Automatic Differentiation

Jason Boisvert

Outline

1. Introduction
2. Modes of automatic differentiation
3. Implementation of AD
4. Popular AD packages

Outline

1. Introduction
2. Modes of automatic differentiation
3. Implementation of AD
4. Popular AD packages

Outline

1. Introduction
2. Modes of automatic differentiation
3. Implementation of AD
4. Popular AD packages

Outline

1 Introduction

2 Modes of automatic differentiation

3 Implementation of AD

4 Popular AD packages

Differentiation in numerical software

A numerical algorithm may require

- derivatives
- gradients
- Jacobians
- Hessian matrices
- Taylor polynomials

Differentiation in numerical software

A numerical algorithm may require

- derivatives
- gradients
- Jacobians
- Hessian matrices
- Taylor polynomials

Differentiation in numerical software

A numerical algorithm may require

- derivatives
- gradients
- Jacobians
- Hessian matrices
- Taylor polynomials

Differentiation in numerical software

A numerical algorithm may require

- derivatives
- gradients
- Jacobians
- Hessian matrices
- Taylor polynomials

Differentiation in numerical software

A numerical algorithm may require

- derivatives
- gradients
- Jacobians
- Hessian matrices
- Taylor polynomials

Differentiation in numerical software

A numerical algorithm may require

- derivatives
- gradients
- Jacobians
- Hessian matrices
- Taylor polynomials

Methods to compute derivatives

Functions can be differentiated in several ways

- finite differences
- symbolic differentiation
- automatic differentiation

Methods to compute derivatives

Functions can be differentiated in several ways

- finite differences
- symbolic differentiation
- automatic differentiation

Methods to compute derivatives

Functions can be differentiated in several ways

- finite differences
- symbolic differentiation
- automatic differentiation

Methods to compute derivatives

Functions can be differentiated in several ways

- finite differences
- symbolic differentiation
- automatic differentiation

Automatic differentiation

What is automatic differentiation?

Automatic differentiation is a set of techniques to numerically differentiate a function through the use of exact formulas and floating-point values. The resulting numerical value involves no approximation error.

Other names for automatic differentiation

- computational differentiation
- algorithmic differentiation

Other names for automatic differentiation

- computational differentiation
- algorithmic differentiation

Several modes of automatic differentiation

- forward automatic differentiation
- reverse automatic differentiation
- a combination of forward and reverse modes of automatic differentiation

Several modes of automatic differentiation

- forward automatic differentiation
- reverse automatic differentiation
- a combination of forward and reverse modes of automatic differentiation

Several modes of automatic differentiation

- forward automatic differentiation
- reverse automatic differentiation
- a combination of forward and reverse modes of automatic differentiation

Forward mode

Forward automatic differentiation divides the expression into a sequence of differentiable elementary operations.

The chain rule and well-known differentiation rules are then applied to each elementary operation.

Using forward automatic differentiation on an example

For example, forward automatic differentiation can be used differentiate the expression

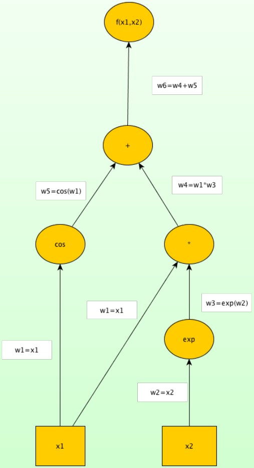
$$f(x_1, x_2) = \cos(x_1) + x_1 \exp(x_2).$$

Elementary operations

The expression can be divided into the following elementary operations

$$
\begin{aligned}
w_1 &= x_1, \\
w_2 &= x_2, \\
w_3 &= \exp(w_2), \\
w_4 &= w_1 w_3, \\
w_5 &= \cos(w_1), \\
w_6 &= w_4 + w_5, \\
f(x_1, x_2) &= w_6.
\end{aligned}
$$

A computational graph



Finding the derivatives of the example

We can find the numerical value of the derivatives to $f(x_1, x_2)$ by using some differentiation rules and applying the chain rule to each of the elementary operations.

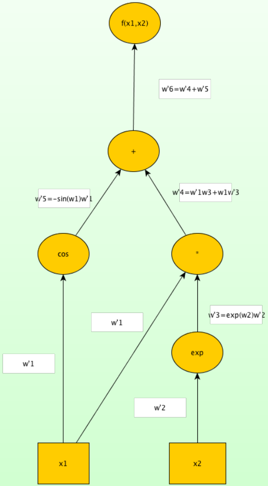
$$
\begin{aligned}
w_1' &= \texttt{seed} \in \{0, 1\} \\
w_2' &= \texttt{seed} \in \{0, 1\} \\
w_3' &= \exp(w_2) w_2' \\
w_4' &= w_1' w_3 + w_1 w_3' \\
w_5' &= -\sin(w_1) w_1' \\
w_6' &= w_4' + w_5' \\
f'(x_1, x_2) &= w_6'
\end{aligned}
$$

Seed values

In order to find a value for the partial derivative of $f'(x_1, x_2)$ with respect to x_1, we set

$$w_1' = 1,$$
$$w_2' = 0.$$

A computational graph of the forward mode



Notes on forward automatic differentiation

- Only one sweep is necessary to compute derivatives for functions $\mathbf{f} : \mathbb{R} \to \mathbb{R}^m$.
  Therefore, this mode is efficient for $m \gg 1$.
- It is not necessary to save values for intermediate expressions.
- Implementation is simple due to how expressions are normally evaluated by computers.

Notes on forward automatic differentiation

- Only one sweep is necessary to compute derivatives for functions $\mathbf{f} : \mathbb{R} \to \mathbb{R}^m$.
  Therefore, this mode is efficient for $m \gg 1$.
- It is not necessary to save values for intermediate expressions.
- Implementation is simple due to how expressions are normally evaluated by computers.

Notes on forward automatic differentiation

- Only one sweep is necessary to compute derivatives for functions $\mathbf{f} : \mathbb{R} \to \mathbb{R}^m$.
  Therefore, this mode is efficient for $m \gg 1$.
- It is not necessary to save values for intermediate expressions.
- Implementation is simple due to how expressions are normally evaluated by computers.

Reverse mode

The reverse mode computes a series of adjoints

$$\bar{w}_i = \sum_{j \in \pi(i)} \bar{w}_j \frac{\partial w_j}{\partial w_i}, \quad i = N, N-1, \ldots, 1,$$

where the $\pi(i)$ are the indices of elementary operations that immediately proceed the elementary operation w_i in the computational process.

An initial value of

$$\bar{w}_N = 1$$

is used.

Reverse mode

The partial derivatives for a function $f : \mathbb{R}^n \to \mathbb{R}$ can be obtained from

$$\frac{\partial f}{\partial x_i} = \bar{w}_i.$$

Applying reverse mode to example

Recall that $f(x_1, x_2) = \cos(x_1) + x_1 \exp(x_2)$ can be broken down into the following elementary operations

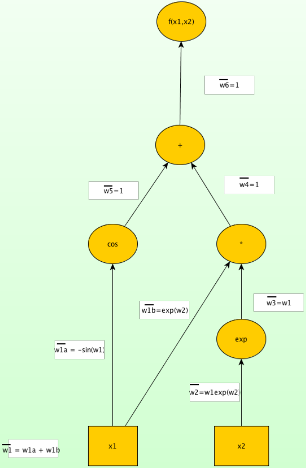
$$
\begin{aligned}
w_1 &= x_1, \\
w_2 &= x_2, \\
w_3 &= \exp(w_2), \\
w_4 &= w_1 w_3, \\
w_5 &= \cos(w_1), \\
w_6 &= w_4 + w_5.
\end{aligned}
$$

Computing the adjoints

The adjoints for this example are

$$\begin{aligned}
\bar{w}_6 &= 1, \\
\bar{w}_5 &= 1, \\
\bar{w}_4 &= 1, \\
\bar{w}_3 &= w_1, \\
\bar{w}_2 &= w_1 \exp(w_2), \\
\bar{w}_1 &= -sin(w_1) + \exp(w_2).
\end{aligned}$$

A computational graph of the reverse mode



Notes on reverse automatic differentiation

- Only one sweep is required to compute all partial derivatives for $f : \mathbb{R}^n \to \mathbb{R}$.
  Therefore, this mode is efficient for $n \gg 1$.
- Adjoints must be saved in order to compute partial derivatives.
- A forward sweep through the expression must be performed before a reverse sweep used to compute adjoints is performed.

Notes on reverse automatic differentiation

- Only one sweep is required to compute all partial derivatives for $f : \mathbb{R}^n \to \mathbb{R}$.
  Therefore, this mode is efficient for $n \gg 1$.
- Adjoints must be saved in order to compute partial derivatives.
- A forward sweep through the expression must be performed before a reverse sweep used to compute adjoints is performed.

Notes on reverse automatic differentiation

- Only one sweep is required to compute all partial derivatives for $f : \mathbb{R}^n \to \mathbb{R}$.
  Therefore, this mode is efficient for $n \gg 1$.
- Adjoints must be saved in order to compute partial derivatives.
- A forward sweep through the expression must be performed before a reverse sweep used to compute adjoints is performed.

A combination of modes

Forward and reverse modes may be combined to more efficiently compute a Jacobian matrix for a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ depending on the sparsity structure of the matrix.

- For example, n sweeps of the forward mode will compute the columns of the Jacobian matrix ...
- whereas m sweeps of the reverse mode will compute the rows of the Jacobian matrix.
- Depending on the Jacobian matrix, a combination of the forward and reverse modes may require fewer sweeps than either mode alone.

A combination of modes

Forward and reverse modes may be combined to more efficiently compute a Jacobian matrix for a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ depending on the sparsity structure of the matrix.

- For example, n sweeps of the forward mode will compute the columns of the Jacobian matrix ...
- whereas m sweeps of the reverse mode will compute the rows of the Jacobian matrix.
- Depending on the Jacobian matrix, a combination of the forward and reverse modes may require fewer sweeps than either mode alone.

A combination of modes

Forward and reverse modes may be combined to more efficiently compute a Jacobian matrix for a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ depending on the sparsity structure of the matrix.

- For example, n sweeps of the forward mode will compute the columns of the Jacobian matrix . . .
- whereas m sweeps of the reverse mode will compute the rows of the Jacobian matrix.
- Depending on the Jacobian matrix, a combination of the forward and reverse modes may require fewer sweeps than either mode alone.

Methods of implementation

- Source-to-source transformation
- Operator overloading

Methods of implementation

- Source-to-source transformation
- Operator overloading

Introduction to source-to-source transformation

Uses a compiler to convert source code with mathematical expressions to source code with automatic differentiation expressions.

Notes on source-to-source transformation

- Requires minimal (if any) changes to the original code.
- The resulting automatic differentiation code can be optimized.
- Knowledge of compiler concepts are required, and therefore implementation can be difficult.
- Additional tools must be installed in order for users to apply automatic differentiation to their code.

Notes on source-to-source transformation

- Requires minimal (if any) changes to the original code.
- The resulting automatic differentiation code can be optimized.
- Knowledge of compiler concepts are required, and therefore implementation can be difficult.
- Additional tools must be installed in order for users to apply automatic differentiation to their code.

Notes on source-to-source transformation

- Requires minimal (if any) changes to the original code.
- The resulting automatic differentiation code can be optimized.
- Knowledge of compiler concepts are required, and therefore implementation can be difficult.
- Additional tools must be installed in order for users to apply automatic differentiation to their code.

Notes on source-to-source transformation

- Requires minimal (if any) changes to the original code.
- The resulting automatic differentiation code can be optimized.
- Knowledge of compiler concepts are required, and therefore implementation can be difficult.
- Additional tools must be installed in order for users to apply automatic differentiation to their code.

Introduction to operator overloading

User-defined data types and the operator overloading features of a language are used to implement automatic differentiation.

Notes on operator overloading

- Users must modify their code in order to make use of automatic differentiation data types.
- Operator overloading can slow down runtime, and therefore operator overloading automatic differentiation can be slow when compared to source-to-source transformation.
- Implementation of forward mode is much more intuitive than source-to-source transformation. However, implementation of reverse mode can be difficult.
- Other than the language libraries, no additional tools are required for the application of automatic differentiation.

Notes on operator overloading

- Users must modify their code in order to make use of automatic differentiation data types.
- Operator overloading can slow down runtime, and therefore operator overloading automatic differentiation can be slow when compared to source-to-source transformation.
- Implementation of forward mode is much more intuitive than source-to-source transformation. However, implementation of reverse mode can be difficult.
- Other than the language libraries, no additional tools are required for the application of automatic differentiation.

Notes on operator overloading

- Users must modify their code in order to make use of automatic differentiation data types.
- Operator overloading can slow down runtime, and therefore operator overloading automatic differentiation can be slow when compared to source-to-source transformation.
- Implementation of forward mode is much more intuitive than source-to-source transformation. However, implementation of reverse mode can be difficult.
- Other than the language libraries, no additional tools are required for the application of automatic differentiation.

Notes on operator overloading

- Users must modify their code in order to make use of automatic differentiation data types.
- Operator overloading can slow down runtime, and therefore operator overloading automatic differentiation can be slow when compared to source-to-source transformation.
- Implementation of forward mode is much more intuitive than source-to-source transformation. However, implementation of reverse mode can be difficult.
- Other than the language libraries, no additional tools are required for the application of automatic differentiation.

Some source-to-source AD packages

- `openAD`
- `ADiMat`

Some source-to-source AD packages

- `openAD`
- `ADiMat`

openAD

A source-to-source transformation automatic differentiation tool for the `Fortran 90` language.

Notes on openAD

- Allows for both forward and reverse automatic differentiation.
- Implemented with an open-source `Open64` compiler toolset.
- Used in software that models ocean circulation.

Notes on openAD

- Allows for both forward and reverse automatic differentiation.
- Implemented with an open-source Open64 compiler toolset.
- Used in software that models ocean circulation.

Notes on `openAD`

- Allows for both forward and reverse automatic differentiation.
- Implemented with an open-source `Open64` compiler toolset.
- Used in software that models ocean circulation.

ADiMat

A source-to-source transformation automatic differentiation tool for Matlab.

Notes on `ADiMat`

- Supports first- and second-order automatic differentiation with forward mode.
- Supports first-order automatic differentiation with reverse mode.
- Performs code optimization.

Notes on ADiMat

- Supports first- and second-order automatic differentiation with forward mode.
- Supports first-order automatic differentiation with reverse mode.
- Performs code optimization.

Notes on `ADiMat`

- Supports first- and second-order automatic differentiation with forward mode.
- Supports first-order automatic differentiation with reverse mode.
- Performs code optimization.

Some operator overloading AD packages

- `MAD`
- `ADOLC`

Some operator overloading AD packages

- `MAD`
- `ADOLC`

MAD

An operator-overloading automatic differentiation tool for `Matlab`.

Notes on `ADiMat`

- Implements a class called `fmad` that supports operator overloading.
- Performance has been optimized for `Matlab`.
- Detects sparsity patterns of matrices for faster derivative calculation at runtime.
- Used along with `bvp4c` to solve BVODEs.

Notes on ADiMat

- Implements a class called `fmad` that supports operator overloading.
- Performance has been optimized for `Matlab`.
- Detects sparsity patterns of matrices for faster derivative calculation at runtime.
- Used along with `bvp4c` to solve BVODEs.

Notes on `ADiMat`

- Implements a class called `fmad` that supports operator overloading.
- Performance has been optimized for `Matlab`.
- Detects sparsity patterns of matrices for faster derivative calculation at runtime.
- Used along with `bvp4c` to solve BVODEs.

Notes on ADiMat

- Implements a class called `fmad` that supports operator overloading.
- Performance has been optimized for `Matlab`.
- Detects sparsity patterns of matrices for faster derivative calculation at runtime.
- Used along with `bvp4c` to solve BVODEs.

ADOLC

An operator-overloading automatic differentiation tool for C++.

Notes on ADOLC

- Supports both forward and reverse automatic differentiation modes.
- Can be used to compute derivatives of any order.
- When evaluating expressions, it records information, e.g., adjoints, on a tape. The tape can later be used for faster automatic differentiation calculations.
- Can be used to calculate sparsity patterns.
- Can be used to solve certain ordinary differentiation equations.

Notes on ADOLC

- Supports both forward and reverse automatic differentiation modes.
- Can be used to compute derivatives of any order.
- When evaluating expressions, it records information, e.g., adjoints, on a tape. The tape can later be used for faster automatic differentiation calculations.
- Can be used to calculate sparsity patterns.
- Can be used to solve certain ordinary differentiation equations.

Notes on ADOLC

- Supports both forward and reverse automatic differentiation modes.
- Can be used to compute derivatives of any order.
- When evaluating expressions, it records information, e.g., adjoints, on a tape. The tape can later be used for faster automatic differentiation calculations.
- Can be used to calculate sparsity patterns.
- Can be used to solve certain ordinary differentiation equations.

Notes on ADOLC

- Supports both forward and reverse automatic differentiation modes.
- Can be used to compute derivatives of any order.
- When evaluating expressions, it records information, e.g., adjoints, on a tape. The tape can later be used for faster automatic differentiation calculations.
- Can be used to calculate sparsity patterns.
- Can be used to solve certain ordinary differentiation equations.

Notes on ADOLC

- Supports both forward and reverse automatic differentiation modes.
- Can be used to compute derivatives of any order.
- When evaluating expressions, it records information, e.g., adjoints, on a tape. The tape can later be used for faster automatic differentiation calculations.
- Can be used to calculate sparsity patterns.
- Can be used to solve certain ordinary differentiation equations.