all inputs in one go and only need to call the adjoint method once as follows,

```
adjoint(2.0, 3.0, 1.0);     // Input: x1 = 3, x2 = 2, f_bar     Output: df⁄(dx_1 )= 8 and df⁄(dx_2 ) = 3
```

Code 5: **Function Risk using Adjoint Mode**

As the function $f(x_1, x_2) = 2x_1^2 + 3x_2$ with derivatives df/dx_1 = 4x1 and df/dx_2 = 3 as per (equations 11 and 12) then (code 5) returns output values of 8 and 3 as per (equation 13).

Conclusion

We presented algorithmic differentiation and how to computes the exact derivative(s) of computer code to machine precision. We gave a summary of AD and using a simple example illustrated how to perform AD by hand using tangent and adjoint calculation modes. Throughout we gave examples in C++ code, which are available for download.

References

(Burgess, N., 2022) SSRN Working Paper: Algorithmic Adjoint Differentiation (AAD) for Swap Pricing and DV01 Risk. Available at: https://bit.ly/SwapAAD with source code https://bit.ly/SwapCodeAAD

(Capriotti, L., 2010) Fast Greeks by Algorithmic Differentiation
Available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1619626

(Giles M. and Glasserman P., 2006) Smoking Adjoints: Fast Monte Carlo Greeks.
Available at: RISK Magazine, January 2006

(NAG, n.d.) Numerical Algorithms Group, Automatic Differentiation Solutions
Available at: https://www.nag.com/content/nagr-automatic-differentiation-solutions

(Savine A., 2018) Textbook: Modern Computational Finance: AAD and Parallel Simulations,
ISBN 978-1119539452. Available at:
https://www.amazon.co.uk/Modern-Computational-Finance-Parallel-Simulations/dp/1119539455