



A combined automatic differentiation and array library for C++

[About](#) [Automatic differentiation](#) [Array features](#) [Optimization](#) [Documentation](#) [Download](#) [Licence](#)

What is Adept?

Adept (Automatic Differentiation using Expression Templates) is a free C++ software library that enables algorithms to be [automatically differentiated](#), very useful for a wide range of applications that involve [mathematical optimization](#). It uses an operator overloading approach, so very little code modification is required. Moreover, the way that [expression templates](#) have been used and several other important optimizations mean that reverse-mode differentiation is significantly faster than other C++ libraries that provide equivalent functionality. [Learn more...](#)

The fundamental object of most mathematical software is the *array*, and it is therefore natural to want to differentiate array expressions. There are numerous mature C++ array libraries that support a wide variety of array operations, but because they also use expression templates to optimize array expressions, they do not work efficiently with Adept. The solution provided in version 2 of Adept is to use a single expression-template framework that incorporates both array and automatic-differentiation capabilities at a low level, thus enabling efficient differentiation of array expressions. Arrays of up to 7 dimensions are supported along with matrix multiplication and basic linear algebra operations. Of course, the library also works well with passive arrays (those that do not need to be differentiated). [Learn more...](#)

News

- February 2021: [Adept 2.1](#) has been released, including various [bounded and unbounded minimization algorithms](#).

Example

Here is an example program that demonstrates both the automatic differentiation and the array capabilities of Adept:

```
#include <iostream>
#include <adept_arrays.h>           // include adept.h if array features not required
int main(int argc, const char** argv) {
    using namespace adept;
    Stack stack;                   // Object to store differential statements
    aVector x(3);                  // Independent variables: active vector with 3 elements
    x << 1.0, 2.0, 3.0;            // Fill vector x
    stack.new_recording();          // Clear any existing differential statements
    aReal J = cbrt(sum(abs(x*x*x))); // Compute dependent variable: L3-norm in this case
    J.set_gradient(1.0);           // Seed the dependent variable
    stack.reverse();               // Reverse-mode differentiation
    std::cout << "dJ/dx = "
                << x.get_gradient() << "\n"; // Print the vector of partial derivatives dJ/dx
    return 0;
}
```

When compiled and executed, this program reports the derivative as:

```
dJ/dx = {0.0917202, 0.366881, 0.825482}
```

The line defining *J* can of course be replaced by a much more complicated algorithm with function calls and so on.

Return to [Clouds Group](#) | [Department of Meteorology](#) | [University of Reading](#)

This page is maintained by [Robin Hogan](#)