SPECIALIZATION PROJECT IN MATHEMATICAL SCIENCES

Title

Author:
SINDRE GRØSTAD

Supervisors:
Professor KNUT-ANDREAS LIE

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING
DEPARTMENT OF MATHEMATICAL SCIENCES

November 19, 2018

Preface

Abstract

Table of Contents

Chapter 1

Introduction

Intro

Chapter 2

Theory

2.1 Automatic differentiation

Automatic differentiation (AD) is a method that makes the computer derive the derivatives with very little effort from the user. If you have not heard of AD before, the first thing that you might think of is algebraic or symbolic differentiation. In this type of differentiation the computer learns the basic rules from calculus like e.g.

$$\frac{d}{dx}x^n = n \cdot x^{n-1}$$
$$\frac{d}{dx}cos(x) = -sin(x)$$
$$\frac{d}{dx}\exp x = \exp x$$

etc. and the chain- and product rule

$$\frac{d}{dx}f(x) \cdot g(x) = f'(x) \cdot g(x) + f(x) \cdot g'(x)$$
$$\frac{d}{dx}f(g(x)) = g'(x) \cdot f'(g(x)).$$

The computer will then use these rules on symbolic variables to obtain the derivative of any function given. This will give perfectly accurate derivatives, but the disadvantage with this approach is that it is computational demanding and as f(x) becomes more complex, the calculations will become slow.

If AD is not symbolic differentiation you might think that it is finite differences, where you use the definition of the derivative

$$\frac{df}{dx} = \frac{f(x+h) - f(x)}{h}$$

with a small h to obtain the numerical approximation of the derivative of f. This approach is not optimal because, first of all, if you choose an h too small, you will get problems with rounding errors on your computer. This is because when h is small, you will subtract two very similar numbers, f(x+h) and f(x) and then divide by a small number h. This means that any small rounding errors in the subtraction will be hugely magnified by the division. Secondly, if you choose h too large your approximation of the derivative will not be accurate.

AD can be split into two different methods - forward AD and backward AD. Both methods are similar to symbolic differentiation in the way that we implement the differentiation rules, but they differ by instead of differentiating symbols and then inserting values for the symbols, we keep track of the function values and the corresponding values of the derivatives as we go. Both methods does this by separating each expression into a finite set of elementary operations.

2.1.1 Forward Automatic Differentiation

In forward AD, the function value is stored in a tuple $[\cdot,\cdot]$. In this way, we can continuously update both the function value and the derivative value for every operation we perform on the function.

As an example, consider the function $f = f(x)$ with its derivative f_x where x is a scalar variable. Then the AD-variable x is the pair $[x,1]$ and for f we have $[f,\ f_x]$. In the pair $[x,1]$, x is the numerical value of x and $1 = \frac{dx}{dx}$. Similar for $f(x)$ where f is the numerical value of $f(x)$ and f_x the numerical value of $f'(x)$. We then define the arithmetic operators for such pairs such that for functions f and g,

$$\begin{aligned}
&[f\ ,\ f_x] \pm [g\ ,\ g_x] = [f \pm g\ ,\ f_x \pm g_x]\ , \\
&[f\ ,\ f_x] \cdot [g\ ,\ g_x] = [f \cdot g\ ,\ f_x \cdot g + f \cdot g_x], \\
&\frac{[f\ ,\ f_x]}{[g\ ,\ g_x]} = \left[\frac{f}{g}\ ,\ \frac{f_x \cdot g - f \cdot g_x}{g^2}\right].
\end{aligned} \tag{2.1}$$

It is also necessary to define the chain rule such that for a function h(x)

$$h\big(f(x)\big) = h\Big([f\ ,\ f_x]\Big) = [h(f)\ ,\ f_x \cdot h'(f)].$$

The only thing that remains to define are the rules concerning elementary functions like

$$\begin{aligned}
&\exp\Big([f\ ,\ f_x]\Big) = [\exp(f)\ ,\ \exp(f) \cdot f_x], \\
&\log\Big([f\ ,\ f_x]\Big) = \left[\log(f)\ ,\ \frac{f_x}{f}\right], \\
&\sin\Big([f\ ,\ f_x]\Big) = [\sin(f)\ ,\ \sin(f) \cdot f_x], \text{ etc.}
\end{aligned} \tag{2.2}$$

When these arithmetic operators and the elementary function are implemented you are able to calculate any scalar function derivative without actually doing any form of differentiation yourselves. Let us look at an step by step example where

$$f(x) = x \cdot \exp(2x) \qquad \text{for} \quad x = 2. \tag{2.3}$$

Then the declaration of the AD-variable x gives $x = [2\ ,\ 1]$. All scalars can be looked at as AD variables with derivative equal to 0 such that

$$\begin{aligned}
2x &= [2\ ,\ 0] \cdot [2\ ,\ 1] \\
&= [2 \cdot 2\ ,\ 0 \cdot 1 + 2 \cdot 1] \\
&= [4\ ,\ 2].
\end{aligned}$$

After this computation we get from the exponential

$$\begin{aligned}\exp(2x) &= \exp\big([4\ ,\ 2]\big)\\ &= [\exp(4)\ ,\ \exp(4)\cdot 2],\end{aligned}$$

and lastly from the product rule we get the correct tuple for $f(x)$

$$\begin{aligned}x\cdot\exp(2x) &= [2\ ,\ 1]\cdot[\exp(4)\ ,\ 2\cdot\exp(4)]\\ &= [2\cdot\exp(4)\ ,\ 1\cdot\exp(4)+2\cdot 2\exp(4)]\\ [f\ ,\ f_x] &= [2\cdot\exp(4)\ ,\ 5\cdot\exp(4)].\end{aligned}$$

This result is equal

$$\big(f(x)\ ,\ f_x(x)\big) = \big(x\cdot\exp(2x)\ ,\ (1+2x)\exp(2x)\big)$$

for $x = 2$.

2.1.2 Dual Numbers

One approach to implementing forward AD is by dual numbers. Similar to complex numbers dual numbers are defined as

$$a + b\epsilon. \tag{2.4}$$

Here a and b are scalars and corresponds to the function value and the derivative value. ϵ is like we have for complex numbers $i^2 = -1$, but the corresponding relation for dual numbers are $\epsilon^2 = 0$. The convenient part of implementing forward AD with dual numbers is that you get the differentiation rules for arithmetic operations for free. Consider the dual numbers x and y on the form of definition (2.4). Then we get for addition

$$\begin{aligned}x + y &= (a + b\epsilon) + (c + d\epsilon)\\ &= a + c + (b + d)\epsilon,\end{aligned}$$

for multiplication

$$\begin{aligned}x\cdot y &= (a + b\epsilon)\cdot(c + d\epsilon)\\ &= ac + (ad + bc)\epsilon + bd\epsilon^2\\ &= ac + (ad + bc)\epsilon,\end{aligned}$$

and for division

$$\begin{aligned}\frac{x}{y} &= \frac{a + b\epsilon}{c + d\epsilon}\\ &= \frac{a + b\epsilon}{c + d\epsilon}\cdot\frac{c - d\epsilon}{c - d\epsilon}\\ &= \frac{ac - (ad - bc)\epsilon - bd\epsilon^2}{c^2 - d\epsilon^2}\\ &= \frac{a}{c} + \frac{bc - ad}{c^2}\epsilon.\end{aligned}$$

This is very convenient, but how does dual numbers handle elementary functions like sin, exp, log etc? If we look at the Taylor expansion of a function $f(x)$ where x is a dual number, we get

$$\begin{aligned} f(x) = f(a+b\epsilon) &= f(a) + \frac{f'(a)}{1!}(b\epsilon) + \frac{f''(a)}{2!}(b\epsilon)^2 + \ldots \\ &= f(a) + f'(a)b\epsilon. \end{aligned}$$

This means that to make dual numbers handle elementary functions, the first order Taylor expansion needs to be implemented. This equals the implementation of elementary differentiation rules described in equations (2.2).

The weakness of implementing AD with dual numbers is clear for functions with multiple variables. Let the function f be defined as $f(x,y,z) = x \cdot y + z$. Let us say we want to know the function value for $(x,y,z) = (2,3,4)$ together with all the derivatives of f. First we evaluate f with x as the only varying parameter and the rest as constants:

$$\begin{aligned} f(x,y,z) &= (2+1\epsilon)\cdot(3+0\epsilon) + (1+0\epsilon) \\ &= 7 + 3\epsilon. \end{aligned}$$

7 is now the function value of f, while 3 is the derivative value of f with respect to x, f_x. To obtain f_y and f_z we need two more function evaluations with respectively y and z as the varying parameters. This example illustrates the weakness of forward AD implemented with dual numbers - when the function evaluated have many input variables, we need equally many function evaluations to determine the jacobian of the function.

2.1.3 Backward Automatic Differentiation

The main disadvantage with forward AD is when there are many input variables and you want the derivative with respect to all variables. This is where Backward AD is a more efficient way of obtaining the derivatives. To explain backward AD it is easier to first consider the approach for forward AD, where the method also can be be explained as an extensive use of the chain rule

$$\frac{\partial f}{\partial t} = \sum_i \left(\frac{\partial f}{\partial g_i} \cdot \frac{\partial g_i}{\partial t} \right). \tag{2.5}$$

Take $f(x) = x \cdot \exp(2x)$ like in the forward AD example (2.3). We then split up the function into a sequence of elementary functions

$$x, \qquad g_1 = 2x, \qquad g_2 = \exp(g_1), \qquad g_3 = x \cdot g_2, \tag{2.6}$$

where clearly $f(x) = g_3$. If we want the derivative of f with respect to x we can obtain expressions for all g's by using the chain rule (2.5)

$$\begin{aligned} \frac{\partial x}{\partial x} &= 1, \\ \frac{\partial g_1}{\partial x} &= 2, \\ \frac{\partial g_2}{\partial x} &= \frac{\partial}{\partial g_1}\exp(g_1) \cdot \frac{\partial g_1}{\partial x} = 2\exp(2x). \end{aligned}$$

Lastly by calculating the derivative of g_3 with respect to x in the same matter yields the expression for the derivative of f

$$\begin{aligned}\frac{\partial f}{\partial x} &= \frac{\partial g_3}{\partial x} \\ &= \frac{\partial x}{\partial x} \cdot g_2 + x \cdot \frac{\partial g_2}{\partial x} \\ &= \exp(2x) + x \cdot 2\exp(2x) \\ &= (1+2x)\exp(2x).\end{aligned}$$

This shows how forward AD actually uses the chain rule on a sequence of elementary functions with respect to the independent variables, in this case x. Backward AD also uses the chain rule, but in the opposite direction. It uses it with respect to dependent variables. The chain rule then has the form

$$\frac{\partial s}{\partial u} = \sum_i \left(\frac{\partial f_i}{\partial u} \cdot \frac{\partial s}{\partial f_i} \right), \tag{2.7}$$

for some s to be chosen. The same example with $f(x) = x \cdot \exp(2x)$ and with the same sequence of elementary functions like in (2.6) gives the expressions from the chain rule (2.7)

$$\begin{aligned}&\frac{\partial s}{\partial g_3} = \text{Unknown} \\ &\frac{\partial s}{\partial g_2} = \frac{\partial g_3}{\partial g_2} \cdot \frac{\partial s}{\partial g_3} && \iff \quad x \cdot \frac{\partial s}{\partial g_3} \\ &\frac{\partial s}{\partial g_1} = \frac{\partial g_2}{\partial g_1} \cdot \frac{\partial s}{\partial g_2} && \iff \quad g_2 \cdot \frac{\partial s}{\partial g_2} \\ &\frac{\partial s}{\partial x} = \frac{\partial g_3}{\partial x} \cdot \frac{\partial s}{\partial g_3} + \frac{\partial g_1}{\partial x} \cdot \frac{\partial s}{\partial g_1} && \iff \quad g_2 \cdot \frac{\partial s}{\partial g_3} + 2 \cdot \frac{\partial s}{\partial g_1}.\end{aligned}$$

By substituting s with g_3 gives

$$\begin{aligned}&\frac{\partial g_3}{\partial g_3} = 1 \\ &\frac{\partial g_3}{\partial g_2} = x \\ &\frac{\partial g_3}{\partial g_1} = \exp(2x) \cdot x \\ &\frac{\partial g_3}{\partial x} = \exp(2x) \cdot 1 + 2 \cdot \exp(2x) \cdot x = (1+2x)\exp(2x),\end{aligned}$$

hence we obtain the correct derivative f_x. By now you might wonder why make this much effort to obtain the derivative of f compared to just using forward AD. The answer to this comes by looking at a more complex function with multiple input parameters. Let $f(x,y,z) = z(\sin(x^2) + yx)$ and

$$g_1 = x^2, \quad g_2 = x \cdot y, \quad g_3 = \sin(g_1), \quad g_4 = g_2 + g_3, \quad g_5 = z \cdot g_4.$$

Now the derivatives from the chain rule in equation (2.7) becomes

$$\frac{\partial s}{\partial g_5} = \text{Unknown} \qquad \frac{\partial s}{\partial g_2} = \frac{\partial s}{\partial g_4} \qquad \frac{\partial s}{\partial y} = x \cdot \frac{\partial s}{\partial g_2}$$
$$\frac{\partial s}{\partial g_4} = z \cdot \frac{\partial s}{\partial g_5} \qquad \frac{\partial s}{\partial g_1} = \cos(g_1)\frac{\partial s}{\partial g_3} \qquad \frac{\partial s}{\partial z} = g_4 \cdot \frac{\partial s}{\partial g_5}$$
$$\frac{\partial s}{\partial g_3} = \frac{\partial s}{\partial g_4} \qquad \frac{\partial s}{\partial x} = 2x \cdot \frac{\partial s}{\partial g_1} + y \cdot \frac{\partial s}{\partial g_2}$$

substituting s with g_5 yields

$$\frac{\partial g_5}{\partial g_5} = 1 \qquad \frac{\partial g_5}{\partial g_2} = z \qquad \frac{\partial g_5}{\partial y} = xz$$
$$\frac{\partial g_5}{\partial g_4} = z \qquad \frac{\partial g_5}{\partial g_1} = \cos(x^2) \cdot z \qquad \frac{\partial g_5}{\partial z} = \sin(x^2) + xy$$
$$\frac{\partial g_5}{\partial g_3} = z \qquad \frac{\partial g_5}{\partial x} = 2x \cdot \cos(x^2) \cdot z + yz$$

The calculation of the derivatives together with a dependency graph can be seen in figure 2.1. This shows that we get all the derivatives of $f(x) = g_5$ with a single function evaluation! Comparing this

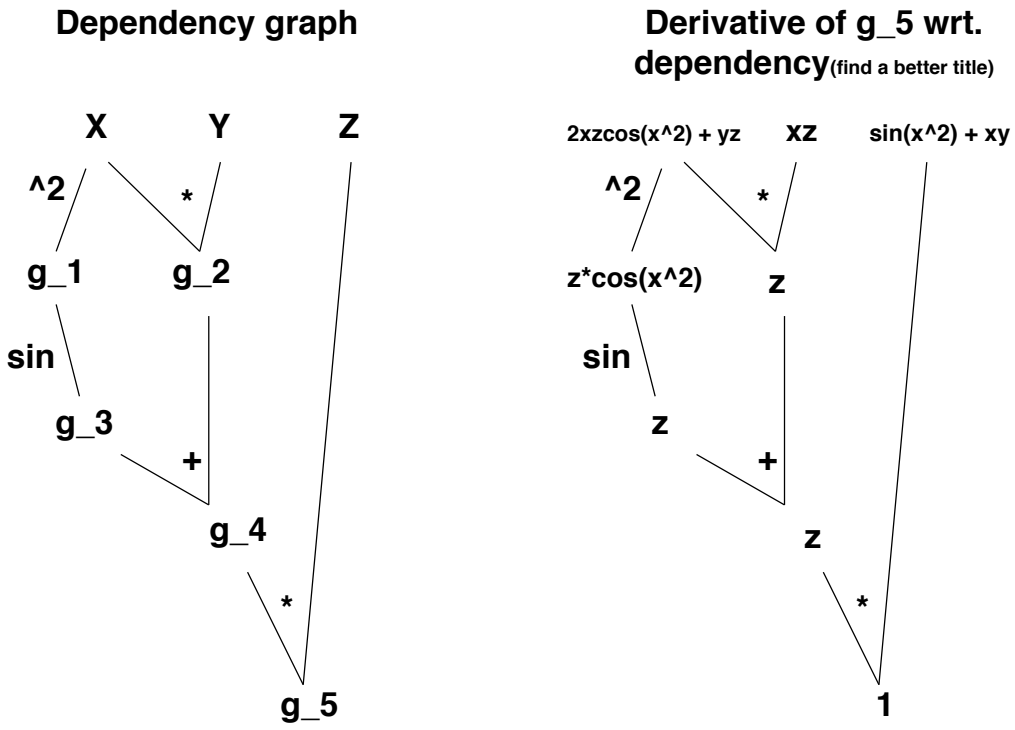

Figure 2.1: NOT A FINISHED FIGURE. Just a quick sketch of the idea I have of visualising the process of backward AD as i felt it got a bit messy with all the expressions. Gladly receiving comments on the thought/suggestions to make it more clear

to the method of Dual Numbers were we would have to evaluate f three times, one for each derivative, this is a big improvement. This illustrates the strength of backward AD - no matter how many input parameters a function have, you only need one function evaluation to get all the derivatives of a function. The disadvantage of backward AD is that, in differ from what we did in the example above when we did it by hand, we still only want to carry along the function value and the derivative value as we did in forward AD. This means that we have to implement the dependency tree shown in figure

2.1. This makes the implementation of backward AD much harder than for forward AD and a bad implementation of this tree will reduce the advantage of backward AD. This is the reason why we here will have focus on implementing forward AD.

2.1.4 Forward Automatic Differentiation with multiple parameters

TODO: Endre under så dette glir inn med det som er skrevet om BackwardAD over To handle the problem of many function evaluations in forward AD when having a function with many input parameters one can look at the method described by Knut-Andreas Lie in *User Guide for the MATLAB Reservoir Simulation Toolbox (MRST)*. Say we have three variables x, y and z. Then the corresponding AD-variables becomes

$$[x\ ,\ (1,0,0)^\top] \quad , \quad [y\ ,\ (0,1,0)^\top] \quad , \quad [z\ ,\ (0,0,1)^\top].$$

There are now not only one scalar as the derivative value, but the gradient to the corresponding variable. The operators defined in Equations 2.1 and the elementary functions in Equations 2.2 are still valid, but instead of scalar products there are now vector products. As an example let $f(x,y,z) = xyz$ and $x = 1$, $y = 2$ and $z = 3$, then

$$\begin{aligned} xyz &= [1\ ,\ (1,0,0)^\top] \cdot [2\ ,\ (0,1,0)^\top] \cdot [3\ ,\ (0,0,1)^\top] \\ &= [1\cdot 2\cdot 3\ ,\ 2\cdot 3\cdot (1,0,0)^\top + 1\cdot 3\cdot (0,1,0)^\top + 1\cdot 2\cdot (0,0,1)^\top] \\ [f\ ,\ f_x] &= [6\ ,\ (6,3,2)^\top]. \end{aligned}$$

This result is equal to the tuple

$$\big(f(x,y,z)\ ,\ \nabla f(x,y,z)\big) = \big(xyz\ ,\ (yz,xz,xy)^\top\big)$$

for the corresponding x, y and z values.

2.2 Julia

Chapter 3

Implementation

3.1 Implementation of Automatic Differentiation

When it comes to implementing Automatic Differentiation(AD) there are two major concerns. First is that it must be easy and intuitive to use, the second is that it must be efficient code as it will be used in computational demanding calculations.

A convenient way to store the AD-variables in Julia is to make a struct that have two member variables, `val` and `jac`, that stores respectively the value and the corresponding Jacobian. The importance of the way you implement the AD operators can be expressed in a short example: Consider you have two variables x and y and you want to compute the function $f(x,y) = y + exp(2xy)$. If the implementation is based on making new functions that take in AD-variables as input parameters, it will look something like this:

$$f = \texttt{ADplus}(y,\texttt{ADexp}(\texttt{ADtimes}(2,\texttt{ADtimes}(x,y)))).$$

This is clearly not a suitable way to implement AD and should be avoided. Instead of making new functions that takes in AD-variables as parameters one should overload the standard operators (+,-,*,/) and the elementary functions (exp, sin, log, etc.). In Julia this involves overloading the Base module such that when you write $x + y$ with x and y as AD-variables, Julia's Multiple Dispatch **ADD REF**, understand that it is your definition of the "+" operator that is meant to be used. This gives us the opportunity to only write $f = y + \exp(2xy)$ if we want to compute $f(x,y)$ for given x and y.

3.2 Applications of Automatic Differentiation

Next: starte med å skrive om newton-solveren jeg har laget som løser f(x) = 0.

Chapter 4

Result and discussion

Chapter 5

Conclusion and Future Work

Bibliography

User Guide for the MATLAB Reservoir Simulation Toolbox (MRST), 2018. An introduction to reservoir simulation using matlab/gnu octave. Accessed 25.10.2018.
URL https://folk.ntnu.no/andreas/mrst/mrst-cam.pdf